ABSTRACT

Title of Dissertation: ON THE EMERGENT STRUCTURES OF DEEP NEURAL NETWORKS

Tianyu He
Doctor of Philosophy, 2025

Dissertation Directed by: Professor Maissam Barkeshli
Department of Physics

Despite the remarkable success of artificial intelligence (AI), we still lack a fundamental understanding of why deep learning works – why certain architectural designs are preferred, how optimization navigates complex landscapes, and what data characteristics enable learning. This dissertation addresses these questions through a phenomenological approach, abstracting essential features from complex real-world experiments and constructing minimal setups that faithfully capture them. The investigation unfolds in three interconnected parts.

In the first part of this dissertation, we study signal propagation in randomly initialized networks under the infinite-width limit. By examining the inter-layer Jacobian norm as an observable at large depth, we identify a phase diagram with two distinct phases. The observable precisely characterizes the transition and predicts that neural networks best maintain signals exactly at the phase boundary. Through extensive empirical validation, we demonstrate that this metric provides universal guidance and an efficient algorithm for selecting network initializations across

diverse architectures, offering a principled alternative to heuristic initialization schemes.

In the second part, we elucidate the optimization landscape of neural networks by carefully analyzing the dynamics of the loss Hessian (sharpness) under gradient descent. Remarkably, we show that even a minimal two-layer linear network trained on a single example exhibits all the essential sharpness phenomenology observed in real-world scenarios. By analyzing the structure of dynamical fixed points in function space, we reveal the fundamental mechanisms governing sharpness evolution during training, thereby clarifying the interplay between initialization, parameterization, and gradient descent dynamics at large learning rates.

In the third part, we investigate how data characteristics and network architecture choices give rise to the emergent capabilities of neural networks. We begin by analyzing networks trained on modular arithmetic tasks. Across different setups, we establish necessary data-diversity thresholds and minimal network sizes for successful task performance, demonstrating that certain computational capabilities emerge only when both architectural capacity and data coverage exceed critical values. Furthermore, we extend this understanding to a more practical setting by tackling Pseudo-Random Number Generator (PRNG) tasks.

Finally, synthesizing insights from the preceding parts, we challenge conventional architectural design paradigms. We propose a new (proto-)architecture that enables arbitrary, learned inter-layer connectivity patterns. Through careful analysis of trained models in both vision and language domains, we observe the emergence of distinct connectivity structures, accompanied by interpretable layer specialization and meaningful correlations between the nature of the data and the network's internal computational patterns.

ON THE EMERGENT STRUCTURES OF DEEP NEURAL NETWORKS

by

Tianyu He

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2025

Advisory Committee:
Professor Maissam Barkeshli, Chair/Co-Advisor
Professor Victor V. Albert
Professor Tom Goldstein
Professor Andrey Gromov, Advisor
Professor Daniel P. Lathrop

© Copyright by
Tianyu He
2025

Dedication

To my parents, for their unconditional love and support.

To the land that shaped me.

Acknowledgments

I am probably the luckiest person to have been surrounded by people who have offered me endless guidance and support. Without the many interactions and accumulated insights from them, it would have been impossible for me to complete this dissertation.

The first and most important person I would like to thank is my advisor, Dr. Andrey Gromov, who guided me to work at the intersection of physics and deep learning. He is not only a physicist with a deep command of the field, but also a thinker who sees far beyond its boundaries. His proactive guidance, broad and unique vision, and remarkable ability to abstract and identify the core concepts within complex problems have deeply influenced and shaped me throughout my journey.

I would also like to thank Dr. Maissam Barkeshli for his guidance, co-advising, and for offering a distinct working style that allowed me to view problems from different perspectives. I am especially grateful for his high standards in scientific writing – although I am still learning to distill and internalize, it has deeply influenced how I present clear and reproducible results to the community.

A good environment is essential for growth – for both humans and AI. I cannot overstate my gratitude for the labmates I have been fortunate to collaborate with: Dr. Darshil Doshi, a good friend and teammate throughout my entire Ph.D. journey, who has mastered the art of "slow thinking", often helping me avoid bad local minima in research; Dayal Kalra, who introduced

me to the world of non-linear dynamics and sparked countless ideas through our spontaneous conversations; and Aritra Das and Tao Tao, who frequently helped me identify hidden loopholes during discussions and pushed me to think more deeply.

To the people at the University of Maryland, I express my deepest gratitude to Josiland Chambers and Dr. Thomas Cohen, who handled all the heavy lifting on administrative matters. I also thank the Condensed Matter Theory Center and all its members for their support and for the insightful conversations during our weekly lunch gatherings.

To the Department of Physics at Brown, I am deeply grateful for providing a calm and supportive environment that allowed me to complete my "mid-training" without the pressure of rushing toward immediate outcomes. I especially thank Dr. Antal Jevicki and Dr. Robert A. Pelcovits for sharing their unique perspectives on physics, distilled through history, which clarified the original contexts from which many "easy-to-show" arguments emerged.

Deep learning is a field deeply rooted in mathematics, and I would like to thank Dr. Boris Hanin for organizing the Princeton Machine Learning Theory Summer School, where I had the opportunity to interact with people from diverse backgrounds who share the same passion for these topics.

To my friends, whom I have been fortunate to know at different stages of my life, and who have supported me in countless ways – through both joy and moments of struggle: Haolan Xu and Ruochen Ma, for the many times we discussed physics and life choices; Naiyuan Zhang, Zezhu Wei, Lecheng Ren, Xianlong Liu, and Lingfeng Li, my roommates, with whom I shared countless moments of physics, cooking, games, and laughter; Saket Tiwari, for the insightful discussions on AI and beyond; and Ziwen Ma, Lun Xi, Ke Jin, Bingjie Wang, Zonghua Qu, Xiyu Han, Zhicheng Bao, Ming Zheng, Yanjie Jing, Yuan Fang, Zijian Song, Shengqi Sang, and Yue

Xu, for the long-lasting friendship that has endured across the Pacific Ocean. To the many other friends too numerous to name, I owe you all my heartfelt thanks.

Finally, I owe the most to my parents, who sheltered me and allowed me to grow happily in my early life, and who later gave me complete freedom to choose my own path. I would not have come this far without their love, care, and thoughtful guidance throughout my life.

Citations to Previously Published Work

All of the research described in this dissertation has been either published or made available as a preprint on arXiv.

Chapter 2 and appendix A are adapted from the published work [1] and the preprint [2]. The works were completed in collaboration with Dr. Darshil Doshi under the guidance of Dr. Andrey Gromov. The author's main contribution was developing the codebase and performing most of the experiments shown in this chapter, including part of the theoretical contribution discussed within Sections 2.3 and 2.5.

Chapter 3 and appendix B are adapted from the published work [3], conducted in collaboration with Dayal Singh Kalra under the supervision of Dr. Maissam Barkeshli. The author's contributions include the early partial development of the theoretical framework, partial implementation of the sharpness computation code, and the design and execution of real-world model experiments in Section 3.6.

Chapter 4 and appendix C are adapted from the published work [4] and the preprint [5], completed in collaboration with Dr. Darshil Doshi and Aritra Das under the supervision of Dr. Andrey Gromov. The author's contributions include early-stage exploration, partial code development, and the design and execution of the experiments presented in Section 4.3.

Chapter 5 and appendix D are adapted from the published work [6], completed in collaboration with Dr. Darshil Doshi and Aritra Das under the supervision of Dr. Andrey Gromov. The

author contributed by developing the whole codebase, together with the design and execution of most of the experiments presented in this chapter.

Chapter 6 and appendix E are adapted from the published work [7]. This work was completed in collaboration with Tao Tao, Dr. Darshil Doshi and Dayal Singh Kalra, under the guidance of Dr. Maissam Barkeshli. The author mainly contributed to Section 6.4 experiments and code.

Chapter 7 and appendix F are adapted from the preprint [8], completed in collaboration with Dr. Aditya Cowsik under the supervision of Dr. Andrey Gromov. The author contributed to the architecture design, implemented most of the codebase, and designed and executed the majority of the experiments.

Note that the contents from [1, 2, 4, 5, 7], which are adapted and appear in the aforementioned chapters of this dissertation, have also been adapted and included in the Ph.D. dissertation of Dr. Darshil Doshi [9].

Verbal permission to include all aforementioned materials has been obtained from all co-authors, further copyright information for published works is detailed in Appendix G.

Table of Contents

Dedication ii

Acknowledgements iii

Citations to Previously Published Work vi

Table of Contents viii

List of Tables xv

List of Figures xvi

List of Abbreviations xxxv

Chapter 1: Introduction 1
- 1.1 Challenges from Scaling Up . . . 1
  - 1.1.1 Pre-LLM Era . . . 1
  - 1.1.2 LLM Era . . . 2
  - 1.1.3 Post-LLM Era . . . 4
- 1.2 Emergence from Scaling . . . 4
- 1.3 Summary and Outline . . . 7

Chapter 2: Emergent Depth Scale at Initialization 11
- 2.1 Introduction and Related Work . . . 11
  - 2.1.1 Related Work . . . 13
- 2.2 Multi-Layer Perceptrons (MLPs) . . . 14
  - 2.2.1 Emergent Depth Scale in the Forward Pass . . . 15
  - 2.2.2 Emergent Depth Scale in the Backward Pass . . . 19
- 2.3 Layer Normalization . . . 22
- 2.4 Residual Connections . . . 24
- 2.5 Residual Connections + Layer Normalization . . . 26
- 2.6 Modern Architectures . . . 31
- 2.7 Automatic Initialization via Jacobian Tuning . . . 33
- 2.8 Discussion . . . 36
  - 2.8.1 Limitations and Outlook . . . 37

Chapter 3: Emergent Low-dimensional Fixed-Point Structures in Sharpness Dynamics 39

3.1 Introduction to Sharpness Dynamics . . . 40
3.2 Notations and Preliminaries . . . 41
3.3 Review of The Four Regimes of Training . . . 42
3.4 Fixed Point Analysis of the UV model . . . 45
3.5 Understanding Sharpness Dynamics in the UV model . . . 48
3.5.1 Understanding Sharpness Trends Throughout Training . . . 48
3.5.2 Understanding Edge of Stability . . . 50
3.6 Predictions and Verifications in Real-world Scenarios . . . 53
3.6.1 Implications of Initialization and Parameterization for Real-world Models 54
3.6.2 Sharpness & Weight-Norm Correlation and the Origin of Four Regimes . 55
3.6.3 The Phase Diagram of Edge of Stability . . . 55
3.6.4 Route to Chaos and Bifurcation Diagrams . . . 56
3.7 Related Work . . . 58
3.8 Discussion . . . 59
3.8.1 Limitations and Future Work . . . 59

Chapter 4: Grokking Modular Arithmetic 61
4.1 Introduction . . . 62
4.2 Setup . . . 63
4.3 Probe Generalizing Circuit with Label Noise . . . 68
4.3.1 Generalizing and Memorizing Circuits . . . 72
4.3.2 General Architectures . . . 75
4.4 Related Works . . . 76
4.5 Discussion . . . 77

Chapter 5: In-Context Learning and Skill Composition 79
5.1 Introduction . . . 79
5.2 Preliminaries . . . 83
5.3 Emergence of In-Context Learning and Task Composition . . . 85
5.3.1 Transition Driven by the Number of Tasks . . . 86
5.3.2 Effect of Model Size and Task Difficulty . . . 89
5.4 Interpreting the Learned Algorithms . . . 91
5.4.1 Attention Heads Implement Skill I . . . 93
5.4.2 MLPs Implement Skill II and III . . . 96
5.5 Related Works . . . 98
5.6 Discussion . . . 100

Chapter 6: Transformers and Pseudo-Random Number Generators 102
6.1 Linear Congruential Generators . . . 102
6.2 Training Setup . . . 103
6.2.1 Dataset Generation and Evaluation . . . 104
6.2.2 Tokenization, Architecture and Optimizer . . . 105
6.3 Training Results . . . 106
6.4 Interpreting How Transformers Predict PRNGs . . . 108
6.4.1 Residual Number System Representations . . . 109

6.4.2 Fixed Modulus . . . 111
6.4.3 Unseen Modulus . . . 117
6.5 Scaling Up the Modulus . . . 123
6.5.1 Fixed Modulus . . . 123
6.5.2 Unseen Modulus . . . 124
6.6 Related Work . . . 126
6.7 Discussion . . . 127
6.7.1 Limitations and Outlook . . . 127

Chapter 7: Towards Distributed Neural Architectures 128
7.1 Emergent Connectivity . . . 130
7.2 Distributed neural architectures . . . 131
7.2.1 General set up and intuition . . . 131
7.2.2 Technical details . . . 135
7.3 DNA for computer vision . . . 138
7.3.1 Experimental details . . . 138
7.3.2 Interpretability . . . 139
7.3.3 Efficiency . . . 146
7.4 Language DNA models . . . 149
7.4.1 Experimental details . . . 149
7.4.2 Interpretability . . . 150
7.4.3 Efficiency . . . 153
7.5 Conclusion and discussion . . . 154
7.5.1 Conclusions . . . 154
7.5.2 Unresolved Issues and Future work . . . 156

Chapter 8: Summary and Future Directions 158

Appendix A: Emergent Length Scale at Initialization 163
A.1 Experimental Details . . . 163
A.2 Batch Normalization . . . 165
A.3 Additional Discussion on ResNets . . . 166
A.4 Discussion on Transformers . . . 167
A.5 Technical Details and Derivations . . . 169
A.5.1 APJN Proofs and Derivations . . . 169
A.5.2 APJN and gradients . . . 172
A.5.3 LayerNorm on Pre-activations . . . 173
A.5.4 LayerNorm on activations . . . 176
A.6 Residual Connections . . . 179
A.7 Residual Connections with LayerNorm on Preactivations (Pre-LN) . . . 181
A.8 Critical Exponents . . . 185
A.9 MLP-Mixer . . . 187
A.10 Results for Scale Invariant Activation Functions . . . 191
A.10.1 NNGP Kernel . . . 191
A.10.2 Jacobian(s) . . . 192

A.10.3 Critical Exponents . . . 193
A.10.4 LayerNorm on Pre-activations . . . 193
A.10.5 LayerNorm on Activations . . . 193
A.10.6 Residual Connections . . . 194
A.10.7 Residual Connections with LayerNorm on Preactivations (Pre-LN) . . . 195
A.11 Results for erf Activation Function . . . 196
A.11.1 NNGP Kernel . . . 196
A.11.2 Jacobians . . . 197
A.11.3 Critical Exponents . . . 198
A.11.4 LayerNorm on Pre-activations . . . 199
A.11.5 LayerNorm on Activations . . . 200
A.11.6 Residual Connections . . . 201
A.11.7 Residual Connections with LayerNorm on Preactivations (Pre-LN) . . . 203
A.12 Results for GELU Activation Function . . . 204
A.12.1 NNGP Kernel . . . 204
A.12.2 Jacobians . . . 205
A.12.3 Critical Exponents . . . 207
A.12.4 LayerNorm on Preactivations . . . 208
A.12.5 LayerNorm on Activations . . . 209
A.12.6 Residual Connections . . . 210
A.12.7 Residual Connections with LayerNorm on Preactivations (Pre-LN) . . . 211
A.13 Additional Experimental Results . . . 212

Appendix B: Emergent Low-dimensional Fixed-Point Structures in Sharpness Dynamics 215
B.1 Further Discussion on Related Works . . . 215
B.2 Experimental details . . . 219
B.2.1 Datasets . . . 219
B.2.2 Models . . . 220
B.2.3 Details of network parameterization . . . 221
B.2.4 Details of Figures . . . 223
B.2.5 Sharpness measurement . . . 225
B.2.6 Power spectrum analysis . . . 225
B.2.7 Estimation of Computational Resources Used . . . 226
B.3 Properties of the UV model . . . 226
B.3.1 Derivation of the Function Space Dynamics . . . 226
B.3.2 Forbidden regions of the UV model . . . 227
B.3.3 Fixed Points and Line . . . 228
B.3.4 The maximum learning rate η_{upper} . . . 229
B.3.5 Sharpness versus the trace of Hessian . . . 230
B.3.6 The distribution of residual and NTK at initialization . . . 231
B.3.7 EoS manifold is a dynamical attractor . . . 233
B.3.8 Critical learning rate for edge of stability . . . 233
B.3.9 Sub-quadratic Loss . . . 236
B.4 Additional Results on Sharpness Dynamics . . . 236
B.4.1 Sharpness Dynamics of NTP Networks . . . 236

B.4.2 The effect of batch size on the four training regimes . . . 238
B.4.3 Generizability of the Four Training Regimes . . . 240
B.5 Sharpness-weight norm correlation . . . 240
B.5.1 Setting the last layer to zero eliminates sharpness reduction . . . 244
B.6 Additional Phase diagrams of EoS . . . 245
B.7 Route to chaos . . . 248
B.7.1 Route to EoS in real datasets . . . 248
B.7.2 The effect of power-law trends in data on sharpness trajectories . . . 250
B.7.3 Route to chaos in synthetic datasets . . . 252

Appendix C: Grokking Modular Arithmetic 255
C.1 Further Experimental Details . . . 255
C.1.1 Figures in Main Text . . . 255
C.2 Details of Resources Used . . . 256
C.3 Quantitative Characterization of Phases . . . 256
C.4 Forgetting Phase . . . 257
C.5 Output Logits . . . 260
C.6 BatchNorm, LayerNorm and minibatch w/o Normalization . . . 261
C.6.1 BatchNorm . . . 261
C.6.2 BatchNorm, LayerNorm and Mini-Batch Noise . . . 262
C.7 Transformers and Deep MLPs . . . 264
C.7.1 Effect of Depth / Number of Heads / Weight Tying . . . 264
C.7.2 Dropout Phase Diagram for Transformers . . . 266
C.7.3 Effect of Parameterization for 3-layer MLPs . . . 267
C.7.4 BatchNorm for 3-Layer ReLU MLPs . . . 269
C.8 Additional Pruning Curves . . . 270
C.8.1 Accuracy curves . . . 270
C.8.2 Loss curves . . . 272
C.9 Additional Training Curves . . . 274
C.10 Further Ablation Studies . . . 276
C.10.1 Choice of Loss Function . . . 276
C.10.2 Choice of Activation Function . . . 278
C.10.3 Effect of of width . . . 279
C.10.4 Modular Multiplication . . . 280
C.11 How many neurons can be pruned while retaining generalization? . . . 281

Appendix D: In-Context Learning and Skill Composition 282
D.1 Experimental Details . . . 282
D.1.1 Model and Training Hyperparameters . . . 282
D.1.2 Additional Details of Each Plot in the Main Text . . . 283
D.2 Details of resources used . . . 284
D.3 Table of log map for $p = 29$ . . . 284
D.4 Task Selection and Sequence Design . . . 284
D.5 Additional Interpretability Results . . . 286
D.5.1 PCA over Embeddings . . . 287

D.5.2 Attention Heads . . . 287
D.5.3 PCA over Attention and MLP Outputs . . . 292
D.5.4 Cosine-similarity analysis for MLPs . . . 293
D.6 Additional Training Curves . . . 295
D.7 Additional Phase Diagrams . . . 296
D.8 Different Choice of p . . . 296

Appendix E: Transformers and Pseudo-Random Number Generators 299
E.1 Experimental Details . . . 299
E.1.1 Dataset Construction . . . 299
E.1.2 Model Architecture Details . . . 301
E.1.3 Training Details . . . 301
E.1.4 Training Cost . . . 302
E.1.5 Hyperparameter Space Shrinking with increasing modulus . . . 302
E.2 LCG Properties . . . 303
E.2.1 The period of the lower k-th bit when m is power of 2 . . . 303
E.2.2 Derivation of Equation (6.4) . . . 304
E.3 Fixed Modules Training Results . . . 306
E.4 Critical Depth for the Unseen Modulus Task . . . 308
E.5 Prime Moduli . . . 309
E.6 Training Time Interpretability . . . 310
E.7 Fixed Modulus Interpretability . . . 311
E.8 Unseen Modulus Interpretability . . . 316
E.8.1 Extra PCA plots for first layer heads . . . 316
E.8.2 Pruning heads corresponding to irrelevant prime factors . . . 316
E.8.3 Patching other heads . . . 316
E.8.4 Inaccurate estimation of m_{test} . . . 318
E.8.5 Evidence for Step iii . . . 319
E.9 Scaling Up the Modulus . . . 320
E.9.1 Base-b tokenization . . . 320
E.9.2 Abacas Embeddings . . . 321
E.9.3 Fixed Modulus . . . 322
E.9.4 Unseen Modulus . . . 323

Appendix F: Distributed Neural Architectures 327
F.1 Experimental details . . . 327
F.1.1 Vision models . . . 328
F.1.2 Language models . . . 329
F.2 Module Usage and Load Balancing . . . 330
F.3 Dreaming Visualizations . . . 331
F.3.1 Experimental Details . . . 331
F.3.2 Further Dreaming Examples . . . 333
F.4 More on vision DNAs . . . 336
F.4.1 Compute distribution over patches . . . 336
F.4.2 Random model paths . . . 336

F.5 More on language DNAs . . . 338
F.5.1 Interpretability . . . 338
F.5.2 Compute distribution over tokens . . . 339
F.6 Breaking Transformer Blocks . . . 342
F.6.1 Vision DNA . . . 342
F.6.2 Language DNA . . . 345

Appendix G: Copyright Information 348

Appendix H: AI Usage Disclosure 350

Bibliography 351

List of Tables

4.1 Qualitative characterization of phases. 72

7.1 Hyperparameters of DNA models and ViT-small. Note that for DNA models, the number of active parameters shown in parentheses refers to non-shared active parameters (as detailed in Sec. 7.3.3). 139

7.2 Hyperparameters of DNA models and GPT-2 medium (no weight-tying). Note that for DNA models, the number of active parameters shown in parentheses refers to non-shared active parameters (as detailed in Sec. 7.4.3). 150

7.3 Final validation loss and zero-shot downstream evaluation. We reported accuracy in all cases with standard error, except for Wikitext, where we reported word-level perplexity. Columns are bolded only when there is a clear winner, taking standard error into account. Hyperparameter details for the model with skipping and the shallower GPT-2 are listed in Appendix F.1. 150

B.1 Fixed line (I) and points (II-IV) and corresponding eigenvalues and eigenvectors of the Jacobian of the update map in Equations (3.2) and (3.3). The stability is determined for $\eta < \eta_{\text{upper}} = 2\eta_c = {}^2\sqrt{n_{\text{eff}}}/\|\boldsymbol{x}\|y$. Above this threshold, training diverges for all initializations except for those at the fixed points, as demonstrated in Appendix B.3.4. 229

C.1 Quantitative characterization of phases. 257

F.1 Hyperparameters of all vision model used in this paper, where only the last two lines are new compared to Table 7.1. Note that i) the number of active parameters column does not take model re-use into account; ii) the backbone for DNA-n models are full transformer blocks. 329

F.2 Hyperparameters of DNA models and GPT-2 medium (no weight-tying). Note that for DNA models, we do not take module re-use into account while counting the number of active parameters. 330

List of Figures

2.1 log–log plots of partial Jacobian $\mathcal{J}^{0,l}$ vs. l. From left to right: **(1)** erf, **(2)** ReLU, **(3)** erf and GELU with LayerNorm applied to preactivations and residual connections of strength 1. The fluctuations get larger towards the output because the aspect ratio (*i.e.* l/N_l) approaches $1/4$. 22

2.2 $\chi^\star_{\mathcal{J}}$ empirical phase diagrams for an MLP with $L = 50, N = 500$. The solid lines indicate the critical lines obtained through the infinite width limit calculations, and the stars indicate the critical points. We see that residual connections in general just shift the phase boundary. Except for erf activation, where the case $\mu = 1$ is subcritical and has a large correlation length. 27

2.3 $\chi^\star_{\mathcal{J}}$ empirical phase diagrams for an MLP with $L = 50, N = 500$. The solid lines indicate the critical lines obtained through the infinite width limit calculations, and the stars indicate the critical points. The dotted lines in the rightmost column correspond to the critical lines for $\mu < 1$ case. For networks with LayerNorm and $\mu = 1$, $\chi^\star_{\mathcal{J}} = 1$ holds on the entire σ_b–σ_w plane. 28

2.4 Trainability (Training Accuracy) of deep MLP ($N_l = 500$, $L = 50$) on Fashon-MNIST dataset. The subcritical case of erf activation with $\mu = 1$ has enhanced trainability. The dashed white lines denote the (analytical) critical lines. 29

2.5 Trainability (Training Accuracy) of deep MLP ($N_l = 500$, $L = 50$) on FashonMNIST dataset. The combination of LayerNorm and $\mu = 1$ makes the network everywhere-trainable. The dashed white lines denote the (analytical) critical lines. 30

2.6 Results for ResNet110 V2 **(a)** LayerNorm and (b) BatchNorm. Within each row from Left to right: **(1)** $\chi^\star_{\mathcal{J}}$ phase diagram with respect to (σ_w^2, σ_b^2) at $\mu = 1$. The network is everywhere-critical for both LayerNorm and BatchNorm. **(2)** $\chi^\star_{\mathcal{J}}$ phase diagram with respect to (σ_w^2, μ). **(3)** Training accuracy w.r.t. σ_w^2, for different values of μ. Trainability improves with with higher μ, as the network gets closer to everywhere-critical phase. For $\mu = 0$ and small σ_w^2 the network is in the ordered phase for LayerNorm and chaotic phase for BatchNorm. **(4)** Training accuracy w.r.t. μ at $(\sigma_w^2, \sigma_b^2) = (2, 0)$. We see a monotonic improvement in trainability as we tune up the value of μ from 0 to 1. 32

2.7 Results for MLP-Mixer. Left to right: **(1, 2)** $\mu = 0.5, 1.0$ phase diagrams ($\chi^\star_{\mathcal{J}}$). The network becomes everywhere-critical when $\mu = 1$. The solid black line indicates the empirical phase boundary. Stars denote points we selected to train on CIFAR-10. **(3, 4)** $\mu = 0.5, 1.0$ MLP-Mixer training curves. All, but one, networks are $L = 100$ blocks deep. We see that as we increase μ from 0.5 to 1, the trainability of all networks increases, and they are less sensitive to initialization. 33

2.8 **(Left, Middle)**: Comparing $\mathcal{J}^{l,l+1}$ and $\mathcal{K}^l(x,x)$ for ResMLP-S12 for Kaiming, original and `AutoInit` initializations. The network function in the `AutoInit` case is very close to identity at initialization. **(Right)** Training and validation accuracy on CIFAR-10 dataset. 35
2.9 **(Left, Middle)**: Comparing $\mathcal{J}^{l,l+1}$ and $\mathcal{K}^l(x,x)$ between PyTorch version VGG19_BN and `AutoInit` version, we ensure $\mathcal{J}^{l,l+1} = 1$ with a high priority ($\lambda = 0.05$). **(Right)** Training and validation accuracy on CIFAR-10 dataset. . 36

3.1 Training loss and sharpness trajectories of ReLU FCNs trained on a 5k subset of CIFAR-10 examples using MSE loss and GD: (a, b) SP with $\sigma_w^2 = 0.5$, (c, d) SP with $\sigma_w^2 = 2.0$, (e, f) μP with $\sigma_w^2 = 2.0$. The dashed lines in the sharpness figures show the $2/\eta$ threshold. 43
3.2 Training trajectories of the UV model with $\|\boldsymbol{x}\| = 1$ and $y = 2$ in the $(\Delta f, \lambda)$ plane for different values of n, n_{eff} and η. The columns show initializations with different n and n_{eff}, while the rows represent increasing learning rates for fixed initializations. The horizontal dash-dot line $\eta\lambda = 2$ separates the stable (solid black vertical line) and unstable (dashed black vertical line) fixed points along the zero loss fixed line I. Forbidden regions, $2\|\boldsymbol{x}\||\Delta f + y|/\sqrt{n_{\text{eff}}} > \lambda$, (see Appendix B.3.2) are shaded gray. The nullclines $\Delta f_{t+1} = \Delta f_t$ and $\lambda_{t+1} = \lambda_t$ are shown as orange and white dashed curves, respectively. Sharpness reduction, progressive sharpening, and divergent regions are colored green, yellow, and blue. The gray arrows indicate the local vector field $\hat{G}(\Delta f, \lambda)$, which is the direction of the updates. The training trajectories are depicted as black lines with arrows, with the star marking the initialization. In all cases, $\eta_c = \sqrt{n_{\text{eff}}}/2$ (introduced in Section 3.5.2). 47
3.3 (a) Bifurcation diagram depicting limiting values of λ obtained by simulating Equation (3.4). (b) Bifurcation diagram of the UV model. In both figures, $\|\boldsymbol{x}\| = 1, y = 2$, $n_{\text{eff}} = 1$ and $\eta_c = 0.5$. 51
3.4 (a, b) Heatmap of $\eta\lambda^H/2$ and test accuracy of ReLU FCNs in SP trained on a 5k subset of CIFAR-10 until 99% training accuracy is achieved, with the weight variance σ_w^2 and learning rate multiplier $c = \eta\lambda_0^H$ as axes. As the color varies from blue to white, $\eta\lambda^H/2$ increases. (b, d) Same heatmaps with fixed $\sigma_w^2 = 2.0$, but varying s continuously. 53
3.5 2-layer linear FCNs trained on (first row) $5,000$ iid random examples with unit output dimension and (second row) $5,000$ CIFAR-10 examples. Different columns correspond to the bifurcation diagram, late-time sharpness trajectories, and the power spectrum of sharpness trajectories. The power spectrum is computed using the last 1000 steps of the trajectories. 56

4.1 Left to right: **(a, b)** Losses and accuracies of a two-layer MLP trained on modular arithmetic data with four variables ($p = 11, \alpha = 0.5$). It exhibits delayed and rapid transition from memorization to generalization, which we call grokking. . . 62

4.2 Grokking the modular addition task with $p = 97$ with 2-layer MLP, trained with AdamW. **(a)** Sharp transition in test accuracy, long after overfitting. $\overline{\text{IPR}} := \mathbb{E}_k [\text{IPR}_k]$ monotonically increases over time, indicating periodic representations. The test accuracy follows a logistic curve of the form $L/(1 + e^{-k(t-t_0)})$, with $L = 1.00 \pm 0.01$, $k = 0.46 \pm 0.01$ and midpoint of steps at $t_0 = 32.76 \pm 0.03$. **(b)** Example row vector $(U_{k.})$ before and after grokking. Generalization is achieved through periodic weights (Equation (4.3)). **(c)** Histogram of per-neuron IPRs before and after grokking – the distribution shifts to high IPR. 67
4.3 Training curves in various phases. All plots are made for networks trained with data-fraction $\alpha = 0.5$ and corruption-fraction $\xi = 0.35$. **(a)** No regularization: *Coexistence* of generalization and memorization – both train and test accuracies are high. **(b, c)** Adding weight decay: The network generalizes on the test data but does not memorize the corrupted training data, resulting in a negative generalization gap. Remarkably, the network predicts the "true" labels for the corrupted examples. We term these phases *Partial Inversion* and *Full Inversion*, based on the degree of memorization of corrupted data. ("inversion" refers to test accuracy being higher than train accuracy.) 68
4.4 Modular addition phase diagrams with respect to data fraction α and corruption fraction ξ. Left to right: different weight decay values (denoted as "wd"). Top to bottom: **(top)** Training accuracy gets worse with higher regularization strength for the same amount of data. **(middle)** Test accuracy improves accordingly. **(bottom)**: Combined phase diagrams, data used from the first two rows. 70
4.5 Modular addition phase diagrams with respect to data fraction α and corruption fraction ξ. Left to right: different Dropout values (denoted as "dp"). The behavior is very similar to weight decay, except for very large Dropout values, where the model is lost in the chaos introduced by the Dropout and fails to memorize the corrupted data. 71
4.6 Distribution of per-neuron IPR for trained networks in various phases. Coexistence phase has a bimodal IPR distribution, where the high and low IPR neurons facilitate generalization and memorization, respectively. Regularization with weight decay or Dropout shifts the IPR distribution towards higher values – Generalizing neurons get more populous compared to memorizing ones; resulting in more robust generalization and *Inversion*. All plots are made for networks trained with data-fraction $\alpha = 0.5$ and corruption-fraction $\xi = 0.35$; with various regularization strengths. 74
4.7 Progressively pruning neurons from a trained model (*Coexistence* phase), based on IPR. (data fraction $\alpha = 0.5$, corruption fraction $\xi = 0.35$). (Left) Pruning out neurons starting from the lowest IPR. The accuracy on corrupted training data decreases, providing evidence that low IPR neurons are responsible for memorization. Test accuracy as well as accuracy on uncorrupted training data remains high. (Right) Pruning out neurons starting from the highest IPR. Test accuracy decreases providing evidence that high IPR neurons are responsible for generalization. Accuracy on corrupted training data remains high. 75

4.8 Modular Addition phase diagrams and IPR histograms with different architectures and weight decay values (denoted by "wd"). The IPR histograms are plotted for neurons connected to the input (embedding) layers. 76

5.1 The dataset. The tasks are labeled by vectors $(a, b) \in \mathbb{Z}_p^2$. Each table contains examples of $ax + by$ mod p. A fraction $1 - \alpha$ of the examples is blacked out; while the remaining examples are flattened into a single "document" in the batch. Each document is organized as a collection of triples $(x, y, ax + by)$ for x, y from the training set (*i.e.* not blacked out in the table). To leverage training compute more effectively we predict not just the last element, but also other elements colored in red. Our training is similar to the traditional next-token prediction (autoregressive); with the main difference that we predict every third token, which are marked in red (x and y are uncorrelated). Every task appears exactly the same number of times in each batch. 81

5.2 Preview of results. **(a)** Phase diagram for a six-layer model. We find four different phases. (1) *in-distribution memorization*: The model *only* performs well on tasks (a, b) *and* examples (x, y) from the training set – it does not generalize on unseen examples or tasks. (2) in-distribution generalization: model generalizes on unseen examples (x, y) but not on unseen tasks (a, b). (3) out-of-distribution memorization: model generalizes on unseen tasks (a, b) but only for examples (x, y) it has seen during training. (4) out-of-distribution generalization: model generalizes on unseen tasks (a, b) for seen as well as unseen examples (x, y). We focus on investigating phase (4) in more detail. **(b)** In-context sample complexity. Accuracy of the model in phase (4) as a function of the number of few-shot examples. **(c)** Representations developed by one of the attention heads in the first layer. These are projections of the embedding of a pair of numbers onto the two largest principal components (PCs) of the internal representation formed after passing through the attention layer and projection matrix. **(d)** First 3 PCs of embeddings separate log_{27}-annotated numbers into even/odd planes, with 0 sandwiched between them. 82

5.3 The performance of the model is characterized into 4 phases, based on their accuracy on seen/unseen tasks and seen/unseen inputs. 84

5.4 Structured selection of pre-training tasks and sequences. (a) Schematic of pre-training task selection with the Rectangular rule. (b) "Balanced" sequence design with 2 tasks and 2 examples per task. 85

5.5 Phase diagram for the depth $d = 6$ models. Accuracy on all four sets used to plot the 5.2 phase diagram, with an early stopping applied. Notably, in the regions when models generalize to o.o.d. sets, the pre-training performance degrades. . . 86

5.6 Selected training curves for experiments in Figure 5.5. **(a, b)** $\alpha = 0.6$ training accuracy and o.o.d. test accuracy (dotted line). For $n_{\text{i.d.}} = 2^8$, we notice that the o.o.d. generalization ability of the model first improves then degrades as we train longer; **(c, d)** $\alpha = 0.6$, loss and accuracy vs context length, measured on $S_{\text{test}}^{\text{o.o.d.}}$ at the end of training, where for $n_{\text{i.d.}} = 2^8$ case the ICL ability fades away. . . . 87

5.7 Phases of depth $d = 4$ and $d = 2$ models. With decreasing model capacity, the performance on both sets degrades. At the same time, the transient nature of ICL does not appear. **(a, b)** from left to right: accuracy phase diagrams on pre-training set $S^{\text{i.d.}}_{\text{train}}$ and on o.o.d. test set $S^{\text{o.o.d.}}_{\text{test}}$, with early stopping; loss and accuracy vs context length on o.o.d. test set $S^{\text{o.o.d.}}_{\text{test}}$ for $\alpha = 0.6$. 90
5.8 $d = 4$ and $d = 2$ models' performance on k-shot inference, on the grid of inputs $(x, y) \in \mathbb{Z}_p^2$ (task vector = $(6, 6)$). **row 1:** Models' predictions on o.o.d. task of the type $(x_1 \;\; y_1 \;\; z_1 \;\; \cdots \;\; x_k \;\; y_k \;\; z_k \;\; x \;\; y \;\; ?)$. **row 2:** Analytical plots showing predictions solely based on Modular Regression algorithm. **row 3:** Subtract row 2 from row 1, by using correct=1 and incorrect=0. The red points correspond to the examples where Ratio Matching does not give the correct predictions but the model predicts correctly. The blue points are examples that the model missed despite Ratio Matching being applicable. This row tells us about the model's ability to implement Modular Regression by *combining* the in-context examples. Note that $d = 4$ model readily learns to combine previous examples, while its $d = 2$ counterpart struggles due to its limited capacity. 91
5.9 Models that generalize o.o.d. (left) exhibit more structured attention maps. Additionally, the top-2 principal components of the features from the corresponding heads also show more structured patterns. The features are computed for sequences with an o.o.d. task vector $(a^t, b^t) = (6, 6)$, loop over (x_i, y_i) at a specific shot while the previous parts of the sequence are fixed. We annotate each PCA plot with the $(\log_{27} x_i, \log_{27} y_i)$ mod p pairs. **(a)** The principal components form a circle of circles where the position of the outer circle is controlled by x_i. This pattern remains the same for different task vectors or the shot choices; **(b)** Only plotted pairs with even $\log_{27} x_i$, with each $\log_{27} x_i$ circled with different colors. The PCA pattern forms a similar double circle as those in (a), with the key difference that those circles depend on task vector choices and the shot choices; **(c, d)** Models without o.o.d. generalization ability. We pick heads from the first block that corresponds to the first column of (a). Clearly, the structure of attention maps and PCA patterns deteriorate as the task diversity decreases. 94
5.10 Cosine-similarities (Equation (5.2)) between layer outputs $\boldsymbol{h}^l$ at token z position ($\cos(\boldsymbol{h}^l_z(x, y), \boldsymbol{h}^l_z(x', y'))$, first row) and token y position ($\cos(\boldsymbol{h}^l_y(x, y), \boldsymbol{h}^l_y(x', y'))$, second row) reveal patterns in the models' internal representations. For clarity, we only show selected x and x' values, where y and y' values range from 0 to 28 between each tick. For the $d = 4$ model, kaleidoscopic patterns in the third layer indicate the generation of all possible y_i/x_i features for subsequent computations. The last layer shows an algorithmic shift from Ratio Matching to Modular Regression. The $d = 2$ model also shows the kaleidoscopic pattern in the first layer, while the second layer identifies the relevant y/x features from the in-context examples for matching. 95
5.11 MLP activations (post-ReLU) with respect to different input (x, y) pairs. 97

6.1 Accuracy of predicting the last number (token) in the sequence: phase diagrams w.r.t. various depths and n_{heads} values. (a) $m_{\text{test}} = 2048$, constant width: $d_{\text{model}} = 768$. (b) $m_{\text{test}} = 4096$, width scaled proportionally: $d_{\text{model}} = 128 \times n_{\text{heads}}$. 106

6.2 Training/test accuracy curves for predicting last number (token). (a) **FM:** ($m = 2048, \text{depth} = 1, n_{\text{heads}} = 1, d_{\text{model}} = 768$) Test accuracy "groks" when training accuracy reaches near-100%. (b) **UM:** ($m_{\text{test}=2048}, \text{depth} = 6, n_{\text{heads}} = 4, d_{\text{model}} = 768$) Test accuracy "groks" simultaneously with training accuracy on sequences with period longer than context length ($\mathcal{T}_m > L = 256$), indicating delayed discovery of underlying rules. 107
6.3 Bit-wise periods in an example LCG sequence generated with $m = 2048, a = 293, c = 1033$, which follows the Hull-Dobell theorem. In binary representation w-th lowest bit has a period of 2^w, for $w \in \{1, \ldots 11\}$. Writing a new sequence by skipping every 2nd step ($r = 2$) reduces the periods of all the bits by a factor of 2, rendering the lowest bit constant. $r = 2^k$ reduces bit-wise periods by a factor of 2^k, with last k digits constant. 109
6.4 **FM:** Test accuracy for $m = 2048$, $\text{depth} = 1$, $n_{\text{heads}} = 1$, $d_{\text{model}} = 768$, averaged over a, c, and seeds. (a) Test accuracy w.r.t. token positions. Ladder-like structure appears, with jumps occurring at 2^k-th positions. (b) We represent numbers as an eleven-digit binary number ($2048 = 2^{11}$) and compute the per-digit test accuracy of model predictions. 112
6.5 **FM:** ($a = 1589, c = 629$) Embedding layer. (a1) 1st principal component groups the numbers mod 2. (a2) 2nd and 3rd principal components group numbers mod 4. (b) Embedding vectors of different numbers exhibit high cosine similarity when they are spaced 2^k apart, with the similarity increasing with k. 113
6.6 **FM:** ($m = 2^{11}, a = 1589, c = 629$) (a) Attention weights: each query attends most strongly to the tokens 2^k and 2^{k-1} distance backward, for the highest possible value of k, enabling copying of lowest k bits. The other faint lines facilitate the prediction of higher bits. (b) Post-ReLU hidden layer MLP activations at token position $t = 129$ (extracted using sequences with different x_0) as a function of the target number x_{130} which it is supposed to predict. Each neuron gets activated only while predicting a specific x_{130}, exhibiting a sparse, periodic pattern. (c) Output of the MLP block projected onto the (un)embedding matrix; after masking out all but a single given hidden-layer neuron. The green dot denotes the value at the target number. Each neuron resolves the correct prediction up to a periodic structure. (d) Output of the MLP block projected onto the (un)embedding matrix; after combining the signal from multiple neurons (i.e. gradually un-masking the neurons). The per-neuron periodic patterns constructively interfere at the correct output. 114
6.7 **UM**: PCA analysis of the embedding layer. 118
6.8 **UM**: PCA analysis of attention heads specializing in different prime factorizations and their effect on per-digit accuracy. **(a1, b1, c1)** PCA patterns of outputs extracted from corresponding heads. The first two principal components group numbers based on their remainder modulo 14, modulo 4, and modulo 3, respectively. **(a2, b2, c2)** Test accuracy for individual digits when numbers are represented according to Equation (6.4) for given m_{test}. **(a3, b3, c3)** Test accuracy for individual digits after pruning the corresponding heads shown in panels (a1, b1, c1). Pruning these specific heads substantially degrades performance on their corresponding digits. 120

6.9 **UM**: Attention head (layer 1, head 6) for estimating m_{test}. (a) The query consistently attends to keys corresponding to the largest values within the given sequence; (b) the head generates features with the highest cosine similarity to m_{test}; (c) when patching the feature with sequences generated using $m_{\text{patch}} = 1024$ but maintaining the same a and c, the model often makes predictions with values of n smaller than m_{patch}. 121
6.10 Visualization of base-b tokenization and abacus embeddings. Abacus embedding 1 is shared by all the digits within the integer, while Abacus embedding 2 varies within the digit but is shared by all integers. 124
6.11 **FM**: (a) Average test accuracy ($m = 2^{32}$) vs number of in-context sequence elements. (b) The number of in-context sequence elements required to achieve 100% test accuracy (minimum of 5 runs). (See Appendix E.9.3 for details) 125
6.12 **UM**: The number of in-context sequence elements required to achieve 60% test accuracy shows a sublinear dependence on the modulus m. The scaling depends on the compatibility between m and tokenization. (a) base-2^8 tokenization, (b) base-3^5 tokenization. 125

7.1 **(a)** DNA from the module's perspective: Each token follows its own trajectory through the network. There is no notion of depth or width. With the help of zero compute experts (identity operation), different tokens may have different path lengths. **(b)** DNA from the token's perspective: The forward pass consists of a series of steps. At any given step, each token is acted on by a module or by an identity operation (black dots). When a module that contains an attention operation acts on several tokens simultaneously, the attention pattern is computed *only* between these tokens. Consequently, the attention part of the architecture can be understood as dynamic (*i.e.* data-dependent) sparsity. 131
7.2 **(a)** Path distribution of top-1 trained DNA vision model: we plot the distribution of paths that tokens take through the model, collected from the test split of ImageNet. The color corresponds to the frequency of the paths and is used in the sub-figure **(b)**. Surprisingly, the distribution of paths through the random model also follows a power law with exponent -1. **(b)** We highlight the paths that different patches take through the DNA by their frequency. We see very clear *emergent* specialization of paths: some paths focus on the object, others on the background, and others on the edges and boundaries. This visualization also illustrates the sparse nature of the attention: patches of different colors do not attend to each other in later steps of the forward pass. 132
7.3 **(a)** Path distribution of top-1 trained DNA language model: we plot the distribution of paths that tokens take through the model. The color corresponds to the frequency of the paths and is used in the sub-figure **(b)**. The distribution is a power-law with exponent -1.2. Similarly, we find that paths through the random model are distributed according to a power-law with exponent -1. **(b)** Examples of tokens that follow frequent (low rank) and rare (high rank) paths. We see that paths specialize in different ways: versions of the verb "to be", embedding of "." which we view as sentence-level attention, adjectives, etc.. 132

7.4 **Top-Left**: Test accuracy on ImageNet validation set during training for DNAs and the dense baseline. **Top-Right** Effective number of compute nodes used per step, for DNA models at the end of training. In both cases, the models exhibit higher diversity in routing decisions near the output. **Bottom**: Flow of patches plotted using the validation set of the ImageNet for the top-1 (left) and top-2 (right) models. Each column of circles contains *all* modules in the DNA. The size of the circles is proportional to the frequency with which each module is activated. If a circle is absent, then it is never activated. We see that the models develop a dense foundation and later split into distributed sparse networks. 139
7.5 **Left**: Paths followed by the patches on the right. The color corresponds to the rank. **Right**: Patches that follow the highlighted paths. The patches that pass through lower-rank paths share a higher-level commonality (color and edges), while patches that go through higher-rank paths are associated with the more specific concept (brass instruments and puzzle pieces). 140
7.6 Three examples of reconstructed images: bell pepper, hummingbird, and Welsh springer spaniel (top to bottom). Images are reconstructed by maximizing the total weight on *all* routing decisions made from step 0 to step i of the forward pass. We can see that the early layers (1-3) are primarily concerned with texture and edges, intermediate layers (4-6) with lighting distributions, and the remainder (7-10) host larger-scale features. The final reconstructed images (top to bottom) are classified as "spotlight" ($p = 0.4826$), "hummingbird" ($p = 0.5502$), and "papillon" ($p = 0.4393$). In the last two cases, all top 5 model guesses are birds and dog breeds correspondingly, reflecting hierarchical nature of ImageNet. Top 5 model guesses for the first row are: spotlight, matchstick, candle, car mirror, snowmobile. 142
7.7 A schematic representation of the dreaming procedure. During inference the DNA chooses a trajectory for each token, building a computational graph towards the output shown in (a). Our dreaming procedure accepts a subset of token trajectories and context from the input and then optimizes the remainder of the input to maximize the weight on those trajectories (b-f). The red boundary delineates the group of tokens whose decisions we maximized. The upper rows have context from the original image in this region while the lower row of each example does not. The main finding is that the boundary patches have non-local information about both object and background: when these patches are provided as a context then upon dreaming procedure generates the object and some background. The patch patterns also nicely illustrate the emergent sparse attention. 144
7.8 Distribution of computational cost on the ImageNet validation set for the top-2 DNA (25% skip) model. We observe that (i) the compute follows roughly Gaussian distribution; we believe that it is a reflection of the dataset rather than the model. (ii) the amount of compute that the model spends on the image correlates with the visual content. Namely, the model prioritizes boundary patches, so in high compute images almost everything looks like a boundary, whereas in low compute images almost everything is a background. 146

7.9 **Left**. Statistics of module reuse and representative images for the top-1 and top-2 DNA models. We see that high model reuse is dedicated to the complex images that lack and object to be segmented. For those images attention tends to spread everywhere. Low module reuse images are less interpretable, but we surmise that most of the image is taken by the object. **Top-Right** Correlation between module reuse for top-1 and top-2 DNA models. Both models tend to reuse modules on the same images (high correlation). **Bottom-Right**. Correlation between module skip and module reuse for the top-2 DNA model. The model uses different features to save compute and to reuse modules (low correlation). 148
7.10 Models trained with 21B tokens sampled from FineWeb-Edu dataset. **Top-Left**: Validation loss measured on a subset that we leave aside. **Top-Right**: Effective number of compute nodes used per step for DNA models at the end of training. The two models exhibit different behaviors in the early steps. Compared to the patterns observed in vision models, they both follow different trends as the number of steps increases. **Bottom**: Flow of tokens plotted using the validation set of the FineWeb-Edu dataset for the top-1 (left) and top-2 (right) DNA models. . . . 149
7.11 **Left**: Two examples illustrate how the router R_1 (at step-1) directs semantically/lexically similar tokens to specialized modules. For instance, M_9 groups tokens related to neural concepts in the first example and to "breakfast, lunch, dinner" in the second, whereas M_{29} tends to receive word-pieces that can be combined into whole words. **Right**: Tokens (separated by |) traverse distinct paths that align with semantic/lexical categories. 151
7.12 top-2 DNA model with a 30% skip rate (final validation loss 2.784). **Left**: Token flow on the FineWeb-Edu validation set. Unlike the model in Fig. 7.10, this model makes similar routing decisions across most tokens in the first few steps, which is similar to the vision cases in Fig. 7.4. **Right**: Effective top-k measured based on the token flow. 152
7.13 **Left**: Compute distribution measured on the validation set of the FineWeb-Edu dataset for the top-2 DNA model with 30% skip. We observe that the distribution is very heavily peaked around *above* the compute threshold of 30% (note the $\log$-scale of x-axis!). Nonetheless, some documents are low-compute as we show on the right. **Right**: Selected tokens that pass through ribbons with different compute. We find that sequences associated with very low-compute ribbons either contain HTML code, large number of web links, are parts of bibliography, contain many characters from the languages that our model has not learned yet, such as Arabic, Greek, Hebrew etc.. 154
7.14 **(a),(b)** Statistics of module reuse and representative sequences for the top-2 DNA models with/without skipping. We find that top-2 model reuses about 50% of the modules, while the model trained to skip reuses $\approx 23\%$. **(c)** Correlation between module reuse of two different top-2 DNA models (one with module skipping and one without). We find a lack of correlation, suggesting that skipping decisions are arbitrary. **(d)** Correlation between module skip and module reuse. We find a strong correlation, however we believe that it has a simple statistical nature: if a module is not skipped, it is more likely to be reused. 155

A.1 $\chi_{\mathcal{J}}$ Phase diagram for Vision Transformer (ViT) ($L = 24$, $n_{\text{heads}} = 8$, $d_{\text{embd}} = 256$). We present cases with pre/post-LN, with/without MLP layers. Top-left: Usual ViT – becomes everywhere-critical when $\mu = 1$; ordered otherwise. Bottom-left: Usual ViT without MLP layers. Top-right: ViT with post-LN (instead of pre-LN). Always ordered for finite σ_w. Bottom-right: ViT with post-LN, without MLP layers. 168
A.2 $\log \chi_{\mathcal{J}}$ Phase diagram for stacked Attention layers with/without MLP layers ($L = 24$, $n_{\text{heads}} = 8$, $d_{\text{embd}} = 256$). We use $\log \chi_{\mathcal{J}}$ and cut off the colorbar at $\chi_{\mathcal{J}} = 10^1$ because the $\chi_{\mathcal{J}}$ values in this case span from ~ 50 orders of magnitude. 168
A.3 $\log(\mathcal{J}^{l_0,l})$-$\sqrt{l}$ for $\mu = 1$, $\sigma_b^2 = 0$, erf. 203
A.4 $\log - \log$ plot of partial Jacobian $\mathcal{J}^{0,l}$ vs. l for (A) LN-erf and (B) erf-LN. 212
A.5 Test accuracy for LayerNorm applied to preactivations. $\sigma_b^2 = 0.5$ for all cases. Correlation lengths are calculated using analytical results of $\chi_{\mathcal{J}}^l$. 213
A.6 Training performance of MLP networks with erf activation function; and LayerNorm applied to preactivations. It continues to train for several orders of magnitude of σ_w^2 (with learning-rate tuning). 214

B.1 Phase portrait of the UV model for different learning rates η. The critical learning rate is $\eta_c = 0.5$ and the maximum learning rate is $\eta_{\text{upper}} = 1.0$. 229
B.2 Training trajectories of the UV model trained on a single example with $\|\boldsymbol{x}\| = 1$ and $y = 2$ using MSE loss and GD: (a, d) NTP with $n = 1$, $\sigma_w^2 = 0.5$, (b, e) NTP with $n = 512$, $\sigma_w^2 = 1.0$, and (c, f) μP with $n = 512$, $\sigma_w^2 = 1.0$. 230
B.3 *UV model shows all four training regimes:* Training trajectories of the UV model trained on a single example $(\boldsymbol{x}, y)$ with $\|\boldsymbol{x}\| = 1$ and $y = 2$ using MSE loss and gradient: (a, d) NTP with $n = 1$ and $\sigma_w^2 = 0.5$ (b, e) NTP with $n = 512$ and $\sigma_w^2 = 1.0$, and (c, f) μP with $n = 1$ and $\sigma_w^2 = 1.0$. 231
B.4 These plots are equivalent to Figure 3.2(a, d) ($\eta_c = 0.5$), but with training trajectory and local vector field plotted for every other step in $(\Delta f, \beta)$ plane. The tilted dash-dotted line indicates the $\eta\lambda = 2$ line. 234
B.5 Training loss and sharpness trajectories of ReLU FCNs in NTP trained on a 5k subset of CIFAR-10 examples using MSE loss and GD: (a, c) $\sigma_w^2 = 0.5$, (b, d) $\sigma_w^2 = 2.0$. 237
B.6 Comparison of SGD trajectories with their GD counterpart for a 3-layer FCNs in SP with $\sigma_w^2 = 0.5$ trained on a subset of CIFAR-10 consisting of $5,000$ training examples with MSE loss. The learning rate is scaled as $\eta = c/\lambda_0^H$ and batch sizes (a-c) $B = 512$, (d-f) $B = 128$, (g-i) $B = 32$ are considered. GD trajectories are plotted using solid lines with transparency. 238
B.7 Same setting as Figure B.6, except we used 3-layer FCNs in μP with $\sigma_w^2 = 2.0$. . 239
B.8 Training trajectories of (a, b) a 5-layer CNN in SP with $n = 64$, and (c, d) ResNet-18 with LayerNorm in SP, also with $n = 64$. Both models are trained on the CIFAR-10 dataset with MSE loss using SGD. The learning rate is scaled as $\eta = c/\lambda_0^H$ and batch size is $B = 512$. In panel (d), λ_t^H becomes negative during early training. This is due to the power iteration method returning the largest eigenvalue by magnitude. 241

B.9 Training loss, sharpness and test loss trajectories of Transformers trained on Wikitext-2 using cross-entropy loss: (a, b, c) Pre-LN Transformers and (d, e, f) Pre-LN Transformer without last LayerNorm. 242
B.10 Sharpness and Weight Norm of 3-layer ReLU FCNs in SP with $\sigma_w^2 = 1/3$, trained on a subset of CIFAR-10 with $5,000$ examples using GD. 242
B.11 Sharpness and Weight Norm of 3-layer ReLU FCNs in SP with $\sigma_w^2 = 1/3$, trained on a subset of CIFAR-10 with $5,000$ examples using GD. 243
B.12 Weight Norm of each layer in 3-layer ReLU FCNs (same experiments as Figure B.10): (a, b, c) SP with $\sigma_w^2 = 1/3$. All results are obtained by training on a subset of CIFAR-10 with $5,000$ examples using GD. 243
B.13 Weight Norm of each layer for 5-layer CNNs in SP (same experiments as Figure B.8(a, b): (a) Total weight norm; (b-f) Weight norm of each layer. We see that for $c = 16$, the initial catapult in sharpness λ^H (Figure B.8(b)) is accompanied by a catapult in total weight norm. Notably, the total weight norm and per-layer weight norm, whether catapults (a, c-e) or not (b, f), show a decreasing trend during the early sharpness decreasing stage, followed by an eventual increase. 244
B.14 Sharpness trajectories of ReLU FCNs trained on a 5k subset of CIFAR-10 examples using MSE loss and GD. (a, b, c) SP with $\sigma_w^2 = 0.5$, SP with $\sigma_w^2 = 2.0$ and μP with $\sigma_w^2 = 2.0$. (d, e, f) the bottom row shows the effect of setting the last layer to zero at initialization. The dashed lines in the sharpness figures show the $2/\eta$ threshold. 245
B.15 Phase diagram of EoS for 3-layer FCNs trained on CIFAR-10 with MSE loss using SGD with three different batch sizes: (a, b) $B = 512$, (c, d), $B = 128$, and (e, f) $B = 32$. The color indicates the value of $\eta\bar{\lambda}^H/2$, where $\bar{\lambda}^H$ is obtained by averaging λ_t^H over the last 200 steps. Except for the batch size, all settings are identical to Figure 3.4. Black dash-dotted lines indicate the phase boundary $\eta\bar{\lambda}^H/2 = 1$. For clarity, these lines are generated from data smoothed with a Gaussian kernel. 246
B.16 Phase diagram of EoS for 5-layer CNNs in SP with width $n = 64$ trained with MSE loss using SGD for $10,000$ steps with learning rate $\eta = c/\lambda_0^H$ and batch size $B = 512$. 247
B.17 Phase diagram of EoS for ResNet-18 in SP with width $n = 64$ trained with MSE loss using SGD for $10,000$ steps with learning rate $\eta = c/\lambda_0^H$ and batch size $B = 512$. For $s = 1$, the average eigenvalue $\bar{\lambda}^H$ is observed to be less than $2/\eta$. Upon detailed investigation of the trajectories, we found that λ_t^H oscillates around a value lower than $2/\eta$. We leave this anomalous behavior as an observation. . . . 247
B.18 4-layer FCN in SP with width $n = 512$ trained on a subset of 5000 examples of MNIST with MSE loss using GD. Both power spectrums are computed using the last 1000 steps of the corresponding trajectories. 249
B.19 4-layer FCN in SP with width $n = 512$ trained on a subset of 5000 examples of Fashion-MNIST with MSE loss using GD. Both power spectrums are computed using the last 1000 steps of the corresponding trajectories. 249
B.20 4-layer FCN in SP with width $n = 512$ trained on a subset of 5000 examples of CIFAR-10 with MSE loss using GD. Both power spectrums are computed using the last 1000 steps of the corresponding trajectories. 249

B.21 5-layer CNN in SP with width $n = 32$ trained on a subset of 1000 examples of CIFAR-10 with MSE loss using GD. 250
B.22 ResNet-18 in SP with width $n = 32$ trained on a subset of 1000 examples of CIFAR-10 with MSE loss using GD. 250
B.23 Bifurcation diagrams, late-time sharpness trajectories, and power spectrum of a 2-layer linear network trained on the power-law dataset for different parameter values: (a-c) $B_x = 0.0, B_y = 0.0$, (d-f) $B_x = 1.0, B_y = 0.0$, (g-i) $B_x = 0.0, B_y = 1.0$, and (j-l) $B_x = 1.0, B_y = 1.0$. All power spectrums are computed using the last 1000 steps of the corresponding trajectories. 251
B.24 2-layer linear FCN in μP trained on the teacher-student task. Both power spectrums are computed using the last 1000 steps of the corresponding trajectories. 252
B.25 4-layer ReLU FCNs in μP trained on the teacher-student task. Both power spectrums are computed using the last 1000 steps of the corresponding trajectories. 253
B.26 4-layer linear FCNs in SP with $\sigma_w^2 = 0.5$ trained on the generative CIFAR-10 task with MSE loss using GD. Both power spectrums are computed using the last 1000 steps of the corresponding trajectories. 253
B.27 4-layer ReLU FCNs in SP with $\sigma_w^2 = 0.5$ trained on the generative CIFAR-10 task with MSE loss using GD. Both power spectrums are computed using the last 1000 steps of the corresponding trajectories. 254

C.1 *Forgetting* phase (data fraction $\alpha = 0.5$, corruption fraction $\xi = 0.35$, weight decay = 20). (a) Training curves. (b) Frobenius norms of network weights with training. 258
C.2 Network output logits on 4 different (corrupted) training examples examples, in various phases ($N = 500, \alpha = 0.5, \xi = 0.35$). Blue dots correspond to the logit value for the "correct" labels. Red dots correspond to the logit value for the corrupted labels. 260
C.3 IPR distributions of networks with/without BatchNorm; the correlation between BatchNorm weights and IPR. The IPR distributions in the two models do not exhibit significant differences. However, BatchNorm helps generalization by assigning higher weights (γ_k) to high IPR neurons compared to low IPR neurons. Both models are trained at data fraction $\alpha = 0.65$ and corruption fraction $\xi = 0.2$, with batch-size = 64. 261
C.4 Phase diagram with BatchNorm (post-BN) 262
C.5 Phase diagram with LayerNorm (post-LN) 263
C.6 Phase diagram with mini-batch training, without any explicit regularization or normalization layers. 263
C.7 The IPR histograms of the Embedding layer for Transformers with different configurations. We see that depth does not affect the IPR distribution, whereas more heads encourages high IPR embeddings and weight tying discourages it. All models are trained with corruption fraction $\alpha = 0.8$ and corruption fraction $\xi = 0.2$. . 265

C.8 Modular Addition phase diagrams and IPR histograms with different dropout probabilities (denoted by "dp"). The IPR histograms are plotted for neurons connected to the embedding layer. We select points trained with data fraction $\alpha = 0.9$ and corruption fraction $\xi = 0.1$. 266
C.9 Phase diagrams and the IPR histograms of the columns of input layer weight and rows of the output layer weights for 3-layer MLPs with different parameterizations. Selected models are trained with data fraction $\alpha = 0.8$ and corruption fraction $\xi = 0.2$. 268
C.10 IPR of a 3-layer MLP in coexistence phase, trained with data fraction $\alpha = 0.8$ and corruption fraction $\xi = 0.1$. We plotted row-wise IPR for input layer weight and column-wise IPR for output layer weight; the correlation between column-wise IPR of output layer weights with the absolute value of BatchNorm weights right before it. The correlation between the BatchNorm weights and IPR values we pointed out in the main text still holds. However, we did not observe a strong correlation between row-wise IPR of the input layer and the BatchNorm weight next to it. 269
C.11 Pruning experiments in all various phases ($N = 500, \alpha = 0.5, \xi = 0.35$). Pruning affects generalization and memorization differently in different phases because the proportion of high and low IPR neurons in trained networks depends on the phase. 271
C.12 Pruning experiments in all various phases ($N = 500, \alpha = 0.5, \xi = 0.35$). Pruning affects generalization and memorization differently in different phases because the proportion of high and low IPR neurons in trained networks depends on the phase. 273
C.13 Training curves with dropout (a) dp=0.0 (b) dp=0.3 (c) dp=0.6 (d) dp=0.9 274
C.14 Training curves in various phases with BatchNorm. All networks are trained with corruption fraction $\xi = 0.3$ and batch-size = 64; with variable data fractions. (a) Coexistence ($\alpha = 0.4$) (b) Partial Inversion ($\alpha = 0.5$) (c) Full Inversion ($\alpha = 0.8$) 274
C.15 Training curves in various phases with LayerNorm. All networks are trained with corruption fraction $\xi = 0.3$ and batch-size = 64; with variable data fractions. (a) Coexistence ($\alpha = 0.4$) (b) Partial Inversion ($\alpha = 0.5$) (c) Full Inversion ($\alpha = 0.8$) 275
C.16 Training curves in various phases without any normalization, but with mini-batches. All networks are trained with corruption fraction $\xi = 0.3$ and batch-size = 64; with variable data fractions. (a) Coexistence ($\alpha = 0.4$) (b) Partial Inversion ($\alpha = 0.5$) (c) Full Inversion ($\alpha = 0.8$) 275
C.17 Modular Addition trained with CrossEntropy Loss. Robustness to label corruption is weaker compared to MSE. 277
C.18 Phase diagram with weight decay, ReLU. 278
C.19 Phase diagram with Dropout, ReLU. 278
C.20 Phase diagram for width $N = 5000$ fully connected network with quadratic activation. Regularization with weight decay. Compared to $N = 500$ network in Figure 6.1(a), the network generalizes marginally poorly. Note: the absence of *Partial Inversion* phase and smaller *Coexistence* phase with $wd = 0.0$. This can be attributed to the increased capacity to memorize due to abundance of neurons. 279

C.21 Phase diagram for width $N = 5000$ fully connected network with ReLU activation. Regularization with weight decay. In this case, the network performance is drastically improved compared to $N = 500$ (Figure C.18). The confusion phase is largely eliminated, due to the increased capacity of the network, as a result of abundance of neurons. 279
C.22 Phase diagram for Modular Multiplication, weight decay. All other settings are identical to Figure 4.4. Note that the phase diagram is identical to Modular Arithmetic case – representative of the similarity in the algorithms learnt by the network. 280
C.23 Pruning as many low IPR neurons as possible while retaining generalization, for various network widths ($\alpha = 0.5, \xi = 0.35$). The curves are averaged over 40 random seeds. Note that the x-axes on both plots have a logarithmic scale. 281

D.1 $d = 6$ phase diagram with task vectors randomly selected, selected the best results from two random seeds. Learning rate $\eta = 10^{-4}$. 285
D.2 $d = 4$ training curve, with loss and accuracy measured per task for the last in context example. Trained with $n_{\text{i.d.}} = 2^8$ and $\alpha = 0.8$. There are six tasks learned almost instantly: $(5, 26)$, $(21, 23)$, $(28, 27)$, $(3, 9)$, $(10, 10)$, $(5, 9)$. 286
D.3 PCA over the embedding layers of $d = 2$ models with different random seeds. Each number is annotated by its $\log_{27}$ value. 288
D.4 PCA over the embedding layers of $d = 4$. Each number is annotated by its $\log_{27}$ value. 288
D.5 PCA over the features of specific heads for $d = 4$ model. Each number is annotated by its $\log_{27}$ value. We plotted PCA over (a, b) (x, y) and (c) (y, z) concatenated features. Where the circle in (b, c) heavily depends on the sequence and (a) remains unchanged. 289
D.6 PCA over the same head in Figure 5.9(a), with feature concatenated differently. We annotate numbers with their corresponding logarithm. 290
D.7 All attention heads in $d = 2$ model. 291
D.8 PCA for concatenated $(x, y)/(x, z)$ features for all heads in $d = 2$ model. 291
D.9 More PCA analysis similar to Figure 5.9. Top-2 components of PCA contribute $\sim 40\%$ variance of the layer 1 head and $\sim 20\%$ variance of the layer 2 head, while the top-4 components contribute $\sim 60\%$ and $\sim 40\%$ accordingly. 292
D.10 PCA analysis of Attention and MLP outputs. Attention modules contribute $\sim 50\%$ and $\sim 30\%$ to top-4 components of PCA of layer 1 and layer 2. MLP modules contribute $\sim 10\%$ and $\sim 12\%$ accordingly. 292
D.11 Cosine-similarity of $d = 4$ model for every block, measured at 2-shot. 293
D.12 Cosine-similarity of $d = 4$ model for every block, measured at 16-shot. 294
D.13 Cosine-similarity of $d = 2$ model for every block, measured at 2-shot. 294
D.14 Cosine-similarity of $d = 2$ model for every block, measured at 10-shot. 295
D.15 Training curves for $d = 4$, averaged over three random seeds. 295
D.16 Training curves for $d = 2$, averaged over three random seeds. 296
D.17 Phase diagrams on all four sets for $d = 4$ and $d = 2$. 297
D.18 We compare the best performance for $d = 6$ models with different $n_{\text{i.d.}}$ values, averaged over three seeds. We use learning rate $\eta = 10^{-4}$ for $p = 37$ and $p = 47$, while keeping other hyperparameters the same as $p = 29$. 298

E.1 The impact of training dataset parameters (n_m, n_a, n_c) on unseen modulus task performance. . . . 300
E.2 Heatmap of test accuracy of a 6 layer Transformer with learning rate and weight decay as the axes. As the modulus is increased from 1024 to 16,384 the range of hyperparameters resulting in reasonable accuracy becomes narrow. . . . 301
E.3 Test accuracy and train/test loss for $m = 2048 = 2^{11}$, depth=1. (a,b,c) $n_{\text{heads}} = 1$ (d,e,f) $n_{\text{heads}} = 4$. . . . 307
E.4 Test accuracy and train/test loss for $m = 7776 = 2^5 \cdot 3^5$, depth=1. (a,b,c) $n_{\text{heads}} = 1$ (d,e,f) $n_{\text{heads}} = 4$. . . . 307
E.5 Test accuracy heatmaps of with depth and embedding dimensions as the two axes. 308
E.6 **FM:** Comparison between $m = 2039$ (prime) and $m = 2048$ (power-of-two), with both models trained for 50,000 steps. Each model has depth 2 and $d_{\text{model}} = 1024$. The test set consists of sequences with the same periods, while the training set includes arbitrary multipliers not present in the test set. . . . 309
E.7 **UM:** Test accuracy comparison between $m = 2039$ (prime), $m = 1801$ (prime) and $m = 2048$ (power-of-two). The model was trained for 100,000 steps on a dataset consisting of 262,144 sequences with 512 distinct training moduli not present in the test set. . . . 310
E.8 (left) Comparison of the training accuracy of sequences with different periods relative to the context length 512, (center) Accuracy when the model is only trained on sequences with period < 512, (right) Accuracy when the model is trained on sequences with period > 512. . . . 310
E.9 Test accuracy vs token positions for $m = 2048 = 2^{11}$, depth=1, various values of a, c. (a,b) $n_{\text{heads}} = 1$ (c,d) $n_{\text{heads}} = 4$. The accuracies for different a and c are exactly on top of each other. . . . 311
E.10 Test accuracy for $m = 2048 = 2^{11}$, depth=1, $n_{\text{heads}} \in \{1, 4\}, d_{\text{model}} = 768$. (a,d) Test accuracy averaged over a, c and initial seeds. (b,e) Multiplication of per-digit test accuracies – matches exactly with the average test accuracy. (c,f) Per-digit accuracy in binary representation. . . . 312
E.11 Test accuracy for $m = 7776 = 2^5\, 3^5$, depth=1, $n_{\text{heads}} \in \{1, 4\}, d_{\text{model}} = 768$. (a,d) Test accuracy averaged over a, c and initial seeds. (b,e) Multiplication of per-digit test accuracies – matches exactly with the average test accuracy. (c,f) Per-digit accuracy in RNS representation. . . . 312
E.12 Projections along Top 6 principal components of the embedding matrix, for $m = 512$, depth=1, $n_{\text{heads}} = 1, d_{\text{model}} = 768$. . . . 313
E.13 Projections along Top 6 principal components of the embedding matrix, for $m = 7776$, depth=1, $n_{\text{heads}} = 1, d_{\text{model}} = 768$. . . . 314

E.14 Per-digit test accuracies in binary representation, with attention masking (at inference time); for $m = 2048 = 2^{11}$, depth=1, $n_{\text{heads}} = 1, d_{\text{model}} = 768$, averaged over a, c and initial seeds. (a) For each query, only the attention to key at position $t - 2^k$ (where $k = \lfloor \log_2 t \rfloor$) is kept. All other attention weights are masked out. In this case, the model can successfully copy the last k bits, but cannot predict higher bits. (b) For each query, only the attention to key at positions $t - 2^k$ and $t - 2^k$ are kept. All other attention weights are masked out. In this case, the model can predict higher bits with fairly high accuracy – this is remarkable given that majority of the attention weights are masked out. 315
E.15 PCA analysis of first-layer heads. Although the ordering of numbers varies, the grouping behavior discussed in Section 6.4.3 remains invariant to changes in a, c, m, x_0, and t. Note that the last head is the head that appears in Figure 6.9, which does not exhibit a strong grouping bias. 317
E.16 Per-digit accuracy for $m_{\text{test}} = 2048$ after pruning specific attention heads. (a) Results from the same model used in Section 6.4.3, identical to Figure 6.8 (b2); (b) Performance after pruning the attention head responsible for grouping numbers by their values modulo 3, which is irrelevant for solving sequences with m_{test} (containing only the prime factor 2). After pruning, the model's performance shows marginal improvement for specific early bits at lower token positions; (c) Performance after pruning the attention head responsible for grouping numbers by their values modulo 14. The model's performance on m_{test} decreases significantly, as this head partially contributes to processing relevant prime factors. However, since the primary head responsible for binary representation remains intact, the model maintains partial functionality. 317
E.17 Per-digit accuracy for $m_{\text{test}} = 2352$ after pruning specific attention heads. (a) Results from the same model used in Section 6.4.3, identical to Figure 6.8 (b2); (b) Performance after pruning the attention head responsible for grouping numbers by their values modulo 3, which is relevant for one specific digit in this case. After pruning, the model's performance shows a clear degradation, with the strongest one happening exactly at the digit corresponding to modulo 3 (c) Performance after pruning the attention head responsible for grouping numbers by their values modulo 2. The model's performance on m_{test} got obliterated, as there are many base-2 digits in the RNS representation of m_{test} in this case. 318
E.18 Patching experiments following the setting of Figure 6.9 in the main text. None of these heads, after patching, make the model believe that the modulus is close to m_{patch}. 318

E.19 Attention patterns and token distance statistics for layer 2, head 3 of the model analyzed in Section 6.4.3. (a1, b1) Results for a sequence with $m_{\text{test}} = 2048$, $a = 5$, and $c = 31$. The statistics reveal that the model consistently looks back at distances that are multiples of 4, which divides m_{test} and enables the correct copy behavior. (a2, b2) Results for a sequence with $m_{\text{test}} = 2352$, $a = 85$, and $c = 5$. In contrast to panels (a1, b1), the same attention head now consistently looks back at distances that are multiples of 14, which allows the model to copy lower digits from the context. This adaptive behavior demonstrates that the model has acquired the ability to dynamically adjust its lookback distance by r iterations to copy the lower digits and leave the higher digits to the later layers for computation. 320

E.20 Test accuracy vs. Number index for $m = 2^{20}$. First Row: Three models trained on the same dataset, each using a unique PyTorch random seed that controls model initialization and batch shuffling. Second Row: Three models trained on different datasets, with each dataset generated using a unique random seed controlling NumPy randomness for sampling a, c, and x_0. All models converged to solutions that achieved and sustained 100% test accuracy, but differed in the number of in-context examples required to reach this performance. 323

E.21 Test accuracy vs Number index for $m = 2^{32}$. First Row: Three models trained on the same dataset, each using a unique PyTorch random seed that controls model initialization and batch shuffling. Second Row: Three models trained on different datasets, with each generated using a unique random seed controlling NumPy randomness for sampling a, c, and x_0. Only one of the five models found a solution that achieved and sustained 100% test accuracy. 324

E.22 Median number of in-context sequence elements required to achieve 100% test accuracy across five runs. The shaded region represents the min-max range. . . . 325

E.23 Test accuracy vs Number index. In (a), the moduli 2048 and 16384 (blue curves) have the same root 2 as the tokenization base 256. The model performs better on these two moduli. In (b), the moduli 2178 and 19683 (orange curves) have the same root 3 as the tokenization base 243. The model performs better on these two moduli. 326

F.1 Patch/Token counts for each module (exclude skipping, note that the module index does not start from 0 whenever we use skip.) for all DNA models we used in main text. **Top**: vision models. **Bottom**: language models. 331

F.2 Reconstruction of the hummingbird and welsh springer spaniel for the top-1 and top-2 models (some parts repeated from main text). Reconstructions are broken up into maximizing path weights from steps $0-S$ (first and third rows) and maximizing decision weights for just step S (second and fourth rows). We see that the top-2 model exhibits interesting low level features before the sixth router, while the top-1 model does not. This is because the top-1 model has an almost input-independent selection of modules before that router. Additionally we see in both cases the same transition at router 7 where global features develop corresponding to the original image. 334

F.3 Series of experiments where we attempt to recover the image from the paths of a group of tokens. The red boundary delineates the group of tokens whose decisions we maximized. Each recovery comes in two varieties: The upper rows have the context from the original image in the region surrounded by the red boundary, while the lower row of each example does not. The main finding is that the boundary patches contain non-local information about both the object and the background. When these patches are provided as context, the dreaming procedure generates the object and some background. The patch patterns also nicely illustrate the emergent sparse attention. 335
F.4 **Left**: Distribution of computational cost on patches from the ImageNet validation set for the top-2 DNA (25% skip) model, which is the same model as the one in Fig. 7.8. **Right**: High compute patches generally contain more textures and boundaries, while low compute patches tend to appear visually flatter. 336
F.5 Illustration of randomly selected patches processed through the same ribbon in a randomly initialized top-2 DNA model. Patches following the same ribbon display strong visual similarities, likely due to the critical initialization that preserves the similarities among different input patches. 337
F.6 Full trajectories through the top-1 DNA language model on Wiki example. 339
F.7 Full trajectories through the top-1 DNA language model on artificial example. . . 340
F.8 **Left**: Compute distribution per token measured on the validation set of the FineWeb-Edu dataset for the top-2 DNA model with 30% skip. **Right**: Selected tokens that pass through ribbons with different compute. We find that tokens associated with very low-compute ribbons are usually particular objects, concepts, or symbols that do not play a crucial role in sequences. 341
F.9 A top-1 DNA-n model (76.4% validation accuracy), where the Transformer blocks are separated into Attention/MLP sub-blocks. **Left** Patch flows collected over ImageNet validation set; **Right** Path distribution. 342
F.10 Visualization patches that go through the same paths for the model shown in Fig. F.9. **Left**: A layout of single image goes through the model; **Left**: Patches that go through a low-rank path. **Right**: Patches that go through a high-rank path. 343
F.11 A top-2 DNA-n model (78.9% validation accuracy), where the Transformer blocks are separated into Attention/MLP sub-blocks. **Left**: Patch flows collected over ImageNet validation set; **Right**: Ribbon distribution. 343
F.12 Visualization patch ribbons for the model shown in Fig. F.11. **Left**: A layout of single image goes through the model; **Middle**: Patches that go through a low-rank ribbon; **Right**: Patches that go through a high-rank ribbon. 344
F.13 Effective top-k for DNA-n models (vision), as a function of number of forward pass steps s. Early on the models are essentially dense, while later on they become more distributed. **Left**: top-1 DNA-n model. **Right**: top-2 DNA-n model. 344
F.14 A top-1 DNA-n model (final validation loss 2.708), where the Transformer blocks are separated into Attention/MLP sub-blocks. **Left** Token flows collected over FineWeb-Edu validation subset; **Right** Ribbon distribution. 345
F.15 A top-2 DNA-n model (final validation loss 2.688), where the Transformer blocks are separated into Attention/MLP sub-blocks. **Left**: Token flows collected over FineWeb-Edu validation subset; **Right**: Ribbon distribution. 346

F.16 Tokens that go through different rank paths/ribbons. **Left**: top-1 model; **Right**: top-2 model. 346
F.17 Effective top-k for DNA-n models (language), as a function of number of forward pass steps s. The trend is very different from the vision models we have shown earlier. **Left**: top-1 DNA-n model. **Right**: top-2 DNA-n model. 347

G.1 Copyright information for NeurIPS: https://neurips.cc/FAQ/Copyright 348
G.2 Copyright information for ICLR: https://iclr.cc/FAQ/Copyright 348
G.3 Copyright information for ICML: https://icml.cc/FAQ/Copyright 349

List of Abbreviations

AGI	Artificial General Intelligence
AI	Artificial Intelligence
AM	Autoregressive Model
APJN	Averaged Partial Jacobian Norm
CNN	Convolutional Neural Network
CSE	Cross-Entropy Loss
DNA	Distributed Neural Architecture
DNN	Deep Neural Networks
EoS	Edge of Stability
Erf	Error Function
FCN	Fully Connected Network
FFN	Feadforward Neural Network
FM	Fixed Modulus
GELU	Gaussian Error Linear Unit
GPT	Generative Pretrained Transformer
i.d.	In Distribution
i.i.d.	Independent and Identically Distributed
ICL	In-context Learning
IPR	Inverse Participation Ratio
JKL	Jacobian Kernel Loss
LCG	Linear Congruential Generator
LLM	Large Language Model
LSD	Least Significant Digit
ML	Machine Learning
MLP	Multi-Layer Perceptron

MoE	Mixture-of-Experts
MSE	Mean Squared Error
NNGP	Neural Network Gaussian Process
NTK	Neural Tangent Kernel
NTP	Neural Tangent Parameterization
o.o.d.	Out of Distribution
PC	Principal Component
PCA	Principal Components Analysis
PRNG	Pseudo-Random Number Generators
ReLU	Rectified Linear Unit
RoPE	Rotary Positional Encoding
SGD	Stochastic Gradient Descent
SoTA	State of The Art
SP	Standard Parameterization
`UM`	Unseen Modulus
wd	Weight Decay
μP	Maximal Update Parameterization

Chapter 1: Introduction

In this chapter, we provide a brief overview of recent developments in Artificial Intelligence (AI) and outline several key challenges currently faced in building better Large Language Models (LLMs). Then we summarize our contributions, which are motivated by these challenges and aimed at addressing them.

1.1 Challenges from Scaling Up

1.1.1 Pre-LLM Era

The landscape of artificial intelligence (AI) changed dramatically in 2011 with the development of the first GPU-efficient backpropagation implementation. This computational breakthrough triggered a fundamental shift in AI research philosophy – from crafting carefully designed, specialized algorithms to embracing general-purpose, scalable computation. The true impact of this transformation became evident with AlexNet [10], a model containing 60 million parameters trained on 1.2 million high-resolution images. By surpassing all existing machine learning methods of its time, AlexNet marked a watershed moment in AI history.

Since this pivotal achievement, scalability has become the cornerstone of modern deep learning. However, researchers soon encountered fundamental limitations when increasing model

depth. For example, VGG [11] suffered from the vanishing-gradient problem at large scales, until the introduction of Kaiming Initialization [12]. Unlike the collection of ad hoc "tricks" previously used to improve training, Kaiming Initialization introduced an extra factor of 2 in the variance of randomly initialized weight matrices, derived from the condition that the signal scale remains $O(1)$ as it propagates through the network.

While this method temporarily alleviated the problem, computing every detail on a case-by-case basis quickly proved impractical, especially as networks became more complex and the $O(1)$ initialization factors that strongly influence model behavior became essentially impossible to determine via calculation. This challenge underscored the need for a deeper, more systematic approach for initializing the network.

About a year later, ResNet [13, 14] addressed this issue by a different approach. Here Kaiming introduced residual connections, which allow information to flow directly from one layer to another without modification, effectively creating shortcuts through the network. The design was intended to form an ensemble of shallower sub-networks, reducing the network's sensitivity to $O(1)$ initialization factors. However, the optimal way to implement these residual connections remained unclear[1]. This uncertainty again called for a principled framework to guide future architecture designs from an initialization perspective.

1.1.2 LLM Era

The pursuit of large-scale computing reached another milestone with the introduction of the Transformer architecture [15] in 2017. Its decoder-only variant ultimately led to GPT-3 [16], a 175-billion-parameter autoregressive model trained on internet-scale data, and later to today's

[1]The existence of two different versions of ResNet serves as evidence of this uncertainty.

GPT-5 and other large language models (LLMs). It is worth noting that the original Transformer used a different residual connection scheme compared to contemporary LLMs, where the main distinction lies in the choice of how to connect normalization layers with residual connections. This difficulty is similar to the ResNet problem, which echos with what we have discussed in the last subsection. Consequently, practitioners can only try with experiments and intuition, without a clear and practical guidance.

Despite the absence of a thorough theoretical explanation for the superiority of the current so-called Pre-LayerNorm choice, it has been widely adopted due to its robustness with respect to $O(1)$ hyperparameter choices in initialization. The discovery of scaling laws [17, 18, 19, 20, 21, 22] further reinforced the belief that, with the right architectural and optimization choices, performance can scale predictably as models grow larger.

However, echoing Anderson's famous argument that "more is different" [23], it becomes clear that while the elementary building blocks of deep learning remain unchanged across scales, fundamentally new behaviors, principles, and challenges emerge as models grow larger. Specifically, two notable "clouds" now appear on the horizon:

1. The parameter efficiency of the widely adopted Pre-LayerNorm design deteriorates as models deepen [24, 25]. The formerly "blessed" intuition of residual networks as ensembles of shallower subnetworks appears increasingly problematic at extreme scales.

2. Hyperparameter tuning is becoming infeasible, as training a single large language model can cost millions of dollars in practice. This cost prohibits systematic exploration and demands more principled strategies for optimization.

These challenges mark a turning point. Empirical experimentation alone is no longer suffi-

cient – the available computational resources cannot exhaustively probe such vast design spaces. A deeper theoretical and phenomenological framework is urgently needed to provide predictive understanding and practical guidance for the next generation of large-scale models.

1.1.3 Post-LLM Era

Looking beyond LLMs, the pursuit of Artificial General Intelligence (AGI) inevitably demands the integration of additional functionalities and capabilities into these systems. As this evolution continues, both the computational cost and the structural complexity of training and inference pipelines are growing at an unprecedented pace.

More importantly, it is essential for us to develop a fundamental understanding of models at their current scale. Such understanding enables us to anticipate and interpret the new phenomena that are likely to emerge as models expand further in size, scope, and capability. Only through this foundational comprehension can we guide the development of future models toward truly general intelligence.

1.2 Emergence from Scaling

In the previous section, we emphasized the importance of understanding scaling behavior through theoretical and phenomenological approaches, and identified several promising directions for continued scaling. In this section, we turn our attention to one of the most intriguing consequences of scaling: emergent capabilities, or more broadly, emergence.

Emergence in general Emergence is a phenomenon extensively studied across many disciplines, particularly within the natural sciences. In these contexts, it refers to complex patterns,

behaviors, or properties that arise from the interactions of many simpler components, which are not present in the individual parts themselves. In other words, these components often self-organize in ways that are hard to interpret or infeasible to precisely describe at a microscopic level (smaller scale). In fact, even in cases where a system can be described in detail at the microscopic level, it does not mean one can easily predict the emergence of mesoscopic or macroscopic phenomena at larger scales[2].

Emergence in neural networks In the neural network context, which is the primary focus of this dissertation, emergence manifests itself similarly: as complexity is added to the system (through additional layers, parameters, or data used for training), unforeseen structures and behaviors arise from the interactions of simpler components. These emergent phenomena appear across different organizational scales – from individual parameter groups, to entire layers, and to complete network functions.

The importance of understanding emergence is perhaps best illustrated by in-context learning (ICL) [16], one of the most powerful emergent capabilities of LLMs. ICL enables an LLM to perform unseen tasks simply from examples presented in its input, without any parameter updates. This remarkable ability remains absent in smaller systems and manifests only when both the volume of training data and the number of model parameters reach sufficiently vast scales [16].

Despite being one of the most significant phenomena discovered in recent years, the precise conditions governing its emergence remain poorly understood. From a microscopic perspective, it is highly unlikely that one could determine how to arrange billions of parameters such that a

[2]This unpredictability may manifest as rapid changes with respect to the scale or complexity of the system.

particular ability arises. Consequently, one must adopt an emergent viewpoint by asking under what conditions, or through what macroscopic observables, these phenomena can be effectively captured and understood. In this spirit, there is a clear need for a series of tractable systems whose complexity can be gradually and controllably increased. In other words, we want a system with neural networks and some datasets that are simple enough to allow controlled experimentation, yet rich enough to capture the essential mechanisms underlying large-scale neural networks.

The importance of studying emergence is also evident from the behavior of scaling laws. In practice, scaling laws are used to predict a model's average performance at scales where extensive experimentation is infeasible. However, while scaling laws [17, 18] provide remarkably accurate predictions of a model's ability to fit training data, they fail to indicate when specific capabilities will emerge. The often abrupt nature of emergence makes it difficult to rely solely on scaling laws to anticipate a qualitative change at larger scales. Because those laws are an average measure, they reveal little about the abrupt transitions that mark qualitative improvements, which potentially undermines their predictive power and can even run counter to our goal of building more capable and reliable systems if misinterpreted.

Understanding and eventually predicting the emergent behavior, therefore, requires careful mechanistic analysis of simplified models that preserve the core properties of real-world systems. Such investigations can reveal under what conditions and how networks internalize structure and capture patterns within data. This, in turn, allows us to improve the pipeline design by identifying the crucial information that current models fail to exploit.

We believe that this approach will help advance the scientific foundations of deep learning and guide the next generation of model development.

1.3 Summary and Outline

Given all the discussions above, we now present our efforts toward addressing the aforementioned challenges. In the remainder of this dissertation, we adopt a theoretical and phenomenological approach, inspired by methods in physics, to investigate the behavior of deep neural networks. In particular, we focus on the following topics:

1. We examine how initialization schemes and architectural design principles determine network behavior at large depths.

2. We analyze the dynamics of sharpness under gradient descent and its relationship with initialization, parameterization, and feature learning across different network widths.

3. We quantify the conditions under which specific skills emerge using modular arithmetic datasets across various model classes, and investigate the underlying mechanisms through mechanistic interpretability studies.

4. We propose distributed neural architectures (DNAs) and elucidate how data properties lead to drastically different emergent connectivity patterns and specializations that challenge the current paradigm of architecture design.

Principled Architecture Design and Initialization (Chapter 2 [1, 2]) In this part, we study the interaction between initialization and architectural design before training begins, adopting methods from statistical physics and nonlinear dynamics [26, 27, 28]. We find that, in the large-width limit, both the forward and backward passes (the Jacobian) at initialization can often be reduced to one-dimensional maps as functions of depth, where the architecture and the scale of

initialization determine the nature of these maps. By analyzing the fixed-point properties of these maps, we identify the *critical initialization* condition under which the map neither diverges nor converges to a trivial fixed point. Furthermore, we explore an architectural design that guarantees the map remains well-behaved at large depth, providing a theoretical explanation for the architecture design principles widely adopted in modern deep learning [14].

Building on this analysis, we also develop `AutoInit`, an algorithm that automatically tunes arbitrary architectures to their critical initialization. This approach enables efficient discovery of suitable initializations for architectures that are otherwise difficult to analyze analytically.

Sharpness Dynamics (Chapter 3, [3]) During gradient descent, the top eigenvalue of the loss Hessian, often referred to as sharpness, exhibits robust and universal patterns throughout training. We show that the dynamics of a simple two-layer linear network trained on a single example can be reduced to a two-dimensional map in function space, which successfully captures the essential sharpness phenomena observed in real-world networks. Furthermore, by adopting an appropriate initialization and parameterization scheme, the resulting low-dimensional dynamics become independent of network width. This analysis offers an intuitive and theoretically grounded understanding of the conditions under which effective feature learning emerges in complex, high-dimensional systems.

Emergent Capabilities (Chapters 4 to 6 [4, 6, 7]) As discussed in the previous section, we seek a simplified experimental setting that allows us to clearly understand the emergence of skills within deep neural networks. The dataset we choose as the foundation of this setting is the modular arithmetic dataset. These datasets are known for exhibiting the *grokking* phenomenon [29],

in which a model undergoes a transition from overfitting to generalization as training progresses or as the dataset size increases.

A key advantage of modular arithmetic datasets is that they admit analytically tractable settings, where the learned weight matrices can be expressed in closed form [30]. Building on these analytical insights, we investigate how grokked models internally represent the algorithms they learn, focusing on the tension between memorization and generalization, especially when the training data is partially corrupted, a case of practical importance.

Extending this analysis to Transformer architectures trained on a diverse collection of modular arithmetic tasks, we identify the conditions under which in-context learning (ICL) [16] emerges, enabling the model to generalize to unseen datasets. Moreover, we reverse engineered the model and examined the compositional structure through which it combines simple learned skills to perform ICL [31].

Finally, by training Transformers to infer pseudo-random number generators (PRNGs), we explore the limits of their capability as sequence learners. Our results show that the models learn to transform input sequences into high-dimensional representations that exhibit remarkably simple and interpretable structures. Moreover, we observe the emergence of additional capabilities beyond a certain dataset size and model depth, which enable the model to effectively "crack" the underlying PRNGs.

Distributed Neural Architectures (Chapter 7 [8]) All the models used or studied above are built upon a fundamental principle of network design: a fixed, homogeneous sequential stack of layers. This design has proven both simple and effective, yet it may impose intrinsic limitations as model scales extend far beyond current practice. To explore alternatives, we introduce a new

architecture that permits arbitrary network topologies. This architecture is trained end-to-end, allowing functional structures to emerge naturally from data and optimization. We then perform interpretability analyses on the resulting learned architectures, providing insights into how real data from different domains influence and shape architectural organization.

Chapter 2: Emergent Depth Scale at Initialization

In this chapter, we discuss how network initialization influences the trainability of *deep* neural networks and how architectural design choices affect their sensitivity to initialization. This chapter also introduces the terminology and conceptual framework that will be used throughout the remainder of this dissertation to describe deep neural networks.

2.1 Introduction and Related Work

Most architectures used in deep learning can be summarized by the following set of forward-pass equations, applied to a N_{in}-dimensional input $\boldsymbol{x} \in \mathbb{R}^{d_{\text{in}}}$:

$$\begin{aligned} \boldsymbol{h}^0(\boldsymbol{x}) &= \text{Embedding}_{\boldsymbol{\theta}^0}(\boldsymbol{x}), \qquad \boldsymbol{h}^0(\boldsymbol{x}) \in \mathbb{R}^{N_0} \\ \boldsymbol{h}^l(\boldsymbol{x}) &= f^l_{\boldsymbol{\theta}^l}(\boldsymbol{h}^{l-1}(\{\boldsymbol{x}\})), \qquad 1 \leq l \leq L \text{ and } \boldsymbol{h}^l(\boldsymbol{x}) \in \mathbb{R}^{N_l}, \end{aligned} \tag{2.1}$$

where $\{\boldsymbol{x}\}$ indicates that the function $f^l_{\boldsymbol{\theta}^l}(\cdot)$ may act on multiple inputs to produce $\boldsymbol{h}^l(\boldsymbol{x})$. The final output of the network is $\boldsymbol{h}^L(\boldsymbol{x})$. Each of $\text{Embedding}_{\boldsymbol{\theta}^0}$, $f^1_{\boldsymbol{\theta}^1}, \cdots, f^L_{\boldsymbol{\theta}^L}$ is usually a continuous function[1] parameterized by $\boldsymbol{\theta}^0, \ldots, \boldsymbol{\theta}^L$. Although we describe the parameters $\boldsymbol{\theta}^l$ in a vectorized form here for generality, they are typically matrices or vectors in practice. Throughout this dissertation, we refer to $\boldsymbol{h}^l(\boldsymbol{x})$ as the feature at layer l.

[1] These functions are typically nonlinear; the set of non-differentiable points has measure zero.

At initialization, the parameters $\boldsymbol{\theta}^0, \ldots, \boldsymbol{\theta}^L$ are independently sampled from some distribution, typically a Gaussian or uniform distribution with zero mean and finite variance. To perform a given task, the network is then trained using a gradient-based optimization method via back-propagation, where the parameter update is given by

$$\boldsymbol{\theta}^l \longrightarrow \boldsymbol{\theta}^l - \eta\, g\left(\nabla_{\boldsymbol{\theta}^l} \mathcal{L}(\boldsymbol{h}(\boldsymbol{x}), \boldsymbol{y}), \cdots\right), \tag{2.2}$$

where $\boldsymbol{y}$ denotes the target (or label) corresponding to the input $\boldsymbol{x}$, $\mathcal{L}$ is the loss function, η represents the learning rate, and g is a continuous function that post-processes the gradient $\nabla_{\boldsymbol{\theta}^l} \mathcal{L}(\boldsymbol{h}(\boldsymbol{x}), \boldsymbol{y})$, potentially using additional information (abbreviated as $\cdots$).

In this chapter, we focus exclusively on vanilla gradient descent, where g is the identity map and $\cdots$ represents an empty set.

Like any optimization problem, initialization plays a crucial role in determining whether the network parameters converge to a minimum – if not a global one. In our setting, as mentioned in Chapter 1, we aim to choose initialization schemes that remain effective across varying network depths. More precisely, this goal reduces to selecting an appropriate distribution, particularly its mean and variance, from which the parameters $\boldsymbol{\theta}^l$ are sampled, such that the model exhibits stable and well-behaved training dynamics from the very first optimization step.

We define "well-behaved" initialization from the perspective of signal propagation. A suitable initialization should satisfy the following conditions:

1. The forward pass should not lead to numerical instabilities.

2. It should preserve correlations between different input examples; equivalently, the char-

acteristic correlation length governing information decay should remain larger than the effective network depth L.

3. During back-propagation, gradient information should be able to flow from the output layer to the input layer without exploding or vanishing exponentially.

In general, these conditions are hard to ensure analytically for arbitrary networks with depth L and width $N_1, \cdots, N_{L-1}$. However, when the number of parameters in each layer becomes large, i.e. $N_1, \cdots, N_{L-1} \longrightarrow \infty$, the description of neural networks dramatically simplifies, which allows us to study depth scaling theoretically in simpler settings. Specifically, the forward propagation often becomes a Gaussian process in this limit [26, 32, 33, 34], with a kernel referred to as the Neural Network Gaussian Process (NNGP) kernel, and the back-propagation is also simplified.

2.1.1 Related Work

The aforementioned properties enable us to study complex neural networks with a few collective variables and their scaling behavior. Furthermore, in this limit, special choices of parameterization, small learning rates and Mean-squared Error (MSE) loss function leads to the linearization of gradient descent dynamics – which can be solved exactly in terms of the neural tangent kernel (NTK) [35, 36]. A large body of work has been devoted to calculating the NNGP kernel and the NTK for different architectures in infinite-width limits, as well as finite width corrections to these quantities [37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 56, 57, 58, 59, 60]. These results were leveraged to analytically determine the critical initialization of parameters in simple architectures [27, 28, 32].

2.2 Multi-Layer Perceptrons (MLPs)

In this section, we make the argument from the previous section concrete by considering multi-layer perceptrons (MLPs) as a representative example.

A vanilla MLP network is defined by restricting Equation (2.1) to a specific form:

$$\begin{aligned} h_i^0(\boldsymbol{x}) &= \sum_{j=1}^{N_{\text{in}}} w_{ij}^0 x_j + b_i^0 \\ h_i^l(\boldsymbol{x}) &= \sum_{j=1}^{N_{l-1}} w_{ij}^l \phi(h_j^{l-1}(\boldsymbol{x})) + b_i^l \,, \qquad 1 \leq l \leq L \,. \end{aligned} \tag{2.3}$$

Here, the weights $\boldsymbol{w}^l \in \mathbb{R}^{N_l \times N_{l-1}}$ and biases $\boldsymbol{b}^l \in \mathbb{R}^{N_l}$ are explicitly written out to replace the general parameter $\boldsymbol{\theta}^l$, and $\phi(\cdot)$ is some element-wise nonlinear activation function. For clarity, we have introduced an extra set of indices $i, j, \cdots$ to remove the vector notation from the features (preactivations) $\boldsymbol{h}^l(\boldsymbol{x})$.

Here the weight matrices $\boldsymbol{w}^l$ are i.i.d. sampled from a normal distribution $\mathcal{N}(0, \sigma_w^2/N_l)$; and the biases $\boldsymbol{b}^l \in \mathbb{R}^{N_l}$ are i.i.d. sampled from a normal distribution $\mathcal{N}(0, \sigma_b^2)$. This choice was made based on previous work [61], which ensures the model is well-defined when we take the infinite width limit $N_1, \cdots, N_{l-1} \longrightarrow \infty$. Note that the exact choice of initialization distribution is not distinguishable in our language, as long as it does not have a long tail.

As we will see soon, the choice of hyperparameters, which includes the zero mean and the values of σ_w, and σ_b, will still play a crucial role as the network depth, L, increases.

2.2.1 Emergent Depth Scale in the Forward Pass

First, we would like to argue why a zero mean is necessary, which can be checked by simply measuring the one-point function for the features. For simplicity, we omit the bias and focus on the recurrence formula from Equation (2.3):

$$\begin{aligned}\mathbb{E}_\theta[h_i^l] &= \mathbb{E}_\theta\left[\sum_{j=1}^{N_{l-1}} w_{ij}^l \phi(h)_j^{l-1} + b_i^l\right] \\ &= \sum_{j=1}^{N_{l-1}} \mu_w \mathbb{E}_\theta[\phi(h)_j^{l-1}] + \mu_b\,,\end{aligned} \tag{2.4}$$

where the expectation $\mathbb{E}_\theta[\cdot]$ is taken by averaging over the distribution in which the collection of all parameters θ is sampled.

From the above formula, we see that when $\phi(\cdot)$ is an $O_{N,L}(1)$ function, the one-point function diverges exponentially with the layer index l, unless $\mu_w \lesssim O(1/N_l)$. Although, in principle, one could choose a mean of order $1/N_l$ to keep the result finite, this approach has two drawbacks: (i) it makes initialization difficult to manage, as it introduces an additional hyperparameter, the mean, alongside the variance (to be discussed shortly); and (ii) a nonzero mean induces a preferred direction, for which there is no prior justification. Therefore, in the infinite-width limit $N_l \to \infty$, the only reasonable choice is $\mu_w = 0$. Likewise, for the bias term, one should choose $\mu_b = 0$ to avoid a linear growth of the one-point function with layer index l.

With a vanishing first moment, the magnitude of the forward signal is then captured by the variance of $\boldsymbol{h}^l$. Specifically, we care about the feature variance $\sum_i^{N_l} \left(h_i^l(\boldsymbol{x})\right)^2 / N_l$, and how this quantity changes with layer index l. Again, we will average it over different parameter initializations, denoted by $\mathbb{E}_\theta[\cdot]$. Note the expectation value of the variance above is the diagonal

term of the Neural Network Gaussian Process (NNGP) Kernel defined as follows:

Definition 2.1. *The NNGP kernel is defined as the averaged covariance of the feature for different inputs:*

$$\mathcal{K}^l(\boldsymbol{x}, \boldsymbol{x}') \coloneqq \mathbb{E}_\theta \left[\frac{1}{N_l} \sum_{i=1}^{N_l} h_i^l(\boldsymbol{x}) h_i^l(\boldsymbol{x}') \right] . \tag{2.5}$$

Then its diagonal term is simply the variance we care about in this chapter $\mathcal{K}^l(\boldsymbol{x}, \boldsymbol{x})$.

Throughout this chapter, we adopt the sequential infinite width limit: $N_0 \to \infty, N_1 \to \infty, \ldots, N_{L-1} \to \infty$. In this limit, the feature variance converges in distribution to its average over parameterizations, due to the Central Limit Theorem (CLT):

$$\frac{1}{N_l} \sum_{i=1}^{N_l} (h_i^l(\boldsymbol{x}))^2 \longrightarrow \mathbb{E}_\theta \left[\frac{1}{N_l} \sum_{i=1}^{N_l} (h_i^l(\boldsymbol{x}))^2 \right] \equiv \mathcal{K}^l(\boldsymbol{x}, \boldsymbol{x}) \tag{2.6}$$

Note that this convergence guarantees that a randomly initialized network with large width will not deviate from the expected behavior predicted by theory, which allows us to remove the expectation value and ensures the practical applicability of the theoretical results. However, for simplicity, we will keep the expectation throughout this Chapter.

The main advantage for introducing the kernel above can be highlighted by the following lemma, which allows us to understand initializations in feature space instead of parameter space:

Lemma 2.2 (Lee et al. [32])**.** *When* $N_l \to \infty$ *for* $l = 1, \ldots, L-1$ *sequentially, the expectation value over parameter initializations for a general function of features:* $\mathcal{O}(\boldsymbol{h}^l(\boldsymbol{x}))$*, can be*

expressed as the averaging over the Gaussian process $\boldsymbol{h}^l(\boldsymbol{x})$ with variance $\mathcal{K}^l(\boldsymbol{x}, \boldsymbol{x})$.

$$\mathbb{E}_\theta\left[\mathcal{O}(h_i^l(\boldsymbol{x}))\right] = \frac{1}{\sqrt{2\pi\mathcal{K}^l(\boldsymbol{x},\boldsymbol{x})}} \int dh_i^l \mathcal{O}(h_i^l(x)) e^{-\frac{(h_i^l(x))^2}{2\mathcal{K}^l(\boldsymbol{x},\boldsymbol{x})}} . \tag{2.7}$$

Theorem 2.3. *Following the same infinite width limit defined earlier, the diagonal NNGP kernel is deterministic, given by the following recurrence relation:*

$$\mathcal{K}^l(\boldsymbol{x},\boldsymbol{x}) = \sigma_w^2 \mathbb{E}_\theta\left[\frac{1}{N_{l-1}}\sum_{i=1}^{N_{l-1}} (\phi(h_i^{l-1}(\boldsymbol{x})))^2\right] + \sigma_b^2 . \tag{2.8}$$

Note that this is a recurrence relation, because the expectation on the right hand side can be replaced by Lemma 2.2.

Proof. We can prove Theorem 2.5 using Wick's theorem for w^l and b^l:

$$\begin{aligned}
\mathcal{K}^l &\equiv \mathbb{E}_\theta\left[\frac{1}{N_l}\sum_{i=1}^{N_l} h_i^l h_i^l\right] \\
&= \mathbb{E}_\theta\left[\frac{1}{N_l}\sum_{i=1}^{N_l}\left(\sum_{j=1}^{N_{l-1}} w_{ij}^l \phi(h_j^{l-1}) + b_i^l\right)\left(\sum_{k=1}^{N_{l-1}} w_{ik}^l \phi(h_k^{l-1}) + b_i^l\right)\right] \\
&= \mathbb{E}_\theta\left[\left(\frac{1}{N_l}\sum_{i=1}^{N_l} w_{ij}^l w_{ik}^l\right)\sum_{j=1}^{N_{l-1}}\sum_{k=1}^{N_{l-1}} \phi(h_j^{l-1})\phi(h_k^{l-1}) + \frac{1}{N_l}\sum_{i=1}^{N_l} b_i^l b_i^l\right] \\
&= \sigma_w^2 \sum_{j=1}^{N_{l-1}} \mathbb{E}_\theta\left[\frac{1}{N_{l-1}}\sum_{j=1}^{N_{l-1}} \phi(h_j^l)^2\right] + \sigma_b^2 .
\end{aligned} \tag{2.9}$$

We have omitted the explicit dependence of $\mathcal{K}^l$ and $\boldsymbol{h}^l$ on $\boldsymbol{x}$ in Equation (2.9) for brevity, and we will use this convention frequently in the remainder of the section. □

Theorem 2.3 can be viewed as a map from $\mathcal{K}^{l-1}$ to $\mathcal{K}^l$ that depending on the choice of ϕ and two hyper-parameters σ_w and σ_b. Then, the requirement that the forward pass should not lead

to numerical instabilities in the large depth limit $L \longrightarrow \infty$, can be translated to $\mathcal{K}^l$ which should converge to a fixed point $O_L(1)$ for the map in Theorem 2.3.

Then, it is natural to ask for a fixed-point analysis. It turns out, commonly used bounded activation functions (e.g. Erf, Tanh) automatically bound $\mathcal{K}^l$, whereas unbounded activation functions (e.g. ReLU, GELU) might not have a fixed point when (σ_w, σ_b) is not well-chosen.

The stability of the fixed point in Equation (2.8) can be characterized by the following quantity [27, 56]:

$$\chi_{\mathcal{K}}^{\star} := \left. \frac{\partial \mathcal{K}^l}{\partial \mathcal{K}^{l-1}} \right|_{l \longrightarrow \infty} \tag{2.10}$$

In general, we want $\chi_{\mathcal{K}}^{\star} \leq 1$ for a finite $\mathcal{K}^{\star} > 0$ fixed point, this gives us some constraints on σ_w and σ_b. For most bounded activations, those conditions are fairly easy to satisfy. However, for unbounded activations, the fixed point often exists only at 0 or ∞. It turns out the only way to ensure numerical instability is to choose very specific values for σ_w and σ_b so $\chi_{\mathcal{K}}^{\star} = 1$. In that case, the growth or decay of the forward pass signal is usually non-exponential with depth L, which allows nice enough behavior empirically.

The quantity $\chi_{cK^{\star}}$ in fact gives us a characteristic length scale

$$\xi_{\mathcal{K}} = \frac{1}{|\log \chi_{\mathcal{K}^{\star}}|} \tag{2.11}$$

which gauges the allowed depth before we might run into numerical instabilities.

A similar analysis can be performed for the off-diagonal component of the NNGP kernel,

$\mathcal{K}^l(\boldsymbol{x}, \boldsymbol{x}+\boldsymbol{\epsilon})$, in the limit $\boldsymbol{\epsilon} \to 0$[2]. Intuitively, this quantity measures whether the network can preserve the relationships between different inputs at initialization. Analyzing this term reveals another emergent depth scale, denoted $\xi_{\mathcal{J}}$, which characterizes the propagation of backward (gradient) signals and will be discussed in the next subsection.

The correlation between forward and backward signal propagation should not be surprising: gradient signals are meaningful only if correlations between distinct data points are maintained throughout the network. Otherwise, the model would be unable to distinguish different examples, leading to degenerate or uninformative gradients.

2.2.2 Emergent Depth Scale in the Backward Pass

We start by calculating the gradient of the loss with respect to a parameter group, which can be expanded using chain-rule:

$$\nabla_{w_{ij}^l} \mathcal{L}(\boldsymbol{x}) = \sum_{k,m,n,q,\cdots} \left(\frac{\partial \mathcal{L}}{\partial h_k^L}\right) \cdot \left(\frac{\partial h_k^L}{\partial h_m^{L-1}} \cdots \frac{\partial h_n^{l+1}}{\partial h_q^l}\right) \cdot \left(\frac{\partial h_q^l}{\partial w_{ij}^l}\right) \tag{2.12}$$

The scale of this gradient depends on three terms, where the first and the last terms are controlled by the behavior of $\mathcal{K}(\boldsymbol{x}, \boldsymbol{x})$ and $\xi_{\mathcal{K}}$ (Proved in Appendix A.5.2), whereas the second term depends on the Jacobian between the features at different layers, and requires extra efforts to handle. The scaling behavior of this quantity is crucial to the backward propagation of the gradient signal, which affects the trainability of the model.

Definition 2.4 (APJN)**.** *To measure the scale of the second term in Equation* (2.12)*, we define the averaged partial Jacobian norm (APJN) as the averaged, normalized Frobenius norm of the*

[2]The order of differentiation and the $\boldsymbol{\epsilon} \to 0$ limit are not interchangeable.

Jacobian between the features of layers l and $l_0 < l$.

$$\mathcal{J}^{l_0,l}(\boldsymbol{x}) \equiv \mathbb{E}_\theta \left[\frac{1}{N_l} \sum_{i=1}^{N_l} \sum_{j=1}^{N_{l_0}} \left(\frac{\partial h_i^l(\boldsymbol{x})}{\partial h_j^{l_0}(\boldsymbol{x})} \right)^2 \right] , \tag{2.13}$$

APJN is a dominant factor in the magnitude of the gradients. Similar to previous discussions on NNGP kernels, we will show that this quantity also forms simple recurrence relations, which allows us to analyze its behavior analytically.

Theorem 2.5. *In the infinite width limit where $N_1, \cdots, N_{L-1} \longrightarrow \infty$ sequentially, APJN $\mathcal{J}^{l_0,l+1}(x)$ is deterministic and satisfies the recurrence relation*

$$\mathcal{J}^{l_0,l}(\boldsymbol{x}) = \chi_{\mathcal{J}}^{l-1} \mathcal{J}^{l_0,l-1}(\boldsymbol{x}) , \tag{2.14}$$

where the multiplicative factor $\chi_{\mathcal{J}}^l$ is given by

$$\chi_{\mathcal{J}}^l = \sigma_w^2 \mathbb{E}_\theta \left[\frac{1}{N_l} \sum_{i=1}^{N_l} \left(\phi'(h_i^l(\boldsymbol{x})) \right)^2 \right] . \tag{2.15}$$

Here $\phi'(\cdot)$ is the derivative of the non-linear activation function $\phi(\cdot)$. The proof of this theorem is detailed in Appendix A.5.1.

Again, if we view Equation (2.14) as a map, then at large l, the behavior of $\mathcal{J}^{l_0,l}$ is determined by $\chi_{\mathcal{J}}^l$, or more precisely by

$$\chi_{\mathcal{J}}^\star = \chi_{\mathcal{J}}^l|_{l\to\infty} \tag{2.16}$$

Note that the recurrence relation is easier to handle than the NNGP kernel case. If $\chi_{\mathcal{J}}^\star > 1$,

the APJN diverges exponentially, corresponding to the *chaotic phase*. This terminology originates from the interpretation of the Jacobian: a divergent Jacobian implies that the forward pass becomes extremely sensitive to small input perturbations. In this phase, two nearby inputs are mapped to completely different feature representations. Conversely, if $\chi_{\mathcal{J}}^{\star} < 1$, the APJN vanishes exponentially, which is referred to as the *ordered phase*, since distinct inputs collapse to nearly identical representations.

In both the chaotic and ordered phases, the ill-behaved APJN indicates information loss as signals propagate through the network. Unlike the $\mathcal{K}$ and $\chi_{\mathcal{K}}$ analysis, here any $\chi_{\mathcal{J}}^{\star} \neq 1$ leads to undesirable behavior: the initial correlation between nearby inputs is either lost entirely or transformed into uncorrelated, chaotic noise.

The exponential behavior also leads to a length scale. Specifically, when we write APJN as $\mathcal{J}^{l_0,l} \approx c_{l_0} e^{\pm \frac{l-l_0}{\xi}}$, with ξ as the correlation length defined as

$$\xi = \frac{1}{|\log \chi_{\mathcal{J}}^{\star}|} \tag{2.17}$$

The only way to avoid the aforementioned information loss is to maintain an infinite correlation length, i.e., $\chi_{\mathcal{J}}^{\star} = 1$, which corresponds to the phase boundary between the ordered and chaotic regimes. At this critical line, the APJN decays algebraically with depth, $\mathcal{J}^{l_0,l} \propto l^{-\zeta}$, characterized by the *critical exponent* ζ. This condition imposes constraints on the choice of (σ_w, σ_b), and any initialization satisfying this condition is referred to as a *critical initialization*. The exponent ζ can be computed explicitly, as detailed in Appendix A.8, and has been empirically verified in Figure 2.1.

When we only impose the critical initialization condition $\chi_{\mathcal{J}}^{\star} = 1$. We get a line in the

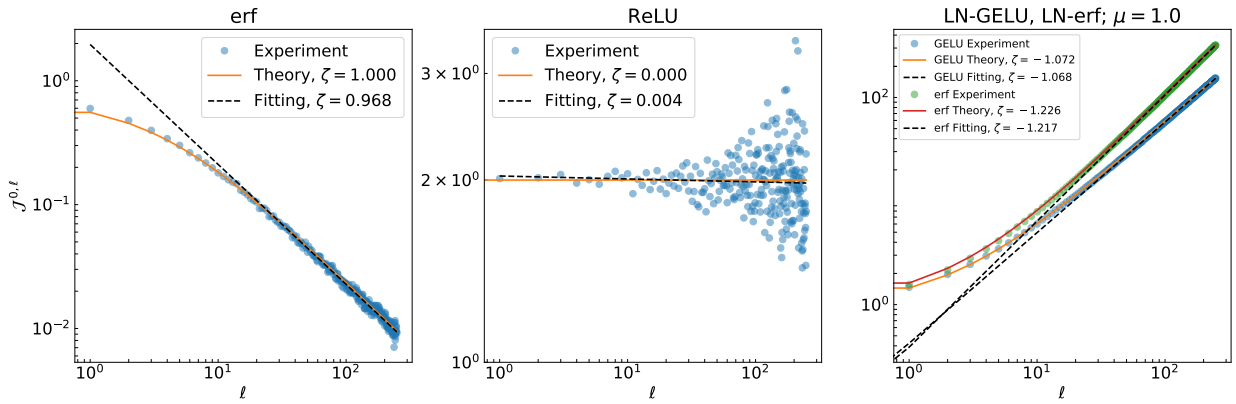


Figure 2.1: log–log plots of partial Jacobian $\mathcal{J}^{0,l}$ vs. l. From left to right: **(1)** erf, **(2)** ReLU, **(3)** erf and GELU with LayerNorm applied to preactivations and residual connections of strength 1. The fluctuations get larger towards the output because the aspect ratio (*i.e.* l/N_l) approaches $1/4$.

$\sigma_w - \sigma_b$ plane that satisfies the condition, which we call the critical line. If we further impose the forward propagation condition $\chi^\star_{\mathcal{K}} \leq 1$, depending on the bounded or unbounded activations, then it constrains the critical line further down to a critical point on the $\sigma_w - \sigma_b$ plane.

2.3 Layer Normalization

The fact that critical initialization must be at a single point $(\sigma_w^\star, \sigma_b^\star)$ is undesirable, as our target is to solve initialization for any network architectures, where the analytical calculation might be infeasible. Historically, methods like LayerNorm [62] and GroupNorm [63] were introduced to control the behavior of $\mathcal{K}^l$. We will see shortly that, although this relaxation removes the need to initialize the network precisely at a single point, it still has nontrivial effects on backward propagation. Therefore, initialization remains an important factor to consider.

We will only discuss LayerNorm, as most results can be applied to GroupNorm with a reasonable group size. Furthermore, we will focus on LayerNorm applied to $\boldsymbol{h}^l$, which is usually

referred to as pre-LayerNorm[3].

Definition 2.6 (Pre-LayerNorm)**.** *LayerNorm applied to $\boldsymbol{h}^l$, i.e. pre-LayerNorm is defined by*

$$\tilde{h}_i^l = \frac{h_i^l - \text{mean}_i[\boldsymbol{h}^l]}{\sqrt{\text{var}_i[\boldsymbol{h}^l]}}\,, \tag{2.18}$$

where we have introduced empirical mean and variance of $\boldsymbol{h}^l$ as $\text{mean}_i[\boldsymbol{h}^l] \coloneqq (\sum_{i=j}^{N_l} h_j^l)/N_l$ and $\text{var}_i[\boldsymbol{h}^l] \coloneqq \left(\sum_{j=1}^{N_l}(h_j^l - \text{mean}_j[\boldsymbol{h}^l])^2\right)/N_l$.

An MLP with pre-LayerNorm can be described by a recurrence relation similar to Equation (2.3), with $\boldsymbol{h}^l$ replaced by $\tilde{\boldsymbol{h}}^l$

$$h_i^l(\boldsymbol{x}) = \sum_{j=1}^{N_{l-1}} w_{ij}^l \phi(\tilde{h}_j^{l-1}(\boldsymbol{x})) + b_i^l\,, \qquad 1 \le l \le L \tag{2.19}$$

In the infinite width limit, it can be easily shown that $\text{mean}_i[\boldsymbol{h}^l] = 0$ and $\text{var}_i[\boldsymbol{h}^l] = \mathcal{K}^l$. Then the distribution of $\tilde{\boldsymbol{h}}^l$ is reduced to

$$\tilde{h}_i^l \to \frac{1}{\sqrt{\mathcal{K}^l}} h_i^l \sim \mathcal{N}(0,1)\,. \tag{2.20}$$

Note that the above equation holds for any l, σ_w, σ_b. Therefore, the constrains imposed by $\chi_{\mathcal{K}}$ becomes trivial. This results in a critical line rather than a critical point.

The recurrence relation (2.8) for the NNGP kernel is also modified.

$$\mathcal{K}^{l+1} = \sigma_w^2 \mathbb{E}_\theta \left[\frac{1}{N_l}\sum_{i=1}^{N_l} \phi(\tilde{h}_i^l)^2\right] + \sigma_b^2 \tag{2.21}$$

[3] Applying LayerNorm to $\phi(\boldsymbol{h}^l)$ can be handled similarly, see Appendix A.5.4 for details.

The Equation (2.15) calculating $\chi^l_{\mathcal{J}}$, consequently, needs to be modified accordingly.

$$\chi^l_{\mathcal{J}} = \frac{\sigma_w^2}{\mathcal{K}^l} \mathbb{E}_\theta \left[\frac{1}{N_l} \sum_{i=1}^{N_l} \phi'(\tilde{h}_i^l)^2 \right] \tag{2.22}$$

As we can see from the above formula, though the LayerNorm controls the forward pass signal, it has non-trivial impact on the backward gradients. This dependency is still affected by the choice of σ_w and σ_b, thus needs to be treated carefully.

2.4 Residual Connections

Another way to reduce the sensitivity with respect to hyperparameters choices for a given depth L is to reduce the effective depth. Following this intuition, the residual connection technique was introduced. It was first proposed by He et al. [13] in the context of Convolutional Neural Networks (CNNs) for image recognition [61]. Today, it is a standard component within many modern network architectures [15, 64].

The specific idea is to introduce an identity map that connects preactivations across two adjacent layers. The forward pass is simply modified to be

$$h_i^l(\boldsymbol{x}) = \sum_{j=1}^{N_{l-1}} w_{ij}^l \phi(h_j^{l-1}(\boldsymbol{x})) + b_i^l + \mu h_i^{l-1}(\boldsymbol{x}) \,, \qquad 1 \leq l \leq L \,, \tag{2.23}$$

Compared to the original work, we have introduced an additional multiplicative factor μ to adjust its strength. Setting $\mu = 1$ recovers the commonly used residual connection. Note that in this case, one must ensure the shape satisfies $N_1 = N_2 = \cdots = N_{l-1}$ for the residual connection to work. Otherwise, an extra weight matrix is required to project features from previous layers.

That will negate the advantage of the residual connection if used frequently, as one then needs to take their initialization hyperparameters into consideration.

Then the recurrence relation for the NNGP kernel (2.8) is modified to

$$\mathcal{K}^{l+1} = \sigma_w^2 \mathbb{E}_\theta \left[\frac{1}{N_l} \sum_{j=1}^{N_l} \phi(h_j^l)^2 \right] + \sigma_b^2 + \mu^2 \mathcal{K}^l \,. \tag{2.24}$$

Note the potential cross terms between $\boldsymbol{h}^{l-1}$ and $\phi(\boldsymbol{h}^{l-1})$ are all zero with the zero-mean assumption of w_{ij}^l and b_i^l.

The recurrence of APJN is unchanged, with $\chi_{\mathcal{J}}^l$ modified to be

$$\chi_{\mathcal{J}}^l = \sigma_w^2 \mathbb{E}_\theta \left[\frac{1}{N_l} \sum_{k=1}^{N_l} \phi'(h_k^l)^2 \right] + \mu^2 \tag{2.25}$$

Following fixed-point analysis, one can then argue that when $\mu < 1$ the fixed line in σ_w-σ_b plan got shifted by rescaling $\sigma_w \longrightarrow \sigma_w/(1-\mu^2)^{0.5}$, with every other conclusion unchanged.

For $\mu > 1$, there is no saving on signal propagation; every quantity one can measure will blow up, no matter what hyperparameters were used.

While at $\mu = 1$, there is no non-trivial fixed point for unbounded activation functions. However, when the activation function $\phi(\cdot)$ is bounded, i.e., $\phi(\infty) < \infty$, the signal scales in a highly non-trivial manner[4], which can be summarized by the following theorem.

Theorem 2.7. *Let $f(x)$ be an MLP with the* erf *activation function and residual connections with $\mu = 1$. Then, in the limit $N_1, \cdots, N_l \to \infty$,*

- *The NNGP kernel K^l diverges linearly with depth l, as the bounded activations saturate to*

[4] Yang and Schoenholz [65] found similar conclusions for the tanh activation

the limit $\phi(\cdot) \longrightarrow 1$ *at large* l.

- $\chi^l_{\mathcal{J}} \longrightarrow 1_+$ *(see second row of Figure 2.2):* $\chi^l_{\mathcal{J}} \approx 1 + \tilde{c}/\sqrt{l}$*, where* $\tilde{c} = 2\sigma_w^2/(\pi\sqrt{\sigma_w^2 + \sigma_b^2})$ *is a non-universal constant. Consequently, APJN diverges as a stretched exponential:* $\mathcal{J}^{l_0,l} = O(e^{\sqrt{l/\lambda}})$*, where* $\lambda = 1/(4\tilde{c}^2)$ *is the new emergent length scale.*

We refer to this case as *subcritical*. Although $\chi^\star_{\mathcal{J}}$ approaches 1_+, the APJN still diverges with depth faster than any power law. The growth is controlled by the new scale λ. To control the gradient, we aim to make λ large, which can be achieved by decreasing σ_w. In this case, trainability is enhanced (see second row of Figure 2.4).

For any unbounded activations. $\mu = 1$ with a finite σ_w will not converge to a sub-critical solution. The APJN will diverge exponentially.

2.5 Residual Connections + Layer Normalization

From the previous sections, we see that in neither case does the model become insensitive to hyperparameter choices – if anything, the situation worsens when just using residual connections. Can we then find a way to improve further?

We find that an appropriate combination of residual connections and LayerNorm yields a nontrivial architecture that facilitates stable signal propagation. The resulting recurrence relation can be expressed as

$$h_i^l(\boldsymbol{x}) = \sum_{j=1}^{N_{l-1}} w_{ij}^l \phi(\tilde{h}_j^{l-1}(\boldsymbol{x})) + b_i^l + \mu h_i^{l-1}(\boldsymbol{x})\,, \qquad 1 \le l \le L\,, \tag{2.26}$$

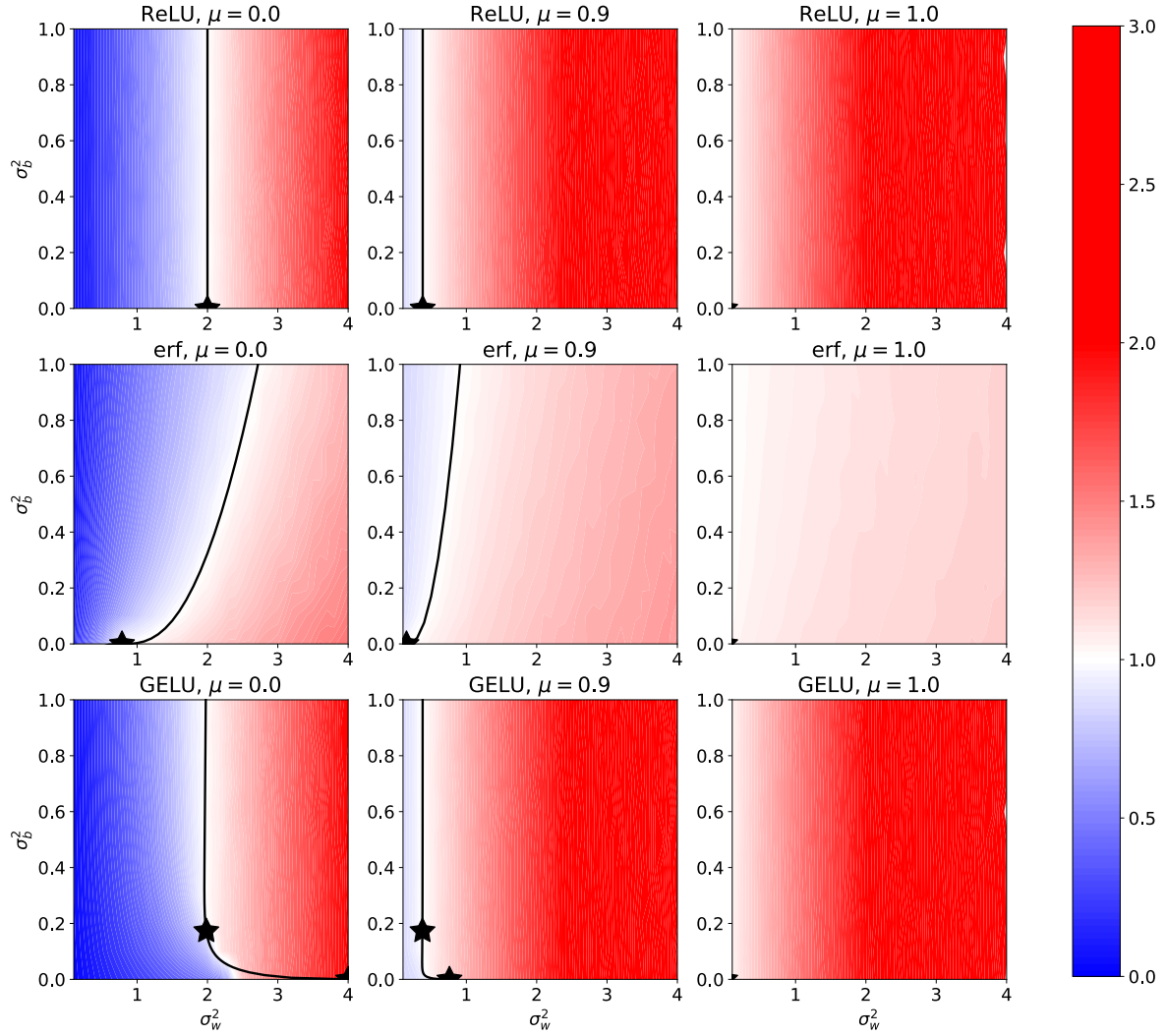


Figure 2.2: $\chi^\star_{\mathcal{J}}$ empirical phase diagrams for an MLP with $L = 50, N = 500$. The solid lines indicate the critical lines obtained through the infinite width limit calculations, and the stars indicate the critical points. We see that residual connections in general just shift the phase boundary. Except for erf activation, where the case $\mu = 1$ is subcritical and has a large correlation length.

The recurrence relation for the NNGP kernel (2.8) is modified to

$$\mathcal{K}^{l+1} = \sigma_w^2 \mathbb{E}_\theta \left[\frac{1}{N_l} \sum_{j=1}^{N_l} \phi(\tilde{h}_j^l)^2 \right] + \sigma_b^2 + \mu^2 \mathcal{K}^l \,, \tag{2.27}$$

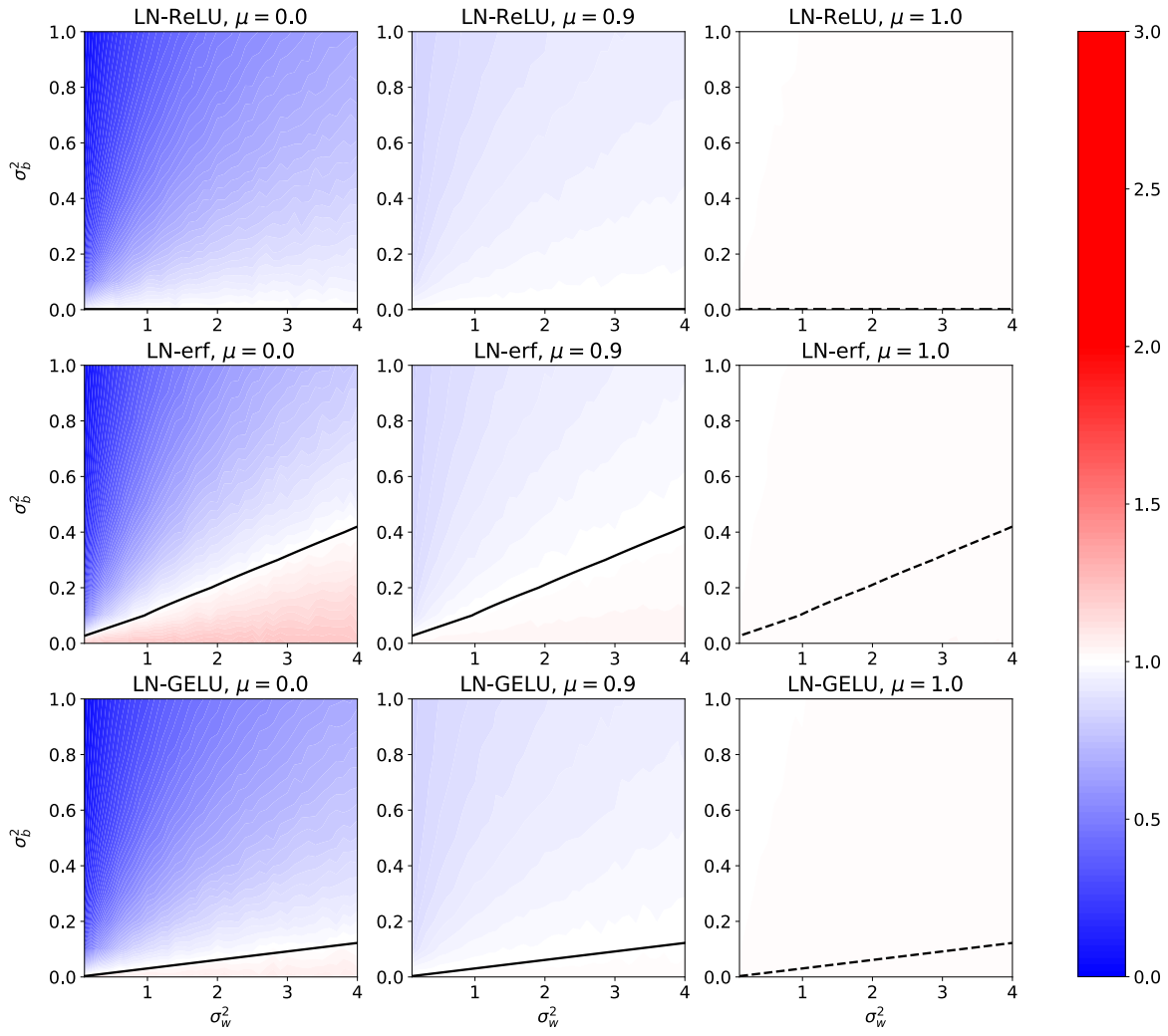


Figure 2.3: $\chi^\star_{\mathcal{J}}$ empirical phase diagrams for an MLP with $L = 50, N = 500$. The solid lines indicate the critical lines obtained through the infinite width limit calculations, and the stars indicate the critical points. The dotted lines in the rightmost column correspond to the critical lines for $\mu < 1$ case. For networks with LayerNorm and $\mu = 1$, $\chi^\star_{\mathcal{J}} = 1$ holds on the entire σ_b–σ_w plane.

while $\chi^l_{\mathcal{J}}$ becomes

$$\chi^l_{\mathcal{J}} = \frac{\sigma_w^2}{\mathcal{K}^l} \mathbb{E}_\theta \left[\frac{1}{N_l} \sum_{k=1}^{N_l} \phi'(\tilde{h}^l_k)^2 \right] + \mu^2 \,. \tag{2.28}$$

Remarkably, for this specific arrangement, we obtain the following theorem describing the

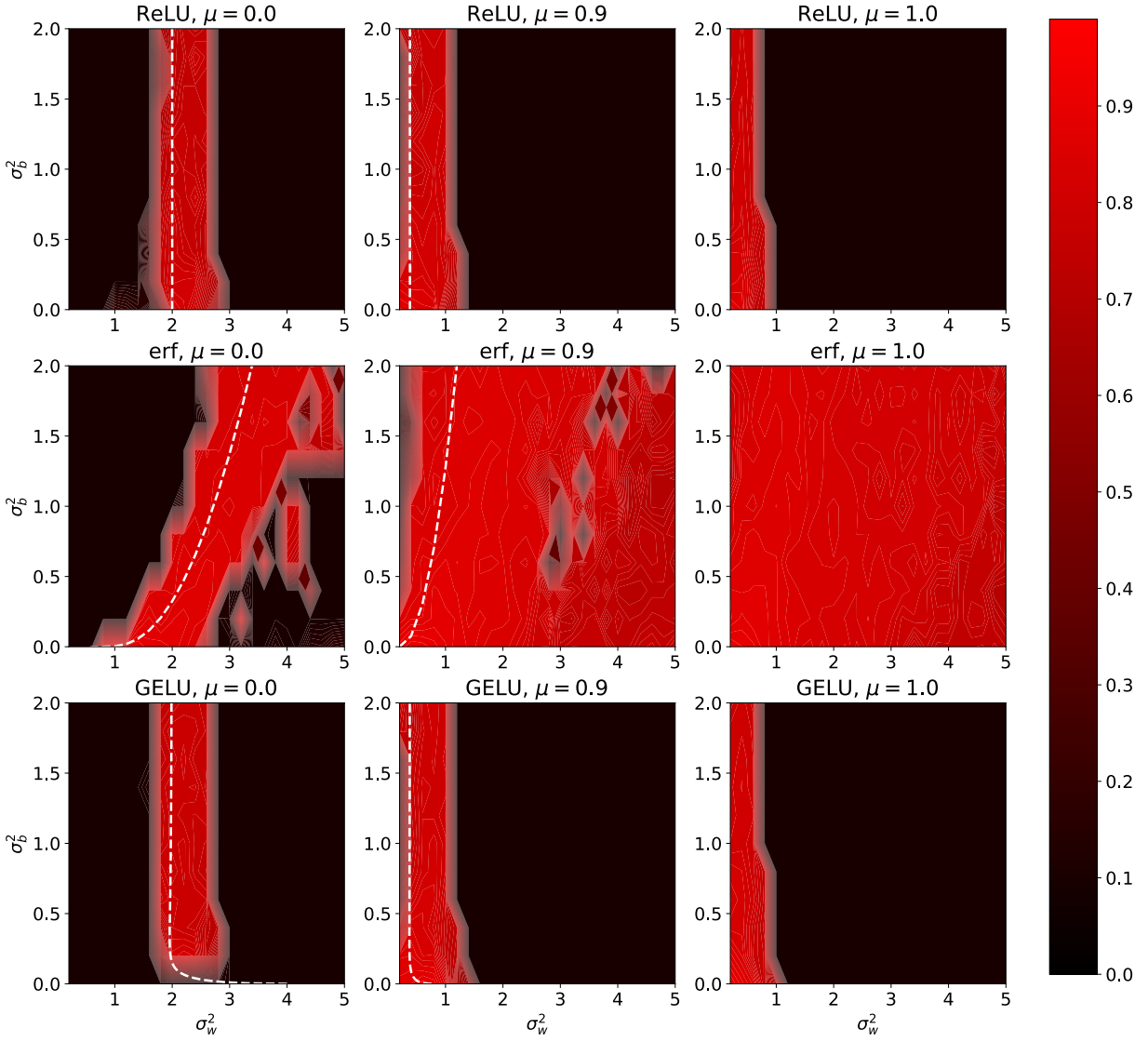


Figure 2.4: Trainability (Training Accuracy) of deep MLP ($N_l = 500$, $L = 50$) on FashonMNIST dataset. The subcritical case of erf activation with $\mu = 1$ has enhanced trainability. The dashed white lines denote the (analytical) critical lines.

correlation length ξ:

Theorem 2.8 (Main result)**.** *Consider a deep MLP network with Lipschitz continuous activation $\phi(\cdot)$, Pre-LayerNorm and residual connections with strength μ (Equation (2.26)). In the sequential limit $(N_1 \to \infty, \ldots, N_l \to \infty, \ldots, N_{L-1} \to \infty)$, for large L, the correlation length*

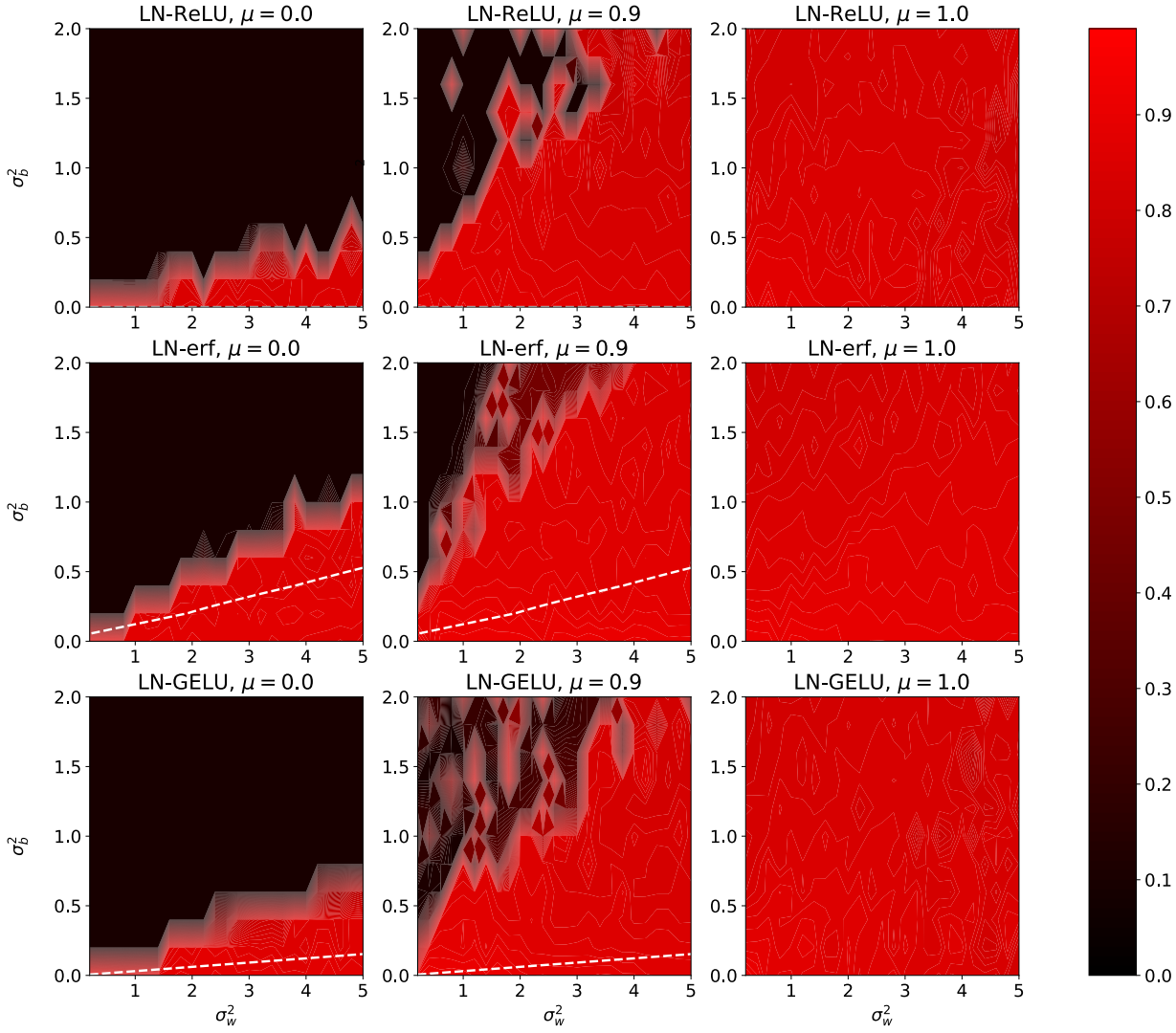


Figure 2.5: Trainability (Training Accuracy) of deep MLP ($N_l = 500$, $L = 50$) on FashonMNIST dataset. The combination of LayerNorm and $\mu = 1$ makes the network everywhere-trainable. The dashed white lines denote the (analytical) critical lines.

$\xi = |\log \chi^\star_{\mathcal{J}}|^{-1}$ *can be written as*

$$\xi = \frac{1}{|\log\left[(1-\mu^2)\frac{A}{B} + \mu^2\right]|}\,, \tag{2.29}$$

where the non-negative constants A and B are given by

$$A = \sigma_w^2 \mathbb{E}_\theta \left[\phi'(\tilde{h}_k^{L-2})^2 \right] , \qquad B = \sigma_w^2 \mathbb{E}_\theta \left[\phi(\tilde{h}_k^{L-2})^2 \right] + \sigma_b^2 ,$$

and $\phi'(\cdot)$ is the derivative of elementwise activation function $\phi(\cdot)$. We omitted the average over the neuron index k in the infinite width limit, and the result does not depend on k.

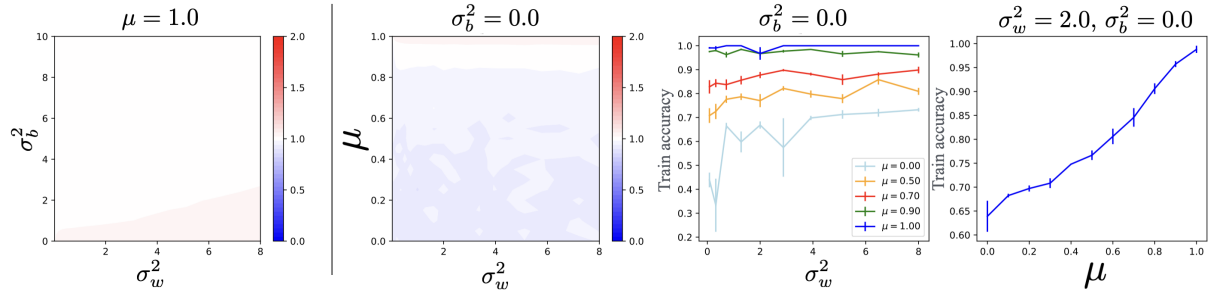
We refer the reader to Appendix A.7 for a detailed proof. Note that from the above theorem, whenever $\mu = 1$, the correlation length ξ diverges. This further implies that different choices of initialization have no effect on the trainability of the network, provided that LayerNorm and residual connections are arranged as stated.

2.6 Modern Architectures

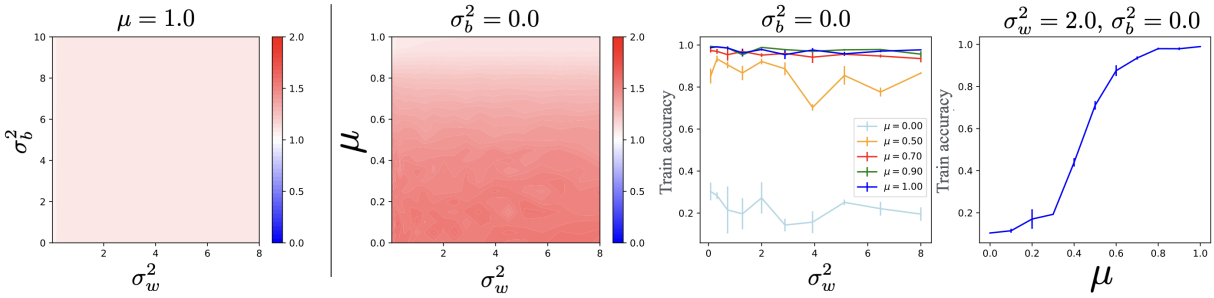
ResNet110 V2. ResNet is one of the most widely used architectures in computer vision[13]. Figure 2.6 shows the results for ResNetV2 [14] (a) with LayerNorm and (b) with Batchnorm. Note that BatchNorm has a similar asymptotic behavior as LayerNorm when $\mu = 1$, but the intuition and the mathematical formulation behind them are drastically different. See Appendix A.2 for a more detailed discussion of BatchNorm.

At $\mu = 1$, $\sigma_w^2 - \sigma_b^2$ phase diagram shows everywhere-criticality, as expected from our theory. For $\sigma_b^2 = 0$, $\sigma_w^2 - \mu$ phase diagram gets closer to criticality as we increase μ, which results in better training at higher μ.

Additionally, networks with larger μ enjoy better trainability due to the suppression of finite width corrections to our theory (decreased effective L/N).



(a) ResNet110 with LayerNorm



(b) ResNet110 with BatchNorm

Figure 2.6: Results for ResNet110 V2 **(a)** LayerNorm and (b) BatchNorm. Within each row from Left to right: **(1)** $\chi^\star_{\mathcal{J}}$ phase diagram with respect to (σ_w^2, σ_b^2) at $\mu = 1$. The network is everywhere-critical for both LayerNorm and BatchNorm. **(2)** $\chi^\star_{\mathcal{J}}$ phase diagram with respect to (σ_w^2, μ). **(3)** Training accuracy w.r.t. σ_w^2, for different values of μ. Trainability improves with with higher μ, as the network gets closer to everywhere-critical phase. For $\mu = 0$ and small σ_w^2 the network is in the ordered phase for LayerNorm and chaotic phase for BatchNorm. **(4)** Training accuracy w.r.t. μ at $(\sigma_w^2, \sigma_b^2) = (2, 0)$. We see a monotonic improvement in trainability as we tune up the value of μ from 0 to 1.

MLP-Mixer. MLP-Mixer architecture is a recent example of MLP approach to computer vision [66]. It can be analyzed using the tools presented above. Fig. 2.7 shows the phase and training diagrams. Further details can be found in the Appendix A.8. Note that the original architecture uses $\mu = 1.0$.

Test accuracies for figs. 2.6 and 2.7 follow similar trends as train accuracies (see Appendix A.8).

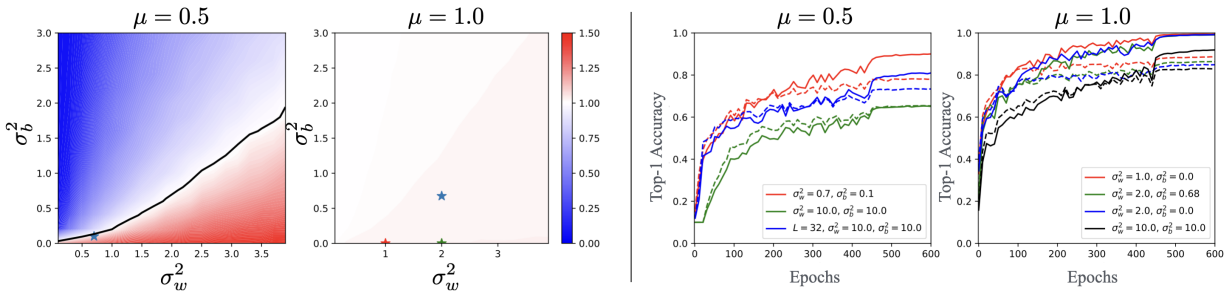


Figure 2.7: Results for MLP-Mixer. Left to right: **(1, 2)** $\mu = 0.5, 1.0$ phase diagrams ($\chi^\star_{\mathcal{J}}$). The network becomes everywhere-critical when $\mu = 1$. The solid black line indicates the empirical phase boundary. Stars denote points we selected to train on CIFAR-10. **(3, 4)** $\mu = 0.5, 1.0$ MLP-Mixer training curves. All, but one, networks are $L = 100$ blocks deep. We see that as we increase μ from 0.5 to 1, the trainability of all networks increases, and they are less sensitive to initialization.

2.7 Automatic Initialization via Jacobian Tuning

As we mentioned earlier, in many cases, it is infeasible to calculate the initialization hyper-parameters for every network manually. To address this challenge and automate the process, we developed the `AutoInit` algorithm, which automatically tunes arbitrary network architectures to criticality by adjusting the APJN in each layer. Unlike the usual way of optimizing parameters, this algorithm does not update the weight or bias parameters directly. Instead, it updates only their scales through scalar multipliers.

First, we define APJN for a general batch size B for this purpose:

Definition 2.9. *A general APJN with multiple examples within a batch is defined as the ffollows*

$$\mathcal{J}^{l_0,l} \equiv \mathbb{E}_\theta \left[\frac{1}{BN_l} \sum_{i=1}^{N_l} \sum_{j=1}^{N_{l_0}} \sum_{\boldsymbol{x},\boldsymbol{x}' \in \{B\}} \left(\frac{\partial h_i^l(\boldsymbol{x}')}{\partial h_j^{l_0}(\boldsymbol{x})} \right)^2 \right] , \tag{2.30}$$

Note that we have generalized Definition 2.4 to incorporate Jacobians between different inputs

$\boldsymbol{x}, \boldsymbol{x}'$.

To efficiently calculate the Jacobian, we use the Hutchinson estimator [67] to estimate APJN, as the sum in Definition 2.9 can be rewritten into a trace form.

Depending on the granularity, we can also impose a stricter requirement for critical initialization, i.e.

$$\chi^l_{\mathcal{J}} = 1 \quad , \forall \quad 1 \leq l \leq L-1 \, . \tag{2.31}$$

Then we can optimize the standard deviation σ_{θ^l} of parameters θ^l via the following Algorithm 1:

Algorithm 1 `AutoInit` (SGD)

Input: Model $\mathcal{M}(\{\sigma^l_\theta;\, a^l_\theta(t) \mid \forall\, 1 \leq l \leq L\,, \forall \theta^l\})$, Loss function $\mathcal{L}(\{\mathcal{J}^{l,l+1}\}_{l=1}^{L})$, T, ϵ, and η.
Set $t = 0$ and $\{a^l_\theta(0) = 1\}$
Evaluate $\mathcal{L}(0)$
while $0 \leq t < T$ and $\mathcal{L}(t) > \epsilon$ **do**
  $a^l_\theta(t+1) = a^l_\theta(t) - \eta \nabla_{a^l_\theta} \mathcal{L}(t)$
  Evaluate $\mathcal{L}(t+1)$
end while
Return $\mathcal{M}(\{\sigma^l_\theta = \sigma^l_\theta a^l_\theta(t);\, 1 \mid \forall\, 1 \leq l \leq L\,, \forall \theta^l\})$

The choice of the objective function $\mathcal{L}$ is crucial. After many exploring many different choices. We use the Jacobian-Kernel Loss (JKL), which can optimize hyperparameters to simultaneously APJN and the NNGP kernel for each layer. It is defined as the the following

$$\mathcal{L}_{\mathcal{J}\mathcal{K}\log} = \frac{1}{2}\sum_{l=1}^{L} \left[\log \chi^l_{\mathcal{J}}\right]^2 + \frac{\lambda}{2}\sum_{l=1}^{L}\left[\log\left(\frac{\mathcal{K}^l}{\mathcal{K}^{l-1}}\right)\right]^2 , \tag{2.32}$$

where the additional hyperparameter λ controls the penalization strength for the ratio between NNGP kernels of consecutive layers. In practice, we usually find set $\lambda = 0$ would work equally

well.

As a proof of concept, we test `AutoInit` algorithm on the VGG [11] and ResMLP [68] architectures.

ResMLP. It is an architecture for image recognition built entirely on MLPs. It offers competitive performance in both image recognition and machine translation tasks. The architecture consists of cross-channel and cross-patch MLP layers, combined with residual connections. In Figure 2.8, we obtain a critical initialization for the ResMLP-S12 model using Algorithm 1.

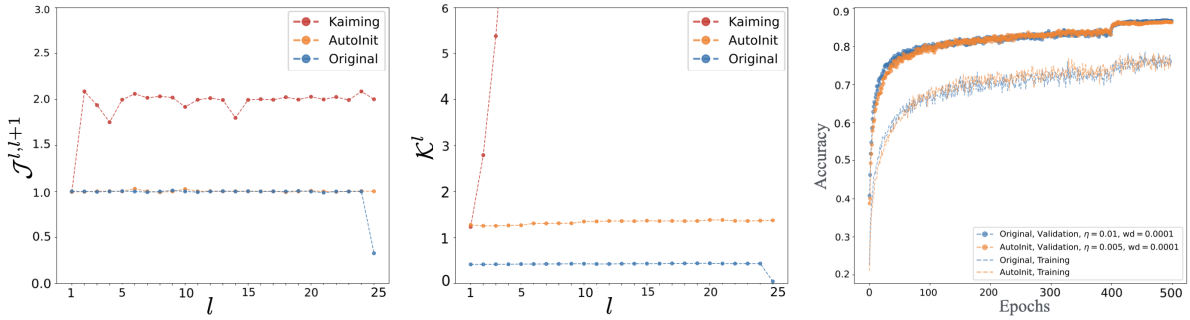


Figure 2.8: **(Left, Middle)**: Comparing $\mathcal{J}^{l,l+1}$ and $\mathcal{K}^l(x,x)$ for ResMLP-S12 for Kaiming, original and `AutoInit` initializations. The network function in the `AutoInit` case is very close to identity at initialization. **(Right)** Training and validation accuracy on CIFAR-10 dataset.

VGG. VGG [11], which was notoriously difficult to train before Kaiming initialization was invented. The BatchNorm variants VGG19_BN further improve the training speed and performance compared to the original version. PyTorch version of VGG [69] is initialized with fan_out Kaiming initialization [12].

We obtain a close to critical initialization using Algorithm 1, where we also introduced the auxiliary parameters a_θ^l for all BatchNorm layers. $\mathcal{J}^{l,l+1}$ is measured by the number of composite (Conv2d-BatchNorm-ReLU) blocks or MaxPool2d layers.

We compared $\mathcal{J}^{l,l+1}$, $\mathcal{K}^{l}(x,x)$ and accuracies on CIFAR-10 datasets between auto-initialized model and the one from PyTorch [70], see Figure 2.9.

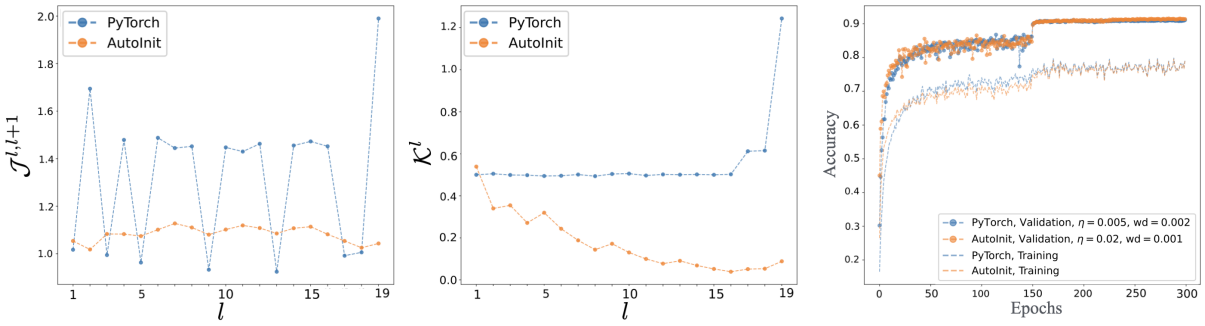


Figure 2.9: **(Left, Middle)**: Comparing $\mathcal{J}^{l,l+1}$ and $\mathcal{K}^{l}(x,x)$ between PyTorch version VGG19_BN and `AutoInit` version, we ensure $\mathcal{J}^{l,l+1} = 1$ with a high priority ($\lambda = 0.05$). **(Right)** Training and validation accuracy on CIFAR-10 dataset.

2.8 Discussion

We have introduced partial Jacobians and their averaged norms as tools to analyze the propagation of gradients through deep neural networks at initialization. The scaling behavior of APJN gives rise to an emergent length scale that characterizes the maximum trainable depth of such networks. Using APJN evaluated close to the output, $\mathcal{J}^{L-2,L-1} \approx \chi^{\star}_{\mathcal{J}}$, we have introduced a very cheap and simple empirical test for criticality. We have also shown that criticality formulated in terms of partial Jacobians is equivalent to criticality studied previously in literature [27, 56, 59]. Additionally, APJN can be utilized to quantify criticality in inhomogeneous (*i.e.* no periodic stacking of blocks) networks [2].

We have investigated homogeneous architectures that include fully-connected layers, normalization layers and residual connections. In the limit of infinite width, we showed that (i) in the presence of LayerNorm, the critical point generally becomes a critical line, making the ini-

tialization problem much easier, (ii) LayerNorm applied to preactivations enhances correlation length leading to improved trainability, (iii) combination of $\mu = 1$ residual connections and erf activation function enhances correlation length driving the network to a *subcritical* phase with APJN growing according to a stretched exponential law, (iv) combination of residual connections and LayerNorm drastically increases correlation length leading to improved trainability, (v) when $\mu = 1$ and LayerNorm is applied to preactivations *the network is critical on the entire* σ_b–σ_w *plane.*

We have considered examples of modern architectures: ResNet and MLP-Mixer. We showed that at $\mu = 1$, they are critical everywhere, due to the interaction between LayerNorm and residual connections. We have studied ResNet at different μ's, showing that higher μ enjoys better trainability, due to improved initialization and suppressed effective L/N corrections.

We have empirically demonstrated that a deep (100 blocks) MLP-Mixer with $\mu = 1$ trains well for various initializations. In comparison, for $\mu = 0.5$, it only trains well close to the critical line.

Our work shows that an architecture can be designed to have a large correlation length, leading to a guaranteed trainability with SGD for any initialization scheme.

Building on all these insights, we developed the `AutoInit` algorithm, which can potentially assist in initializing any neural network architecture that may be designed in the future.

2.8.1 Limitations and Outlook

So far, our method can be applied to most commonly used architectures, such as Transformers (see Appendix A.4 for further discussion). However, our framework currently operates

at a coarse level of granularity, meaning that the fine-grained details of each layer cannot be captured accurately.

More importantly, the training dynamics depend strongly on the choice of initialization. As an extreme example, in the pre-LayerNorm architecture with residual connections described earlier, there exists a trivial solution where all parameters are initialized to zero. Although this initialization perfectly satisfies the criticality conditions, it leads to no learning at all. Clearly, such an initialization is undesirable. One possible remedy is to initialize certain layers within each block with smaller scales, ensuring both good initialization and reasonable training dynamics. However, it remains unclear what the optimal strategy is for implementing such schemes, particularly from a theoretical standpoint.

This contradiction highlights a deeper issue in architectural design. As the community increasingly pursues stability through Pre-LayerNorm architectures, we inadvertently expose ourselves to trivial minima in which the network effectively reduces to the identity function. Even when the network is not entirely trivial, there is preliminary evidence that it may behave like a much shallower model. Thus, the "safety" we strive for can ultimately become a limitation. From this perspective, our goal is to use the proposed tools to initialize more nonlinear networks that avoid this bias – potentially leading to fundamentally better architectural designs.

Chapter 3: Emergent Low-dimensional Fixed-Point Structures in Sharpness Dynamics

In the previous chapter, we discussed how to initialize deep and wide neural networks from the perspective of signal propagation. Recall that we considered the limit in which the network width tends to infinity, which under an appropriate choice of variance, yields $O_N(1)$ feature magnitudes, ensuring that width scaling does not pose a problem. However, this alone does not guarantee reasonable training dynamics at a large width, as we will see shortly. It is therefore essential to examine whether the chosen width (and depth, though not covered in this dissertation) scaling corresponds to a good initialization from an optimization perspective.

Another important topic we address in this chapter is *parameterization*, which can be viewed as a kind of gauge freedom in the forward pass. Specifically, the following forward propagation rule is equivalent to Equation (2.3):

$$h_i^l(\boldsymbol{x}) = \sum_{j=1}^{N_{l-1}} \frac{1}{\sqrt{N_{l-1}^{1-p}}} w_{ij}^l \, \phi(h_j^{l-1}(\boldsymbol{x})) + b_i^l, \qquad 1 \le l \le L\,, \tag{3.1}$$

where $w_{ij}^l \sim \mathcal{N}(0, \sigma_w^2/N_{l-1}^p)$, for any value of p. Although the forward passes are equivalent for different p, taking derivatives with respect to w_{ij}^l during gradient descent (GD) leads to substantially different training dynamics and loss landscapes. This motivates a deeper investigation into

how parameterization and initialization jointly influence optimization behavior.

Interestingly, the aforementioned topics are closely related to phenomena recently observed in the non-convex optimization community. In particular, the top eigenvalue of the loss Hessian, $\lambda^H = \nabla^2_{\boldsymbol{\theta}}\mathcal{L}$, often referred to as the *sharpness*, has been shown to exhibit rich and dynamic behavior during training. As we will see shortly, these behaviors are strongly influenced by optimization hyperparameters, architectural choices, initialization, and parameterization.

3.1 Introduction to Sharpness Dynamics

Before proceeding to the analysis, we first provide additional insights into sharpness dynamics in non-convex optimization setting. Recent studies have observed that during the later stages of training, gradient descent (GD) typically exhibits a phenomenon known as *progressive sharpening*, where the sharpness gradually increases over time until it reaches approximately $2/\eta$, with η denoting the learning rate. Once the sharpness reaches this threshold, it ceases to grow and instead oscillates around $2/\eta$, a late-time training phenomenon referred to as the *edge of stability (EoS)* [71]. In contrast, during the early stages of training, a decrease in sharpness is often observed, referred to as *sharpness reduction* [72], before the dynamics settle into a temporary plateau.

For large enough learning rates, training temporarily destabilizes early on, and the network "catapults" out of its local basin, leading to a temporary sudden increase in the loss in the first few steps, before eventually settling down in a flatter region of the loss landscape characterized by lower sharpness [73]. Similar to the loss, sharpness may also spike within the first few steps of training and quickly decrease (sharpness catapult). A rich phase diagram as a function of network

depth, width and learning rate summarizes the early training dynamics [72].

The discovery of these intriguing sharpness phenomena has attracted significant attention, inspiring the study of various toy models that exhibit similar behavior. However, the precise conditions under which these phenomena arise, as well as their underlying mechanisms, remain elusive. In the remainder of this chapter, we analyze a simple toy model, a two-layer linear network trained on a single example, referred to as the UV model. In this setting, we uncover emergent low-dimensional fixed-point structures and a period-doubling route to chaos in the $\mathcal{L}$–sharpness phase space. We demonstrate that all of the phenomena described above can be reproduced in the UV model with appropriate choices of learning rate, initialization, parameterization, and training example. Through this exploration, we offer new insights into the mechanisms underlying sharpness dynamics and present guidance on choosing parameterization and initializations that we validate in realistic architectures using both real and synthetic datasets.

3.2 Notations and Preliminaries

This section describes the fundamental concepts and notations that form the basis of our analysis. Note we have switched some notations for better readability.

Dynamical Systems and Fixed Points: Consider a discrete dynamical system described by $\boldsymbol{\theta}_{t+1} = M(\boldsymbol{\theta}_t)$. A fixed point $\boldsymbol{\theta}^*$ of the dynamics satisfies $M(\boldsymbol{\theta}^*) = \boldsymbol{\theta}^*$. The linear stability of a fixed point $\boldsymbol{\theta}^*$ is determined by analyzing the eigenvalues $\{\lambda_i^{J^*}\}$ of the Jacobian $J_M(\boldsymbol{\theta}^*) \coloneqq \nabla_\theta M(\theta)\,|_{\theta=\theta^*}$. An eigendirection $u_i^{J^*}$ of a fixed point θ^* is stable if $|\lambda_i^{J^*}| < 1$ and unstable if $|\lambda_i^{J^*}| > 1$ [74]. The dynamics is captured by the vector field of updates $G(\theta) \coloneqq M(\theta) - \theta$. The corresponding unit vector is denoted $\hat{G}(\theta) \coloneqq {}^{G(\boldsymbol{\theta})}/_{\|G(\boldsymbol{\theta})\|}$. Nullclines refer to curves where one of

the variables, θ_i, remains invariant, i.e., $\theta_{i;t} = M_i(\boldsymbol{\theta_t})$.

Parameterizations in Neural Networks: Sharpness phenomena in neural networks are intrinsically tied to network parameterization. Standard Parameterization (SP) [75] and Neural Tangent Parameterization (NTP) [76] are two commonly used parameterizations, which converge to kernel methods at infinite width. [77] proposed Maximal update Parameterization (μP), which allows for feature learning at infinite width. For implementation details, see Appendix B.2.3.

UV Model: The UV model refers to a 2-layer linear network $f : \mathbb{R}^d \to \mathbb{R}$ trained on a single example. We parameterize f as $f(\boldsymbol{x}; \theta) = \frac{1}{\sqrt{n^{1-p}}} \boldsymbol{v}^T U \boldsymbol{x}$, where $\boldsymbol{x} \in \mathbb{R}^d$ is the input, n is the network width, and $\boldsymbol{v} \in \mathbb{R}^n, U \in \mathbb{R}^{n \times d}$ are trainable parameters, with each component drawn i.i.d. at initialization from a normal distribution $\mathcal{N}(0, \sigma_w^2/n^p)$. Here, $p \in [0, 1]$ is a parameter that interpolates between NTP and μP, and $n_{\text{eff}} := n^{1-p}$ is referred to as the effective width. We consider the network trained on a single training example $(\boldsymbol{x}, y)$ using MSE loss $\ell\left(f(\boldsymbol{x}; \theta), y\right) = \frac{1}{2}\left(f(\boldsymbol{x}; \theta) - y\right)^2$.

3.3 Review of The Four Regimes of Training

Typical training trajectories of neural networks can be categorized into four training regimes [72], as shown in Figure 3.1(a, b):

(T1) *Early time transient:* This corresponds to the first few steps of training. At small learning rates ($\eta < \eta_{\text{loss}}$), loss and sharpness decrease monotonically. At larger learning rates ($\eta > \eta_{\text{loss}}$), training catapults out of the initial basin, temporarily increasing the loss, and finally converges to a flatter region [73]. By the end of this regime, sharpness has decreased from

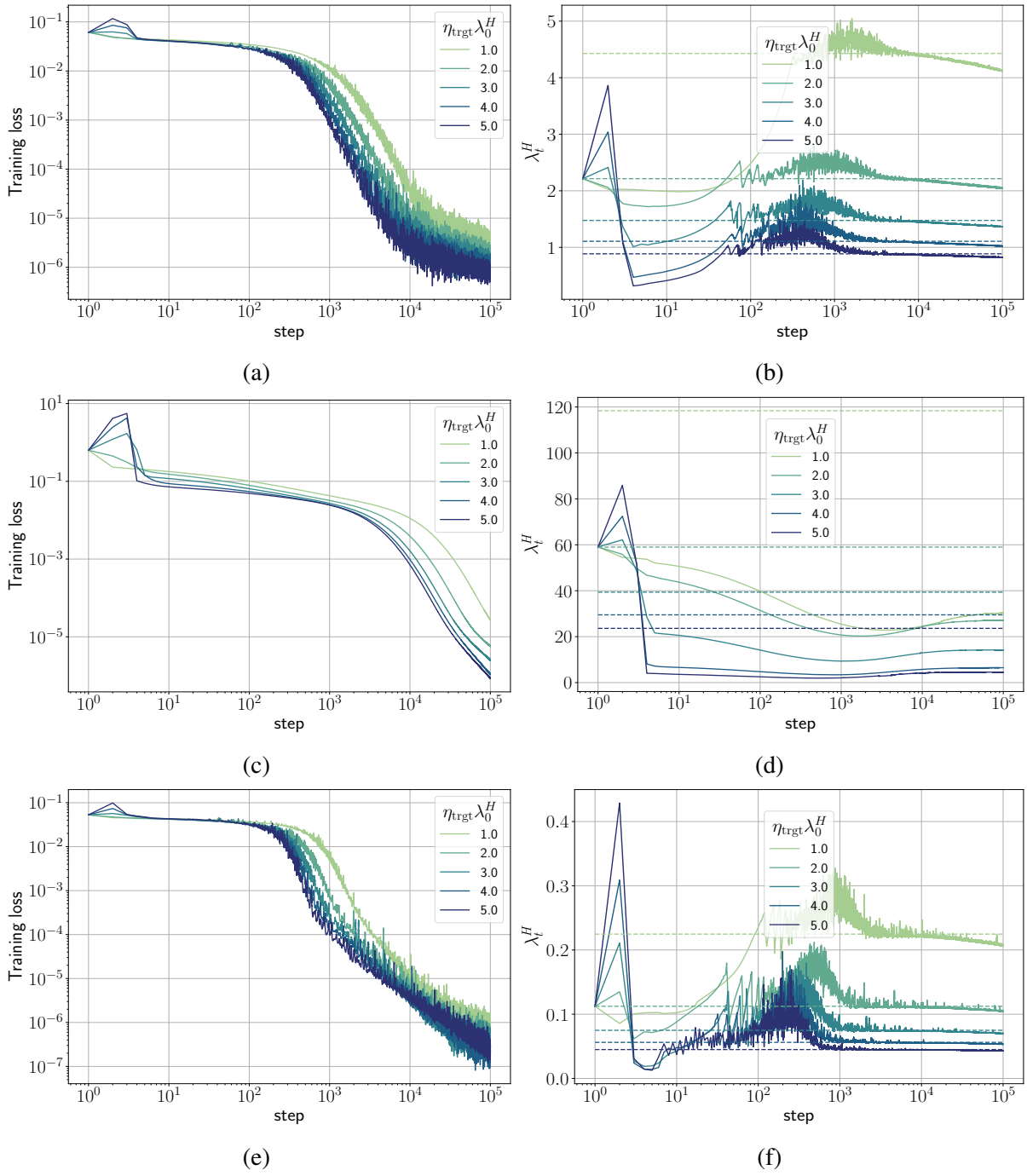


Figure 3.1: Training loss and sharpness trajectories of ReLU FCNs trained on a 5k subset of CIFAR-10 examples using MSE loss and GD: (a, b) SP with $\sigma_w^2 = 0.5$, (c, d) SP with $\sigma_w^2 = 2.0$, (e, f) μP with $\sigma_w^2 = 2.0$. The dashed lines in the sharpness figures show the $2/\eta$ threshold.

initialization for all learning rates, and more substantially at larger learning rates.

(T2) *Intermediate saturation:* Following the initial transient regime, sharpness approxi-

mately plateaus before gradually increasing.

(T3) *Progressive sharpening:* In this regime, sharpness continues to increase until it reaches $\lambda^H \approx 2/\eta$ [71, 78]. At large effective widths or small learning rates, training may conclude before reaching this threshold.

(T4) *Late-time dynamics (EoS):* After progressive sharpening, for MSE loss, sharpness oscillates around $2/\eta$. For cross-entropy loss, the sharpness oscillates when reaching approximately $2/\eta$, while decreasing over longer time scales [71].

In this work, we show that the sharpness dynamics heavily depends on the initialization and parameterization of the network, and not every training trajectory shows all four regimes. For instance, Figure 3.1(c, d) shows that FCNs in SP with large initialization (or large effective width) do not exhibit EoS, even when loss decreases to a value below 10^{-5}. Following the early transient regime, sharpness monotonically decreases, with only a nominal increase towards late training. In contrast, Figure 3.1(e, f) shows that FCNs in μP (or small effective width) do not experience an initial sharpness reduction at small learning rates ($\eta < \eta_{\text{loss}}$). Rather, sharpness continues to increase until it reaches $2/\eta$ and then oscillates around it. At large learning rates ($\eta > \eta_{\text{sharp}}$), sharpness catapults and eventually settles into the same trend as above.

These different training regimes are generically observed for more complex architectures and datasets as we show in Appendix B.4.3, including CNNs and ResNets, trained on CIFAR-10 and Transformers trained on Wikitext-2.

In Appendix B.4.1, we show that these trends remain consistent when NTP is used instead of SP. Given this similarity in the training dynamics between SP and NTP, we use NTP for theoretical analysis for clarity and SP in realistic experiments for implementation convenience.

Figure 3.2 (and Figure B.2 in Appendix B.3.5) demonstrates that the UV model displays

all four training regimes. It also captures the cases where sharpness reduction or EoS is not observed. Therefore, the simplified UV model can serve as an effective model for understanding these universal behaviors in the sharpness dynamics. In the subsequent section, we perform fixed point analysis of the UV model and probe the origin of these complex phenomena in later sections.

3.4 Fixed Point Analysis of the UV model

Under GD, the parameters of the UV model are updated as $U_{t+1} = U_t - \eta \frac{\Delta f_t \boldsymbol{v}_t \boldsymbol{x}^T}{\sqrt{n_{\text{eff}}}}$, $\boldsymbol{v}_{t+1} = \boldsymbol{v}_t - \eta \frac{\Delta f_t U_t \boldsymbol{x}}{\sqrt{n_{\text{eff}}}}$, where η is the learning rate and $\Delta f_t \coloneqq f(\boldsymbol{x}; \theta_t) - y$ is the residual at training step t. In function space, the dynamics can be completely described using the residual Δf_t and trace of the loss Hessian $\lambda \coloneqq \text{Tr}\, H = \frac{1}{n_{\text{eff}}} \left(\boldsymbol{x}^T U^T U \boldsymbol{x} + \boldsymbol{v}^T \boldsymbol{v}\, \boldsymbol{x}^T \boldsymbol{x} \right)$, which is also the scalar neural tangent kernel in this case. The function space dynamics of the UV model can be fully described using two coupled non-linear equations (for derivation, see Appendix B.3.1):

$$\Delta f_{t+1} = \Delta f_t \left(1 - \eta \lambda_t + \frac{\eta^2 \|\boldsymbol{x}\|^2}{n_{\text{eff}}} \Delta f_t (\Delta f_t + y) \right), \tag{3.2}$$

$$\lambda_{t+1} = \lambda_t + \frac{\eta \, \|\boldsymbol{x}\|^2}{n_{\text{eff}}} \Delta f_t^2 \left(\eta \lambda_t - 4 \frac{(\Delta f_t + y)}{\Delta f_t} \right), \tag{3.3}$$

with effectively three parameters: η, $\|\boldsymbol{x}\|/\sqrt{n_{\text{eff}}}$ and y. While similar equations have been considered in previous works [73, 79, 80], the generalization to generic parameterizations is novel and would be crucial in observing different sharpness phenomena such as EoS. The $y = 0$ case has been analyzed in prior works [72, 73] to understand catapult dynamics during early

training. However, neither of these works recognized the importance of a finite y value, nor did they perform a complete analysis across the entire training trajectory, thereby missing the rich phenomena that we will soon describe below. Here, λ can only decrease with time, as can be seen from Equation (3.3) with $\eta < \eta_{\text{max}} = 4/\lambda_0$ (training diverges if $\eta > \eta_{\text{max}}$). As a result, the model does not exhibit progressive sharpening and EoS. Below we focus on the case $y > 0$, which allows for λ to increase in time and consequently, much richer dynamics.

Equations (3.2) and (3.3) have four distinct fixed points/lines (referred to as I-IV) as detailed in Table B.1 of Appendix B.3.3. The fixed line I defines a zero-loss line, meaning $\ell = 0$ for all points in I; the points in I are stable for $\eta\lambda < 2$ and unstable otherwise. Fixed point II at $(-y, 0)$ corresponds to the origin in parameter space $(U, \mathbf{v} = 0)$ and it is a saddle point of the dynamics for convergent learning rates η. Both I and II are also fixed points of the GD optimization, i.e., critical points of the loss. The loss Hessian at I is positive definite, while fixed point II is a saddle point in the loss landscape. The remaining two fixed points III and IV are unstable and exist only in function space, representing non-trivial parameter space dynamics that leave the function space dynamics invariant.

Figure 3.2 shows the fixed points and the vector field $\hat{G}(\Delta f, \lambda)$ determined by Equations (3.2) and (3.3), which illustrates the direction of the updates at each point. Note that the stability of the fixed line (I) does not follow from $\hat{G}$ alone, as the magnitude G is required to determine stability. Figure 3.2 also shows training trajectories for various parameter values. Using λ as a proxy for sharpness, we see there are regions where λ increases (colored yellow) and decreases (colored green) along the flow, which we refer to as progressive sharpening and sharpness reduction, respectively. It follows from Equation (3.3) that the condition $\eta\lambda\Delta f = 4(\Delta f + y)$ separates these regions. Importantly, the parameters $\eta, {\|\boldsymbol{x}\|}/{\sqrt{n_{\text{eff}}}}$ and y influence the position of

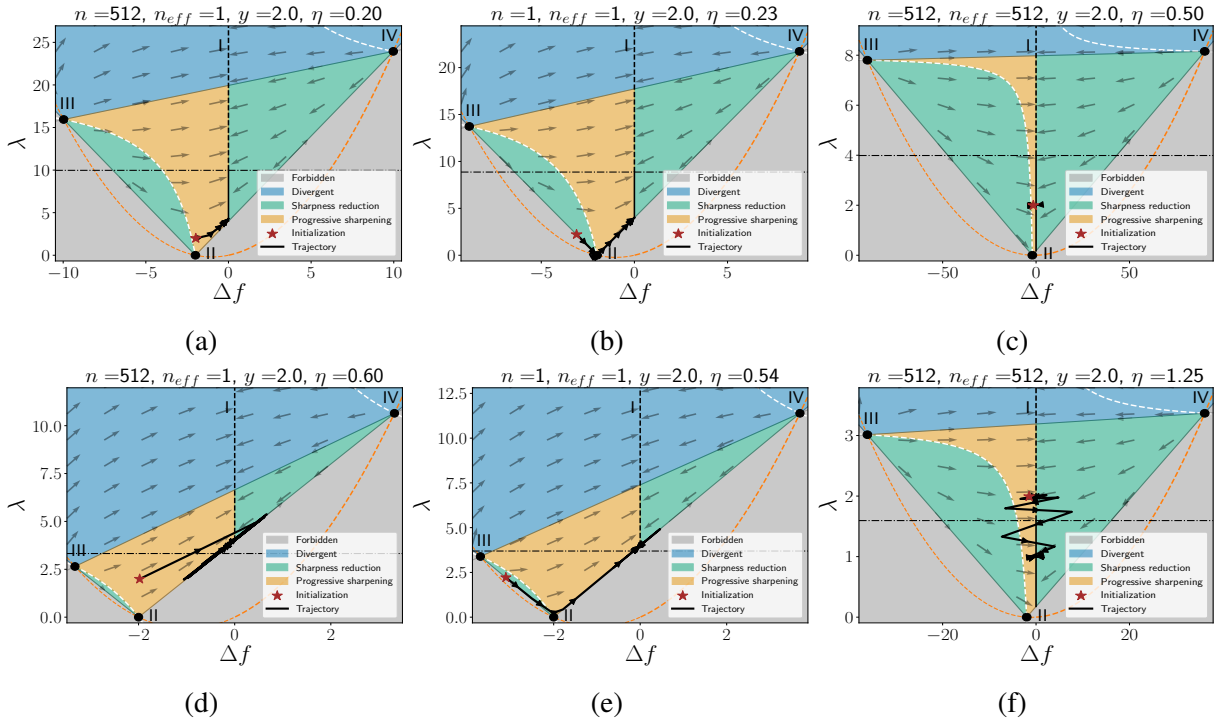


Figure 3.2: Training trajectories of the UV model with $\|\boldsymbol{x}\| = 1$ and $y = 2$ in the $(\Delta f, \lambda)$ plane for different values of n, n_{eff} and η. The columns show initializations with different n and n_{eff}, while the rows represent increasing learning rates for fixed initializations. The horizontal dash-dot line $\eta\lambda = 2$ separates the stable (solid black vertical line) and unstable (dashed black vertical line) fixed points along the zero loss fixed line I. Forbidden regions, $2\|\boldsymbol{x}\||\Delta f + y|/\sqrt{n_{\text{eff}}} > \lambda$, (see Appendix B.3.2) are shaded gray. The nullclines $\Delta f_{t+1} = \Delta f_t$ and $\lambda_{t+1} = \lambda_t$ are shown as orange and white dashed curves, respectively. Sharpness reduction, progressive sharpening, and divergent regions are colored green, yellow, and blue. The gray arrows indicate the local vector field $\hat{G}(\Delta f, \lambda)$, which is the direction of the updates. The training trajectories are depicted as black lines with arrows, with the star marking the initialization. In all cases, $\eta_c = \sqrt{n_{\text{eff}}}/2$ (introduced in Section 3.5.2).

the fixed points. This, in turn, affects the extent of different regions and the vector field $\hat{G}$, as illustrated in Figure 3.2. In particular, on decreasing effective width n_{eff}, or increasing learning rate η, fixed points III and IV move inward (see fixed point expressions in Table B.1), which relatively enlarges the progressive sharpening region while shrinking the overall convergent region. Overall, these illustrations demonstrate how the local stability and relative position of the fixed points collectively impact the dynamics. In the subsequent section, we discuss the dynamics in detail.

3.5 Understanding Sharpness Dynamics in the UV model

In this section, we describe the origin of different robust phenomena in the dynamics of sharpness using the fixed point and linear stability analysis from the previous section. This explains the four training regimes observed in the UV model. We will discuss the influence of effective width and initializations, shedding light on the differences between NTP and μP. For simplicity, we assume $\|\boldsymbol{x}\| = 1$, while allowing n_{eff} to vary continuously. Note that we use $\lambda := \operatorname{Tr} H$ from the previous section as a proxy for sharpness; we have verified that the top eigenvalue of the Hessian of the loss also follows λ (see Appendix B.3.5), although it is more difficult to analyze analytically.

3.5.1 Understanding Sharpness Trends Throughout Training

Figure 3.2 shows that the training dynamics can exhibit different behavior depending on the initial region. Below we summarize these based on empirical observations.

(R1) *Progressive sharpening region*: As shown in Figure 3.2(a, d), initialization in this region experiences an upward push due to the flow originating from fixed point II, resulting in a steady increase in λ. Depending on η relative to a critical learning rate η_c (introduced in Section 3.5.2) different late-time dynamics arises. For $\eta < \eta_c$, training converges to stable fixed points on the zero-loss line (I), as shown in Figure 3.2(a). When $\eta > \eta_c$, all points along the zero-loss line (I) become unstable, as shown in Figure 3.2(d). In this case, the network eventually converges to a line segment joining fixed points II and IV (the EoS manifold), where it continues to oscillate indefinitely between these fixed points, leading to the EoS phenomena. This will be analyzed in more depth in the subsequent section.

(R2) *Sharpness reduction region between fixed points II and III*: Figure 3.2(b, e) show that initializations in this region undergo a decrease in λ as the flow is towards saddle point II. On approaching this saddle point, the dynamics slows down, resulting in the intermediate saturation regime. Eventually, training moves away from this saddle and enters the progressive sharpening region. From here on, the dynamics becomes akin to the case (R1).

(R3) *Sharpness reduction region b/w fixed line I and point IV*: Initializations in this region either converge to the nearby zero-loss solution for ($\eta < \eta_c$) or enter the progressive sharpening region for ($\eta > \eta_c$). In the latter case, the dynamics resembles those of case (R1).

So far, we have described the resultant dynamics when training is initialized in different regimes. Below, we describe the conditions that typically exhibit these regimes.

Neural Tangent Parameterization: In NTP, Δf and λ follow normal distributions: $\Delta f_0 \sim \mathcal{N}(-y, \sigma_w^4)$ and $\lambda_0 \sim \mathcal{N}(2\sigma_w^2, {}^{4\sigma_w^4}/_n)$. Hence, the model can be initialized in any of the three regions described above. Moreover, fixed points III and IV move outward with increasing width, affecting the local vector field $\hat{G}(\Delta f, \lambda)$. At large widths $n \gg 1$, $\hat{G}(\Delta f_0, \lambda_0)$ at initialization points along $\begin{bmatrix} 1 & 0 \end{bmatrix}^T$ towards the zero-loss line. For small learning rates ($\eta < {}^2/_{\lambda_0}$), training exponentially converges to the nearest zero-loss solution (see Figure 3.2(c)). Regardless of the initialization region, the change in λ is minimal, receiving $\mathcal{O}({}^1/_n)$ updates as per Equation (3.3). For large learning rates ($\eta > {}^2/_{\lambda_0}$), the nearby zero-loss solution becomes unstable. Consequently, training catapults to a region with smaller λ, while bouncing between fixed points III and IV. This is the catapult effect studied in [73] and Figure 3.2(f) demonstrates such a trajectory. By comparison, at small widths, the dynamics follows cases (R1-R3) discussed above.

Maximal Update (μP) Parameterizations: In contrast to NTP, the position of fixed points III-IV do not change with width n, and Δf_0 follows the distribution: $\Delta f_0 \sim \mathcal{N}(-y, \sigma_w^4/n)$, while λ_0 distribution remains unchanged. Consequently, at large widths, the model is initialized at $(-y, 2\sigma_w^2)$, right above fixed point II in the progressive sharpening region (R1), satisfying the condition $\eta\lambda_0\Delta f_0 < 4(\Delta f_0 + y)$. Figure 3.2(a, d) shows such a trajectory. At small widths, fluctuations increase, making it plausible for μP networks to start in the sharpness reduction regions. In this case, the dynamics follow case (R2) or (R3).

3.5.2 Understanding Edge of Stability

This section analyzes the EoS behavior in the UV model, particularly from the fixed point perspective. As discussed in the previous section, the EoS behavior in the UV model arises when all fixed points along the zero-loss line (I) become unstable wrt the learning rate. Yet, the gradient updates (shown as gray arrows in Figure 3.2) continue to point towards the zero loss line. As a result, training is trapped in this region, converging to the line segment that joins fixed points II and IV —referred to as the EoS manifold —where it oscillates indefinitely.

EoS Manifold is an Attractor: By examining the two-step dyanmics akin to [80, 81], we show in Appendix B.3.7 that training converges to the EoS manifold above a critical learning rate η_c. For $\eta < \eta_c$, training converges to the stable fixed points on the zero-loss line. By comparison, for $\eta > \eta_c$, all points along the zero-loss line become unstable and the EoS manifold becomes a dynamical attractor. The critical η_c for which all points on the zero-loss line become unstable thus gives a necessary condition for EoS:

Result 1. A necessary condition for the UV model to exhibit EoS is $\eta > \eta_c = \sqrt{n_{\text{eff}}}/\|\boldsymbol{x}\|y$ (see

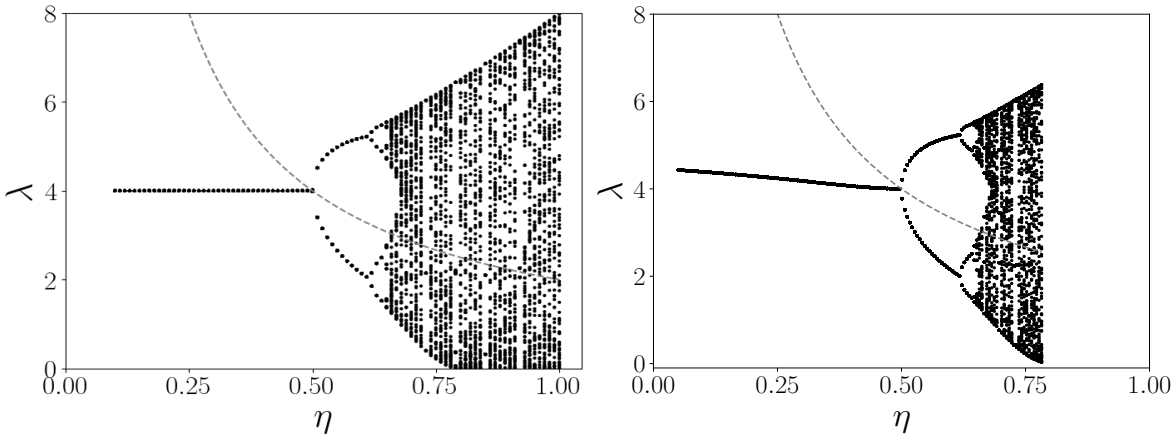


Figure 3.3: (a) Bifurcation diagram depicting limiting values of λ obtained by simulating Equation (3.4). (b) Bifurcation diagram of the UV model. In both figures, $\|\boldsymbol{x}\| = 1, y = 2$, $n_{\text{eff}} = 1$ and $\eta_c = 0.5$.

Appendix B.3.8 for details). It is useful to scale the learning rate as $\eta = c/\lambda_0$, in which case this condition becomes $\lambda_0 < {}^{c\|\boldsymbol{x}\|y}/_{\sqrt{n_{\text{eff}}}}$.

For learning rates $\eta > 2/\lambda_0$, training can catapult to regions with $\lambda_T < \lambda_0$. In such cases, the condition $\lambda_T < {}^{c\|\boldsymbol{x}\|y}/_{\sqrt{n_{\text{eff}}}}$ also applies.

Dynamics on the EoS Manifold and Route to Chaos: The dynamics on the EoS manifold satisfies $\lambda = {}^{2\|\boldsymbol{x}\|(\Delta f+y)}/_{\sqrt{n_{\text{eff}}}}$, coupling Δf and λ together. This yields the map $\Delta f_{t+1} = M_f(\Delta f_t)$ describing the dynamics on the EoS manifold, with M_f defined as

$$M_f(\Delta f_t) := \Delta f_t + \frac{\eta \Delta f_t}{\eta_c y}\left(\frac{\eta \Delta f_t}{\eta_c y} - 2\right)(\Delta f_t + y). \tag{3.4}$$

Figure 3.3(a) shows the limiting values of λ (i.e. the values of λ that the network jumps between at late times) as a function of learning rate, obtained by simulating Equation (3.4). We refer to this as the bifurcation diagram. As mentioned before, for $\eta > \eta_c$, the zero-loss solution

becomes unstable with λ oscillating around $2/\eta$ instead of converging. These fluctuations exhibit a fractal structure, as the system undergoes a series of period-doubling transitions with an increasing learning rate. This is the well-known *period-doubling route to chaos* [74]. Figure 3.3(b) shows the bifurcation diagram of the UV model for $y = 2$. The bifurcation diagram diagram extends up to $\eta \approx 0.8$ before diverging at higher learning rates. This leads us to the following corollary of Result 1.

Corollary 3.1. *Let $\eta_{\max}$ be the maximum trainable learning rate for a given initialization. The bifurcation diagram is observed up to $\eta < \eta_{\max}$. If $\eta_{\max} < \eta_c$, the UV model does not exhibit EoS.*

These results suggest that models with small λ_0 and n_{eff} are more prone to show EoS behavior. As a result, μP networks or those with small initial weight variance are more likely to exhibit EoS. On the other hand, large-width NTP networks may not show EoS behavior at all. In the next section, we will validate this prediction in real-world scenarios.

Connections to sub-quadratic loss : [82] demonstrated that GD on sub-quadratic loss with large learning rates inherently results in EoS behavior. Here, we show that the loss on the EoS manifold of the UV model is sub-quadratic near its minimum. As noted above, the dynamics on the EoS manifold satisfies $\lambda = 2\|\boldsymbol{x}\|(\Delta f + y)/\sqrt{n_{\text{eff}}}$. The loss on the EoS manifold is then given by $\mathcal{L}(\theta) = \frac{1}{2}\Delta f^2 = \frac{y^2}{2}(\frac{\eta_c \lambda}{2} - 1)^2$ (see Appendix B.3.9 for derivation), where θ denotes the parameters. Since $\lambda \sim \mathcal{O}(\|\theta\|^2)$, the loss is of the form $\mathcal{L}(\theta) \approx \frac{1}{2}(a\|\theta\|^2 - b)^2$ and is sub-quadratic near its minimum. The GD dynamics near the minimum is given by a cubic map, which is known to show the period-doubling route to chaos [83]. [84] showed a similar route to chaos by considering a two-layer network with quadratic activation, with the last layer vector

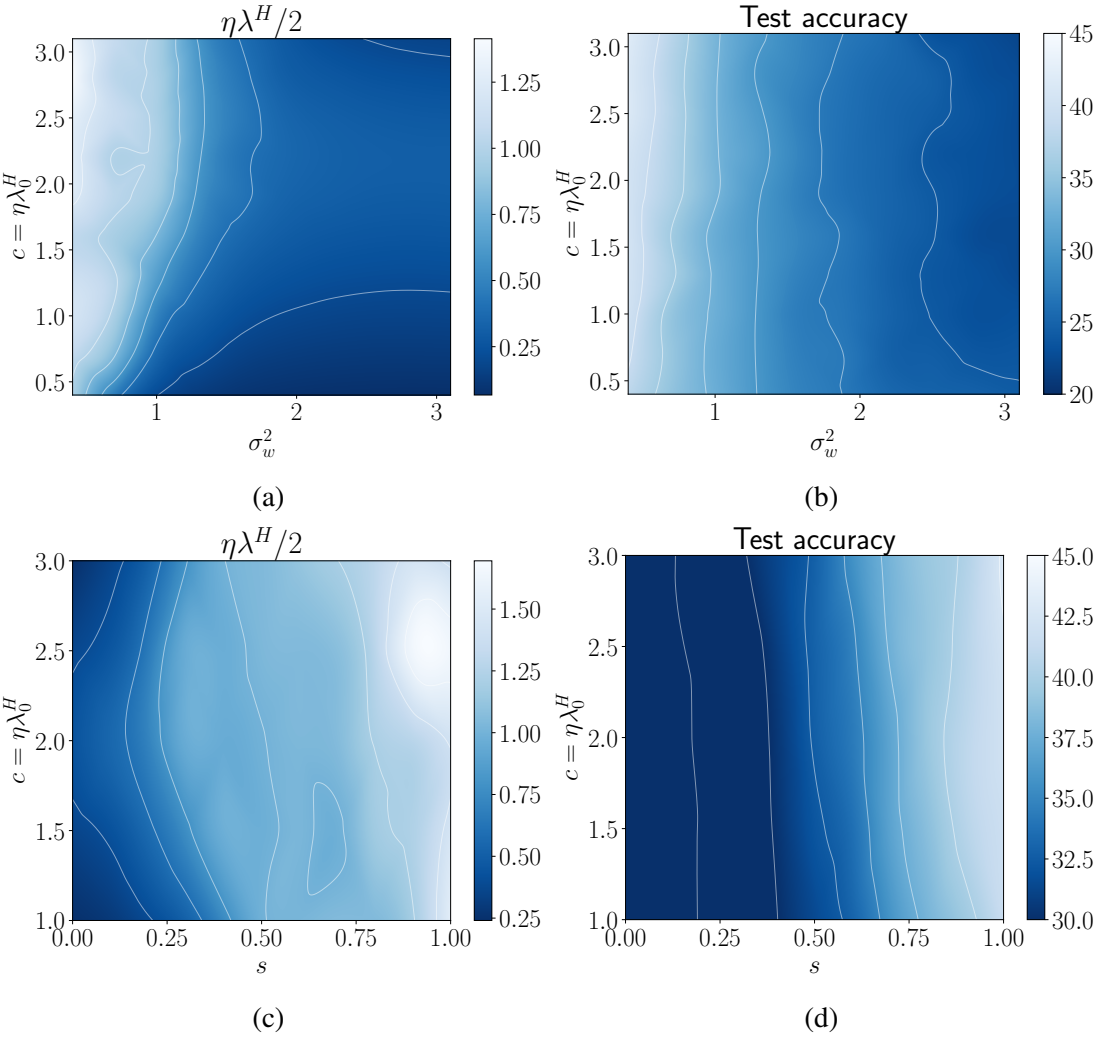


Figure 3.4: (a, b) Heatmap of $\eta\lambda^H/2$ and test accuracy of ReLU FCNs in SP trained on a 5k subset of CIFAR-10 until 99% training accuracy is achieved, with the weight variance σ_w^2 and learning rate multiplier $c = \eta\lambda_0^H$ as axes. As the color varies from blue to white, $\eta\lambda^H/2$ increases. (b, d) Same heatmaps with fixed $\sigma_w^2 = 2.0$, but varying s continuously.

$\boldsymbol{v}$ fixed through training and each entry set to one. In this model, the loss is sub-quadratic by *construction* $((\|U\boldsymbol{x}\|^2 - y)^2$ and the dynamics is given by a cubic map.

3.6 Predictions and Verifications in Real-world Scenarios

The preceding analysis offers broader insights and predictions for optimization in real-world models. In this section, we study realistic architectures with real and synthetic datasets and

examine the extent to which insights from the UV model generalize.

Experimental Setup: Consider a network $f(\boldsymbol{x}; \theta)$, with trainable parameters θ, initialized using normal distribution with zero mean and variance σ_w^2 in appropriate parameterization. In this section, we use the interpolating parameterization with $s \in [0, 1]$ (detailed in Appendix B.2.3), where networks with $s = 0$ are equivalent to networks in SP as width n goes to infinity and those with $s = 1$ are in μP. The network is trained on a dataset with P examples using MSE loss and GD. The learning rate is scaled as $\eta = c/\lambda_0^H$, where c is the learning rate constant, and λ_0^H is the sharpness at initialization. Additional details provided in figure captions and Appendix B.2.2.

3.6.1 Implications of Initialization and Parameterization for Real-world Models

The analysis in Section 3.5.1 unveils crucial insights into the implicit biases of parameterization in real-world networks. Figure 3.2(a, d) shows that μP networks begin training in a flat region of the landscape, where gradients point towards increasing sharpness, and approach the zero loss line while maintaining a low sharpness bias. In contrast, networks in NTP (or equivalently SP), characterized as large initializations, experience sharpness reduction during early training and might not approach with a minimal sharpness bias. The agreement of these observations with networks trained on real-world datasets (Figure 3.1) suggests that these inherent biases hold in practical scenarios.

3.6.2 Sharpness & Weight-Norm Correlation and the Origin of Four Regimes

Section 3.5.1 revealed that, for a wide variety of initializations, at early times trajectories move closer to the saddle point II, resulting in an interim decrease in λ (also proportional to the weight norm in this case), before eventually increasing. This critical point where all parameters are zero also exists in real-world models. We thus anticipate that in real-world models, the origin of the four training regimes may be related to a similar mechanism. This would predict a decrease in weight norm as training passes near the saddle point, followed by an eventual increase. In Appendix B.5, we validate this hypothesis. During the sharpness reduction and intermediate saturation regimes, we see a decrease in the weight norm, followed by an increase in the weight norm as the network undergoes progressive sharpening, following the prediction from the UV model.

3.6.3 The Phase Diagram of Edge of Stability

Result 1 presents a necessary condition for EoS to occur in the UV model: $\lambda_0 < {}^{c\|\boldsymbol{x}\|y}/\sqrt{n_{\text{eff}}}$. In real-world models, the initial sharpness λ_0^H can be controlled using the initial variance of the weights σ_w^2. Therefore, this result predicts that real-world models with (i) small initial weight variance σ_w^2, (ii) large interpolating parameter s, or (iii) large learning constant c are more likely to exhibit EoS behavior. Figure 3.4(a, c) show the phase diagram of EoS, validating these predictions. Additional phase diagrams in Appendix B.6 indicate an enhanced tendency for CNNs and ResNets to exhibit EoS. Furthermore, Figure 3.4(b, d) show the corresponding test accuracy heatmaps for the above experiments, revealing a positive correlation between observing EoS and improved model performance.

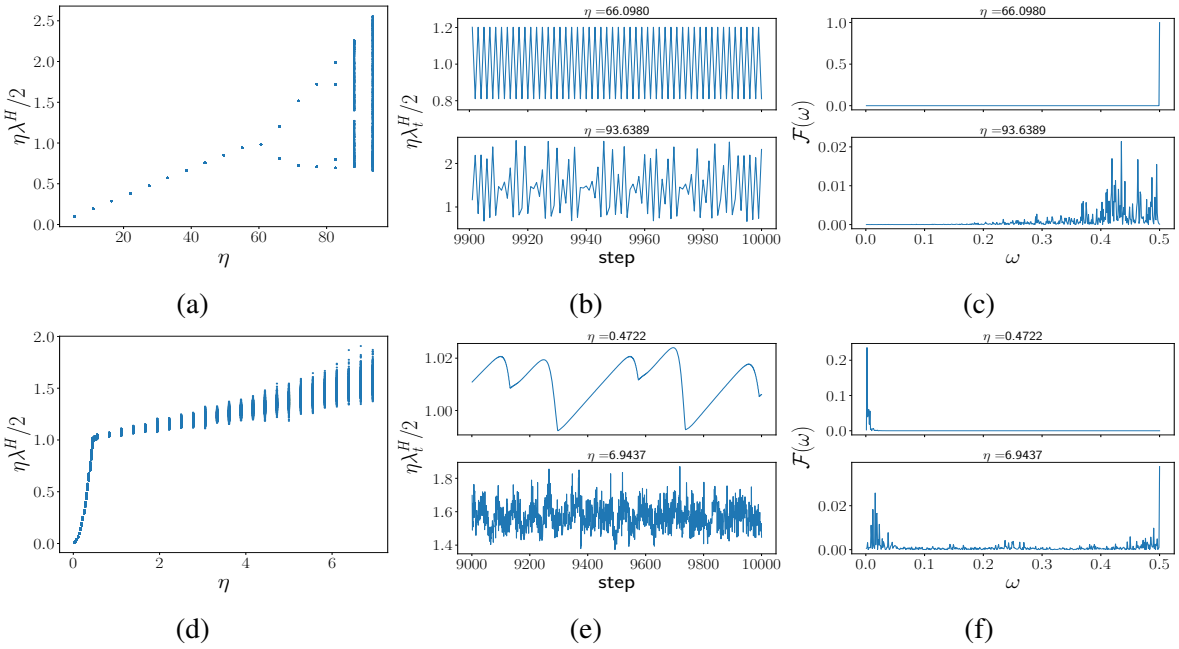


Figure 3.5: 2-layer linear FCNs trained on (first row) $5,000$ iid random examples with unit output dimension and (second row) $5,000$ CIFAR-10 examples. Different columns correspond to the bifurcation diagram, late-time sharpness trajectories, and the power spectrum of sharpness trajectories. The power spectrum is computed using the last 1000 steps of the trajectories.

3.6.4 Route to Chaos and Bifurcation Diagrams

The analysis in Section 3.5.2 unveiled structured fluctuations in λ at the EoS, with a period-doubling route to chaos observed as the learning rate is tuned. This motivates us to analyze fluctuations at the EoS in real-world models trained on realistic and synthetic datasets. Figure 3.5 shows the bifurcation diagram, late-time sharpness trajectories, and power spectrum of sharpness trajectories for a 2-layer linear FCN. In the first row, the model is trained on random synthetic data with $5,000$ iid examples with unit output dimension, whereas, in the second row, on a $5,000$ example subset of CIFAR-10. Similar to the UV model, FCNs trained on random data exhibit a period-doubling route to chaos, as shown in Figure 3.5(a). By comparison, FCNs trained on CIFAR-10 only show dense bands in the sharpness rather than exhibiting a clear period-doubling route to chaos.

On analyzing the sharpness trajectories at EoS, we observe long-range correlations in time in real datasets, with fluctuations increasing with the learning rate (see Figure 3.5(e)). By comparison, sharpness trajectories of models trained on random datasets exhibit short-period oscillations (see Figure 3.5(b)). The power spectrum of these sharpness trajectories further quantifies these observations, as shown in Figure 3.5(c, f). In the random dataset case, high-frequency modes corresponding to the period-doubling route to chaos emerge at EoS as shown in Figure 3.5(c). In contrast, real datasets exhibit low-frequency modes at small learning rates. As the learning rate is increased, high-frequency modes, reminiscent of the period-doubling route to chaos, start emerging (see Figure 3.5(f)). In Appendix B.7.1, we demonstrate that CNNs and ResNets trained on image datasets show dense bands of sharpness similar to those in FCNs.

To understand when the period-doubling route to chaos arises, we perform further analysis in Appendix B.7. A key determining feature appears to be whether the singular value spectrum of the input-input and output-input covariance matrices are flat or have power-law decay. In Appendix B.7.2, we show that a 2-layer FCNs trained on a random dataset with power-law singular value spectrum in the input exhibits dense sharpness bands. In Appendix B.7.3 we show that linear FCNs trained on synthetic datasets with random inputs, such as teacher-student settings and generative settings (details in Appendix B.2.1), exhibit the period-doubling route to chaos. In contrast, non-linear networks trained on these tasks exhibit dense sharpness bands as observed in real datasets. These observations shed some light on the nature of EoS observed in realistic settings. Nevertheless, a complete understanding of sharpness fluctuations at EoS requires a separate detailed examination.

3.7 Related Work

Using the top eigenvalue of the Neural Tangent Kernel (NTK) λ^K at initialization ($t = 0$), [73] revealed a 'catapult phase', $2/\lambda_0^K < \eta < \eta_{\max}$, in which training converges despite an initial spike in training loss. After the early training phase, sharpness continues to increase until it reaches a break-even point [78], beyond which GD dynamics typically enters the EoS regime [71]. This has motivated various theoretical studies to understand GD dynamics at large learning rates [81, 82, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93]. These include an analysis by [85], who show analyzed EoS in a 2-layer linear network using the norm of the last layer, restricting to cases that show progressive sharpening from initialization. [80] suggested that the UV model does not exhibit EoS behavior but showed that a modified model exhibits progressive sharpening and two-step oscillations at EoS using NTK as the proxy. In contrast, we show that the UV model shows EoS behavior under the appropriate choice of parameterization and training example. Meanwhile [81] analyzed two-step gradient updates of a single-neuron network and matrix factorization to gain insights into EoS. Furthermore, [93] analyzed a 2-layer linear network under logistic loss and demonstrated that sharpness at late training times oscillates around $2f/\eta\ell'$, where f is the network output and ℓ' is the derivative of the loss. Another study by [84] analyzed large learning rate dynamics of toy models which are characterized by a one-dimensional cubic map. Our work, in contrast, delves into various sharpness phenomena occurring throughout the training trajectory and analyzes their origins. It is worth noting that a concurrent study by [94] also examines sharpness throughout training. [95] relate sharpness dynamics to learning rate transfer in μP networks by showing that sharpness trajectories do not change appreciably when depth and width are varied. Given that our analysis spans the entire training trajectory, it relates to numerous

studies. Hence, we defer a comprehensive discussion of related works to Appendix B.1.

3.8 Discussion

In this chaper, we analyzed the crucial effect of initializations and parameterizations on the sharpness dynamics of neural networks and characterized conditions under which sharpness phenomena, such as sharpness reduction, progressive sharpening, and EoS, occur during training.

To develop a deeper understanding of these sharpness phenomena, we analyzed the UV model, which exhibits these sharpness trends. Through a fixed point analysis, we uncover the underlying mechanisms behind these complex sharpness phenomena, such as (i) the mechanism behind early sharpness reduction and progressive sharpening, (ii) the required conditions for the edge of stability, (iii) the crucial role of initialization and parameterization, and (iv) a period-doubling route to chaos on the edge of stability manifold as the learning rate is increased. Finally, we demonstrated that various predictions from this simplified model generalize to real-world scenarios.

3.8.1 Limitations and Future Work

While the UV model does capture major sharpness phenomena, it does have some limitations. The UV model does not capture the non-monotonic decrease in the training loss at EoS and should be viewed as a model describing a subspace of real-world settings. This is because the UV model dynamics is effectively described by two variables and the EoS condition $\lambda = 2/\eta$ puts another constraint. As a result, the loss also oscillates along with sharpness at EoS. In comparison, real-world neural networks have a large number of degrees of freedom, and the training

can still converge in stable eigendirections. Furthermore, the UV model does not exhibit loss catapult behavior for small effective widths ($n_{\text{eff}} = 1$). In such cases, the threshold for loss catapult ($\eta = {}^{2}/_{\lambda_0}$) is comparable to the maximum trainable learning rate. Finally, the UV model does not capture long-range correlations in sharpness dynamics observed in realistic datasets. Through extensive experiments, we identified that long-range correlations in the input covariance matrix results in these long-range correlations.

The applicability of the fixed point analysis extends well beyond the UV model and can be employed in settings involving complex architectures and adaptive optimizers. A prerequisite for applying this method is the closure of the dynamical equations describing the model. While this closure requirement is manageable for deep linear networks through additional function space variables, it becomes challenging for non-linear DNNs where such closure with a few variables is difficult. By analyzing the fixed points of such equations in broader classes of models where closure is achievable, we can gain significant insights into their training dynamics, thereby advancing our understanding of non-convex optimization in neural networks.

Finally, as evident from the discussion in this chapter, sharpness provides a natural guide for selecting the learning rate and its scheduling. However, determining an appropriate learning rate in practice remains challenging. For $\eta \leq c/\lambda^H$, if $c < 2$, the system typically undergoes progressive sharpening, making the loss landscape increasingly sharp and consequently forcing a rapid decay of the learning rate due to cascading effects. On the other hand, choosing $c > 2$ can lead to instabilities near the EoS. The principles governing optimal learning rate schedules are therefore still unclear and represent an important direction for future exploration.

Chapter 4: Grokking Modular Arithmetic

In the previous chapters, we discussed how architectural choices, initialization schemes, and parameterization influence neural network learning. However, we have not yet addressed the most crucial factor, arguably the one that has contributed most to the success of deep learning – the data itself.

Understanding the role of data and its effects on pretrained large-scale models remains extremely challenging, particularly from a quantitative and mechanistic perspective. To make progress, we turn to modular arithmetic datasets, which are both highly interpretable and non-trivial for neural networks to learn. These datasets enable us to investigate many core topics in modern deep learning in a controlled setting. Moreover, the dataset introduced in this chapter can be naturally extended to more complex scenarios that bridge toward real-world models (Chapter 5), while also remaining closely connected to practical problems of interest (Chapter 6).

Using this dataset, we construct controlled experiments where the mechanisms of generalization and memorization, an issue that has remained central to deep learning research, can be explicitly disentangled and analyzed. We show that studying such simplified yet expressive setups yields valuable insights into how neural networks acquire algorithmic and generalizable structures from data.

4.1 Introduction

The term *grokking*[1] refers to a phenomenon in which a model abruptly transitions from memorization (overfitting) to generalization (see Figure 4.1). This behavior was first reported by Power et al. [29] in modular arithmetic datasets. Specifically, it was observed when models were trained to solve modular arithmetic tasks of the form

$$(x + y) \bmod p = ?\,, \tag{4.1}$$

where $x, y \in \mathbb{Z}_p$ and p denotes a prime number. Notably, a similar transition is observed when plotting test accuracy as a function of training set size, i.e. number of (x, y) pairs used for training.

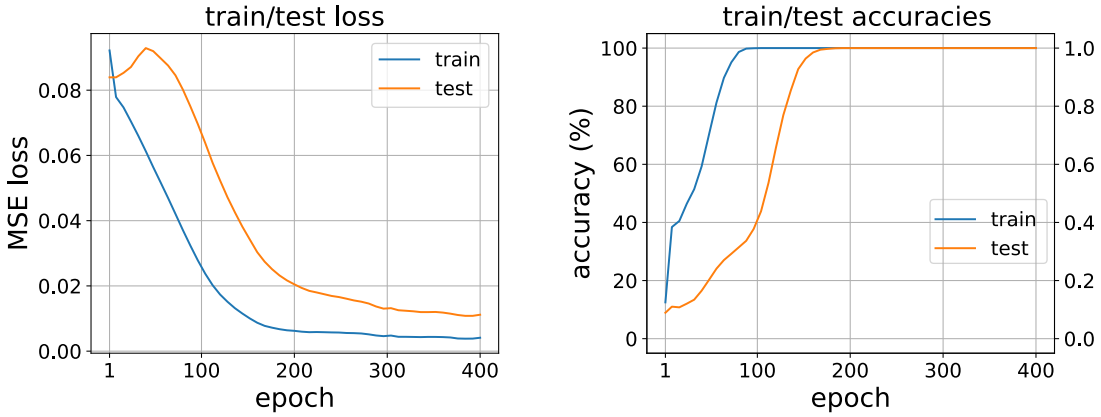


Figure 4.1: Left to right: **(a, b)** Losses and accuracies of a two-layer MLP trained on modular arithmetic data with four variables ($p = 11, \alpha = 0.5$). It exhibits delayed and rapid transition from memorization to generalization, which we call grokking.

During this transition, the model is believed to undergo an internal circuit change, in which

[1]The word "grok" was first coined by Heinlein [96] and has since taken on multiple interpretations. Here, we use it in the sense of "to understand intuitively."

a robust circuit capturing the true underlying data distribution emerges. This observation raises several important questions: (i) under what conditions (e.g., dataset size, training time, regularization) does this emergence occur? (ii) Are the memorization and generalization circuits qualitatively different? and (iii) is this phenomenon robust against data error/noise?

These questions are particularly relevant for understanding modern LLMs. Determining the conditions under which new model capabilities emerge, and uncovering the internal mechanisms underlying these behaviors, are crucial for gathering high-quality training data. Furthermore, because noisy data is inevitable at the scale of the internet, studying the robustness of these phenomena to noise becomes essential. By elucidating how generalization capabilities emerge in a controlled setting, we aim to gain insights that advance our understanding of complex LLMs and ultimately inform the development of more robust and reliable models.

4.2 Setup

Gromov [30] introduced a minimal setup to study the grokking phenomenon, where a key feature of the setting is the existence of an analytical solution that is qualitatively similar to the one learned through gradient descent. In this chapter, we adopt the same minimal setup: a two-layer MLP of width N with quadratic ($\phi(x) = x^2$) activations, trained to solve the modular arithmetic task described in Equation (4.1).

Such a setup can be described as follows:

$$\boldsymbol{f}(x, y) = \boldsymbol{W}\phi\left(\boldsymbol{W}_{in}(\boldsymbol{e}_x \oplus \boldsymbol{e}_y)\right) = \boldsymbol{W}\phi\left(\boldsymbol{U}\,\boldsymbol{e}_x + \boldsymbol{V}\,\boldsymbol{e}_x\right) . \tag{4.2}$$

Here, $\boldsymbol{e}_x, \boldsymbol{e}_y \in \mathbb{R}^p$ are `one_hot` encoded numbers x, y. "$\oplus$" denotes concatenation of vectors

($\boldsymbol{e}_x \oplus \boldsymbol{e}_y \in \mathbb{R}^{2p}$). $\boldsymbol{W}_{in} \in \mathbb{R}^{N\times 2p}$ and $\boldsymbol{W} \in \mathbb{R}^{p\times N}$ are the first and second layer weight matrices, respectively. ϕ is the activation function. $\boldsymbol{W}_{in}$ is decomposed into two $N \times P$ blocks: $\boldsymbol{U}, \boldsymbol{V} \in \mathbb{R}^{N\times p}$. $\boldsymbol{U}, \boldsymbol{V}$ serve as embedding vectors for m, n respectively. $\boldsymbol{f}(x, y) \in \mathbb{R}^p$ is the network-output on one example pair (m, n). The targets are `one_hot` encoded answers $\boldsymbol{e}_{(x+y)\,\%\,p}$.

The dataset consists of p^2 example pairs[2]; from which we randomly pick αp^2 examples for training (memorization), and use the other $(1-\alpha)p^2$ examples as a test set (generalization). We will refer to α as the *data fraction*. Throughout this chapter, networks are trained with Full-batch using AdamW optimizer [97] and MSE loss, unless otherwise stated.

Analytical Solution Choosing the quadratic activation function $\phi(x) = x^2$ makes the problem analytically solvable [30]. Upon grokking, the trained network weights are qualitatively similar to the analytical solution presented in Gromov [30].[3] For sufficiently large width N, the following analytical expression for the network weights achieves 100% generalization accuracy:

$$
\begin{aligned}
U_{ki} &= \left(\frac{2}{N}\right)^{-\frac{1}{3}} \cos\left(\frac{2\pi}{p} i\sigma(k) + \phi_k^{(u)}\right), \\
V_{kj} &= \left(\frac{2}{N}\right)^{-\frac{1}{3}} \cos\left(\frac{2\pi}{p} j\sigma(k) + \phi_k^{(v)}\right), \\
W_{qk} &= \left(\frac{2}{N}\right)^{-\frac{1}{3}} \cos\left(-\frac{2\pi}{p} q\sigma(k) - \phi_k^{(u)} - \phi_k^{(v)}\right),
\end{aligned}
\tag{4.3}
$$

where $\sigma(k)$ denotes a random permutation of k in permutation group S_N. The phase $\phi_k^{(u)}$ and $\phi_k^{(v)}$ are uniformly i.i.d. sampled between $(-\pi, \pi]$. Then the argument above can be easily shown[4] by writing down the network function with input pair (x, y), then `one_hot` embedding will

[2] There are p choices for x and y accordingly.
[3] Periodicity of weights is approximate in trained networks.
[4] Note this is adapted from [4], and originally appeared in [30].

naturally pick the rows from U and V with x, y as their indices:

$$
\begin{aligned}
f_z(x,y) &= \sum_{k=1}^{N} W_{zk}(U_{kx}+V_{ky})^2 \\
&= \frac{2}{N}\sum_{k=1}^{N} \cos\left(-\frac{2\pi}{p}\sigma(k)z - (\phi_k^{(u)}+\phi_k^{(v)})\right)\cdot \\
&\qquad \cdot\left(\cos\left(\frac{2\pi}{p}\sigma(k)x+\phi_k^{(u)}\right)+\cos\left(\frac{2\pi}{p}\sigma(k)y+\phi_k^{(v)}\right)\right)^2 \\
&= \frac{2}{N}\sum_{k=1}^{N}\left[\frac{1}{4}\cos\left(\frac{2\pi}{p}\sigma(k)(2x-z)+\phi_k^{(u)}-\phi_k^{(v)}\right)\right. \\
&\qquad +\frac{1}{4}\cos\left(\frac{2\pi}{p}\sigma(k)(2x+z)+3\phi_k^{(u)}+\phi_k^{(v)}\right) \\
&\qquad +\frac{1}{4}\cos\left(\frac{2\pi}{p}\sigma(k)(2y-z)-\phi_k^{(u)}+\phi_k^{(v)}\right) \\
&\qquad +\frac{1}{4}\cos\left(\frac{2\pi}{p}\sigma(k)(2y+z)+\phi_k^{(u)}+3\phi_k^{(v)}\right) \\
&\qquad +\boxed{\frac{1}{2}\cos\left(\frac{2\pi}{p}\sigma(k)(x+y-z)\right)} \\
&\qquad +\frac{1}{2}\cos\left(\frac{2\pi}{p}\sigma(k)(x+y+z)+2\phi_k^{(u)}+2\phi_k^{(v)}\right) \\
&\qquad +\frac{1}{2}\cos\left(\frac{2\pi}{p}\sigma(k)(x-y-z)-2\phi_k^{(v)}\right) \\
&\qquad +\frac{1}{2}\cos\left(\frac{2\pi}{p}\sigma(k)(x-y+z)+2\phi_k^{(u)}\right) \\
&\qquad \left.+\cos\left(-\frac{2\pi}{p}\sigma(k)z-\phi_k^{(u)}-\phi_k^{(v)}\right)\right]. \qquad (4.4)
\end{aligned}
$$

Here, we have highlighted the term that will give us the desired output with a box, where other terms vanish at a large width N due to the cancellation of random phases. Now, if we recall the discretize Fourier Transform of the Kronecker Delta, then we have at large width N

$$
f_z(x,y) \approx \frac{1}{N}\sum_{k=1}^{N}\cos\left(\frac{2\pi}{p}\sigma(k)(x+y-z)\right) \approx \delta^p(x+y-z)\,, \qquad (4.5)
$$

where the Kronecker Delta is defined modulo p as

$$\delta^p(a) = \begin{cases} 1 & \text{, if } a = np \quad (n \in \mathbb{Z}) \\ 0 & \text{, otherwise} \end{cases} \tag{4.6}$$

Consequently

$$f_z(x, y) \approx \begin{cases} 1 & \text{, if } z = x + y \mod p \\ 0 & \text{, otherwise} \end{cases}, \tag{4.7}$$

which is the desired function that can solve the task Equation (4.1).

Inverse Participation Ratio To characterize the generalizing representations quantitatively, we utilize a quantity familiar from the physics of localization[30, 98]: the *inverse participation ratio (IPR)*. Its role is to detect periodicity in the weight matrix.

Let $\boldsymbol{U}_{k\cdot}, \boldsymbol{V}_{k\cdot}(\boldsymbol{W}_{k\cdot})$ denote the k^{th} row(column) vectors of the weights $\boldsymbol{U}, \boldsymbol{V}(\boldsymbol{W})$. Consider the discrete Fourier transforms of these vectors, denoted by $\widetilde{\boldsymbol{U}}_{k\cdot}, \widetilde{\boldsymbol{V}}_{k\cdot}(\widetilde{\boldsymbol{W}}_{k\cdot})$. The Fourier decompositions of the periodic rows(columns) of these weights are highly localized. We leverage this to quantify the similarity of trained weights to the analytic solution (Equation (4.3)). To that end, we define the *Inverse Participation Ratio (IPR)* [30, 98, 99] for these vectors: [5]

$$\text{IPR}_k^{(u)} := \left(\frac{\|\widetilde{\boldsymbol{U}}_{k\cdot}\|_4}{\|\widetilde{\boldsymbol{U}}_{k\cdot}\|_2}\right)^4; \quad \text{IPR}_k^{(v)} := \left(\frac{\|\widetilde{\boldsymbol{V}}_{k\cdot}\|_4}{\|\widetilde{\boldsymbol{V}}_{k\cdot}\|_2}\right)^4; \quad \text{IPR}_k^{(w)} := \left(\frac{\|\widetilde{\boldsymbol{W}}_{\cdot k}\|_4}{\|\widetilde{\boldsymbol{W}}_{\cdot k}\|_2}\right)^4; \tag{4.8}$$

where $\|\cdot\|_P$ denotes the L^P-norm of the vector. One can readily see from (4.8) that $IPR \in$

[5] In general IPR is defined as $(\|U_{k\cdot}\|_{2r}/\|U_{k\cdot}\|_2)^{2r}$. We set $r = 2$, a common choice.

$[1/p, 1]$, with higher values for periodic vectors and lower values for non-periodic ones. It is useful to quantify the periodicity of each neuron in the hidden layer of the network (indexed by k). We define per-neuron IPR by averaging over the weight-vectors connected to the neuron:

$$\text{IPR}_k := \frac{1}{3}\left(\text{IPR}_k^{(u)} + \text{IPR}_k^{(v)} + \text{IPR}_k^{(w)}\right) . \tag{4.9}$$

Since trained networks generalize via periodic features, a larger population of high-IPR neurons leads to better generalization. To quantify the overall similarity of the trained network to the analytical solution (Equation (4.3)), we average IPR_k over all hidden neurons: $\overline{\text{IPR}} := \mathbb{E}_k\left[\text{IPR}_k\right]$.

To verify this intuition, we trained networks from random initialization with $p = 97$ and tracked the IPR values throughout training. As shown in Figure 4.2, the IPR values and their distribution before and after the transition align precisely with our expectations.

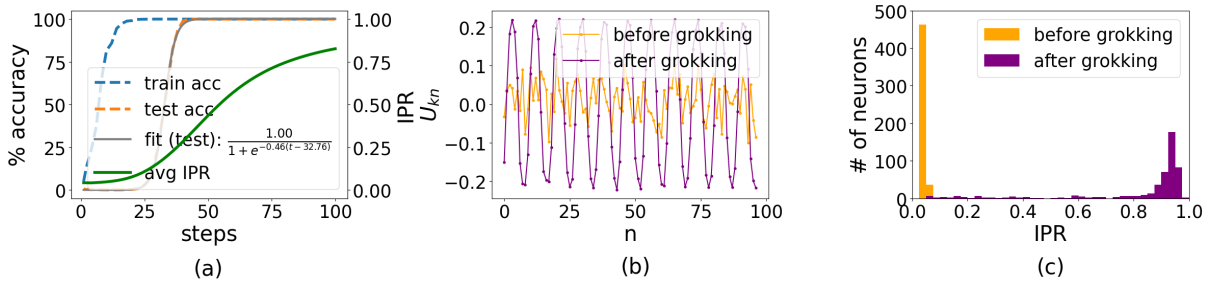


Figure 4.2: Grokking the modular addition task with $p = 97$ with 2-layer MLP, trained with AdamW. **(a)** Sharp transition in test accuracy, long after overfitting. $\overline{\text{IPR}} := \mathbb{E}_k\left[\text{IPR}_k\right]$ monotonically increases over time, indicating periodic representations. The test accuracy follows a logistic curve of the form $L/(1+e^{-k(t-t_0)})$, with $L = 1.00 \pm 0.01$, $k = 0.46 \pm 0.01$ and midpoint of steps at $t_0 = 32.76 \pm 0.03$. **(b)** Example row vector $(U_{k\cdot})$ before and after grokking. Generalization is achieved through periodic weights (Equation (4.3)). **(c)** Histogram of per-neuron IPRs before and after grokking – the distribution shifts to high IPR.

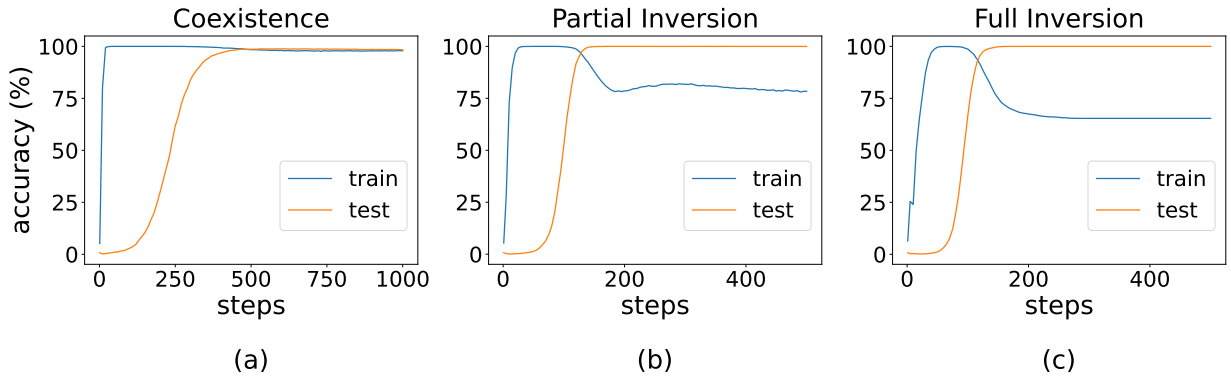


Figure 4.3: Training curves in various phases. All plots are made for networks trained with data-fraction $\alpha = 0.5$ and corruption-fraction $\xi = 0.35$. **(a)** No regularization: *Coexistence* of generalization and memorization – both train and test accuracies are high. **(b, c)** Adding weight decay: The network generalizes on the test data but does not memorize the corrupted training data, resulting in a negative generalization gap. Remarkably, the network predicts the "true" labels for the corrupted examples. We term these phases *Partial Inversion* and *Full Inversion*, based on the degree of memorization of corrupted data. ("inversion" refers to test accuracy being higher than train accuracy.)

4.3 Probe Generalizing Circuit with Label Noise

As mentioned earlier, this setup provides an ideal environment for studying memorization and generalization in the presence of noisy data. One might naturally conjecture that memorized noisy data should not activate neurons with high IPR, while generalization should continue to be driven by high-IPR neurons. This distinction allows us to separate the memorization circuit from the generalization circuit at a mechanistic level, thereby enhancing our understanding of these fundamental processes.

Our approach for better disentangling memorization from generalization is to introduce label corruption, where intuitively, a corrupted datapoint can only be memorized by the model, and the others can be solved by the true underlying algorithm. We choose a fraction ξ of the training examples, and replace the true labels with random labels. The corrupted labels are uniformly i.i.d. sampled from all possible labels. This is often called *symmetric* label noise in literature

[100].

Grokking MLP on the modular addition dataset is remarkably robust to label corruption. Even without explicit regularization, the model can generalize to near 100% accuracy with sizable label corruption. In many cases, the network surprisingly manages to "correct" some of the corrupted examples, resulting in *Inversion* (*i.e.* test accuracy $>$ training accuracy). We emphasise that this is in stark contrast to the common belief that grokking requires explicit regularization. Adding regularization makes grokking more robust to label corruption, with stronger *Inversion*. To quantify these behaviours, we observe the training and test performance of the network with varying sizes of training dataset, amount of label corruption and regularization. We summarize our findings in the form of empirical phase diagrams (Figures 4.4 and 4.5), featuring the following phases.

Coexistence. In the absence of explicit regularization, the network generalizes on test data *and* memorizes the corrupted training data. In other words, both train and test accuracy are close to 100% despite label corruption.

Partial Inversion. In this phase, the network generalizes on the test data but only memorizes a fraction of the corrupted training data. Remarkably, the network predicts the "true" labels on the remaining corrupted examples. In other words, the network *corrects* a fraction of the corrupted data. Consequently, we see $< 100\%$ train accuracy but near-100% test accuracy, resulting in a *negative* generalization gap (Figure 4.3(b))! We term this phenomenon *Partial Inversion*; where "inversion" refers to the test accuracy being higher than train accuracy. Remarkably, partial inversion occurs even in the absence of any explicit regularization, but only when there is ample

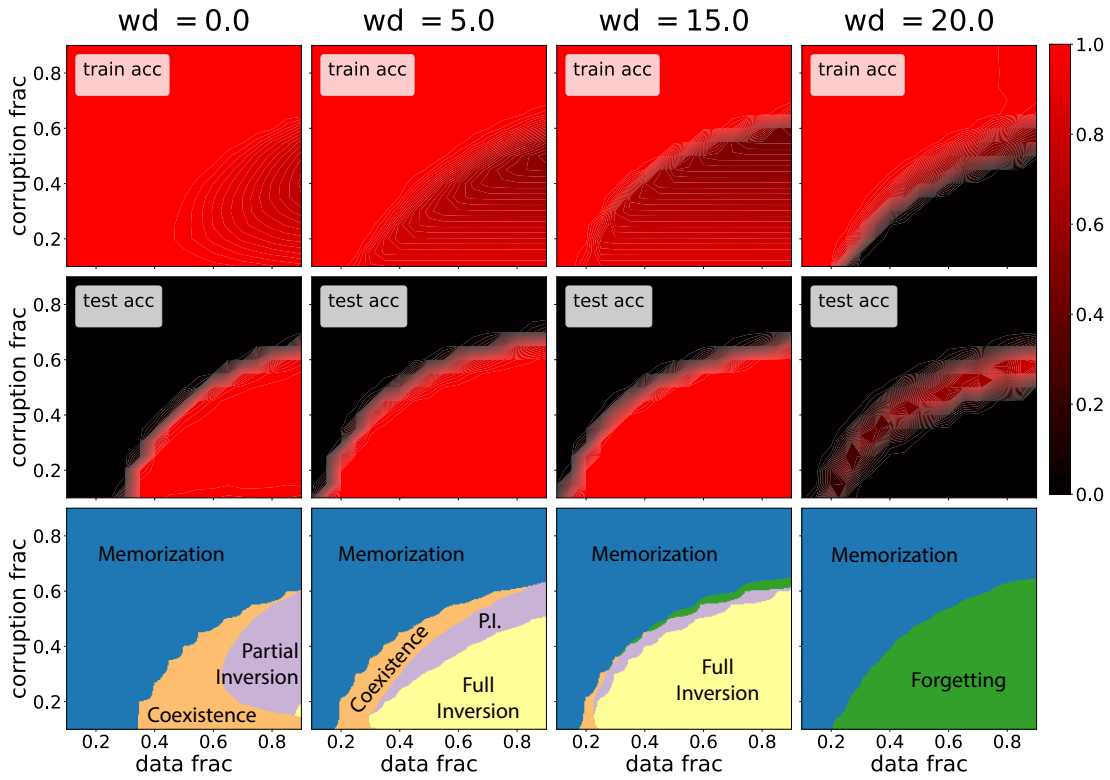


Figure 4.4: Modular addition phase diagrams with respect to data fraction α and corruption fraction ξ. Left to right: different weight decay values (denoted as "wd"). Top to bottom: **(top)** Training accuracy gets worse with higher regularization strength for the same amount of data. **(middle)** Test accuracy improves accordingly. **(bottom)**: Combined phase diagrams, data used from the first two rows.

training data (left-most panels in Figures 4.4 and 4.5).

Full Inversion. Upon adding regularization, the network often generalizes on the test set but does not memorize any of the corrupted examples. This results in training accuracy plateauing close to $100(1-\xi)\%$ (ξ is the *corruption fraction*); while the test accuracy is at 100% (Figure 4.3(c)). Increasing regularization enhances this phase (middle columns in Figures 4.4 and 4.5.

Memorization. With very high label corruption and/or very low training data-fraction the network memorizes all of the training data (corrupted as well as non-corrupted), but does not gener-

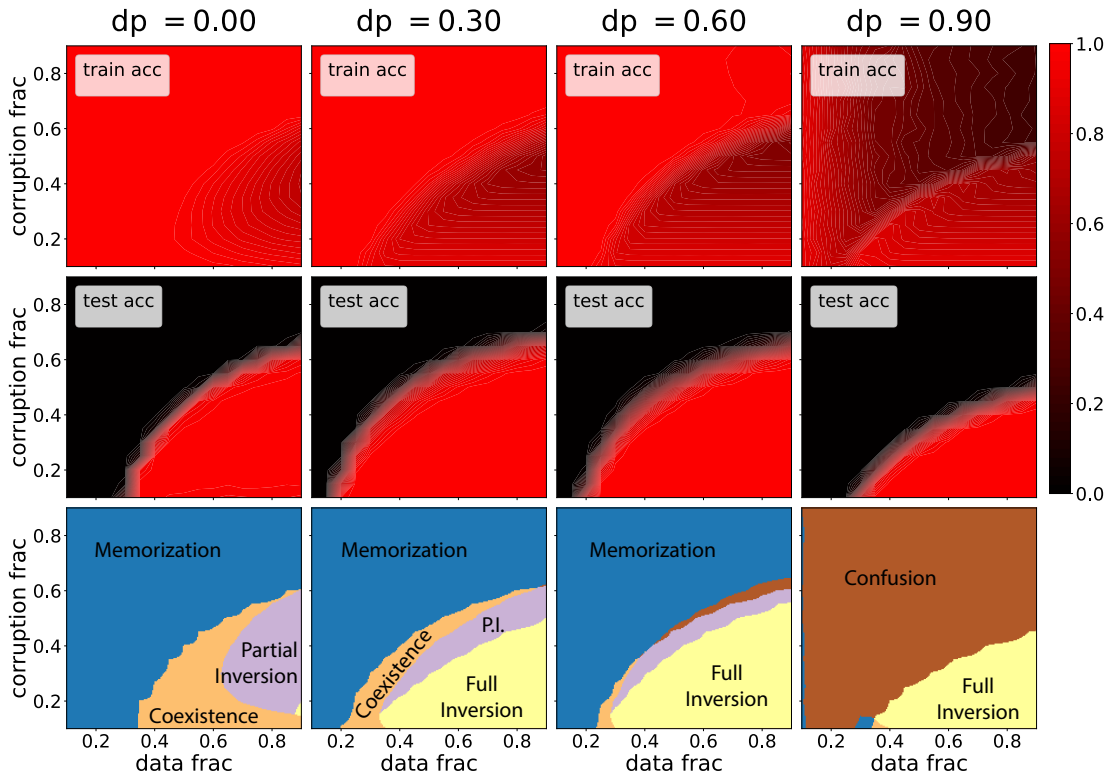


Figure 4.5: Modular addition phase diagrams with respect to data fraction α and corruption fraction ξ. Left to right: different Dropout values (denoted as "dp"). The behavior is very similar to weight decay, except for very large Dropout values, where the model is lost in the chaos introduced by the Dropout and fails to memorize the corrupted data.

alize. We get close to 100% accuracy on training data, but random guessing accuracy on test data in this phase.

Forgetting. With very high weight decay, the network performs poorly on both training and test data (right-most panel in Figure 4.4. Note here, we called it forgetting because we find that the performance drop is a result of the norms of weights collapsing to arbitrarily small values at a later training time. We also find that forgetting exclusively occurs in grokked networks with activation functions of degree higher than 1 (e.g., we use $\phi(x) = x^2$) and with adaptive optimizers.

Confusion. With very high Dropout values, the network becomes effectively narrow for each instance, resulting in reduced capacity to memorize. Consequently, even in the absence of grokking, the network does not manage to memorize all the corrupted labels (right-most panel in Figure 4.5(b)).

Here we provide Table 4.1 as a heuristic summarization the characteristics of each phase, with a more quantitative discussion in Appendix C.3.

	Coexistence	**Partial Inversion**	**Full Inversion**	**Memorization**	**Forgetting / Confusion**
Memorizes corrupted train data	Yes	Partially	No	Yes	No
Generalizes on test data	Yes	Yes	Yes	No	No

Table 4.1: Qualitative characterization of phases.

4.3.1 Generalizing and Memorizing Circuits

The generalizing features in our setup are "mechanistically" interpretable (Equation (4.3)). As a result, we can leverage measures of periodicity such as IPR to quantitatively characterize various phases presented above. It also allows us to analyze the effect of various regularization schemes. To that end, we state the following hypothesis and extensively examine it with experiments.

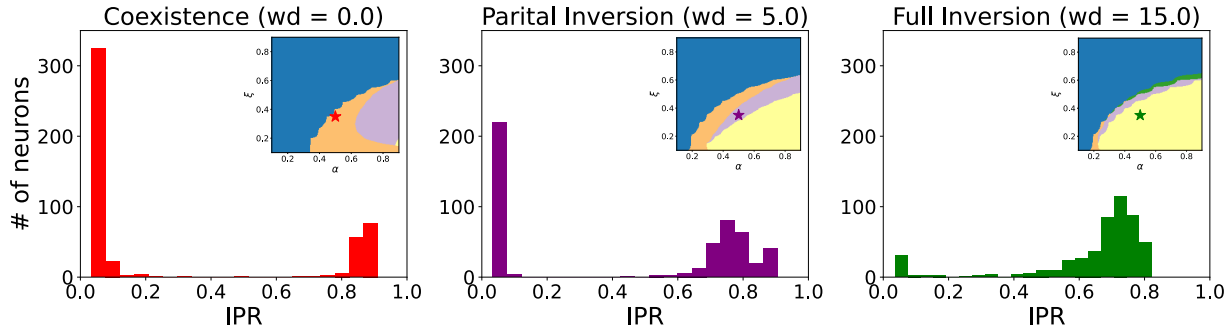
Hypothesis 4.1. *For a two-layer MLP with quadratic activation trained on (one-hot encoded) modular addition datasets with label corruption, the high IPR neurons facilitate generalization via feature-learning whereas the low IPR neurons cause memorization of corrupted training data.*

To test Hypothesis 4.1 we study the distribution of per-neuron IPR in various phases, with

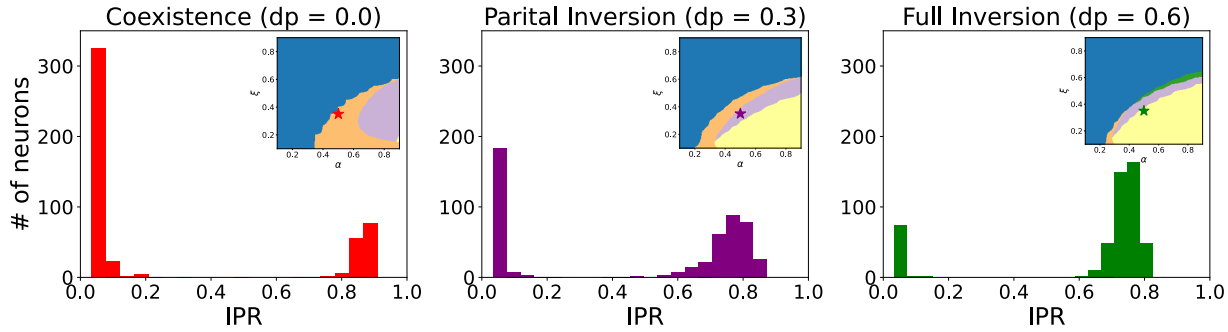
different levels of regularization. In the *Coexistence* phase, the network simultaneously generalizes and memorizes the corrupted train data. From Hypothesis 4.1, we expect the presence of both generalizing and memorizing neurons. Indeed, we find a bi-modal distribution of IPRs in trained networks (left column of Figure 4.6). Remarkably, the network not only distinguishes between the tasks of feature-learning and memorization, it also assigns distinct and independent sub-networks to each.

As mentioned in the previous section, adding regularization in the form of weight decay or Dropout makes the generalization more robust and facilitates *Inversion*. Since these phases see decreased memorization of corrupted labels, we expect to see a decrease in the number of memorizing neurons and a corresponding increase in the generalizing ones. Indeed, we observe a distribution-shift towards higher IPR in the *Partial Inversion* phase (middle column of Figure 4.6). In the *Full Inversion* phase (induced by weight decay or dropout), we see that memorizing neurons are almost entirely eliminated in favor of generalizing ones (right column of Figure 4.6). This provides a mechanism by which the network recovers the "true" labels even for the corrupted data – in the absence of memorizing sub-network, its predictions are guided by the underlying rule and not corrupted samples.

To further isolate the effect of individual neurons on performance, we perform the two complementary pruning experiments (Figure 4.7) : Starting from a trained network in the Co-existence phase, we gradually prune out neurons, one-at-a-time, (i) starting from the lowest IPR neuron (ii) starting from the highest IPR neuron, while keeping track of training and test accuracies. To distinguish network's ability to memorize, we also track the accuracies on corrupted and uncorrupted parts of the training dataset. Before pruning, the network has near-100% accuracy on both train and test data. In case (i) (Figure 4.7(left)), we see a monotonic decrease in accuracy on



(a) Regularization with weight decay. (wd denotes the weight decay value)



(b) Regularization with Dropout. (dp denotes the Dropout probability)

Figure 4.6: Distribution of per-neuron IPR for trained networks in various phases. Coexistence phase has a bimodal IPR distribution, where the high and low IPR neurons facilitate generalization and memorization, respectively. Regularization with weight decay or Dropout shifts the IPR distribution towards higher values – Generalizing neurons get more populous compared to memorizing ones; resulting in more robust generalization and *Inversion*. All plots are made for networks trained with data-fraction $\alpha = 0.5$ and corruption-fraction $\xi = 0.35$; with various regularization strengths.

corrupted examples as more and more low IPR neurons are pruned. The accuracy on test data as well as ucorrupted training data remains high. The resulting (overall) training accuracy plateaus around $100(1-\xi)\%$, corresponding to the fraction of uncorrupted examples. In case (ii) (Figure 4.7(right)), as more high IPR neurons are pruned, we see a decline in test accuracy (as well as the accuracy on the uncorrupted training data). The network retains its ability to memorize, so the accuracy on corrupted training data remains high.

We emphasize that, although each generalizing neuron learns the useful periodic features,

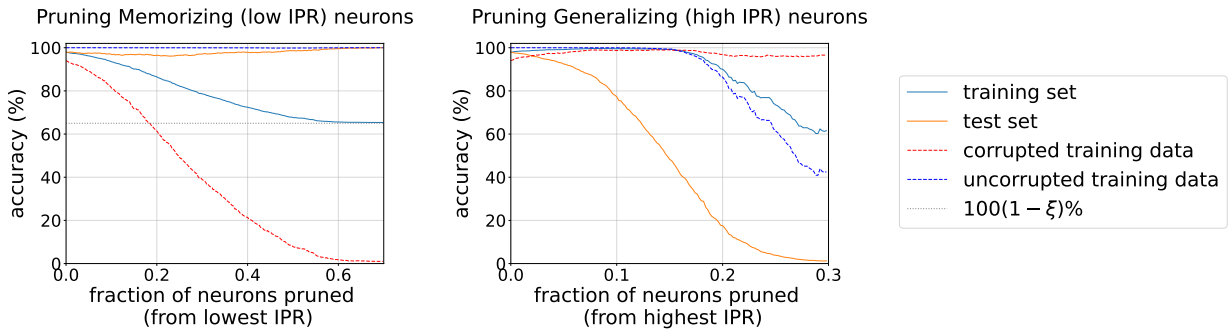


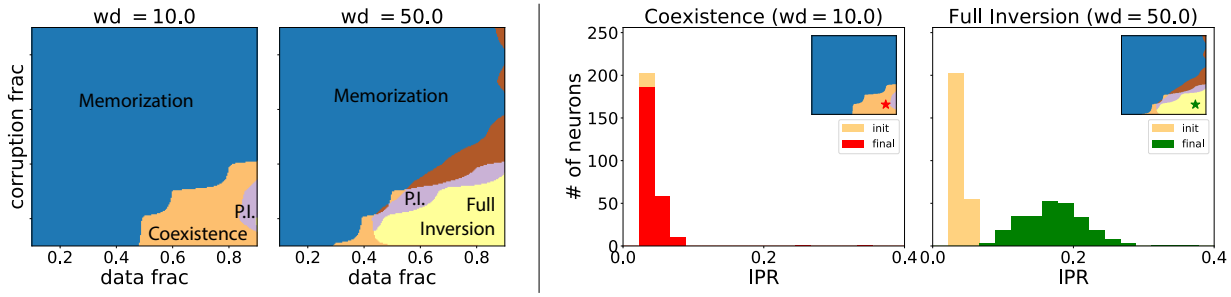
Figure 4.7: Progressively pruning neurons from a trained model (*Coexistence* phase), based on IPR. (data fraction $\alpha = 0.5$, corruption fraction $\xi = 0.35$). (Left) Pruning out neurons starting from the lowest IPR. The accuracy on corrupted training data decreases, providing evidence that low IPR neurons are responsible for memorization. Test accuracy as well as accuracy on uncorrupted training data remains high. (Right) Pruning out neurons starting from the highest IPR. Test accuracy decreases providing evidence that high IPR neurons are responsible for generalization. Accuracy on corrupted training data remains high.

the overall computation in the network is emergent: it is performed collectively by *all* generalizing neurons.

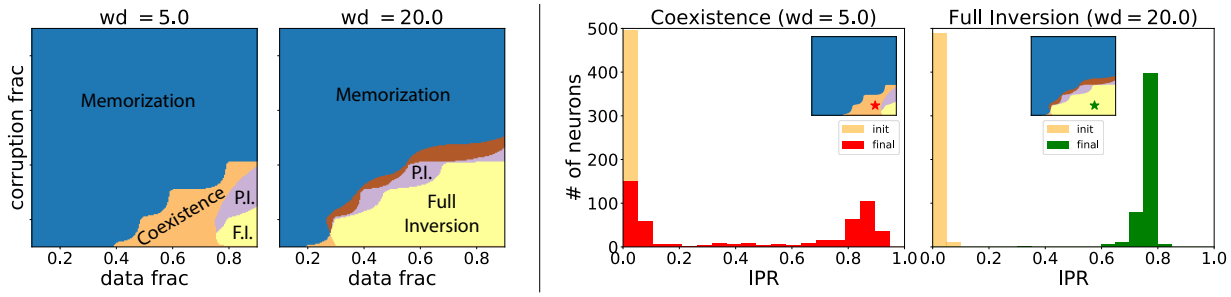
4.3.2 General Architectures

In this subsection, we aim to check if (i) we can quantify memorization/generalization capabilities of general architectures using IPR distribution of the neurons; (ii) our understanding of regularization can be generalized to such cases.

Hypothesis 4.2. *For any network that achieves non-trivial generalization accuracy on the modular addition dataset with label corruption, its operation can be decomposed into two steps: (I) The input (embedding) layer with row-wise periodic weight maps the data onto periodic features; (II) The remainder of the network nonlinearly implements trigonometric operations over the periodic neurons and maps them to the output predictions. We hypothesize that step (I) depends only on the dataset characteristics; and hence, is a universal property across various network architectures.*



(a) 1-layer Encoder-only Transformer



(b) 3-layer ReLU MLP

Figure 4.8: Modular Addition phase diagrams and IPR histograms with different architectures and weight decay values (denoted by "wd"). The IPR histograms are plotted for neurons connected to the input (embedding) layers.

To verify Hypothesis 4.2 and better understand the impact of regularization, we conduct further experiments on 1-layer Transformers and 3-layer ReLU MLPs. Our findings, illustrated in the first two columns of Figure 4.8 and paralleling the observations in Figure 4.4(a), indicate that increased weight decay promotes generalization over memorization. IPR histograms in the last two columns of Figure 4.8 further confirm that weight decay encourages the correct features and corroborates the assertions made in Hypothesis 4.2.

4.4 Related Works

Grokking and Interpretability. Grokking was first reported by Power et al. [29] for Transformers trained on modular arithmetic datasets. Liu et al. [101, 102] presented explanations for grokking in terms of quality of representation learning as well as weight-norms. They also

found grokking-like behaviors on other datasets. Nanda et al. [103], Zhong et al. [104] reverse-engineered grokked Transformer models and revealed the underlying learned algorithm. Barak et al. [105] observed grokking on the sparse-parity problem, which they attributed to the amplification of the so-called Fourier gap. Varma et al. [106] demonstrated new grokking-behaviours by investigating the critical amount of data needed for grokking.

Label Noise. Zhang et al. [107] showed that neural networks can easily fit random labels. A large body of work has been dedicated to developing techniques for robust training with label noise: explicit regularization[108, 109], noise-robust loss functions [110], sample selection [111, 112], noise modeling and dedicated architectures [113, 114, 115] etc.

Regularization and Pruning. Morcos et al. [116] investigated the interplay between label corruption and regularization in vision tasks. They showed that BatchNorm [109] and Dropout [108] have qualitatively different effects. Dropout effectively reduces the network-size but does not discourage over-reliance on a few special directions in weights space; while BatchNorm spreads the generalizing features over many dimensions. Bartoldson et al. [117] found that there is a trade-off between stability and generalization when pruning trained networks.

4.5 Discussion

Although the trained models in our setup are completely interpretable by means of analytical solutions, the training dynamics that lead to these solutions remains an open question. Solving the dynamics should also shed more light on the effects of regularization and label corruption.

While our work is limited to the exactly solvable models and algorithmic datasets, it paves

the way to quantitative investigation of generalization and memorization in more realistic settings.

Chapter 5: In-Context Learning and Skill Composition

Building on our understanding of modular arithmetic tasks from the previous chapter, we now increase the task complexity to achieve two objectives: (i) bring the setup closer to real-world tasks commonly used to evaluate LLM capabilities, and (ii) preserve interpretability and controllability, enabling us to study how, and under what conditions, in-context learning (ICL), one of the most important abilities of modern LLMs to adopt to unseen tasks, emerges within these models.

5.1 Introduction

One key property that has made LLMs widely adopted today is their ability to perform tasks absent from their training data. This capability is typically achieved through in-context learning [16]. More importantly, LLMs can perform an even larger variety of very complex tasks upon appropriate prompting or fine-tuning. The latter ability to perform complex tasks is usually attributed to the following mechanism. First, LLMs learn a large variety of simple tasks and, then, how to *compose* those skills to form very complex skills [31]. Furthermore, LLMs also exhibit "emergent capabilities" – a sudden emergence of a new complex skill as a function of scale (either model size, compute or data) [118, 119]. It is plausible that these sudden performance improvements are due to one or both of these mechanisms. For example, LLMs show grokking

on algorithmic tasks [29], which results from the model learning very structured representations [30, 101, 103]. Once these representations emerge, the model abruptly learns how to perform the task.

In this Chapter, we set out to examine skill composition both empirically and mechanistically. Inspired by the prior work that investigated emergence of in-context learning on linear regression tasks [120, 121], we introduce a finite collection of discrete modular arithmetic tasks [29] generalized to the in-context learning setting. Each task corresponds to learning a linear function $z = a\,x + b\,y \bmod p$ over $\mathbb{Z}_p$ from the examples provided in context of the Auto-regressive Model(AM). In the bi-variate case there are p^2 such functions labeled by the vector (a, b). The main objective of this algorithmic dataset is to probe *how* AM utilizes the tasks it has learnt during training to solve the new tasks.

Our analysis shows that the solution found by the AM after optimization is qualitatively different from the linear regression cases studied before [120]. In those cases, due to the continuous nature of the task, AM develops an emergent first-order optimization method that minimizes an emergent quadratic loss function. Furthermore, as it was shown in [120], a single linear attention layer can solve the regression problem, while adding extra layers and non-linearities slightly modifies the gradient descent. In the modular arithmetic case, AM first learns how to solve the pre-training tasks and later (assuming enough different tasks) develops a generalizing solution by combining the solved tasks.

The algorithmic dataset is illustrated in Figure 5.1. Our main findings are (see Figure 5.2): (i) there are **four** different phases in the end of pre-training depending on the number of tasks, $n_{\text{i.d.}}$, and number of examples per task, α. (ii) At inference time, there is a generalization transition in the number of few-shot examples, as the number of examples grows, the models starts to

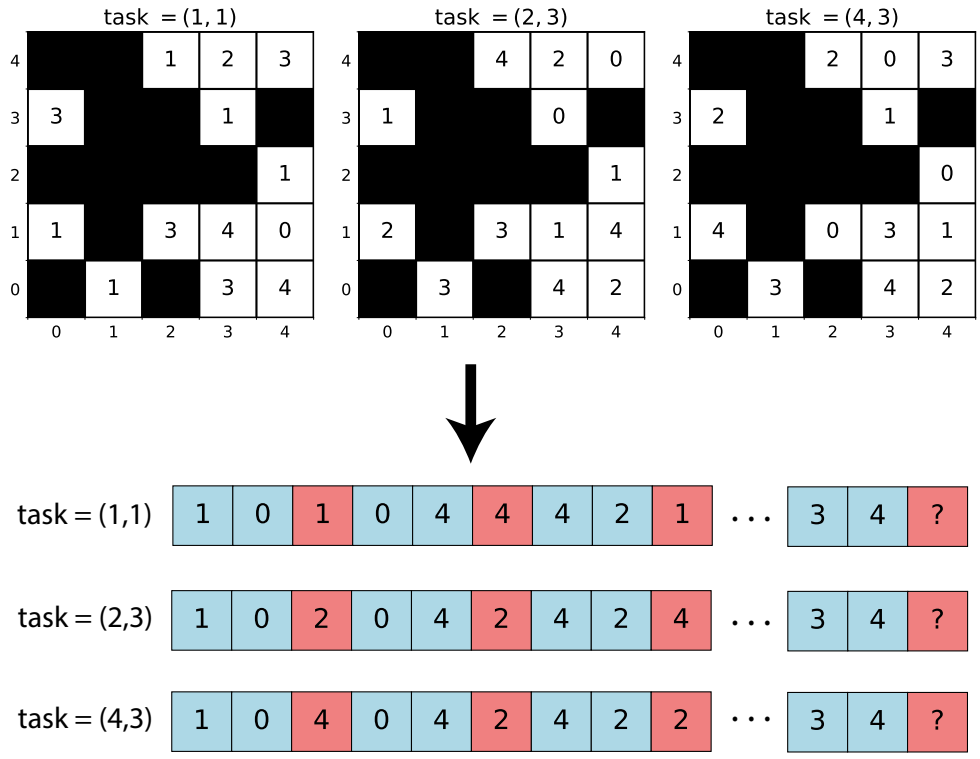


Figure 5.1: The dataset. The tasks are labeled by vectors $(a, b) \in \mathbb{Z}_p^2$. Each table contains examples of $ax + by$ mod p. A fraction $1 - \alpha$ of the examples is blacked out; while the remaining examples are flattened into a single "document" in the batch. Each document is organized as a collection of triples $(x, y, ax + by)$ for x, y from the training set (*i.e.* not blacked out in the table). To leverage training compute more effectively we predict not just the last element, but also other elements colored in red. Our training is similar to the traditional next-token prediction (autoregressive); with the main difference that we predict every third token, which are marked in red (x and y are uncorrelated). Every task appears exactly the same number of times in each batch.

generalize. This effect is somewhat similar to the transition in sample complexity for the modular arithmetic found in [29]. (iii) model develops a striking circular representation for all of the tasks that naturally generalizes the circular representations found in the original work [29].

We further find that the deeper models are easier to optimize, but much harder to interpret. The optimization is discussed in more detail in the main text. Here we will highlight that optimization for these tasks is challenging and the AM tends to prefer the minimum that just solve a

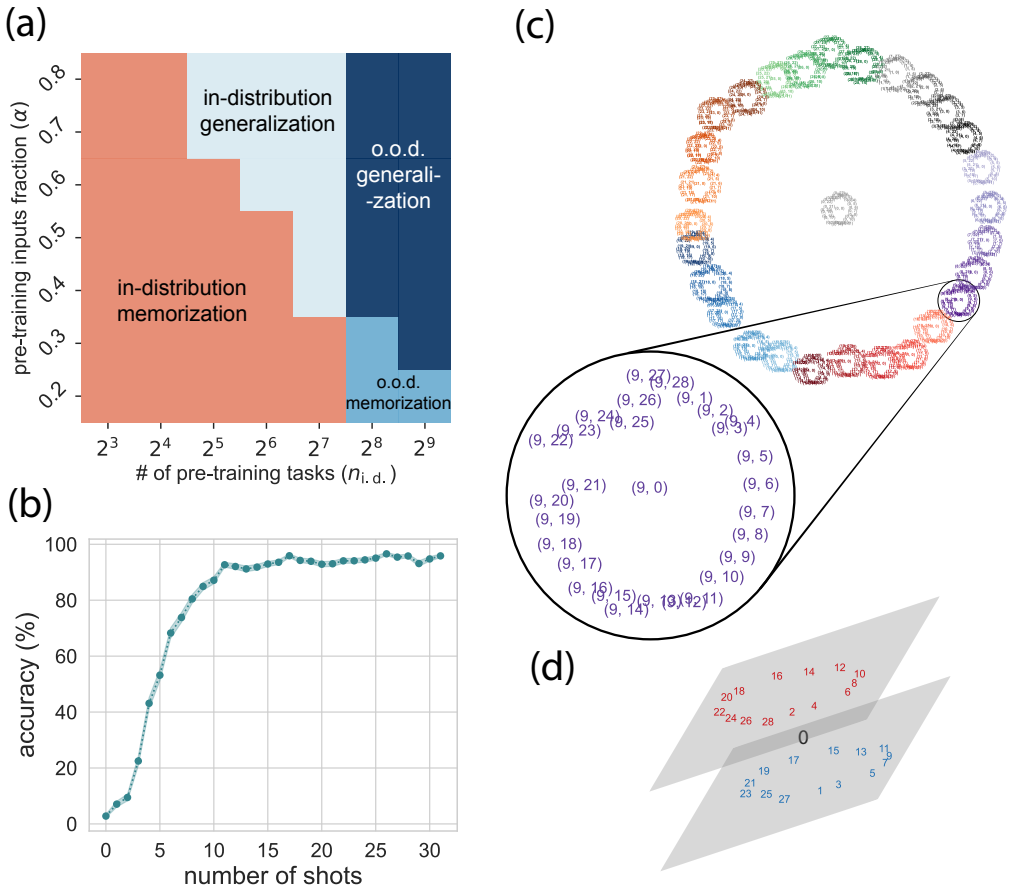


Figure 5.2: Preview of results. **(a)** Phase diagram for a six-layer model. We find four different phases. (1) *in-distribution memorization*: The model *only* performs well on tasks (a, b) *and* examples (x, y) from the training set – it does not generalize on unseen examples or tasks. (2) in-distribution generalization: model generalizes on unseen examples (x, y) but not on unseen tasks (a, b). (3) out-of-distribution memorization: model generalizes on unseen tasks (a, b) but only for examples (x, y) it has seen during training. (4) out-of-distribution generalization: model generalizes on unseen tasks (a, b) for seen as well as unseen examples (x, y). We focus on investigating phase (4) in more detail. **(b)** In-context sample complexity. Accuracy of the model in phase (4) as a function of the number of few-shot examples. **(c)** Representations developed by one of the attention heads in the first layer. These are projections of the embedding of a pair of numbers onto the two largest principal components (PCs) of the internal representation formed after passing through the attention layer and projection matrix. **(d)** First 3 PCs of embeddings separate log_{27}-annotated numbers into even/odd planes, with 0 sandwiched between them.

handful of tasks and memorize the training set. To avoid such minima we make sure that *every batch* contains equal number of tasks (meaning that no tasks is over- or under-represented in each batch). We further find that for larger models early stopping is necessary because the generalizing

solution is transient.

5.2 Preliminaries

Linear Modular Functions. We consider modular arithmetic tasks of the form: $z_i^t = a^t x_i + b^t y_i \mod p$. We will refer to the coefficients $(a^t, b^t) \in \mathbb{Z}_p^2$ as the *task vector*. The superscript $t \in \{1, \cdots, p^2\}$ labels the p^2 possible tasks. We will refer to $(x_i, y_i) \in \mathbb{Z}_p^2$ as the *input vector*, which is labeled by the subscript $i \in \{1, \cdots, p^2\}$.

In-Context Learning with Transformers. We use GPT-like transformers [16] with ReLU activation function and Rotary Positional Embedding (RoPE) [122]. The model has d consecutive blocks, H attention-heads, and embedding dimension d_{embed}. Each number is tokenized as an independent token. The pre-training is done following a slightly modified next-token prediction setup, with sequences of the form:

$$\boldsymbol{s}^t = (x_1 \quad y_1 \quad z_1^t \quad x_2 \quad y_2 \quad z_2^t \quad \cdots \quad x_{n_{\text{ctx}}} \quad y_{n_{\text{ctx}}} \quad z_{n_{\text{ctx}}}^t) \in \mathbb{Z}_p^{3 \times n_{\text{ctx}}}, \tag{5.1}$$

where n_{ctx} is the maximum number of in-context examples. The model is asked to predict *only* the labels $\{z_1^t, \cdots, z_{n_{\text{ctx}}}^t\}$. We emphasize that we do not explicitly provide the task vectors (a^t, b^t) to the model – this information is implicit in the in-context labels $\{z_i^t\}$. In order for the model to generalize, it must determine the underlying task vector (a^t, b^t) from the few-shot examples.

Generalization. There are two notions of generalization in this setup. (i) *In-distribution*: Generalization to *unseen* input vectors (x_i, y_i), but on task vector (a^t, b^t) the model has seen during pre-training. (ii) *Out-of-distribution*: Generalization to task vectors the model has *not* seen during

= performs well = performs poorly

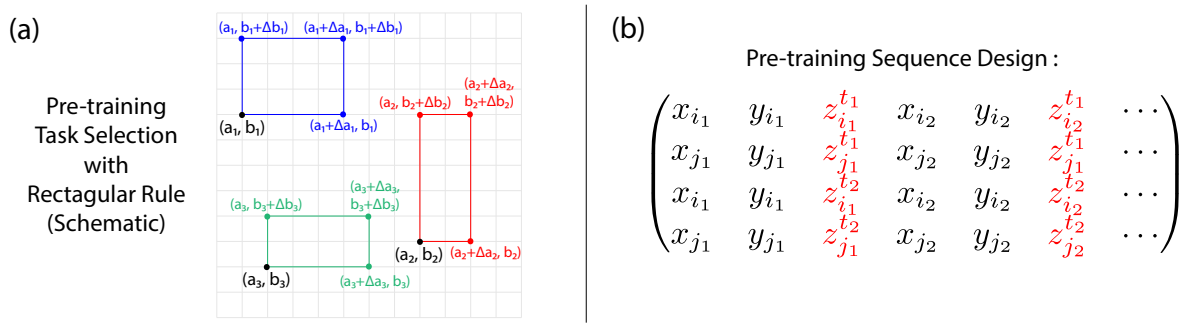
	$S^{i.d.}_{train}$	$S^{i.d.}_{test}$	$S^{o.o.d.}_{train}$	$S^{o.o.d.}_{test}$
In-distribution memorization	✓	✗	✗	✗
In-distribution generalization	✓	✓	✗	✗
Out-of-distribution memorization	✓	✗	✓	✗
Out-of-distribution generalization	✓	✓	✓	✓

Figure 5.3: The performance of the model is characterized into 4 phases, based on their accuracy on seen/unseen tasks and seen/unseen inputs.

pre-training. To clearly separate these regimes, we split the task vectors into in-distribution (i.d.) set $\mathcal{T}_{\text{i.d.}} \coloneqq \{(a^t, b^t)\}_{\text{i.d.}}$ and out-of-distribution (o.o.d.) set $\mathcal{T}_{\text{o.o.d.}} \coloneqq \{(a^t, b^t)\}_{\text{o.o.d.}}$. Similarly, we split the input vectors into train and test sets: $\mathcal{X}_{\text{train}} \coloneqq \{(x_i, y_i)\}_{\text{train}}, \mathcal{X}_{\text{test}} \coloneqq \{(x_i, y_i)\}_{\text{test}}$. This results in four distinct sets of sequences constructed from those sets; we name them $S^{\text{i.d.}}_{\text{train}}, S^{\text{i.d.}}_{\text{test}}, S^{\text{o.o.d.}}_{\text{train}}$ and $S^{\text{o.o.d.}}_{\text{test}}$. The set $S^{\text{i.d.}}_{\text{train}}$ is used for pre-training, while the other three sets are used for evaluations.

Pre-Training Task Selection and Sequence Design. We always sample the pre-training task vectors $\mathcal{T}_{\text{i.d.}}$ in sets of 4, following the rectangular rule, shown in Figure 5.4(a). Additionally, each batch contains an equal representation from all the task vectors in the set $\mathcal{T}_{\text{i.d.}}$. Moreover, all the tasks share the same sequence of inputs. For example, a batch with two different task vectors $(a^{t_1}, b^{t_1}); (a^{t_2}, b^{t_2})$ and two distinct input sequences per task (resulting in four total sequences) is shown in Figure 5.4(b).

This structured approach creates a coherent signal from the sequences within each batch; ensuring that the model learns multiple task vectors with reasonable batch sizes. Alternatively, if the batches are sampled i.i.d., then the model is confused by the batch noise and cannot learn any



$$\begin{pmatrix} x_{i_1} & y_{i_1} & z_{i_1}^{t_1} & x_{i_2} & y_{i_2} & z_{i_2}^{t_1} & \cdots \\ x_{j_1} & y_{j_1} & z_{j_1}^{t_1} & x_{j_2} & y_{j_2} & z_{j_1}^{t_1} & \cdots \\ x_{i_1} & y_{i_1} & z_{i_1}^{t_2} & x_{i_2} & y_{i_2} & z_{i_2}^{t_2} & \cdots \\ x_{j_1} & y_{j_1} & z_{j_1}^{t_2} & x_{j_2} & y_{j_2} & z_{j_2}^{t_2} & \cdots \end{pmatrix}$$

Figure 5.4: Structured selection of pre-training tasks and sequences. (a) Schematic of pre-training task selection with the Rectangular rule. (b) "Balanced" sequence design with 2 tasks and 2 examples per task.

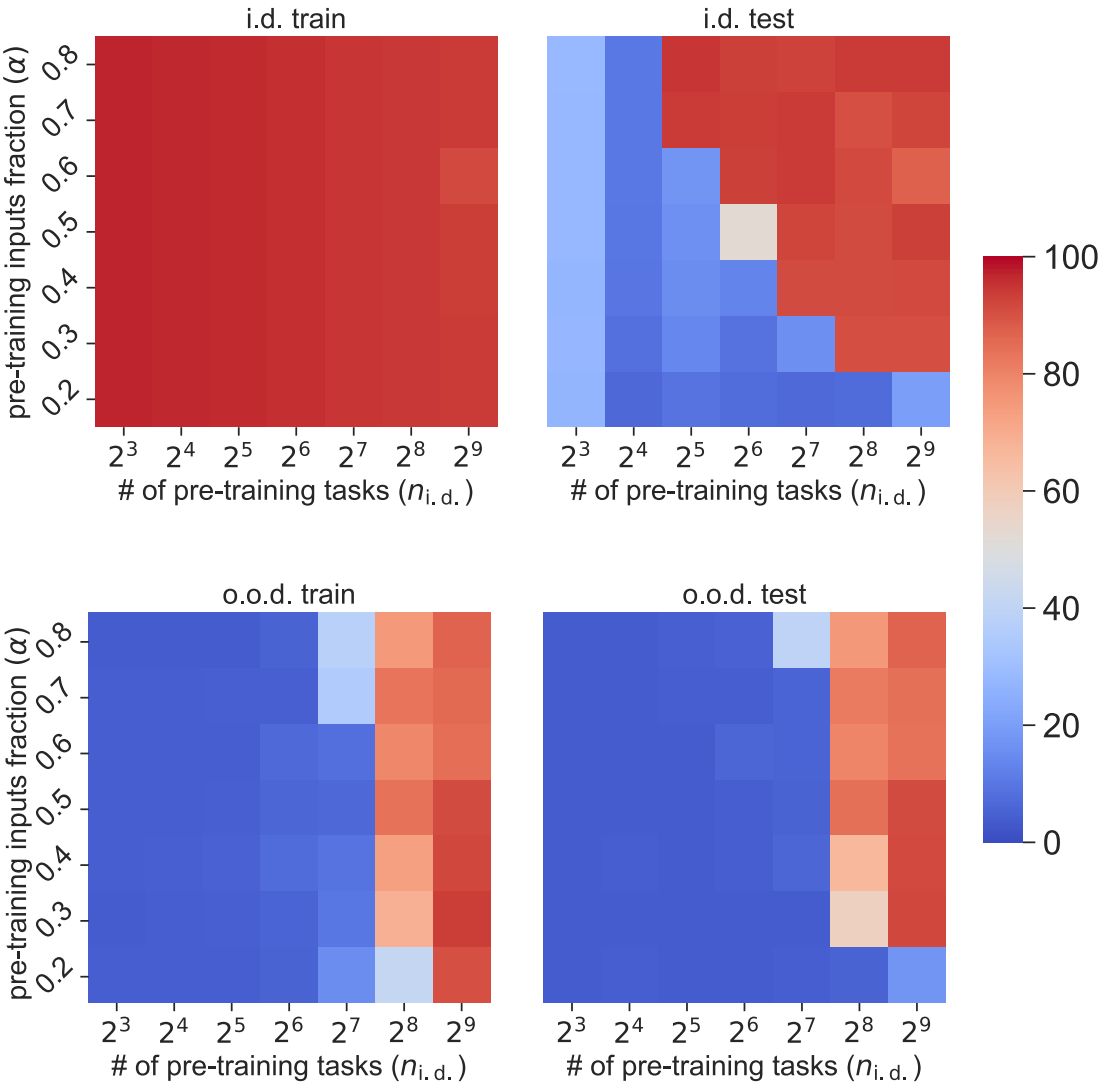
tasks.

Detailed discussions on task selection and sequence design are presented in Appendix D.4.

Default Setting. Unless stated explicitly, we will use $p = 29$, the number of heads $H = 4$, and embedding dimension $d_{\text{embed}} = 512$, with $n_{\text{ctx}} = 32$ in-context examples. All models are trained with AdamW optimizer [97] with batch size $B = 1024$ for 200k steps. We have also tied the embedding layer of the model with the readout layer.

5.3 Emergence of In-Context Learning and Task Composition

In this section, we demonstrate that a transformer model with depth $d \geq 2$ can develop ICL and out-of-distribution generalization on modular arithmetic tasks. We delve deeper into the two notions of generalization (i.d. and o.o.d.), and discuss the relevant factors. We find that the model's ability to generalize out-of-distribution is predominantly determined by the number of pretraining tasks $n_{\text{i.d.}}$.

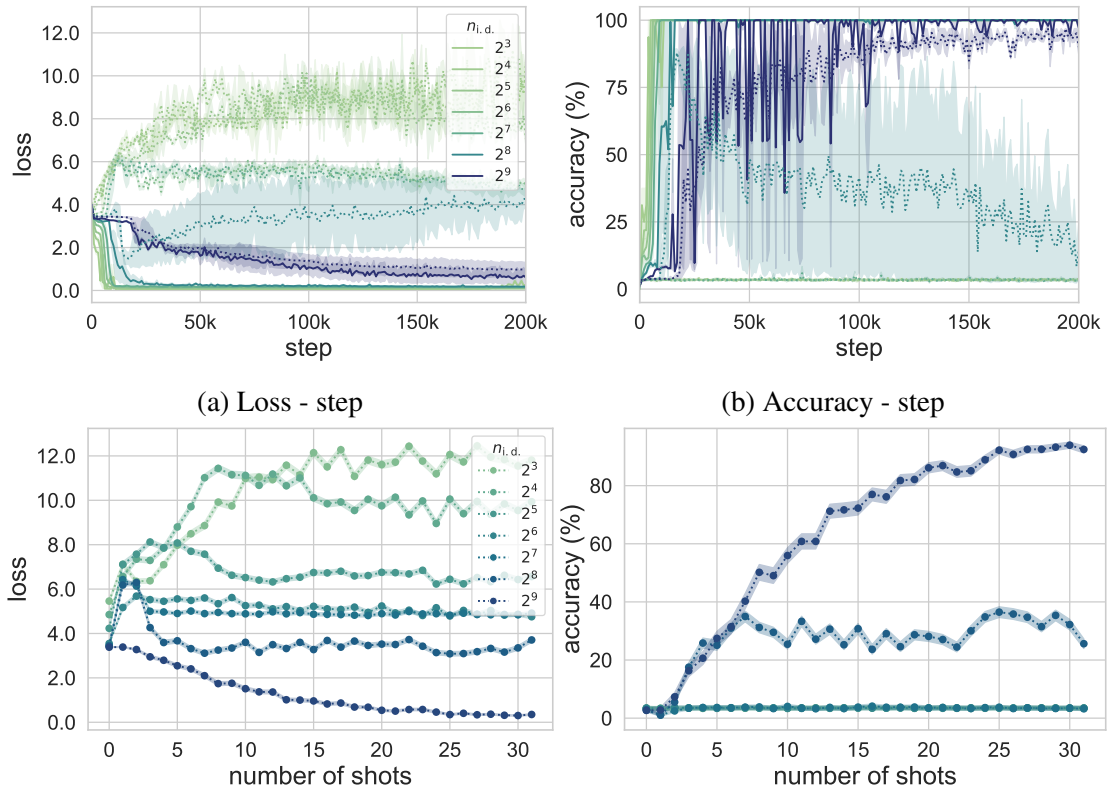


(a) Accuracy phase diagrams of all four sets

Figure 5.5: Phase diagram for the depth $d = 6$ models. Accuracy on all four sets used to plot the 5.2 phase diagram, with an early stopping applied. Notably, in the regions when models generalize to o.o.d. sets, the pre-training performance degrades.

5.3.1 Transition Driven by the Number of Tasks

In Figure 5.5, we show the accuracy of $d = 6$ models vs the number of training tasks $n_{\text{i.d.}}$ and the number of few-shot examples quantified by the fraction of the total number of data points, α; on sets $S^{\text{i.d.}}_{\text{train}}$, $S^{\text{i.d.}}_{\text{test}}$, $S^{\text{o.o.d.}}_{\text{train}}$ and $S^{\text{o.o.d.}}_{\text{test}}$. The phase diagram in Figure 5.2 is constructed



(a) Loss - step

(b) Accuracy - step

(c) Loss - # of shots

(d) Accuracy - # of shots

Figure 5.6: Selected training curves for experiments in Figure 5.5. **(a, b)** $\alpha = 0.6$ training accuracy and o.o.d. test accuracy (dotted line). For $n_{\text{i.d.}} = 2^8$, we notice that the o.o.d. generalization ability of the model first improves then degrades as we train longer; **(c, d)** $\alpha = 0.6$, loss and accuracy vs context length, measured on $S_{\text{test}}^{\text{o.o.d.}}$ at the end of training, where for $n_{\text{i.d.}} = 2^8$ case the ICL ability fades away.

by merging the last shot accuracy version of these four diagrams shown in Figure D.17(a) of Appendix D.7.

The ability of the model to generalize in-distribution increases with α, as can be seen by comparing the first two panels of Figure 5.5. This behavior is in correspondence with the original work on grokking, where the transition to generalizing solution is driven by the amount of data. Further, we observe that an increase in $n_{\text{i.d.}}$ enhances the in-context sample efficiency, *i.e.* the model generalizes at inference time with fewer few-shot examples. This indicates the onset of the transition from a task-memorizing solution to the one that generalizes out-of-distribution. The

model switches to a new algorithmic way of solving the task and the solution is more few-shot-sample-efficient.

Shifting our focus to the last two panels of Figure 5.5, we see that when $n_{\text{i.d.}} \geq 256$, the model can solve new tasks that were absent in the training set. Notably, there appears to be a trade-off between memorization and generalization when the model attains this o.o.d. generalization ability. As the o.o.d. performance increases, the pre-training performance simultaneously degrades. This phenomenon indicates a shift in the algorithm implemented by the model. Prior to this transition, the model primarily needed to select possible vectors from the list of memorized tasks and apply them. However, post-transition, the model adopts a more universal approach to solve the task in-context. We emphasize, that the model learns to perform ICL in *both scenarios*. The difference lies in the approach to generalization. When the model can only generalize in-distribution it's task is to classify the sequence as one of the seen tasks or as unknown. Once it matches the sequence to one of the memorized task vectors, it does well for x, y pairs that only appear in the test set. However, as the number of tasks vectors grows the model fails to store them all and is forced to find a method of determining the task vector algorithmically at inference time. In that case model performs equally well on seen and un-seen tasks alike. In fact, the small two-layer model we study has such a low capacity that it entirely skips the in-distribution generalization phase and immediately jumps from pure memorization to out-of-distribution generalization.

Next, to further illustrate the effect of task diversity, we plot the pre-training accuracy (set $S^{\text{i.d.}}_{\text{train}}$) and the o.o.d. test accuracy (set $S^{\text{o.o.d.}}_{\text{test}}$) as a function of training steps (Figure 5.6(a, b)); for various values of $n_{\text{i.d.}}$. We observe a clear memorization-to-generalization transition as task diversity increases. Interestingly, for $n_{\text{i.d.}} = 2^8$, the ICL ability on $S^{\text{o.o.d.}}_{\text{test}}$ set exhibits non-

monotonic behavior, where the o.o.d. performance rises and falls along the training process. This phenomenon is likely due to a competition between the memorizing and generalizing circuits inside the model. Note that this phenomenon is akin to the one analyzed in Singh et al. [123], albeit with a different setup.

Further evidence supporting the two-circuit competition can be observed in panel (c) of Figure 5.6. The loss curves show a "monotonic $\rightarrow$ non-monotonic $\rightarrow$ monotonic" transition as the task diversity increases. With a minimal number of pre-training tasks, the model primarily engages in memorization, resulting in a monotonically increasing o.o.d. loss curve. As the number of pre-training tasks increases, the loss curve exhibits non-monotonic behavior, indicating competition between two distinct neural circuits. This transient nature of o.o.d. generalization for $n_{\text{i.d.}} = 2^8$ is a peculiar case where memorization circuits are initially suppressed but eventually prevail. With substantial task diversity, the circuits responsible for generalization take over, culminating in a monotonic loss curve. Similar insights can be derived from examining the monotonicity of the accuracy curves in panel Figure 5.6 (d).

5.3.2 Effect of Model Size and Task Difficulty

A natural question to ask is if similar phenomena can be observed with different model sizes or task difficulties. Here we present our results with $d = 4$ and $d = 2$ in Figure 5.7 and leave the results for other prime p values in Appendix D.8.

When comparing phase diagrams in Figure 5.5 with Figure 5.7, we observe that those phase diagrams across different depths are qualitatively similar, where the o.o.d. generalization only emerges with a large enough number of pre-training tasks. As model capacity decreases,

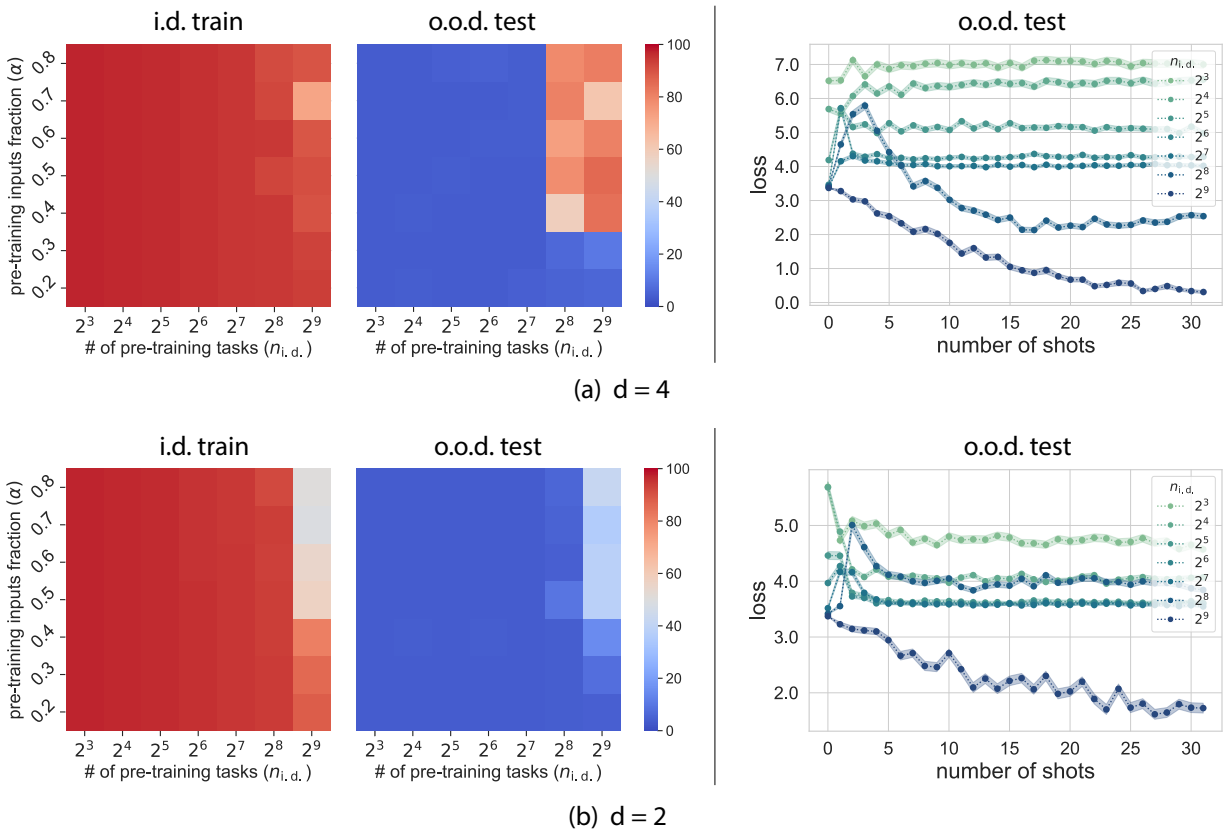


Figure 5.7: Phases of depth $d = 4$ and $d = 2$ models. With decreasing model capacity, the performance on both sets degrades. At the same time, the transient nature of ICL does not appear. **(a, b)** from left to right: accuracy phase diagrams on pre-training set $S^{\text{i.d.}}_{\text{train}}$ and on o.o.d. test set $S^{\text{o.o.d.}}_{\text{test}}$, with early stopping; loss and accuracy vs context length on o.o.d. test set $S^{\text{o.o.d.}}_{\text{test}}$ for $\alpha = 0.6$.

performance on both the pre-training set and the o.o.d. test set degrades. This is particularly evident in the $d = 2$ case, where the pre-training accuracy falls drastically as the model gains o.o.d. generalization ability.

Interesting observations can be made by comparing loss and accuracy on the o.o.d. test set as a function of context length at the end of training. First, it is evident that as the model depth decreases, the 1-shot loss surge attributable to memorization becomes milder. Notably, for models with $n_{\text{i.d.}} = 2^9$, there is no loss surge in the 1-shot case across all three depths. Furthermore, the $d = 4$ model with $n_{\text{i.d.}} = 2^8$ behaves significantly differently from the corresponding one with

$d = 2$ case, where the model fails to perform ICL for the o.o.d. test set. This is also distinct from the $d = 6$ case, where the model tends heavily toward memorization due to its excessive capacity. Instead, the $d = 4$ model manages to maintain a better balance between memorization and generalization at the end of pre-training. Consequently, the model has a 1-shot loss surge followed by a notable drop in ICL loss. This suggests that $d = 4$ optimally leverages the available model capacity to facilitate effective learning dynamics for o.o.d. generalization.

5.4 Interpreting the Learned Algorithms

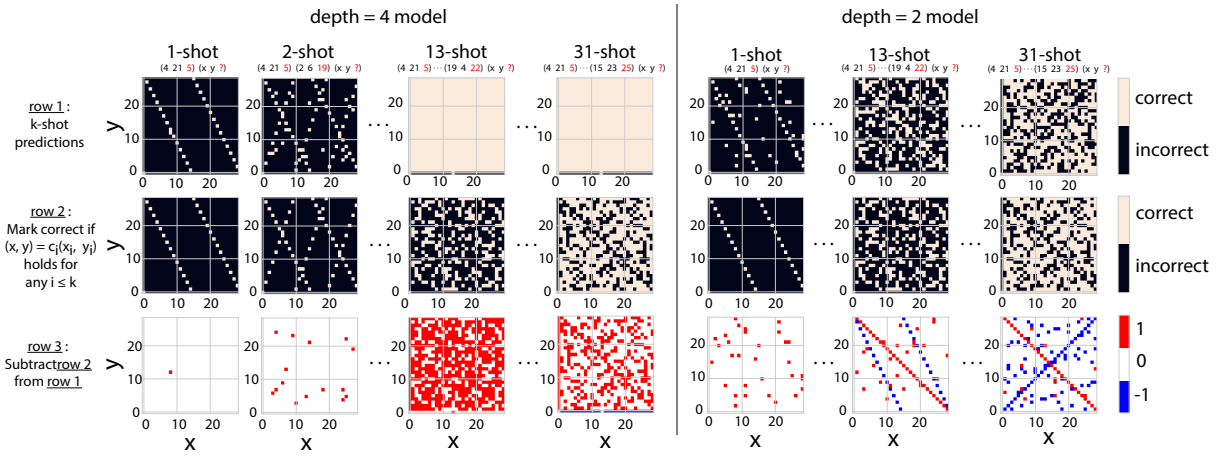


Figure 5.8: $d = 4$ and $d = 2$ models' performance on k-shot inference, on the grid of inputs $(x, y) \in \mathbb{Z}_p^2$ (task vector = $(6, 6)$). **row 1:** Models' predictions on o.o.d. task of the type $(x_1 \;\; y_1 \;\; z_1 \;\; \cdots \;\; x_k \;\; y_k \;\; z_k \;\; x \;\; y \;\; ?)$. **row 2:** Analytical plots showing predictions solely based on Modular Regression algorithm. **row 3:** Subtract row 2 from row 1, by using correct=1 and incorrect=0. The red points correspond to the examples where Ratio Matching does not give the correct predictions but the model predicts correctly. The blue points are examples that the model missed despite Ratio Matching being applicable. This row tells us about the model's ability to implement Modular Regression by *combining* the in-context examples. Note that $d = 4$ model readily learns to combine previous examples, while its $d = 2$ counterpart struggles due to its limited capacity.

We now explain the algorithms implemented by the models to achieve o.o.d generalization. We find that the models implement two distinct algorithms depending on the depth and the

number of in-context examples during inference.

Ratio Matching: If there exists i s.t. $c_i(x_i, y_i) = (x, y) \mod p$ then $z = c_i z_i^t \mod p$. This algorithm only works when $y/x = y_i/x_i \mod p$ holds for at least one of the in-context examples.

Modular Regression: Find $\{c_1, c_2, \ldots, c_k\}$ s.t. $\sum_{i=1}^{k} c_i(x_i, y_i) = (x, y) \mod p$. Then the prediction is $z = \sum_{i=1}^{k} c_i z_i \mod p$. This can be viewed as discretized circular regression over $\mathrm{GF}(p)$.

We find telling signatures of these algorithms upon analyzing model predictions with varying numbers of in-context examples (Figure 5.8). With very few in-context examples, the models implement the Ratio Matching algorithm. As a canonical example, consider the 1-shot scenario $(x_1 \;\; y_1 \;\; z_1 \;\; x \;\; y \;\; ?)$. In this case, Ratio Matching will only solve the task for inputs that obey $(x, y) = c_1(x_1, y_1) \mod p$ for some c_1. Indeed, this is exactly what we observe in Figure 5.8 for both $d = 2, 4$ models. With many (~ 10) in-context examples, $d = 2$ and $d = 4$ models implement different algorithms. The $d = 4$ model can *combine* the in-context examples using the Modular Regression algorithm, leading to near-perfect o.o.d. generalization. On the other hand, the $d = 2$ model still uses Ratio Matching and shows sub-optimal performance. We ascribe this to the limited capacity of the $d = 2$ models. In other words, the $d = 4$ models exhibits an *algorithmic shift* from Ratio Matching to Modular Regression as them number of in-context examples increases. Whereas, the $d = 2$ models shows no such shift.

To implement these algorithms, the model needs to perform linear operations over $\mathrm{GF}(p)$, which can be broken down into the following essential skills (in addition to copying information, which is readily implemented by attention heads).

I. *Modular Map*: Encode the tokens such that operations over $\mathrm{GF}(p)$ can be easily implemented

II. *Multiplication*: A necessary skill to rescale in-context examples in both algorithms

III. *Addition*: A necessary skill for combining in-context examples in Modular Regression algorithm

While both $d = 2, 4$ models learn skills I, II perfectly; the $d = 4$ model outperforms its $d = 2$ counterpart in skill III – which helps it *combining* the in-context examples.

In the remainder of this section, we show the special attention heads that implement skill I (Section 5.4.1) and MLPs that implement skills II, III (Section 5.4.2). We explicitly show the deterioration in this structures as pre-training task diversity ($n_{i.d.}$) decreases. We also elucidate the algorithmic shift in the $d = 4$ models as the number of in-context examples increases.

5.4.1 Attention Heads Implement Skill I

So far, we have presented "black-box" evidence suggesting that the model is implementing the proposed algorithms (Figure 5.8). Now, we turn to "white-box" interpretability, to identify the components within the transformer model that are responsible for the essential skills. In Figure 5.9(a, b), we analyze the important attention heads in the $d = 2$ model. Specifically, we compare the emergent structures in models trained with different pre-training task diversities.

In Figure 5.9(a), we show the attention head from layer 1 that implements skill I. In the top panel, we see that each query only pays attention to itself and the two preceding keys. This pattern likely stems from the fact that each example in the sequence contains three tokens x_i, y_i, z_i^t; and suggests that this head is mostly focused on the information within the example.

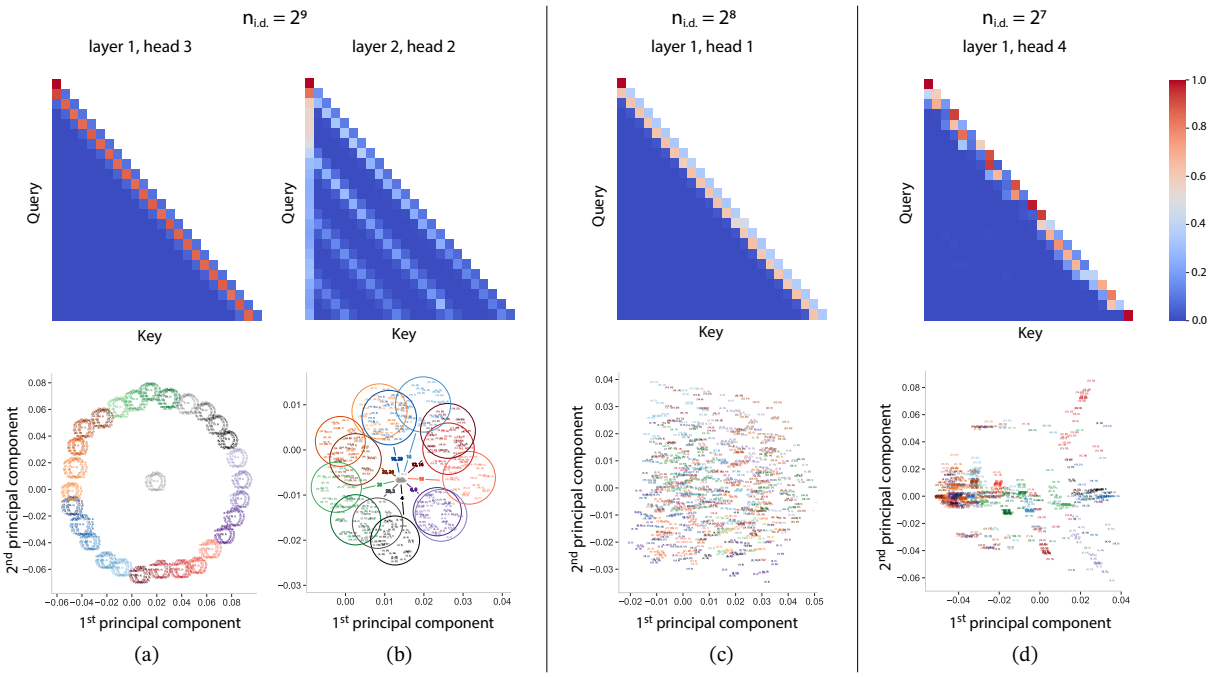


Figure 5.9: Models that generalize o.o.d. (left) exhibit more structured attention maps. Additionally, the top-2 principal components of the features from the corresponding heads also show more structured patterns. The features are computed for sequences with an o.o.d. task vector $(a^t, b^t) = (6, 6)$, loop over (x_i, y_i) at a specific shot while the previous parts of the sequence are fixed. We annotate each PCA plot with the $(\log_{27} x_i, \log_{27} y_i) \bmod p$ pairs. **(a)** The principal components form a circle of circles where the position of the outer circle is controlled by x_i. This pattern remains the same for different task vectors or the shot choices; **(b)** Only plotted pairs with even $\log_{27} x_i$, with each $\log_{27} x_i$ circled with different colors. The PCA pattern forms a similar double circle as those in (a), with the key difference that those circles depend on task vector choices and the shot choices; **(c, d)** Models without o.o.d. generalization ability. We pick heads from the first block that corresponds to the first column of (a). Clearly, the structure of attention maps and PCA patterns deteriorate as the task diversity decreases.

In the bottom panel, we perform principal component analysis (PCA) on the outputs of this head. Specifically, we feed a batched k-shot sequence of the form $(x_1 \;\; y_1 \;\; z_1 \;\; \cdots \;\; x_k \;\; y_k \;\; z_k \;\; x \;\; y \;\; z)$, where the first k inputs are fixed and the last input (x, y) is scanned over all p^2 possible pairs. We concatenate the resulting features from x and y, resulting in $p^2 \times 2d_{\text{embed}}$ batch of features – and perform PCA on this matrix. We project all these $2d_{\text{embed}}$ dimensional features onto the first two principal components. Annotating the pairs (x, y)

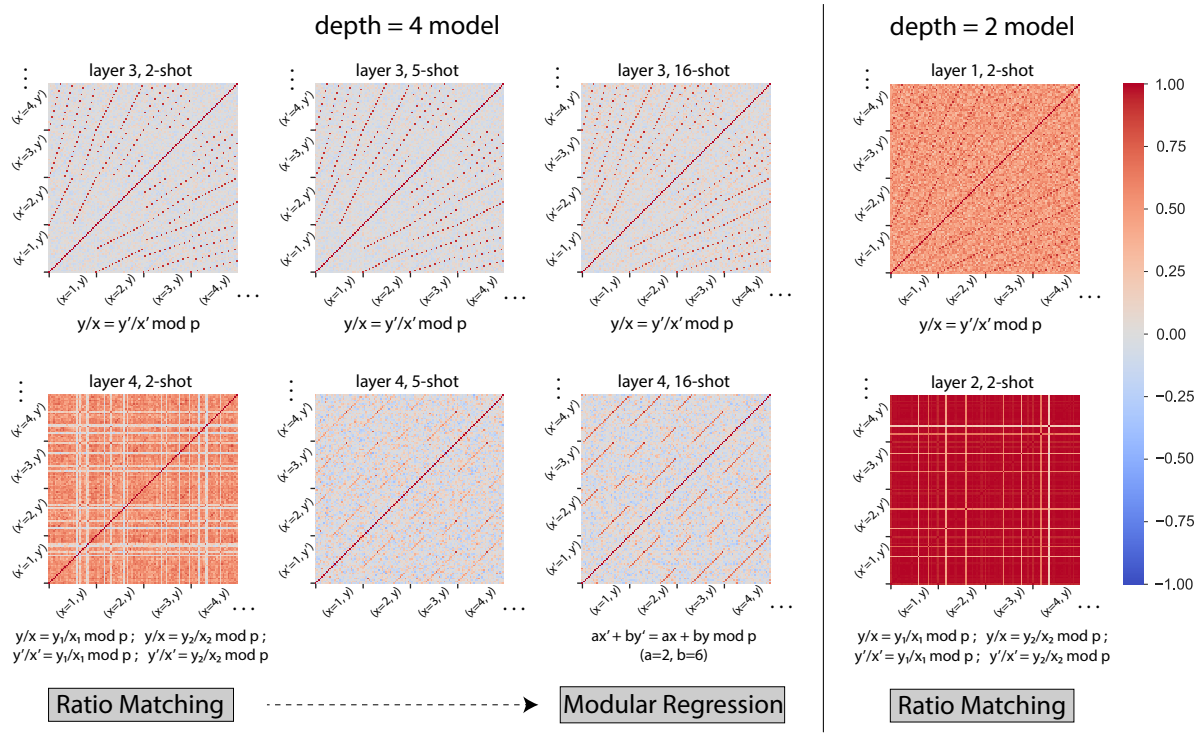


Figure 5.10: Cosine-similarities (Equation (5.2)) between layer outputs $\boldsymbol{h}^l$ at token z position ($\cos(\boldsymbol{h}^l_z(x,y), \boldsymbol{h}^l_z(x',y'))$, first row) and token y position ($\cos(\boldsymbol{h}^l_y(x,y), \boldsymbol{h}^l_y(x',y'))$, second row) reveal patterns in the models' internal representations. For clarity, we only show selected x and x' values, where y and y' values range from 0 to 28 between each tick. For the $d = 4$ model, kaleidoscopic patterns in the third layer indicate the generation of all possible y_i/x_i features for subsequent computations. The last layer shows an algorithmic shift from Ratio Matching to Modular Regression. The $d = 2$ model also shows the kaleidoscopic pattern in the first layer, while the second layer identifies the relevant y/x features from the in-context examples for matching.

with $(\log_{27} x, \log_{27} y)$[1], we find a "circle-of-circles" – where circles of period 28 are arranged in a bigger circle of period 28. Number 0 is located at the center of the circles[2]. We observe similar circle-of-circles for concatenated features from (x, z) and (y, z) as well. We refer the reader to Appendix D.5 for further details.

In Figure 5.9(b), we analyze the head from layer 2 that effectively "copies" in-context examples. The upper panel shows the highly structured attention map that focuses on the current as well as the preceding examples. This pattern aligns with the Ratio Matching algorithm where

[1]We use 27 as the base of logarithm, which is a primitive root of $\mathrm{GF}(29)$

[2]This is expected since $\log_{27} 0$ is mathematically ill-defined and needs special treatment.

the model compares (x, y) pairs across different in-context examples.

In the bottom panel, by conducting a PCA analysis, we again identify circles when annotating examples in the $(\log_{27} x, \log_{27} y)$ format. However, unlike the previously discussed pattern, the specifics of this "circle-of-circles" arrangement vary depending on the position and the choice of task vector (a^t, b^t). This variability suggests that the head in question possesses information about the specific task from the context.

We further highlight the importance of the structure we find in the above paragraphs via comparison with models, trained with lower pre-training task diversity, that do *not* generalize o.o.d (Figure 5.9 c, d). Note that as the number of pre-training tasks $n_{\text{i.d.}}$ decreases, the attention map starts to get mosaicked (top panels); and the PCA projections lose their shape (bottom panels). As a result, these models lose the ability to perform ICL on modular arithmetic out-of-distribution.

5.4.2 MLPs Implement Skill II and III

After applying Skill I, the model maps numbers onto circular feature spaces, enabling discrete modular operations such as multiplication, division, addition, and subtraction through subsequent model components. To analyze this behavior, we compute the cosine-similarity between layer l (i.e. l^{th} Transformer block) k-shot output vectors (standardized) at token position y: $\boldsymbol{h}^l_y(x_1, y_1, z_1, \cdots, x_k, y_k, z_k, x, y)$.

$$\cos(\boldsymbol{h}^l_y(x, y), \boldsymbol{h}^l_y(x', y')) = \frac{\boldsymbol{h}^l_y(\ldots, x_k, y_k, z_k, x, y) \cdot \boldsymbol{h}^l_y(\ldots, x_k, y_k, z'_k, x', y')}{\|\boldsymbol{h}^l_y(\ldots, x_k, y_k, z_k, x, y)\| \|\boldsymbol{h}^l_y(\ldots, x_k, y_k, z_k, x', y')\|} \tag{5.2}$$

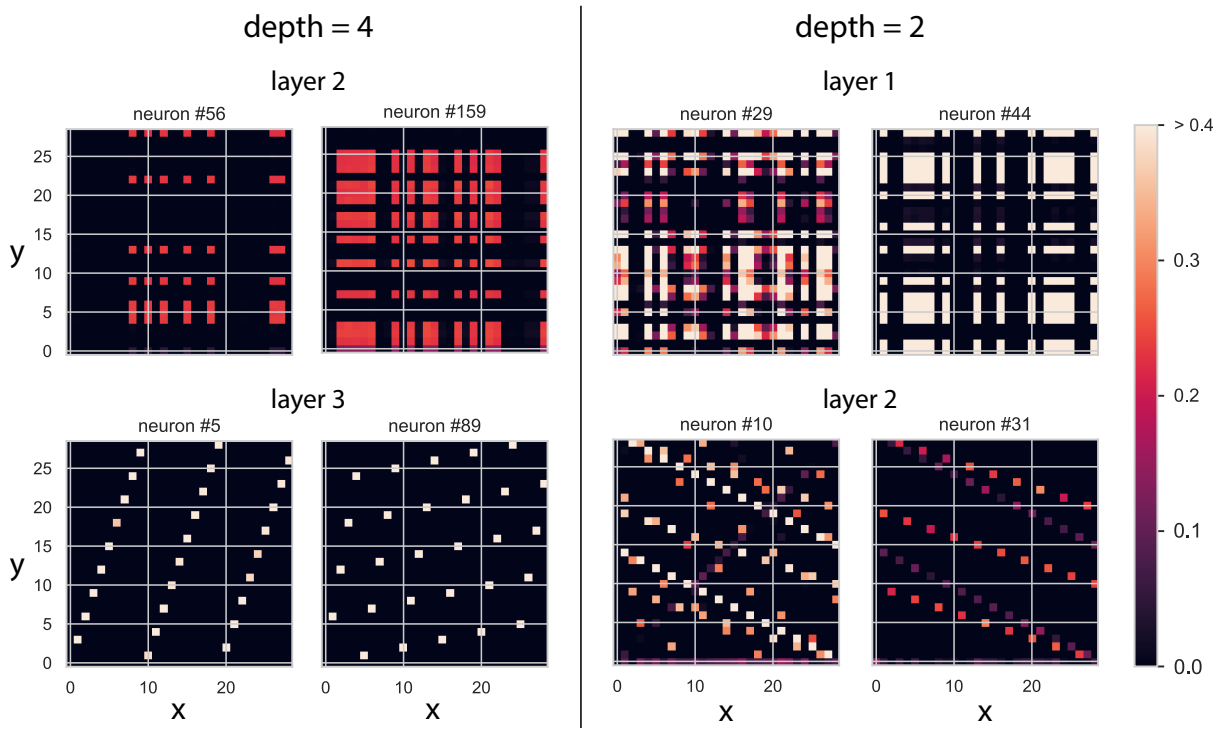


Figure 5.11: MLP activations (post-ReLU) with respect to different input (x, y) pairs.

We evaluate this similarity metric across all possible input pairs (x, y) and (x', y'), resulting in a $p^2 \times p^2$ matrix. We also repeat the analysis to layer output vectors at z tokens $\boldsymbol{h}_z^l(\cdot)$. In line with our previous analysis, we use controlled (fixed) in-context examples $(x_1\ y_1\ z_1\ \ldots\ x_k\ y_k\ z_k)$.

For the $d = 4$ model, the top panel of Figure 5.10 reveals a distinctive kaleidoscopic pattern of high cosine-similarities. The pattern corresponds to input pairs (x, y) that share the same ratio $y/x \bmod p$. In the MLP modules, we find highly structured activations as functions of inputs (x, y) (Figure 5.11 left). Together with our earlier analysis of Skill I, these observations suggest that the MLP layer following the attention module with a "circle-of-circles" head (layer 2, head 4 for $d = 4$ model, see Figure D.5 (a)) implements division operations over $\mathrm{GF}(p)$. The bottom panel of Figure 5.10 succinctly demonstrates the algorithmic shift from Ratio Matching to Modular Regression as the number of in-context examples increases. At 2-shot, layer 4 features collapse to identical vectors except when the ratio y/x matches one of the ratios (y_1/x_1 or y_2/x_2).

At higher shots, we see a transition to a task-dependent pattern where features align for input pairs (x, y) for which $a\,x + b\,y \bmod p$ match – a signature of Modular Regression.

For $d = 2$ models, we again observe the kaleidoscopic pattern in cosine-similarities (Figure 5.10 top-right) and structured MLP activations (Figure 5.11 right). However, in this case we only find signatures of Ratio Matching in the cosine similarities (Figure 5.10 bottom-right), as expected.

Thus, we conclude that the $d = 2$ model has scarcely acquired skill III, due to its limited capacity. On the other hand, $d = 4$ model is very good at combining equations via skill III, explaining its superior performance. For a more detailed discussion, see Appendix D.5.

5.5 Related Works

In-Context Learning (ICL). Brown et al. [16] first demonstrated that large models performance improves substantially when a few examples of the task at hand are provided at inference time, in the prompt. Ahn et al. [120], Akyürek et al. [124], von Oswald et al. [125] showed that the AM implements emergent first-order optimization on an emergent objective function to solve linear regression tasks. Furthermore, Akyürek et al. [124] showed that larger models learn to perform Bayesian estimation. Garg et al. [126] demonstrated that transformers can learn several simple classes of functions in context. Kirsch et al. [127] presented how task diversity and model size would affect the ICL performance for unseen tasks using a mixture of modified MNIST datasets. Raventos et al. [121] investigated the relation between task diversity and out-of-distribution ICL ability on linear regression tasks. Lin and Lee [128] identified two operating modes in ICL using a mixture of linear regression tasks, where for the first several shots, the

model tries to figure out the correct task vector and later uses it to predict the correct results. Boix-Adserà et al. [129] showed theoretically and experimentally that with enough pre-training data, a transformer model can perform abstract reasoning that a MLP cannot do. Guo et al. [130] showed that transformers can use lower layers to memorize and upper layers to perform ICL in a feature regression setting. It was found in [123] that ICL is a transient phase from the optimization point of view: it goes away once the model is over-trained. Hendel et al. [131], Liu et al. [132] showed that language models form in-context vectors, which can be used to steer model predictions.

Modular Arithmetic. Power et al. [29] discovered Grokking, where models trained on modular arithmetic datasets have an abrupt change from random guessing to generalization on the test set way after the model memorized the training set. Gromov [30], Nanda et al. [103], Gu et al. [133] showed that for modular addition tasks, models learned to map integers to Fourier features to solve modular arithmetic tasks. Liu et al. [101] showed that grokking is related to learning highly structural features, and the grokking transition can be explained by a toy model. Zhong et al. [104] showed that there is more than one algorithm that a model can implement to solve modular addition. Doshi et al. [4] showed that corruption of the label does not prevent the models from finding a generalizing solution. Doshi et al. [5] showed that MLP and transformer models can solve a specific family of modular polynomial tasks by bijectively mapping them to modular addition tasks.

Interpretability. Elhage et al. [134], Olsson et al. [135] showed that transformers can form induction heads that predict the next token in a sequence by identifying and copying patterns

from earlier in the sequence. With several indirect empirical evidences, they showed that those heads might constitute the core mechanism of ICL. Nichani et al. [136] showed theoretically and empirically how disentangled transformers learn causal structures from in-context Markov chains by forming induction heads.

5.6 Discussion

We have investigated the emergence of in-context learning and skill composition in autoregressive models on a novel algorithmic dataset. The dataset includes a large discrete set of modular arithmetic tasks and is specifically designed to force models to learn how to solve a variety of tasks. It consists of learning linear modular functions, where the model is expected to identify *and* perform a modular operation in-context. When the number of tasks is relatively small the models can only generalize in-distribution. Although such models develop ICL capabilities, they simply memorize these task vectors and use them to classify the input vectors. Once the number of training tasks becomes too large, the models transition to a qualitatively different algorithmic approach, where the task vector is determined at inference time.

Finally, we have examined the learnt representations and shown that qualitatively different circuits are formed in different phases. In the o.o.d. generalization phase, we explain the learnt algorithms and highlight an in-context algorithmic shift in deeper models.

Limitations We have limited this work to algorithmic datasets. It remains to be investigated what lessons can be translated to realistic language models, and what lessons are specific to the current setting. White-box interpretability analysis of deeper models has proved to be much more difficult than that of shallower models. Consequently, we still do not understand the role of every

individual component of the network in the deeper cases.

More importantly, this way of crafting an dataset and interpret it via human efforts are not scalable and will not work for large scale models. Thus it would be necessary to consider building more generic algorithms for thinking about interpretability for real-world LLMs.

Chapter 6: Transformers and Pseudo-Random Number Generators

In the previous chapter, we explored how emergent capabilities arise in modular arithmetic settings when models possess sufficient capacity and are provided with adequate training data and in-context examples. In this chapter, we extend that understanding to a closely related yet practically significant domain – pseudo-random number generators (PRNGs).

PRNGs are fundamental to modern cryptography and computer science. Although they are designed to produce outputs that pass statistical tests for randomness, their sequences ultimately arise from deterministic mathematical patterns that, in principle, can be learned by sufficiently powerful models.

Here, we deepen our investigation into modular arithmetic tasks under different underlying distributions, aiming to understand whether models can learn to "crack" PRNGs. Furthermore, with an increased complexity of data, we seek to identify the conditions under which such capabilities emerge, as well as to analyze the specific structures and patterns captured by the model when it succeeds in learning the behavior of these generators.

6.1 Linear Congruential Generators

Here we first introduce the Linear Congruential Generator (LCG) [137, 138]. LCG is a simple PRNG that generates the next number in a sequence $(x_0, x_1, \ldots, x_t)$ according to the

map:

$$x_{t+1} = ax_t + c \mod m\,, \tag{6.1}$$

where $m > 0$ is the modulus, $0 < a < m$ is the multiplier and $0 \leq c < m$ is referred to as the increment. An LCG map is uniquely defined by the choice of m, a, c and the initial seed x_0. An important quantity that determines the complexity of an LCG sequence is its period: $1 \leq \mathcal{T}_m(a, c) \leq m$. As we will show in the following sections, the period of a sequence plays a major role in the difficulty of prediction with Transformers. According to the Hull-Dobell Theorem [139], the period $\mathcal{T}_m(a, c) = m$ if and only if the values of a and c satisfy the following criteria: (i) m and c are coprime, (ii) $a - 1$ is divisible by all prime factors of m, (iii) $a - 1$ is divisible by 4 if m is divisible by 4. We evaluate (test) all our models exclusively on sequences that obey the criteria of this theorem.

LCGs are widely utilized for their speed and simplicity, often forming the core of more complex PRNGs like PCG-64, which is used in NumPy. LCGs perform poorly at small bit sizes but improve rapidly with larger state sizes. For instance, an LCG with 88 bits of state can pass the stringent BigCrush randomness test [140].

6.2 Training Setup

We train decoder-only Transformers to autoregressively predict the next number in LCG sequences. This means it takes as input an LCG sequence $(x_0, \cdots, x_{L-1})$, outputs a sequence $(y_0, \cdots, y_{L-1})$, and trained so y_t matches x_{t+1}. Here L is the maximum context length we used throughout this Chapter.

To predict an unknown LCG sequence, the Transformer needs to infer m, a, and c in-context. We test the model's generalization ability in two distinct paradigms of increasing difficulty: **`FM:`** The model is trained and tested on sequences with a fixed modulus m. **`UM:`** Model is trained on varying moduli, and tested on unseen moduli $M_{\text{test}} = \{m_{\text{test}}\}$.

6.2.1 Dataset Generation and Evaluation

We outline the experimental setup used in Section 6.3 and Section 6.4 here. We used a modified setup when we scale up the modulus in Section 6.5, which we will describe later.

Fixed Modulus (`FM`): Given a modulus m, we apply the Hull-Dobell Theorem to determine the possible values of (a, c) that maximize the period. We then randomly select 64 values of a and c to generate the test dataset. To generate the training dataset, we *exlude* these test choices of (a, c) and uniformly sample $N = 100,000$ LCG sequences of length $L + 1$ (where L is the context length), with n_a values of multipliers and n_c values of increments. For each set of parameters (a, c), we sample an LCG sequence with a randomly selected initial seed x_0. Note that the training dataset includes sequences with varying periods, while the test data only contains sequences that maximize the period.

Generalization to Unseen Modulus (`UM`): In this more challenging paradigm, we first select a set of test moduli $M_{\text{test}} = \{m_{\text{test}}\}$ that would be reserved exclusively for evaluation. For each test modulus $m_{\text{test}} \in M_{\text{test}}$, we determine the values of (a, c) that maximize the period. We then randomly select 64 values of a and c each to generate the test dataset. These 64^2 (a, c) pairs are not considered while generating the training dataset.

For the training dataset generation, we sample n_m modulus values from the range $[L, m_{\text{max}}]$, with $m_{\text{max}} = \lfloor 1.2 \max(M_{\text{test}}) \rfloor$, while excluding all the values in M_{test}. For each modulus value m, we uniformly select n_a multipliers and n_c increments, excluding the ones reserved for testing. For each triplet (a, c, m), we generate a sequence of length $L+1$ using a randomly selected initial seed x_0. This results in a total of $N = n_m \times n_a \times n_c$ training sequences.

We found that $n_m \gtrsim m_{\text{test}}/4$ yields good generalization performance. Based on this relationship and our target total number of training examples N, we sample $n_a = n_c = \sqrt{\frac{N}{m_{\text{test}}/4}}$ values of multipliers and increments. Unless explicitly specified, we use this setting as the default configuration for all experiments.

In both paradigms, test accuracy is averaged over all (a, c) test pairs and multiple initial seeds x_0 per pair. Accuracies are tracked at all sequence positions $1 \leq t \leq L$.

6.2.2 Tokenization, Architecture and Optimizer

In Sections 6.3 and 6.4, each number is tokenized as a unique token, using a dictionary of size m (**FM**) or m_{max} (**UM**). We employ GPT-style Transformers with learnable positional embeddings and weight tying [141].

When we scale up to larger moduli in Section 6.5, we restrict the dictionary size to b by tokenizing each number in the sequence in base-b (e.g. $b = 2^8, 3^5$). This results in $\lceil \log_b m \rceil$ tokens for each number. We also apply (modified) abacus positional embeddings [142].

The model architecture is characterized by the number of blocks (depth), embedding dimension (d_{model}), number of attention heads (n_{heads}). Models are trained with AdamW [97] and CrossEntropy loss.

6.3 Training Results

We begin by investigating the minimal model that can solve the two tasks in consideration. Surprisingly, we found that Transformers only require one layer and one attention head to learn the **FM** task, as shown in Figure 6.1 (a) (for further results, see Appendix E.3).

Conversely, the **UM** task requires a stronger architecture and careful hyperparameter tuning. Appendix E.5 shows that model performance depends on the modulus, with prime moduli being challenging in the **FM** setting but not in the **UM** setting.

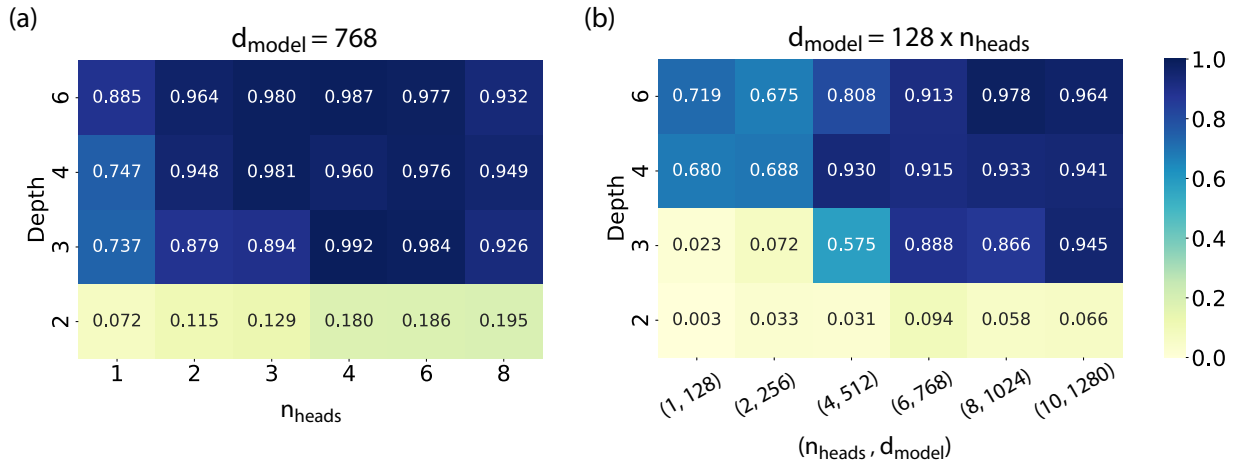


Figure 6.1: Accuracy of predicting the last number (token) in the sequence: phase diagrams w.r.t. various depths and n_{heads} values. (a) $m_{\text{test}} = 2048$, constant width: $d_{\text{model}} = 768$. (b) $m_{\text{test}} = 4096$, width scaled proportionally: $d_{\text{model}} = 128 \times n_{\text{heads}}$.

In Figure 6.1 we show how the performance varies with model depth and the number of attention heads. In Figure 6.1(a) we keep the embedding dimension fixed to $d_{\text{model}} = 768$, whereas in Figure 6.1(b) we scale it proportionally to the number of heads ($d_{\text{model}} = 128 \times n_{\text{heads}}$.). In both cases, we find that a minimum of three layers are required for effective generalization, with performance degrading sharply below this threshold.

Further analysis across multiple m_{test} values (see Appendix E.4) confirms that this minimal

depth requirement is universal. We also observe that additional attention heads improve model performance, with a substantial gain occurring when increasing from one to two heads.

Several prior studies have also observed sharp changes in model capabilities as a function of model depth. This includes induction head formation [143], in-context learning of modular addition [6] and various in-context generalization tasks [144]. In general it is unclear to what extent these sharp depth-dependences are due to jumps in expressivity or trainability.

The **UM** task shows strong sensitivity to hyperparameters. As we increase m_{test} while keeping the model size fixed, we observe two key phenomena: the optimal learning rate (η) and weight decay strength (λ) shift significantly, and simultaneously, the range of hyperparameter resulting in effective performance narrows (see Appendix E.1.5).

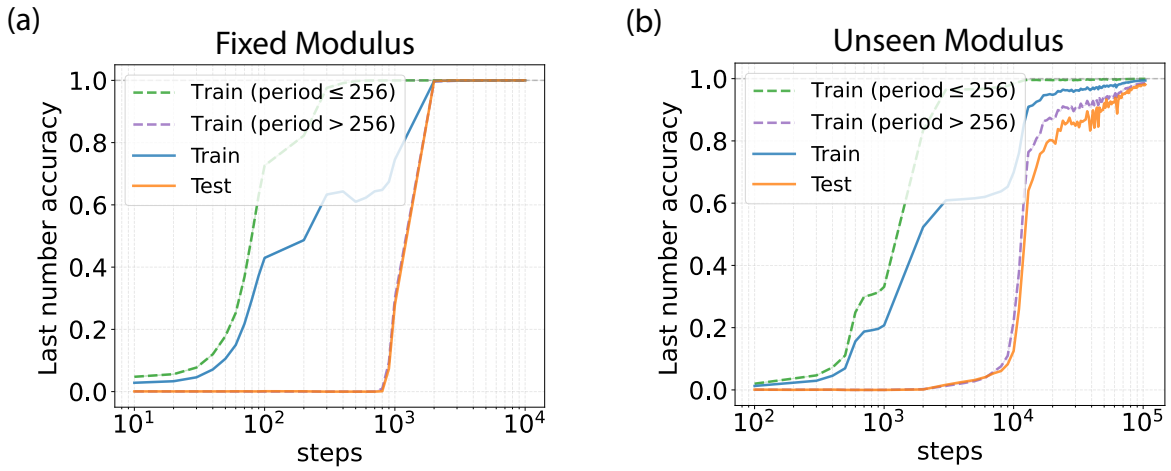


Figure 6.2: Training/test accuracy curves for predicting last number (token). (a) **FM:** ($m = 2048, \text{depth} = 1, n_{\text{heads}} = 1, d_{\text{model}} = 768$) Test accuracy "groks" when training accuracy reaches near-100%. (b) **UM:** ($m_{\text{test}=2048}, \text{depth} = 6, n_{\text{heads}} = 4, d_{\text{model}} = 768$) Test accuracy "groks" simultaneously with training accuracy on sequences with period longer than context length ($\mathcal{T}_m > L = 256$), indicating delayed discovery of underlying rules.

We then carefully examine the training dynamics in both **FM** and **UM** settings (Figure 6.2).

We categorize the training sequences into two groups: i) sequences with periods shorter than the context length, which can be solved through simple copying, and ii) sequences with

periods longer than the context length, which require the model to deduce underlying rules for prediction. Our analysis reveals that the model first acquires copying ability for group i) in the early stages of training, and later "groks" the solution for group ii) [29]. Notably, the model's ability to generalize to test modulus m_{test} emerges simultaneously with this grokking phenomenon. These results demonstrate that the model develops different capabilities at distinct stages of training, with generalization ability emerging only after the model learns the underlying rules through solving the more challenging sequences.

In Appendix E.6, we present an ablation study where models are trained exclusively on either short-period or long-period sequences. Our findings indicate that training exclusively on long-period sequences enables model generalization.

6.4 Interpreting How Transformers Predict PRNGs

In this section, we uncover the underlying algorithms implemented by the models for both **`FM`** and **`UM`** cases. While certain details of the algorithms differ in the two cases, they share common properties originating from the underlying LCG structure.

We first discuss properties of LCG sequences that will be useful in interpreting model behaviors.

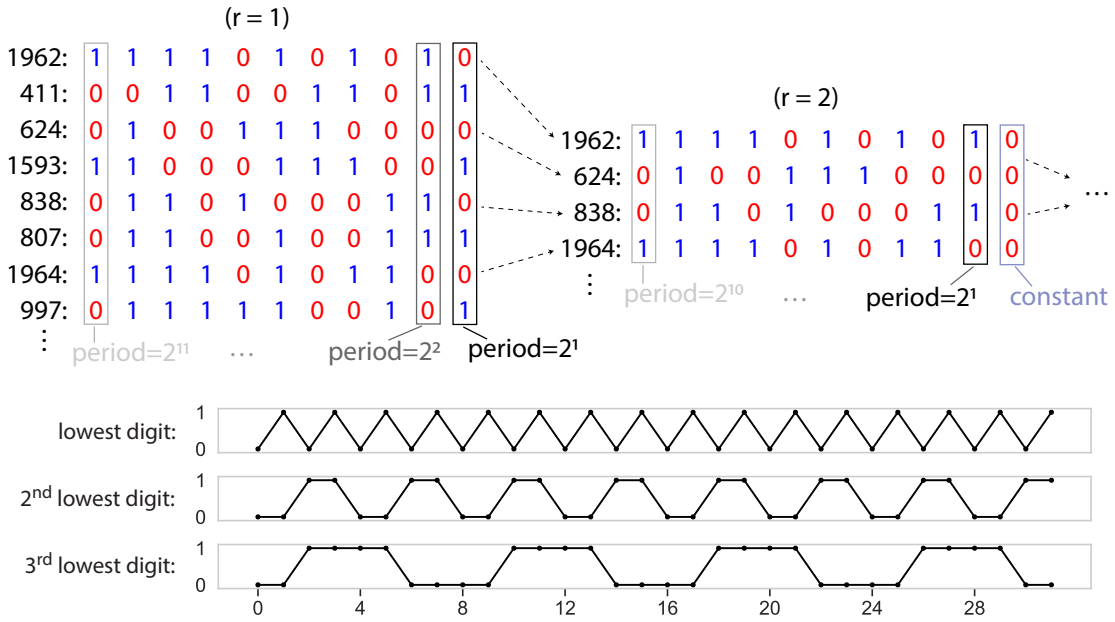


Figure 6.3: Bit-wise periods in an example LCG sequence generated with $m = 2048, a = 293, c = 1033$, which follows the Hull-Dobell theorem. In binary representation w-th lowest bit has a period of 2^w, for $w \in \{1, \dots 11\}$. Writing a new sequence by skipping every 2nd step ($r = 2$) reduces the periods of all the bits by a factor of 2, rendering the lowest bit constant. $r = 2^k$ reduces bit-wise periods by a factor of 2^k, with last k digits constant.

6.4.1 Residual Number System Representations

Consider an LCG sequence with modulus $m = 2048 = 2^{11}$. Each number in this sequence can be represented as an 11-digit binary number:

$$x \bmod 2^{11} = \alpha_0\, 2^0 + \alpha_1\, 2^1 + \cdots + \alpha_{10}\, 2^{10}\ , \tag{6.2}$$

where $\{\alpha_0, \dots, \alpha_{10}\}$ are the binary-valued digits (bits).

A useful property of LCGs with modulus $m = 2^{(\cdot)}$ is that each digit in the binary representation has a fixed period along the sequence. As shown in Figure 6.3, for a sequence of period $\mathcal{T}_m = m = 2^{11}$, the w^{th} lowest digit has a period of 2^w [145]. Thus, lower (higher) digits have smaller (larger) periods along LCG sequences. (See Appendix E.2 for a detailed deriva-

tion.) We will see later that trained Transformers have emergent structures that find these binary representations and utilize them to make systematic predictions from the context. Notably, the per-digit period plays and important role in the prediction accuracy of that digit. To understand this, consider the r-step iteration of Equation (6.1):

$$x_{t+r} = a^r x_t + \sum_{i=1}^{r} a^{i-1} c \mod m \,. \tag{6.3}$$

In this new sequence wherein we skip r steps, the period of each digit α_{w-1} reduces from 2^w to $2^w / \gcd(r, 2^w)$. Consequently, the higher digits become relatively simpler to predict due to reduced periods while some lower digits become trivial to predict due to being constant along this new sequence. We demonstrate this for $m = 2048$ in Figure 6.3 (top panel). The digit-wise periods in the new sequence with $r = 2$ are reduced by a factor of 2, while the last digit is simply constant. Higher values of $r = 2^k$ will lead to even further simplifications of the sequence. Transformers can simplify the task of predicting LCG sequences by utilizing r-step iterations from the in-context examples – with longer contexts leading to larger values of r. Consequently, the per-digit and overall accuracies improve substantially with context (see Figure 6.4).

While moduli of the form $m = 2^{(\cdot)}$ lead Transformers to find binary representations, similar simplifications in composite moduli require more general representations of the *Residual Number System (RNS)* [146]. RNS represents numbers by their values modulo pairwise coprime factorizations of m. Specifically, consider sequences with a composite modulus m, which has a prime factorization $m = p_1^{w_1} p_2^{w_2} \cdots p_q^{w_q}$. In this case, we can uniquely represent each number $(x \bmod m)$ as the tuple of residuals $(x \bmod p_1^{w_1}, x \bmod p_2^{w_2}, \ldots, x \bmod p_q^{w_q})$. Analogous to Equation (6.2), we can further decompose each residual,

$$x \bmod p_j^{w_j} = \alpha_{j,0}\, p_j^0 + \alpha_{j,1}\, p_j^1 + \cdots + \alpha_{j,w_j-1}\, p_j^{w_j-1} \tag{6.4}$$

where $\alpha_{j,w} \in \{0, 1, \ldots, p_j - 1\}$ are base-p_j digits. We refer to $\{\alpha_{j,w}\}$ as the "RNS representation" in the remainder of the text. When $\mathcal{T}_m = m$, each digit $\alpha_{j,w}$ has a period of p_j^w (derivation in Appendix E.2.2). The r step iteration Equation (6.3) reduces the period of each digit $\alpha_{j,w}$ from p_j^w to $p_j^w / \gcd(r, p_j^w)$. This results in simplification of the prediction task whenever $r = p_1^{k_1} p_2^{k_2} \cdots p_q^{k_q}$. We will see that identifying the RNS representations is a key simplification that the Transformer discovers in learning LCG sequences.

6.4.2 Fixed Modulus

Qualitative Algorithm (fixed modulus):

i. Find RNS representations of inputs from the learned prime factorization of m

ii. Look back $r = p_j^k$ steps in the context and copy the lowest k digits, for different prime factors (p_j) of m

iii. Using these r-step iterations, predict the higher digits of the simplified sequence

We now discuss the algorithm implemented by Transformers trained on LCG with a fixed modulus. We will show that, despite being provided integers as inputs, the model develops emergent structures that create and leverage the RNS representation of the inputs. We will focus on the setup with a 1-layer, single-head Transformer trained on LCG with $m = 2048$. Similar results

for composite moduli (e.g. $m = 7776$) and different model sizes are presented in Appendix E.7. We emphasize that this algorithm works for arbitrary a, c, x_0 for a given m.

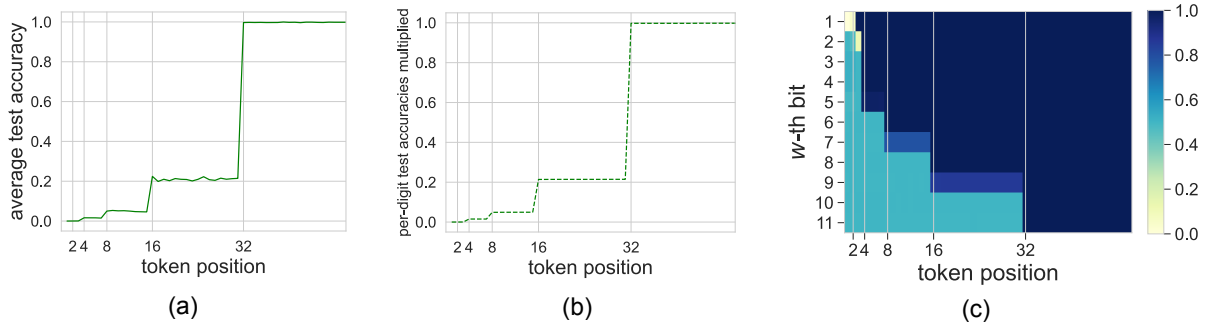


Figure 6.4: **FM:** Test accuracy for $m = 2048$, depth $= 1$, $n_{\text{heads}} = 1$, $d_{\text{model}} = 768$, averaged over a, c, and seeds. (a) Test accuracy w.r.t. token positions. Ladder-like structure appears, with jumps occurring at 2^k-th positions. (b) We represent numbers as an eleven-digit binary number ($2048 = 2^{11}$) and compute the per-digit test accuracy of model predictions.

We begin by analyzing the average accuracy of a trained Transformer model as a function of token position along the context, shown in Figure 6.4 (a). (Recall that in this section a token corresponds to an integer of the LCG sequence). The accuracy exhibits a ladder-like structure, with each jump occurring exactly at the 2^k-th token position. These successive transitions can be explained using binary representations and r-step recurrence (Equation (6.3)). Specifically, to predict the token at position $t \geq 2^k$, the model can look back in the context at position $t - 2^k$ and implement $r = 2^k$-step iteration. This allows the model to (i) copy the lowest k bits since they remain unchanged; and (ii) simplify the higher bits, since their periods get reduced by a factor 2^k. We note that the accuracy trend remains unchanged across different choices of a and x_0 (see Figure E.9).

Next, in Figure 6.4(b), we compute the per-digit accuracy by converting both model predictions and ground truth labels to their binary representations according to Equation (6.2). We observe that the model can predict more digits correctly with more context, with sharp transitions

occurring at 2^k-th token positions. This is a direct result of the sequences becoming increasingly more simplified as the model can look farther back in the context and utilize $(r = 2^k)$-step iterations. Since these simplifications occur in the form of digit-wise periods, Figure 6.4(b) serves as direct evidence that the model is internally developing and utilizing binary representations. The sequential learning of digits also explains the ladder-like structure of the accuracy in Figure 6.4(a). We find that the overall average accuracy (Figure 6.4(a)) multiplicatively depends on the per-digit accuracies (Figure 6.4(b)) (empirical proof in Figure E.10(b)) [1]

$$\text{acc}_{\text{overall}} = (\text{acc}_{\text{digit 1}})\,(\text{acc}_{\text{digit 2}})\,\cdots\,(\text{acc}_{\text{digit 11}})\,. \tag{6.5}$$

Next, we investigate how various components of the model implement the algorithm outlined earlier this section.

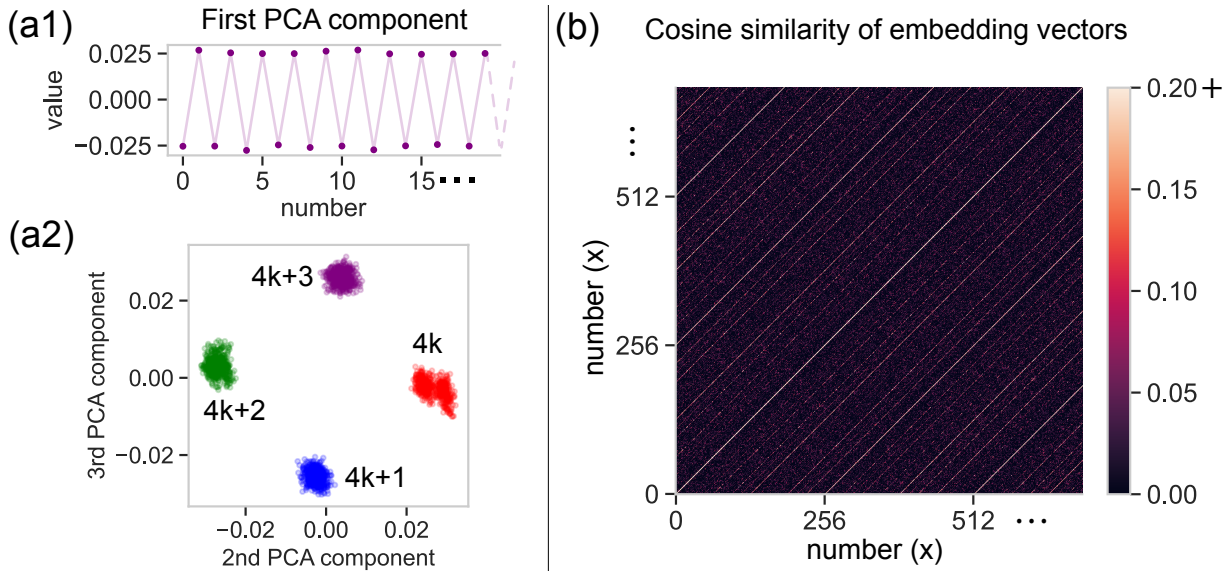


Figure 6.5: **FM:** $(a = 1589, c = 629)$ Embedding layer. (a1) 1st principal component groups the numbers mod 2. (a2) 2nd and 3rd principal components group numbers mod 4. (b) Embedding vectors of different numbers exhibit high cosine similarity when they are spaced 2^k apart, with the similarity increasing with k.

[1] This holds if the per-digit accuracies are independent, which we confirm empirically.

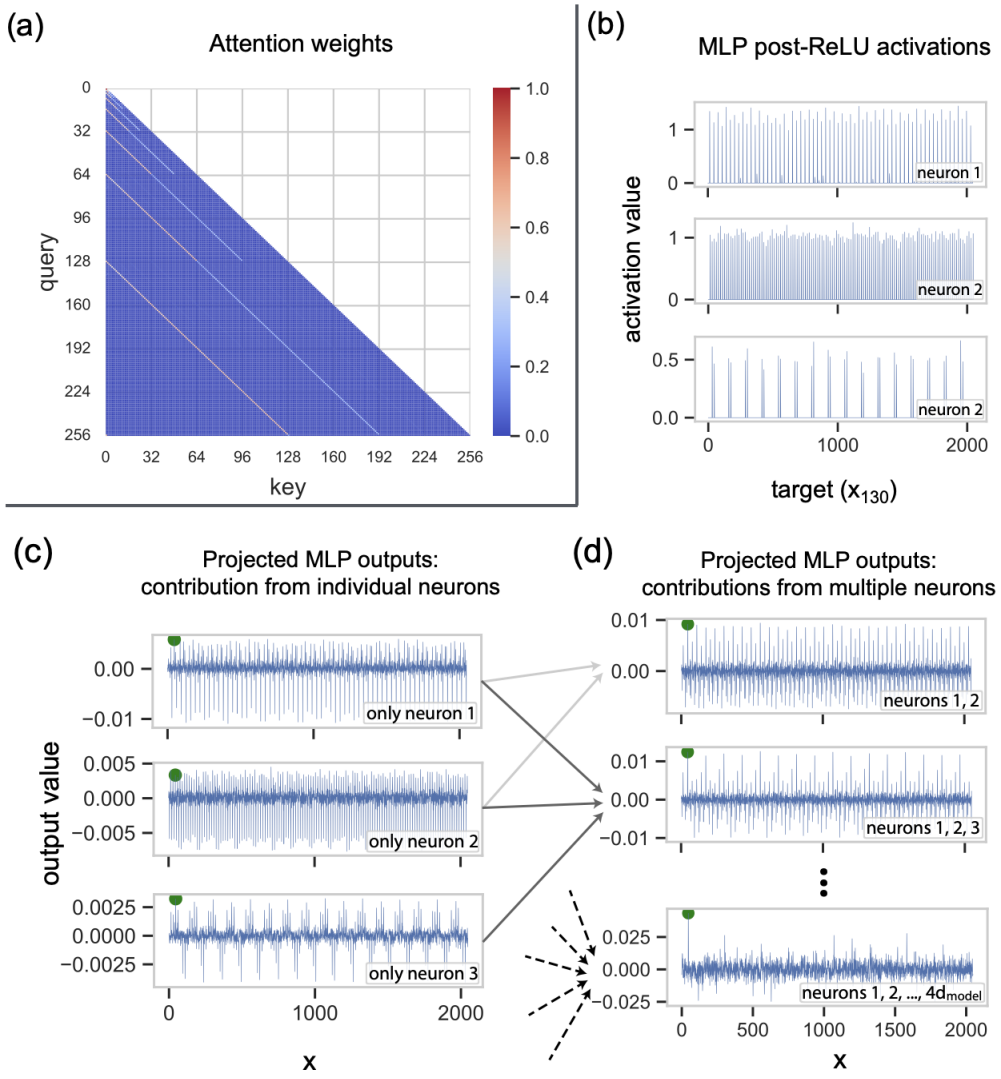


Figure 6.6: **FM:** ($m = 2^{11}, a = 1589, c = 629$) (a) Attention weights: each query attends most strongly to the tokens 2^k and 2^{k-1} distance backward, for the highest possible value of k, enabling copying of lowest k bits. The other faint lines facilitate the prediction of higher bits. (b) Post-ReLU hidden layer MLP activations at token position $t = 129$ (extracted using sequences with different x_0) as a function of the target number x_{130} which it is supposed to predict. Each neuron gets activated only while predicting a specific x_{130}, exhibiting a sparse, periodic pattern. (c) Output of the MLP block projected onto the (un)embedding matrix; after masking out all but a single given hidden-layer neuron. The green dot denotes the value at the target number. Each neuron resolves the correct prediction up to a periodic structure. (d) Output of the MLP block projected onto the (un)embedding matrix; after combining the signal from multiple neurons (i.e. gradually un-masking the neurons). The per-neuron periodic patterns constructively interfere at the correct output.

Step i: We begin by conducting Principal Components Analysis (PCA) of the embedding matrix, which shows how the model performs prime factorization to develop the binary representations (RNS for general m). Figure 6.5(a1) shows the projections of all numbers $x \in \{0, \dots 2047\}$ along the first principal component of the embedding. We observe that the model groups the numbers into modulo 2 clusters along the first principal component. Similarly, the 2nd and 3rd principal components group the numbers into modulo 4 clusters (Figure 6.5(a2)). In general, we find principal directions that group the numbers into modulo $2^{(\cdot)}$ (see Figure E.12). By clustering the numbers according to their remainder modulo different prime-powers, these principal directions naturally encode the digit-wise representations of the inputs. In Figure 6.5(b), we check the cosine similarity between the embedding vectors of different numbers. We see that the more digits two numbers share in the binary representation, the higher the cosine similarity of their embedding. This is a consequence of these numbers having similar components along principal directions corresponding to those digits.

Step ii, iii: In Figure 6.6(a) we examine the attention weights and find that to predict the number at position t, the model attends most strongly to the position $t - 2^k$ for the highest possible value of k s.t. $t \geq 2^k$ (i.e. $k = \lfloor \log_2 t \rfloor$). This corresponds to the brightest line in Figure 6.6(a). Using the binary representation of the $(t - 2^k)$-th token, the model can copy the lowest k bits and simplify the prediction of higher bits. Additionally, the second brightest line appeares at the position $t - 2^{k-1}$, along with other faint lines at intermediate distances (multiples of $t - 2^{k'}$ for $k' < k - 1$). The information obtained from all these lines are utilized by the model in predicting the higher bits.

To verify that the brightest line enables the copying of lower bits and that the second bright-

est line facilitates the prediction of higher bits, we performed the following two experiments. (i) For each query in the attention head, we mask out all keys except for the one at position $t - 2^k$. We then measure the performance of this ablated model, shown in Figure E.14(a). We observe that the model retains the ability to copy the k lowest bits, but loses the ability to predict the higher bits. (ii) Next, we repeat the above ablation experiment while masking out all keys except the ones at positions $t - 2^k$ and $t - 2^{k-1}$, shown in Figure E.14(b). This results in a drastic improvement in the prediction of the higher bits compared to (i), confirming our assertions.

After the attention layer collects information about previous tokens, the MLP block[2] processes the information to make predictions. We find that each hidden neuron (post-ReLU) in the MLP exhibits a periodic response with a distinct period, as a function of the target prediction. Consider the MLP at token position t. The input to the transformer at this position is x_t and the output y_t, should match the next number in the sequence x_{t+1}. In Figure 6.6(b) we show the activation value of selected neurons at token position $t = 129$ as a function of the target x_{130}, for m different sequences obtained by changing the seed x_0 for a given LCG sequence. We observe that only a sparse set of neurons are activated for a given target x_{130} and that there is a strong spiked periodic structure in the response of each neuron as a function of x_{130}, with neuron-specific frequencies.

The subsequent fully connected layer in the MLP block aggregates the contributions from all the activated neurons to make the correct prediction [147]. To visualize the contributions from an individual neuron, we mask out the contribution from all other neurons and extract the MLP output for a fixed input sequence. We then project this output onto the (un)embedding matrix,

[2]MLP block: fully connected $\rightarrow$ ReLU $\rightarrow$ fully connected.

which shows us the contribution of that single neuron in making the correct prediction.[3] In Figure 6.6(c) we show that each neuron resolves the target number x_{130} up to a distinct periodic pattern. The periodic patterns from different neurons constructively interfere at the target. In Figure 6.6(d), we observe that gradually adding contributions from multiple neurons resolves the correct output with increasing accuracy.

6.4.3 Unseen Modulus

Qualitative Algorithm (unseen modulus):

i. Encode information about various possible prime factorizations

ii. Estimate the modulus via the largest number in context

iii. Combine steps i and ii to construct correct RNS representations, then implement steps ii and iii from the fixed modulus algorithm

Unlike the **FM** case, the training set for **UM** is generated using many different moduli m, unseen at test time. Since each modulus has its own RNS representation incompatible with other moduli, the model must implement a different, more general algorithm to solve **UM** tasks.

We analyze how the embedding layer and attention heads of a 4-layer Transformer model help implement the above algorithm to solve the **UM** task. The model is trained on a dataset generated with $n_m = n_a = n_c = 128$, where $m_{\max} = 2457$. To avoid leakage between training and test sets, we specifically exclude the moduli $m_{\text{test}} \in \{1800 = 2^3 \cdot 3^2 \cdot 5^2, 2048 = 2^{11}, 2352 = 2^4 \cdot 3 \cdot 7^2\}$ from the training set. Note that the choice $m_{\max} = \lfloor 1.2 \cdot 2048 \rfloor$ is made to maintain

[3] We read-out MLP outputs instead of network outputs to avoid distortion from the skip connection.

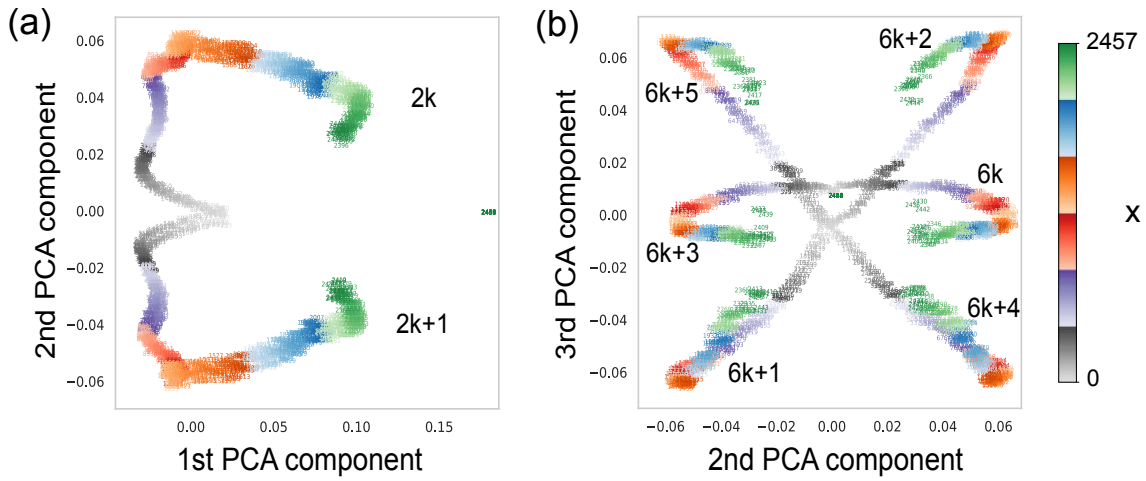


Figure 6.7: **UM**: PCA analysis of the embedding layer.

consistency with the setting in Figure 6.1(a).

Step i: We first analyze the embedding layer. In Figure 6.7(a), PCA shows a semi-circular structure along the first principal component, with the second principal component separating even and odd numbers. This semi-circle resembles the circular patterns seen in modular arithmetic tasks [29, 104], but since our model is trained on multiple moduli, it cannot form a closed circle by identifying a unique 0 value. Figure 6.7(b) further shows that the 2nd and 3rd principal components group numbers by their remainders modulo 2 and 3, which likely reflects their prevalence as prime factors in the training set.

Note with varying moduli, the model need not form a binary encoding as before. Instead, we find attention heads in first layer group embedded numbers by remainders modulo different primes, each head specializing in a particular factor. This specialization allows the model to construct RNS representations with various prime bases.

To investigate head specialization, we analyze affordmentioned first-layer attention heads in Figure 6.8, presenting PCA results for selected heads in panels (a1, b1, c1). For each, we

input sequences using the corresponding m_{test} and randomly selected (a, c) pairs (per the Hull-Dobell Theorem), then isolate the output $\boldsymbol{H}^{(h)}$ for each head by zeroing out all other heads. We perform PCA at token position $t = 0$ ($\texttt{PCA}(\boldsymbol{H}^{(h)}[:, 0, :])$), then projecting each number's feature vector $\boldsymbol{H}^{(h)}[x, t, :]$ onto the top two principal components and labeling each point with its corresponding x.

The analysis shows that each head groups numbers by their remainder modulo different prime factors, enabling the model to select suitable representations in later layers. The prominence of small primes in the top principal components likely reflects their frequency in the training data. Furthermore, we believe the performance gains observed in Figure 6.1 with more heads can be partly attributed to the model's improved capacity to capture additional prime factors for constructing RNS representations.

To further demonstrate that these heads directly influence the model's performance, we measured per-digit accuracy before and after pruning each specialized head, as shown in panels (a2, b2, c2) and (a3, b3, c3). For pruning, we replace the head's output with its mean value $\texttt{mean}(\boldsymbol{H}^{(h)}[:, :, :]) \in \mathbb{R}$ across all positions (using 10% of randomly selected training sequences), which preserves signal scale and avoids catastrophic model degradation. The results show that pruning a head responsible for a particular prime factor significantly impairs performance on corresponding digits, while other digits are less affected. For instance, in panel (a3), the model's ability to compute 7^1 and 7^2 digits is lost, while base-2 and base-3 digits remain above chance. Similarly, in (b3), removing the modulo-2 head erases all corresponding digit accuracy, and in (c3), pruning the modulo-3 head greatly reduces base-3 digit performance, but leaves base-2 and base-5 largely intact.

Finally, we further validate the link between head specialization and digit-wise accuracy

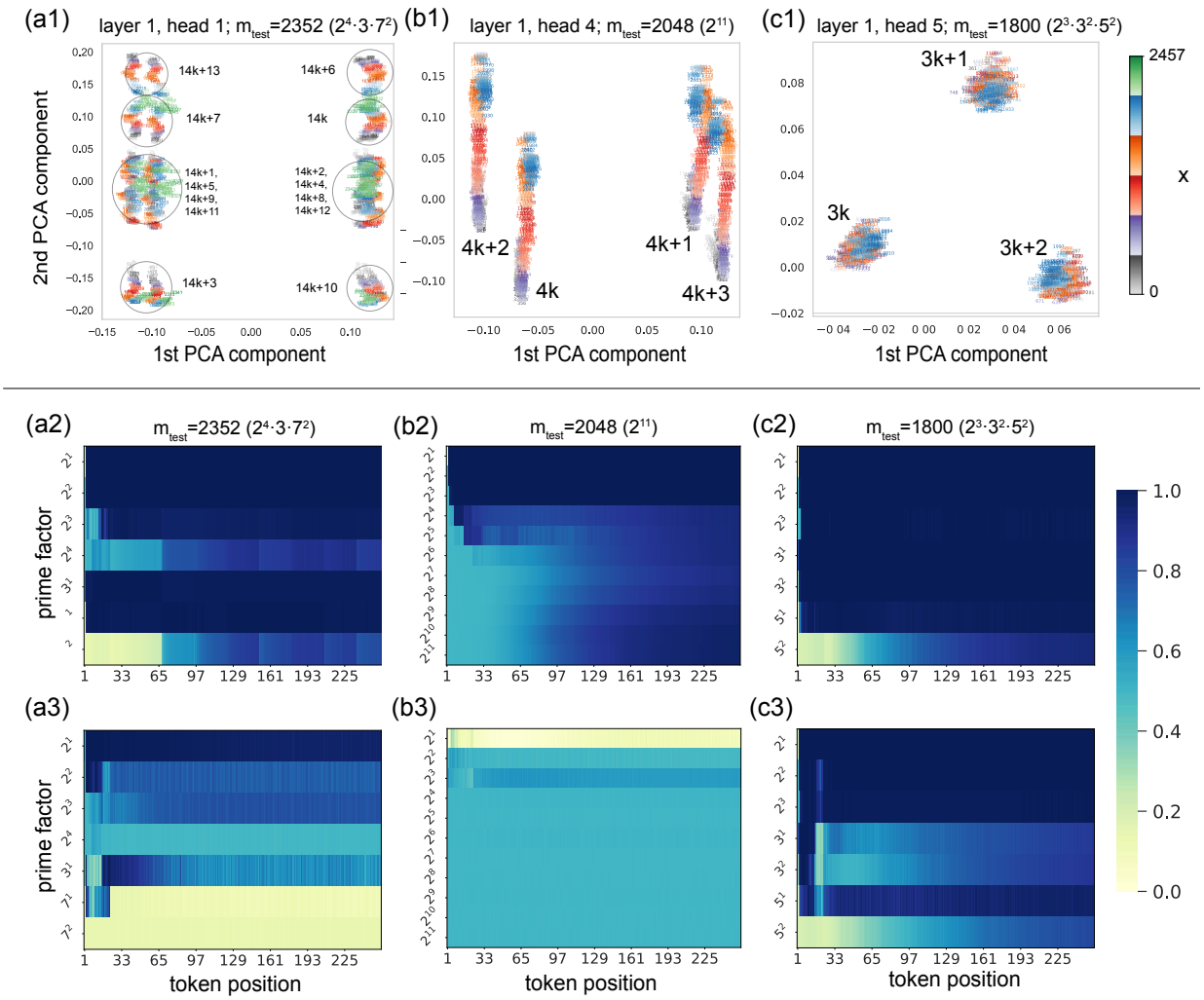


Figure 6.8: **UM**: PCA analysis of attention heads specializing in different prime factorizations and their effect on per-digit accuracy. **(a1, b1, c1)** PCA patterns of outputs extracted from corresponding heads. The first two principal components group numbers based on their remainder modulo 14, modulo 4, and modulo 3, respectively. **(a2, b2, c2)** Test accuracy for individual digits when numbers are represented according to Equation (6.4) for given m_{test}. **(a3, b3, c3)** Test accuracy for individual digits after pruning the corresponding heads shown in panels (a1, b1, c1). Pruning these specific heads substantially degrades performance on their corresponding digits.

by pruning irrelevant heads. For $m_{\text{test}} = 2048$, pruning heads responsible for modulo 7 or 3 (as in panels (a1) or (c1)) yields the results in Figure E.16. Interestingly, removing the modulo-3 head sometimes improves performance for specific token positions, while others show minimal degradation. These findings reinforce the connection between head specialization and digit-wise computation; further details can be found in Appendix E.8.2.

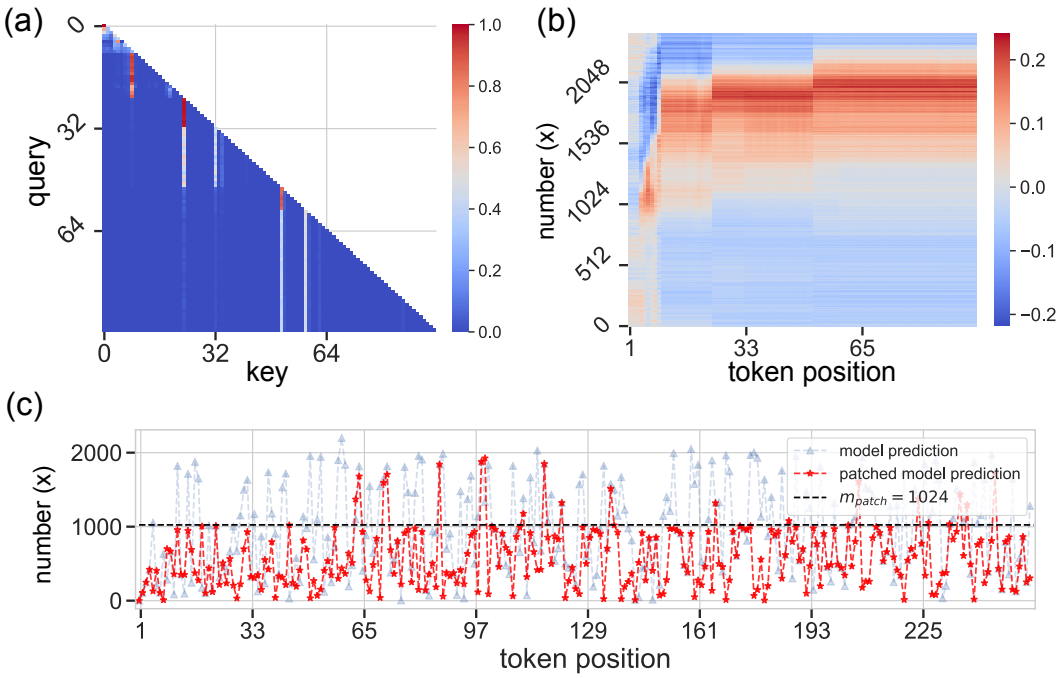


Figure 6.9: **UM**: Attention head (layer 1, head 6) for estimating m_{test}. (a) The query consistently attends to keys corresponding to the largest values within the given sequence; (b) the head generates features with the highest cosine similarity to m_{test}; (c) when patching the feature with sequences generated using $m_{\text{patch}} = 1024$ but maintaining the same a and c, the model often makes predictions with values of n smaller than m_{patch}.

Step ii: The performance variations observed above suggest that the model is internally uncertain about which RNS representation to use, likely due to difficulty in determining m_{test}. As we discuss below, the model appears to estimate m_{test} greedily via in-context learning.

In Figure 6.9(a), using a sequence with $x_0 = 1$, $a = 5$, $c = 31$, and $m_{\text{test}} = 2048$, we observe a first-layer attention head that attends primarily to the largest numbers, as seen by the vertical lines in the attention weights. To analyze this further, in panel (b), we extract the output $\boldsymbol{H}^{(h)} \in \mathbb{R}^{1\times L\times d_{\text{model}}}$ from this head for the same sequence and compute its cosine similarity with token embeddings for all $x < m_{\text{test}}$, producing a $\mathbb{R}^{L\times m_{\text{test}}}$ matrix for the heatmap. This reveals that the head's output consistently has the highest similarity with the largest numbers that has been seen in context, indicating a greedy estimation of m_{test}.

To further verify that this head estimates m_{test} for later layers, we conduct a patching experiment [148], as shown in Figure 6.9(c). We generate a new sequence with the same a, c, and x_0, but with $m_{\text{patch}} = 1024$, and extract $\boldsymbol{H}^{(h)}\text{patch} \in \mathbb{R}^{1\times L\times d\text{model}}$ from this head. We then overwrite the output $\boldsymbol{H}^{(h)}$ in a forward pass for the original sequence ($m_{\text{test}} = 2048$) with $\boldsymbol{H}^{(h)}\text{patch}$. The model now frequently predicts numbers smaller than $m\text{patch}$. As shown in Figure E.18 (Appendix E.8.3), patching other heads disrupts predictions but never induces a similar qualitative shift, confirming this head's unique role in estimating m_{test}.

One might expect that even a small error in estimating m_{test} would invalidate predictions, but this is not necessarily the case. As shown in Section 6.4.2, if the correct representation is chosen, the lower bits maintain a strong periodic signal. Since the model can prepare multiple RNS representations (as demonstrated in **step i**), a sufficiently close estimate of m_{test} allows these lower bits to guide the model toward the correct representation. Further discussion is provided in Appendix E.8.4 and **step iii**.

Step iii: Once the necessary features are prepared, subsequent layers implement the rest of the algorithm. As in the **FM** case, we observe a ladder pattern in digit-wise accuracy for lower digits at early token positions. This pattern, which demonstrates the copying bias, is visible in Figure 6.8(b2).

More careful inspection of Figure 6.8(b2) shows the model copies the lowest 5 digits. Interestingly, given the estimated $m_{\text{test}} = 2033$ from **step ii**, the model effectively reduces it to $2033/2^5 \approx 63.53$. Futhermore, since the model operates on integers, it could round it to 64, yielding $64\cdot 2^5 = 2048 = m_{\text{test}}$. Thus, the model in principle, can predict higher digits accurately without the exact value of m_{test}.

While some algorithmic transition from lower to higher digits clearly occurs, the exact mechanism remains unclear. The sharp transition in per-digit performance at the 2^5 digit in Figure 6.8(b2) supports this argument. Nevertheless, we believe the model applies an algorithm similar to **FM** for higher bits, which we will elaborate on in Appendix E.8.5.

6.5 Scaling Up the Modulus

In this section, we investigate training upon scaling up the LCG modulus, with the following modifications:

Base-b tokenization. To avoid massive dictionary sizes for large m, we implement a base-b tokenization scheme. Each integer is decomposed into a sequence of base-b digits, beginning with the least significant digit; resulting in a vocabulary size of b for any m. Based on the discussion in Section 6.4.1, it is beneficial to choose b such that $\gcd(b, m) = b$.

Abacus Embeddings. We encode positional information using a variant of the Abacus embedding [142], as a sum of two learnable vectors. One vector encodes the position of the integer within the sequence, while the other encodes the position of each digit within the integer (see Figure 6.10).

6.5.1 Fixed Modulus

For each modulus $m = 2^k$, where k is an integer in the range $16 \leq k \leq 32$, we train a 2-layer model with $d_{\text{model}} = 1024$ and a vocabulary size of 256. We select training sequences via the Hull-Dobell theorem, setting $n_a = n_c = 1024$ (See Appendix E.9.3 for training details). For

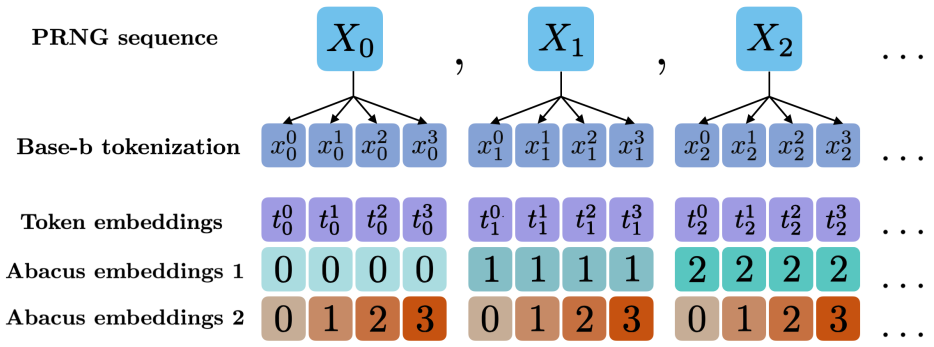


Figure 6.10: Visualization of base-b tokenization and abacus embeddings. Abacus embedding 1 is shared by all the digits within the integer, while Abacus embedding 2 varies within the digit but is shared by all integers.

the test dataset, we choose 512 values of a and 64 values of c that differ from those in the training set.

The quality of an LCG largely depends on its multiplier, traditionally evaluated via the spectral test [145]. In Figure 6.11(a), we test our model on both spectrally optimal Steele multipliers [149] and arbitrary multipliers for $m = 2^{32}$. While achieving 100% test accuracy with equal in-context sequence lengths, the model performs consistently worse on Steele-generated sequences compared to those from arbitrary multipliers.

In Figure 6.11(b), a log-log plot reveals that the number of in-context sequence elements needed for 100% test accuracy scales sublinearly with modulus m as m^γ, where $\gamma \approx 1/4$.

6.5.2 Unseen Modulus

For the **UM** case, we train a 6-layer Transformer on a dataset with $n_m = 32{,}768$, $n_a = 128$, $n_c = 1$, with $1024 < m_{\text{train}} < 65{,}536$. Sequences are length 512, each integer tokenized as two bytes (context length 1023). As before, test data uses unseen m_{test} and (a, c), focusing on $m_{\text{test}} = 2^k, 3^k$. Figure 6.12 shows that the number of in-context sequence elements needed to reach 60% test accuracy scales as m_{test}^γ ($0.24 \leq \gamma \leq 0.33$). The averaged test accuracy of

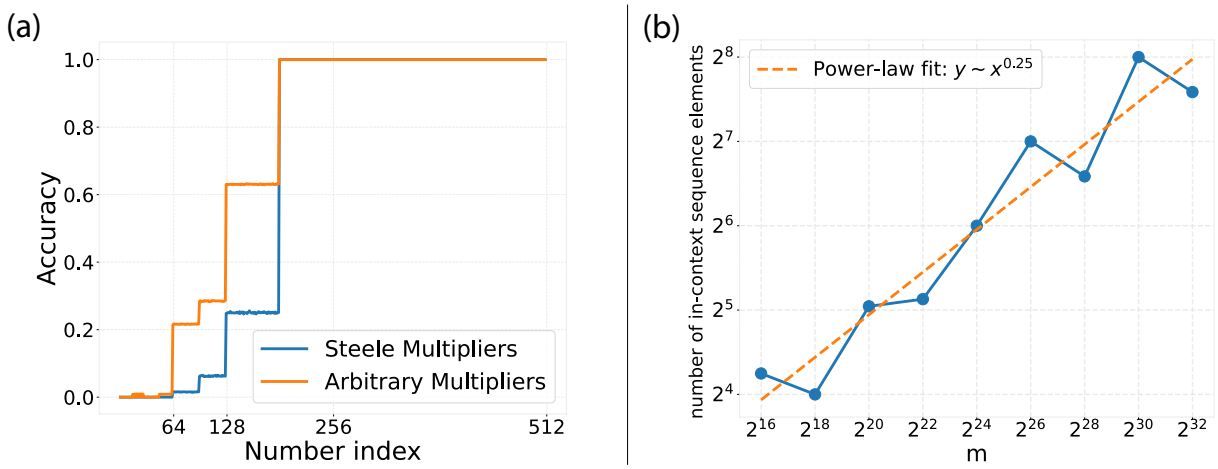


Figure 6.11: **FM**: (a) Average test accuracy ($m = 2^{32}$) vs number of in-context sequence elements. (b) The number of in-context sequence elements required to achieve 100% test accuracy (minimum of 5 runs). (See Appendix E.9.3 for details)

each number in the sequence is shown in Appendix E.9.4. Test performance is influenced by the tokenization base, since the tokenization base highlights the periodic structure of LCGs making it more apparent and easier for the model to leverage during training and prediction. To confirm this, we train a model with tokenization base 243. In Figure 6.12, $m_{\text{test}} = 2^k$ (3^k) sequences scale better when the tokenization base is $256 = 2^8$ ($243 = 3^5$).

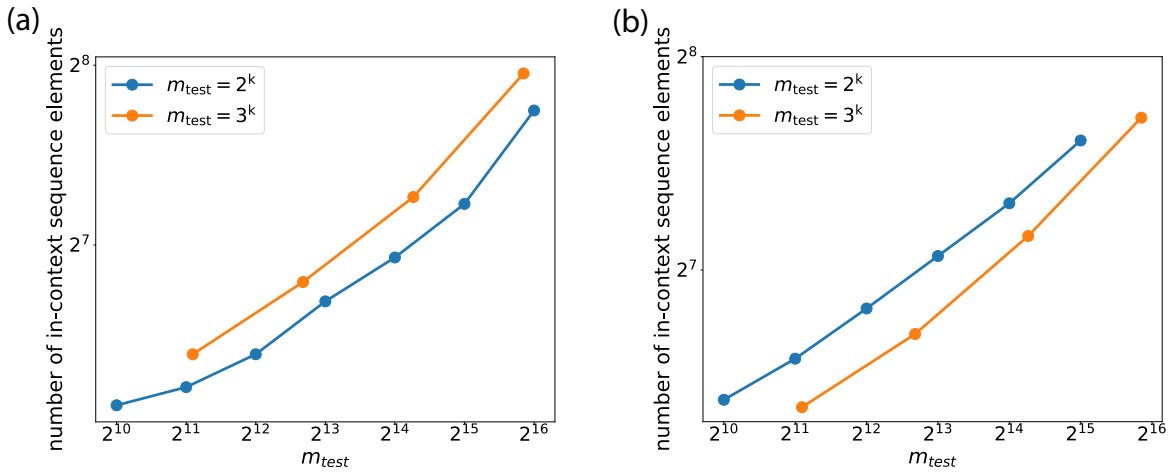


Figure 6.12: **UM**: The number of in-context sequence elements required to achieve 60% test accuracy shows a sublinear dependence on the modulus m. The scaling depends on the compatibility between m and tokenization. (a) base-2^8 tokenization, (b) base-3^5 tokenization.

6.6 Related Work

Interpretability and Modular Arithmetic. A growing body of work examines the circuits, algorithms and structures learned by Transformers [120, 124, 125, 131, 132, 143, 150]. A notably fruitful setting involves simple modular arithmetic problems [4, 5, 6, 29, 30, 103, 104]. Our work adds to this by reverse-engineering the underlying algorithms and uncovering emergent structures in learning pseudo-random number sequences.

Cracking PRNGs. There is a classic duality between cryptography and learning theory [151], and cracking PRNGs is an important topic in cryptography. Nevertheless, deep learning-based attacks have received limited attention in the post-Transformer era. Amigo et al. [152] demonstrated that a fully-connected neural network can predict the outputs of a modified LCG with fixed (irrational) parameters $(a, c, m) = (1, \pi, 1)$. In comparison, we systematically analyze the harder cases of unseen parameters using Transformers, reverse-engineer the learned algorithms, and study effects of scale and complexity.

Formal Grammars. LCG can also be viewed as a formal language (Type-3 regular grammar) lying within the Chomsky hierarchy [153]. Formal languages provide an interesting setting for synthetic datasets that can be used to understand the properties of neural networks in controlled settings [154, 155, 156, 157].

Chaotic time-series. A major application of neural networks is predicting time-series for chaotic dynamics, such as weather prediction [158] and financial modeling. PRNGs provide an analog of such dynamics in the discrete setting.

6.7 Discussion

We have investigated Transformer training on LCG sequences, focusing on fixed modulus training as well as generalization to unseen moduli. In both cases, we have uncovered the algorithm used by the model to solve these tasks and highlighted the model components that implement the steps of the algorithm. We have found that the model finds and utilizes prime factorizations of m and RNS representations of numbers to simplify the sequences and make predictions. We have provided the modified training recipe for scaling up the modulus in both **FM** and **UM** settings, and shown their scaling behaviors.

6.7.1 Limitations and Outlook

The results of this paper were limited to scales $m \leq 2^{32}$. It would be interesting to test our results on much larger moduli as well. We leave the exploration of PRNGs that are built upon LCGs, such as PCGs and truncated LCGs for future works. It would also be interesting to make the training even more unbiased, by training on general classes of arithmetic sequences.

Another interesting direction for future investigation concerns the scaling law. It remains unclear what determines the scaling exponent discussed in Section 6.5. Any improvement in this scaling behavior would not only make models substantially more powerful in practice but could also potentially provide deeper insights into data/sample efficiency across other domains.

Chapter 7: Towards Distributed Neural Architectures

In the previous chapters, we have covered several key components of deep learning, including initialization, parameterization, optimization, and the interplay between architectures and data. Building upon this foundation, we are now prepared to extend our investigation to real-world settings and address broader challenges in deep learning.

Among the many factors at play, we focus on understanding and improving one of the most fundamental components widely accepted in the deep learning community: the connectivity between layers. Throughout this dissertation, we have consistently referred to "layer l" or "depth L," implicitly assuming that all layers are structurally similar and arranged in a fixed, sequential order. This chapter challenges that assumption and explores new perspectives on network connectivity.

The primary practical motivation for revisiting this paradigm stems from scaling. As the scaling laws continue to hold empirically [17, 159, 160], and as increasingly powerful models are developed by adhering to these principles, we are rapidly approaching a point where the next generation of models will be too large to train within a single data center. At that scale, training must inevitably span multiple data centers, introducing fundamental physical constraints:

1. Failures caused by random thermal or power fluctuations, as well as radiation-induced bit-flip errors, will become non-negligible

2. Communication between data centers is inherently limited by the network bandwidth and speed of light

It is important to note that these limitations arise in part from the fact that current model architectures are constructed with layers arranged in a strictly sequential order, requiring computation to proceed layer-by-layer[1] and often demanding extensive synchronization. One can imagine that if different parts of a model could operate at least partially independently, in other words, in a distributed manner. Then many of these issues could be alleviated:

1. A distributed network enables independent sub-network execution, providing asynchrony and improved fault tolerance.

2. By leveraging independently operable sub-networks, communication overhead can be reduced through careful network topology design, thereby decreasing the frequency of inter-component communication.

Although we have outlined the potential benefits of distributed network design, it is important to acknowledge that these advantages are likely to become significant only at scales far beyond what is currently feasible. Moreover, while physical constraints are an important consideration, the natural preferences inherent in data represent another crucial factor that can exert non-trivial influences on architectural design. Therefore, as a proof of principle, this chapter focuses on the following key concepts from the perspective of emergent structure:

- **Dynamic compute allocation**

- **Interpretable modularity and specialization**

[1]This does not sound as bad, since this restricted design allows people to employ pipeline parallelism to overlap communication with computation.

The challenges associated with fully distributed and asynchronous training are left for future work.

7.1 Emergent Connectivity

To achieve our objectives, the most natural approach is to adopt an emergent paradigm, in which the computation and connectivity patterns of the network arise organically during training. This idea is largely inspired by prior work on conditional computation [161], with Mixture-of-Experts (MoE) architectures serving as a canonical example. In MoEs, different tokens are allowed to follow distinct computational paths through the network, with each path determined by a series of learned routers during training.

We leverage a similar concept to construct DNA models. The key distinction is that DNAs are initialized with a collection of routers and a diverse set of modules (such as MLPs, attention mechanisms, Transformers, etc.). Each token is routed along its own path through the network, with both the modules and routers being jointly optimized. This flexible construction encompasses feed-forward, MoE, Mixture of Depth (MoD)[162], weight sharing, and early exit architectures as special cases that can emerge during optimization. In practice, we find that a mixture of *all* these methods tends to emerge from end-to-end training.

We test this idea on two types of settings: (i) discriminative models in vision classification, trained on ImageNet [163] and (ii) generative language models trained on a subset of fineweb-edu. Here we provide a brief summary of what we find in Figures 7.1 to 7.3.

The rest of this Chapter is organized as follows: in Section 7.2 we explain the general set up, establish some terminology, and introduce certain choices we made to ensure trainability.

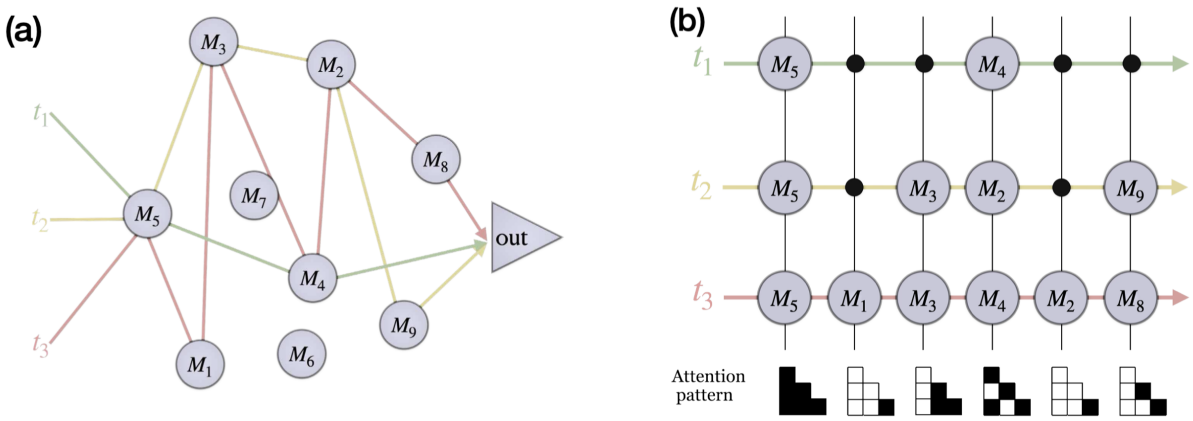


Figure 7.1: **(a)** DNA from the module's perspective: Each token follows its own trajectory through the network. There is no notion of depth or width. With the help of zero compute experts (identity operation), different tokens may have different path lengths. **(b)** DNA from the token's perspective: The forward pass consists of a series of steps. At any given step, each token is acted on by a module or by an identity operation (black dots). When a module that contains an attention operation acts on several tokens simultaneously, the attention pattern is computed *only* between these tokens. Consequently, the attention part of the architecture can be understood as dynamic (*i.e.* data-dependent) sparsity.

In Section 7.3 we present the vision DNA models and analyze their inner-workings, while in Section 7.4 we do the same for the generative language DNA models. In Section 7.5 we present our conclusions and discuss the new emerging research directions.

7.2 Distributed neural architectures

7.2.1 General set up and intuition

In this section, we describe the general structure of DNAs, as well as different ways of reasoning about its nature. The main idea can be traced back to the classic work of [164]: we view a neural network as a collection of computational modules that develop specialization, learn interaction and composition end-to-end. The internal structure of the computational modules as well as communication pattern between them are *emergent*. Our objective is to train well-performing

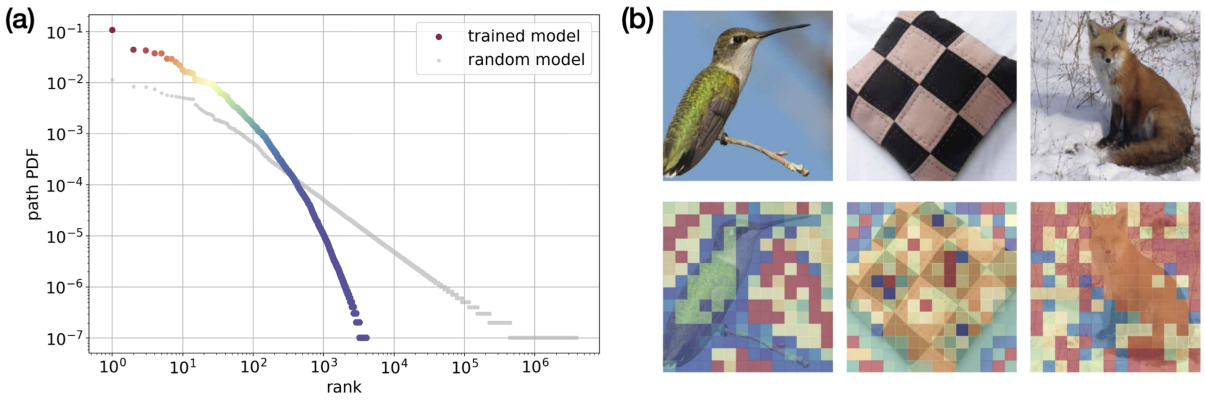


Figure 7.2: **(a)** Path distribution of top-1 trained DNA vision model: we plot the distribution of paths that tokens take through the model, collected from the test split of ImageNet. The color corresponds to the frequency of the paths and is used in the sub-figure **(b)**. Surprisingly, the distribution of paths through the random model also follows a power law with exponent -1. **(b)** We highlight the paths that different patches take through the DNA by their frequency. We see very clear *emergent* specialization of paths: some paths focus on the object, others on the background, and others on the edges and boundaries. This visualization also illustrates the sparse nature of the attention: patches of different colors do not attend to each other in later steps of the forward pass.

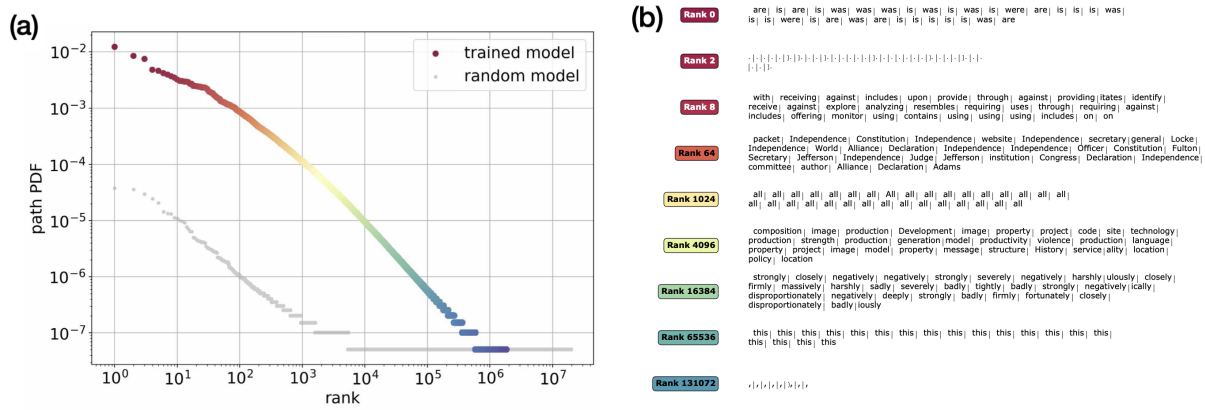


Figure 7.3: **(a)** Path distribution of top-1 trained DNA language model: we plot the distribution of paths that tokens take through the model. The color corresponds to the frequency of the paths and is used in the sub-figure **(b)**. The distribution is a power-law with exponent -1.2. Similarly, we find that paths through the random model are distributed according to a power-law with exponent -1. **(b)** Examples of tokens that follow frequent (low rank) and rare (high rank) paths. We see that paths specialize in different ways: versions of the verb "to be", embedding of "." which we view as sentence-level attention, adjectives, etc..

models and then to understand the distributed nature of computation, emerging connectivity and specialization rather than saturating any particular benchmark. In order to take full advantage of

DNAs the infrastructure has to be co-designed with the emergent structure (or, equivalently, infrastructure should shape/constrain the emergent structure). We leave this direction to the future work.

General DNAs. In the general setting, the proto-architecture of DNAs is a collection of the following components

- Input node (including embedding/patchify layers)
- Output node (including unembedding layer)
- N_m distinct computational modules, M_i
- N_r distinct routers, R_j

Each computational module operates on sequences and can be a transformer, MLP, attention or any other operation. The total number of modules N_m determines the total number of parameters, but not the compute. Each router is a classifier with (at most) N_m classes. Routers process tokens in parallel and send them out to the corresponding modules. The router can be trained with a general top-k choice. In our experiments we take $k = 1, 2$.

The forward pass (after traditional embedding operations) works as follows: (i) token embeddings are routed to relevant modules, (ii) modules perform computation on token embeddings they received, (iii) recomputed embeddings are put back together and step (i) is repeated. In Fig. 7.1 we show the forward path from the perspective of modules and from the perspective of tokens. The latter makes it clear that the forward pass is fully causal (in contrast to MoD [162]), while the former emphasizes distributed nature of DNA.

Paths and ribbons. This design naturally introduces a new unit for interpreting the network behavior: instead of focusing on activations, we can study the routing decisions and communication patterns between modules. To extend the discussion to the top-k case with $k > 1$, we introduce an additional concept: each token simultaneously follows multiple paths that collectively form a *ribbon*, which generalizes the notion of a single path. Given a set of modules $\{M_i\}$, a path is defined as an ordered list of integers corresponding to the indices of the modules it traverses

$$\text{path} = [i_1, i_2, \ldots, i_{s_{\max}}] \ , \tag{7.1}$$

while a ribbon is defined as a list of k-tuples

$$\text{ribbon} = [\tau_1, \tau_2, \ldots, \tau_{s_{\max}}] \ , \tag{7.2}$$

where each $\tau_n = \left(i_1^{(n)}, i_2^{(n)}, \ldots, i_k^{(n)}\right)$ is a k-tuple.

Each k-ribbon can be viewed as a collection of $k^{s_{\max}}$ paths[2], however this interpretation becomes prohibitively costly to implement even in the case of $k = 2$. A sequence of T tokens is characterized by T ribbons. The number of possibilities for the paths and ribbons is exponentially large. Next, we will build some intuition and discuss different ways of thinking about DNAs.

Dynamically sparse attention. From the perspective of the attention modules, routing can be viewed as dynamically generated sparsity. As explained on Fig. 7.1 the routing is (i) fully causal and (ii) at each step of the forward pass, each attention module computes the attention matrix *only* over the tokens it has received. The tokens that were skipped at a given step do not participate

[2]For further accuracy paths can be weighted by the corresponding probabilities.

in any attention at that step. The choice of the routed tokens as well as the attention pattern is often human-interpretable as can be seen in Fig. 7.7 and Fig. 7.11. Finally, dynamically sparse attention requires less compute.

Interpretability of trained DNA models. Routed models (including MoE, MoD and DNA) present a new path for interpretability research, as the fundamental units for analyzing is no longer restricted to neuron paths. Since each path/ribbon is assembled from a collection of routing decisions. Thus it is natural to attempt to interpret these paths as well as the individual routing decisions. For example, we find that maximizing the probability that an image follows a particular path allows to reconstruct the essential features of the image Fig. 7.6. We perform a collection of experiments interpreting routing decisions, ultimately confirming that routers contain human-understandable structure.

7.2.2 Technical details

In this paper, we made a few purely empirical design choices that allowed to take advantage of optimizations such as flash attention as well as to reduce the search space. Each of our modules can be chosen from the classic GELU-transformer block or its attention/MLP component. In all cases, we use the Pre-LayerNorm (Pre-LN) version. Many improvements/gains are certainly left on the table.

Routing. Our routers are linear (token-choice) classifiers. The probabilities of selecting a given M_i at step-s for token t is obtained by applying softmax over router logits, where the routing decision is made by sampling with hard top-k. This softmax results in $\rho^{(s,t)} = \mathrm{softmax}(R_s(\boldsymbol{h}^{(s,t)}))$.

Each token chooses k modules to participate in, leading the i^{th} module to have an input, $\boldsymbol{h}_i^{(s)}$, which is the collection of those tokens. At each step-s, the outputs of the k chosen modules are combined as follows[3]

$$\boldsymbol{h}^{(s+1,t)} = \boldsymbol{h}^{(s,t)} + \sum_{i \in \text{top-}k_\star(\rho_\star^{(s)})} \rho_i^{(s)} \left(M_i^t(\boldsymbol{h}_i^{(s)}) - \boldsymbol{h}^{(s,t)} \right) , \tag{7.3}$$

where $\boldsymbol{h}^{(s)}$ is the pre-activation at step-s and $\star$ denotes the collection of all possible module indices, and the t superscript on M denotes the t^{th} component of the output. This specific choice is made building on [1, 165] to ensure good signal and gradient propagation.

The routers are arranged as follows: at each token-processing step s a router R_s makes the decision for *all* tokens. The total number of token processing steps is capped by a hyperparameter s_{max} which determines the maximum compute per token. While this is not the most general way of arranging the routers, we had to limit our exploration to converge in finite time.

Emergent compute efficiency. In order to teach the DNA to use compute intelligently we introduce several *identity* modules that do not apply any operation: If the router sends a token to the identity module then nothing happens to the token ($\boldsymbol{h}^{(s+1)} = \boldsymbol{h}^{(s)}$). To encourage the network to skip modules (and save compute) we follow the bias trick from Deepseek model [166]: We introduce one bias $b_i^{(s)}$ for each step s. These biases are decoupled from the Autograd as in [166]. While in the original work the biases were used to enforce load balancing we use them to encourage the models to use identity layers by modifying the top-k selection part of the forward

[3]Somewhat awkward for of Eq.7.3 is explained by the fact that $M_i^t(\boldsymbol{h}_i^{(s)})$ is assumed to have a skip connection built-in, so we subtract $\boldsymbol{h}^{(s,t)}$ and the add it later to make sure we do not overcount it and end up with blowing up signal.

pass via

$$i \in \text{top-}k \left(\rho_\star^{(s)} + b_\star^{(s)} \right) . \tag{7.4}$$

Note the biases are only non-zero when the index i corresponds to Identity module and will not affect the probability in Fig. 7.3.

We initialize all biases at 0 and only update the term corresponds to identity modules during training

$$b_i^{(s)}(t+1) = b_i^{(s)}(t) + u \cdot \text{Sign} \left(r \cdot k \cdot \bar{c}^{(s)}(t) - \sum_{i \in \text{Id}} c_i^{(s)}(t) \right) , \tag{7.5}$$

where $c_i^{(s)}$ represents the token counts that pass through a given module, and $\bar{c}^{(s)}$ denotes the average token counts across all modules, r is introduced to control the ratio of tokens to skip and u is for controlling the update speed of the bias term. We emphasize that because of the identity modules each token is processed using different amount of compute. Furthermore, the amount of compute per sequence varies due to sparse attention patterns.

Optimization objective. We optimize a traditional categorical cross-entropy loss function with AdamW optimizer with learning rate warm-up and $\cos$-decay for vision and warm-up stable decay for language. The values of hyperparameters, initialization scheme, etc can be found in the Fig. F.1. We do *not* use load-balancing because our objective is to let models develop the structures they need in order to solve the task and then to understand these structures. Optimization of DNAs for real-world inference is delegated to the followup.

Backbone. We found that the optimization converges much better if first few modules are not routed, meaning they process all tokens and that choice is hard-coded. We refer to these few layers as *backbone* and denote the number of layers in the backbone as N_b. In our experiments $N_b = 0, 1, 2$.

7.3 DNA for computer vision

In this Section we describe DNAs trained on ImageNet. Our models are competitive with the dense baselines, yet the computation performed by the models is quite different.

7.3.1 Experimental details

We trained several DNA models at a ViT-Small scale [64], with the complete set of hyperparameters reported in Table 7.1. In all cases, we used the same data-augmentation pipeline—RandomCrop, RandomHorizontalFlip, and ColorJitter [10]; AutoAugment [167]; Random Erasing [168]; MixUp [169]; and CutMix [170]. All models were trained for 300 epochs with a batch size of 2048. We performed a grid search over learning rates $\eta \in \{0.001, 0.0015, 0.002\}$ and weight-decay values $\lambda \in \{0.02, 0.05, 0.1, 0.2\}$. For futher details, we refer the readers to Fig. F.1.

The training curves for the best run of each model and the routing patterns of the top-1 and top-2 DNA models are shown in Fig. 7.4.

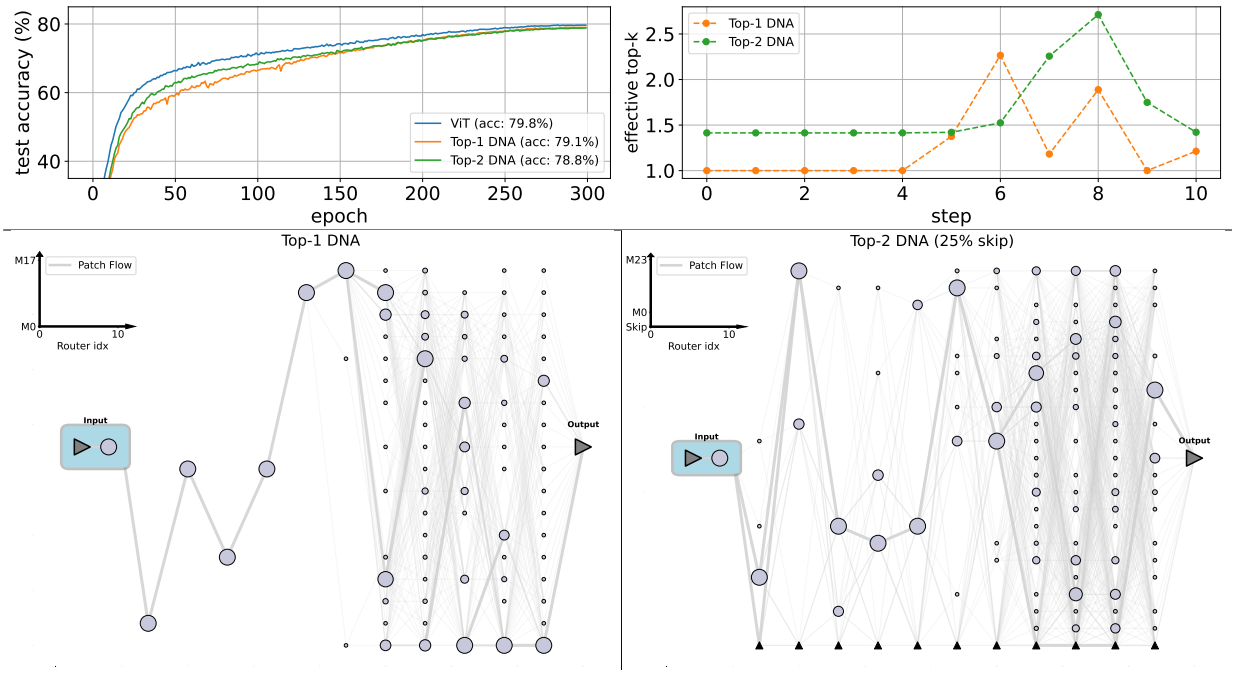


Figure 7.4: **Top-Left**: Test accuracy on ImageNet validation set during training for DNAs and the dense baseline. **Top-Right** Effective number of compute nodes used per step, for DNA models at the end of training. In both cases, the models exhibit higher diversity in routing decisions near the output. **Bottom**: Flow of patches plotted using the validation set of the ImageNet for the top-1 (left) and top-2 (right) models. Each column of circles contains *all* modules in the DNA. The size of the circles is proportional to the frequency with which each module is activated. If a circle is absent, then it is never activated. We see that the models develop a dense foundation and later split into distributed sparse networks.

Model	N_b	N_m	N_r	d_{embed}	d_{MLP}	N_{head}	Active Params	Params
ViT-small	-	12	-	384	1536	6	22M	22M
top-1 DNA	1	18	11	384	1536	6	22M (17M)	34M
top-2 DNA (25% skip)	1	24	11	256	1024	4	18M (15M)	18M

Table 7.1: Hyperparameters of DNA models and ViT-small. Note that for DNA models, the number of active parameters shown in parentheses refers to non-shared active parameters (as detailed in Sec. 7.3.3).

7.3.2 Interpretability

In this section we analyze how the trained models work with an eye for improving them later.

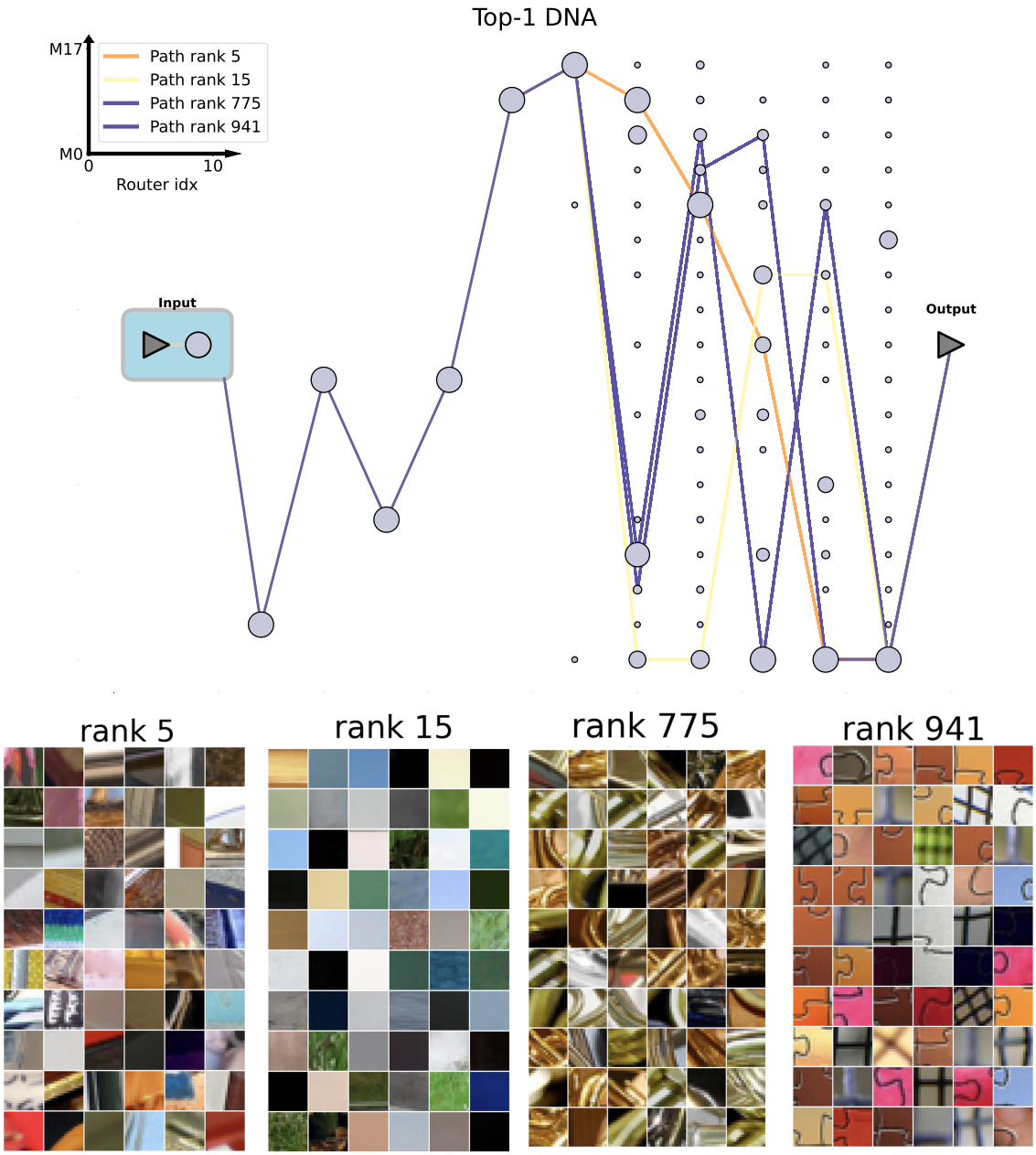


Figure 7.5: **Left**: Paths followed by the patches on the right. The color corresponds to the rank. **Right**: Patches that follow the highlighted paths. The patches that pass through lower-rank paths share a higher-level commonality (color and edges), while patches that go through higher-rank paths are associated with the more specific concept (brass instruments and puzzle pieces).

Patches and Paths. We found that the paths taken by patches are highly interpretable. To demonstrate this, we used the top-1 DNA model shown in Fig. 7.4 and collected the paths of all patches from images in the ImageNet validation set. In Fig. 7.5, we highlight four representative

paths (colored by their corresponding CDF values), along with 60 randomly selected patches that follow each path.

We observed that frequently taken paths (i.e., those with low rank) tend to aggregate patches from diverse images that nevertheless share high-level features—for example, edges in the case of the rank-5 path, and flat color regions for the rank-15 path. In contrast, infrequently taken paths (i.e., those with high rank) tend to group patches from visually similar images with low-level similarities—for instance, brass instruments for the rank-775 path and puzzle pieces for the rank-941 path. To establish a baseline, we performed a similar analysis on a randomly initialized DNA model. Somewhat surprisingly, we found that it also can cluster images, however it uses a very different similarity measure, leading to nearly identical patches being clustered based on superficial features[4]. See Fig. F.4.2 for further discussion.

We have also found that in the cases when an image has a clearly defined objects such as in Fig. 7.2 and Fig. 7.7 there is a group of patches that follows the boundary between object and the background. In fact, the most compute-heavy images are the ones with very intricate collection of boundaries Fig. 7.8. These observations seem to be related to the work [171] where it was found that the patches most critical for classification are of the same nature. Finally, in Fig. 7.7 we observe that boundary patches can be used to (almost holographically) reconstruct part of the object via deep-dream-like method.

Deep-dream analysis of paths. To understand routing decision and module specialization more generally, without relying on validation data, we use "activation maximization". Instead of maximizing the output of a neuron or channel as is typical, we maximize the weight or proba-

[4]One could have expected these results in view of the work on signal propagation [28] which argues that for properly initialized (random) networks similar inputs have similar representations

bility of tokens following a given path. We view the collection of paths taken by an image through the network as a collective variable that describes the configuration of the network. We then generate other typical images that follow the same paths as by starting with noise and maximizing the probability that the each token follows the same path (or ribbon) as the initial image Fig. 7.6.

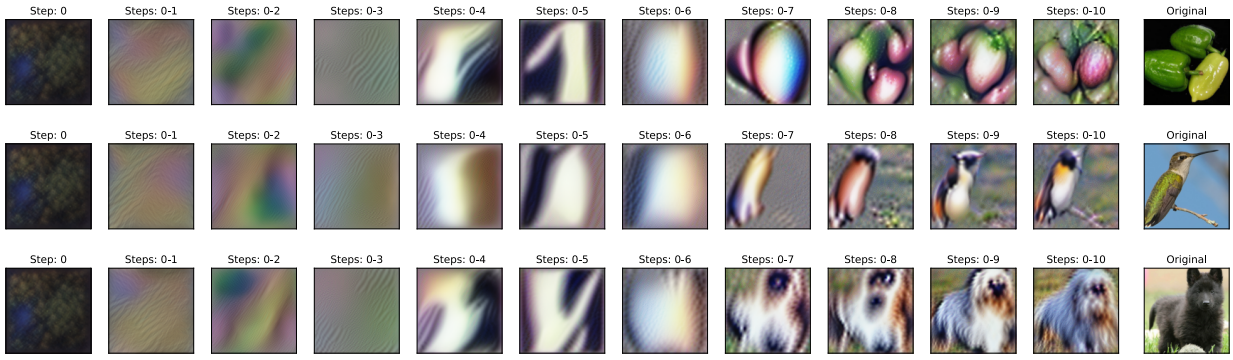


Figure 7.6: Three examples of reconstructed images: bell pepper, hummingbird, and Welsh springer spaniel (top to bottom). Images are reconstructed by maximizing the total weight on *all* routing decisions made from step 0 to step i of the forward pass. We can see that the early layers (1-3) are primarily concerned with texture and edges, intermediate layers (4-6) with lighting distributions, and the remainder (7-10) host larger-scale features. The final reconstructed images (top to bottom) are classified as "spotlight" ($p = 0.4826$), "hummingbird" ($p = 0.5502$), and "papillon" ($p = 0.4393$). In the last two cases, all top 5 model guesses are birds and dog breeds correspondingly, reflecting hierarchical nature of ImageNet. Top 5 model guesses for the first row are: spotlight, matchstick, candle, car mirror, snowmobile.

The simplest objective, O_S, is the sum of the probabilities, up to step $S \leq s_{\max}$,

$$O_S \equiv \sum_{s=1}^{S} \sum_{t} \sum_{i \in \text{Topk}_\star(\rho_\star^{(s,t)})} \rho_i^{(s,t)}, \tag{7.6}$$

which we maximize by performing gradient descent with respect to the input. The choices, i are fixed at their original values throughout. One way to interpret this quantity is to consider ρ to be the conditional probability, $\rho_i^{(s)} = \mathrm{P}\left[i|\boldsymbol{h}^{(0)}\right]$, of the choice given the input. While ρ might not be a true probability (since we sample top-k) or we might substitute it with the corresponding logit, the intuition will remain the same.

In this case,

$$\nabla_{\boldsymbol{h}^{(0)}} \rho_i^{(s)} = \nabla_{\boldsymbol{h}^{(0)}} \frac{\mathrm{P}\left[\boldsymbol{h}^{(0)}|i\right] \mathrm{P}\left[i\right]}{\mathrm{P}\left[\boldsymbol{h}^{(0)}\right]} = \rho_i^{(s)} \nabla_{\boldsymbol{h}^{(0)}} \left(\log \mathrm{P}\left[\boldsymbol{h}^{(0)}|i\right] - \log \mathrm{P}\left[\boldsymbol{h}^{(0)}\right]\right), \tag{7.7}$$

so this procedure updates the input towards a direction which increases its probability conditioned on the choice i more than decreasing the prior probability over all choices.

In Fig. 7.6 we see the results of optimizing the objectives O_S for $S = 0, \ldots, 10$ for three selected images. It is clear from the penultimate column that optimizing O_{10} results in generating a *semantically similar* image to the original input, which demonstrates that the routing decisions are highly informative about the original image. The network's choices encode information about the relative location of objects, the type of object and the separation between the objects and the background exemplified by the three reconstructed peppers, the clear reconstruction of a bird and a dog, as well as the reconstruction of a grassy background in the bird image. Looking at Eq. 7.7, we interpret the generated image as the one most likely to result in the particular choices made while classifying the original image. Therefore, the differences between the original and reconstructed image are also important. For example, the final reconstructed image does not maintain the colors from the original: pink peppers rather than yellow, and a white dog rather than a black one. This shows that choices are being made on the basis of specifically semantic boundaries.

Additionally, in the earlier choices, $S < 10$, the structure emerges from lower-level details building up to higher-level ones, with a sharp transition between steps 6 and 7. In the early steps, the routing choices look at edges (1-3) and on patches of light and dark (4-6). At step 7 we can suddenly see the object emerge which implies that the model begins to collect the processed

information from prior steps and to classify the image. The consistency of this effect across different samples is interesting, and suggests an common structure across images. Steps 7-10 are an iterative refinement of details in the image, such as the arrangement of animal parts and even texture[5]. Overall, we see that the router learns to intelligently route tokens through the network.

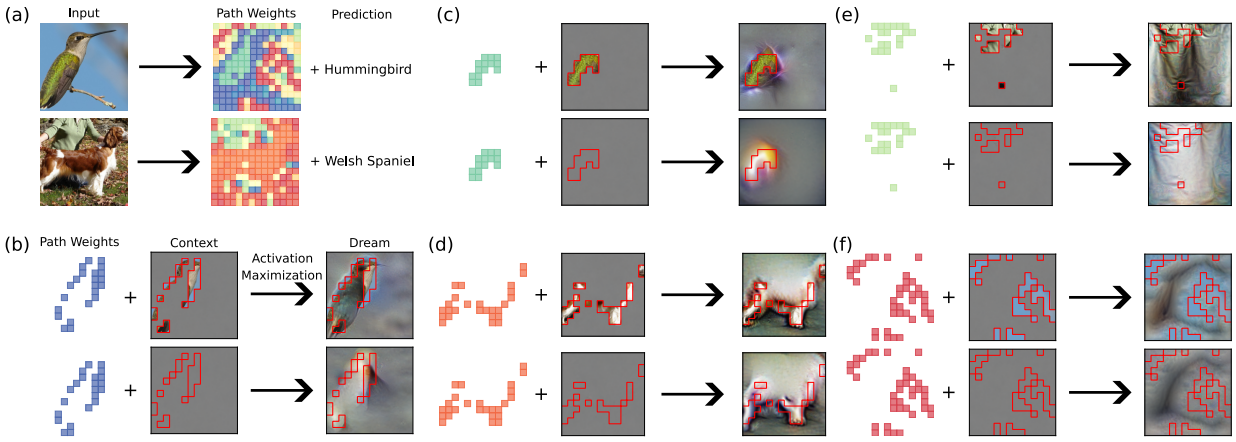


Figure 7.7: A schematic representation of the dreaming procedure. During inference the DNA chooses a trajectory for each token, building a computational graph towards the output shown in (a). Our dreaming procedure accepts a subset of token trajectories and context from the input and then optimizes the remainder of the input to maximize the weight on those trajectories (b-f). The red boundary delineates the group of tokens whose decisions we maximized. The upper rows have context from the original image in this region while the lower row of each example does not. The main finding is that the boundary patches have non-local information about both object and background: when these patches are provided as a context then upon dreaming procedure generates the object and some background. The patch patterns also nicely illustrate the emergent sparse attention.

Deep-dream analysis of segmentation. We now use the same framework to understand why the distinct choices made by different groups of tokens (visualized in Fig. 7.2 through the colored overlay) are responsible for classifying the image, and how these choices rely on the context of the input image. To do this we define a more general objective which optimizes the path weights for a subset, T, of input tokens, and which may condition a subgroup of patches to be the same as

[5]We noticed that our models are particularly sensitive to details in images of animals and nature, which makes sense as ImageNet has a large number of these images.

the original input. This allows us to explore how context of the set T can influence those choices made by tokens in T. To be concrete this objective

$$O_S^T \equiv \sum_{s=1}^{S} \sum_{t \in T} \sum_{i \in \text{Topk}_\star(\rho_\star^{(s,t)})} \rho_i^{(s,t)}, \tag{7.8}$$

optimizes the path weights for tokens in T up to step S. We will choose $S = 10$ from now on.

We naturally group tokens by placing tokens which follow the same path throughout the network into the same group. After separating choices into these (disjoint) subsets we perform two experiments. First, we optimize the whole image to maximize O_S^T. As shown in Fig. 7.7 without any context this procedure reconstructs some features from the original image such as the whiteness of the dog in (d), cloth texture (e), and cloudy texture (f). Though they share some features of the original image these reconstructions lack specific detail.

We compare these reconstructions with those augmented by the original pixels of the image inside the group T. When we add this information we can see that the reconstructions in the remainder of the image are much more detailed. For example adding the boundary of the hummingbird suddenly reproduces the interior of the bird, or adding the color of the shirt reproduces the color across the shirt. This implies that context from the original image is important in explaining why certain choices were made, or what they are sensitive to in the remainder of the image.

In either case, the portion of the image reproduced is only that corresponding to the object itself (e.g. the hummingbird and not the background sky, the shirt and not the dog below, or the sky and not the hummingbird in front). This implies that the decisions, made with the context from the input, are primarily dependent on the each object separately. Based on this we

hypothesize that decisions are largely related to object segmentation.

7.3.3 Efficiency

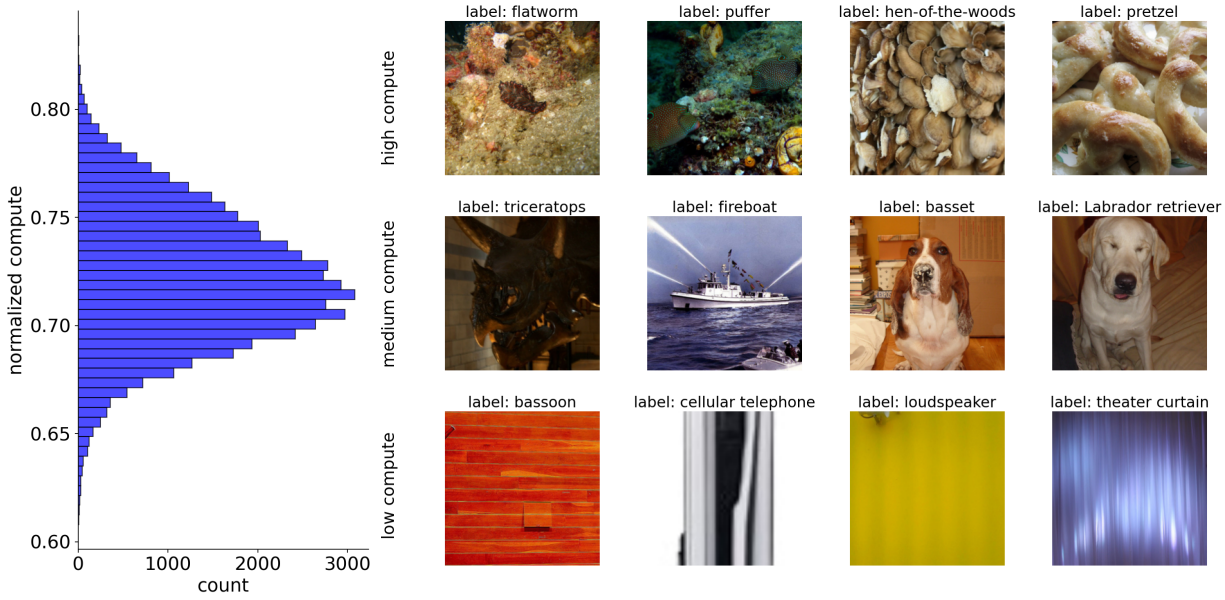


Figure 7.8: Distribution of computational cost on the ImageNet validation set for the top-2 DNA (25% skip) model. We observe that (i) the compute follows roughly Gaussian distribution; we believe that it is a reflection of the dataset rather than the model. (ii) the amount of compute that the model spends on the image correlates with the visual content. Namely, the model prioritizes boundary patches, so in high compute images almost everything looks like a boundary, whereas in low compute images almost everything is a background.

Compute. When we allow patches to take paths of varying lengths, we find that the model does so in an interpretable manner at the image level, suggesting that its decisions are strongly contextual. We examine this behavior using the Top-2 DNA model (with 25% skip rate) shown in Fig. 7.4 by measuring the compute allocated to each image. Specifically, we count the number of computational modules used along the ribbon of each patch, average this count over all patches within an image, and normalize the result such that the normalized compute equals 1 when the ribbon never skips a module.

The results are shown in Fig. 7.8. On the left, we plot the distribution of compute across all images in the ImageNet validation set. On the right, we randomly select four images for each compute level—high (top 1% of images), medium (49%–51%), and low (bottom 1%). We observe that images requiring lower compute tend to be visually simpler, containing no "object". On the other hand, high compute images are very vibrant with textures and the model burns all compute it can trying to segment it. Similar experiments can be done at patch level, where we also find a similar correlation between patches and compute, see Fig. F.4 in Appendix F.4.1.

Parameter sharing. We observe that both top-1 and top-2 models tend to reuse certain modules, leading to *emergent*, input-dependent weight sharing. We emphasize that the models have not been explicitly incentivized to share parameters. This raises a natural question: what kind of data leads to more weight sharing. On Fig. 7.9 we show several results. First, we quantify how much both top-1 and top-2 models re-use parameters. We find that the distribution is gaussian implying that on average 25% and 15% of parameters is reused. Second, we show that examples of heavy and light reuse images. It is clear that high re-use images do not contain the object. The models like to reuse the module that has full (*i.e.* not sparse) attention. AS a sanity check we verified that images that use the least active parameters are classified correctly and with high confidence.

Next we ask: is the parameter-sharing behavior similar for different models? In other words, does it depend on the model or on the data? To answer this we consider a correlation between module re-use of top-1 and top-2 models. We find that there is a good correlation between the two. This suggests that active-parameter saving techniques developed by the models rely on similar features.

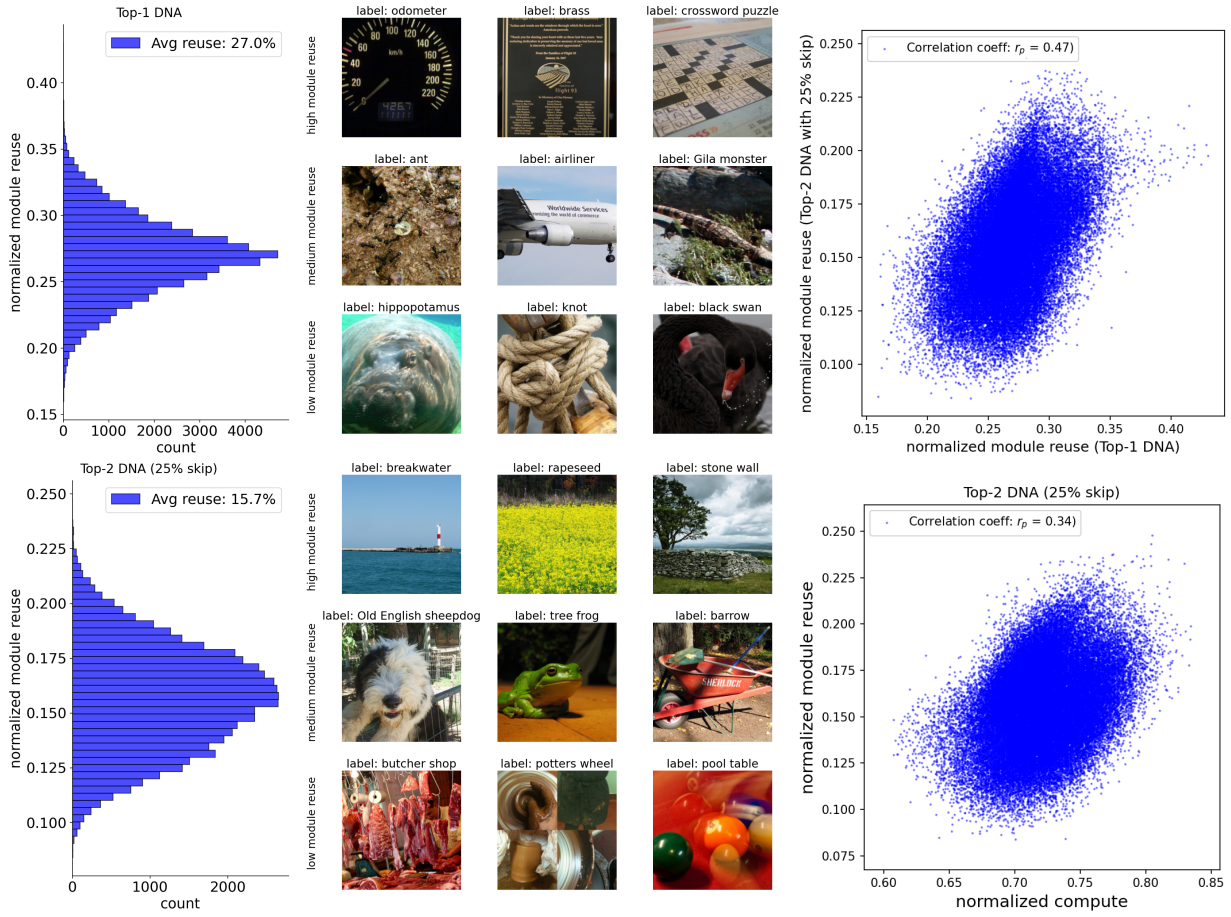


Figure 7.9: **Left**. Statistics of module reuse and representative images for the top-1 and top-2 DNA models. We see that high model reuse is dedicated to the complex images that lack and object to be segmented. For those images attention tends to spread everywhere. Low module reuse images are less interpretable, but we surmise that most of the image is taken by the object. **Top-Right** Correlation between module reuse for top-1 and top-2 DNA models. Both models tend to reuse modules on the same images (high correlation). **Bottom-Right**. Correlation between module skip and module reuse for the top-2 DNA model. The model uses different features to save compute and to reuse modules (low correlation).

Finally, in the top-2 case skipping the modules reduces *both* FLOPs and number of active parameters. We would like to know if FLOP savings correlate with module reuse. To this en we make a correlation plot between the amount of module reuse and module skip for the top-2 model. We observe that these decisions are not correlated. Thus, we conclude that model uses different criteria to save compute and to re-use modules.

7.4 Language DNA models

In this Section we discuss DNA language models. Before diving in we emphasize that language and image datasets (and objectives) are *very* different. In particular, FineWeb-edu is vastly more complex than ImageNet. Consequently, our models are way too small to truly absorb it (or even 21B token part of it). Nonetheless, already at this small scale and vastly "under-parametrized" regime we observe many interesting effects.

7.4.1 Experimental details

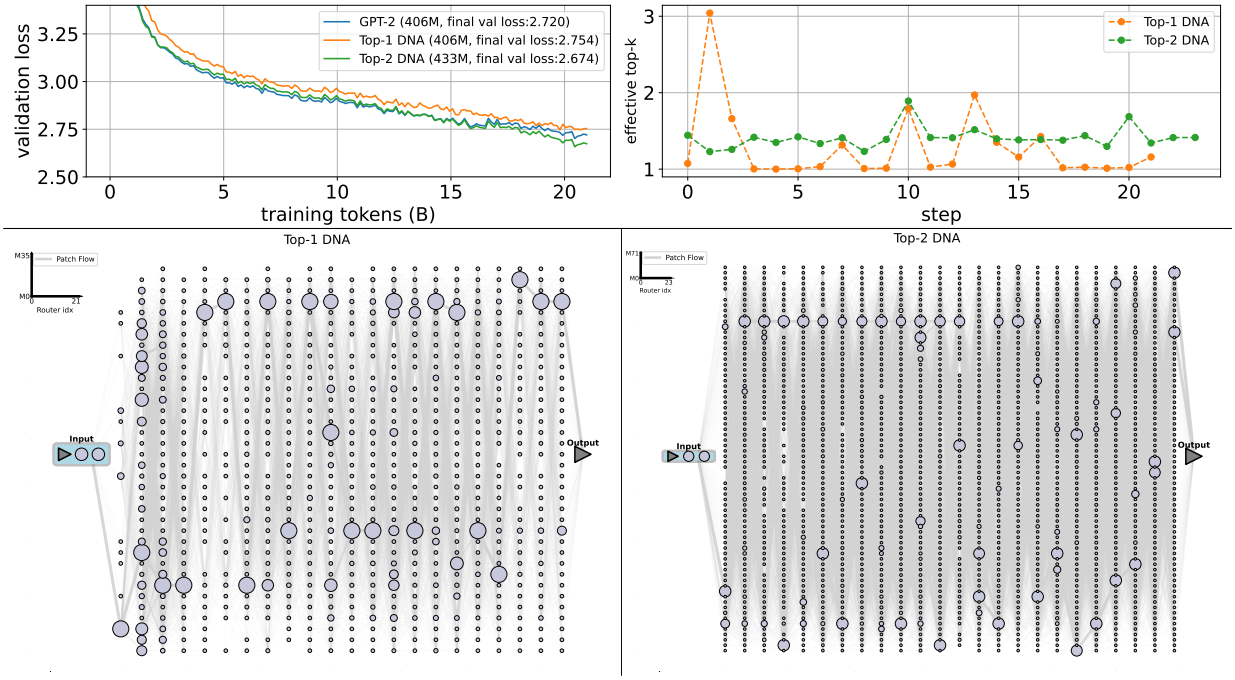


Figure 7.10: Models trained with 21B tokens sampled from FineWeb-Edu dataset. **Top-Left**: Validation loss measured on a subset that we leave aside. **Top-Right**: Effective number of compute nodes used per step for DNA models at the end of training. The two models exhibit different behaviors in the early steps. Compared to the patterns observed in vision models, they both follow different trends as the number of steps increases. **Bottom**: Flow of tokens plotted using the validation set of the FineWeb-Edu dataset for the top-1 (left) and top-2 (right) DNA models.

We trained several DNA models at a scale comparable to GPT-2 Medium [172], with the

Model	N_b	N_m	N_r	d_{embed}	d_{MLP}	N_{head}	Active Params	Params
GPT-2 medium	-	24	-	1024	4096	16	406M	406M
top-1 DNA	2	36	22	1024	4096	16	406M (242M)	583M
top-2 DNA	2	72	24	768	3072	12	433M (266M)	603M

Table 7.2: Hyperparameters of DNA models and GPT-2 medium (no weight-tying). Note that for DNA models, the number of active parameters shown in parentheses refers to non-shared active parameters (as detailed in Sec. 7.4.3).

complete set of hyperparameters listed in Table 7.2. All models were trained on the FineWeb-Edu [173] 100B-token subset, for approximately 21B tokens in total ($40,000$ steps with a batch size of 512 and a context length of 1024). For each model, we searched over three learning rates, $\eta \in \{0.0004, 0.0008, 0.0016\}$, with a fixed weight decay value $\lambda = 0.1$. All models are trained using the same learning rate schedule, which includes a linear warmup for the first $1,000$ steps and a linear decay to $0.1 \cdot \eta$ for the last $8,000$ steps. The training curves of the best run for each model, together with the routing patterns of the top-1 and top-2 DNA models, are shown in Fig. 7.10. We present a few standard benchmark results in Table 7.3.

Model	Loss (↓)	ARC-E (↑)	BoolQ (↑)	HellaS (↑)	LAMBADA (↑)	PIQA (↑)	RACE (↑)	Wiki (↓)
Random	10.825	25.0	50.0	25.0	0.0	50.0	25.0	$\sim$ inf
GPT-2 (406M)	2.720	58.9 ± 1.0	60.5 ± 0.9	40.5 ± 0.5	33.8 ± 0.7	66.9 ± 1.1	32.3 ± 1.4	33.7
top-1 (406M)	2.754	56.9 ± 1.0	60.8 ± 0.9	38.6 ± 0.5	28.7 ± 0.6	65.8 ± 1.1	30.9 ± 1.4	38.7
top-2 (433M)	**2.674**	59.2 ± 1.0	61.0 ± 0.9	**41.8** ± 0.5	34.0 ± 0.7	67.9 ± 1.1	31.1 ± 1.4	**31.5**
GPT-2 (30% shallower)	2.772	58.0 ± 1.0	54.9 ± 0.9	37.9 ± 0.5	31.4 ± 0.7	65.9 ± 1.1	30.1 ± 1.4	38.0
top-2 (30% skip)	2.784	52.5 ± 1.0	52.9 ± 0.9	35.5 ± 0.6	23.8 ± 0.6	64.2 ± 1.1	28.1 ± 1.4	52.6

Table 7.3: Final validation loss and zero-shot downstream evaluation. We reported accuracy in all cases with standard error, except for Wikitext, where we reported word-level perplexity. Columns are bolded only when there is a clear winner, taking standard error into account. Hyperparameter details for the model with skipping and the shallower GPT-2 are listed in Appendix F.1.

7.4.2 Interpretability

In this subsection, we focus on understanding the behavior of the top-1 DNA model shown in Fig. 7.10. We find that the early routers learn to group semantically or lexically similar tokens.

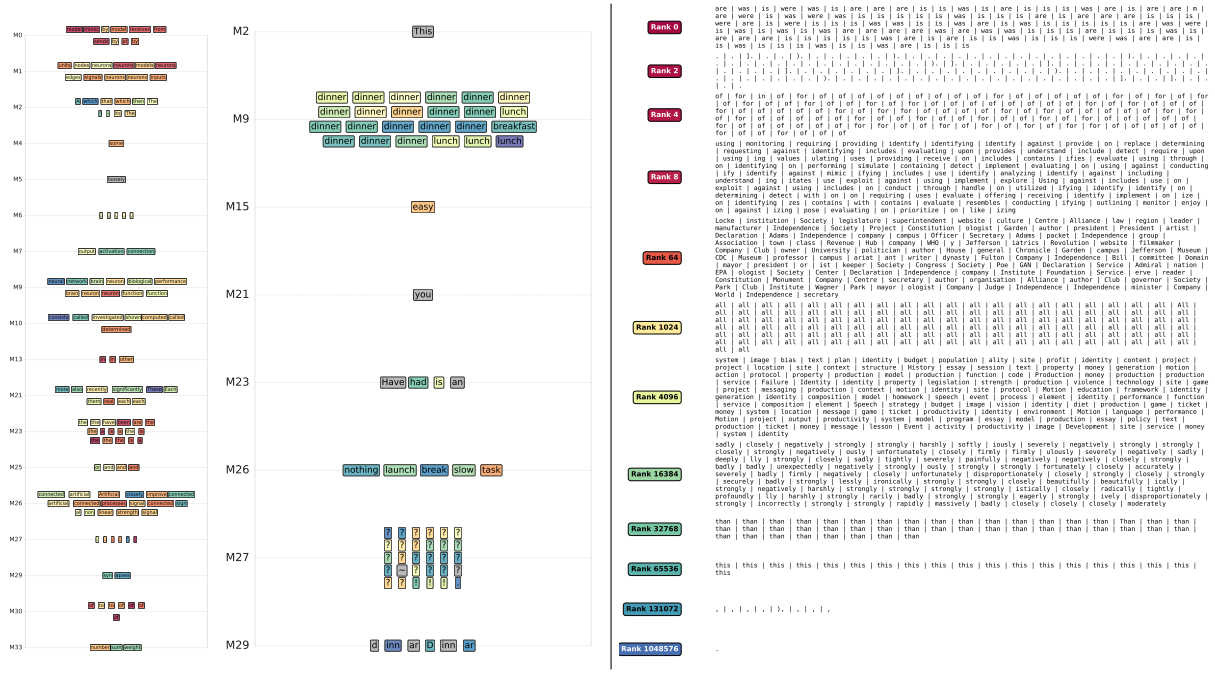


Figure 7.11: **Left**: Two examples illustrate how the router R_1 (at step-1) directs semantically/lexically similar tokens to specialized modules. For instance, M_9 groups tokens related to neural concepts in the first example and to "breakfast, lunch, dinner" in the second, whereas M_{29} tends to receive word-pieces that can be combined into whole words. **Right**: Tokens (separated by |) traverse distinct paths that align with semantic/lexical categories.

Early routers group similar tokens. We selected two example paragraphs: One from Wikipedia and one hand-crafted (see Fig. F.5.1 for the full paragraphs and Fig. F.6,F.7 for full token flows). Then we visualized the routing decisions made by R_1 in the left panel of Fig. 7.11. Each token is colored according to the CDF value of the path it follows, using the same color scheme as in Fig. 7.3.

We found that the router R_1 consistently sends semantically similar words to M_9, punctuation marks to M_{27}, and word pieces to M_{29} in both examples. In the Wikipedia example, the router additionally groups plural nouns and routes them to M_1, verb variants to M_{10}, and related prepositions to M_{25} and M_{30} accordingly.

Interpreting path distribution. We start with the NLP version of Fig. 7.5. Namely, we visualize the tokens that follow the same path for several different ranks in the right panel of Fig. 7.11. As expected, low-rank (*i.e.* frequent) paths tend to capture frequently used words, such as linking verbs, or words that share similar conceptual roles (e.g., rank-8 corresponds to relationships between actions and their targets or contexts; rank-64 relates to human endeavors). We also find that rank-2 path focuses on end of sentence tokens, which can be interpreted as sentence-level attention.

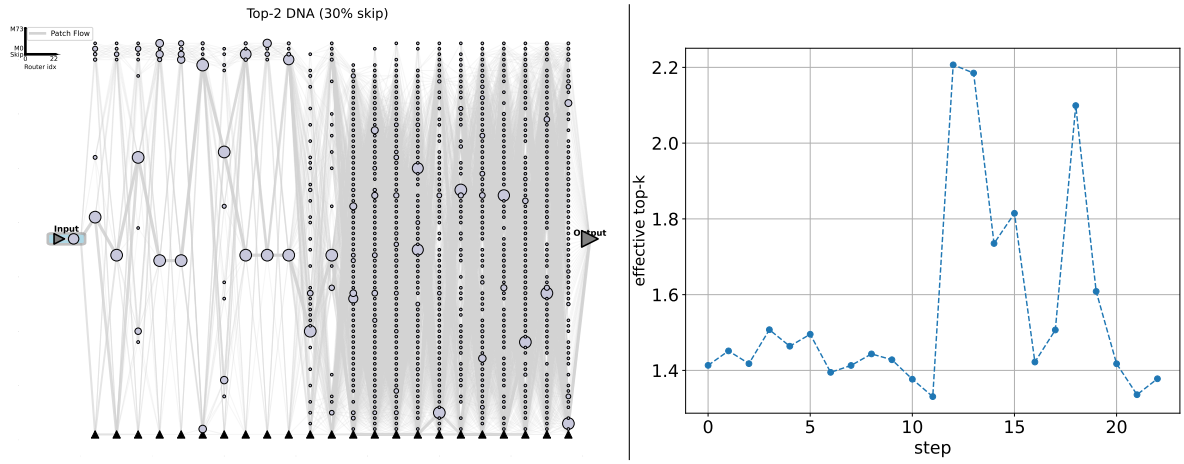


Figure 7.12: top-2 DNA model with a 30% skip rate (final validation loss 2.784). **Left**: Token flow on the FineWeb-Edu validation set. Unlike the model in Fig. 7.10, this model makes similar routing decisions across most tokens in the first few steps, which is similar to the vision cases in Fig. 7.4. **Right**: Effective top-k measured based on the token flow.

In contrast, mid- and high-rank (*i.e.* rare) paths exhibit more variability. These include, for example, distinct adverbs (rank-16384), or even commonly used words appearing in unexpectedly high-rank paths (e.g., ranks 1024, 32768, 65536, etc.). This may seem surprising, as one might expect high-rank paths to capture rare or domain-specific concepts. However, we hypothesize that due to the highly contextual nature of language, common words routed through high-rank paths likely carry context-specific information rather than simply reflecting their base lexical identity. A more detailed investigation of this phenomenon at a larger scale is left for future work.

7.4.3 Efficiency

Compute. Similar to the vision case, we trained a top-2 DNA model that targets skipping 30% of the tokens, where most hyperparameters are the same as those shown in Fig. 7.10, except $N_b = 1$ and N_r=23 in this case. The token flows and ribbon distribution are shown in Fig. 7.12. We make two observations: (i) compute distribution for images and text is very different, showing that text has significantly higher diversity/complexity (see Fig. 7.13), leading to the majority of examples requiring (roughly) the same amount of compute. Note the $\log$-scale of the x-axis in Fig. 7.12. (ii) There are examples in the tail of the compute distribution (a few percent of the total). By inspection, we see that these examples are qualitatively different from the "average examples" in that they either contain HTML code, a large number of links, are parts of the bibliography, or contain many characters from the languages that our model has not learned yet, such as Arabic, Greek, Hebrew etc. Similar experiments can be done at token level, see Fig. F.8 in Appendix F.5.2 for the results.

Parameter sharing. We find that language DNA models also tend to like parameter sharing (see Fig. 7.14. However, unlike in the vision case we find that parameters sharing does not correlate with any notable features in the text. We also make a more careful check by measuring to types of correlation (similar to the vision case). First, we check if module reuse is correlated to the compute. Unlike in the vision case, we find a strong correlation. However, we believe that this correlation has a simple explanation: if fewer modules are skipped, the probability of reuse goes up. Second we check if there are any similarities between text documents that yield higher module reuse. Unlike in the vision case, we find that there is no correlation between two



Figure 7.13: **Left**: Compute distribution measured on the validation set of the FineWeb-Edu dataset for the top-2 DNA model with 30% skip. We observe that the distribution is very heavily peaked around *above* the compute threshold of 30% (note the $\log$-scale of x-axis!). Nonetheless, some documents are low-compute as we show on the right. **Right**: Selected tokens that pass through ribbons with different compute. We find that sequences associated with very low-compute ribbons either contain HTML code, large number of web links, are parts of bibliography, contain many characters from the languages that our model has not learned yet, such as Arabic, Greek, Hebrew etc..

different DNA models. Consequently we conclude that module reuse is most likely random in the language case. This suggests that language DNAs can be further improved by discouraging module reuse.

7.5 Conclusion and discussion

7.5.1 Conclusions

In this work we have introduced distributed neural architectures that process each token differently, depending on its content and context. We introduced initial model architectures and showed that they are trainable and competitive with the corresponding dense baselines in both vision and language domains. We also showed that DNA models can be trained to allocate

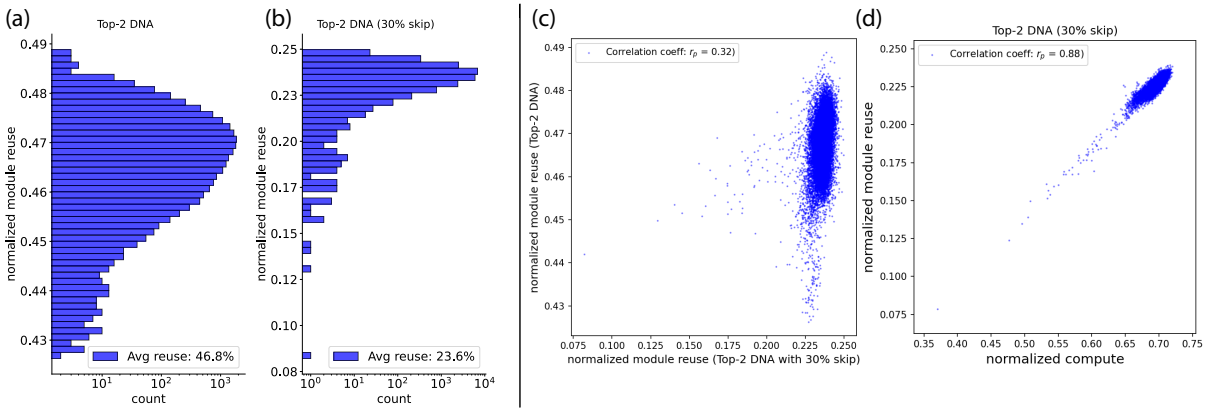


Figure 7.14: **(a),(b)** Statistics of module reuse and representative sequences for the top-2 DNA models with/without skipping. We find that top-2 model reuses about 50% of the modules, while the model trained to skip reuses $\approx 23\%$. **(c)** Correlation between module reuse of two different top-2 DNA models (one with module skipping and one without). We find a lack of correlation, suggesting that skipping decisions are arbitrary. **(d)** Correlation between module skip and module reuse. We find a strong correlation, however we believe that it has a simple statistical nature: if a module is not skipped, it is more likely to be reused.

compute intelligently based on the input, and this allocation is interpretable.

Next we focused on interpreting how these models process information. We found that the paths taken by the tokens through the network are interpretable in both domains. In vision setting we found that the model essentially learns to segment the image, sending similar tokens to the same paths and focusing on the patches that contain boundaries between objects. We also found, using deep-dream-type methods, that the routers concentrate on increasingly more complex features as the number of forward path steps increase.

In the language generation setting, we found that early routers group similar tokens, while paths exhibit more complex behavior, such as focusing on adverbs, emergent sentence-level attention, or semantically complex notions. We also found that some documents require less compute due to their non-informative/noisy nature. We leave a more detailed interpretability study to future work, once we can train larger DNA models.

7.5.2 Unresolved Issues and Future work

Here we list a few directions that we believe would be interesting to ourselves and, at the same time, interesting to the community.

Restricted Connectivity. At present, it remains unclear how to effectively impose locality in these architectures. At the same time, we know that the current DNA design will not scale, since the all-to-all communication required at every forward step is costly and actually undermines our original motivation described at the start of this chapter.

Module variety. In this work we only studied two cases: transformer compute modules and separate attention and MLP modules (see Appendix F.6). Already in this case we found that attention and MLP modules are not collapsing into transformer modules, but organize themselves as a function of depth: early layers prefer attention, while later on the model prefers MLP. It makes sense to introduce a variety of modules: attention (including efficient attentions), MLP of various sizes, etc., and allow the DNA to arrange them in the way it wants.

Local routers. In the present work we have focused on simple linear routers that are labeled by the step of the forward pass. We made this choice primarily to simplify training dynamics and engineering. A more natural distributed choice would assign routers directly to the nodes. This implementation is challenging, especially in $k > 1$ case, but is also more in line with the distributed perspective we took here.

Hierarchical Routers. As we have discussed throughout this chapter, most data inherently possesses its own hierarchy. Therefore, it is reasonable to argue that routing can occur at different

levels and may need to be trained using different algorithms accordingly. How to effectively achieve this remains a very open question.

Architectures search. It is interesting to extract lessons from the DNAs that might be applicable to more rigid, traditional architectures. First, an interesting direction would be to introduce inhomogeneity in the model architecture. Earlier layers should be dense and (likely) not too wide, while later layers should be much wider, and, possibly, sparse. Second, as we discuss in Appendix F.6: MLP and Attention do not need to be attached to each other. We train DNA models that can freely choose between MLP and attention layers. Such DNAs do not glue MLP and attention to get back the transformer architecture. Instead, they prefer to use more attention during early steps and use more MLP during later steps. We also find that the models really like weight-sharing, however it is not clear how beneficial it really is.

Data Filtering / Out-of-Distribution Detection. Another interesting possibility is to invert the DNA framework and design a setting where DNA serves as a smart data filtering system. Based on our experiments, we observe that, for any data collection, DNA can naturally identify underrepresented data points, making it a promising outlier detector.

Chapter 8: Summary and Future Directions

In this dissertation, we have studied various aspects of deep learning, centered around network architecture and emergent phenomena. We began by examining initialization and its interaction with architectural design, emphasizing that a good initialization provides a strong first step in the learning process. To this end, we developed theoretical and empirically efficient tools to diagnose initialization quality across different network architectures. Along the way, we identified emergent depth scales that characterize the effective trainable depth of a network at initialization. These findings, in turn, guide architectural design by suggesting that networks should be constructed to maximize these scales.

However, initialization alone is not sufficient to guarantee successful training. We therefore studied the effects of parameterization and initialization on training dynamics, particularly sharpness dynamics, using a simple two-layer model that captures many of the essential characteristics of real-world networks. Drawing insights from non-linear dynamical systems, we developed an intuitive description of function space dynamics and, through analysis of the corresponding fixed-point structure, derived a universal guideline for stable model training, which we then verified across multiple datasets.

A parallel, though not independent, direction of our research focuses on emergent structures arising from data. By steadily increasing the complexity of modular arithmetic datasets and

leveraging insights from theoretical analysis, we were able to isolate the generalization circuit from the memorization network. Moreover, we empirically and mechanistically examined the emergence of in-context learning (ICL) capabilities. This analysis provides valuable insights into how Transformer models acquire such abilities from data and reveals the minimal model depth required for them to emerge.

Finally, building on the insights and experiences from the previous chapters, we challenge one of the most fundamental assumptions in neural network architecture design – inter-layer connectivity. We propose a new architecture DNA, that allows arbitrary inter-layer connectivity patterns. By training DNA models on both vision and language datasets, we observed the emergence of diverse connectivity patterns, and computation distributions that correlated with data complexity. This architecture also serves as a powerful interpretability tool: by introducing token paths, or *ribbons*, as observables, we can clearly interpret fundamental abilities that emerge during training and are often difficult to characterize in traditional architectures.

Before discussing future directions, we would like to discuss the spirit and methodology of this dissertation. We believe it should become self-evident as we proceed, yet it is worth emphasizing the methodological framework we draw from physics and, more broadly, the natural sciences. This approach can be viewed as an iterative combination of the following key elements:

1. Observation: Identify a key phenomenon, often one that appears unexpected or unnatural.

2. Hypothesis: Propose an underlying cause for the observed phenomenon, typically accompanied by a simplified model consistent with the hypothesis.

3. Prediction: Derive testable predictions or expected outcomes from the hypothesized model.

4. Experiment: Conduct experiments to verify or falsify the hypothesis.

This methodology has led to many successes in deep learning research, and we believe it will continue to shine in the future. Our experience suggests that the more complex a system becomes, the more effective this methodology tends to be.

Now, we would like to outline several promising and important directions for future research.

Rethinking Architectural Design and Scaling As discussed in Chapter 7, several existing works suggest that the current paradigm of scaling up model size using Pre-Norm architectures may be reaching a limit, where adding more layers no longer yields the expected benefits. However, this design choice arose for historical and practical reasons. Since such architectures can be viewed as depth expansions along an identity mapping, with Normalization layers controlling the effective coefficient, they offer remarkable stability and robustness to initialization. Yet, this blessing also becomes a curse: it introduces shortcuts that allow the model to minimize loss using effectively shallower subnetworks. Once this preferential attachment becomes embedded in the architecture, it becomes difficult for the model to fully utilize its maximum depth.

On the other hand, these shortcuts also emerge naturally from the properties of language data. In the early stages of training, models tend to capture low-order statistical features (such as n-gram patterns with a small n) because they are the easiest to learn. Since the intrinsic structure of language cannot easily be changed, improvement must come from both optimization strategies and architectural innovations. From an optimization perspective, the training pipeline could be designed to prevent the model from learning too quickly at the early stage. From an architectural perspective, one could design structures that discourage such shortcuts, though this may come at the cost of the stability advantages provided by Pre-LayerNorm designs.

This contradiction motivates a shift in our design paradigm. Three immediate directions can be envisioned:

1. Different connectivity: human brains do not work like a one layer after another with a fixed pattern, and most real-world data is not strictly within that format as well. This way of thinking would allow us to have a chance to break the curse of preferential attachment, which we believe is the main improvement from Mixture-of-Experts (MoE) architectures compare to the conventional dense models.

2. Model growth: train smaller models more thoroughly, then progressively expand them in a way that avoids shortcut formation. This may help preserve depth utilization during scaling.

3. Model shrinking: begin with a larger model than needed, allowing its internal shortcuts to correspond to deeper subnetworks of the eventual smaller model. This approach could retain the benefits of Pre-LayerNorm stability while mitigating its drawbacks.

While both directions have been explored in recent work, there is still no clear consensus. Different studies adopt distinct perspectives on how to reconcile stability, depth utilization, and optimization efficiency in large-scale neural architectures. However, no matter what we do, it would be very exciting to see what new structures could emerge with this potential paradigm shift.

Going Beyond LLM Pre-Training Most of our discussion has focused on the pre-training stage. However, the field has now entered a phase where post-training and test-time scaling increasingly dominate the landscape. This shift calls for a reexamination of which architectures are

naturally suited for extending computational capacity beyond pre-training – a significant challenge for current attention-based models, whose computational cost scales quadratically with the number of generated tokens. To address this limitation, new architectural variants that can effectively replace the traditional attention mechanism are urgently needed. Although numerous works have explored this direction, no fully convincing alternatives have yet emerged.

Another promising direction along this line is to avoid generating explicit tokens and instead perform computation directly within the model's high dimensional latent space, a paradigm often referred to as latent reasoning. Current progress suggests that achieving this will likely require architectural modifications to make models inherently capable of reasoning and computation in latent space.

Finally, we conclude with one overarching lesson: meaningful progress in this rapidly evolving field demands both deep expertise in specific domains and a broad, integrative vision that connects them across disciplines.

Appendix A: Emergent Length Scale at Initialization

A.1 Experimental Details

We implemented our methods using PyTorch [69] hooks and an efficient Jacobian approximate algorithm [67].

Figure 2.1. We generated MNIST-like inputs, where all elements are sampled from the Gaussian distribution $\mathcal{N}(0,1)$. $\mathcal{J}^{0,l}$ data was averaged over 100 different parameter-initializations. Networks were initialized width $N_l = 1000$. For erf plot we initialized at critical point $(\sigma_w, \sigma_b) = (\sqrt{\frac{\pi}{4}}, 0)$, used depth $L = 250$ and the fitting was done with data points collected at depth $l > 100$; for ReLU plot we initialized at critical point $(\sigma_w, \sigma_b) = (\sqrt{2}, 0)$, used depth $L = 100$ and the fitting was done with all data points; for the $\mu = 1$ Pre-LN plot, we initialized both networks at $(\sigma_w, \sigma_b) = (\sqrt{2}, 0)$, used depth $L = 250$ and the fitting was done with $l > 100$ data points.

Figures 2.2 and 2.3. All the phase diagrams were plotted using $\chi_{\mathcal{J}}^{L-1}$ generated from networks with $L = 50$ and $N_l = 500$. We used hooks to obtain the gradients that go into calculating $\chi_{\mathcal{J}}^{L-1}$. $\chi_{\mathcal{J}}^{L-1}$ data was averaged over 100 different parameter-initializations. Inputs were generated from a normal Gaussian distribution and have dimension 28×28. Generating the data for the figure

took approximately 2 days on Google Colab Pro (single Tesla P100 GPU).

Figures 2.4 and 2.5. In all cases, networks are trained for 10 epochs using stochastic gradient descent with CrossEntropy loss. We used the Fashion MNIST dataset [174]. All networks had depth $L = 50$ and width $N_l = 500$. The learning rates were logarithmically sampled within $(10^{-5}, 1)$. Generating the data for the figure took approximately 12 days on Google Colab Pro (single Tesla P100 GPU).

Figure 2.6. (1)We made the $\sigma_w^2 - \sigma_b^2$ phase diagram for ResNet110(LayerNorm) by averaging over 100 different parameter-initializations. The $\sigma_w^2 - \mu$ phase diagram was made by averaging over 200 parameters initialization. (3)(4) We used SGD with momentum$= 0.9$ and batch size 128. For selecting the learning rate we ran a grid-search over $0.001, 0.005, 0.01, 0.02, 0.5$ for 10 epochs; with weight decay $\lambda = 10^{-4}$. All models were trained for 50 epochs and averaged over 3 random seeds. It takes 6 GPU days in total on a single NVIDIA RTX 3090 GPU.

Figure 2.7. (1)(2)We made the phase diagram for MLP-Mixer with 30 blocks and averaged over 100 different parameter-initializations. (3)(4)We used network with $L = 100$, patch size 4×4, hidden size $C = 128$, two MLP dimensions $N_{tm} = N_{cm} = 256$. The $L = 32$ point has doubled widths. All networks have 10 million parameters. Notice that for all Mixer Layers we used NTK initialization. We trained all cases on CIFAR-10 dataset using vanilla SGD paired with CSE. Batch size bs $= 256$, weight decay $\lambda = 10^{-4}$ was selected from $\{10^{-5}, 10^{-4}\}$, mixup rate $\alpha = 0.8$ was selected from $\{0.4, 0.8\}$. We also used RandAgument and horizontal flip with default settings in PyTorch. For all cases we searched learning rates within $\{0.005, 0.01, 0.05, 0.1, 0.2, 0.5\}$. We also tried a linear warm-up schedule for first 3000 iterations, but we did not see any improvement

in performance. Generating the data for the figure took approximately 4 days on Google Colab Pro (single Tesla P100 GPU).

Figure 2.8. Models were trained on CIFAR-10 dataset for 600 epochs using `LAMB` optimizer[175] with $B = 256$. The learning rate is decreased by a factor of 0.1 at 450 and 550 epochs. Training accuracy is measured on training samples with Mixup $\alpha = 0.8$. Both models interpolate the original training set. Results obtained using a single NVIDIA RTX 3090 GPU.

Figure 2.9. Models were trained on CIFAR-10 using SGD with $\text{momentum} = 0.9$. and $B = 256$ for 300 epochs, where we decrease the learning rate by a factor of 0.1 at 150 and 225 epochs. Training accuracy is measured on training samples with mixup $\alpha = 0.8$. Both models interpolate the original training set. Results obtained using a single NVIDIA RTX 3090 GPU.

A.2 Batch Normalization

The operation of BatchNorm on a preactivation (pre-BN) in an MLP can be described as follows:

$$\begin{aligned} \tilde{h}_i(\boldsymbol{x}) &= \frac{h_i(\boldsymbol{x}) - \mu_{i,B}}{\sigma_{i,B}}\,, \\ \mu_{i,B} = \frac{1}{|B|}\sum_{\boldsymbol{x}'\in B} h(\boldsymbol{x}') \quad &and \quad \sigma_{i,B} = \sqrt{\frac{1}{|B|}\sum_{x'\in B}(h(\boldsymbol{x}') - \mu_{i,B})^2}\,, \end{aligned} \tag{A.1}$$

where B is the batch that x belongs to and $|B|$ is batch size.

The works He et al. [2], Yang et al. [39] show that for large batch size, the effect of Batch-Norm for NNGP kernel and Jacobian Norm is deterministic. We summarize the results for the

pre-BN MLP setup:

$$\mathcal{K}^{l+1}(\boldsymbol{x},\boldsymbol{x}') = \frac{\sigma_w^2}{N_l}\sum_{j=1}^{N_l}\mathbb{E}_\theta\left[\phi(\tilde{h}_j^l(\boldsymbol{x}))\phi(\tilde{h}_j^l(\boldsymbol{x}'))\right] + \mu^2\mathcal{K}^l(\boldsymbol{x},\boldsymbol{x}') \tag{A.2}$$

$$\mathcal{J}^{l,l+1} = \frac{\sigma_w^2}{\mathcal{K}^l(\boldsymbol{x},\boldsymbol{x}) - \mathcal{K}^l(\boldsymbol{x},\boldsymbol{x}')}\mathbb{E}_\theta\left[\phi(\tilde{h}_j^l(\boldsymbol{x}))\phi(\tilde{h}_j^l(\boldsymbol{x}))\right] + \mu^2\,, \tag{A.3}$$

where x' in the APJN term can be any $x' \neq x$, since for a large batch size all choices are equivalent.

From the above result, we can see that most results we have for LayerNorm can be translated to BatchNorm with an easy replacement $K^l(x,x) \to \big(\mathcal{K}^l(\boldsymbol{x},\boldsymbol{x}) - \mathcal{K}^l(\boldsymbol{x},\boldsymbol{x}')\big)$. As a simple example, consider a pre-BN ResNet architecture, but with all the Convolutional layers replaced with Linear (Fully Connected) layers. For such a network, we have the following result for $\mu < 1$:

$$\mathcal{J}^{l,l+1} = \frac{\pi^2(1-\mu^2)}{(\pi-1)^2} + \mu^2\,. \tag{A.4}$$

For $\mu = 1$, we have

$$\mathcal{J}^{l,l+1} = 1 + O(l^{-1}) \tag{A.5}$$

A.3 Additional Discussion on ResNets

For Convolution Layers, the NNGP kernel is a 4-index tensor: $\mathcal{K}^l_{\mu\nu;ij}(\boldsymbol{x},\boldsymbol{x}')$, where the Greek letters (μ,ν) index the channels, whereas the Latin letters (i,j) index the pixels. The infinite width limit in this case is achieved by taking the number of channels to infinity (sequen-

tially). In this limit, most of our equations for MLP can be easily rewritten using the convolutional NNGP kernel. However, in this case, the kernel is only diagonal in channel dimension: $\mathcal{K}^l_{\mu\nu;ij}(\boldsymbol{x}, \boldsymbol{x}') = \mathcal{K}^l_{ij}(\boldsymbol{x}, \boldsymbol{x}')\delta_{\mu\nu}$. This additional structure in the kernel makes it difficult to get a closed-form solution for $\mathcal{J}^{l,l+1}$ in general.

For both LayerNorm and BatchNorm setups, we observe common behaviors. In Figure 2.6(2), we see that the network is critical close to $\mu = 1$, as expected from our analysis. One would naively expect the $\mu < 1$ cases also to be critical, since for MLP with ReLU and Pre-LN, $\sigma_b = 0$ is critical regardless of σ_w and μ. However, in Figure 2.6(2), the region away from $\mu = 1$ is in the ordered phase for LayerNorm and in the chaotic phase for BatchNorm. This is likely a result of the kernel $\mathcal{K}^l_{\mu\nu;ij}(\boldsymbol{x}, \boldsymbol{x}')$ not being diagonal in spatial dimensions. We emphasize that the $\mu = 1$ case stays unaffected by this, since the existence of criticality does not depend on the details of the NNGP kernel in this case. This can be readily seen from (A.47).

A.4 Discussion on Transformers

As preliminary empirical results, we applied our method to Vision Transformers (ViTs). Figure A.1 supports our argument in the main text that the Pre-LN architecture is generally insensitive to initialization when $\mu = 1$, where Post-LN is only stable with small σ_w^2; Figure A.2 further demonstrates that the attention mechanism is highly sensitive to initialization and thus vulnerable to the gradient vanishing/diverging problem.

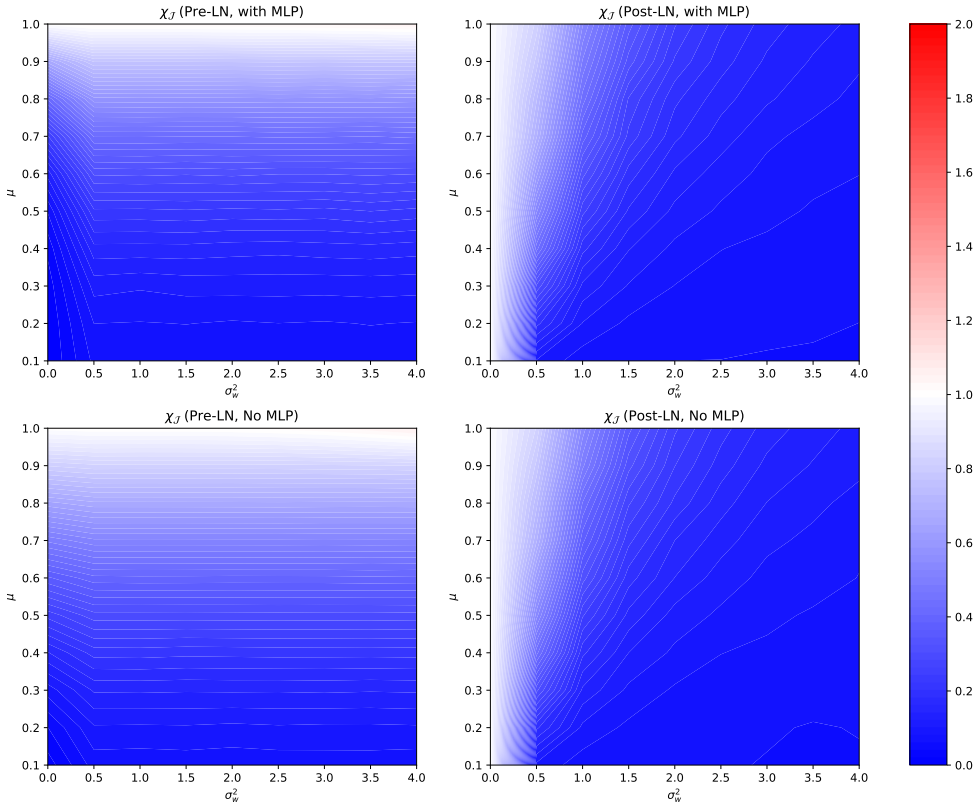


Figure A.1: $\chi_{\mathcal{J}}$ Phase diagram for Vision Transformer (ViT) ($L = 24$, $n_{\text{heads}} = 8$, $d_{\text{embd}} = 256$). We present cases with pre/post-LN, with/without MLP layers. Top-left: Usual ViT – becomes everywhere-critical when $\mu = 1$; ordered otherwise. Bottom-left: Usual ViT without MLP layers. Top-right: ViT with post-LN (instead of pre-LN). Always ordered for finite σ_w. Bottom-right: ViT with post-LN, without MLP layers.

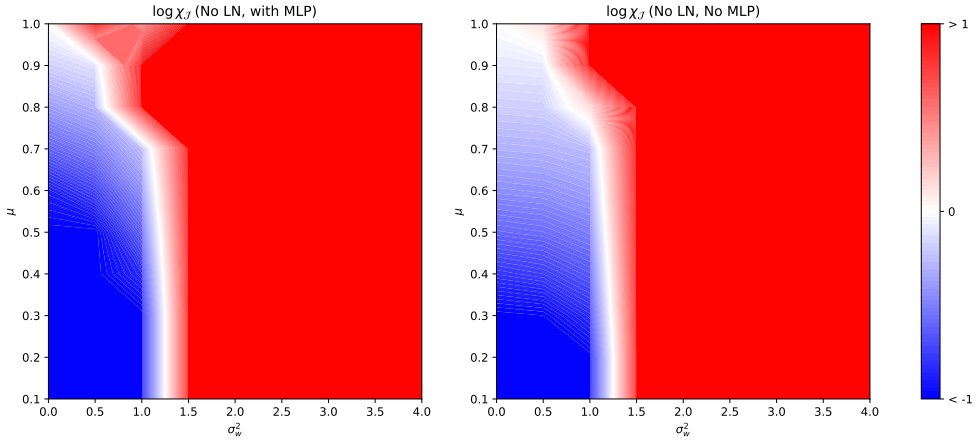


Figure A.2: $\log \chi_{\mathcal{J}}$ Phase diagram for stacked Attention layers with/without MLP layers ($L = 24$, $n_{\text{heads}} = 8$, $d_{\text{embd}} = 256$). We use $\log \chi_{\mathcal{J}}$ and cut off the colorbar at $\chi_{\mathcal{J}} = 10^1$ because the $\chi_{\mathcal{J}}$ values in this case span from ~ 50 orders of magnitude.

A.5 Technical Details and Derivations

We will drop the dependence of $h_i^l(x)$ on x throughout the Appendices. It should not cause any confusion since we are *always* considering a single input.

A.5.1 APJN Proofs and Derivations

Here we provide an extended proof of theorem 2.5 in the main text.

Proof of theorem 2.5. We start from the definition of the averaged partial Jacobian norm (APJN) $(l > l_0)$

$$\begin{aligned}
\mathcal{J}^{l_0,l+1} &\equiv \frac{1}{N_{l+1}}\mathbb{E}_\theta\left[\sum_{i=1}^{N_{l+1}}\sum_{j=1}^{N_{l_0}}\frac{\partial h_i^{l+1}}{\partial h_j^{l_0}}\frac{\partial h_i^{l+1}}{\partial h_j^{l_0}}\right] \\
&= \frac{1}{N_{l+1}}\mathbb{E}_\theta\left[\sum_{i=1}^{N_{l+1}}\sum_{j=1}^{N_{l_0}}\left(\sum_{k=1}^{N_l}\frac{\partial h_i^{l+1}}{\partial h_k^l}\frac{\partial h_k^l}{\partial h_j^{l_0}}\right)\left(\sum_{m=1}^{N_l}\frac{\partial h_i^{l+1}}{\partial h_m^l}\frac{\partial h_m^l}{\partial h_j^{l_0}}\right)\right] \\
&= \frac{1}{N_{l+1}}\mathbb{E}_\theta\left[\sum_{i=1}^{N_{l+1}}\sum_{j=1}^{N_{l_0}}\sum_{k,m=1}^{N_l}\left(w_{ik}^{l+1}\phi'(h_k^l)\right)\left(w_{im}^{l+1}\phi'(h_m^l)\right)\left(\frac{\partial h_k^l}{\partial h_j^{l_0}}\frac{\partial h_m^l}{\partial h_j^{l_0}}\right)\right] \\
&= \frac{1}{N_{l+1}}\mathbb{E}_\theta\left[\sum_{i=1}^{N_{l+1}}\sum_{j=1}^{N_{l_0}}\sum_{k,m=1}^{N_l}w_{ik}^{l+1}w_{im}^{l+1}\phi'(h_k^l)\phi'(h_m^l)\frac{\partial h_k^l}{\partial h_j^{l_0}}\frac{\partial h_m^l}{\partial h_j^{l_0}}\right] \\
&= \frac{1}{N_{l+1}}\sum_{i=1}^{N_{l+1}}\sum_{j=1}^{N_{l_0}}\sum_{k=1}^{N_l}\frac{\sigma_w^2}{N_l}\mathbb{E}_\theta\left[\phi'(h_k^l)\phi'(h_k^l)\frac{\partial h_k^l}{\partial h_j^{l_0}}\frac{\partial h_k^l}{\partial h_j^{l_0}}\right] \\
&= \frac{\sigma_w^2}{N_l}\sum_{k=1}^{N_l}\mathbb{E}_\theta\left[\phi'(h_k^l)\phi'(h_k^l)\left(\sum_{j=1}^{N_{l_0}}\frac{\partial h_k^l}{\partial h_j^{l_0}}\frac{\partial h_k^l}{\partial h_j^{l_0}}\right)\right] . \qquad \text{(A.6)}
\end{aligned}$$

where we used the chain rule and took the expectation value over w^{l+1}. Next, we take the chain

rule again:

$$
\begin{aligned}
\mathcal{J}^{l_0,l+1} &= \frac{\sigma_w^2}{N_l}\sum_{k=1}^{N_l}\mathbb{E}_\theta\left[\phi'(h_k^l)\phi'(h_k^l)\left(\sum_{j=1}^{N_{l_0}}\sum_{m,n=1}^{N_{l-1}} w_{km}^l w_{kn}^l \phi'(h_m^{l-1})\phi'(h_n^{l-1})\frac{\partial h_m^{l-1}}{\partial h_j^{l_0}}\frac{\partial h_n^{l-1}}{\partial h_j^{l_0}}\right)\right] \\
&= \frac{\sigma_w^4}{N_l N_{l-1}}\sum_{k=1}^{N_l}\mathbb{E}_\theta\left[\phi'(h_k^l)\phi'(h_k^l)\left(\sum_{j=1}^{N_{l_0}}\sum_{m=1}^{N_{l-1}} \phi'(h_m^{l-1})\phi'(h_m^{l-1})\frac{\partial h_m^{l-1}}{\partial h_j^{l_0}}\frac{\partial h_m^{l-1}}{\partial h_j^{l_0}}\right) + O(1/N_{l-1})\right] \\
&= \frac{\sigma_w^4}{N_l N_{l-1}}\sum_{k=1}^{N_l}\mathbb{E}_\theta\left[\phi'(h_k^l)\phi'(h_k^l)\right]\mathbb{E}_\theta\left[\left(\sum_{j=1}^{N_{l_0}}\sum_{m=1}^{N_{l-1}} \phi'(h_m^{l-1})\phi'(h_m^{l-1})\frac{\partial h_m^{l-1}}{\partial h_j^{l_0}}\frac{\partial h_m^{l-1}}{\partial h_j^{l_0}}\right)\right] \\
&= \frac{\sigma_w^2}{N_l}\sum_{k=1}^{N_l}\mathbb{E}_\theta\left[\phi'(h_k^l)\phi'(h_k^l)\right]\cdot\frac{\sigma_w^2}{N_{l-1}}\mathbb{E}_\theta\left[\left(\sum_{j=1}^{N_{l_0}}\sum_{m=1}^{N_{l-1}} \phi'(h_m^{l-1})\phi'(h_m^{l-1})\frac{\partial h_m^{l-1}}{\partial h_j^{l_0}}\frac{\partial h_m^{l-1}}{\partial h_j^{l_0}}\right)\right] \\
&= \chi_{\mathcal{J}}^l\,\mathcal{J}^{l_0,l}\,,
\end{aligned}
\tag{A.7}
$$

where we integrate (by parts) over w^l to get the second line. We take $N_{l-1}\to\infty$ and then $N_l\to\infty$ to get the third line. We rearrange terms and use the Eq.(2.15) to get the fourth and fifth lines. Notice that to get the third line we used the fact in the infinite width limit, the distribution of h_i^l is independent of h_i^{l-1}. Thus we proved

$$
\mathcal{J}^{l_0,l+1} = \chi_{\mathcal{J}}^l \mathcal{J}^{l_0,l}\,. \tag{A.8}
$$

□

The critical line is defined by requiring $\chi_{\mathcal{J}}^\star = 1$, where critical points are reached by further requiring $\chi_{\mathcal{K}}^\star = 1$.

As we mentioned in the main text, $l_0 = 0$ is subtle since the input dimension is fixed N_0, which can not be assumed to be infinity. Even though for a dataset like MNIST, usually N_0 is not significantly smaller than width N_l. We show how to take finite $O(N_0^{-1})$ correction into account

by using one example.

Lemma A.1. *Consider a one hidden layer network with a finite input dimension N_0. In the infinite width limit, the APJN is still deterministic and the first step of the recurrence relation is modified to:*

$$\mathcal{J}^{0,2} = \left(\chi^1_{\mathcal{J}} + \frac{2\sigma_w^2}{N_0}\chi^1_{\Delta}\sum_k^{N_0}\frac{1}{N_0}h^0_k h^0_k\right)\mathcal{J}^{0,1}\,, \tag{A.9}$$

where $\mathcal{J}^{0,1} = \sigma_w^2$.

Proof.

$$\begin{aligned}
\mathcal{J}^{0,2} =& \frac{1}{N_2}\mathbb{E}_\theta\left[\sum_{i=1}^{N_2}\sum_{j=1}^{N_0}\frac{\partial h_i^2}{\partial h_j^0}\frac{\partial h_i^2}{\partial h_j^0}\right]\\
=& \frac{1}{N_2}\mathbb{E}_\theta\left[\sum_{i=1}^{N_2}\sum_{j=1}^{N_0}\sum_{k,m=1}^{N_1} w_{ik}^2 w_{im}^2\phi'(h_k^1)\phi'(h_m^1)\frac{\partial h_k^1}{\partial h_j^0}\frac{\partial h_m^1}{\partial h_j^0}\right]\\
=& \frac{1}{N_2}\sum_{i=1}^{N_2}\sum_{j=1}^{N_0}\sum_{k,m=1}^{N_1}\mathbb{E}_\theta[w_{ik}^2 w_{im}^2 w_{kj}^1 w_{mj}^1\phi'(h_k^1)\phi'(h_m^1)]\\
=& \sum_{j=1}^{N_0}\sum_{k=1}^{N_1}\frac{\sigma_w^2}{N_1}\mathbb{E}_\theta[w_{kj}^1 w_{kj}^1\phi'(h_k^1)\phi'(h_k^1)]\\
=& \sigma_w^2\left(\chi^1_{\mathcal{J}} + \frac{2\sigma_w^2}{N_0}\chi^1_{\Delta}\sum_k^{N_0}\frac{1}{N_0}h^0_k h^0_k\right)\\
=& \left(\chi^1_{\mathcal{J}} + \frac{2\sigma_w^2}{N_0}\chi^1_{\Delta}\sum_k^{N_0}\frac{1}{N_0}h^0_k h^0_k\right)\mathcal{J}^{0,1}\,,
\end{aligned} \tag{A.10}$$

where to get the result we used integrate by parts, then explicitly integrated over w_{ij}^1. We have introduced a coefficient of finite width corrections, χ^l_{Δ}, defined as follows. □

Definition A.2 (Coefficient of Finite Width Corrections)**.**

$$\chi_{\Delta}^{l} = \frac{\sigma_w^2}{N_l} \sum_{i=1}^{N_l} \mathbb{E}_{\theta}[\phi''(h_i^l)\phi''(h_i^l) + \phi'''(h_i^l)\phi'(h_i^l)] \,. \tag{A.11}$$

Remark A.3. Notice that the correction to $\mathcal{J}^{0,2}$ is order $O(N_0^{-1})$. If one calculates the recurrence relation for deeper layers, the correction to $\mathcal{J}^{0,l}$ will be $O(\sum_{l'=0}^{l} N_{l'}^{-1})$, which means the contribution from hidden layers can be ignored in infinite width limit.

The $\mathcal{J}^{0,2}$ example justifies the factorization of the integral when we go from the last line of Eq.(A.6) to Eq.(A.8).

Finally, the full Jacobian in infinite width limit can be written as

Lemma A.4 (APJN with $l_0 = 0$)**.** *The APJN (with $l_0 = 0$) of a given network can be written as*

$$\mathcal{J}^{0,l} = \sigma_w^2 \left(\chi_{\mathcal{J}}^{1} + \frac{2\sigma_w^2}{N_0} \chi_{\Delta}^{1} \sum_{k}^{N_0} \frac{1}{N_0} h_k^0 h_k^0 \right) \prod_{l'=2}^{l-1} \chi_{\mathcal{J}}^{l'} \,. \tag{A.12}$$

Note that APJN with $l_0 > 0$ does not receive the $O(N_0^{-1})$ correction.

A.5.2 APJN and gradients

As mentioned in the main text Section 2.2.2, APJN is an important tool in studying the exploding and vanishing gradients problem. Its utility stems from the fact that it is a dominant factor in the norm of the gradients. This can be readily by looking at the (squared) L_2 norm of the gradient of any flattened parameter matrix θ^l, at initialization. Here, we prove the argument

in Section 2.2.2. In the infinite width limit, one gets

$$
\begin{aligned}
\|\nabla_{\theta^l}\mathcal{L}\|_2^2 &= \left(\sum_{all}\frac{\partial\mathcal{L}}{\partial h_i^L}\frac{\partial h_i^L}{\partial h_j^{L-1}}\cdots\frac{\partial h_k^{l+1}}{\partial h_m^l}\frac{\partial h_m^l}{\partial\theta_n^l}\right)^2 \\
&= \sum_{all}\left(\frac{\partial\mathcal{L}}{\partial h_i^L}\frac{\partial\mathcal{L}}{\partial h_{i'}^L}\right)\left(\frac{\partial h_i^L}{\partial h_j^{L-1}}\frac{\partial h_{i'}^L}{\partial h_{j'}^{L-1}}\right)\cdots\left(\frac{\partial h_k^{l+1}}{\partial h_m^l}\frac{\partial h_{k'}^{l+1}}{\partial h_{m'}^l}\right)\left(\frac{\partial h_m^l}{\partial\theta_n^l}\frac{\partial h_{m'}^l}{\partial\theta_n^l}\right) \\
&= \sum_{all}\left(\frac{\partial\mathcal{L}}{\partial h_i^L}\frac{\partial\mathcal{L}}{\partial h_{i'}^L}\right)\left(\frac{\partial h_i^L}{\partial h_j^{L-1}}\frac{\partial h_{i'}^L}{\partial h_{j'}^{L-1}}\right)\cdots\left(\frac{\partial h_k^{l+1}}{\partial h_m^l}\frac{\partial h_{k'}^{l+1}}{\partial h_{m'}^l}\right)\delta_{mm'}\left\|\frac{\partial h^l}{\partial\theta^l}\right\|_F^2 \\
&= \sum_{all}\left(\frac{\partial\mathcal{L}}{\partial h_i^L}\frac{\partial\mathcal{L}}{\partial h_{i'}^L}\right)\left(\frac{\partial h_i^L}{\partial h_j^{L-1}}\frac{\partial h_{i'}^L}{\partial h_{j'}^{L-1}}\right)\cdots\delta_{kk'}\mathcal{J}^{l,l+1}\left\|\frac{\partial h^l}{\partial\theta^l}\right\|_F^2 \\
&= \sum_{i,i',j,j'}\left(\frac{\partial\mathcal{L}}{\partial h_i^L}\frac{\partial\mathcal{L}}{\partial h_{i'}^L}\right)\left(\frac{\partial h_i^L}{\partial h_j^{L-1}}\frac{\partial h_{i'}^L}{\partial h_{j'}^{L-1}}\right)\delta_{jj'}\cdots\mathcal{J}^{l,l+1}\left\|\frac{\partial h^l}{\partial\theta^l}\right\|_F^2 \\
&= \sum_{i,i'}\left(\frac{\partial\mathcal{L}}{\partial h_i^L}\frac{\partial\mathcal{L}}{\partial h_{i'}^L}\right)\delta_{ii'}\mathcal{J}^{L-1,L}\cdots\mathcal{J}^{l,l+1}\left\|\frac{\partial h^l}{\partial\theta^l}\right\|_F^2 \\
&= \left\|\frac{\partial\mathcal{L}}{\partial h^L}\right\|_2^2\mathcal{J}^{L-1,L}\cdots\mathcal{J}^{l,l+1}\left\|\frac{\partial h^l}{\partial\theta^l}\right\|_F^2 \\
&= \left\|\frac{\partial\mathcal{L}}{\partial h^L}\right\|_2^2\mathcal{J}^{l,L}\left\|\frac{\partial h^l}{\partial\theta^l}\right\|_F^2,
\end{aligned}
\tag{A.13}
$$

where $\|\cdot\|_2$ denotes the L_2 norm and $\|\cdot\|_F$ denotes the Frobenius norm.

A.5.3 LayerNorm on Pre-activations

Definition A.5 (Layer Normalization)**.** *The most general definition of LayerNorm involves learnable parameters γ_i^l and β_i^l.*

$$
\tilde{h}_i^l = \frac{h_i^l - \mathbb{E}[h^l]}{\sqrt{\mathbb{E}[(h^l)^2] - \mathbb{E}[h^l]^2}}\gamma_i^l + \beta_i^l\,, \tag{A.14}
$$

Hoowever, at initialization, the parameters γ_i^l and β_i^l take the values 1 and 0, respectively. This leads to the form in equation (2.18). In infinite width limit, it has the following form

$$\tilde{h}_i^l = \frac{h_i^l - \mathbb{E}_\theta[h^l]}{\sqrt{\mathbb{E}_\theta[(h^l)^2] - \mathbb{E}_\theta[h^l]^2}} \,. \tag{A.15}$$

Remark A.6. With only LayerNorm, the network recursion is given by

$$h_i^{l+1} = \sum_{j=1}^{N_l} w_{ij}^{l+1} \phi(\tilde{h}_j^l) + b_i^{l+1} \,. \tag{A.16}$$

Remark A.7. In the limit of infinite width, using the law of large numbers, the average over neurons $\mathbb{E}\left[\cdots\right]$ can be replaced by the average of parameter-initializations $\mathbb{E}_\theta\left[\cdots\right]$. Additionally, in this limit, the preactivations are i.i.d. Gaussian distributed : $h^l \sim \mathcal{N}(0, \mathcal{K}^l)$.

$$\mathbb{E}\left[h^l\right] = \mathbb{E}_\theta\left[h^l\right] = 0 \,, \tag{A.17}$$

$$\mathbb{E}\left[\left(h^l\right)^2\right] = \mathbb{E}_\theta\left[\left(h^l\right)^2\right] = \mathcal{K}^l \,. \tag{A.18}$$

The normalized preactivation then simplifies to the form of Eq.(2.18).

Lemma A.8. *With LayerNorm on preactivations, the gaussian average is modified to*

$$\mathbb{E}_\theta\left[O(\tilde{h}_i^l)\right] = \frac{1}{\sqrt{2\pi}} \int d\tilde{h}_i^l \, O(\tilde{h}_i^l) \, e^{-\frac{(\tilde{h}_i^l)^2}{2}} \,. \tag{A.19}$$

Proof. By definition $\tilde{h}_i^l$ is sampled from a standard normal distribution $\mathcal{N}(0,1)$, then use lemma 2.2 to get the final form. □

Theorem A.9. *In the infinite width limit the recurrence relation for the NNGP kernel with Lay-*

erNorm on preactivations is

$$\mathcal{K}^{l+1} = \frac{\sigma_w^2}{N_l} \sum_{j=1}^{N_l} \mathbb{E}_\theta \left[\phi(\tilde{h}_j^l) \phi(\tilde{h}_j^l) \right] + \sigma_b^2 \,. \tag{A.20}$$

Proof.

$$\begin{aligned}
\mathcal{K}^{l+1} =& \frac{1}{N_{l+1}} \sum_{i=1}^{N_{l+1}} \mathbb{E}_\theta \left[h_i^{l+1} h_i^{l+1} \right] \\
=& \frac{1}{N_{l+1}} \sum_{i=1}^{N_{l+1}} \mathbb{E}_\theta \left[\left(\sum_{j=1}^{N_l} w_{ij}^{l+1} \phi(\tilde{h}_j^l) + b_i^{l+1} \right) \left(\sum_{k=1}^{N_l} w_{ik}^{l+1} \phi(\tilde{h}_k^l) + b_i^{l+1} \right) \right] \\
=& \frac{\sigma_w^2}{N_l} \sum_{j=1}^{N_l} \mathbb{E}_\theta \left[\phi(\tilde{h}_j^l) \phi(\tilde{h}_j^l) \right] + \sigma_b^2 \,.
\end{aligned} \tag{A.21}$$

□

Theorem A.10. *In the infinite width limit the recurrence relation for partial Jacobian with LayerNorm on preactivations is*

$$\mathcal{J}^{l_0,l+1} = \chi_{\mathcal{J}}^l \mathcal{J}^{l_0,l} \,, \tag{A.22}$$

where $\chi_J^l = \frac{\sigma_w^2}{N_l \mathcal{K}^l} \sum_{i=1}^{N_l} \mathbb{E}_\theta \left[\phi'(\tilde{h}_i^l)^2 \right]$.

Proof.

$$\begin{aligned}
\mathcal{J}^{l_0,l+1} &= \frac{1}{N_{l+1}} \mathbb{E}_\theta \left[\sum_{i=1}^{N_{l+1}} \sum_{j=1}^{N_{l_0}} \frac{\partial h_i^{l+1}}{\partial h_j^{l_0}} \frac{\partial h_i^{l+1}}{\partial h_j^{l_0}} \right] \\
&= \frac{1}{N_{l+1}} \mathbb{E}_\theta \left[\sum_{i=1}^{N_{l+1}} \sum_{j=1}^{N_{l_0}} \left(\sum_{k=1}^{N_l} \frac{\partial h_i^{l+1}}{\partial \tilde{h}_k^l} \frac{\partial \tilde{h}_k^l}{\partial h_k^l} \frac{\partial h_k^l}{\partial h_j^{l_0}} \right) \left(\sum_{m=1}^{N_l} \frac{\partial h_i^{l+1}}{\partial \tilde{h}_m^l} \frac{\partial \tilde{h}_m^l}{\partial h_m^l} \frac{\partial h_m^l}{\partial h_j^{l_0}} \right) \right]
\end{aligned}$$

$$
\begin{aligned}
&= \frac{1}{N_{l+1}} \mathbb{E}_\theta \left[\sum_{i=1}^{N_{l+1}} \sum_{j=1}^{N_{l_0}} \sum_{k,m=1}^{N_l} \left(w_{ik}^{l+1} \phi'(\tilde{h}_k^l) \frac{1}{\sqrt{\mathcal{K}^l}} \right) \left(w_{im}^{l+1} \phi'(\tilde{h}_m^l) \frac{1}{\sqrt{\mathcal{K}^l}} \right) \left(\frac{\partial h_k^l}{\partial h_j^{l_0}} \frac{\partial h_m^l}{\partial h_j^{l_0}} \right) \right] \\
&= \frac{\sigma_w^2}{N_l \mathcal{K}^l} \sum_{k=1}^{N_l} \mathbb{E}_\theta \left[\phi'(\tilde{h}_k^l) \phi'(\tilde{h}_k^l) \left(\sum_{j=1}^{N_{l_0}} \frac{\partial h_k^l}{\partial h_j^{l_0}} \frac{\partial h_k^l}{\partial h_j^{l_0}} \right) \right] \\
&= \frac{\sigma_w^2}{N_l \mathcal{K}^l} \sum_{k=1}^{N_l} \mathbb{E}_\theta \left[\phi'(\tilde{h}_k^l) \phi'(\tilde{h}_k^l) \right] \mathcal{J}^{l_0,l} \\
&= \chi_{\mathcal{J}}^l \mathcal{J}^{l_0,l} \,, \qquad \text{(A.23)}
\end{aligned}
$$

□

A.5.4 LayerNorm on activations

In the main text, we only discussed LayerNorm applied to preactivations, i.e. pre-LayerNrom. Here, we discuss an alternate application of LayerNorm, which is applied after the activation fucntions – called post-LayerNorm. The general definition of LayerNorm on activations is given as follows.

Definition A.11 (post-LayerNorm)**.**

$$
\widetilde{\phi(h_i^l)} = \frac{\phi(h_i^l) - \mathbb{E}[\phi(h^l)]}{\sqrt{\mathbb{E}[\phi(h^l)^2] - \mathbb{E}[\phi(h^l)]^2}} \gamma_i^l + \beta_i^l \,. \qquad \text{(A.24)}
$$

Remark A.12. The recurrence relation for preactivations gets modified to

$$
h_i^{l+1} = \sum_{j=1}^{N_l} w_{ij}^{l+1} \widetilde{\phi(h_j^l)} + b_i^{l+1} \,. \qquad \text{(A.25)}
$$

Remark A.13. At initialization, the parameters γ_i^l and β_i^l take the values 1 and 0, respectively.

This leads to the form

$$\begin{aligned}\widetilde{\phi(h_i^l)} &= \frac{\phi(h_i^l) - \mathbb{E}[\phi(h^l)]}{\sqrt{\mathbb{E}[\phi(h^l)^2] - \mathbb{E}[\phi(h^l)]^2}} \\ &= \frac{\phi(h_i^l) - \mathbb{E}_\theta\left[\phi(h^l)\right]}{\sqrt{\mathbb{E}_\theta\left[\phi(h^l)^2\right] - \mathbb{E}_\theta\left[\phi(h^l)\right]^2}}\,,\end{aligned} \tag{A.26}$$

where the first line follows from the fact that at initialization, the parameters γ_i^l and β_i^l take the values 1 and 0 respectively. In the second line, we have invoked the infinite width limit.

Remark A.14. Evaluating the Gaussian average in this case is similar to the cases in the previous section. The only difference is that the averages are taking over the distribution $h^{l-1} \sim \mathcal{N}(0, \mathcal{K}^{l-1} = \sigma_w^2 + \sigma_b^2)$. Again this can be summarized as

$$\mathbb{E}_\theta\left[O(h_i^l)\right] = \frac{1}{\sqrt{2\pi(\sigma_w^2 + \sigma_b^2)}} \int dh_i^l\, O(h_i^l)\, e^{-\frac{(h_i^l)^2}{2(\sigma_w^2+\sigma_b^2)}}\,. \tag{A.27}$$

Next, we calculate the modifications to the recurrence relations for the NNGP kernel and Jacobians.

Theorem A.15. *In the infinite width limit the recurrence relation for the NNGP kernel with LayerNorm on activations is*

$$\mathcal{K}^{l+1} = \sigma_w^2 + \sigma_b^2\,. \tag{A.28}$$

Proof.

$$\mathcal{K}^{l+1} = \frac{1}{N_{l+1}} \sum_{i=1}^{N_{l+1}} \mathbb{E}_\theta\left[h_i^{l+1} h_i^{l+1}\right]$$

$$
\begin{aligned}
&= \frac{1}{N_{l+1}} \sum_{i=1}^{N_{l+1}} \mathbb{E}_\theta \left[\left(\sum_{j=1}^{N_l} w_{ij}^{l+1} \widetilde{\phi(h_j^l)} + b_i^{l+1} \right) \left(\sum_{k=1}^{N_l} w_{ik}^{l+1} \widetilde{\phi(h_k^l)} + b_i^{l+1} \right) \right] \\
&= \frac{\sigma_w^2}{N_l} \sum_{j=1}^{N_l} \mathbb{E}_\theta \left[\widetilde{\phi(h_j^l)}^2 \right] + \sigma_b^2 \\
&= \frac{\sigma_w^2}{N_l} \sum_{j=1}^{N_l} \mathbb{E}_\theta \left[\left(\frac{\phi(h_j^l) - \mathbb{E}_\theta \left[\phi(h^l) \right]}{\sqrt{\mathbb{E}_\theta \left[\phi(h^l)^2 \right] - \mathbb{E}_\theta \left[\phi(h^l) \right]^2}} \right)^2 \right] + \sigma_b^2 \\
&= \frac{\sigma_w^2}{N_l} \sum_{j=1}^{N_l} \frac{\mathbb{E}_\theta \left[\left(\phi(h_j^l) - \mathbb{E}_\theta \left[\phi(h^l) \right] \right)^2 \right]}{\mathbb{E}_\theta \left[\phi(h^l)^2 \right] - \mathbb{E}_\theta \left[\phi(h^l) \right]^2} + \sigma_b^2 \\
&= \sigma_w^2 + \sigma_b^2 \, .
\end{aligned} \tag{A.29}
$$

□

Theorem A.16. *In the infinite width limit the recurrence relation for partial Jacobian with LayerNorm on activations is*

$$
\mathcal{J}^{l_0, l+1} = \chi_J^l \mathcal{J}^{l_0, l} \, , \tag{A.30}
$$

where $\chi_J^l \equiv \sigma_w^2 \frac{\mathbb{E}_\theta \left[\phi'(h^l)^2 \right]}{\mathbb{E}_\theta \left[\phi(h^l)^2 \right] - \mathbb{E}_\theta \left[\phi(h^l) \right]^2}$.

Proof.

$$
\begin{aligned}
\mathcal{J}^{l_0, l+1} &= \frac{1}{N_{l+1}} \mathbb{E}_\theta \left[\sum_{i=1}^{N_{l+1}} \sum_{j=1}^{N_{l_0}} \frac{\partial h_i^{l+1}}{\partial h_j^{l_0}} \frac{\partial h_i^{l+1}}{\partial h_j^{l_0}} \right] \\
&= \frac{1}{N_{l+1}} \mathbb{E}_\theta \left[\sum_{i=1}^{N_{l+1}} \sum_{j=1}^{N_{l_0}} \left(\sum_{k=1}^{N_l} \frac{\partial h_i^{l+1}}{\partial h_k^l} \frac{\partial h_k^l}{\partial h_j^{l_0}} \right) \left(\sum_{m=1}^{N_l} \frac{\partial h_i^{l+1}}{\partial h_m^l} \frac{\partial h_m^l}{\partial h_j^{l_0}} \right) \right] \\
&= \frac{1}{N_{l+1}} \mathbb{E}_\theta \left[\sum_{i=1}^{N_{l+1}} \sum_{j=1}^{N_{l_0}} \sum_{k,m=1}^{N_l} \left(w_{ik}^{l+1} \widetilde{\phi'(h_k^l)} \right) \left(w_{im}^{l+1} \widetilde{\phi'(h_m^l)} \right) \left(\frac{\partial h_k^l}{\partial h_j^{l_0}} \frac{\partial h_m^l}{\partial h_j^{l_0}} \right) \right]
\end{aligned}
$$

$$
\begin{aligned}
&= \frac{\sigma_w^2}{N_l} \sum_{k=1}^{N_l} \sum_{j=1}^{N_{l_0}} \mathbb{E}_\theta \left[\widetilde{\phi'(h_k^l)} \widetilde{\phi'(h_k^l)} \left(\sum_{j=1}^{N_{l_0}} \frac{\partial h_k^l}{\partial h_j^{l_0}} \frac{\partial h_k^l}{\partial h_j^{l_0}} \right) \right] \\
&= \frac{\sigma_w^2}{N_l} \sum_{k=1}^{N_l} \mathbb{E}_\theta \left[\widetilde{\phi'(h_k^l)}^2 \right] \mathcal{J}^{l_0,l} \\
&= \sigma_w^2 \frac{\mathbb{E}_\theta \left[\phi'(h^l)^2 \right]}{\mathbb{E}_\theta \left[\phi(h^l)^2 \right] - \mathbb{E}_\theta \left[\phi(h^l) \right]^2} \mathcal{J}^{l_0,l} \\
&= \chi_J^l \mathcal{J}^{l_0,l} \,, \qquad \text{(A.31)}
\end{aligned}
$$

□

A.6 Residual Connections

Definition A.17. *We define residual connections by the modified the recurrence relation for pre-activations*

$$
h_i^{l+1} = \sum_{j=1}^{N_l} w_{ij}^{l+1} \phi(h_j^l) + b_i^{l+1} + \mu h_i^l \,, \qquad \text{(A.32)}
$$

where the parameter μ controls the strength of the residual connection.

Remark A.18. Note that this definition requires $N_{l+1} = N_l$. We ensure this by only adding residual connections to the hidden layers, which are of the same width. More generally, one can introduce a tensor parameter μ_{ij}.

Remark A.19. In general, the parameter μ could be layer-dependent (μ^l). But we suppress this dependence here since we are discussing self-similar networks.

Theorem A.20. *In the infinite width limit, the recurrence relation for the NNGP kernel with*

residual connections is changed by an additional term controlled by μ

$$\mathcal{K}^{l+1} = \frac{\sigma_w^2}{N_l} \sum_{j=1}^{N_l} \mathbb{E}_\theta \left[\phi(h_j^l) \phi(h_j^l) \right] + \sigma_b^2 + \mu^2 \mathcal{K}^l \,. \tag{A.33}$$

Proof.

$$\begin{aligned}
\mathcal{K}^{l+1} &= \frac{1}{N_{l+1}} \sum_{i=1}^{N_{l+1}} \mathbb{E}_\theta \left[h_i^{l+1} h_i^{l+1} \right] \\
&= \frac{1}{N_{l+1}} \sum_{i=1}^{N_{l+1}} \mathbb{E}_\theta \left[\left(\sum_{j=1}^{N_l} w_{ij}^{l+1} \phi(h_j^l) + b_i^{l+1} + \mu h_i^l \right) \left(\sum_{k=1}^{N_l} w_{ik}^{l+1} \phi(h_k^l) + b_i^{l+1} + \mu h_i^l \right) \right] \\
&= \frac{1}{N_{l+1}} \sum_{i=1}^{N_{l+1}} \mathbb{E}_\theta \left[\sum_{j=1}^{N_l} \sum_{k=1}^{N_l} w_{ij}^{l+1} w_{ik}^{l+1} \phi(h_j^l) \phi(h_k^l) + b_i^{l+1} b_i^{l+1} + \mu^2 h_i^l h_i^l \right] \\
&= \frac{1}{N_{l+1}} \sum_{i=1}^{N_{l+1}} \mathbb{E}_\theta \left[\frac{\sigma_w^2}{N_l} \sum_{j=1}^{N_l} \phi(h_j^l) \phi(h_j^l) + \sigma_b^2 \right] + \mu^2 \frac{1}{N_{l+1}} \sum_{i=1}^{N_{l+1}} \mathbb{E}_\theta \left[h_i^l h_i^l \right] \\
&= \frac{\sigma_w^2}{N_l} \sum_{j=1}^{N_l} \mathbb{E}_\theta \left[\phi(h_j^l) \phi(h_j^l) \right] + \sigma_b^2 + \mu^2 \mathcal{K}^l \,,
\end{aligned} \tag{A.34}$$

where we used the fact $N_{l+1} = N_l$ to get the last line. □

Theorem A.21. *In the infinite width limit, the recurrence relation for partial Jacobians with residual connections has a simple multiplicative form*

$$\mathcal{J}^{l_0, l+1} = \chi_{\mathcal{J}}^l \mathcal{J}^{l_0, l} \,, \tag{A.35}$$

where the recurrence coefficient is shifted to $\chi_{\mathcal{J}}^l = \sigma_w^2 \mathbb{E}_\theta \left[\phi'(h_k^l) \phi'(h_k^l) \right] + \mu^2$.

Proof.

$$
\begin{aligned}
\mathcal{J}^{l_0,l+1} &\equiv \frac{1}{N_{l+1}}\mathbb{E}_\theta\left[\sum_{i=1}^{N_{l+1}}\sum_{j=1}^{N_{l_0}}\frac{\partial h_i^{l+1}}{\partial h_j^{l_0}}\frac{\partial h_i^{l+1}}{\partial h_j^{l_0}}\right] \\
&= \frac{1}{N_{l+1}}\mathbb{E}_\theta\left[\sum_{i=1}^{N_{l+1}}\sum_{j=1}^{N_{l_0}}\left(\sum_{k=1}^{N_l}\frac{\partial h_i^{l+1}}{\partial h_k^{l}}\frac{\partial h_k^{l}}{\partial h_j^{l_0}}\right)\left(\sum_{m=1}^{N_l}\frac{\partial h_i^{l+1}}{\partial h_m^{l}}\frac{\partial h_m^{l}}{\partial h_j^{l_0}}\right)\right] \\
&= \frac{1}{N_{l+1}}\mathbb{E}_\theta\left[\sum_{i=1}^{N_{l+1}}\sum_{j=1}^{N_{l_0}}\sum_{k,m=1}^{N_l}\left(w_{ik}^{l+1}\phi'(h_k^l)+\mu\delta_{ik}\right)\left(w_{im}^{l+1}\phi'(h_m^l)+\mu\delta_{im}\right)\left(\frac{\partial h_k^{l}}{\partial h_j^{l_0}}\frac{\partial h_m^{l}}{\partial h_j^{l_0}}\right)\right] \\
&= \frac{1}{N_{l+1}}\mathbb{E}_\theta\left[\sum_{i=1}^{N_{l+1}}\sum_{j=1}^{N_{l_0}}\sum_{k,m=1}^{N_l}\left(w_{ik}^{l+1}w_{im}^{l+1}\phi'(h_k^l)\phi'(h_m^l)+\mu^2\delta_{ik}\delta_{im}\right)\frac{\partial h_k^{l}}{\partial h_j^{l_0}}\frac{\partial h_m^{l}}{\partial h_j^{l_0}}\right] \\
&= \frac{\sigma_w^2}{N_l}\sum_{k=1}^{N_l}\mathbb{E}_\theta\left[\phi'(h_k^l)\phi'(h_k^l)\left(\sum_{j=1}^{N_{l_0}}\frac{\partial h_k^{l}}{\partial h_j^{l_0}}\frac{\partial h_k^{l}}{\partial h_j^{l_0}}\right)\right]+\frac{1}{N_l}\sum_{k=1}^{N_l}\mathbb{E}_\theta\left[\mu^2\left(\sum_{j=1}^{N_{l_0}}\frac{\partial h_k^{l}}{\partial h_j^{l_0}}\frac{\partial h_k^{l}}{\partial h_j^{l_0}}\right)\right] \\
&= \left(\sigma_w^2\mathbb{E}_\theta\left[\phi'(h_k^l)\phi'(h_k^l)\right]+\mu^2\right)\mathbb{E}_\theta\left[\frac{1}{N_l}\sum_{k=1}^{N_l}\sum_{j=1}^{N_{l_0}}\frac{\partial h_k^{l}}{\partial h_j^{l_0}}\frac{\partial h_k^{l}}{\partial h_j^{l_0}}\right] \\
&= \left(\sigma_w^2\mathbb{E}_\theta\left[\phi'(h_k^l)\phi'(h_k^l)\right]+\mu^2\right)\mathcal{J}^{l_0,l} \\
\mathcal{J}^{l_0,l+1} &= \chi_{\mathcal{J}}^{l}\mathcal{J}^{l_0,l}\,. \qquad \text{(A.36)}
\end{aligned}
$$

□

A.7 Residual Connections with LayerNorm on Preactivations (Pre-LN)

We recall the recurrence relation (2.26):

$$
h_i^{l+1} = \sum_{j=1}^{N_l} w_{ij}^{l+1}\phi(\tilde{h}_j^l) + b_i^{l+1} + \mu h_i^l\,. \qquad \text{(A.37)}
$$

Theorem A.22. *In the infinite width limit, the recurrence relation for the NNGP kernel is then*

modified to

$$\mathcal{K}^{l+1} = \frac{\sigma_w^2}{N_l}\sum_{j=1}^{N_l}\mathbb{E}_\theta\left[\phi(\tilde{h}_j^l)\phi(\tilde{h}_j^l)\right] + \sigma_b^2 + \mu^2\mathcal{K}^l\,. \tag{A.38}$$

Proof.

$$\begin{aligned}
\mathcal{K}^{l+1} =& \frac{1}{N_{l+1}}\sum_{i=1}^{N_{l+1}}\mathbb{E}_\theta\left[h_i^{l+1}h_i^{l+1}\right] \\
=& \frac{1}{N_{l+1}}\sum_{i=1}^{N_{l+1}}\mathbb{E}_\theta\left[\left(\sum_{j=1}^{N_l} w_{ij}^{l+1}\phi(\tilde{h}_j^l) + b_i^{l+1} + \mu h_i^l\right)\left(\sum_{k=1}^{N_l} w_{ik}^{l+1}\phi(\tilde{h}_k^l) + b_i^{l+1} + \mu h_i^l\right)\right] \\
=& \frac{\sigma_w^2}{N_l}\sum_{j=1}^{N_l}\mathbb{E}_\theta\left[\phi(\tilde{h}_j^l)\phi(\tilde{h}_j^l)\right] + \sigma_b^2 + \mu^2\mathcal{K}^l\,.
\end{aligned} \tag{A.39}$$

□

Remark A.23. For $\mu < 1$, the recursion relation has a fixed point

$$\mathcal{K}^\star = \frac{\sigma_w^2}{N_{l^\star}(1-\mu^2)}\sum_{j=1}^{N_{l^\star}}\mathbb{E}_\theta\left[\phi(\tilde{h}_j^{l^\star})\phi(\tilde{h}_j^{l^\star})\right] + \frac{\sigma_b^2}{1-\mu^2}\,. \tag{A.40}$$

where the average here is exactly the same as cases for LayerNorm applied to preactivations without residue connections. $l^\star$ labels some very large depth l.

Remark A.24. For $\mu = 1$ case, the solution of (A.38) is

$$\mathcal{K}^l = \mathcal{K}^0 + \sum_{l'=1}^{l}\left(\frac{\sigma_w^2}{N_l}\sum_{j=1}^{N_l}\mathbb{E}_\theta\left[\phi(\tilde{h}_j^{l'})\phi(\tilde{h}_j^{l'})\right] + \sigma_b^2\right)\,. \tag{A.41}$$

which is linearly growing since the expectation does not depend on depth. $\mathcal{K}^0$ is the NNGP kernel after the input layer.

Theorem A.25. *In the infinite width limit, the recurrence relation for Jacobians changes by a*

constant shift in the recursion coefficient.

$$\mathcal{J}^{l_0,l+1} = \chi^l_{\mathcal{J}} \mathcal{J}^{l_0,l} \,, \tag{A.42}$$

where for this case

$$\chi^l_{\mathcal{J}} = \frac{\sigma_w^2}{N_l \mathcal{K}^l} \sum_{k=1}^{N_l} \mathbb{E}_\theta \left[\phi'(\tilde{h}^l_k) \phi'(\tilde{h}^l_k) \right] + \mu^2 \,. \tag{A.43}$$

Proof.

$$\begin{aligned}
\mathcal{J}^{l_0,l+1} &= \frac{1}{N_{l+1}} \mathbb{E}_\theta \left[\sum_{i=1}^{N_{l+1}} \sum_{j=1}^{N_{l_0}} \frac{\partial h^{l+1}_i}{\partial h^{l_0}_j} \frac{\partial h^{l+1}_i}{\partial h^{l_0}_j} \right] \\
&= \frac{1}{N_{l+1}} \mathbb{E}_\theta \left[\sum_{i=1}^{N_{l+1}} \sum_{j=1}^{N_{l_0}} \left(\sum_{k=1}^{N_l} \frac{\partial h^{l+1}_i}{\partial \tilde{h}^l_k} \frac{\partial \tilde{h}^l_k}{\partial h^l_k} \frac{\partial h^l_k}{\partial h^{l_0}_j} \right) \left(\sum_{m=1}^{N_l} \frac{\partial h^{l+1}_i}{\partial \tilde{h}^l_m} \frac{\partial \tilde{h}^l_m}{\partial h^l_m} \frac{\partial h^l_m}{\partial h^{l_0}_j} \right) \right] \\
&= \frac{1}{N_{l+1}} \mathbb{E}_\theta \left[\sum_{i=1}^{N_{l+1}} \sum_{j=1}^{N_{l_0}} \sum_{k,m=1}^{N_l} \left(\frac{w^{l+1}_{ik} \phi'(\tilde{h}^l_k)}{\sqrt{\mathcal{K}^l}} + \mu \delta_{ik} \right) \left(\frac{w^{l+1}_{im} \phi'(\tilde{h}^l_m)}{\sqrt{\mathcal{K}^l}} + \mu \delta_{ik} \right) \left(\frac{\partial h^l_k}{\partial h^{l_0}_j} \frac{\partial h^l_m}{\partial h^{l_0}_j} \right) \right] \\
&= \mathbb{E}_\theta \left[\left(\frac{\sigma_w^2}{N_l \mathcal{K}^l} \sum_{k=1}^{N_l} \phi'(\tilde{h}^l_k) \phi'(\tilde{h}^l_k) + \mu^2 \right) \left(\sum_{j=1}^{N_{l_0}} \frac{\partial h^l_k}{\partial h^{l_0}_j} \frac{\partial h^l_k}{\partial h^{l_0}_j} \right) \right] \\
&= \left(\frac{\sigma_w^2}{N_l \mathcal{K}^l} \sum_{k=1}^{N_l} \mathbb{E}_\theta \left[\phi'(\tilde{h}^l_k) \phi'(\tilde{h}^l_k) \right] + \mu^2 \right) \mathcal{J}^{l_0,l} \\
&= \chi^l_{\mathcal{J}} \mathcal{J}^{l_0,l} \,,
\end{aligned} \tag{A.44}$$

□

Remark A.26. Assuming $l \geq l^\star$, we combine (A.40) and (A.43) to derive Equation (2.29) in the

Theorem 2.8.

$$\chi^{*}_{\mathcal{J}} = (1-\mu^2)\frac{\frac{\sigma_w^2}{N_l}\sum_{k=1}^{N_l}\mathbb{E}_\theta\left[\phi'(\tilde{h}^l_k)\phi'(\tilde{h}^l_k)\right]}{\frac{\sigma_w^2}{N_l}\sum_{j=1}^{N_l}\mathbb{E}_\theta\left[\phi(\tilde{h}^l_j)\phi(\tilde{h}^l_j)\right]+\sigma_b^2} + \mu^2$$
$$= (1-\mu^2)\frac{A}{B} + \mu^2\,, \tag{A.45}$$

where $A = \sum_{k=1}^{N_l}\mathbb{E}_\theta\left[\phi'(\tilde{h}^l_k)\phi'(\tilde{h}^l_k)\right]$ and $B = \frac{\sigma_w^2}{N_l}\sum_{j=1}^{N_l}\mathbb{E}_\theta\left[\phi(\tilde{h}^l_j)\phi(\tilde{h}^l_j)\right] + \sigma_b^2$. Recall that $\xi = |\log\chi^{\star}_{\mathcal{J}}|^{-1}$. This directly leads us to Equation (2.29).

$$\xi = \frac{1}{|\log\left(\chi^{\star}_{\mathcal{J}}\right)|} = \frac{1}{|\log\left((1-\mu^2)\frac{A}{B}+\mu^2\right)|} \tag{A.46}$$

Note that in Equation (2.29), we discard the average over neurons in A and B, since we are in the infinite width limit.

Remark A.27. As we mentioned above $\mu = 1$ needs extra care. Plugging $\mu = 1$ into the result (A.41) and (A.43) we find out that

$$\chi^{l}_{\mathcal{J}}\,|_{\mu=1} = \frac{\sigma_w^2\sum_{k=1}^{N_l}\mathbb{E}_\theta\left[\phi'(\tilde{h}^l_k)\phi'(\tilde{h}^l_k)\right]}{N_l\mathcal{K}^0 + \sum_{l'=1}^{l}\left(\sigma_w^2\sum_{j=1}^{N_l}\mathbb{E}_\theta\left[\phi(\tilde{h}^l_j)\phi(\tilde{h}^l_j)\right] + N_l\sigma_b^2\right)} + 1$$
$$\sim 1 + O\left(\frac{1}{l}\right)\,. \tag{A.47}$$

Thus, we get everywhere criticality: critical behavior independent of the choice of initialization (σ_b, σ_w). It also follows that the correlation length ξ diverges in this case. (A.47) leads to power law behaviour in Jacobians (with exponent ζ) at large depth. Note that the exponent ζ is not universal.

Remarks A.26 and A.27 together serve as the complete proof of Theorem 2.8

A.8 Critical Exponents

We will show how to calculate the critical exponents mentioned in the main text in this section.

First, we show how to get the critical exponent of the NNGP kernel [56].

Lemma A.28. *In the infinite width limit, consider a critically initialized network with an activation function ϕ. The scaling behavior of the fluctuation $\delta\mathcal{K}^l \equiv \mathcal{K}^l - \mathcal{K}^\star$ in non-exponential. If the recurrence relation can be expand to leading order δK^l as $\delta\mathcal{K}^{l+1} \approx \delta\mathcal{K}^l - c_n(\delta\mathcal{K}^l)^n$ for $n \geq 2$. The solution of $\delta\mathcal{K}^l$ is*

$$\delta\mathcal{K}^l = \frac{1}{c_n(n-1)}\, l^{-\zeta_\mathcal{K}}\,, \tag{A.48}$$

where $\zeta_\mathcal{K} = \frac{1}{n-1}$.

Remark A.29. The constant c_n and the order of the first non-zero term n is determined by the choice of activation function.

Proof. We can expand the recurrence relation for the NNGP kernel (2.8) to the second order of $\delta\mathcal{K}^l = \mathcal{K}^l - \mathcal{K}^\star$ on both sides.

$$\delta\mathcal{K}^{l+1} \approx \delta\mathcal{K}^l - c_n(\delta\mathcal{K}^l)^n\,. \tag{A.49}$$

Use power law ansatz $\delta\mathcal{K}^l = A\, l^{-\zeta_\mathcal{K}}$ then

$$(l+1)^{-\zeta_\mathcal{K}} = l^{-\zeta_\mathcal{K}} - c_n A\, l^{-n\zeta_\mathcal{K}}\,. \tag{A.50}$$

Multiply $l^{\zeta_\mathcal{K}}$ on both side then use Taylor expansion $(\frac{l}{l+1})^{\zeta_\mathcal{K}} \approx 1 - \frac{\zeta_\mathcal{K}}{l}$

$$\frac{\zeta_\mathcal{K}}{l} = c_n A l^{-(n-1)\zeta_\mathcal{K}}\,. \tag{A.51}$$

For arbitrary l, the only non-trivial solution of the equation above is

$$A = \frac{1}{c_n(n-1)} \text{ and } \zeta_\mathcal{K} = \frac{1}{n-1}\,. \tag{A.52}$$

□

Proof of $\mathcal{J}^{l_0,l} \propto l^{-\zeta}$ scaling at critical line. We will assume $c_2 \neq 0$. Then use lemma A.28, we can expand $\chi^l_\mathcal{J}$ in terms of $\delta\mathcal{K}^l$. To leading order l^{-1}

$$\begin{aligned}\chi^l_\mathcal{J} \approx& 1 - d_1 \delta\mathcal{K}^l \\ =& 1 - \frac{d_1}{c_2} l^{-1}\,.\end{aligned} \tag{A.53}$$

Consider a sufficiently large l. In this case $O(l^{-1})$ approximation is valid. We write recurrence relations of Jacobians as

$$\mathcal{J}^{l_0,l} = \prod_{l'=l_0}^{l-1} \left(1 - \frac{d_1}{c_2} l'^{-1}\right) \mathcal{J}^{l_0,l_0}$$

$$\approx c_{l_0} \cdot l^{-\zeta} \,. \tag{A.54}$$

When $c_n = 0$ for all $n \geq 2$, from lemma A.28 we have $\delta\mathcal{K}^l = 0$. Thus the Jacobian saturates to some constant. □

We checked the scaling empirically by plotting $\mathcal{J}^{0,l}$ vs. l in a log–log plot and fitting the slope. These results are presented in Fig.2.1.

A.9 MLP-Mixer

In this section we would like to analyze an architecture called MLP-Mixer [66], which is based on multi-layer perceptrons (MLPs). A MLP-Mixer (i) chops images into patches, then applies affine transformations per patch, (ii) applies several Mixer Layers, (iii) applies pre-head LayerNorm, Global Average Pooling, and an output affine transformation. We will explain the architecture by showing forward pass equations.

Suppose one has a single input with dimension (C_{in}, H_{in}, W_{in}). We label it as $x_{\mu i}$, where the Greek letter labels channels and the Latin letter labels flattened pixels.

First of all the (i) is realized by a special convolutional layer, where kernel size f is equal to the stride s. Then, the first convolution layer can be written as

$$h^0_{\mu i} = \sum_{j=1}^{f^2} \sum_{\nu=1}^{C_{in}} W^0_{\mu\nu;j} x_{\nu,j+(i-1)s^2} + b^0_{\mu i} \,, \tag{A.55}$$

where f is the size of filter and s is the stride. In our example $f = s$. Notice in PyTorch both bias

and weights are sampled from a uniform distribution $\mathcal{U}(-\sqrt{k}, \sqrt{k})$, where $k = (C_{in} f^2)^{-1}$.

$$\mathbb{E}_\theta[W^0_{\mu\nu;i} W^0_{\rho\sigma;j}] = \frac{1}{3C_{in} f^2} \delta_{\mu\rho} \delta_{\nu\sigma} \delta_{ij} \,, \tag{A.56}$$

$$\mathbb{E}_\theta[b^0_{\mu i} b^0_{\nu j}] = \frac{1}{3C_{in} f^2} \delta_{\mu\nu} \delta_{ij} \,. \tag{A.57}$$

Notice that the output of Conv2d: $h^0_{\mu i} \in \mathbb{R}^{C \times N_p}$, where C stands for channels and $N_p = H_{in} W_{in} / f^2$ stands for patches, both of them will be mixed later by Mixer layers.

Next, we stack l Mixer Layers. A Mixer Layer contains LayerNorms and two MLPs, where the first one mixed patches i, j (token mixing) with a hidden dimension N_{tm}, the second one mixed channels μ, ν (channel-mixing) with a hidden dimension N_{cm}. Notice that for Mixer Layers we use the standard parameterization.

- First LayerNorm. It acts on channels μ.

$$\tilde{h}^{6l}_{\mu i} = \frac{h^{6l}_{\mu i} - \mathbb{E}_C[h^{6l}_{\rho i}]}{\sqrt{\mathrm{Var}_C[h^{6l}_{\rho i}]}} \,, \tag{A.58}$$

  where we defined a channel mean $\mathbb{E}_C[h^{6l}_{\rho i}] \equiv \frac{1}{C} \sum_{\rho=1}^{C} h^{6l}_{\rho i}$ and channel variance $\mathrm{Var}_C \equiv \mathbb{E}_C \left[\left(h^{6l}_{\rho i} \right)^2 \right] - \left(\mathbb{E}_C[h^{6l}_{\rho i}] \right)^2$.

- First MlpBlock. It mixes patches i, j, preactivations from different channels that share the same weight and bias.

  - $6l + 1$: Linear Affine Layer.

$$h^{6l+1}_{\mu j} = \sum_{k=1}^{N_p} w^{6l+1}_{jk} \tilde{h}^{6l}_{\mu k} + b^{6l+1}_j \,. \tag{A.59}$$

– $6l+2$: Affine Layer.

$$h_{\mu i}^{6l+2} = \sum_{j=1}^{N_{tm}} w_{ij}^{6l+2} \phi(h_{\mu j}^{6l+1}) + b_i^{6l+2}\,, \tag{A.60}$$

where N_{tm} stands for the hidden dimension of "token mixing".

– $6l+3$: Residual Connections.

$$h_{\mu i}^{6l+3} = h_{\mu i}^{6l+2} + \mu h_{\mu i}^{6l}\,. \tag{A.61}$$

- Second LayerNorm. It again acts on channels μ.

$$\tilde{h}_{\mu i}^{6l+3} = \frac{h_{\mu i}^{6l+3} - \mathbb{E}_C[h_{\rho i}^{6l+3}]}{\sqrt{\mathrm{Var}_C[h_{\rho i}^{6l+3}]}}\,. \tag{A.62}$$

- Second MlpBlock. It mixes channels μ, ν, preactivations from different patches share the same weight and bias.

– $6l+4$: Linear Affine Layer.

$$h_{\nu i}^{6l+4} = \sum_{\rho=1}^{C} w_{\nu\rho}^{6l+4} \tilde{h}_{\rho i}^{6l+3} + b_\nu^{6l+4}\,. \tag{A.63}$$

– $6l+5$. Affine Layer.

$$h_{\mu i}^{6l+5} = \sum_{\nu=1}^{N_{cm}} w_{\mu\nu}^{6l+5} \phi(h_{\nu i}^{6l+4}) + b_\mu^{6l+5}\,. \tag{A.64}$$

– $6l + 6$. Residual Connections.

$$h_{\mu i}^{6l+6} = h_{\mu i}^{6l+5} + \mu h_{\mu i}^{6l+3}. \tag{A.65}$$

Suppose the network has L Mixer layers. After those layers the network has a pre-head LayerNorm layer, a global average pooling layer, and an output layer. The pre-head LayerNorm normalizes over channels μ can be described as the following

$$\tilde{h}_{\mu i}^{6L} = \frac{h_{\mu i}^{6L} - \mathbb{E}_C[h_{\rho i}^{6L}]}{\sqrt{\mathrm{Var}_C[h_{\rho i}^{6L}]}}\,. \tag{A.66}$$

Global Average Pool over patches i.

$$h_{\mu}^{p} = \frac{1}{N_p}\sum_{i=1}^{N_p} \tilde{h}_{\mu i}^{6L}\,. \tag{A.67}$$

Output Layer

$$f_{\mu} = \sum_{\nu=1}^{C} w_{\mu\nu} h_{\nu}^{p} + b_{\mu}\,. \tag{A.68}$$

We plotted the phase diagram using the following quantity from repeating Mixer Layers:

$$\chi_{\mathcal{J}}^{\star} = \lim_{L\to\infty}\left(\frac{1}{N_p C}\sum_{i=1}^{N_p}\sum_{\mu=1}^{C}\mathbb{E}_{\theta}\left[\sum_{\rho=1}^{C}\sum_{k=1}^{N_p}\frac{\partial h_{\mu i}^{6L}}{\partial h_{\rho k}^{6L-6}}\frac{\partial h_{\mu i}^{6L}}{\partial h_{\rho k}^{6L-6}}\right]\right)\,. \tag{A.69}$$

A.10 Results for Scale Invariant Activation Functions

Definition A.30 (Scale invariant activation functions)**.**

$$\phi(x) = a_+ \, x \, \Theta(x) + a_- \, x \, \Theta(-x) \, , \tag{A.70}$$

where $\Theta(x)$ *is the Heaviside step function. ReLU is the special case with* $a_+ = 1$ *and* $a_- = 0$*.*

A.10.1 NNGP Kernel

First evaluate the average using lemma 2.2

$$\begin{aligned} \mathbb{E}_\theta \left[\phi(h_i^l) \phi(h_i^l) \right] =& \frac{1}{\sqrt{2\pi \mathcal{K}^l}} \int dh_i^l \left(a_+^2 + a_-^2 \right) \left(h_i^l \right)^2 e^{-\frac{(h_i^l)^2}{2K^l}} \\ =& \frac{a_+^2 + a_-^2}{2} \mathcal{K}^l \, . \end{aligned} \tag{A.71}$$

Thus we obtain the recurrence relation for the NNGP kernel with scale invariant activation function.

$$\mathcal{K}^{l+1} = \frac{\sigma_w^2 (a_+^2 + a_-^2)}{2} \mathcal{K}^l + \sigma_b^2 \, . \tag{A.72}$$

Finite fixed point of the recurrence relation above exists only if

$$\chi_{\mathcal{K}}^\star = \frac{\sigma_w^2 (a_+^2 + a_-^2)}{2} \leq 1 \, . \tag{A.73}$$

As a result

$$\sigma_w^2 \leq \frac{2}{a_+^2 + a_-^2} \, . \tag{A.74}$$

For $\sigma_w^2 = \frac{2}{a_+^2 + a_-^2}$ case, finite fixed point exists only if $\sigma_b^2 = 0$.

A.10.2 Jacobian(s)

The calculation is quite straightforward, by definition

$$\begin{aligned}\chi_{\mathcal{J}}^l =& \sigma_w^2 \mathbb{E}_\theta \left[\phi'(h_i^l)\phi'(h_i^l)\right] \\ =& \frac{\sigma_w^2}{\sqrt{2\pi\mathcal{K}^l}} \int dh_i^l \left[a_+\Theta(h_i^l) - a_-\Theta(h_i^l)\right]^2 e^{-\frac{(h_i^l)^2}{2\mathcal{K}^l}} \\ =& \frac{\sigma_w^2(a_+^2 + a_-^2)}{2}\,, \end{aligned} \tag{A.75}$$

where we used the property $x\delta(x) = 0$ for Dirac's delta function to get the first line.

Thus the critical line is defined by

$$\sigma_w = \sqrt{\frac{2}{a_+^2 + a_-^2}}\,. \tag{A.76}$$

For ReLU with $a_+ = 1$ and $a_- = 0$, the network is at critical line when

$$\sigma_w = \sqrt{2}\,, \tag{A.77}$$

where the critical point is located at

$$(\sigma_w, \sigma_b) = (\sqrt{2}, 0)\,. \tag{A.78}$$

A.10.3 Critical Exponents

Since the recurrence relations for the NNGP kernel and Jacobians are linear. Then from lemma A.28 and $\mathcal{J}^{l_0,l} \propto l^{-\zeta}$

$$\zeta_{\mathcal{K}} = 0 \text{ and } \zeta = 0\,. \tag{A.79}$$

A.10.4 LayerNorm on Pre-activations

Use lemma A.8 and combine all known results for scale-invariant functions

$$\begin{aligned}\chi^l_{\mathcal{J}} &= \frac{\sigma_w^2}{N_l \mathcal{K}^l} \sum_{k=1}^{N_l} \mathbb{E}_\theta \left[\phi'(\tilde{h}^l_k)\phi'(\tilde{h}^l_k)\right]\Big|_{\tilde{\mathcal{K}}^{l-1}=1} \\ &= \frac{\sigma_w^2(a_+^2 + a_-^2)}{\sigma_w^2(a_+^2 + a_-^2) + 2\sigma_b^2}\,.\end{aligned} \tag{A.80}$$

In this case,

$$\chi^l_{\mathcal{J}} \leq 1 \tag{A.81}$$

is always true. The equality only holds at $\sigma_b = 0$ line.

A.10.5 LayerNorm on Activations

First we substitute $\mathcal{K}^{l-1} = \sigma_w^2 + \sigma_b^2$ into known results

$$\mathbb{E}_\theta \left[\phi'(h^l_i)\phi'(h^l_i)\right] = \frac{a_+^2 + a_-^2}{2}\,, \tag{A.82}$$

$$\mathbb{E}_\theta \left[\phi(h^l_i)\phi(h^l_i)\right] = \frac{a_+^2 + a_-^2}{2}(\sigma_w^2 + \sigma_b^2)\,. \tag{A.83}$$

There is a new expectation value we need to show explicitly

$$\begin{aligned}\mathbb{E}_\theta\left[\phi(h_i^l)\right] &= \frac{1}{\sqrt{2\pi(\sigma_w^2+\sigma_b^2)}}\int_{-\infty}^{\infty} dh_i^l \phi(h_i^l) e^{-\frac{1}{2}h_i^l(\sigma_w^2+\sigma_b^2)^{-1}h_i^l} \\ &= \frac{1}{\sqrt{2\pi(\sigma_w^2+\sigma_b^2)}}\int_{0}^{\infty} dh_i^l (a_+ - a_-) h_i^l e^{-\frac{(h_i^l)^2}{2(\sigma_w^2+\sigma_b^2)}} \\ &= (a_+ - a_-)\sqrt{\frac{\sigma_w^2+\sigma_b^2}{2\pi}}\,. \end{aligned} \tag{A.84}$$

Thus

$$\chi_{\mathcal{J}}^l = \frac{\sigma_w^2}{\sigma_w^2+\sigma_b^2}\cdot\frac{\pi(a_+^2+a_-^2)}{\pi(a_+^2+a_-^2)-(a_+-a_-)^2}\,. \tag{A.85}$$

The critical line is defined by $\chi_{\mathcal{J}}^\star = 1$, which can be solved as

$$\sigma_b = \sqrt{\frac{(a_+-a_-)^2}{\pi(a_+^2+a_-^2)-(a_+-a_-)^2}}\sigma_w\,. \tag{A.86}$$

For ReLU with $a_+ = 1$ and $a_- = 0$

$$\begin{aligned}\sigma_b =& \sqrt{\frac{1}{\pi-1}}\sigma_w \\ \approx& 0.683\sigma_w\,. \end{aligned} \tag{A.87}$$

A.10.6 Residual Connections

The recurrence relation for the NNGP kernel can be evaluated to be

$$\mathcal{K}^{l+1} = \frac{\sigma_w^2(a_+^2+a_-^2)}{2}\mathcal{K}^l + \sigma_b^2 + \mu^2\mathcal{K}^l\,. \tag{A.88}$$

The condition for the existence of fixed point

$$\chi_{\mathcal{K}}^{\star} = \frac{\sigma_w^2(a_+^2 + a_-^2)}{2} + \mu^2 \leq 1 \tag{A.89}$$

leads us to

$$\sigma_w^2 \leq \frac{2(1-\mu^2)}{a_+^2 + a_-^2}\,. \tag{A.90}$$

For $\sigma_w^2 = \frac{2(1-\mu^2)}{a_+^2+a_-^2}$, finite fixed point exists only if $\sigma_b^2 = 0$. (Diverges linearly otherwise)

The recurrence coefficient for Jacobian is evaluated to be

$$\chi_{\mathcal{J}}^{\star} = \frac{\sigma_w^2(a_+^2 + a_-^2)}{2} + \mu^2\,. \tag{A.91}$$

The critical line is defined as

$$\sigma_w = \sqrt{\frac{2(1-\mu^2)}{a_+^2 + a_-^2}}\,. \tag{A.92}$$

The critical point is located at $\left(\sqrt{\frac{2(1-\mu^2)}{a_+^2+a_-^2}}, 0\right)$.

For ReLU, the critical point is at $\left(\sqrt{2(1-\mu^2)}, 0\right)$.

A.10.7 Residual Connections with LayerNorm on Preactivations (Pre-LN)

Again use lemma A.8 and combine all known results for scale-invariant functions

$$\begin{aligned}\chi_{\mathcal{J}}^{\star} &= \lim_{l\to\infty}\left(\frac{\sigma_w^2}{N_l \mathcal{K}^l}\sum_{k=1}^{N_l}\mathbb{E}_\theta\left[\phi'(\tilde{h}_k^l)\phi'(\tilde{h}_k^l)\right]\Big|_{\tilde{\mathcal{K}}^{l-1}=1} + \mu^2\right) \\ &= \frac{\sigma_w^2(a_+^2 + a_-^2)(1-\mu^2)}{\sigma_w^2(a_+^2 + a_-^2) + 2\sigma_b^2} + \mu^2\end{aligned}$$

$$= 1 - \frac{2\sigma_b^2(1-\mu^2)}{\sigma_w^2(a_+^2 + a_-^2) + 2\sigma_b^2} \tag{A.93}$$

Similar to the case without residue connections

$$\chi_{\mathcal{J}}^l \leq 1 \tag{A.94}$$

is always true. The equality only holds at $\sigma_b = 0$ line for $\mu < 1$.

Notice there is a very special case $\mu = 1$, where the whole $\sigma_b - \sigma_w$ plane is critical.

A.11 Results for erf Activation Function

Definition A.31 (erf activation function)**.**

$$\phi(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt \,. \tag{A.95}$$

A.11.1 NNGP Kernel

To evaluate lemma 2.2 exactly, we introduce two dummy variables λ_1 and λ_2[176].

$$\begin{aligned}
\mathbb{E}_\theta \left[\phi(\lambda_1 h_i^l)\phi(\lambda_2 h_i^l)\right] &= \int d\lambda_1 \int d\lambda_2 \frac{d^2}{d\lambda_1 d\lambda_2} \mathbb{E}_\theta \left[\phi(\lambda_1 h_i^l)\phi(\lambda_2 h_i^l)\right] \\
&= \int d\lambda_1 \int d\lambda_2 \int dh_i^l \frac{4}{\sqrt{2\pi^3 \mathcal{K}^l}} \left(h_i^l\right)^2 e^{-\left(\lambda_1^2+\lambda_2^2+\frac{1}{2\mathcal{K}^l}\right)\left(h_i^l\right)^2} \\
&= \int d\lambda_1 \int d\lambda_2 \frac{4\mathcal{K}^l}{\pi\left(1 + 2\mathcal{K}^l(\lambda_1^2+\lambda_2^2)\right)} \\
&= \frac{2}{\pi} \arcsin\left(\frac{2\mathcal{K}^l \lambda_1 \lambda_2}{1 + 2\mathcal{K}^l(\lambda_1^2+\lambda_2^2)}\right) .
\end{aligned} \tag{A.96}$$

We use the special case where $\lambda_1 = \lambda_2 = 1$.

Thus the recurrence relation for the NNGP kernel with erf activation function is

$$\mathcal{K}^{l+1} = \frac{2\sigma_w^2}{\pi} \arcsin\left(\frac{2\mathcal{K}^l}{1+2\mathcal{K}^l}\right) + \sigma_b^2 \,. \tag{A.97}$$

As in the scale-invariant case, finite fixed point only exists when

$$\chi_{\mathcal{K}}^\star = \frac{4\sigma_w^2}{\pi} \frac{1}{(1+2\mathcal{K}^\star)\sqrt{1+4\mathcal{K}^\star}} \leq 1 \,. \tag{A.98}$$

Numerical results show the condition is satisfied everywhere in $\sigma_b - \sigma_w$ plane, where $\chi_{\mathcal{K}}^\star = 1$ is only possible when $\mathcal{K}^\star = 0$.

A.11.2 Jacobians

Follow the definition

$$\begin{aligned}
\chi_{\mathcal{J}}^l &= \sigma_w^2 \mathbb{E}_\theta \left[\phi'(h_i^l)\phi'(h_i^l)\right] \\
&= \frac{4\sigma_w^2}{\sqrt{2\pi^3 \mathcal{K}^l}} \int dh_i^l \, e^{-2(h_i^l)^2} \, e^{-\frac{(h_i^l)^2}{2\mathcal{K}^l}} \\
&= \frac{4\sigma_w^2}{\pi} \frac{1}{\sqrt{1+4\mathcal{K}^l}} \,.
\end{aligned} \tag{A.99}$$

To find phase boundary $\chi_{\mathcal{J}}^\star = 1$, we need to combine Eq.(A.97) and Eq.(A.99) and evaluate them at $\mathcal{K}^\star$.

$$\mathcal{K}^\star = \frac{2\sigma_w^2}{\pi} \arcsin\left(\frac{2\mathcal{K}^\star}{1+2\mathcal{K}^\star}\right) + \sigma_b^2 \,, \tag{A.100}$$

$$\chi^\star_{\mathcal{J}} = \frac{4\sigma_w^2}{\pi}\frac{1}{\sqrt{1+4\mathcal{K}^\star}} = 1\,. \tag{A.101}$$

One can solve the equations above and find the critical line

$$\sigma_b = \sqrt{\frac{16\sigma_w^4-\pi^2}{4\pi^2} - \frac{2\sigma_w^2}{\pi}\arcsin\left(\frac{16\sigma_w^4-\pi^2}{16\sigma_w^4+\pi^2}\right)}\,. \tag{A.102}$$

Critical point is reached by further requiring $\chi^\star_{\mathcal{K}} = 1$. Since $\chi^\star_{\mathcal{K}} \leq \chi^\star_{\mathcal{J}}$, the only possible case is $\mathcal{K}^\star = 0$, which is located at

$$(\sigma_w, \sigma_b) = \left(\sqrt{\frac{\pi}{4}}, 0\right)\,. \tag{A.103}$$

A.11.3 Critical Exponents

We show how to extract critical exponents of the NNGP kernel and Jacobians of erf activation function.

Critical point for erf is at $(\sigma_b, \sigma_w) = (0, \sqrt{\frac{\pi}{4}})$, with $\mathcal{K}^\star = 0$. Now suppose l is large enough such that the deviation of $\mathcal{K}^l$ from fixed point value $\mathcal{K}^\star$ is small. Define $\delta\mathcal{K}^l \equiv \mathcal{K}^l - \mathcal{K}^\star$. Eq.(A.97) can be rewritten as

$$\begin{aligned}\delta\mathcal{K}^{l+1} =& \frac{1}{2}\arcsin\left(\frac{2\delta\mathcal{K}^l}{1+2\delta\mathcal{K}^l}\right)\\ \approx& \delta\mathcal{K}^l - 2(\delta\mathcal{K}^l)^2\,.\end{aligned} \tag{A.104}$$

From lemma A.28

$$A = \frac{1}{2} \text{ and } \zeta_{\mathcal{K}} = 1\,. \tag{A.105}$$

Next we analyze critical exponent of Jacobians by expanding (A.99) around $\mathcal{K}^\star = 0$ critical point $(\sigma_b, \sigma_w) = (0, \sqrt{\frac{\pi}{4}})$.

To leading order l^{-1} we have

$$\begin{aligned} \chi^l_{\mathcal{J}} \approx & 1 - 2\delta K^l \\ \approx & 1 - \frac{1}{l} \,. \end{aligned} \tag{A.106}$$

Thus the recurrence relation for partial Jacobian, at large l, takes form

$$\mathcal{J}^{l_0,l+1} = \left(1 - \frac{1}{l}\right) \mathcal{J}^{l_0,l} \,. \tag{A.107}$$

At large l

$$\mathcal{J}^{l_0,l} = c_{l_0}\, l^{-1} \,, \tag{A.108}$$

with a non-universal constant c_{l_0}.

The critical exponent is

$$\zeta = 1 \,, \tag{A.109}$$

which is the same as $\zeta_{\mathcal{K}}$.

A.11.4 LayerNorm on Pre-activations

Use lemma A.8, we have

$$\chi^l_{\mathcal{J}} = \frac{\sigma_w^2}{N_l \mathcal{K}^l} \sum_{k=1}^{N_l} \mathbb{E}_\theta \left[\phi'(\tilde{h}^l_k) \phi'(\tilde{h}^l_k) \right] \Big|_{\tilde{\mathcal{K}}^{l-1}=1}$$

$$= \frac{4\sigma_w^2}{\sqrt{5}\left[2\sigma_w^2 \arcsin\left(\frac{2}{3}\right) + \pi\sigma_b^2\right]} \,. \tag{A.110}$$

The critical line is then defined by

$$\sigma_b = \sqrt{\frac{2}{\pi}\left[\frac{2}{\sqrt{5}} - \arcsin\left(\frac{2}{3}\right)\right]}\sigma_w$$

$$\approx 0.324\sigma_w \,. \tag{A.111}$$

A.11.5 LayerNorm on Activations

Due to the symmetry of erf activation function $\mathbb{E}_\theta\left[\phi(h_i^l)\right] = 0$, we only need to modify our known results.

$$\mathbb{E}_\theta\left[\phi'(h_i^l)\phi'(h_i^l)\right] = \frac{4}{\pi}\frac{1}{\sqrt{1 + 4(\sigma_w^2 + \sigma_b^2)}} \,, \tag{A.112}$$

$$\mathbb{E}_\theta\left[\phi(h_i^l)\phi(h_i^l)\right] = \frac{2}{\pi}\arcsin\left(\frac{2(\sigma_w^2 + \sigma_b^2)}{1 + 2(\sigma_w^2 + \sigma_b^2)}\right) \,. \tag{A.113}$$

Thus

$$\chi_{\mathcal{J}}^l = \frac{2\sigma_w^2}{\sqrt{1 + 4(\sigma_w^2 + \sigma_b^2)}} \cdot \frac{1}{\arcsin\left(\frac{2(\sigma_w^2 + \sigma_b^2)}{1 + 2(\sigma_w^2 + \sigma_b^2)}\right)} \,, \tag{A.114}$$

where the phase boundary is defined by the transcendental equation $\chi_{\mathcal{J}}^l = 1$.

A.11.6 Residual Connections

The recurrence relation for the NNGP kernel can be evaluated to be

$$\mathcal{K}^{l+1} = \frac{2\sigma_w^2}{\pi} \arcsin\left(\frac{2\mathcal{K}^l}{1+2\mathcal{K}^l}\right) + \sigma_b^2 + \mu^2 \mathcal{K}^l \,. \tag{A.115}$$

Finite fixed point only exists when

$$\chi_{\mathcal{K}}^\star = \frac{4\sigma_w^2}{\pi} \frac{1}{(1+2\mathcal{K}^\star)\sqrt{1+4\mathcal{K}^\star}} + \mu^2 \leq 1 \,. \tag{A.116}$$

Notice that $\chi_{\mathcal{K}}^\star \leq \chi_{\mathcal{J}}^\star$ still holds, where the equality holds only when $\mathcal{K}^\star = 0$.

The recurrence coefficient for Jacobian is evaluated to be

$$\chi_{\mathcal{J}}^\star = \frac{4\sigma_w^2}{\pi} \frac{1}{\sqrt{1+4\mathcal{K}^\star}} + \mu^2 \,. \tag{A.117}$$

The critical line is defined as

$$\sigma_b = \sqrt{\frac{16\sigma_w^4 - \pi^2(1-\mu^2)^2}{4\pi^2(1-\mu^2)} - \frac{2\sigma_w^2}{\pi} \arcsin\left(\frac{16\sigma_w^4 - \pi^2(1-\mu^2)^2}{16\sigma_w^4 + \pi^2(1-\mu^2)^2}\right)} \,. \tag{A.118}$$

Critical point is reached by further requiring $\chi_{\mathcal{K}}^\star = 1$. Since $\chi_{\mathcal{K}}^\star \leq \chi_{\mathcal{J}}^\star$, the only possible case is $\mathcal{K}^\star = 0$, which is located at

$$(\sigma_w, \sigma_b) = \left(\sqrt{\frac{\pi(1-\mu^2)}{4}}, 0\right) \,. \tag{A.119}$$

Note that for $\mu = 1$, one needs to put extra effort into analyzing the scaling behavior. First

we notice that $\mathcal{K}^l$ monotonically increases with depth l – the recurrence relation for the NNGP kernel at large l (or large $\mathcal{K}^l$) is

$$\mathcal{K}^{l+1} \approx \sigma_w^2 + \sigma_b^2 + \mathcal{K}^l \,, \tag{A.120}$$

which regulates the first term in (A.117).

For $\mu = 1$ at large depth

$$\chi_{\mathcal{J}}^l \sim 1 + \frac{4\sigma_w^2}{\pi\sqrt{C_0 + 4(\sigma_w^2 + \sigma_b^2)l}} \,. \tag{A.121}$$

Here C_0 is a constant that depends on the input.

We can approximate the asymptotic form of $\log \mathcal{J}^{l_0,l}$ as follows

$$\begin{aligned}
\log \mathcal{J}^{l_0,l} &= \log\left(\prod_{l'=l_0}^{l} \chi_{\mathcal{J}}^{l'}\right) \\
&= \sum_{l'=l_0}^{l} \log\left(1 + \frac{4\sigma_w^2}{\pi\sqrt{C_0 + 4(\sigma_w^2 + \sigma_b^2)l'}}\right) \\
&\approx \int_{l_0}^{l} dl' \log\left(1 + \frac{4\sigma_w^2}{\pi\sqrt{C_0 + 4(\sigma_w^2 + \sigma_b^2)l'}}\right) \\
&\sim 2\tilde{c}\sqrt{l} + O(\log l) \,,
\end{aligned} \tag{A.122}$$

where $\tilde{c} = \frac{2\sigma_w^2}{\pi\sqrt{\sigma_w^2+\sigma_b^2}}$.

We conclude that at large depth, the APJN for $\mu = 1$, erf networks can be written as

$$\mathcal{J}^{l_0,l} \sim O\left(e^{2\tilde{c}\sqrt{l}+O(\log l)}\right) \,. \tag{A.123}$$

This result checks out empirically, as shown in Figure A.3.[1]

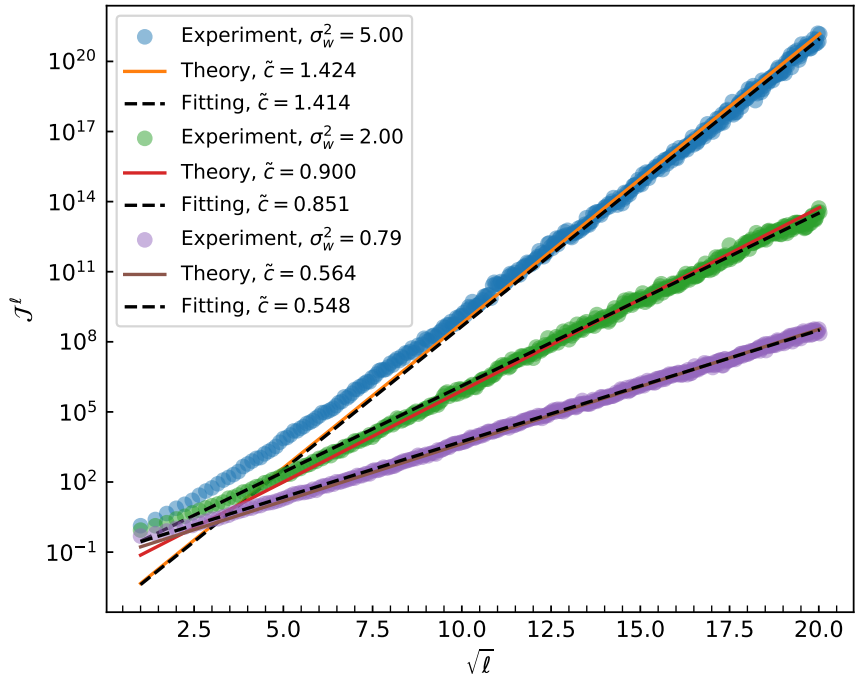


Figure A.3: $\log(\mathcal{J}^{l_0,l})$-$\sqrt{l}$ for $\mu = 1$, $\sigma_b^2 = 0$, erf.

A.11.7 Residual Connections with LayerNorm on Preactivations (Pre-LN)

Use lemma A.8 and the results we had without residue connections for erf with LayerNorm on preactivations.

$$\begin{aligned}\chi_{\mathcal{J}}^{*} &= \lim_{l\to\infty}\left(\frac{\sigma_w^2}{N_l\mathcal{K}^l}\sum_{k=1}^{N_l}\mathbb{E}_\theta\left[\phi'(\tilde{h}_k^l)\phi'(\tilde{h}_k^l)\right]\Big|_{\tilde{\mathcal{K}}^{l-1}=1} + \mu^2\right)\\ &= \frac{4\sigma_w^2(1-\mu^2)}{\sqrt{5}\left[2\sigma_w^2\arcsin\left(\frac{2}{3}\right)+\pi\sigma_b^2\right]} + \mu^2\,. \end{aligned} \tag{A.124}$$

[1] We used NTK parameterization for this experiment. However, we emphasize that it does not affect the final result.

The critical line is then defined by

$$\begin{aligned}\sigma_b &= \sqrt{\frac{2}{\pi}\left[\frac{2}{\sqrt{5}} - \arcsin\left(\frac{2}{3}\right)\right]}\sigma_w \\ &\approx 0.324\sigma_w\,.\end{aligned} \tag{A.125}$$

A.12 Results for GELU Activation Function

Definition A.32 (GELU activation function)**.**

$$\begin{aligned}\phi(x) =& \frac{x}{2}\left[1 + erf\left(\frac{x}{\sqrt{2}}\right)\right] \\ =& \frac{x}{2}\left[1 + \frac{2}{\sqrt{\pi}}\int_0^{\frac{x}{\sqrt{2}}} e^{-t^2}dt\right]\,.\end{aligned} \tag{A.126}$$

A.12.1 NNGP Kernel

Use lemma 2.2 for GELU

$$\begin{aligned}\mathbb{E}_\theta\left[\phi(h_i^l)\phi(h_i^l)\right] =& \frac{1}{\sqrt{2\pi\mathcal{K}^l}}\int dh_i^l \frac{(h_i^l)^2}{4}\left[1 + \text{erf}\left(\frac{h_i^l}{\sqrt{2}}\right)\right]^2 e^{-\frac{(h_i^l)^2}{2\mathcal{K}^l}} \\ =& \frac{1}{\sqrt{2\pi\mathcal{K}^l}}\int dh_i^l \frac{(h_i^l)^2}{4}\left[1 + \text{erf}^2\left(\frac{h_i^l}{\sqrt{2}}\right)\right] e^{-\frac{(h_i^l)^2}{2\mathcal{K}^l}} \\ =& \frac{\mathcal{K}^l}{4} + \frac{1}{\sqrt{32\pi\mathcal{K}^l}}\int dh_i^l\,(h_i^l)^2\text{erf}^2\left(\frac{h_i^l}{\sqrt{2}}\right) e^{-\frac{(h_i^l)^2}{2\mathcal{K}^l}} \\ =& \frac{\mathcal{K}^l}{4} + \frac{\mathcal{K}^l}{\sqrt{32\pi\mathcal{K}^l}}\int dh_i^l\,\text{erf}^2\left(\frac{h_i^l}{\sqrt{2}}\right) e^{-\frac{(h_i^l)^2}{2\mathcal{K}^l}} \\ &+ \frac{(\mathcal{K}^l)^2}{\sqrt{32\pi\mathcal{K}^l}}\int dh_i^l\left[\text{erf}'\left(\frac{h_i^l}{\sqrt{2}}\right)\text{erf}'\left(\frac{h_i^l}{\sqrt{2}}\right) + \text{erf}\left(\frac{h_i^l}{\sqrt{2}}\right)\text{erf}''\left(\frac{h_i^l}{\sqrt{2}}\right)\right] e^{-\frac{(h_i^l)^2}{2\mathcal{K}^l}} \\ =& \frac{\mathcal{K}^l}{4} + \frac{\mathcal{K}^l}{2\pi}\left[\arcsin\left(\frac{\mathcal{K}^l}{1+\mathcal{K}^l}\right) + \frac{2\mathcal{K}^l}{(1+\mathcal{K}^l)\sqrt{1+2\mathcal{K}^l}}\right]\,,\end{aligned} \tag{A.127}$$

where from the third line to the fourth line we used integrate by parts twice, and to get the last line we used results from erf activations.

Thus the recurrence relation for the NNGP kernel is

$$\mathcal{K}^{l+1} = \left[\frac{\mathcal{K}^l}{4} + \frac{\mathcal{K}^l}{2\pi}\arcsin\left(\frac{\mathcal{K}^l}{1+\mathcal{K}^l}\right) + \frac{(\mathcal{K}^l)^2}{\pi(1+\mathcal{K}^l)\sqrt{1+2\mathcal{K}^l}}\right]\sigma_w^2 + \sigma_b^2\,. \tag{A.128}$$

As a result

$$\chi_{\mathcal{K}}^{\star} = \frac{\sigma_w^2}{4} + \frac{\sigma_w^2}{2\pi}\left[\arcsin\left(\frac{\mathcal{K}^\star}{1+\mathcal{K}^\star}\right) + \frac{4(\mathcal{K}^\star)^3 + 11(\mathcal{K}^\star)^2 + 5\mathcal{K}^\star}{(1+\mathcal{K}^\star)^2(1+2\mathcal{K}^\star)^{\frac{3}{2}}}\right]\,. \tag{A.129}$$

A.12.2 Jacobians

Follow the definition

$$\begin{aligned}
\chi_{\mathcal{J}}^l =& \sigma_w^2\mathbb{E}_\theta\left[\phi'(h_i^l)\phi'(h_i^l)\right] \\
=& \frac{\sigma_w^2}{\sqrt{2\pi\mathcal{K}^l}}\int dh_i^l\left[\frac{1}{2} + \frac{1}{2}\text{erf}\left(\frac{h_i^l}{\sqrt{2}}\right) + \frac{e^{-\frac{(h_i^l)^2}{2}}h_i^l}{\sqrt{2\pi}}\right]^2 e^{-\frac{(h_i^l)^2}{2\mathcal{K}^l}} \\
=& \frac{\sigma_w^2}{\sqrt{2\pi\mathcal{K}^l}}\int dh_i^l\left[\frac{1}{4} + \frac{1}{4}\text{erf}\left(\frac{h_i^l}{\sqrt{2}}\right)^2 + \frac{h_i^l\text{erf}\left(\frac{h_i^l}{\sqrt{2}}\right)e^{-\frac{(h_i^l)^2}{2}}}{\sqrt{2\pi}} + \frac{e^{-(h_i^l)^2}(h_i^l)^2}{2\pi}\right] e^{-\frac{(h_i^l)^2}{2\mathcal{K}^l}} \\
=& \frac{\sigma_w^2}{4} + \frac{\sigma_w^2}{2\pi}\left[\arcsin\left(\frac{\mathcal{K}^l}{1+\mathcal{K}^l}\right) + \frac{\mathcal{K}^l(3+5\mathcal{K}^l)}{(1+\mathcal{K}^l)(1+2\mathcal{K}^l)^{\frac{3}{2}}}\right]\,,
\end{aligned} \tag{A.130}$$

where we dropped odd function terms to get the third line, and to get the last line we used known result for erf in the second term, integration by parts in the third term.

Here to get the critical line is harder. One can use the recurrence relation for the NNGP

kernel at fixed point $\mathcal{K}^\star$ and $\chi^\star_{\mathcal{J}} = 1$

$$\mathcal{K}^\star = \frac{\sigma_w^2}{4}\mathcal{K}^\star + \frac{\sigma_w^2}{2\pi}\left[\arcsin\left(\frac{\mathcal{K}^\star}{1+\mathcal{K}^\star}\right) + \frac{\sigma_w^2\mathcal{K}^\star}{\pi(1+\mathcal{K}^\star)\sqrt{1+2\mathcal{K}^\star}}\right]\mathcal{K}^\star + \sigma_b^2\,, \tag{A.131}$$

$$\chi^\star_{\mathcal{J}} = \frac{\sigma_w^2}{4} + \frac{\sigma_w^2}{2\pi}\left[\arcsin\left(\frac{\mathcal{K}^\star}{1+\mathcal{K}^\star}\right) + \frac{\mathcal{K}^\star(3+5\mathcal{K}^\star)}{(1+\mathcal{K}^\star)(1+2\mathcal{K}^\star)^{\frac{3}{2}}}\right] = 1\,. \tag{A.132}$$

Cancel the $\arcsin$ term, σ_w and σ_b then can be written as a function of $\mathcal{K}^\star$

$$\sigma_w = 2\left[1 + \frac{2\mathcal{K}^\star(3+5\mathcal{K}^\star)}{\pi(1+\mathcal{K}^\star)(1+2\mathcal{K}^\star)^{\frac{3}{2}}} + \frac{2}{\pi}\arcsin\left(\frac{\mathcal{K}^\star}{1+\mathcal{K}^\star}\right)\right]^{-\frac{1}{2}}\,, \tag{A.133}$$

$$\sigma_b = \frac{\mathcal{K}^\star}{\sqrt{2\pi}(1+2\mathcal{K}^\star)^{\frac{3}{4}}}\sigma_w\,. \tag{A.134}$$

One can then scan $\mathcal{K}^\star$ to draw the critical line.

In order to locate the critical point, we further require $\chi^\star_{\mathcal{K}} = 1$. To locate the critical point, we solve $\chi^\star_{\mathcal{J}} - \chi^\star_{\mathcal{K}} = 0$ instead. We have

$$\frac{\sigma_w^2[(\mathcal{K}^\star)^3 - 3(\mathcal{K}^\star)^2 - 2\mathcal{K}^\star]}{2\pi(1+\mathcal{K}^\star)^2(1+2\mathcal{K}^\star)^{\frac{3}{2}}} = 0\,, \tag{A.135}$$

which has two non-negative solutions out of three

$$\mathcal{K}^\star = 0 \text{ and } \mathcal{K}^\star = \frac{3+\sqrt{17}}{2}\,. \tag{A.136}$$

One can then solve σ_b and σ_w by plugging corresponding $K^\star$ values.

$$(\sigma_w, \sigma_b) = (2, 0)\,, \text{ for } \mathcal{K}^\star = 0\,, \tag{A.137}$$

$$(\sigma_w, \sigma_b) \approx (1.408, 0.416)\,, \text{ for } \mathcal{K}^\star = \frac{3+\sqrt{17}}{2}\,. \tag{A.138}$$

A.12.3 Critical Exponents

GELU behaves in a different way compared to erf. First we discuss the $\mathcal{K}^\star = 0$ critical point, which is located at $(\sigma_b, \sigma_w) = (0, 2)$. We expand Eq.(A.128), and keep next to leading order $\delta\mathcal{K}^l = \mathcal{K}^l - \mathcal{K}^\star$

$$\delta\mathcal{K}^{l+1} \approx \delta\mathcal{K}^l + \frac{6}{\pi}(\delta\mathcal{K}^l)^2\,. \tag{A.139}$$

From lemma A.28

$$A = -\frac{\pi}{6} \text{ and } \zeta_{\mathcal{K}} = 1\,, \tag{A.140}$$

which is not possible since $\delta\mathcal{K}^l \geq 0$ for this case. This result means scaling analysis is not working here.

Next, we consider the other fixed point with $\mathcal{K}^\star = \frac{3+\sqrt{17}}{2}$ at $(\sigma_b, \sigma_w) = (0.416, 1.408)$. Expand the NNGP kernel recurrence relation again.

$$\delta\mathcal{K}^{l+1} \approx \delta\mathcal{K}^l + 0.00014(\delta\mathcal{K}^l)^2\,. \tag{A.141}$$

Following the same analysis, we find

$$\delta\mathcal{K}^l \approx -7142.9\, l^{-1}\,. \tag{A.142}$$

Looks like scaling analysis works for this case, since $\mathcal{K}^\star > 0$. The solution shows that the critical point is half-stable[56]. If $\mathcal{K}^l < \mathcal{K}^\star$, the fixed point is repealing, while when $\mathcal{K}^l > \mathcal{K}^\star$,

the fixed point is attractive. However, the extremely large coefficient in the scaling behavior of $\delta\mathcal{K}^l$ embarrasses the analysis. Since for any network with a reasonable depth, the deviation $\delta\mathcal{K}^l$ is not small.

Now we can expand $\chi^l_{\mathcal{J}}$ at some large depth, up to leading order l^{-1}.

$$\chi^l_{\mathcal{J}} \approx 1 - \frac{66.668}{l}\,. \tag{A.143}$$

Then

$$\delta\mathcal{J}^{l_0,l} \approx c_{l_0} l^{-66.668}\,, \tag{A.144}$$

where c_{l_0} is a positive non-universal constant.

Critical exponent

$$\zeta = 66.668\,. \tag{A.145}$$

Which in practice is not traceable.

A.12.4 LayerNorm on Preactivations

Use lemma A.8, we have

$$\begin{aligned}\chi^l_{\mathcal{J}} =& \frac{\sigma_w^2}{N_l \mathcal{K}^l} \sum_{k=1}^{N_l} \mathbb{E}_\theta \left[\phi'(\tilde{h}^l_k)\phi'(\tilde{h}^l_k)\right]\Big|_{\tilde{\mathcal{K}}^{l-1}=1} \\ =& \frac{\sigma_w^2(6\pi + 4\sqrt{3})}{\sigma_w^2(6\pi + 3\sqrt{3}) + 18\pi\sigma_b^2}\,.\end{aligned} \tag{A.146}$$

The critical line is then at

$$\begin{aligned}\sigma_b =& \left(6\sqrt{3}\pi\right)^{-\frac{1}{2}} \sigma_w \\ \approx& 0.175\sigma_w \,. \end{aligned} \tag{A.147}$$

A.12.5 LayerNorm on Activations

First, we need to evaluate a new expectation value

$$\begin{aligned}\mathbb{E}_\theta\left[\phi(h_i^l)\right] =& \frac{1}{\sqrt{2\pi(\sigma_w^2+\sigma_b^2)}} \int dh_i^l \frac{h_i^l}{2}\left[1+\text{erf}\left(\frac{x}{\sqrt{2}}\right)\right] e^{-\frac{(h_i^l)^2}{2(\sigma_w^2+\sigma_b^2)}} \\ =& \frac{\sigma_w^2+\sigma_b^2}{\sqrt{2\pi(1+\sigma_w^2+\sigma_b^2)}} \,, \end{aligned} \tag{A.148}$$

where we used integration by parts to get the result.

The other integrals are modified to

$$\mathbb{E}_\theta\left[\phi'(h_i^l)\phi'(h_i^l)\right] = \frac{1}{4} + \frac{1}{2\pi}\left[\arcsin\left(\frac{\sigma_w^2+\sigma_b^2}{1+\sigma_w^2+\sigma_b^2}\right) + \frac{(\sigma_w^2+\sigma_b^2)[3+5(\sigma_w^2+\sigma_b^2)]}{(1+\sigma_w^2+\sigma_b^2)[1+2(\sigma_w^2+\sigma_b^2)]^{\frac{3}{2}}}\right] , \tag{A.149}$$

$$\mathbb{E}_\theta\left[\phi(h_i^l)\phi(h_i^l)\right] = \frac{\sigma_w^2+\sigma_b^2}{4} + \frac{\sigma_w^2+\sigma_b^2}{2\pi}\arcsin\left(\frac{\sigma_w^2+\sigma_b^2}{1+\sigma_w^2+\sigma_b^2}\right) + \frac{(\sigma_w^2+\sigma_b^2)^2}{\pi(1+\sigma_w^2+\sigma_b^2)\sqrt{1+2(\sigma_w^2+\sigma_b^2)}} \,. \tag{A.150}$$

One can then combine those results to find $\chi_{\mathcal{J}}^l$

$$\chi_{\mathcal{J}}^l = \frac{\sigma_w^2\left(1+\sigma_w^2+\sigma_b^2\right)\left[\pi + 2\arcsin\left(\frac{\sigma_w^2+\sigma_b^2}{1+\sigma_w^2+\sigma_b^2}\right) + \frac{2(\sigma_w^2+\sigma_b^2)(3+5(\sigma_w^2+\sigma_b^2))}{(1+\sigma_w^2+\sigma_b^2)(1+2(\sigma_w^2+\sigma_b^2))^{\frac{3}{2}}}\right]}{\pi(\sigma_w^2+\sigma_b^2)(1+\sigma_w^2+\sigma_b^2) - 2(\sigma_w^2+\sigma_b^2)^2 + \frac{4(\sigma_w^2+\sigma_b^2)^2}{\sqrt{1+2(\sigma_w^2+\sigma_b^2)}} + 2(\sigma_w^2+\sigma_b^2)(1+\sigma_w^2+\sigma_b^2)\arcsin\left(\frac{\sigma_w^2+\sigma_b^2}{1+\sigma_w^2+\sigma_b^2}\right)} \,. \tag{A.151}$$

The critical line defined by $\chi^l_{\mathcal{J}} = 1$, one can numerically solve it by scanning over σ_b and σ_w.

A.12.6 Residual Connections

The recurrence relation for the NNGP kernel is

$$\mathcal{K}^{l+1} = \left[\frac{\mathcal{K}^l}{4} + \frac{\mathcal{K}^l}{2\pi}\arcsin\left(\frac{\mathcal{K}^l}{1+\mathcal{K}^l}\right) + \frac{(\mathcal{K}^l)^2}{\pi(1+\mathcal{K}^l)\sqrt{1+2\mathcal{K}^l}}\right]\sigma_w^2 + \sigma_b^2 + \mu^2\mathcal{K}^l\,. \tag{A.152}$$

Fixed point exists if

$$\chi^\star_{\mathcal{K}} = \frac{\sigma_w^2}{4} + \frac{\sigma_w^2}{2\pi}\left[\arcsin\left(\frac{\mathcal{K}^\star}{1+\mathcal{K}^\star}\right) + \frac{4(\mathcal{K}^\star)^3 + 11(\mathcal{K}^\star)^2 + 5\mathcal{K}^\star}{(1+\mathcal{K}^\star)^2(1+2\mathcal{K}^\star)^{\frac{3}{2}}}\right] + \mu^2 \leq 1\,. \tag{A.153}$$

The recurrence coefficient for Jacobian is

$$\chi^\star_{\mathcal{J}} = \frac{\sigma_w^2}{4} + \frac{\sigma_w^2}{2\pi}\left[\arcsin\left(\frac{\mathcal{K}^\star}{1+\mathcal{K}^\star}\right) + \frac{\mathcal{K}^\star(3+5\mathcal{K}^\star)}{(1+\mathcal{K}^\star)(1+2\mathcal{K}^\star)^{\frac{3}{2}}}\right] + \mu^2\,. \tag{A.154}$$

Phase boundary is shifted

$$\sigma_w = 2\sqrt{1-\mu^2}\left[1 + \frac{2\mathcal{K}^\star(3+5\mathcal{K}^\star)}{\pi(1+\mathcal{K}^\star)(1+2\mathcal{K}^\star)^{\frac{3}{2}}} + \frac{2}{\pi}\arcsin\left(\frac{\mathcal{K}^\star}{1+\mathcal{K}^\star}\right)\right]^{-\frac{1}{2}}\,, \tag{A.155}$$

$$\sigma_b = \frac{\mathcal{K}^\star}{\sqrt{2\pi}(1+2\mathcal{K}^\star)^{\frac{3}{4}}}\sigma_w\,. \tag{A.156}$$

One can again scan over $\mathcal{K}^\star$ to draw the critical line.

In order to locate the critical point, we further require $\chi^\star_{\mathcal{K}} = 1$. To locate the critical point,

we solve $\chi^\star_{\mathcal{J}} - \chi^\star_{\mathcal{K}} = 0$ instead. We have

$$\frac{\sigma_w^2[(\mathcal{K}^\star)^3 - 3(\mathcal{K}^\star)^2 - 2\mathcal{K}^\star]}{2\pi(1+\mathcal{K}^\star)^2(1+2\mathcal{K}^\star)^{\frac{3}{2}}} = 0\,, \tag{A.157}$$

which has two non-negative solutions out of three

$$\mathcal{K}^\star = 0 \text{ and } \mathcal{K}^\star = \frac{3+\sqrt{17}}{2}\,. \tag{A.158}$$

One can then solve σ_b and σ_w by plugging corresponding $K^\star$ values.

$$(\sigma_w, \sigma_b) = (2\sqrt{1-\mu^2}, 0)\,, \text{ for } \mathcal{K}^\star = 0\,, \tag{A.159}$$

$$(\sigma_w, \sigma_b) \approx (1.408\sqrt{1-\mu^2}, 0.416\sqrt{1-\mu^2})\,, \text{ for } \mathcal{K}^\star = \frac{3+\sqrt{17}}{2}\,. \tag{A.160}$$

A.12.7 Residual Connections with LayerNorm on Preactivations (Pre-LN)

Use lemma A.8 and the results we had without residue connections for GELU.

$$\begin{aligned}
\chi^*_{\mathcal{J}} =& \lim_{l\to\infty}\left(\frac{\sigma_w^2}{N_l\mathcal{K}^l}\sum_{k=1}^{N_l}\mathbb{E}_\theta\left[\phi'(\tilde{h}^l_k)\phi'(\tilde{h}^l_k)\right]\Big|_{\tilde{\mathcal{K}}^{l-1}=1} + \mu^2\right)\\
=&\frac{\sigma_w^2(6\pi+4\sqrt{3})(1-\mu^2)}{\sigma_w^2(6\pi+3\sqrt{3})+18\pi\sigma_b^2} + \mu^2\\
=&1 - \frac{(\sqrt{3}\sigma_w^2 - 18\pi\sigma_b^2)(1-\mu^2)}{\sigma_w^2(6\pi+3\sqrt{3})+18\pi\sigma_b^2}\,.
\end{aligned} \tag{A.161}$$

The critical line is then at

$$\begin{aligned}\sigma_b &= \left(6\sqrt{3}\pi\right)^{-\frac{1}{2}} \sigma_w \\ &\approx 0.175\sigma_w \,,\end{aligned} \tag{A.162}$$

just like without residue connections.

A.13 Additional Experimental Results

In the following training results, we used NTK parameterization for the linear layers in the MLP. We emphasize that this choice has little effect on the training and convergence in this case, compared to standard initialization.

In figure A.4, we showed empirically that the critical exponent of partial Jacobians are vanished for erf with LayerNorm.

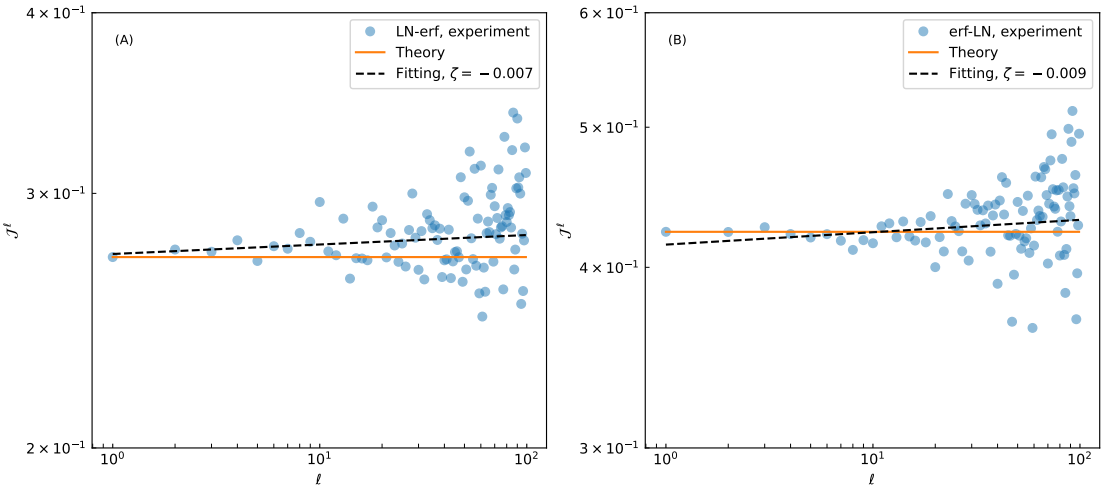


Figure A.4: $\log - \log$ plot of partial Jacobian $\mathcal{J}^{0,l}$ vs. l for (A) LN-erf and (B) erf-LN.

In figure A.5, we tested $6k$ samples from CIFAR-10 dataset[70] with kernel regression

based on neural tangents library [36, 177, 178]. Test accuracy from kernel regression reflects the trainability (training accuracy) with SGD in the ordered phase. We found that the trainable depth is predicted by the correlation length $c\xi$ with LayerNorm applied to preactivations, where the prefactor $c = 28$. The prefactor we had is the same as vanilla cases in [179]. The difference is from the fact that they used $\log_{10}$ and we used $\log_e$.

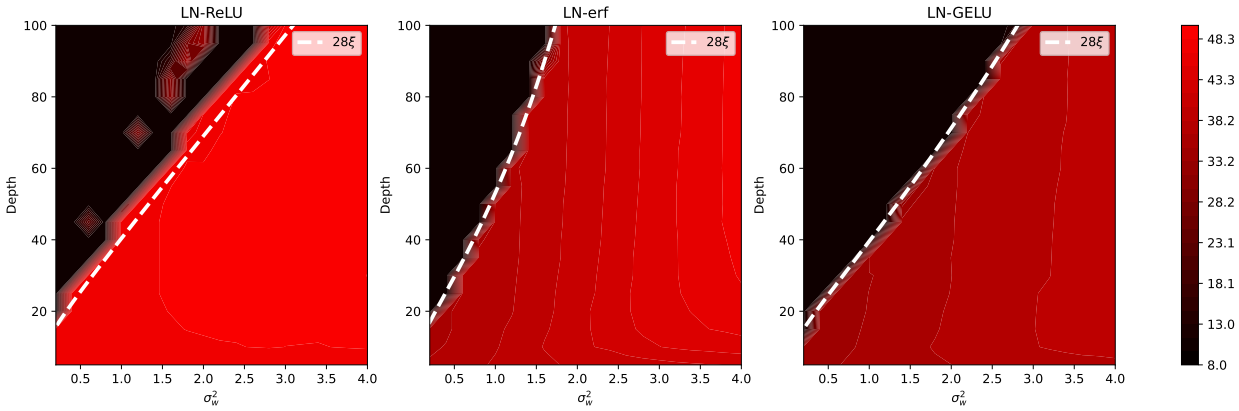


Figure A.5: Test accuracy for LayerNorm applied to preactivations. $\sigma_b^2 = 0.5$ for all cases. Correlation lengths are calculated using analytical results of $\chi^l_{\mathcal{J}}$.

In figure A.6, we explore the broad range in σ_w^2 of the performance of MLP network with erf activation function and LayerNorm on preativations. The network has depth $L = 50$ and width $N_l = 500$; and is trained using SGD on Fashion MNIST. The learning rates are chosen based on a logarithmic scan with a short training time.

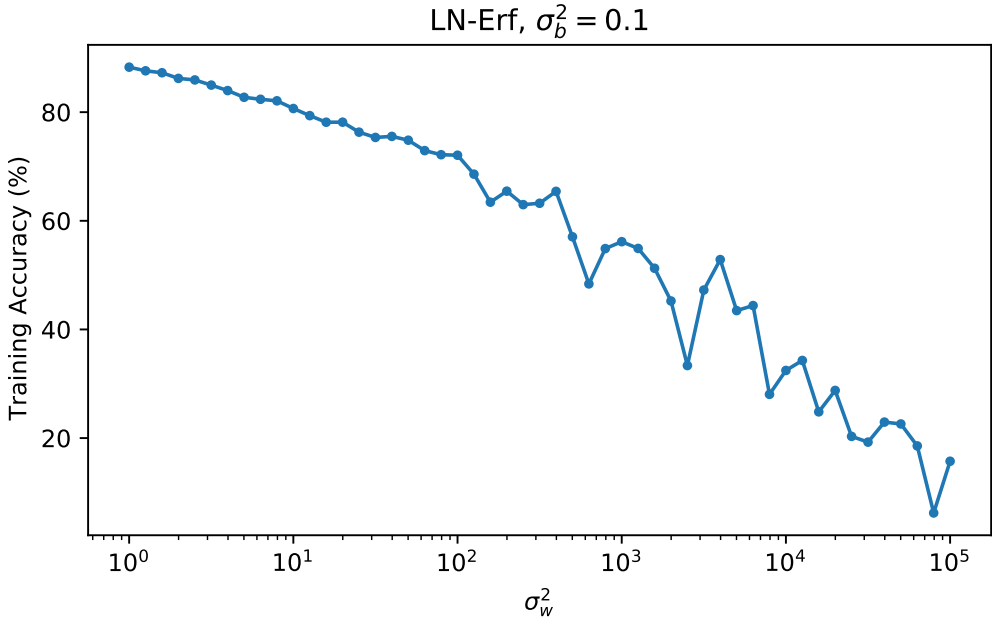


Figure A.6: Training performance of MLP networks with erf activation function; and LayerNorm applied to preactivations. It continues to train for several orders of magnitude of σ_w^2 (with learning-rate tuning).

Appendix B: Emergent Low-dimensional Fixed-Point Structures in Sharpness Dynamics

B.1 Further Discussion on Related Works

Lewkowycz et al. [73] examined the early training dynamics of wide networks at large learning rates. Using the top eigenvalue of the Neural Tangent Kernel (NTK) λ^K at initialization ($t = 0$), they revealed a 'catapult phase', $2/\lambda_0^K < \eta < \eta_{\max}$, in which training converges despite an initial spike in training loss. Kalra and Barkeshli [72] analyzed early training dynamics for arbitrary depths and width and revealed a 'sharpness reduction phase', $2/\lambda_0^H < \eta < c_{\text{loss}}/\lambda_0^H$, which opens up significantly as c_{loss} increases with depth and $1/$width.

Beyond early training, sharpness continues to increase, until it reaches a break-even point [78], beyond which GD dynamics typically enters the EoS regime [71]. This has motivated various theoretical studies to understand GD dynamics at large learning rates: Chen and Bruna [81], Chen et al. [84], Arora et al. [86], Rosca et al. [88], Zhu et al. [89], Wu et al. [90], Ahn et al. [91], Kreisler et al. [92], Song and Yun [93]. In particular, Ma et al. [82] showed that loss functions with sub-quadratic growth exhibit EoS behavior. Arora et al. [86] show that normalized gradient descent reaches the EoS regime. Damian et al. [87] analyze the dynamics of the cubic approximation of the loss. Assuming a negative correlation between the gradient direction and

the top eigenvector of Hessian, they show that gradient descent dynamics enters a stable cycle in the EoS regime. Ahn et al. [91] analyze EoS in a single-neuron 2-layer network and a simplified three-parameter ReLU network assuming the existence of a 'forward invariant subset' near the minima. Kreisler et al. [92] analyzed scalar linear networks to show that the sharpness attained by the gradient flow dynamics monotonically decreases in the EoS regime. Wu et al. [90] demonstrate that gradient descent, with any learning rate in the EoS regime, optimizes logistic regression with linearly separated data over large time scales. Below, we discuss closely related works in detail and clarify the distinction with our work.

Wang et al. [85] analyze EoS in a 2-layer linear network using the norm of the last layer. They solely focus on cases that exhibit progressive sharpening right from initializations by considering assumptions (refer to Assumptions 4.1 and 4.2 of their paper) on the training dataset. Contrary to these assumptions, Figure 3.1(e) demonstrates that such assumptions are invalid in many realistic settings, where progressive sharpening is not observed at all.

Agarwala et al. [80] showed that a modified model exhibits progressive sharpening and two-step oscillations at EoS using NTK as the proxy. They state that the UV model does not exhibit EoS behavior (see Section 3.2.1 of the referenced paper). This is because their analysis is restricted to the Standard Parmaeterization corresponding to Figure 3.2(c, f) (c.f. Figure 8 of [80]). In contrast, we show that the UV model exhibits EoS behavior under the appropriate choice of parameterization and training example.

Zhu et al. [89] proved EoS convergence for the loss $\frac{1}{4}(x^2y^2 - 1)^2$, where $x, y \in \mathbb{R}$. Additionally, they empirically demonstrated a bifurcation diagram in the space of abstract variables of x and y. It is worth noting that while these bifurcations arise from the same underlying behavior, they contrast with our route to chaos bifurcation diagrams which quantify sharpness fluctuations

with learning rate.

Chen and Bruna [81] analyze two-step gradient updates of a single-neuron network and matrix factorization to gain insights into EoS. Similar to our work, they show a bifurcation diagram of sharpness against the learning rate for the matrix factorization problem. While the scalar matrix factorization problem can be mapped to the UV model with a specific choice of $\frac{||x||}{\sqrt{n_{\text{eff}}}}$, it is not straightforward to apply their conclusions to the neural network setting, as it requires the correct choice of parameterization. In particular, the UV model under NTP parameterization, as shown in Figure 3.2(c, f), does not display EoS behavior at considerable widths, a finding also noted by [80]. Observing EoS requires the correct choice of the parameterization (μP) and training example. Furthermore, although the scalar matrix factorization in [81] can be mapped to a special case of the UV model considered in our work, we provide significant additional insights. In particular, with respect to the bifurcation phenomena in the UV model, we explain the existence of an attractor submanifold on which the EoS behavior occurs. We further show that on the EoS submanifold, the loss becomes subquadratic in nature and the gradient descent dynamics therefore become approximated by the cubic map, which is well-studied in the chaos literature. This makes clear the origin of the period-doubling route to chaos. Additionally, in Section 6, we extend the analysis of EoS beyond the UV model, comparing sharpness trajectories of synthetic and real datasets at EoS. In contrast to the synthetic datasets, sharpness trajectories of real datasets show long-range correlations in time. We take the first steps by attributing these long-range correlations to correlations in the dataset. Note that this setting cannot be mapped to the matrix factorization setting.

Song and Yun [93] show that late-time trajectories oscillate around ${}^{2f}/_{\eta\ell'}$, where f is the network output and ℓ' is the derivative of the loss. They refer to the term bifurcation diagram

to describe these phenomena, contrasting with the sharpness versus learning rate bifurcation diagrams presented in our study. We quote Section 3 from their paper "..we plot the bifurcation diagram $q = r(p) = \ell'(p)/p$ and observe that GD trajectories tend to align with this curve.." Here, p and q correspond to Δf and $\frac{2}{\eta\lambda}$ in our setting. They plot trajectories in the $(p, q) \equiv (\Delta f, \frac{2}{\eta\lambda})$ plane and their condition $q = r(p)$ simply corresponds to the EoS condition $\lambda = \frac{2}{\eta}$ for MSE loss. In contrast, our work presents bifurcation diagrams resulting from how the sharpness fluctuations vary with the learning rate. Therefore, the bifurcation diagrams from these works are not directly related to the route-to-chaos bifurcation diagrams presented in our work.

Kong and Tao [180] were the first ones to show that gradient descent dynamics becomes chaotic at large learning rates and converges to a statistical distribution instead of a minimum.

Chen et al. [84] analyzed large learning rate dynamics of toy models which are characterized by a one-dimensional cubic map and demonstrated five different training phases: (a) monotonic, (b) catapult, (c) periodic, (d) chaotic, and (e) divergent. In particular, they considered a two-layer network with quadratic activation, where the last layer vector $\mathbf{v}$ is not trained and each entry is set to one. This model belongs to a family that is effectively described by one variable Δf. In this model, the loss is sub-quadratic by construction $(||Ux||^2 - y)^2$. In contrast, the UV model that we study is an effectively two-variable model and in these cases, training dynamically finds the attractive EoS manifold such that the loss has a sub-quadratic nature on this submanifold.

Concurrent work by Wang et al. [94] categorizes training trajectories into three stages: (i) sharpness reduction, (ii) progressive sharpening, and (iii) edge of stability. They argue that different large learning rate behavior depends on the 'regularity' of the loss landscape. Specifically, they generalize toy landscapes from existing studies with parameters controlling the regularity.

They show that models with good regularity first experience a decrease in sharpness and then progressive sharpening and enter the edge of stability.

Noci et al. [95] examined the sharpness dynamics of networks with parameterization. They argue that the learning rate transfer property of μP is correlated with consistent sharpness trajectories across varying depths and widths.

B.2 Experimental details

B.2.1 Datasets

Standard image datasets: We considered the MNIST [181], Fashion-MNIST [182], and CIFAR-10 [183] datasets. The images are standardized to have zero mean and unit variance across the feature dimensions, and target labels are represented as one-hot encodings.

Random dataset: We construct a random dataset $(X, Y) = \{(\boldsymbol{x}^\mu, \boldsymbol{y}^\mu)\}_{\mu=1}^P$ with $\boldsymbol{x}^\mu \sim \mathcal{N}(0, I)$ and $\boldsymbol{y}^\mu \sim \mathcal{N}(0, I)$, both sampled independently. Note there is no correlation between inputs and outputs.

Teacher-student dataset: Consider a teacher network $f(\boldsymbol{x}; \theta_0)$ with θ_0 initialized randomly as described in Appendix B.2.2. Then, we construct a teacher-student dataset $(X, Y) = \{(\boldsymbol{x}^\mu, \boldsymbol{y}^\mu)\}_{\mu=1}^P$ with $\boldsymbol{x}^\mu \sim \mathcal{N}(0, I)$ and $\boldsymbol{y}^\mu = f(\boldsymbol{x}^\mu; \theta_0)$.

Random power-law dataset: Starting with the random dataset (X', Y'), we utilize the singular value decomposition of the input and output matrices

$$X' = P_x S_{x'} Q_x^T, \qquad Y' = P_y S_{y'} Q_y^T. \tag{B.1}$$

Next, we rescale the k^{th} singular value of $S_{x'}$ and $S_{y'}$ as

$$(S_x)_k = A_x (S_{x'})_k k^{-B_x} \qquad (S_y)_k = A_y (S_{y'})_k k^{-B_y}, \tag{B.2}$$

and re-construct input and output matrices as below

$$X = P_x S_x Q_x^T, \qquad Y = P_y S_y Q_y^T. \tag{B.3}$$

The variables A_x, B_x, A_y, and B_y uniquely characterize the dataset.

Generative image dataset: Given a pre-trained network $f(\boldsymbol{x}; \theta)$ on a standard image dataset listed above, we construct a generative image dataset $(X, Y) = \{(\boldsymbol{x}^\mu, \boldsymbol{y}^\mu)\}_{\mu=1}^P$ with $\boldsymbol{x}^\mu \sim \mathcal{N}(0, I)$ and $\boldsymbol{y}^\mu = f(\boldsymbol{x}^\mu; \theta)$.

B.2.2 Models

FCNs: We considered ReLU FCNs without bias with uniform hidden layer width n.

CNNs: We considered Myrtle family ReLU CNNs [184] without any bias with a fixed number of channels in each layer, which we refer to as the width of the network.

ResNets: We adapted ResNet [185] implementations from Flax examples. Our implementation uses Layer norm and initialize the weights as $\mathcal{N}(0, \sigma_w^2/\text{fan}_{\text{in}})$. For ResNets, we refer to the number of channels in the first block as the width.

We implemented all models using the JAX [186], and Flax libraries [187].

B.2.3 Details of network parameterization

In this section, we describe different parameterizations used in the paper. For simplicity, we describe the parameterizations for FCNs. Nevertheless, these arguments generalize to other architectures.

Standard Parameterization (SP): Consider a neural network $f : \mathbb{R}^{d_{\text{in}}} \to \mathbb{R}^{d_{\text{out}}}$ with d layers and constant width n. Then, standard parameterization is defined as follows:

$$\begin{aligned} \boldsymbol{h}^{(1)}(\boldsymbol{x}) &= W^{(1)}\boldsymbol{x}, \\ \boldsymbol{h}^{(l+1)}(\boldsymbol{x}) &= W^{(l+1)}\phi\left(\boldsymbol{h}^{(l)}(\boldsymbol{x})\right), \\ \boldsymbol{f}(\boldsymbol{x};\theta) &= W^{(d)}\phi\left(\boldsymbol{h}^{(d-1)}(\boldsymbol{x})\right), \end{aligned} \tag{B.4}$$

where $W^{(1)} \sim \mathcal{N}(0, \sigma_w^2/d_{\text{in}})$, $W^{(l)} \sim \mathcal{N}(0, \sigma_w^2/n)$ for $1 < l < d$, and $W^{(d)} \sim \mathcal{N}(0, 1/n)$; $\phi(\cdot)$ is the elementwise activation function. The input is normalized such that $\|\boldsymbol{x}\|^2 = d_{\text{in}}$.

Neural Tangent Parameterization (NTP): Consider a neural network $f : \mathbb{R}^{d_{\text{in}}} \to \mathbb{R}^{d_{\text{out}}}$ with d layers and constant width n. Then, the Neural Tangent Parameterization is defined as follows:

$$\begin{aligned}
\boldsymbol{h}^{(1)}(\boldsymbol{x}) &= \frac{\sigma_w}{\sqrt{d_{\text{in}}}} W^{(1)} \boldsymbol{x}, \\
\boldsymbol{h}^{(l+1)}(\boldsymbol{x}) &= \frac{\sigma_w}{\sqrt{n}} W^{(l+1)} \phi\left(\boldsymbol{h}^{(l)}(\boldsymbol{x})\right), \\
\boldsymbol{f}(\boldsymbol{x}; \theta) &= \frac{1}{\sqrt{n}} W^{(d)} \phi\left(\boldsymbol{h}^{(d-1)}(\boldsymbol{x})\right),
\end{aligned} \tag{B.5}$$

where $W^{(l)} \sim \mathcal{N}(0, 1)$ for $1 \leq l \leq d$ and $\phi(\cdot)$ is the elementwise activation function. The input is normalized such that $\|\boldsymbol{x}\|^2 = d_{\text{in}}$.

Both SP and NTP are closely related parameterizations – SP with learning rate $\eta = \Theta(1/n)$ learning rate becomes equivalent to NTP when the input dimension is equal to the width ($d_{\text{in}} = n$) [77].

Interpolating Parameterization: Consider a neural network $f : \mathbb{R}^{d_{\text{in}}} \to \mathbb{R}^{d_{\text{out}}}$ with d layers and constant width n. Let $W^{(l)}$ denote the weight matrix at layer l. Then, "interpolating parameterization" is defined as follows:

$$\begin{aligned}
\boldsymbol{h}^{(1)}(\boldsymbol{x}) &= n^{s/2} W^{(1)} \boldsymbol{x}, \\
\boldsymbol{h}^{(l+1)}(\boldsymbol{x}) &= W^{(l+1)} \phi\left(\boldsymbol{h}^{(l)}(\boldsymbol{x})\right), \\
\boldsymbol{f}(\boldsymbol{x}; \theta) &= \frac{1}{n^{s/2}} W^{(d)} \phi\left(\boldsymbol{h}^{(d-1)}(\boldsymbol{x})\right),
\end{aligned} \tag{B.6}$$

Here, s is a parameter that interpolates between standard-like parameterization and maximal update parameterization. The weight matrices are sampled from Gaussian distributions: $W^{(1)} \sim \mathcal{N}(0, \sigma_w^2/n^s)$, $W^{(l)} \sim \mathcal{N}(0, \sigma_w^2/n)$ for $1 < l < d$, and $W^{(d)} \sim \mathcal{N}(0, 1/n)$. We normalize the input such that $\|\boldsymbol{x}\| = 1$.

Maximal update Parameterization (μP): The maximal update parameterization corresponds to the $s = 1$ case in the above setting.

B.2.4 Details of Figures

Figure 3.1: Training loss and sharpness trajectories of 4-layer ReLU FCNs with $n = 512$, trained on a subset of $5,000$ CIFAR-10 examples using MSE loss and GD: (a, d) SP with $\sigma_w^2 = 0.5$, (b, e) SP with $\sigma_w^2 = 2.0$, (c, f) μP with $\sigma_w^2 = 2.0$.

Figure 3.2 Training trajectories of the UV model with $\|\boldsymbol{x}\| = 1$ and $y = 2$ in the $(\Delta f, \lambda)$ plane for different values of n, n_{eff} and η. The columns show initializations with different n and n_{eff}, while the rows represent increasing learning rates for fixed initializations. The horizontal dash-dot line $\eta\lambda = 2$ separates the stable (solid black vertical line) and unstable (dashed black vertical line) fixed points along the zero loss fixed line I. Forbidden regions, $2\|\boldsymbol{x}\||\Delta f + y|/\sqrt{n_{\text{eff}}} > \lambda$, (see Appendix B.3.2) are shaded gray. The nullclines $\Delta f_{t+1} = \Delta f_t$ and $\lambda_{t+1} = \lambda_t$ are shown as orange and white dashed curves, respectively. Sharpness reduction, progressive sharpening, and divergent regions are colored green, yellow, and blue. The gray arrows indicate the local vector field $\hat{G}(\Delta f, \lambda)$, which is the direction of the updates. The training trajectories are depicted as black lines with arrows, with the star marking the initialization. In all cases, $\eta_c = \sqrt{n_{\text{eff}}}/2$

(introduced in Section 3.5.2).

Figure 3.3: *UV model dynamics on the EoS manifold*:(a) Bifurcation diagram depicting late-time limiting values of λ obtained by simulating Equation (3.4). (b) Bifurcation diagram of the UV model. In both figures, $\|\boldsymbol{x}\| = 1, y = 2$ and $n_{\text{eff}} = 1$ and $\eta_c = 0.5$.

Figure 3.4: *Phase diagram of EoS*: (a) Heatmap of $\eta\bar{\lambda}^H/2$ of 3-layer ReLU FCNs with $s = 0$ trained on a subset of $5,000$ CIFAR-10 examples for 10k steps, with the weight variance σ_w^2 and learning rate multiplier $c = \eta\lambda_0^H$ as axes. $\bar{\lambda}^H$ is obtained by averaging λ_t^H over last 200 steps. As the color varies from blue to white, $\eta\bar{\lambda}^H/2$ increases, where the brightest white region indicates the EoS regime with $\eta\bar{\lambda}^H/2 \geq 1$. (b) Same heatmap with fixed $\sigma_w^2 = 2.0$, but varying s continuously.

Figure 3.5: *EoS in synthetic vs real-datasets:* 2-layer linear FCN trained on (first row) $5,000$ iid random examples with unit output dimension and (second row) $5,000$ CIFAR-10 examples. Different columns correspond to the bifurcation diagram, late-time sharpness trajectories, and the power spectrum of sharpness trajectories. Both models are trained for 10k steps using GD.

Figure B.4: *Two-step phase portrait of UV model in* $(\Delta f, \beta)$ *phase plane*: These plots are equivalent to Figure 3.2(d-f), but with training trajectory and local are plotted for every other step.

Figure B.10: Sharpness and Weight Norm of 3-layer ReLU FCNs in SP with $\sigma_w^2 = 1/3$ and width 200, trained on a subset of CIFAR-10 with $5,000$ examples using GD.

B.2.5 Sharpness measurement

We measure sharpness using the power iteration method with m iterations. Typically, $m = 20$ iterations ensure convergence. Exceptions requiring more iterates are discussed separately.

B.2.6 Power spectrum analysis

For a given signal $x'(t)$, we standardize the signal

$$x(t) = \frac{x'(t) - \mu}{\sigma}, \tag{B.7}$$

where μ is the mean and σ^2 is the variance of the signal. Subtracting the mean removes the zero frequency component in the power spectrum. Next, consider the discrete Fourier transform $\mathcal{F}(\omega)$ of $x(t)$:

$$\mathcal{F}(\omega) = \frac{1}{T} \sum_{t=0}^{T-1} e^{-i2\pi\omega t/T} x(t), \tag{B.8}$$

Then, the power spectrum is $P(\omega) = |\mathcal{F}(\omega)|^2$. The normalization by T in the Fourier transform ensures that the sum of the power spectrum is equal to the variance of the signal, i.e., $\sum_\omega P(\omega) = \sigma^2$.

B.2.7 Estimation of Computational Resources Used

Most of our experiments, aside from the phase diagrams, required minimal computational resources, estimated to be less than 50 A100 hours. In contrast, each phase diagram required 50 A100 hours, totaling 500 A100 hours for all phase diagrams. Including initial experiments, we expect our total usage to be under 600 A100 hours.

B.3 Properties of the UV model

B.3.1 Derivation of the Function Space Dynamics

Equations (3.2) and (3.3) can be derived using the gradient descent update equations:

$$U_{t+1} = U_t - \eta \frac{\Delta f_t \boldsymbol{v}_t \boldsymbol{x}^T}{\sqrt{n_{\text{eff}}}}, \tag{B.9}$$

$$\boldsymbol{v}_{t+1} = \boldsymbol{v}_t - \eta \frac{\Delta f_t U_t \boldsymbol{x}_t}{\sqrt{n_{\text{eff}}}}. \tag{B.10}$$

At step $t+1$, the residual Δf_{t+1} can be written in terms of the gradient updates of U and $\boldsymbol{v}$:

$$\Delta f_{t+1} = f_{t+1} - y \tag{B.11}$$

$$= \frac{1}{\sqrt{n_{\text{eff}}}} \boldsymbol{v}_{t+1}^T U_{t+1} \boldsymbol{x} - y \tag{B.12}$$

$$= \frac{1}{\sqrt{n_{\text{eff}}}} \left(\boldsymbol{v}_t - \eta \frac{\Delta f_t U_t \boldsymbol{x}}{\sqrt{n_{\text{eff}}}} \right)^T \left(U_t - \eta \frac{\Delta f_t \boldsymbol{v}_t \boldsymbol{x}^T}{\sqrt{n_{\text{eff}}}} \right) \boldsymbol{x} - y \tag{B.13}$$

$$= \Delta f_t - \frac{\eta \Delta f_t}{n_{\text{eff}}} \left(\boldsymbol{x}^T U_t^T U_t \boldsymbol{x} + \boldsymbol{v}_t^T \boldsymbol{v}_t \boldsymbol{x}^T \boldsymbol{x} \right) + \frac{\eta^2 \|\boldsymbol{x}\|^2 \Delta f_t^2}{n_{\text{eff}}} \left(\frac{1}{\sqrt{n_{\text{eff}}}} \boldsymbol{x}^T U_t^T \boldsymbol{v}_t \right) \tag{B.14}$$

$$= \Delta f_t \left(1 - \eta \lambda_t + \frac{\eta^2 \|\boldsymbol{x}\|^2}{n_{\text{eff}}} \Delta f_t (\Delta f_t + y) \right). \tag{B.15}$$

Here, Δf_{t+1} only depends on Δf_t and λ_t. Similarly, we write down the λ_{t+1} using the gradient update equations:

$$\lambda_{t+1} = \frac{1}{n_{\text{eff}}} \left(\boldsymbol{x}^T U_{t+1}^T U_{t+1} \boldsymbol{x} + \boldsymbol{v}_{t+1}^T \boldsymbol{v}_{t+1} \boldsymbol{x}^T \boldsymbol{x} \right) \tag{B.16}$$

$$= \lambda_t - 4 \frac{\eta \|\boldsymbol{x}\|^2}{n_{\text{eff}}} \Delta f_t (\Delta f_t + y) + \frac{\eta^2 \|\boldsymbol{x}\|^2 \Delta f_t^2}{n_{\text{eff}}} \lambda_t \tag{B.17}$$

$$= \lambda_t + \frac{\eta \|\boldsymbol{x}\|^2}{n_{\text{eff}}} \Delta f_t^2 \left(\eta \lambda_t - 4 \frac{\Delta f_t + y}{\Delta f_t} \right). \tag{B.18}$$

Equations (B.15) and (B.18) form a closed system. This means that Δf_{t+1} and λ_{t+1} are completely described using Δf_t and λ_t. As a result, the complete dynamics of the UV model can be fully described using only these two variables with three parameters effective parameters η, $\frac{\|x\|^2}{n_{\text{eff}}}$ and y.

B.3.2 Forbidden regions of the UV model

In this section, we utilize the non-negativity of λ to derive the condition for allowed regions within the phase plane for the UV model. Consider the function space equations written in terms of the pre-activation $\boldsymbol{h}(\boldsymbol{x}) = U\boldsymbol{x}$:

$$f(\boldsymbol{x};\theta) = \frac{1}{\sqrt{n_{\text{eff}}}}\boldsymbol{v}^T\boldsymbol{h}(\boldsymbol{x}) \tag{B.19}$$

$$\lambda = \frac{1}{n_{\text{eff}}}\left(\|\boldsymbol{v}\|^2\|\boldsymbol{x}\|^2 + \|\boldsymbol{h}\|^2\right). \tag{B.20}$$

Let $\cos(\boldsymbol{h}, \boldsymbol{v})$ denote the cosine similarity between $\boldsymbol{v}$ and $\boldsymbol{h}$. Then, the network output is bounded as

$$|\cos(\boldsymbol{h}, \boldsymbol{v})| = \frac{\sqrt{n_{\text{eff}}}|\Delta f + y|}{\|\boldsymbol{v}\|\|\boldsymbol{h}\|} \leq 1 \tag{B.21}$$

Next, using $(\|\boldsymbol{v}\|\|\boldsymbol{x}\| - \|\boldsymbol{h}\|)^2 \geq 0$, we can bound the product $\|\boldsymbol{v}\|\|\boldsymbol{h}\|$ using λ

$$\frac{2\|\boldsymbol{x}\|}{\sqrt{n_{\text{eff}}}}|\Delta f + y| \leq \lambda. \tag{B.22}$$

The derived inequality describes the allowed phase plane regions for the UV model.

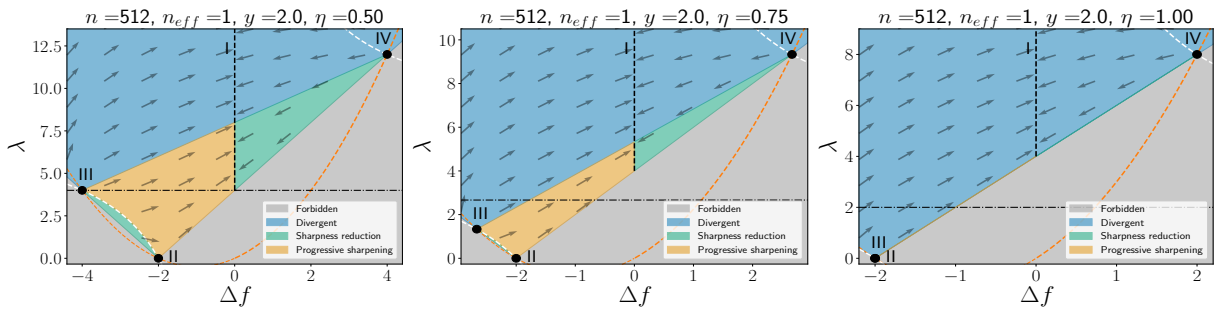
B.3.3 Fixed Points and Line

To identify the fixed points, we set $\Delta f_{t+1} = \Delta f_t$ and $\lambda_{t+1} = \lambda_t$ in Equations (3.2) and (3.3). This yields the dynamical fixed points of the UV model. Table B.1 lists these fixed points along with their stability. Additionally, it provides the eigenvalues and eigenvectors of the Jacobian for the update maps described by Equations (3.2) and (3.3), evaluated at the fixed points.

	$(\Delta f^*, \lambda^*)$	eigenvalues	eigenvectors	Linear stability
I	$(0, \lambda)$ for $\lambda \geq \frac{2\|\boldsymbol{x}\|y}{\sqrt{n_{\text{eff}}}}$	$1, 1-\eta\lambda$	$\begin{bmatrix}0\\1\end{bmatrix}, \begin{bmatrix}\frac{n_{\text{eff}}\lambda}{4\|\boldsymbol{x}\|^2 y}\\1\end{bmatrix}$	$\begin{cases}\text{stable} & \eta\lambda < 2\\ \text{unstable} & \eta\lambda > 2\end{cases}$
II	$(-y, 0)$	$(1-\frac{\eta\|\boldsymbol{x}\|y}{\sqrt{n_{\text{eff}}}})^2, (1+\frac{\eta\|\boldsymbol{x}\|y}{\sqrt{n_{\text{eff}}}})^2$	$\begin{bmatrix}-\frac{\sqrt{n_{\text{eff}}}}{2\|\boldsymbol{x}\|}\\1\end{bmatrix}, \begin{bmatrix}\frac{\sqrt{n_{\text{eff}}}}{2\|\boldsymbol{x}\|}\\1\end{bmatrix}$	saddle
III	$\left(\frac{-2\sqrt{n_{\text{eff}}}}{\|\boldsymbol{x}\|\eta}, \frac{4}{\eta} - \frac{2\|\boldsymbol{x}\|y}{\sqrt{n_{\text{eff}}}}\right)$	$9, 5-\frac{2\eta\|\boldsymbol{x}\|y}{\sqrt{n_{\text{eff}}}}$	$\begin{bmatrix}\frac{n_{\text{eff}}}{\eta\|\boldsymbol{x}\|^2 y}\\1\end{bmatrix}, \begin{bmatrix}-\frac{\sqrt{n_{\text{eff}}}}{2\|\boldsymbol{x}\|}\\1\end{bmatrix}$	unstable
IV	$\left(\frac{2\sqrt{n_{\text{eff}}}}{\|\boldsymbol{x}\|\eta}, \frac{4}{\eta} + \frac{2\|\boldsymbol{x}\|y}{\sqrt{n_{\text{eff}}}}\right)$	$9, 5+\frac{2\eta\|\boldsymbol{x}\|y}{\sqrt{n_{\text{eff}}}}$	$\begin{bmatrix}\frac{n_{\text{eff}}}{\eta\|\boldsymbol{x}\|^2 y}\\1\end{bmatrix}, \begin{bmatrix}\frac{\sqrt{n_{\text{eff}}}}{2\|\boldsymbol{x}\|}\\1\end{bmatrix}$	unstable

Table B.1: Fixed line (I) and points (II-IV) and corresponding eigenvalues and eigenvectors of the Jacobian of the update map in Equations (3.2) and (3.3). The stability is determined for $\eta < \eta_{\text{upper}} = 2\eta_c = {}^{2\sqrt{n_{\text{eff}}}}/_{\|\boldsymbol{x}\|y}$. Above this threshold, training diverges for all initializations except for those at the fixed points, as demonstrated in Appendix B.3.4.

B.3.4 The maximum learning rate η_{upper}



(a) (b) (c)

Figure B.1: Phase portrait of the UV model for different learning rates η. The critical learning rate is $\eta_c = 0.5$ and the maximum learning rate is $\eta_{\text{upper}} = 1.0$.

In Section 3.4, we stated that for $\eta > \eta_{\text{upper}} = 2\eta_c$, training diverges for all initializations except for those at the fixed points. Here, we justify this claim.

First, Figure B.1 shows that as η approaches η_{upper}, fixed point III merges with fixed point II, reducing the convergence region to the EoS manifold. At this learning rate, the stability of fixed point II changes from saddle to unstable as the corresponding eigenvalue $(1-\frac{\eta}{\eta_c})^2\Big|_{\eta=2\eta_c}$ surpasses 1. Consequently, any initialization outside the EoS manifold results in divergence. Next, Figure 3.3(a) shows that on the EoS manifold training diverges for $\eta > \eta_{\text{upper}}$. This corroborates

our initial claim.

B.3.5 Sharpness versus the trace of Hessian

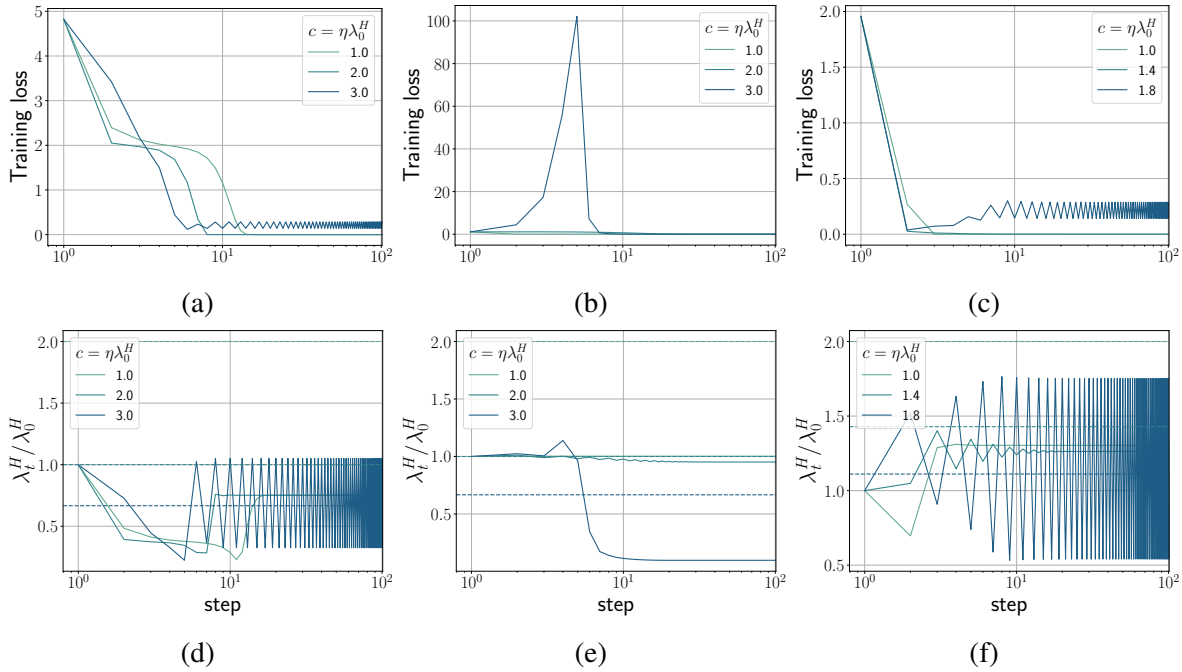


Figure B.2: Training trajectories of the UV model trained on a single example with $\|\boldsymbol{x}\| = 1$ and $y = 2$ using MSE loss and GD: (a, d) NTP with $n = 1$, $\sigma_w^2 = 0.5$, (b, e) NTP with $n = 512$, $\sigma_w^2 = 1.0$, and (c, f) μP with $n = 512$, $\sigma_w^2 = 1.0$.

In this section, we show that the trace of the Hessian λ (which is also the scalar NTK in this case), is an adequate proxy for sharpness. Figure B.3 shows training trajectories of the UV model, with λ as a proxy for sharpness and learning rate scaled as $\eta = {}^{k}/_{\lambda_0}$. These λ trajectories show similar trends to those of λ^H observed in Figure B.2, with one key difference: during early training, λ does not catapult during early training at large widths (compare Figure B.2(e) and Figure B.3(e)). Otherwise, λ effectively captures other qualitative behavior of λ^H.

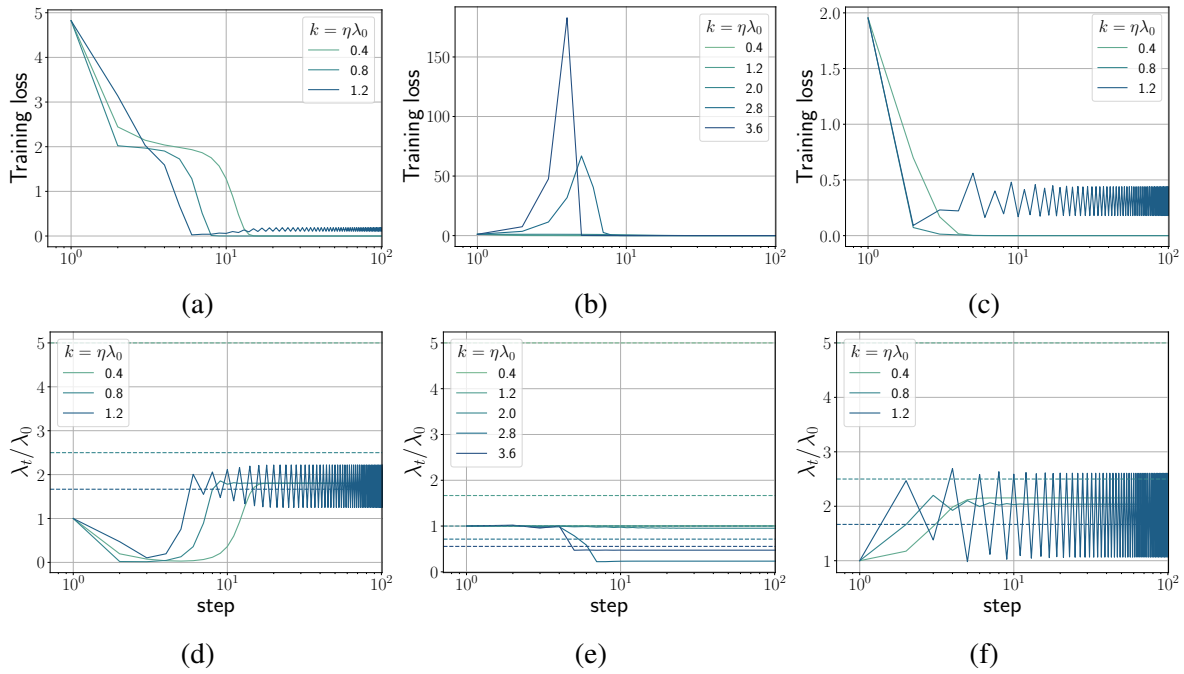


Figure B.3: *UV model shows all four training regimes:* Training trajectories of the UV model trained on a single example $(\boldsymbol{x}, y)$ with $\|\boldsymbol{x}\| = 1$ and $y = 2$ using MSE loss and gradient: (a, d) NTP with $n = 1$ and $\sigma_w^2 = 0.5$ (b, e) NTP with $n = 512$ and $\sigma_w^2 = 1.0$, and (c, f) μP with $n = 1$ and $\sigma_w^2 = 1.0$.

B.3.6 The distribution of residual and NTK at initialization

In this section, we compute the distribution of Δf and λ for the UV model at initialization. Consider the UV model written in terms of the pre-activation $h(x) = Ux$,

$$f(\boldsymbol{x}; \theta) = \frac{1}{\sqrt{n^{1-p}}} \boldsymbol{v}^T \boldsymbol{h}(\boldsymbol{x}) \tag{B.23}$$

$$\lambda = \frac{1}{n^{1-p}} \left(\|\boldsymbol{v}\|^2 \|\boldsymbol{x}\|^2 + \|\boldsymbol{h}\|^2 \right), \tag{B.24}$$

with $v_i, U_{ij} \sim \mathcal{N}(0, \sigma_w^2/n^p)$. Then, each pre-activation h_i is normally distributed at initialization with zero mean and variance

$$\mathbb{E}_\theta[h_i^2] = \sum_{j,k=1}^{d_{\text{in}}} \langle U_{ij} U_{ik} \rangle x_j x_k = \sum_{j,k=1}^{d_{\text{in}}} \frac{\sigma_w^2}{n^p} \delta_{jk} x_j x_k = \frac{\sigma_w^2 \|\boldsymbol{x}\|^2}{n^p}. \tag{B.25}$$

Hence, each pre-activation is distributed as $h_i \sim \mathcal{N}(0, {}^{\sigma_w^2 \|\boldsymbol{x}\|^2}/_{n^p})$. It follows that the network output is also normally distributed at initialization with zero mean and variance

$$\mathbb{E}_\theta[f_0^2] = \frac{1}{n^{1-p}} \sum_{i,j=1}^{n} \langle v_i v_j \rangle \langle h_i h_j \rangle = \frac{1}{n^{1-p}} \sum_{i=1}^{n} \frac{\sigma_w^2}{n^p} \frac{\sigma_w^2 \|\boldsymbol{x}\|^2}{n^p} = \frac{\sigma_w^4 \|\boldsymbol{x}\|^2}{n^p}. \tag{B.26}$$

Hence, the residual at initialization is distributed as $\Delta f_0 \sim \mathcal{N}\left(-y, {}^{\sigma_w^4 \|\boldsymbol{x}\|^2}/_{n^p}\right)$. Similarly, we can also compute the distribution of λ at initialization. The mean value of λ is given by

$$\mathbb{E}_\theta[\lambda_0] = \frac{1}{n^{1-p}} \left(\|\boldsymbol{x}\|^2 \langle \|\boldsymbol{v}\|^2 \rangle + \langle \|\boldsymbol{h}\|^2 \rangle \right) = 2\sigma_w^2 \|\boldsymbol{x}\|^2, \tag{B.27}$$

where we have used $\langle \|\boldsymbol{v}\|^2 \rangle = \sigma_w^2 n^{1-p}$ and $\langle \|\boldsymbol{h}\|^2 \rangle = \sigma_w^2 \|\boldsymbol{x}\|^2 n^{1-p}$. Using similar computations, the second moment of λ is given by:

$$\mathbb{E}_\theta[\lambda_0^2] = \frac{1}{n^{2-2p}} \left(\|\boldsymbol{x}\|^4 \langle \|\boldsymbol{v}\|^4 \rangle + \langle \|\boldsymbol{h}\|^4 \rangle + 2\|\boldsymbol{x}\|^2 \langle \|\boldsymbol{v}\|^2 \|\boldsymbol{h}\|^2 \rangle \right) = \frac{4(n+1)}{n} \sigma_w^4 \|\boldsymbol{x}\|^4 . \tag{B.28}$$

Hence, the λ at initialization is distributed as $\lambda_0 \sim \mathcal{N}\left(2\sigma_w^2 \|\boldsymbol{x}\|^2, {}^{4\sigma_w^4 \|\boldsymbol{x}\|^4}/_{n}\right)$.

B.3.7 EoS manifold is a dynamical attractor

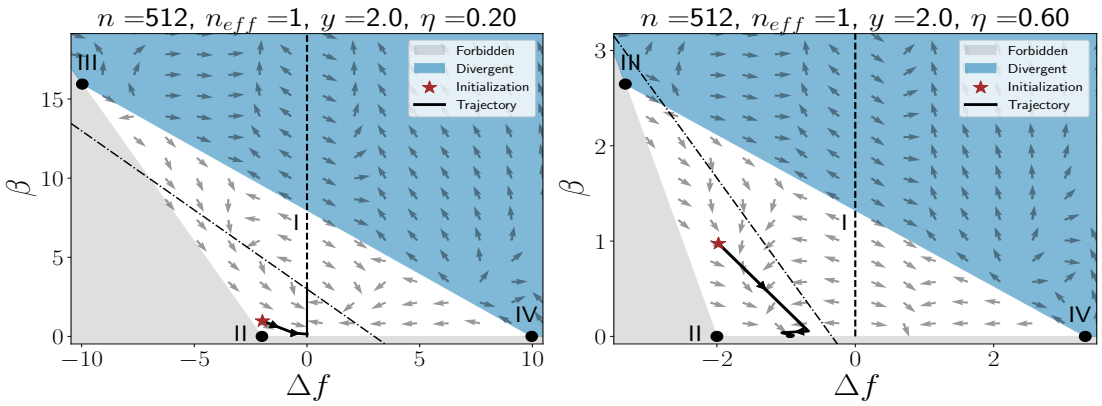
To demonstrate that late time trajectories for $\eta > \eta_c$ converge to the EoS manifold, we define $\beta := \frac{\sqrt{n_{\text{eff}}}}{2\|\boldsymbol{x}\|}\lambda - (\Delta f + y)$. β lies on the direction orthogonal to the EoS manifold, such that $\beta = 0$ corresponds to the manifold itself, while $\beta < 0$ is forbidden. Under this transformation, β updates as $\beta_{t+1} = \beta_t(1 + \frac{\eta\|\boldsymbol{x}\|\Delta f_t}{\sqrt{n_{\text{eff}}}})^2$. It follows that $\beta^* = 0$ stays invariant under the dynamics and defines a nullcline.

Due to oscillations in Δf near convergence, it is instructive to examine the two-step dynamics [80, 81], compactly denoted as $(\Delta f_{t+2}, \lambda_{t+2}) := M^{(2)}(\Delta f_t, \lambda_t)$. Figure B.4 shows the two-step trajectories and the corresponding vector field $\hat{G}^{(2)}(\Delta f, \beta)$ in the $(\Delta f, \beta)$ plane.

We observe that there exists a critical η_c such that for $\eta < \eta_c$, $\hat{G}^{(2)}(\Delta f, \beta)$ points towards the stable zero-loss line (see Figure B.4(a)). By comparison, for $\eta > \eta_c$, all points along the zero-loss line become unstable and the vector field directs towards points on the $\beta = 0$ line, as shown in Figure B.4(b). The critical η_c for which all points on the zero-loss line become unstable thus gives a necessary condition for EoS:

B.3.8 Critical learning rate for edge of stability

In this section, we estimate the required condition on the learning rate for the UV model to exhibit EoS. We specifically focus on the case with $y > 0$ as for $y = 0$, λ can only decrease. As a result, the model does not exhibit progressive sharpening and EoS. In Section 3.5, we observed that the EoS occurs as the zero-loss minima with the smallest λ becomes unstable. From Equation (B.22) it follows that the smallest λ with zero loss is



(a) (b)

Figure B.4: These plots are equivalent to Figure 3.2(a, d) ($\eta_c = 0.5$), but with training trajectory and local vector field plotted for every other step in $(\Delta f, \beta)$ plane. The tilted dash-dotted line indicates the $\eta\lambda = 2$ line.

$$\lambda_{\text{min}} = \frac{2\|\boldsymbol{x}\|y}{\sqrt{n_{\text{eff}}}}. \tag{B.29}$$

This minimum becomes unstable if the learning rate η exceeds a critical value η_c, given by

$$\eta_c = \frac{\sqrt{n_{\text{eff}}}}{\|\boldsymbol{x}\|y} \tag{B.30}$$

It is worth noting that this is a necessary condition for λ to oscillate around $2/\eta$. Otherwise, training converges to the zero-loss minimum with $\lambda = \lambda_{\text{min}}$ for $\eta < \eta_{\text{max}}$.

We can also derive the exact same result by analyzing the dynamics on the EoS manifold. As discussed in Section 3.5.2, the dynamics on the EoS manifold is given by the map $\Delta f_{t+1} = M(\Delta f_t)$, where

$$M(\Delta f) = \Delta f \left(1 - \frac{2\eta\|\boldsymbol{x}\|}{\sqrt{n_{\text{eff}}}}(\Delta f + y) + \left(\frac{\eta\|\boldsymbol{x}\|}{\sqrt{n_{\text{eff}}}}\right)^2 \Delta f(\Delta f + y)\right). \tag{B.31}$$

As demonstrated in Section 3.5.2, EoS in the UV model follows the period doubling route to chaos, with the period two cycle marking the onset. Hence, the conditions required for the emergence of the period two cycle are also necessary conditions for EoS. Consider the two-step dynamics on the EoS manifold given by the map $M^2(\Delta f) \coloneqq M(M(\Delta f))$. This map has six fixed points (excluding three fixed points of the map M) summarized below

$$\Delta f^* = \frac{\eta\tilde{x}(1 - \eta\tilde{x}y) \pm \sqrt{\eta^2\tilde{x}^2(\eta\tilde{x}y - 1)(3 + \eta\tilde{x}y)}}{2\eta^2\tilde{x}^2} \tag{B.32}$$

$$\Delta f^* = \frac{3 + h(\eta, \tilde{x}, y) \pm \eta\tilde{x}(\pm y + \sqrt{2}\sqrt{-\frac{-5 + h(\eta,\tilde{x},y) + \eta\tilde{x}y(-2 - \eta\tilde{x}y + h(\eta,\tilde{x},y))}{\eta^2\tilde{x}^2}})}{4\eta\tilde{x}}. \tag{B.33}$$

Here $\tilde{x} = \frac{\|\boldsymbol{x}\|}{\sqrt{n_{\text{eff}}}}$ and $h(\eta, \tilde{x}, y) = \sqrt{-7 + \tilde{x}y\eta(2 + \tilde{x}y\eta)}$. For the fixed points to exist, we require the expressions inside the square root to be non-negative, i.e.,

$$\left(\frac{\eta\|\boldsymbol{x}\|y}{\sqrt{n_{\text{eff}}}} - 1\right)\left(\frac{\eta\|\boldsymbol{x}\|y}{\sqrt{n_{\text{eff}}}} + 3\right) \geq 0 \implies \eta \geq \eta_1 = \frac{\sqrt{n_{\text{eff}}}}{\|\boldsymbol{x}\|y} \tag{B.34}$$

$$\frac{\eta\|\boldsymbol{x}\|y}{\sqrt{n_{\text{eff}}}}\left(\frac{\eta\|\boldsymbol{x}\|y}{\sqrt{n_{\text{eff}}}} + 2\right) - 7 \geq 0 \implies \eta \geq \eta_2 = \frac{(\sqrt{32} - 2)}{2}\frac{\sqrt{n_{\text{eff}}}}{\|\boldsymbol{x}\|y}. \tag{B.35}$$

As $\eta_1 < \eta_2$, the necessary condition for the period two cycle to emerge is $\eta > \eta_1 = \sqrt{n_{\text{eff}}}/\|\boldsymbol{x}\|y$, which coincides with the condition obtained earlier in this section.

B.3.9 Sub-quadratic Loss

In this section, we detail how the UV model naturally leads to a sub-quadratic loss on the EoS manifold demonstrated by [82].

First, recall that the EoS manifold that connects fixed points II and IV satisfies the relation $\lambda = {}^{2\|\boldsymbol{x}\|(\Delta f+y)}/_{\sqrt{n_{\text{eff}}}}$. Then:

$$\text{1. Starting Relation:} \quad \lambda = \frac{2\|\boldsymbol{x}\|(\Delta f + y)}{\sqrt{n_{\text{eff}}}} \tag{B.36}$$

$$\text{2. Solving for } \Delta f: \quad \Delta f = \frac{\lambda\sqrt{n_{\text{eff}}}}{2\|\boldsymbol{x}\|} - y \tag{B.37}$$

$$\text{3. Factoring Out } y: \quad \Delta f = y\left(\frac{\lambda\sqrt{n_{\text{eff}}}}{2\|\boldsymbol{x}\|y} - 1\right) \tag{B.38}$$

$$\text{4. Defining } \eta_c: \quad \eta_c = \frac{\sqrt{n_{\text{eff}}}}{\|\boldsymbol{x}\|y} \tag{B.39}$$

$$\text{5. Expressing } \Delta f \text{ with } \eta_c: \quad \Delta f = y\left(\frac{\eta_c\lambda}{2} - 1\right) \tag{B.40}$$

$$\text{6. Computing the Loss:} \quad L = \frac{1}{2}\Delta f^2 = \frac{y^2}{2}\left(\frac{\eta_c\lambda}{2} - 1\right)^2 \tag{B.41}$$

As we mentioned in the main text, since $\lambda \sim O(\|\theta\|^2)$, the loss is sub-quadratic near its minimum.

B.4 Additional Results on Sharpness Dynamics

B.4.1 Sharpness Dynamics of NTP Networks

Figure B.5 shows that the sharpness dynamics of FCNs in NTP aligns with the behavior of FCNs in SP demonstrated in Figure 3.1.

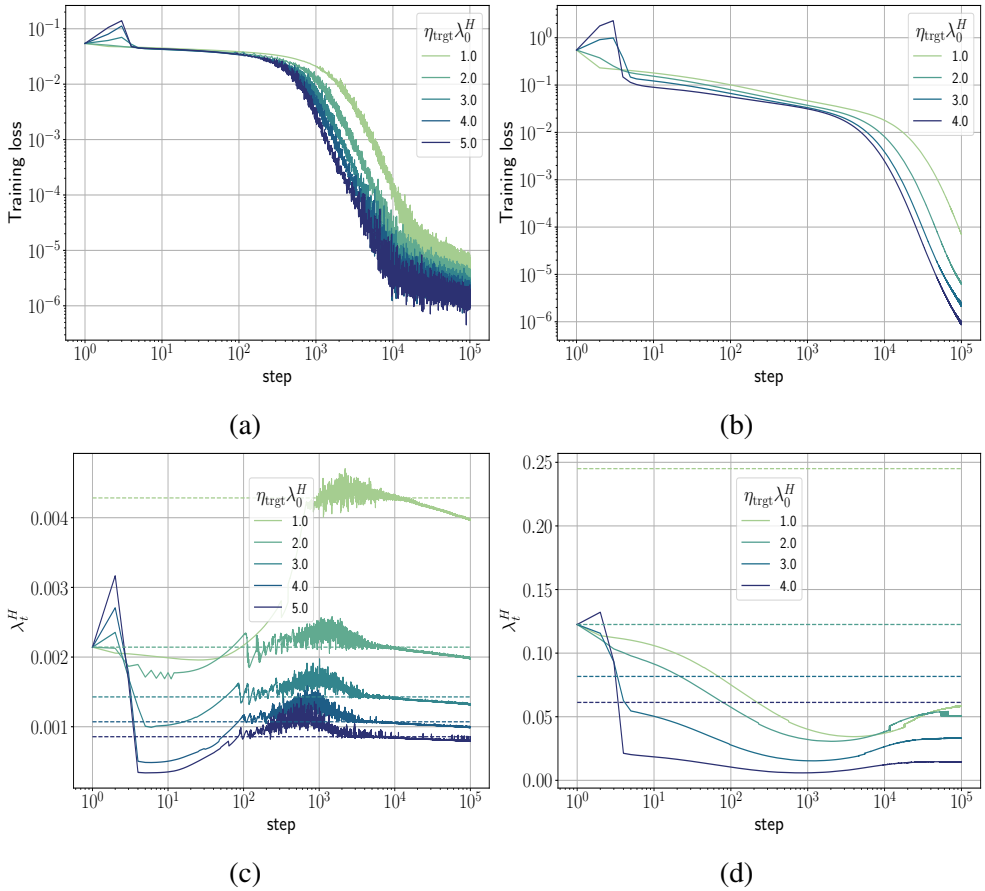


Figure B.5: Training loss and sharpness trajectories of ReLU FCNs in NTP trained on a 5k subset of CIFAR-10 examples using MSE loss and GD: (a, c) $\sigma_w^2 = 0.5$, (b, d) $\sigma_w^2 = 2.0$.

B.4.2 The effect of batch size on the four training regimes

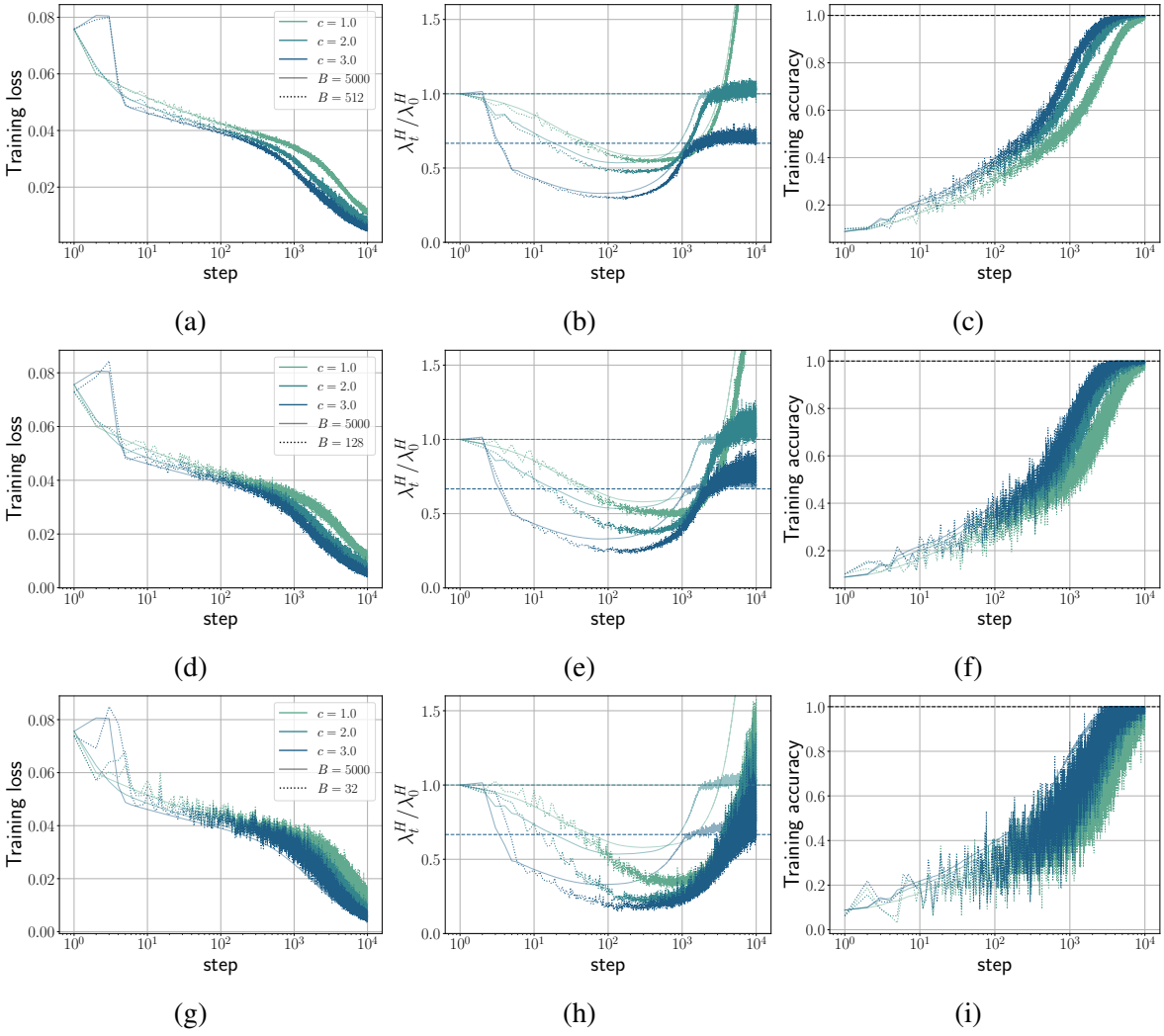


Figure B.6: Comparison of SGD trajectories with their GD counterpart for a 3-layer FCNs in SP with $\sigma_w^2 = 0.5$ trained on a subset of CIFAR-10 consisting of $5,000$ training examples with MSE loss. The learning rate is scaled as $\eta = c/\lambda_0^H$ and batch sizes (a-c) $B = 512$, (d-f) $B = 128$, (g-i) $B = 32$ are considered. GD trajectories are plotted using solid lines with transparency.

In this section, we examine the effect of batch size B on the results presented in the main text. We find that our conclusions are robust for reasonable batch sizes around $B \approx 512$. For even smaller batch sizes, the dynamics becomes noise-dominated, and separating the inherent dynamics from noise becomes challenging. This observation further supports the use of SGD

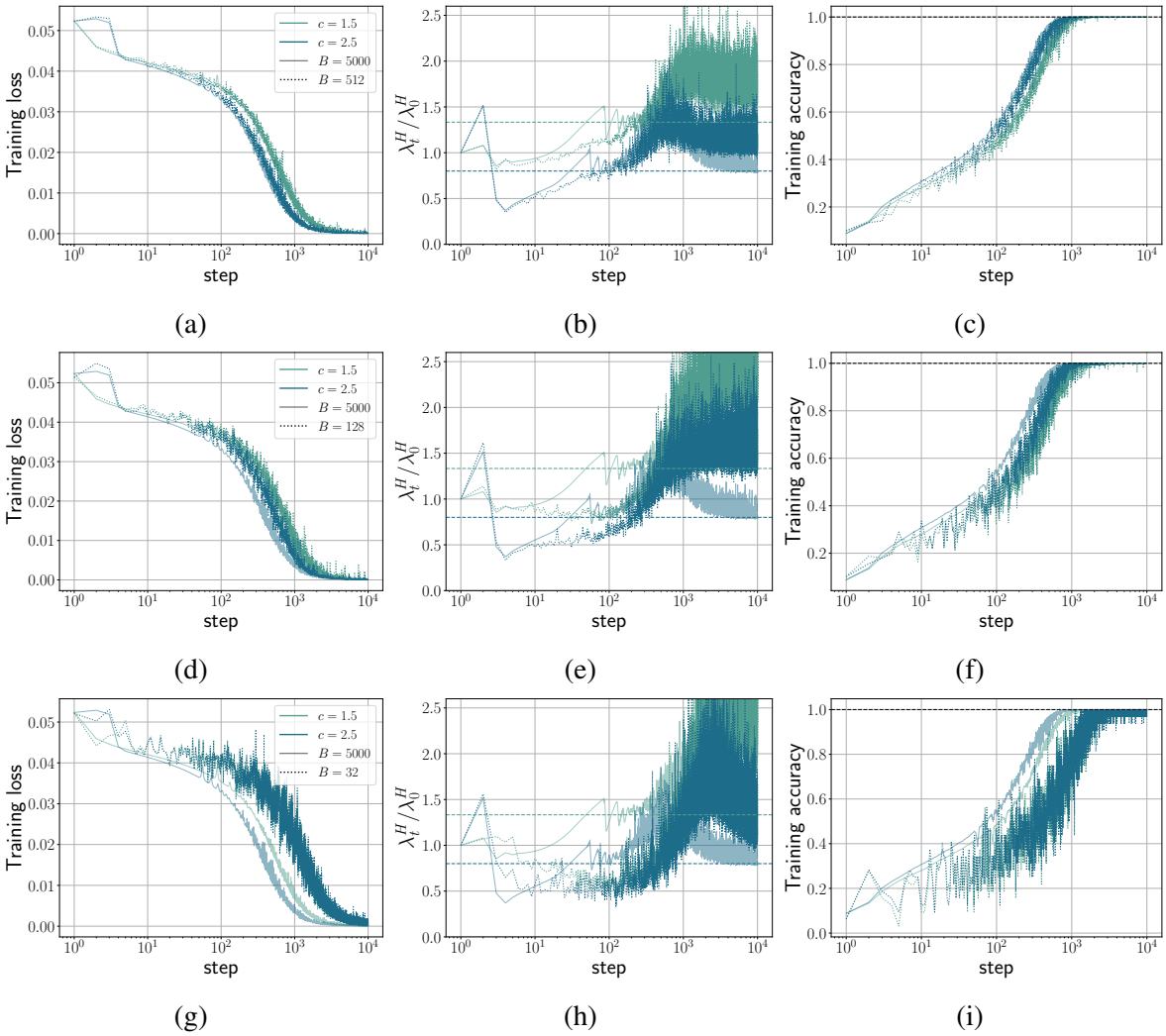


Figure B.7: Same setting as Figure B.6, except we used 3-layer FCNs in μP with $\sigma_w^2 = 2.0$.

to reduce the computational cost of experiments in the subsequent sections involving CNNs and ResNets.

Figure B.6 shows that SGD trajectories of FCNs in SP begin to deviate from their GD counterpart significantly for batch sizes around $B \approx 128$. In contrast, for μP networks this deviation begins at a larger batch size of $B \approx 512$ as shown in Figure B.7.

B.4.3 Generizability of the Four Training Regimes

This section shows that the four regimes are generically observed for different architectures, loss functions, and datasets. Figure B.8 shows training trajectories of CNNs and ResNets trained SGD with batch size $B = 512$. Figure B.9 show the training trajectories of Transformers trained on the WikiText-2 dataset using the cross-entropy loss. These results further exemplify that four regimes of training are generically observed for complex architectures, datasets, and loss functions.

B.5 Sharpness-weight norm correlation

Section 3.5.1 reveals that several aspects of the training dynamics are controlled by the fact that, for a wide variety of initializations, at early times trajectories move closer to the saddle point II, resulting in an interim decrease in λ (also proportional to the weight norm in this case), before eventually increasing. This critical point where all parameters are zero also exists in real-world models. We thus anticipate that in real-world models, the origin of the four training regimes may be related to a similar mechanism. This would predict a decrease in weight norm as training passes near the saddle point, followed by an eventual increase.

Figure B.10 validates this hypothesis. During the sharpness reduction and intermediate saturation regimes, we see a decrease in the weight norm, followed by an increase in the weight norm as the network undergoes progressive sharpening, following the prediction from the UV model. Similar correlations between the last layer weight norm and sharpness are utilized by [85] to analyze the EoS phase. By comparison, we focus on the correlation between sharpness and weight norm during early training to attribute the emergence of four regimes to the critical

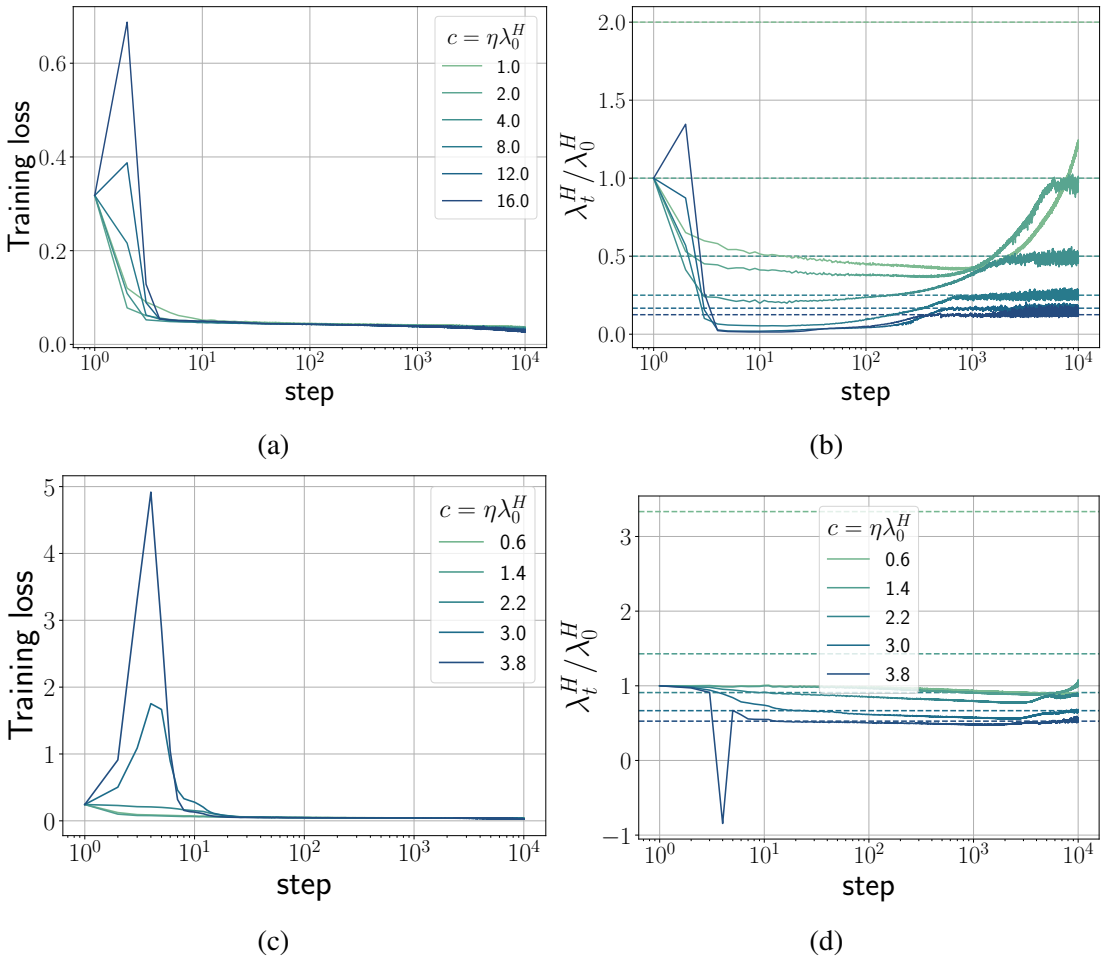


Figure B.8: Training trajectories of (a, b) a 5-layer CNN in SP with $n = 64$, and (c, d) ResNet-18 with LayerNorm in SP, also with $n = 64$. Both models are trained on the CIFAR-10 dataset with MSE loss using SGD. The learning rate is scaled as $\eta = c/\lambda_0^H$ and batch size is $B = 512$. In panel (d), λ_t^H becomes negative during early training. This is due to the power iteration method returning the largest eigenvalue by magnitude.

point corresponding to all parameters being zero. In Appendix B.5, we provide further evidence for this correlation between sharpness and weight norm, extending this relationship to CNNs and ResNets.

This section presents additional results for Section 3.6, further supporting the relationship between sharpness and weight norm during training.

Figure B.11 is an extended version of Figure B.10, where we plotted the whole training trajectories and measured Pearson correlation

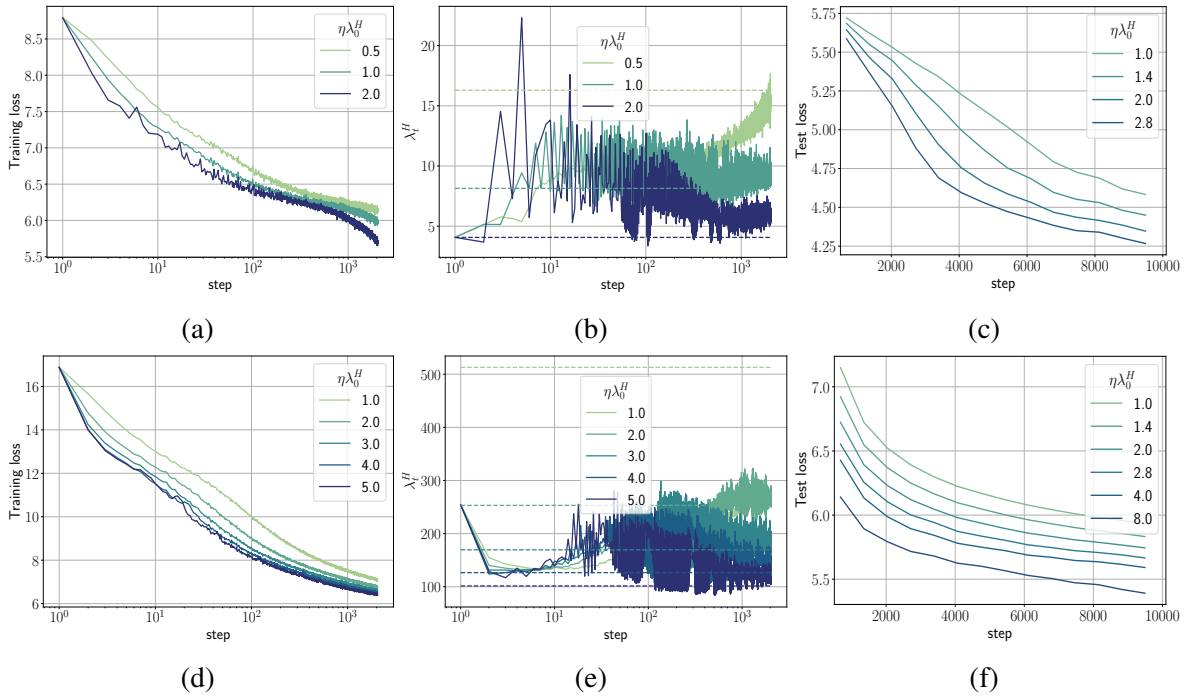


Figure B.9: Training loss, sharpness and test loss trajectories of Transformers trained on Wikitext-2 using cross-entropy loss: (a, b, c) Pre-LN Transformers and (d, e, f) Pre-LN Transformer without last LayerNorm.

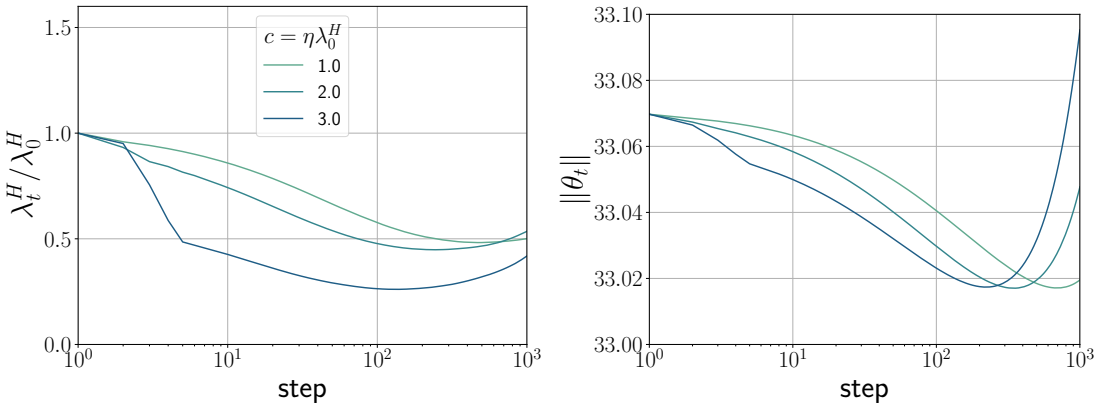


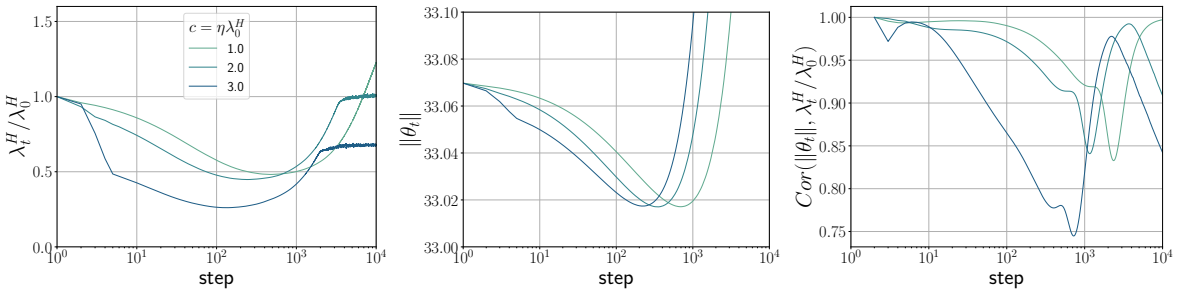
Figure B.10: Sharpness and Weight Norm of 3-layer ReLU FCNs in SP with $\sigma_w^2 = 1/3$, trained on a subset of CIFAR-10 with $5,000$ examples using GD.

$$\mathrm{Cor}(\|\theta_t\|, \lambda_t^H/\lambda_0^H) := \frac{\sum_{t'=1}^{t} \left(\theta_{t'} - \bar{\theta}_t\right) \left(\lambda_{t'}^H/\lambda_0^H - \overline{(\lambda^H/\lambda_0^H)}_t\right)}{\sqrt{\sum_{t'=1}^{t} \left(\theta_{t'} - \bar{\theta}_t\right)^2 \sum_{t'=1}^{t} \left(\lambda_{t'}^H/\lambda_0^H - \overline{(\lambda^H/\lambda_0^H)}_t\right)^2}} \tag{B.42}$$

Here $t \geq 2$ and $\bar{\theta}_t = (\sum_{t'=1}^{t} \theta_{t'})/t$.

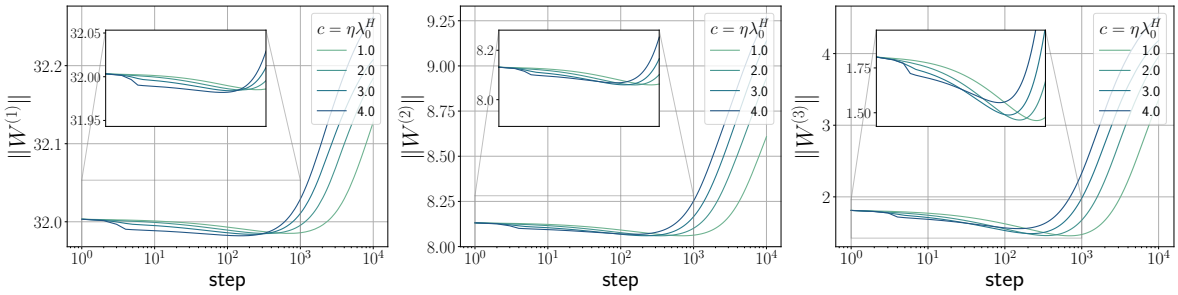
Figure B.12 shows the weight norm of each layer separately for the experiment in Figure B.10. This result shows a high correlation between weight norm and sharpness through training.

We also confirm these correlations between weight norm and sharpness in CNNs for the experiment in Figure B.8(a, b).



(a) (b) (c)

Figure B.11: Sharpness and Weight Norm of 3-layer ReLU FCNs in SP with $\sigma_w^2 = 1/3$, trained on a subset of CIFAR-10 with $5,000$ examples using GD.



(a) (b) (c)

Figure B.12: Weight Norm of each layer in 3-layer ReLU FCNs (same experiments as Figure B.10): (a, b, c) SP with $\sigma_w^2 = 1/3$. All results are obtained by training on a subset of CIFAR-10 with $5,000$ examples using GD.

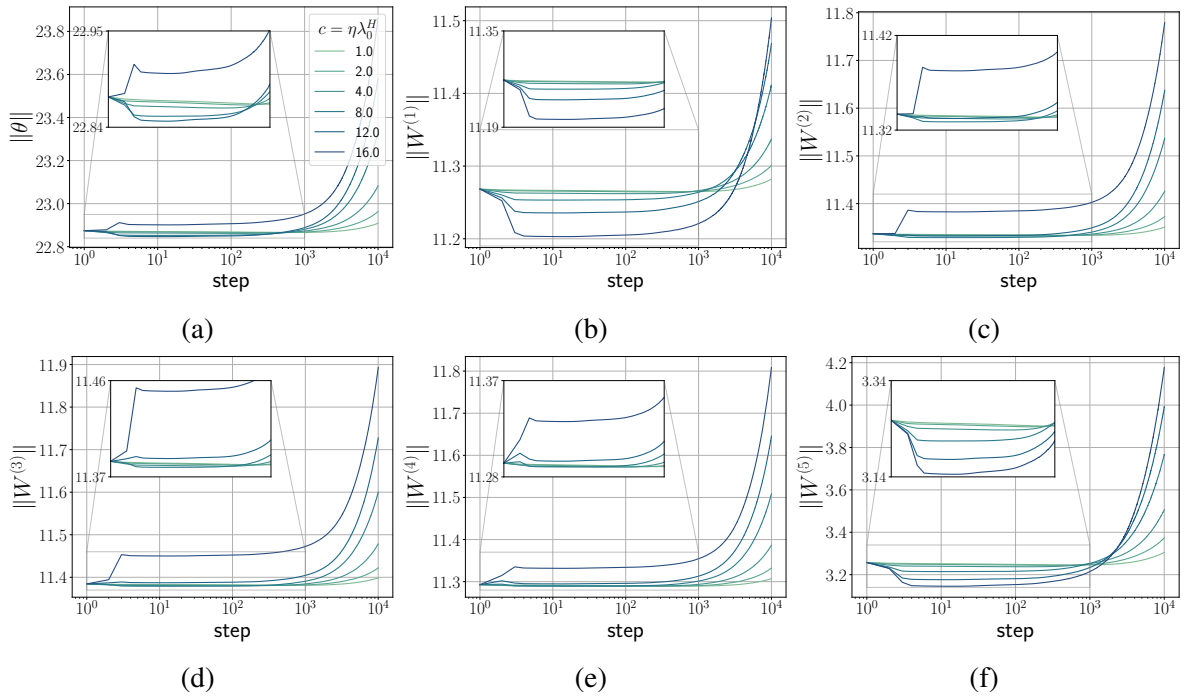


Figure B.13: Weight Norm of each layer for 5-layer CNNs in SP (same experiments as Figure B.8(a, b): (a) Total weight norm; (b-f) Weight norm of each layer. We see that for $c = 16$, the initial catapult in sharpness λ^H (Figure B.8(b)) is accompanied by a catapult in total weight norm. Notably, the total weight norm and per-layer weight norm, whether catapults (a, c-e) or not (b, f), show a decreasing trend during the early sharpness decreasing stage, followed by an eventual increase.

B.5.1 Setting the last layer to zero eliminates sharpness reduction

Figure B.14 shows setting the last layer to zero eliminates sharpness reduction during early training.

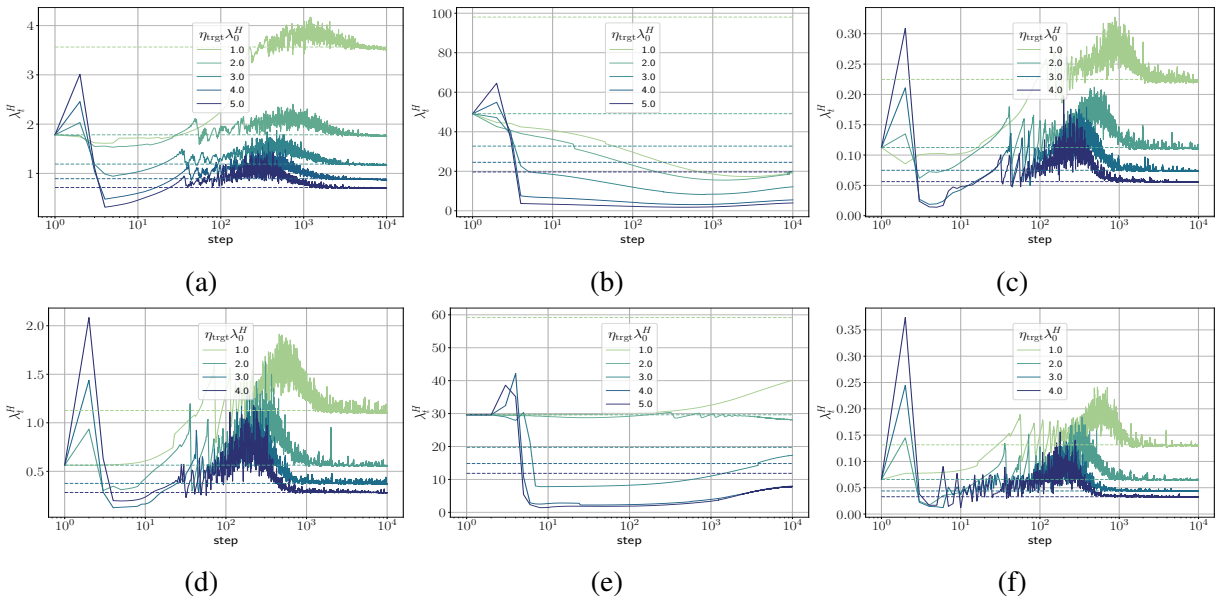


Figure B.14: Sharpness trajectories of ReLU FCNs trained on a 5k subset of CIFAR-10 examples using MSE loss and GD. (a, b, c) SP with $\sigma_w^2 = 0.5$, SP with $\sigma_w^2 = 2.0$ and μP with $\sigma_w^2 = 2.0$. (d, e, f) the bottom row shows the effect of setting the last layer to zero at initialization. The dashed lines in the sharpness figures show the $2/\eta$ threshold.

B.6 Additional Phase diagrams of EoS

This section demonstrates additional phase diagrams of EoS and quantifies the effect of batch size in the EoS regime. Figure B.15 shows phase diagrams of EoS for FCNs trained on CIFAR-10 with MSE loss using SGD for $10,000$ steps for three different batch sizes. We observe that as the batch size decreases, λ_t^H oscillates at a value different from $2/\eta$ depending on σ_w^2 and s. For large σ_w^2 and small s, λ^H favors a smaller value for smaller batch size, which is in agreement with the observation in [71]. In contrast, λ^H can be larger than $2/\eta$ for small σ_w^2 and large s at late training times.

Figures B.16 and B.17 show the phase diagrams of EoS for CNNs and ResNets trained on the CIFAR-10 dataset with MSE loss using SGD for $10,000$ steps with learning rate $\eta = c/\lambda_0^H$ and batch size $B = 512$. In contrast to the FCN phase diagrams, these architectures exhibit EoS

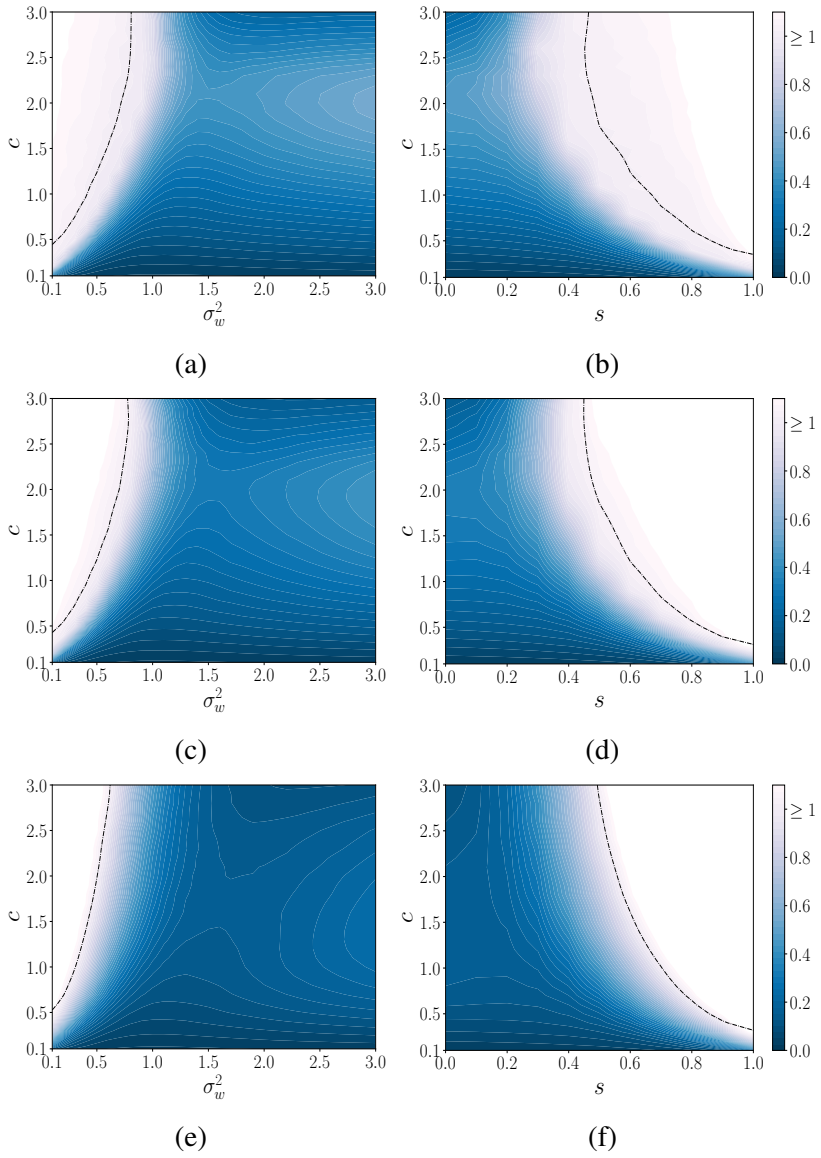
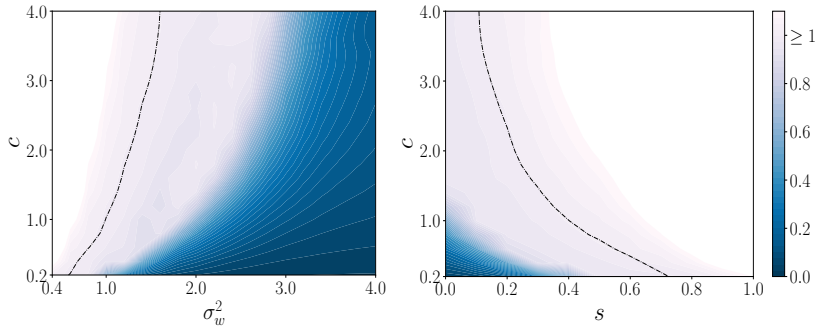


Figure B.15: Phase diagram of EoS for 3-layer FCNs trained on CIFAR-10 with MSE loss using SGD with three different batch sizes: (a, b) $B = 512$, (c, d), $B = 128$, and (e, f) $B = 32$. The color indicates the value of $\eta\bar{\lambda}^H/2$, where $\bar{\lambda}^H$ is obtained by averaging λ_t^H over the last 200 steps. Except for the batch size, all settings are identical to Figure 3.4. Black dash-dotted lines indicate the phase boundary $\eta\bar{\lambda}^H/2 = 1$. For clarity, these lines are generated from data smoothed with a Gaussian kernel.

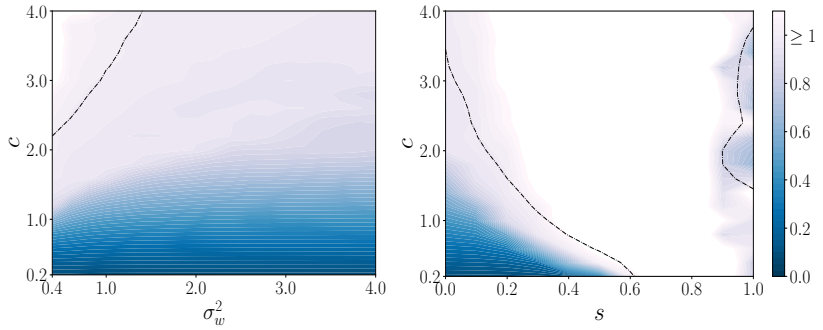
behavior at smaller values of s and larger values of σ_w^2, indicating their implicit bias towards EoS. Moreover, we observe in ResNets, that EoS is less sensitive to change σ_w^2, likely due to a combination of LayerNorm and residual connections [1].

It is worth noting that EoS boundaries in these phase diagrams are time-dependent. For instance, models close to the EoS boundary may eventually reach EoS on training longer (see Figure B.8(b) $c = 1.0$ for example), causing a shift in the EoS boundary. Nevertheless, models with small learning rates, large σ_w^2, and small s may never show EoS behavior, regardless of training duration, as predicted by the UV model and seen in Figure 3.1(e).



(a) (b)

Figure B.16: Phase diagram of EoS for 5-layer CNNs in SP with width $n = 64$ trained with MSE loss using SGD for $10,000$ steps with learning rate $\eta = c/\lambda_0^H$ and batch size $B = 512$.



(a) (b)

Figure B.17: Phase diagram of EoS for ResNet-18 in SP with width $n = 64$ trained with MSE loss using SGD for $10,000$ steps with learning rate $\eta = c/\lambda_0^H$ and batch size $B = 512$. For $s = 1$, the average eigenvalue $\bar{\lambda}^H$ is observed to be less than $2/\eta$. Upon detailed investigation of the trajectories, we found that λ_t^H oscillates around a value lower than $2/\eta$. We leave this anomalous behavior as an observation.

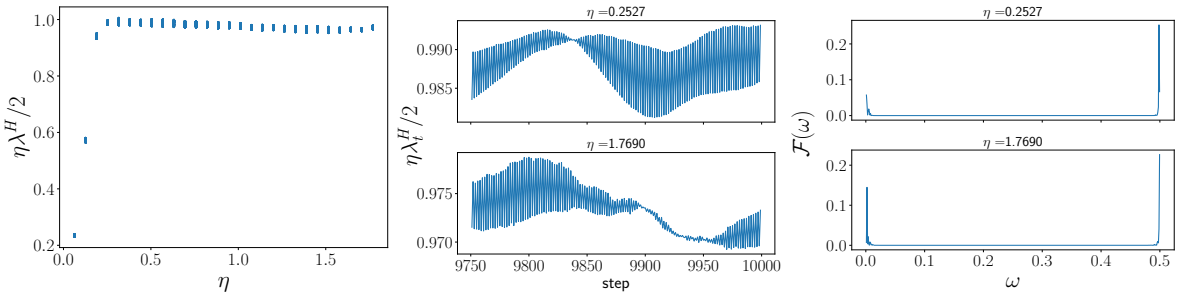
B.7 Route to chaos

B.7.1 Route to EoS in real datasets

This section presents additional bifurcation diagrams for different architectures and datasets. These results show the reminiscent of the period-doubling route to chaos observed in different architectures and datasets. In all figures, we choose the smallest and largest learning rate exhibiting EoS for plotting the trajectories and power spectrum. The structured route to chaos in realistic experiments can be disrupted due to a variety of reasons. Below, we discuss a few of them.

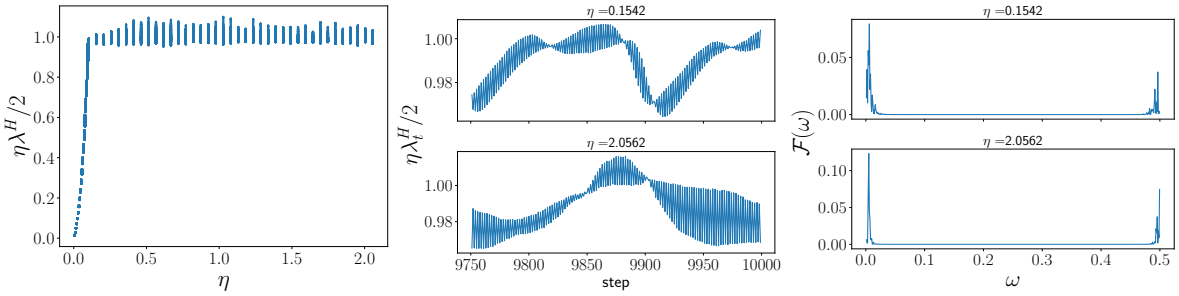
Measurement of only the top eigenvalue of Hessian: In our experiments, we only measured the top eigenvalue of the Hessian. However, when multiple eigenvalues of Hessian enter EoS, plotting only the top eigenvalue of Hessian is a projection that could obscure all the structured routes to chaos that the system may exhibit.

The effect of correlations in real-world datasets: Real-world datasets inherently contain correlations between different samples $(\boldsymbol{x}, \boldsymbol{y})$. These correlations can be quantified using the input-input covariance matrix $\Sigma_{XX} = XX^T \in \mathbb{R}^{d_{\text{in}} \times d_{\text{in}}}$ and output-input covariance matrices $\Sigma_{YX} = YX^T \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$. In Appendix B.7.2, we find that a key determining factor in observing route-to-chaos is whether the power spectrum of Σ_{XX} is flat or exhibits power law decay. We show that power-law decay in the singular values of the Σ_{XX} results in long-range correlations in time and dense sharpness bands observed in real datasets.



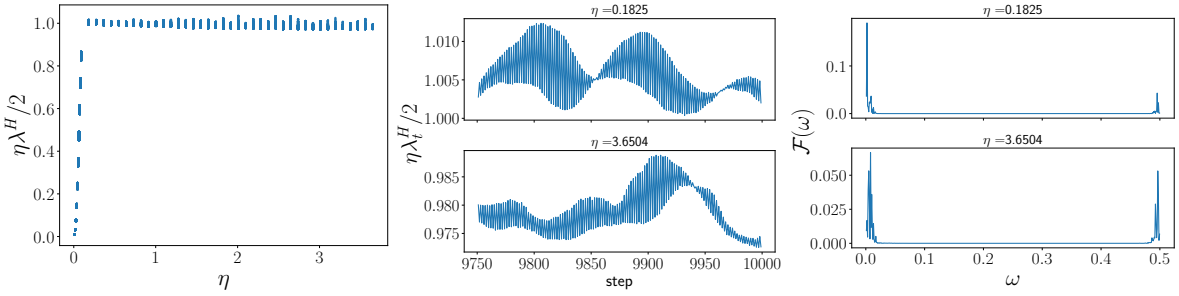
(a) (b) (c)

Figure B.18: 4-layer FCN in SP with width $n = 512$ trained on a subset of 5000 examples of MNIST with MSE loss using GD. Both power spectrums are computed using the last 1000 steps of the corresponding trajectories.



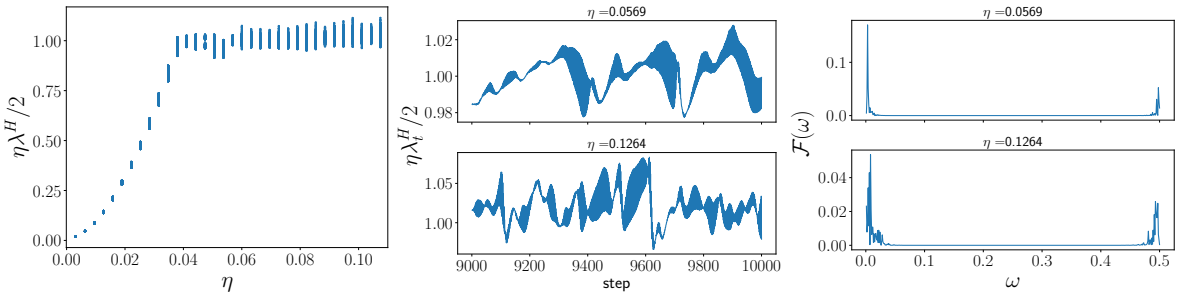
(a) (b) (c)

Figure B.19: 4-layer FCN in SP with width $n = 512$ trained on a subset of 5000 examples of Fashion-MNIST with MSE loss using GD. Both power spectrums are computed using the last 1000 steps of the corresponding trajectories.



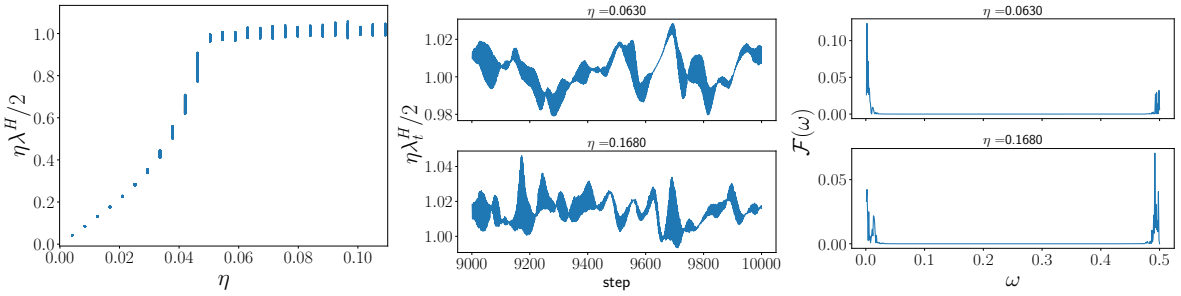
(a) (b) (c)

Figure B.20: 4-layer FCN in SP with width $n = 512$ trained on a subset of 5000 examples of CIFAR-10 with MSE loss using GD. Both power spectrums are computed using the last 1000 steps of the corresponding trajectories.



(a) (b) (c)

Figure B.21: 5-layer CNN in SP with width $n = 32$ trained on a subset of 1000 examples of CIFAR-10 with MSE loss using GD.



(a) (b) (c)

Figure B.22: ResNet-18 in SP with width $n = 32$ trained on a subset of 1000 examples of CIFAR-10 with MSE loss using GD.

B.7.2 The effect of power-law trends in data on sharpness trajectories

In this section, we analyze a 2-layer linear FCN trained on the power law dataset described in Appendix B.2.1 to understand the origin of long-range correlations in sharpness trajectories and dense sharpness bands in realistic datasets.

Figure B.23 shows the bifurcation diagram, late time trajectories, and the associated power spectrum of the network trained on the power-law dataset with the same $A_x = 1.0$ and $A_y = 1.0$, for four different combinations of power-law exponents: (i) $B_x = 0.0, B_y = 0.0$, (ii) $B_x = 1.0, B_y = 0.0$, (iii) $B_x = 0.0, B_y = 1.0$, and (iv) $B_x = 1.0, B_y = 1.0$. We observe that a power-law trend to the singular values of the input matrix results in dense sharpness bands observed

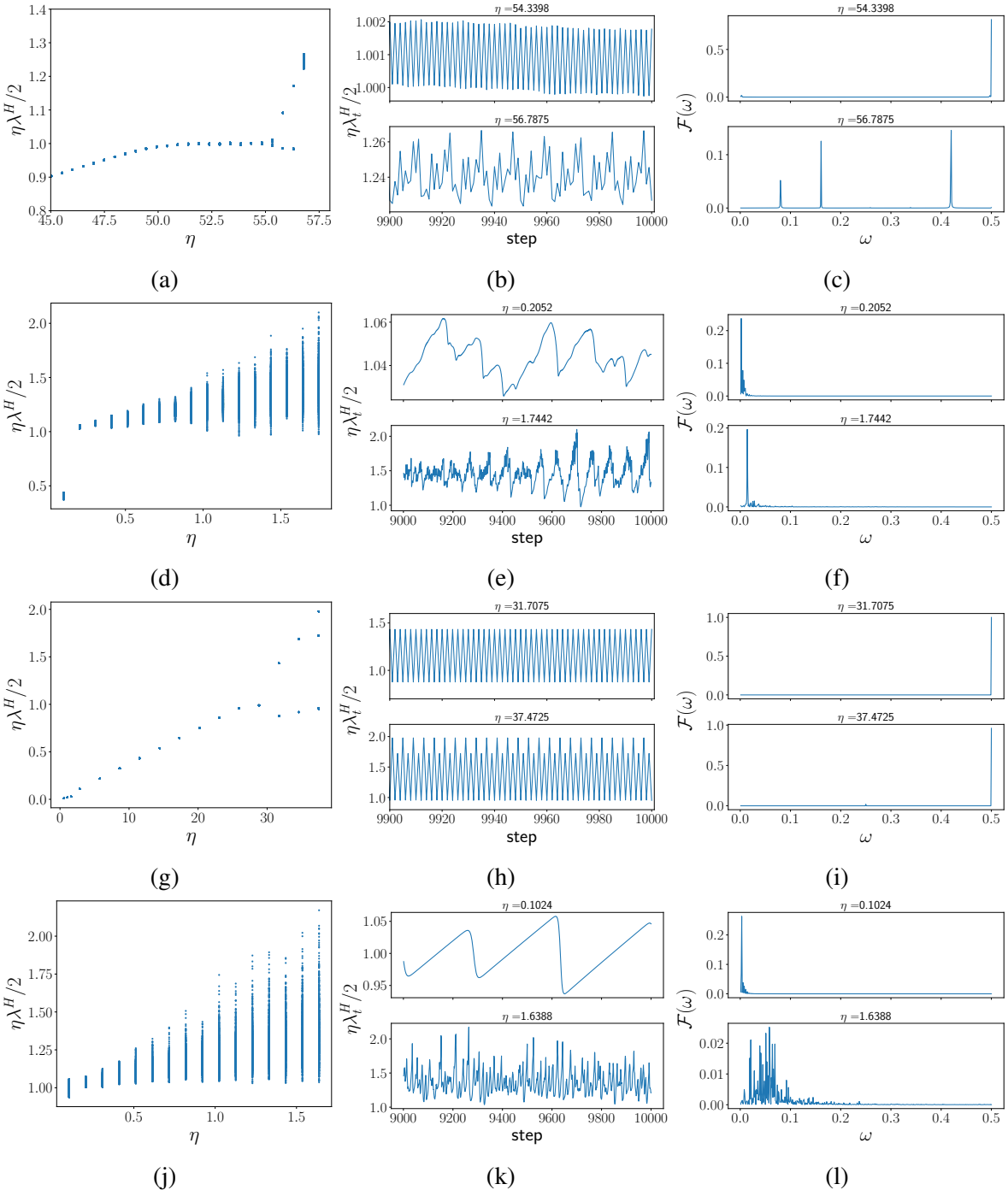


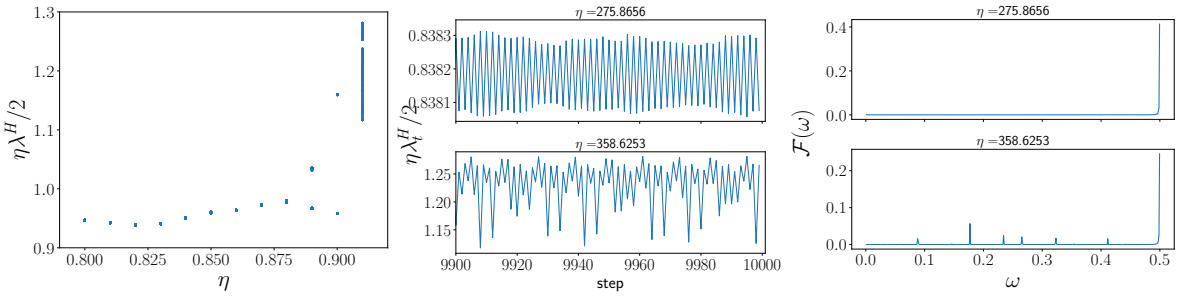
Figure B.23: Bifurcation diagrams, late-time sharpness trajectories, and power spectrum of a 2-layer linear network trained on the power-law dataset for different parameter values: (a-c) $B_x = 0.0, B_y = 0.0$, (d-f) $B_x = 1.0, B_y = 0.0$, (g-i) $B_x = 0.0, B_y = 1.0$, and (j-l) $B_x = 1.0, B_y = 1.0$. All power spectrums are computed using the last 1000 steps of the corresponding trajectories.

in real datasets. It is worth noting that this is one way to obtain dense sharpness bands and in general, there can be many other methods.

B.7.3 Route to chaos in synthetic datasets

In this section, we analyze the route to chaos in synthetic datasets to gain insights into the dense sharpness bands in realistic datasets. We considered two datasets, defined as follows:

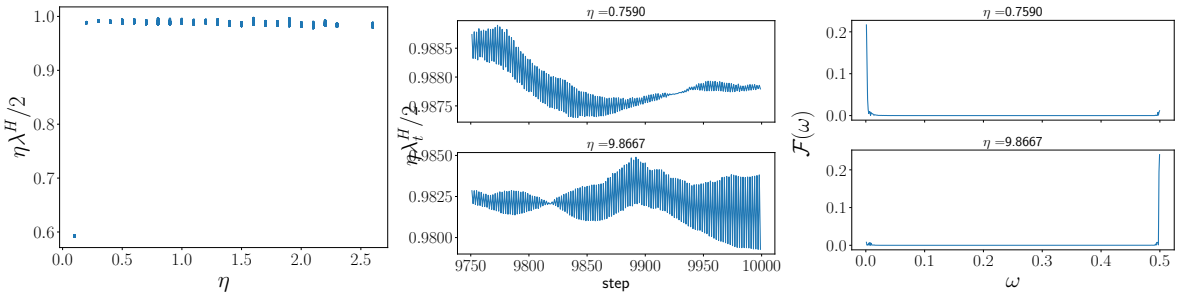
Teacher-student dataset: Consider a teacher FCN $f : \mathbb{R}^{d_{\text{in}}} \to \mathbb{R}^{d_{\text{out}}}$ with $d_{\text{in}} = 3072$, $d_{\text{out}} = 10$, depth d, and width $n = 512$ in Standard Paramaterization. Then, we construct a teacher-student dataset (X, Y) consisting of $P = 5000$ examples with $\boldsymbol{x}^\mu \sim \mathcal{N}(0, I)$ and $\boldsymbol{y}^\mu = \boldsymbol{f}(\boldsymbol{x}^\mu; \theta_0)$. Next, we train a student FCN with the same depth d and depth n as the teacher FCN on this dataset.



(a) (b) (c)

Figure B.24: 2-layer linear FCN in μP trained on the teacher-student task. Both power spectrums are computed using the last 1000 steps of the corresponding trajectories.

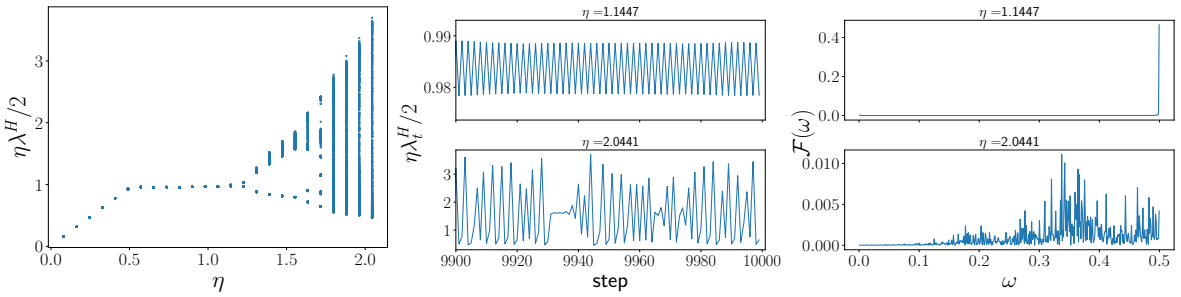
Figures B.24 and B.25 show the bifurcation diagram, late time sharpness trajectories and the associated power spectrum of linear and ReLU FCNs trained on the teacher-student task. These figures show that while linear FCN shows the period-doubling route to chaos, ReLU FCN shows long-range correlations as observed in real datasets.



(a) (b) (c)

Figure B.25: 4-layer ReLU FCNs in μP trained on the teacher-student task. Both power spectrums are computed using the last 1000 steps of the corresponding trajectories.

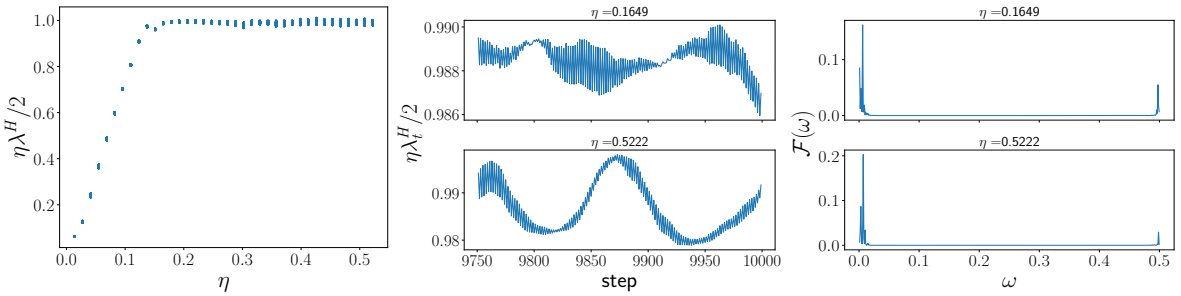
Generative dataset: Consider a 5-layer CNN $\boldsymbol{f}(\boldsymbol{x}, \theta)$ in SP with $n = 64$, trained on the CIFAR-10 dataset with MSE loss using SGD with learning rate $\eta = {}^{12}/_{\lambda_0^H}$ and momentum $m = 0.9$ for 100k steps. This model achieves a test accuracy of 76.9%. Then, we construct a generative image dataset (X, Y) consisting of $P = 5000$ examples with $\boldsymbol{x}^\mu \sim \mathcal{N}(0, I)$ and $\boldsymbol{y}^\mu = f(\boldsymbol{x}^\mu; \theta)$. Next, we train an FCN in SP with depth d, width n, and weight variance $\sigma_w^2 = 0.5$ on the generated dataset.



(a) (b) (c)

Figure B.26: 4-layer linear FCNs in SP with $\sigma_w^2 = 0.5$ trained on the generative CIFAR-10 task with MSE loss using GD. Both power spectrums are computed using the last 1000 steps of the corresponding trajectories.

Figures B.26 and B.27 show the bifurcation diagram, late time trajectories and the associated power spectrum of a 4-layer ReLU FCN with linear and ReLU activations, trained on the generative CIFAR-10 dataset. We observe that while the linear network shows a period-doubling



(a) (b) (c)

Figure B.27: 4-layer ReLU FCNs in SP with $\sigma_w^2 = 0.5$ trained on the generative CIFAR-10 task with MSE loss using GD. Both power spectrums are computed using the last 1000 steps of the corresponding trajectories.

route to chaos, the ReLU shows long-range correlations as observed in real datasets.

Appendix C: Grokking Modular Arithmetic

C.1 Further Experimental Details

In every plot presented, unless explicitly stated otherwise, we use the following default setting:

Default setting. A 2-layer fully connected network with quadratic activation function and width $N = 500$, initial weights sampled from $\mathcal{N}(0, (16N)^{-2/3})$, trained on $p = 97$ Modular Addition dataset. The networks are trained for 2000 optimization steps, using full-batch, MSE loss, AdamW optimizer, learning rate $\eta = 0.01$ and $\beta = (0.9, 0.98)$.

C.1.1 Figures in Main Text

Figure 4.3: Default setting with $\alpha = 0.5, \xi = 0.35$ and (a) $wd = 0$ (b) $wd = 5$ and (c) $wd = 15$.

Figure 4.1: Default setting with $\alpha = 0.5, \xi = 0.0, wd = 5.0$.

Figure 4.4: Each plot is scanned over 17 different data fractions α ranging from 0.1 to 0.9 and 19 noise levels ξ from 0.0 to 0.9 using the default setting.

Figures 4.6 and 4.7: Default setting with $\alpha = 0.5, \xi = 0.35$.

General setting for minibatch training: We train the network with batch size 256 for 2000

steps. For batch sizes 8 and 64, we scale the number of total steps so as to keep the number of epochs roughly equal. (In other words, all the networks see the same amount of data.) The learning rate is scaled as $0.01 \cdot \sqrt{\text{batch size}/256}$.

Figure 4.8: All networks are trained with MSE loss using AdamW optimizer with learning rate $\eta = 0.001$ and $(\beta_1, \beta_2) = (0.9, 0.98)$. (a) Encoder-only Transformer, embedding dimension $d_{\text{embed}} = 128$ with ReLU activation, trained for 3000 steps with 3 random seeds; (b) Width $N = 500$ trained for 5000 steps with 3 random seeds.

C.2 Details of Resources Used

Figure 4.4 requires 72 hours on a single NVIDIA RTX 3090 GPU.

Figure 4.8 used 60 hours on a $1/7$ NVIDIA A100 GPU.

All the remaining figures except Figure 4.8 in the main text are either further experiments performed on the data from the Figure 4.4 experiment; or are very quick to perform on any machine.

We used 300 GPU hours for early-stage exploration and making phase diagrams in the appendix.

C.3 Quantitative Characterization of Phases

All phase diagrams are plotted using the empirical accuracies of trained networks.

Trained networks with $\geq 90\%$ test accuracies get classified into *Coexistence*, *Partial Inversion* and *Full Inversion* based on their training accuracy (specifically, their ability to memorize corrupted examples). *Coexistence* phase has $\geq 90\%$ training accuracy, since it memorizes most

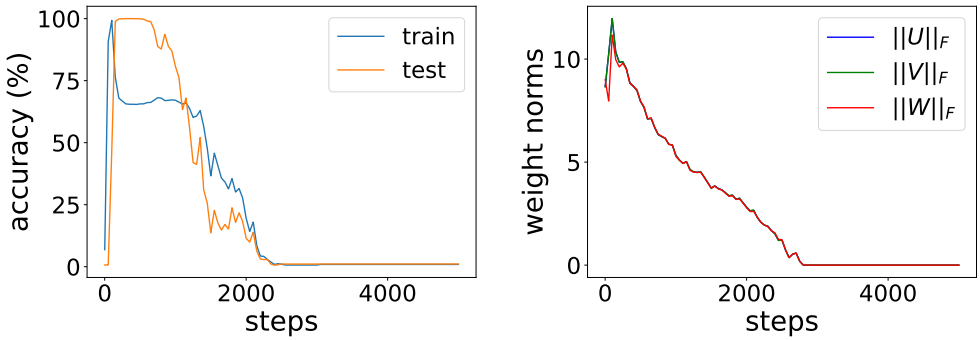
of the corrupted training examples. *Partial Inversion* phase has $< 90\%$ but $\geq 100(1.05 - \xi)\%$ training accuracy, where ξ is the corruption fraction. Note that we subtract ξ from 1.05 instead of 1.00 to account for the $1-2\%$ random guessing accuracy as well as statistical fluctuations around this value. *Full Inversion* phase has $< 100(1.05 - \xi)\%$ training accuracy, since it does not memorize any of the corrupted examples. *Memorization* phase has $< 90\%$ test accuracy but $\geq 90\%$ training accuracy, indicating that the network memorizes all of the training data, including the corrupted and uncorrupted examples. *Forgetting* or *confusion* phase has $< 90\%$ test as well as training accuracies. the network neither generalizes nor manages to memorize all the training examples. We distinguish *Forgetting* from *confusion* using the training curve. In the *Forgetting phase* the grokked network loses its performance due to steadily decreasing weight-norms. Whereas in the *confusion* phase, the network never reaches high performance.

	Coexistence	**Partial Inversion**	**Full Inversion**	**Memorization**	**Forgetting / Confusion**
Test accuracy	$\geq 90\%$	$\geq 90\%$	$\geq 90\%$	$< 90\%$	$< 90\%$
Training accuracy	$\geq 90\%$	$< 90\%$ and $\geq 100(1.05 - \xi)\%$	$< 100(1.05 - \xi)\%$	$\geq 90\%$	$< 90\%$

Table C.1: Quantitative characterization of phases.

C.4 Forgetting Phase

We saw in Figure 4.4(a) (right-most panel, wd=20.0) that extremely high weight decay leads to *Forgetting* phase. This phase has random guessing accuracies on both training and test datasets at the end of training. Naively, this would seem similar to the *Confusion* phase. However, Upon close inspection of their training dynamics, we find rich behaviour. Namely, the network initially undergoes dynamics similar to *Full Inversion*, *i.e.*, the test accuracy reaches 100% while the train accuracy plateaus around $100(1 - \xi)\%$. However, both accuracies subsequently decay



(a) (b)

Figure C.1: *Forgetting* phase (data fraction $\alpha = 0.5$, corruption fraction $\xi = 0.35$, weight decay = 20). (a) Training curves. (b) Frobenius norms of network weights with training.

down to random guessing. This curious phenomenon is explained by the effect of the high weight decay on the network weights. We notice that the Frobenius norms of the weights of the network steadily decay down to arbitrarily small values.

This curious behaviour only occurs when we use activation functions of degree higher than 1 (e.g. $\phi(x) = x^2$ in this case). (More precisely, we mean that the first term in the series expansion should be of degree at least 2.) This can be intuitively explained by examining the gradients carefully. With quadratic activation functions and MSE loss, gradients take the following form

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial U_{ab}} &= \frac{4}{|D|p} \sum_{(b,n)\in D} \sum_{q}^{p} \left(\sum_{k=1}^{N} W_{qk} \left(U_{kb} + V_{kn}\right)^2 - \delta^p(b+n-q) \right) W_{qa} \left(U_{ab} + V_{an}\right) \\
\frac{\partial \mathcal{L}}{\partial V_{ab}} &= \frac{4}{|D|p} \sum_{(m,b)\in D} \sum_{q}^{p} \left(\sum_{k=1}^{N} W_{qk} \left(U_{km} + V_{kb}\right)^2 - \delta^p(b+n-q) \right) W_{qa} \left(U_{am} + V_{ab}\right) \\
\frac{\partial \mathcal{L}}{\partial W_{cd}} &= \frac{4}{|D|p} \sum_{(m,n)\in D} \left(\sum_{k=1}^{N} W_{ck} \left(U_{km} + V_{kn}\right)^2 - \delta^p(m+n-q) \right) \left(U_{dm} + V_{dn}\right)^2 ,
\end{aligned} \tag{C.1}$$

where D is the training dataset and $|D|$ is its size.

For non-generalizing solutions, the term inside the bracket is of degree 0 in weights, due to the delta function term. But close to generalizing solutions, the term inside the bracket becomes

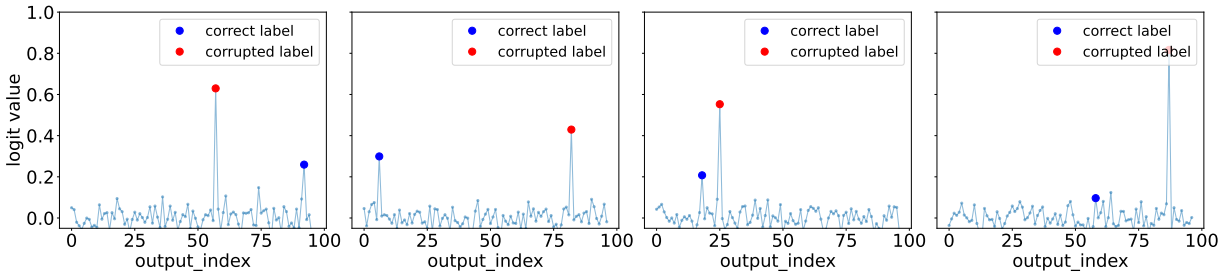
effectively degree 3 in weights, since the delta function term cancels with the network output. Consequently, around the generalizing solution, the weight gradients have a higher degree dependence on the weights themselves. This makes such solutions more prone to the collapse due to high weight decay. This serves as a heuristic justification as to why only grokked networks seemed to be prone to *Forgetting*, while memorizing networks seem to be immune.

If we use vanilla Gradient Descent optimizer, the updates would plateau once the balance between gradient updates and weight decay contribution is reached. But this is not the case in Adaptive optimizers. Indeed, we find that this phenomenon exclusively occurs with Adaptive optimizers (e.g. Adam in this case).

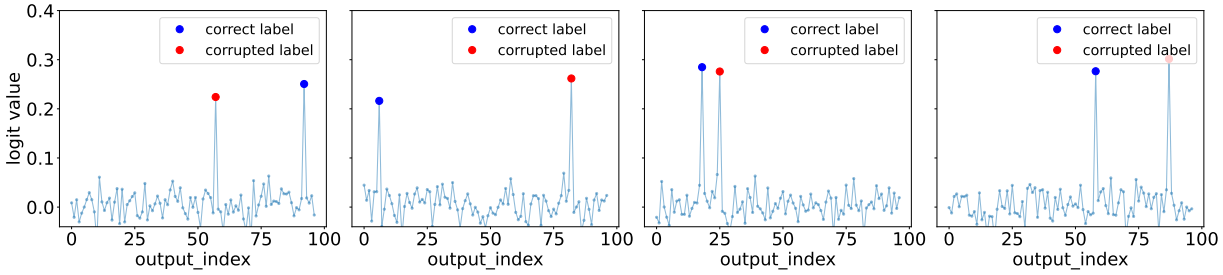
We believe that it is possible to get further insight into this phase by closely examining the optimization landscape and network dynamics. Since this behavior is exclusive to grokked networks, understanding this phase would lead to further understanding of grokking dynamics. Alternatively, a better understanding of the dynamics of grokking should give us insights into this curious phenomenon. We defer this analysis for future work and welcome an examination by our readers.

C.5 Output Logits

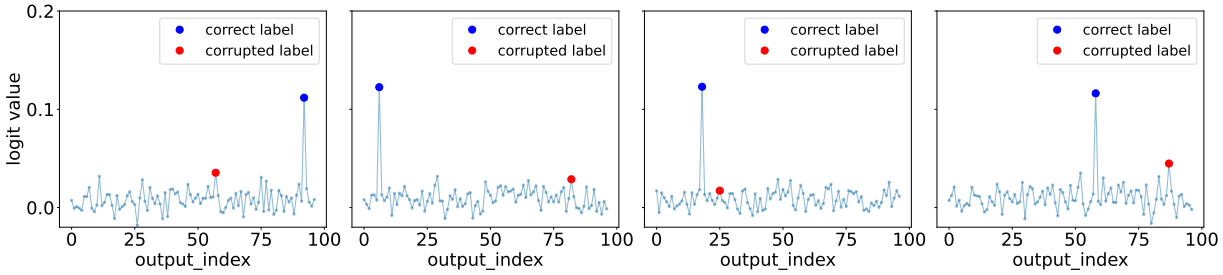
Here we show how the output logits of trained networks look like in various phases (i.e. different amount of weight decay) on corrupted training examples. In all cases, we see two peaks in logits, corresponding the the corrupted and the "correct" labels. Their relative value determines the degree of memorization, and hence, the phase.



(a) *Coexistence* phase (wd=0.0). The logits corresponding to corrupted label have a higher values than the correct ones, which explains the memorization of corrupted data in this phase.



(b) *Partial Inversion* phase (wd=5.0). The logits corresponding to corrupted labels have similar values to the correct ones, which explains partial memorization of corrupted data in this phase.



(c) *Full Inversion* phase (wd=15.0). The logits corresponding to corrupted labels are heavily suppressed compared to the correct ones, which explains the absence of memorization of corrupted data in this phase.

Figure C.2: Network output logits on 4 different (corrupted) training examples examples, in various phases ($N = 500, \alpha = 0.5, \xi = 0.35$). Blue dots correspond to the logit value for the "correct" labels. Red dots correspond to the logit value for the corrupted labels.

C.6 BatchNorm, LayerNorm and minibatch w/o Normalization

C.6.1 BatchNorm

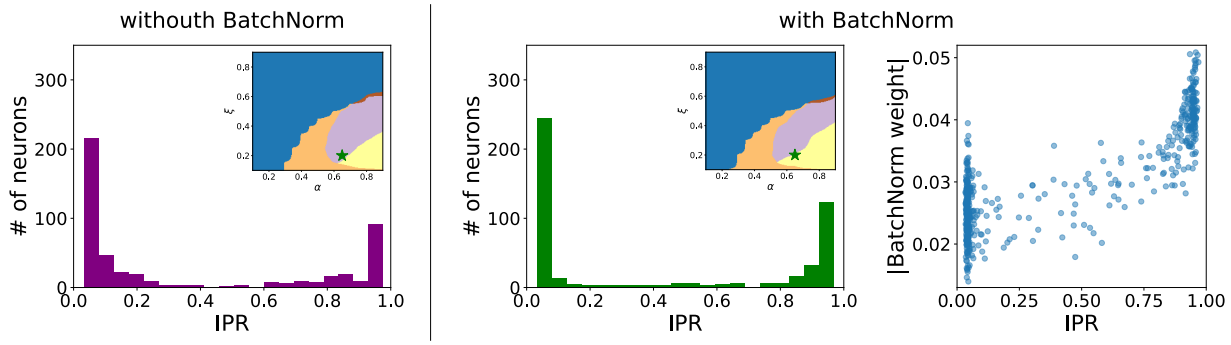


Figure C.3: IPR distributions of networks with/without BatchNorm; the correlation between BatchNorm weights and IPR. The IPR distributions in the two models do not exhibit significant differences. However, BatchNorm helps generalization by assigning higher weights (γ_k) to high IPR neurons compared to low IPR neurons. Both models are trained at data fraction $\alpha = 0.65$ and corruption fraction $\xi = 0.2$, with batch-size = 64.

BatchNorm also has a regularizing effect on training (Figure C.3), albeit, through a qualitatively distinct mechanism. In the network (Equation (4.2)), we apply BatchNorm *after* the activation function (post-BN):

$$(\mathbf{x}, \mathbf{y}) = \boldsymbol{W}\, \mathtt{BN}\left(\phi\left(\boldsymbol{U}\, \boldsymbol{e}_{\mathbf{x}} + \boldsymbol{V}\, \boldsymbol{e}_{\mathbf{y}}\right)\right) . \tag{C.2}$$

BatchNorm layer does does not affect the IPR distribution of the neurons significantly – the low IPR neurons persist even in the *Full Inversion* phase (Figure C.3). Instead, BatchNorm adjusts its own weights to de-amplify the low-IPR (memorizing) neurons, biasing the network towards generalizing representations. This can be seen from the strong correlation between IPR_k and the corresponding BatchNorm weights in Figure C.3(right). BatchNorm effectively prunes the network and it learns to do this directly from the data! Note that this effect of BatchNorm acts in

conjunction with the implicit regularization from the mini-batch noise in this case. We find that decreasing the batch-size enhances *Inversion* even without explicit regularization (Figure C.6).

C.6.2 BatchNorm, LayerNorm and Mini-Batch Noise

Here we examine the effects of BatchNorm, LayerNorm and mini-batch noise. We find that generalization and *Inversion* are indeed enhanced by decreasing the batch-size. This demonstrates that implicit regularization from mini-batch noise favours simpler solutions in our setup.

Additionally, upon comparing Figure C.4, Figure C.6 and Figure C.5 we find enhanced *Inversion* in cases with normalization layers (especially BatchNorm). For BatchNorm, we provide an explanation in the main text, with the the BatchNorm weights de-amplifying memorizing neurons in favour of generalizing ones. For LayerNorm, however, we do not find such an effect.

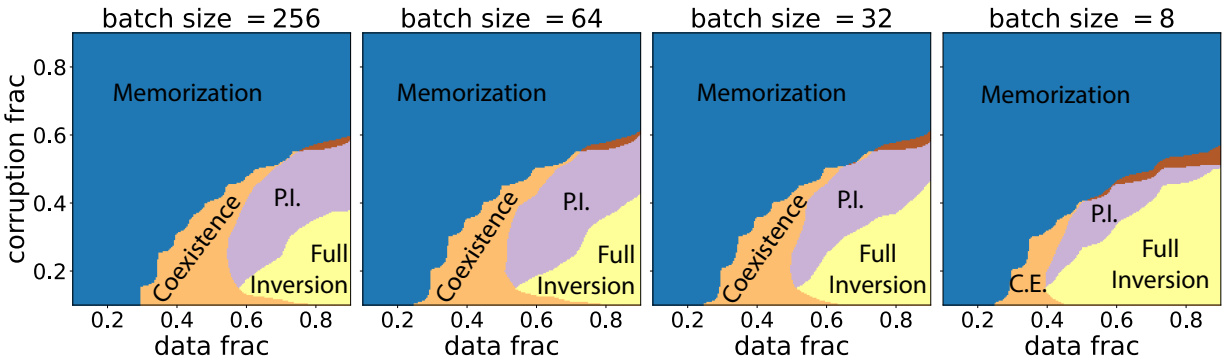


Figure C.4: Phase diagram with BatchNorm (post-BN)

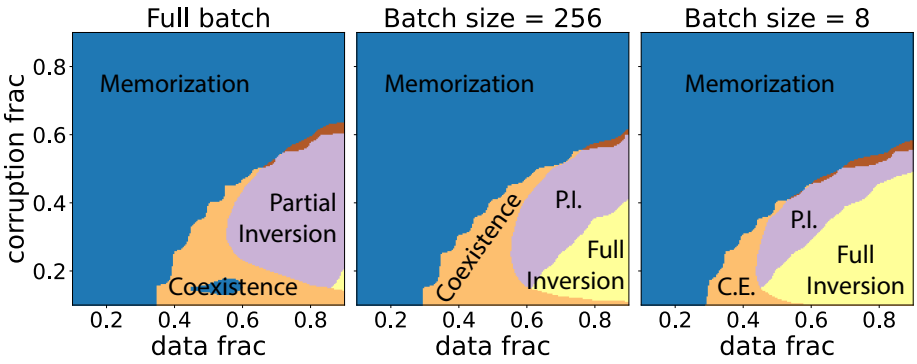


Figure C.5: Phase diagram with LayerNorm (post-LN)

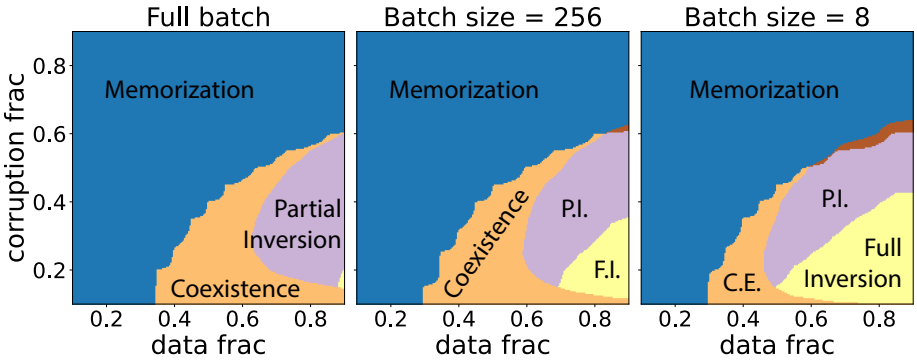


Figure C.6: Phase diagram with mini-batch training, without any explicit regularization or normalization layers.

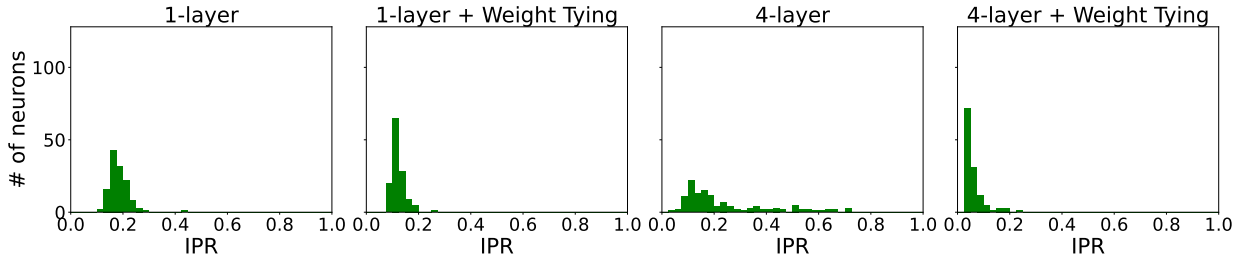
C.7 Transformers and Deep MLPs

In this section, we present (i) more weight decay experiments on different configurations of Transformers; (ii) the Effect of Dropout on Transformers; (iii) the Effect of BatchNorm on 3-layer ReLU MLPs.

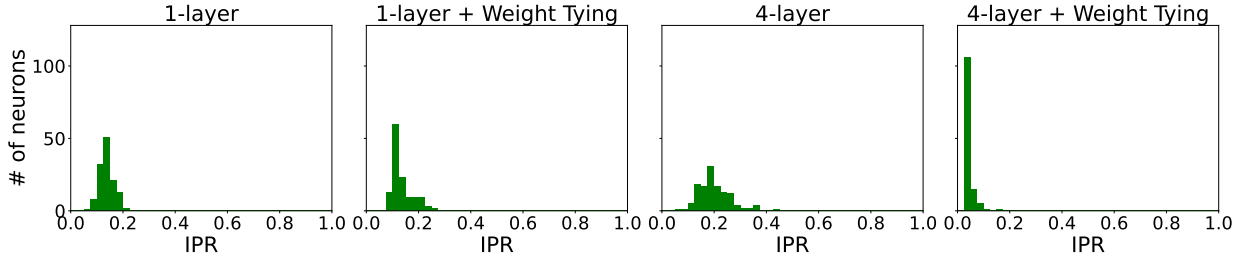
For reasons that we do not fully understand, 3-layer ReLU MLPs with even 0.05 Dropout probabilities failed to memorize the dataset for any data fraction $\alpha > 0.3$ and noise level $\xi \geq 0.1$, even after an extensive hyperparameter search. We leave the discussion about this for the future.

C.7.1 Effect of Depth / Number of Heads / Weight Tying

To further support our Hypothesis 4.2, we conducted further experiments by varying the configuration of Transformer models. Figure C.7 suggests that our Hypothesis holds irrespective of the Transformer configurations.



(a) Transformer with 8 heads and embedding dimension 128.

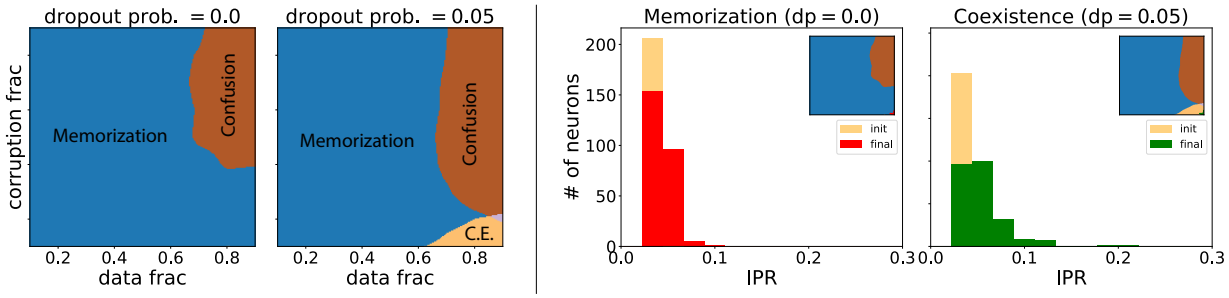


(b) Transformer with 4 heads and embedding dimension 128.

Figure C.7: The IPR histograms of the Embedding layer for Transformers with different configurations. We see that depth does not affect the IPR distribution, whereas more heads encourages high IPR embeddings and weight tying discourages it. All models are trained with corruption fraction $\alpha = 0.8$ and corruption fraction $\xi = 0.2$.

C.7.2 Dropout Phase Diagram for Transformers

We conduct further experiments to check the effect of Dropout on Transformers. From Figure C.8, we see that with a small dropout probability, the model managed to have a coexistence phase with an average higher IPR for neurons connected to the embedding layer. Notably, any dropout probability larger than 0.1 does not help.



(a) 1-layer Transformer.

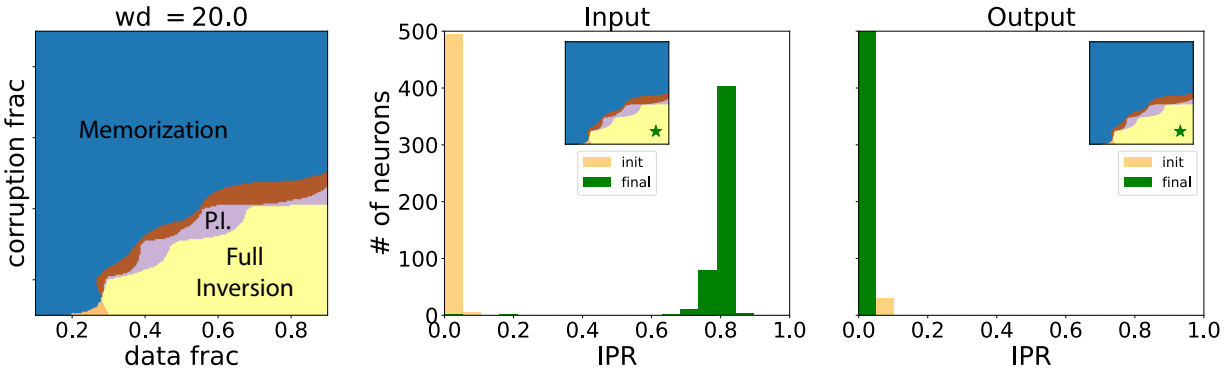
Figure C.8: Modular Addition phase diagrams and IPR histograms with different dropout probabilities (denoted by "dp"). The IPR histograms are plotted for neurons connected to the embedding layer. We select points trained with data fraction $\alpha = 0.9$ and corruption fraction $\xi = 0.1$.

C.7.3 Effect of Parameterization for 3-layer MLPs

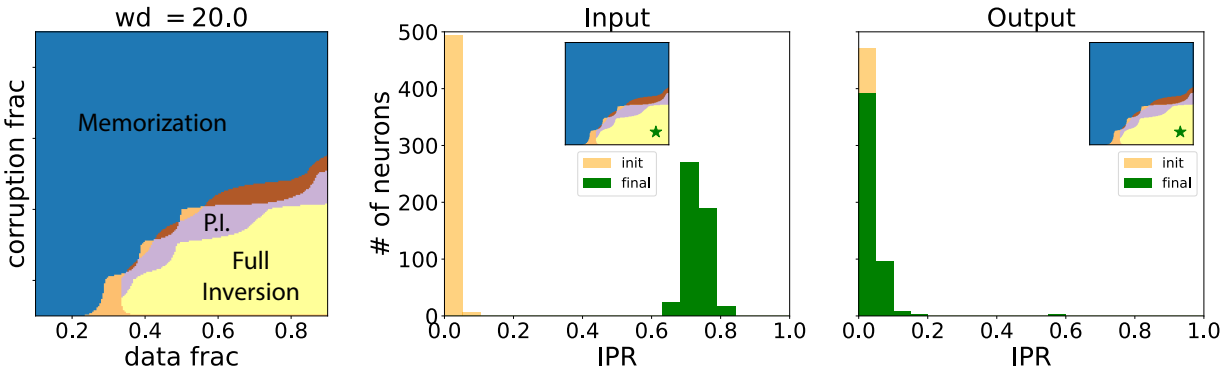
As we mentioned in Hypothesis 4.2, the input layer should have high IPR columns irrespective of the details of the network. In this section, we show that different parameterization of the network does not break our Hypothesis.

In particular, we consider two different parameterizations: (i) The standard parameterization with weights sampled from $\mathcal{N}(0, 2/\text{fan_in})$; (ii) The parameterization with weights sampled from $\mathcal{N}(0, 2/N^{2/3})$, which corresponds to a network parameterized in a more lazy training regime [188].

Our results in Figure C.9 show that as the network gets into a lazier regime, the row-wise IPR distribution of the input layer weight barely shifts, whereas the column-wise IPR distribution of the output layer weight has larger values. We interpret this shift as happening because the network with lazier parameterization fails to utilize the hidden layers to decode the periodic features embedded by input layers.



(a) MLPs in standard parameterization.

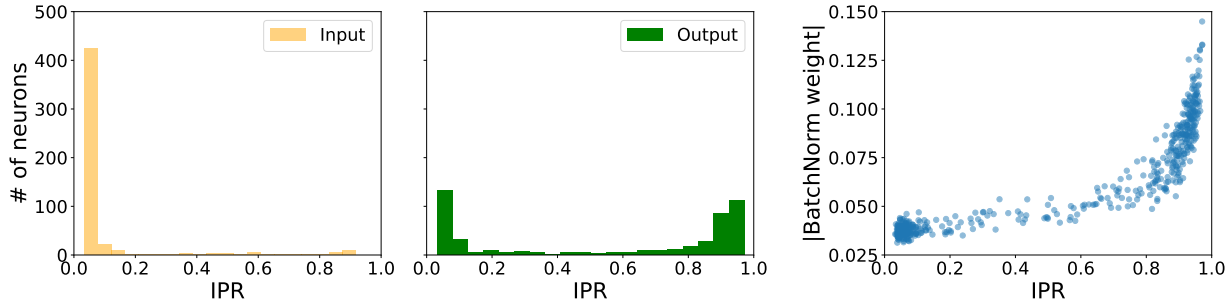


(b) MLPs in a lazier parameterization.

Figure C.9: Phase diagrams and the IPR histograms of the columns of input layer weight and rows of the output layer weights for 3-layer MLPs with different parameterizations. Selected models are trained with data fraction $\alpha = 0.8$ and corruption fraction $\xi = 0.2$.

C.7.4 BatchNorm for 3-Layer ReLU MLPs

This subsection tests the correlation between BatchNorm weights and high IPR neurons for 3-layer ReLU MLPs. We found that BatchNorm for 3-layer MLPs only has coexistence phases and can perfectly memorize and generalize for small corruption fractions ξ, we showed a representative training curve in Figure C.10. We find that with BatchNorm, the 3-layer MLP tends to have high IPR neurons close to the output layer instead of having high IPR for neurons close to the input layer. We believe that failure to remove the very low IPR neurons connected to the input layer is why this network is always in the coexistence phase.



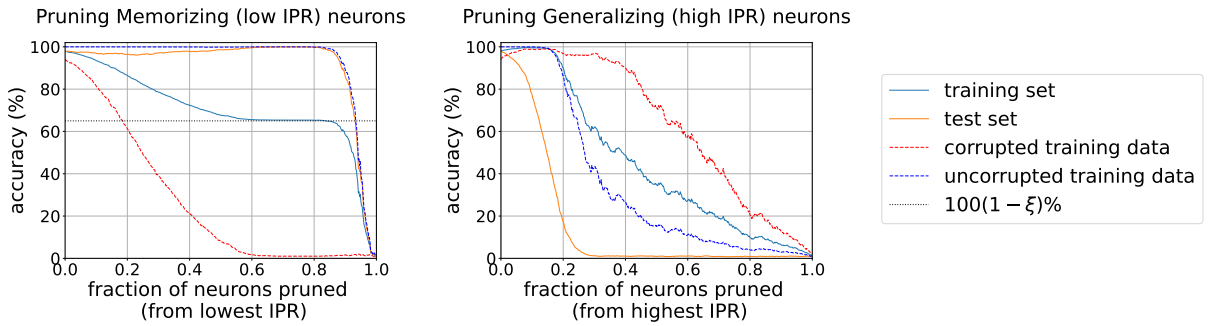
(a) 3-layer ReLU MLP, trained with batch size $B = 64$.

Figure C.10: IPR of a 3-layer MLP in coexistence phase, trained with data fraction $\alpha = 0.8$ and corruption fraction $\xi = 0.1$. We plotted row-wise IPR for input layer weight and column-wise IPR for output layer weight; the correlation between column-wise IPR of output layer weights with the absolute value of BatchNorm weights right before it. The correlation between the BatchNorm weights and IPR values we pointed out in the main text still holds. However, we did not observe a strong correlation between row-wise IPR of the input layer and the BatchNorm weight next to it.

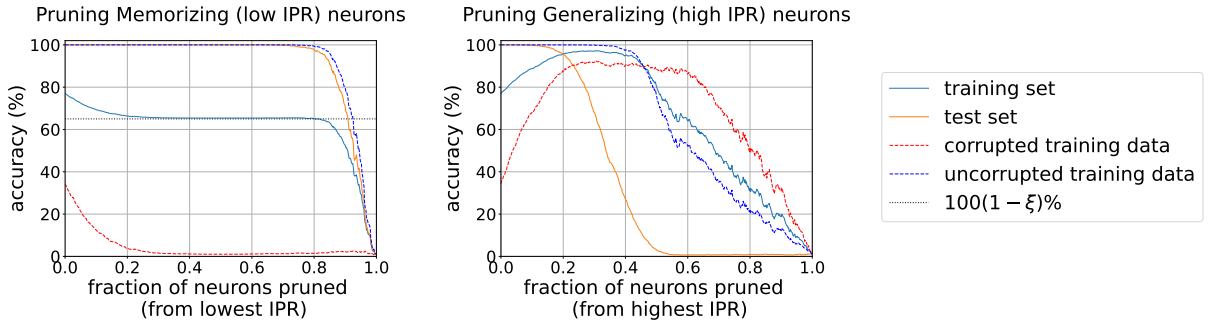
C.8 Additional Pruning Curves

C.8.1 Accuracy curves

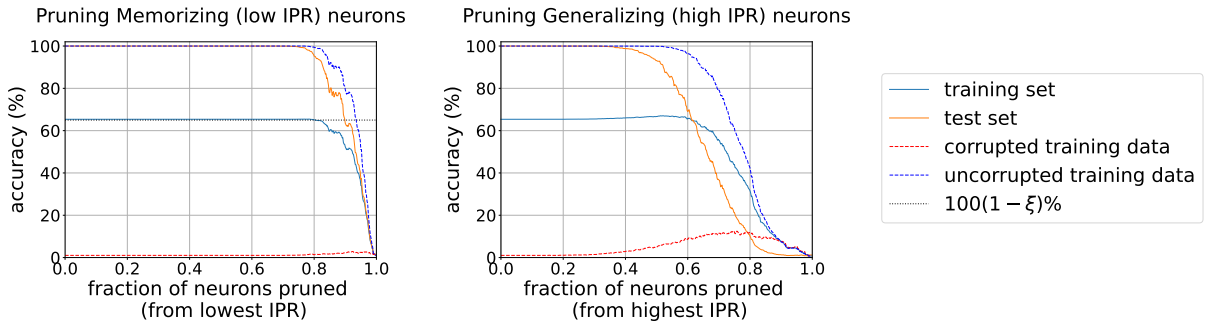
In Figure 4.7, we showed pruning low and high IPR neurons in the *Coexistence* phase. Here we show the extended version of the pruning experiments. In contrast to the plots in Figure 4.7, In Figure C.11 we plot longer x-axes – we show accuracies with fraction of pruned neurons ranging from 0.0 to 1.0. We also perform the pruning experiment for *Partial Inversion* and *Full Inversion* phases. Pruning affects generalization and memorization differently in different phases because the proportion of high and low IPR neurons in trained networks depends on the phase.



(a) *Coexistence* phase (weight decay = 0.0). (Left) Pruning from low IPR neurons retains performance on test data and uncorrupted training data; while decreases performance on corrupted training data down to to $100 * (1 - \xi)\%$. (Right) Pruning high IPR neurons decreases performance on test data; while retaining performance on training data. Note that the accuracy on corrupted data is retained longer than that on uncorrupted data.



(b) *Partial Inversion* phase (weight decay = 5.0). (Left) Pruning from low IPR neurons retains performance on test data and uncorrupted training data; while decreases performance on corrupted training data down to to $100 * (1 - \xi)\%$. (Right) Pruning high IPR neurons decreases performance on test data. It improves the performance on corrupted training data and retains it on uncorrupted training data. In the absence of periodic neurons, the (memorizing) low-IPR neurons have an increased influence on the network predictions, which leads to this increase in memorization.

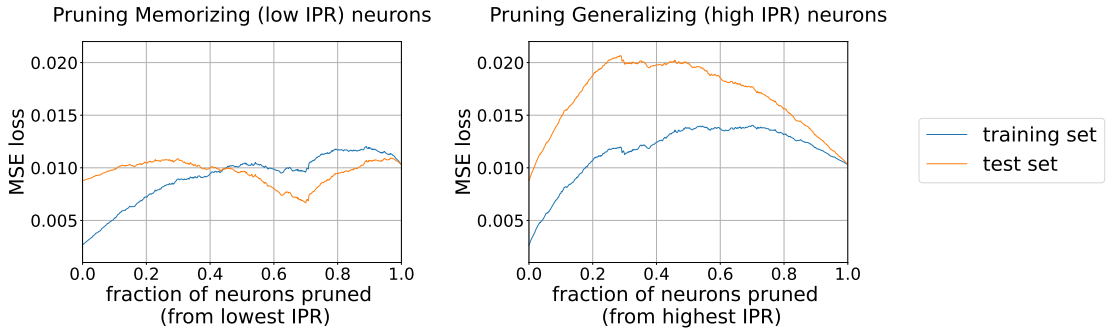


(c) *Full Inversion* phase (weight decay = 15.0). (Left) Pruning from low IPR neurons retains performance on test data and uncorrupted training data; while decreases performance on corrupted training data to $100 * (1 - \xi)\%$. (Right) Pruning high IPR neurons does not improve memorization significantly, except for the small increase in accuracy on corrupted training data towards the end. This is because the fraction of the neurons that have low IPR is very small in this case.

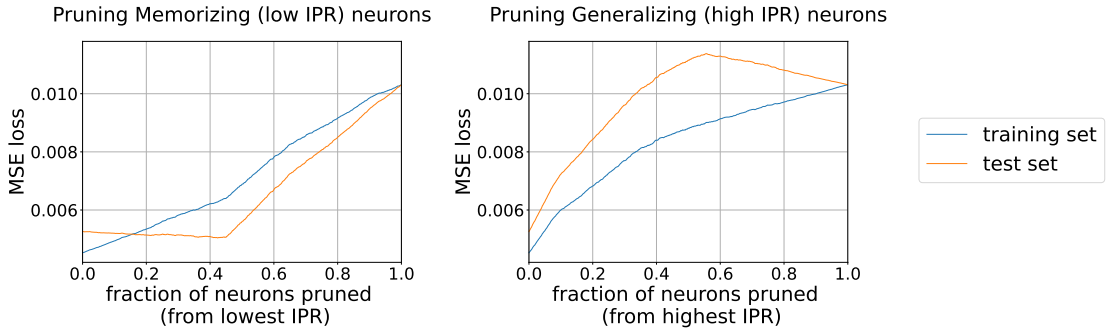
Figure C.11: Pruning experiments in all various phases ($N = 500, \alpha = 0.5, \xi = 0.35$). Pruning affects generalization and memorization differently in different phases because the proportion of high and low IPR neurons in trained networks depends on the phase.

C.8.2 Loss curves

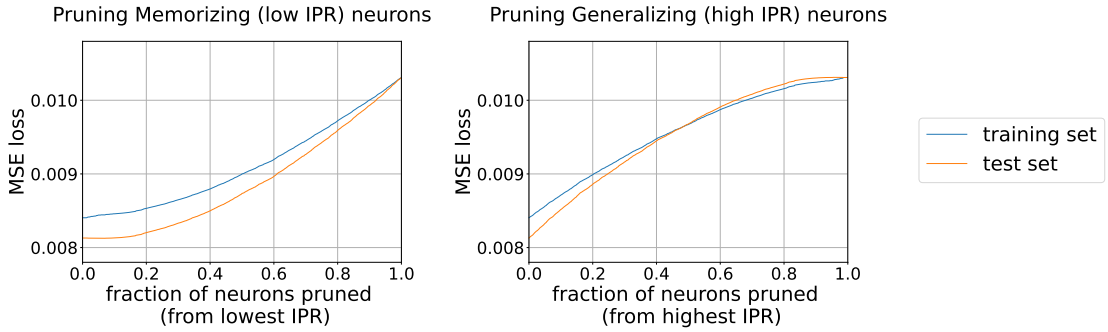
Here we show how the train and test losses are affected by pruning. Train loss is lowered by, both, memorizing (low IPR) *and* generalizing (high IPR) neurons. This leads to monotonic behaviors in all the cases. Test loss is lowered by periodic neurons, while increased by memorizing neurons. An effect that sits on top of these is the effective cancellation of sub-leading terms in the analytical solution Equation (4.4). Since the accuracies only depend on the largest logit, these sub-leading terms have little effect on them (at sufficient width). However, losses are significantly affected by them. With zero or small weight decay values, this cancellation is achieved by low IPR neurons. Thus, in these cases, low IPR neurons serve the dual purpose of memorization as well as canceling the sub-leading terms. With large weight decay values, this cancellation is achieved by a different algorithm, where "secondary peaks" appear in the Fourier spectrum of the periodic neurons. We leave an in-depth analysis of these algorithmic biases for future work.



(a) *Coexistence* phase (weight decay = 0.0). (Left) Pruning from low IPR neurons (almost) monotonically increases the train loss. The test loss has two competing factors: lower memorization and fewer neurons that cancel the sub-leading terms. As a result, we see non-monotonic behaviour in the first half. (Right) Pruning high IPR neurons increases both train and test losses. We see a non-monotonic behaviour in the second half since the cancellation of sub-leading terms is irrelevant in the absence of periodic neurons.



(b) *Partial Inversion* phase (weight decay = 5.0). (Left) Pruning from low IPR neurons: Initially, the removal of memorizing neurons causes the train loss to increase and the test loss to decrease. In the latter half, once we start removing the periodic neurons, both train and test losses increase. (Right) Pruning from high IPR neurons: Initially, both train and test losses increase due to the removal of periodic neurons. In the latter half, the removal of the memorizing neurons causes the train and test losses to increase and decrease, respectively.



(c) *Full Inversion* phase (weight decay = 15.0). (Left) Pruning from low IPR neurons monotonically increases the train loss, since it is affected by removal of memorizing *and* periodic neurons. On the other hand, test loss briefly remains constant (during the removal of the low IPR neurons), followed by a monotonic increase. (Right) Removing high IPR neurons monotonically increases train and test losses.

Figure C.12: Pruning experiments in all various phases ($N = 500, \alpha = 0.5, \xi = 0.35$). Pruning affects generalization and memorization differently in different phases because the proportion of high and low IPR neurons in trained networks depends on the phase.

C.9 Additional Training Curves

In this section, we show training curves for various different settings we have used in this appendix and Chapter 4.

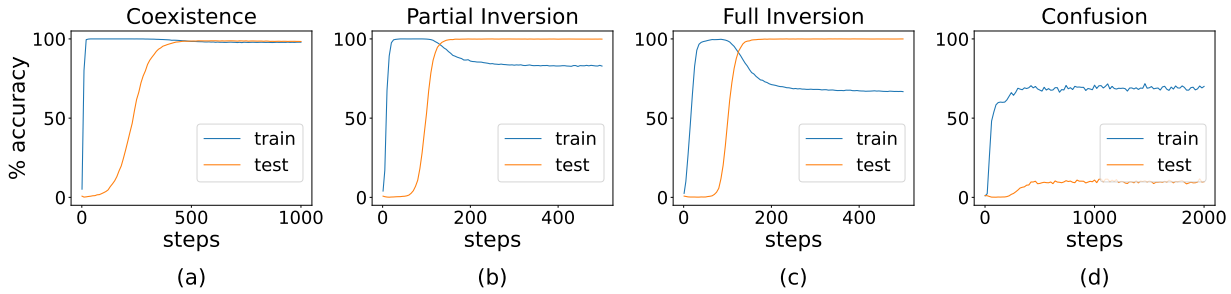


Figure C.13: Training curves with dropout (a) dp=0.0 (b) dp=0.3 (c) dp=0.6 (d) dp=0.9

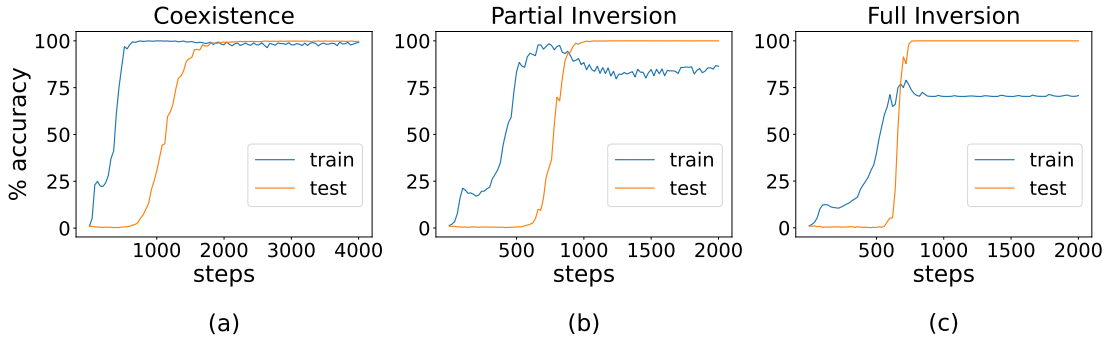


Figure C.14: Training curves in various phases with BatchNorm. All networks are trained with corruption fraction $\xi = 0.3$ and batch-size = 64; with variable data fractions. (a) Coexistence ($\alpha = 0.4$) (b) Partial Inversion ($\alpha = 0.5$) (c) Full Inversion ($\alpha = 0.8$)

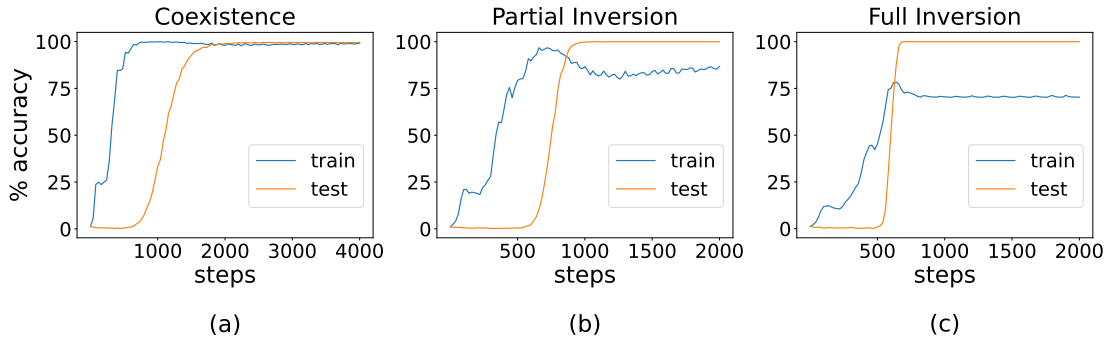


Figure C.15: Training curves in various phases with LayerNorm. All networks are trained with corruption fraction $\xi = 0.3$ and batch-size = 64; with variable data fractions. (a) Coexistence ($\alpha = 0.4$) (b) Partial Inversion ($\alpha = 0.5$) (c) Full Inversion ($\alpha = 0.8$)

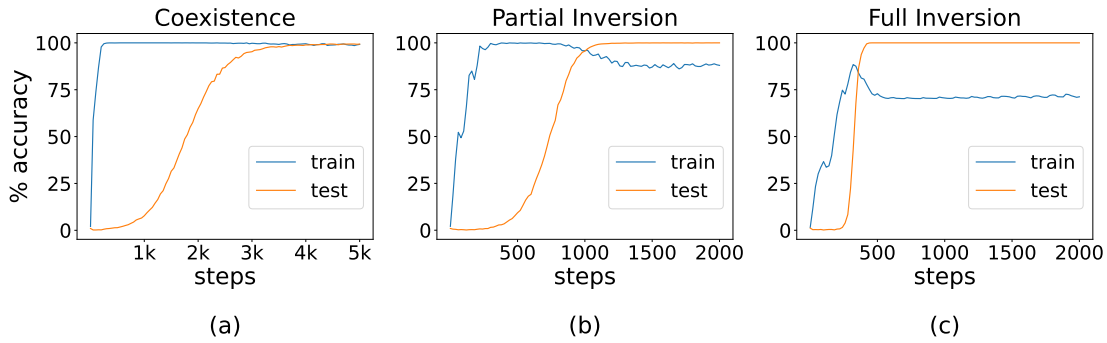


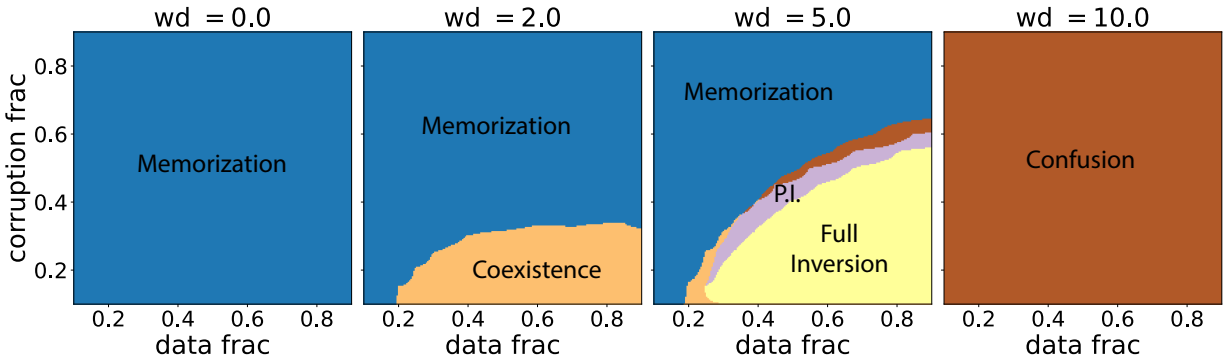
Figure C.16: Training curves in various phases without any normalization, but with mini-batches. All networks are trained with corruption fraction $\xi = 0.3$ and batch-size = 64; with variable data fractions. (a) Coexistence ($\alpha = 0.4$) (b) Partial Inversion ($\alpha = 0.5$) (c) Full Inversion ($\alpha = 0.8$)

C.10 Further Ablation Studies

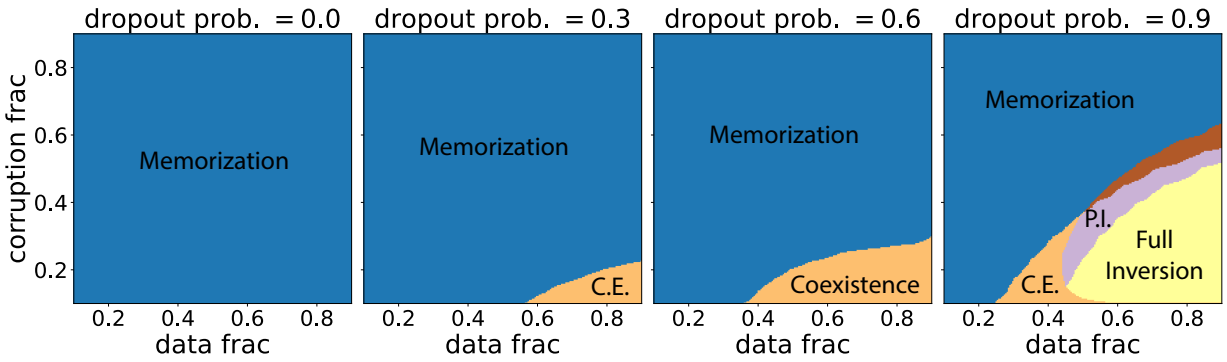
In this section, we perform ablation studies on different loss functions, activation functions, and network widths, and examine whether our conclusions remain universal across other modular arithmetic datasets.

C.10.1 Choice of Loss Function

CrossEntropy loss is known to be sensitive to label noise [110]. By adding regularization, we recover similar phase diagrams. The requirement for regularization is much greater for CrossEntropy loss compared to MSE. We find that CrossEntropy training is very sensitive to weight decay, with *Inversion* occuring for a narrow band of weight decay values, beyond which we find *Confusion*. We do not observe a *Forgetting* phase in this case. See Figure C.17. With Dropout, we find that CrossEntropy training requires very high dropout probabilities to grok and/or attain *Inversion*.



(a) Weight decay



(b) Dropout

Figure C.17: Modular Addition trained with CrossEntropy Loss. Robustness to label corruption is weaker compared to MSE.

C.10.2 Choice of Activation Function

To check if the story depends on the activation function, we did the modular addition experiments for 2-layer ReLU network, see Figures C.18 and C.19. The requirement for regularization for grokking is higher compared to quadratic activation function. We noticed that the phase diagram for weight decay behaves similarly to quadratic activation functions, except for the disappearance of the *Forgetting* phase. Dropout does not seem as effective for ReLU networks, with the network entering *Confusion* phase for even moderate dropout probabilites.

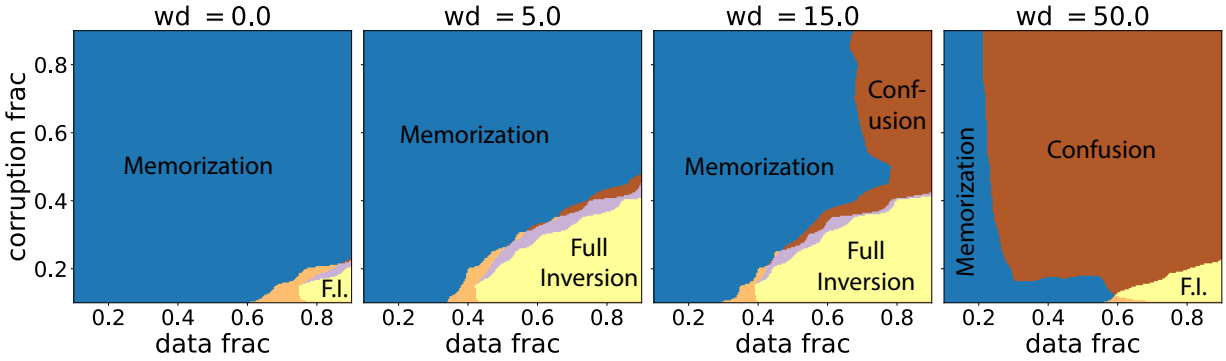


Figure C.18: Phase diagram with weight decay, ReLU.

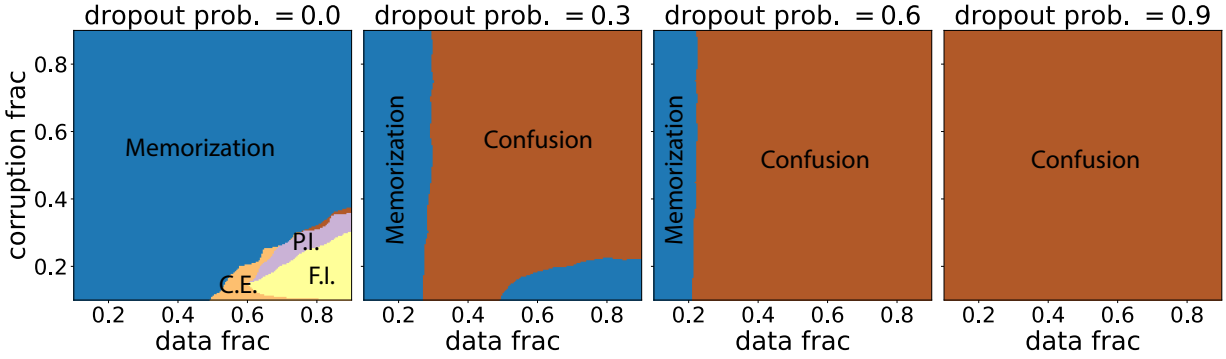


Figure C.19: Phase diagram with Dropout, ReLU.

C.10.3 Effect of of width

Width affects models' capacity, which further interact with regularization methods. In this subsection, we aim to understand the effect of different network widths. See Figures C.20 and C.21 for quadratic activation and ReLU activation results.

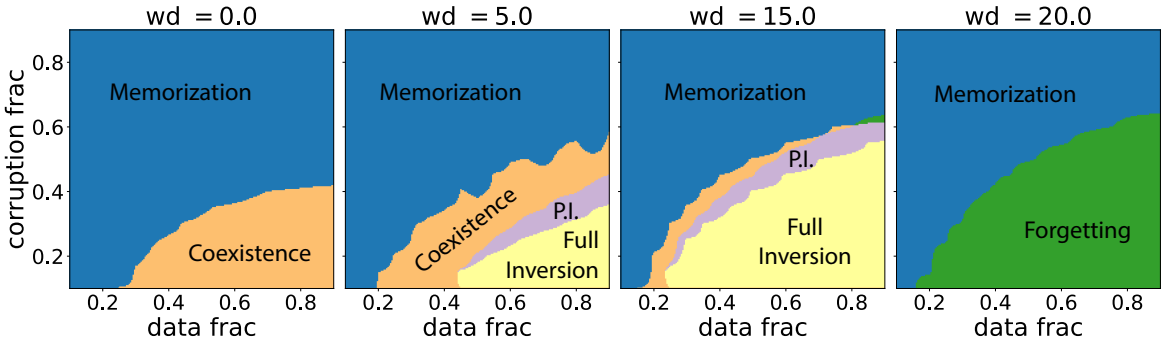


Figure C.20: Phase diagram for width $N = 5000$ fully connected network with quadratic activation. Regularization with weight decay. Compared to $N = 500$ network in Figure 6.1(a), the network generalizes marginally poorly. Note: the absence of *Partial Inversion* phase and smaller *Coexistence* phase with $wd = 0.0$. This can be attributed to the increased capacity to memorize due to abundance of neurons.

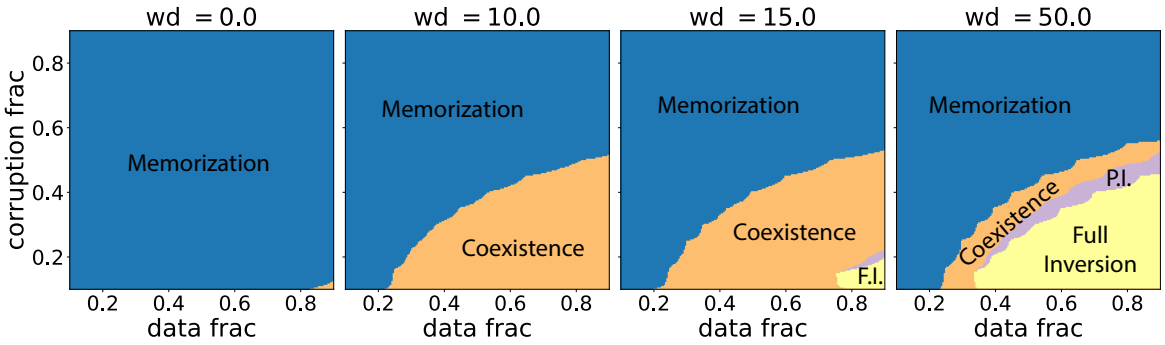


Figure C.21: Phase diagram for width $N = 5000$ fully connected network with ReLU activation. Regularization with weight decay. In this case, the network performance is drastically improved compared to $N = 500$ (Figure C.18). The confusion phase is largely eliminated, due to the increased capacity of the network, as a result of abundance of neurons.

C.10.4 Modular Multiplication

Here we check if the phase diagram story is unique to modular addition. We repeat our experiments with modular multiplication ($z = x \cdot y \mod p$) and obtain almost identical phase diagrams to modular addition (see Figure C.22).

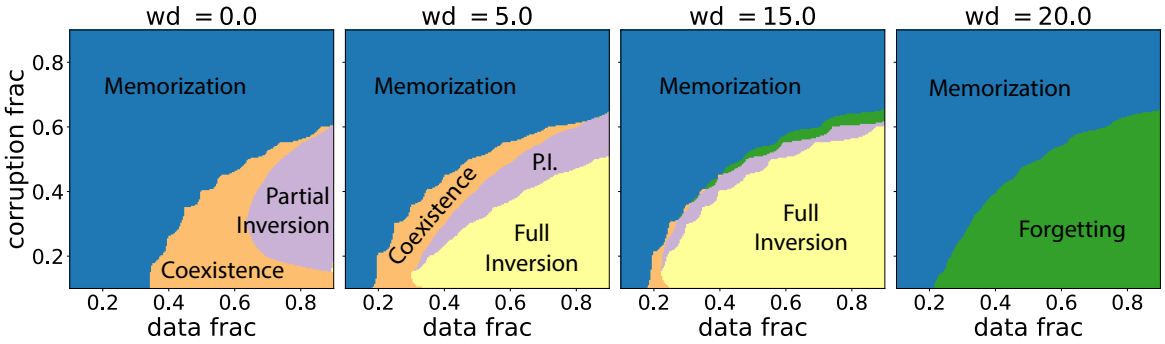
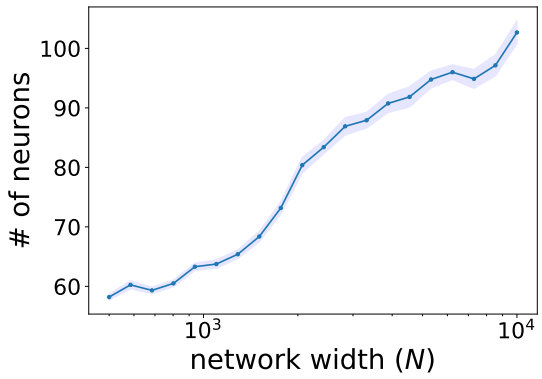


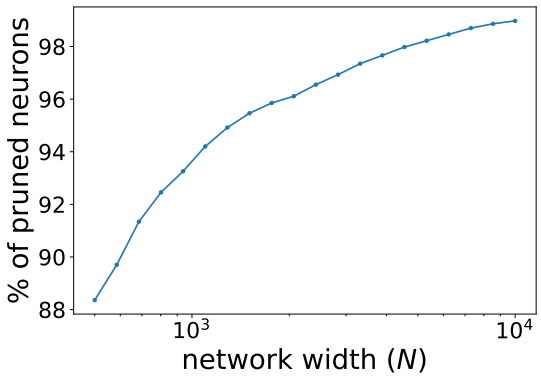
Figure C.22: Phase diagram for Modular Multiplication, weight decay. All other settings are identical to Figure 4.4. Note that the phase diagram is identical to Modular Arithmetic case – representative of the similarity in the algorithms learnt by the network.

C.11 How many neurons can be pruned while retaining generalization?

We showed in the pruning experiments (Figures 6, 17) that a significant proportion of neurons (starting from low IPR) can be pruned out without decreasing test error. In Figure C.23 we show how many neurons can pruned from a trained network at various widths. We train networks with widths between 500 and 10000; with learning rate 0.005 and weight decay 1.0, with AdamW optimizer and MSE loss. We use corrupted modular addition data, with data fraction $\alpha = 0.5$ and corruption fraction $\xi = 0.35$. We repeat the experiment 40 times, averaging over different random seeds.



(a) Number of neurons required after pruning (from low IPR) to retain generalization. We see a sub-linear increase in the required neurons with width. shaded region denoted the standard error over 40 runs.



(b) % of neurons that can be pruned (from low IPR) while retaining generalization.

Figure C.23: Pruning as many low IPR neurons as possible while retaining generalization, for various network widths ($\alpha = 0.5, \xi = 0.35$). The curves are averaged over 40 random seeds. Note that the x-axes on both plots have a logarithmic scale.

Appendix D: In-Context Learning and Skill Composition

D.1 Experimental Details

D.1.1 Model and Training Hyperparameters

Model Architecture. We used GPT-like architectures with Rotary Positional Embedding ($\theta = 10,000$) and ReLU activations. We fix the number of heads $H = 4$, embedding dimension $d_{\text{embed}} = 512$ and MLP widening factor 4 throughout every model. We use $d = 2, 4, 6$ throughout the paper. Embedding layers and the output layer are tied via weight tying.

Initialization. All linear layers and embedding layer weights are sampled from Gaussian distribution $\mathcal{N}(0, 0.02^2)$ at initialization, with the exception that the last linear layer in each MLP is sampled from $\mathcal{N}(0, 0.02^2/2d)$. No bias is used in any layer.

Optimization and Schedule. We trained most models using AdamW optimizer with learning rate $\eta = 1.5 \times 10^{-4}$, weight decay $\lambda = 2.0$, $\beta_1 = 0.9$, $\beta_2 = 0.98$, $\epsilon = 10^{-8}$, batch size $B = 1024$, $n_{\text{ct}} = 32$ in-context examples for 200k steps, together with a 5% linear warmup starting from 0.01η and a cosine annealing towards the end to 0.1η. Weight decay is not applied to LayerNorm layers.

Hyperparameter Choice. For $d = 2$ models we scanned learning rates $\eta \in \{7.5 \times 10^{-5}, 1.5 \times 10^{-4}, 3 \times 10^{-4}, 6 \times 10^{-4}\}$ and weight decay values $\lambda \in \{0.5, 1.0, 2.0, 5.0\}$. Then we transfer our hyperparameters to other depths. Benefiting from the extra scaling in the initialization of the last linear in MLP, we find that the hyperparameters perform well for other depths. For larger p values, we lowering down the learning rate to 10^{-4}.

D.1.2 Additional Details of Each Plot in the Main Text

Figure 5.2. (a) Phase diagram constructed using $d = 6$ data shown in Figure D.17, threshold for defining each phase is set to 75% for the corresponding set; (b) accuracy - number of shots curve for $d = 4$, $n_{\text{i.d.}} = 2^8$, $p = 29$ and $\alpha = 0.7$. (c, d) $d = 2$ model with $p = 29$, $n_{\text{i.d.}} = 2^9$ and $\alpha = 0.6$.

Figures 5.5 and 5.6. (a) Selected the best out of three random seeds with early stopping. (b, c) averaged over three random seeds with standard error labeled. All o.o.d. data are measured every $1,000$ step for 16 randomly sampled sequences along the pre-training. (d, e) Used checkpoint at the end of pre-training, averaged over 128 random sequences sampled from $S_{\text{test}}^{\text{o.o.d.}}$.

Figure 5.7. (a, b) Both phase diagrams are the best selected from three random seeds with early stopping. The loss - number of shots curves are plotted with $\alpha = 0.6$.

Figure 5.8. We use a $d = 2$ model trained with $\alpha = 0.6$ and $n_{\text{i.d.}} = 2^9$, and a $d = 4$ model trained with $\alpha = 0.8$ and $n_{\text{i.d.}} = 2^9$ with batch size $B = 1536$.

Figure 5.9. Attention heads extracted form $d = 2$ model trained with $n_{\text{i.d.}} = 2^9$ and $\alpha = 0.6$. (c, d) Both models are trained with $\alpha = 0.6$.

D.2 Details of resources used

To generate each phase diagram and training curve, we used 90 GPU days on NVIDIA A100 40GB, with automatic mixed precision (BFloat16) and Flash Attention implemented in the PyTorch library[69]. During the exploring stage, we also used around another 90 GPU days. Most inferences were running on 1/7-NVIDIA A100 40GB GPU, of which the cost is negligible compared to the pre-training cost.

D.3 Table of log map for $p = 29$

n	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$\log_{27}(n)$	0[1]	28	15	19	2	22	6	12	17	10	9	11	21	18	27

n	15	16	17	18	19	20	21	22	23	24	25	26	27	28
$\log_{27}(n)$	13	4	7	25	23	24	3	26	20	8	16	5	1	14

D.4 Task Selection and Sequence Design

Task Selection. During the initial exploration phase, we observed that it was challenging for the model to learn multiple modular arithmetic tasks simultaneously. Typically, the loss would

stagnate at a plateau indefinitely.

We hypothesize that when the model is trained on multiple modular arithmetic tasks, the strong bias inherent in each individual task may interfere significantly. When tasks are selected randomly, the resultant noise appears to inhibit the learning capabilities of the model, preventing it from acquiring any meaningful patterns or rules from the data.

Ultimately, we adopted a more structured approach to task selection as illustrated in Figure Figure 5.4, based on the following rationale: If the model is to learn a specific task vector (a^t, b^t), it would presumably be more straightforward if one of the two components – either a^t or b^t – has already learned by the model. This would leave the model with only the other component to decipher, which we assume is a comparatively simpler task. Thus, we decided to employ the rectangular rule for sampling task vectors, which we believe strategically reduces the complexity of the learning process by partially leveraging previously acquired knowledge.

In Figure D.1, we show an ablation plot of phase diagrams with a randomly selected task vector collection for pre-training. We still use the special sequence design.

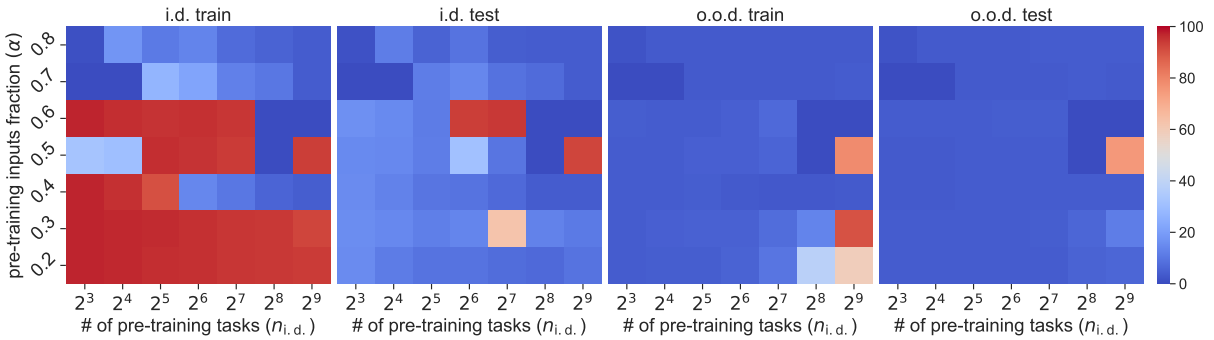


Figure D.1: $d = 6$ phase diagram with task vectors randomly selected, selected the best results from two random seeds. Learning rate $\eta = 10^{-4}$.

Sequence Design. Following a similar spirit, we use a balanced batch where sequences generated from all task vectors appear exactly the same number of times during the training. We

further align the examples across sequences generated by different task vectors, which we believe reduces the chance of the model getting confused by the same input (x, y) appearing at different positions within the batch. Without this design, we could not make the model train.

Here, we also show the training curve per task in Figure D.2, trained with all of our tricks. We see that the model first learned very few tasks and then eventually found its way out. For training without the task selection and sequence design, the loss typically plateau around ~ 3.0.

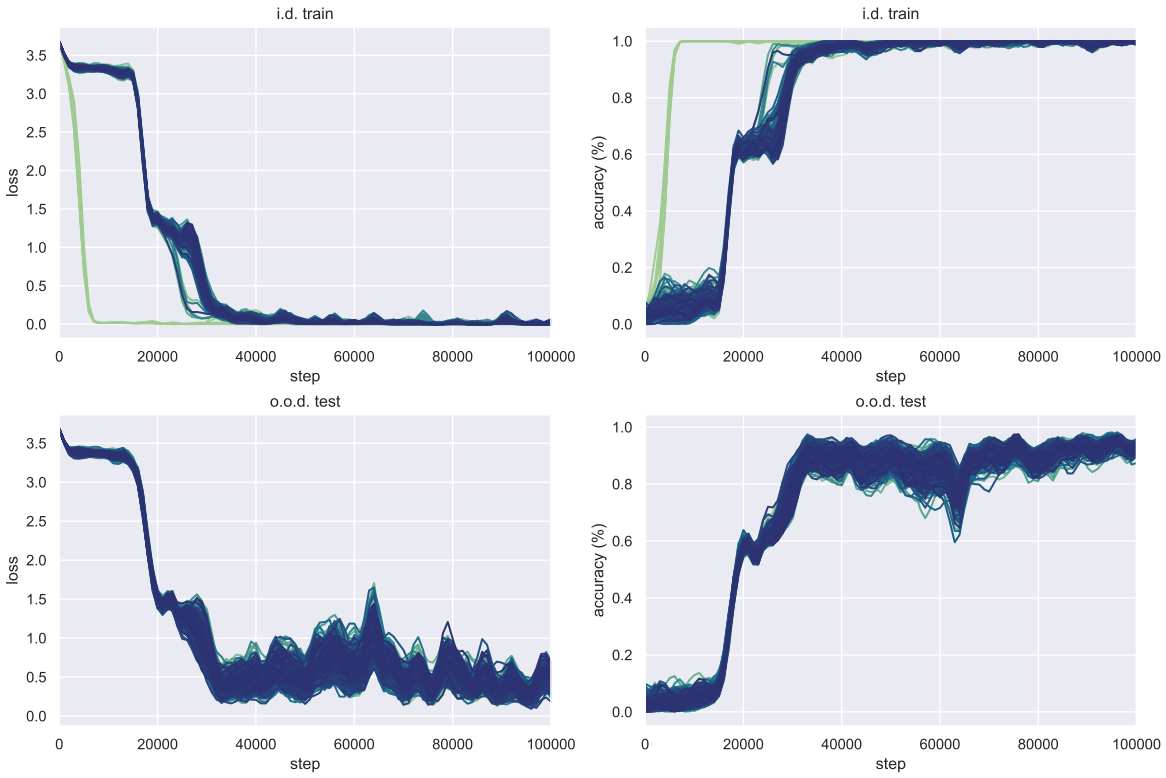


Figure D.2: $d = 4$ training curve, with loss and accuracy measured per task for the last in context example. Trained with $n_{\text{i.d.}} = 2^8$ and $\alpha = 0.8$. There are six tasks learned almost instantly: $(5, 26)$, $(21, 23)$, $(28, 27)$, $(3, 9)$, $(10, 10)$, $(5, 9)$.

D.5 Additional Interpretability Results

In this section, we show additional results on interpretability. Where we continue our discussion on how the algorithm might be implemented inside the model. Importantly, This

includes PCA analysis of embedding (Appendix D.5.1), other attention heads (Appendix D.5.2), attentions and MLPs outputs (Appendix D.5.3), and finally, the role of MLPs (Appendix D.5.4). We think it is also crucial to study LayerNorm layers, which we leave for future work.

D.5.1 PCA over Embeddings

We begin the discussion with the role of embedding layers. By further examining different models, we find highly structured embedding from $d = 2$ models. $d = 4$ models, on the other hand, does not have such structured embedding layers.

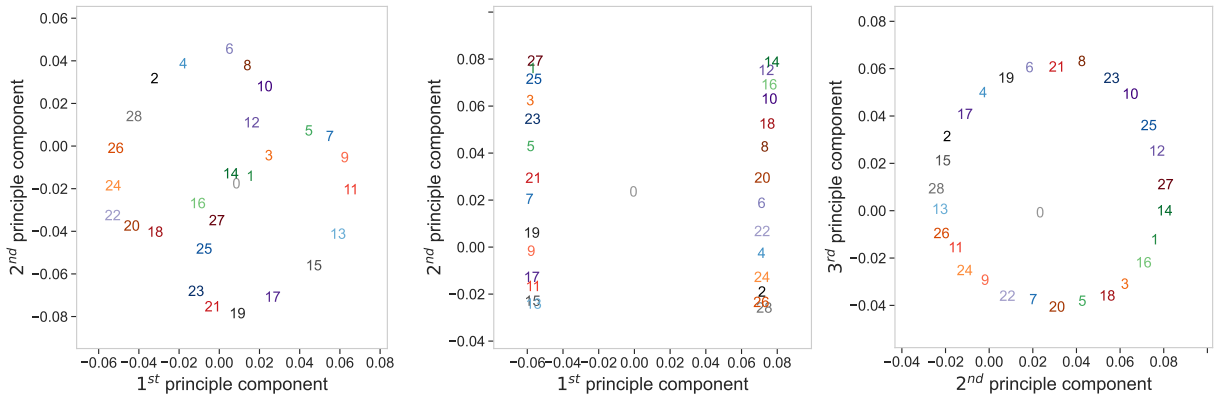
First we focus the embedding of $d = 2$ models. As shown in Figure D.3, clearly, the logarithm of each number is split into even and odd groups, and each group forms one circle, which is a suitable embedding for doing modular multiplications. However, one should note that this will not obviate the importance of the head shown in Figure 5.9(a), as the model still needs a way to distinguish the same number that appears in the different positions in the context.

Curiously, we could not find such a structured embedding for the $d = 4$ model, as shown in Figure D.4. However, as we will show in the next subsection, this non-structural embedding, together with the first layer, prepares a foundation for the essential heads in the latter layer to create a similar "circle-of-circles" feature as we shown in Figure 5.9(a, b).

D.5.2 Attention Heads

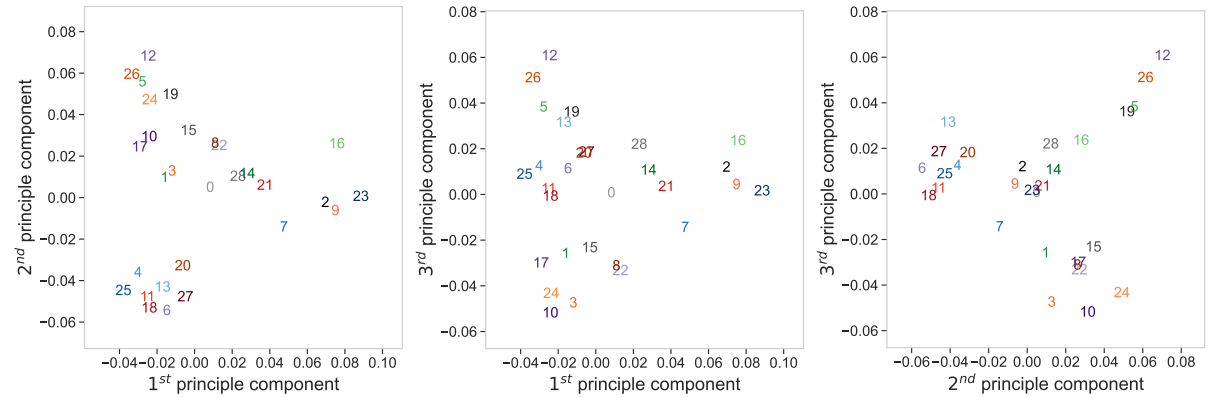
D.5.2.1 $d = 4$ Model

To continue the story, we analyse the attention heads in different models. We first study the $d = 4$ model, where we also find a similar head with "circle-of-circles" (Figure 5.9(a)).



(a) seed 2 (b) seed 1

Figure D.3: PCA over the embedding layers of $d = 2$ models with different random seeds. Each number is annotated by its $\log_{27}$ value.

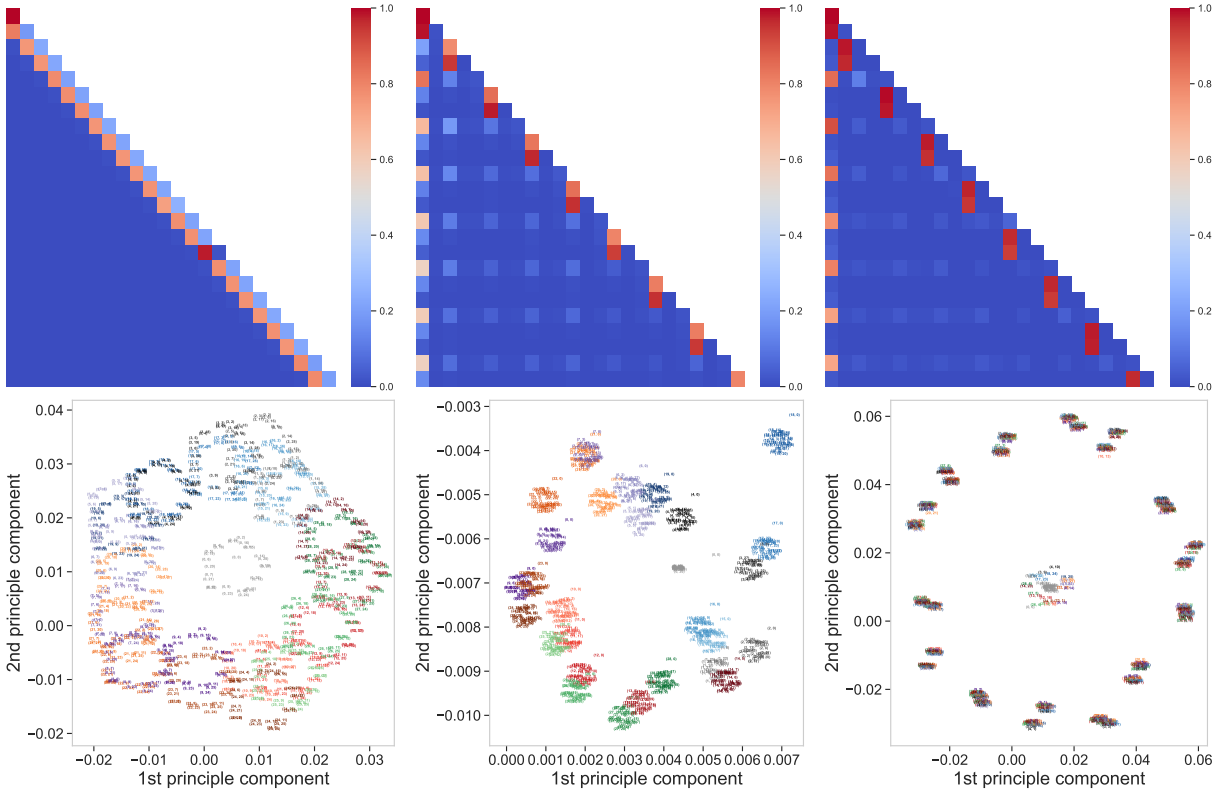


(a) PCA over the embedding layer for top-3 principal components

Figure D.4: PCA over the embedding layers of $d = 4$. Each number is annotated by its $\log_{27}$ value.

Moreover, two other heads put together are seemingly equivalent to Figure 5.9(b). We surmise that any model that solves modular arithmetic in-context requires such heads.

From Figure D.5(a), we see that the head still pays attention locally within three token positions. Importantly, it also creates a circle-of-circles while performing PCA over concatenated (x, y) features. The difference here is that the circle winds twice to go back to its origin. We believe that this factor of two differences in the period means that this head is effectively combing the row of the embedding layer of the $d = 2$ model and the role of the head in Figure 5.9(a).



(a) layer 2 head 4 (b) layer 3 head 1 (c) layer 3 head 4

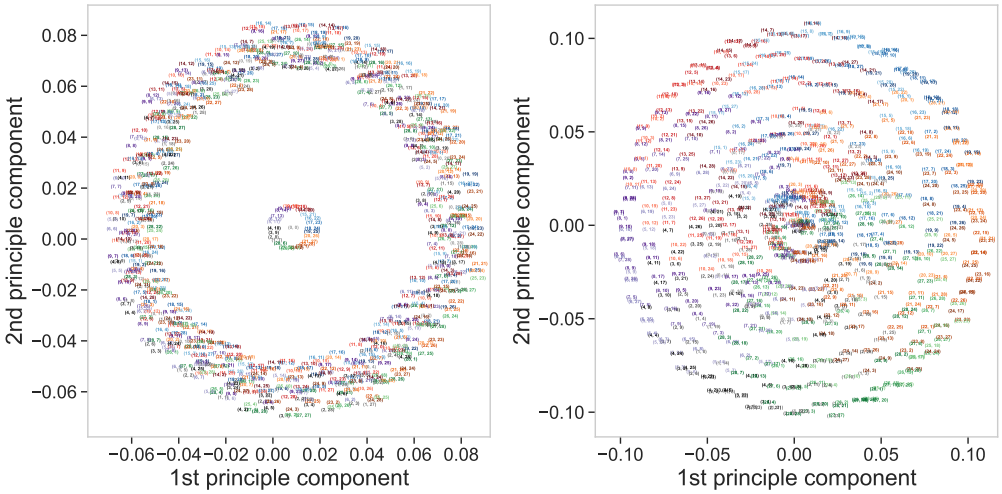
Figure D.5: PCA over the features of specific heads for $d = 4$ model. Each number is annotated by its $\log_{27}$ value. We plotted PCA over (a, b) (x, y) and (c) (y, z) concatenated features. Where the circle in (b, c) heavily depends on the sequence and (a) remains unchanged.

Overall, we do not yet know why the model needs to split the logarithm of numbers into even and odd groups.

From Figure D.5(b, c), we find two heads that are structured but different from Figure 5.9(b). The PCA pattern forms circles while annotated within logarithm space, but depends on the choice of the sequences. This hints that those two heads are trying to exchange information with the previous inputs. We leave PCA analysis across different in-context examples to future work.

D.5.2.2 $d = 2$ Model

Next we focus on the $d = 2$ model. In Figure D.6, we plot similar PCA to the one in Figure 5.9(a), but with different concatenated features from the same heads: (x, z) and (y, z). We again see circle-of-circles, solidifying our argument that this head provides proper spaces for later heads to perform modular operations.



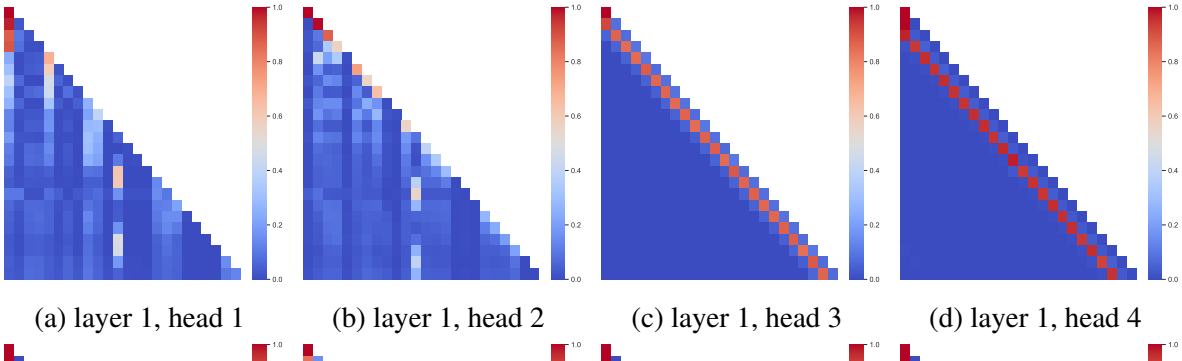
(a) (x, z) (b) (y, z)

Figure D.6: PCA over the same head in Figure 5.9(a), with feature concatenated differently. We annotate numbers with their corresponding logarithm.

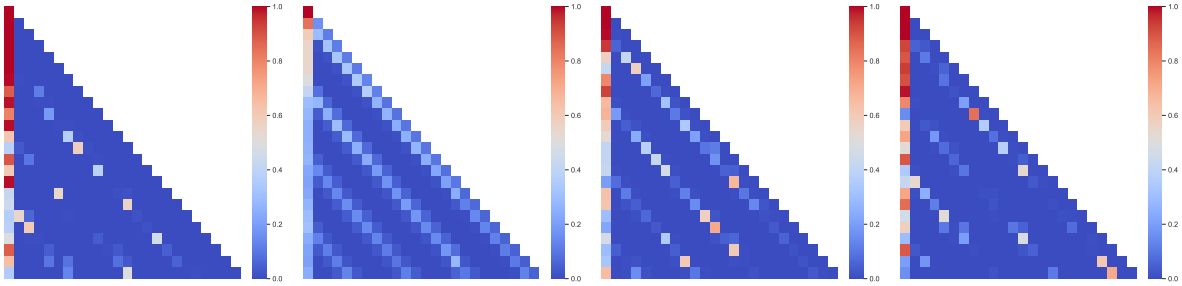
In Figure D.7, we dump all the attention patterns we had for $d = 2$ models. The corresopnding PCA over concatenated (x, y) features are shown in Figure D.8 (except for layer 1 head 4, for which we plot for (x, z)). The special head we choose here has a similar behavior to the one in Figure 5.9(a), with its main focus shifted by one position but still within two preceding tokens.

In Figure D.9, we extend our PCA analysis to different task vectors and up to top-4 components, where the spatial structure of the features is better depicted.

We have reasons to believe that the non-circle heads are not essential to the models' performance. However, we leave a careful exploration of this front for future work.

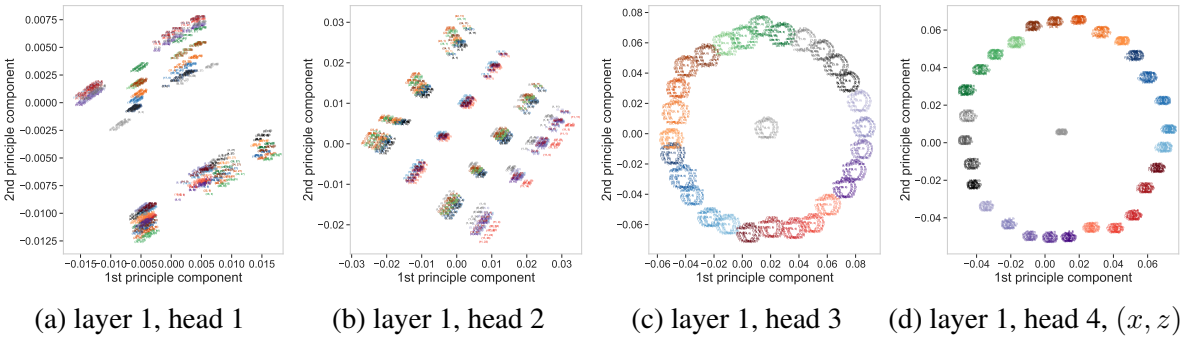


(a) layer 1, head 1 (b) layer 1, head 2 (c) layer 1, head 3 (d) layer 1, head 4

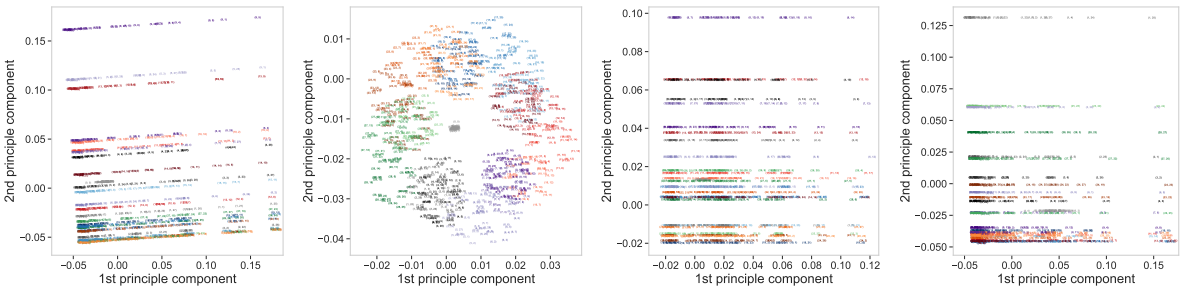


(e) layer 2, head 1 (f) layer 2, head 2 (g) layer 2, head 3 (h) layer 2, head 4

Figure D.7: All attention heads in $d = 2$ model.



(a) layer 1, head 1 (b) layer 1, head 2 (c) layer 1, head 3 (d) layer 1, head 4, (x, z)



(e) layer 2, head 1 (f) layer 2, head 2 (g) layer 2, head 3 (h) layer 2, head 4

Figure D.8: PCA for concatenated (x, y)/(x, z) features for all heads in $d = 2$ model.

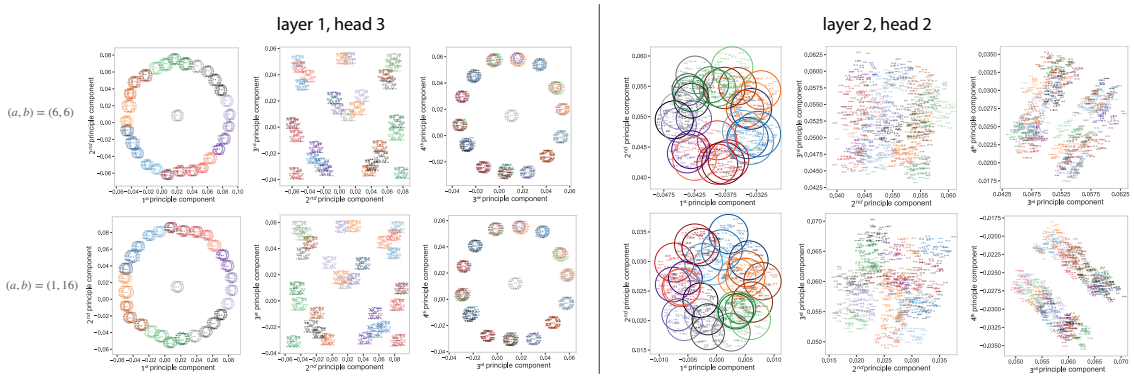


Figure D.9: More PCA analysis similar to Figure 5.9. Top-2 components of PCA contribute $\sim 40\%$ variance of the layer 1 head and $\sim 20\%$ variance of the layer 2 head, while the top-4 components contribute $\sim 60\%$ and $\sim 40\%$ accordingly.

D.5.3 PCA over Attention and MLP Outputs

In Figure D.10, we plot $d = 2$ PCA analysis up to top-4 components for attention and MLP outputs.

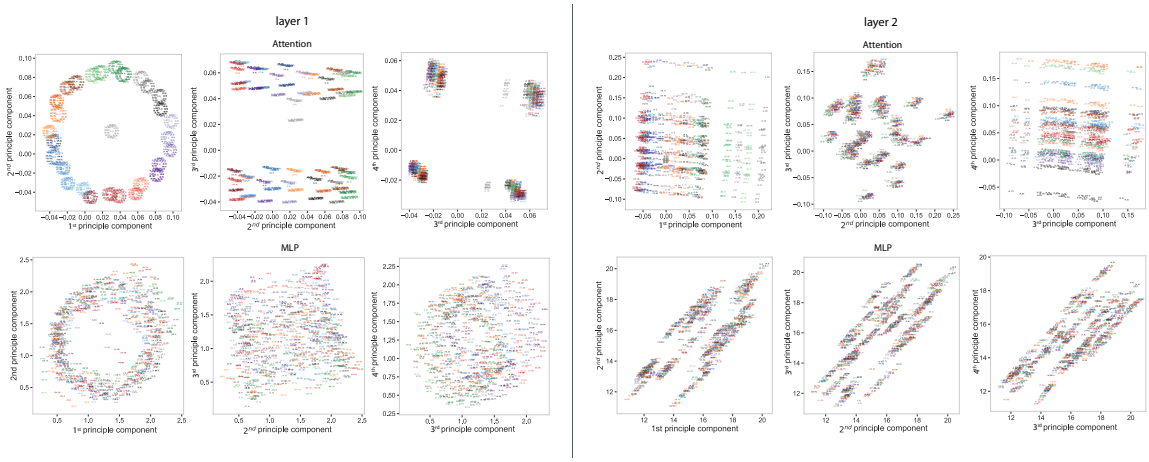


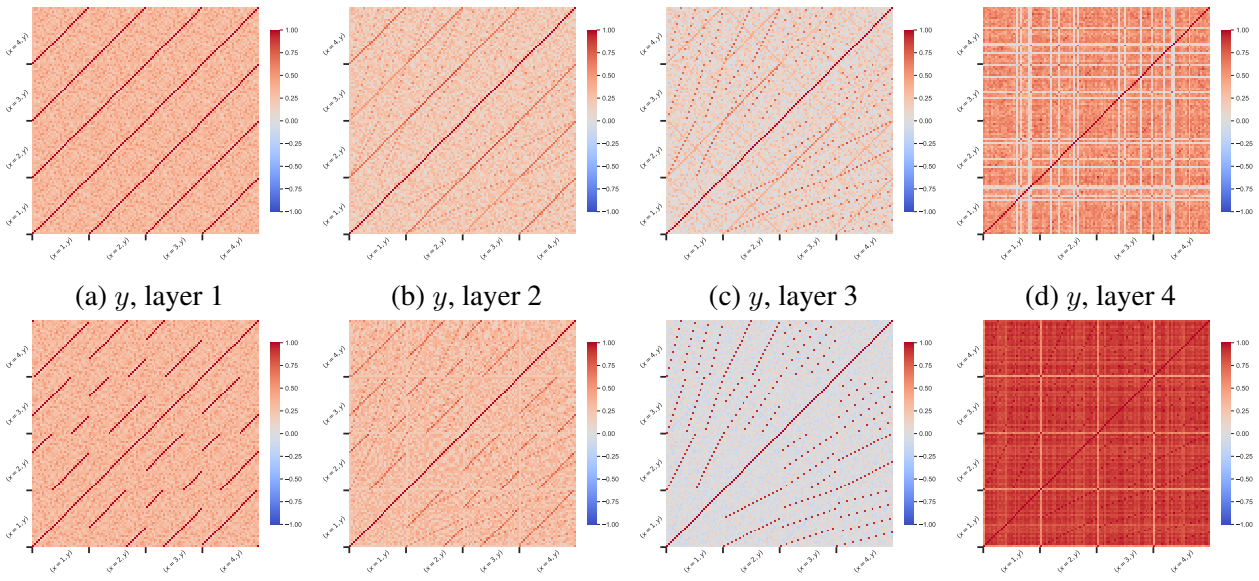
Figure D.10: PCA analysis of Attention and MLP outputs. Attention modules contribute $\sim 50\%$ and $\sim 30\%$ to top-4 components of PCA of layer 1 and layer 2. MLP modules contribute $\sim 10\%$ and $\sim 12\%$ accordingly.

D.5.4 Cosine-similarity analysis for MLPs

As we mentioned in Section 5.4.2, Skills II and III are likely implemented within the MLP layer. While we provided extensive details on how the signature of Skill II was observed in MLP, we could not find clear signals similar to those in Gromov [30], Nanda et al. [103] from MLP layers on Skill III.

D.5.4.1 $d = 4$ model

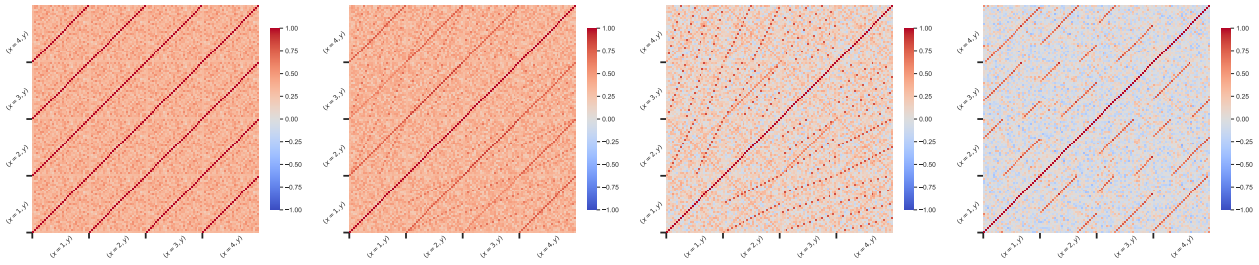
Here, we show an extended version of Figure 5.10 for all layers from the $d = 4$ model. We plot for 2-shot (Figure D.11) and 16-shot (Figure D.12), the task vector used is $(a, b) = (2, 6)$.



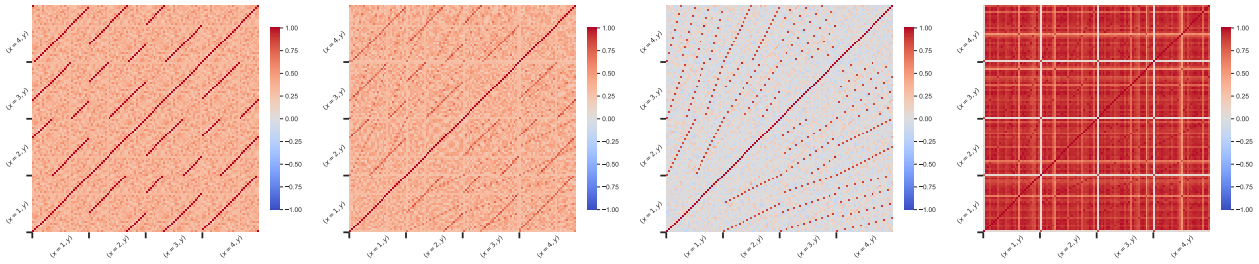
(a) y, layer 1 (b) y, layer 2 (c) y, layer 3 (d) y, layer 4

(e) z, layer 1 (f) z, layer 2 (g) z, layer 3 (h) z, layer 4

Figure D.11: Cosine-similarity of $d = 4$ model for every block, measured at 2-shot.



(a) y, layer 1 (b) y, layer 2 (c) y, layer 3 (d) y, layer 4

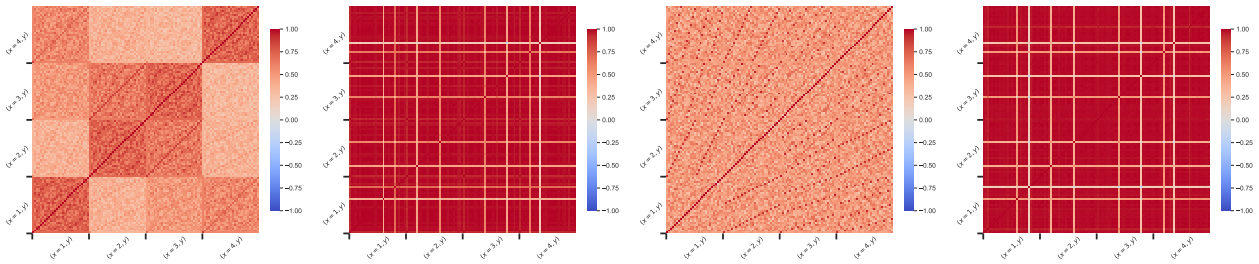


(e) z, layer 1 (f) z, layer 2 (g) z, layer 3 (h) z, layer 4

Figure D.12: Cosine-similarity of $d = 4$ model for every block, measured at 16-shot.

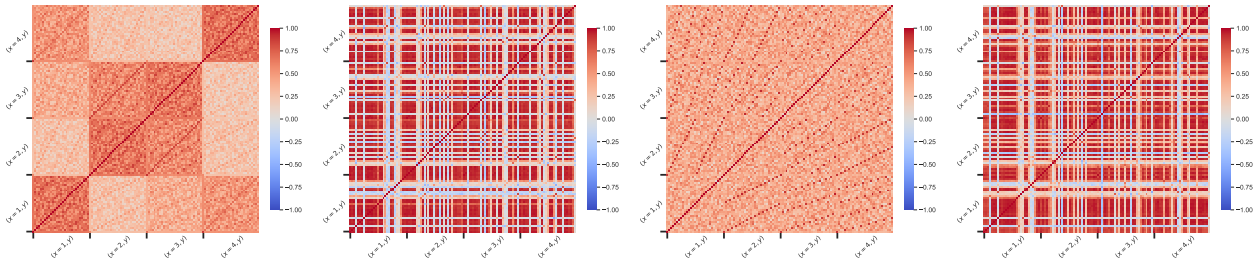
D.5.4.2 $d = 2$ model

Here, we show an extended version of Figure 5.10 for all layers from the $d = 2$ model. We plot for 2-shot (Figure D.13) and 10-shot (Figure D.14), the task vector used is $(a, b) = (2, 6)$.



(a) y, layer 1 (b) y, layer 2 (c) z, layer 1 (d) z, layer 2

Figure D.13: Cosine-similarity of $d = 2$ model for every block, measured at 2-shot.

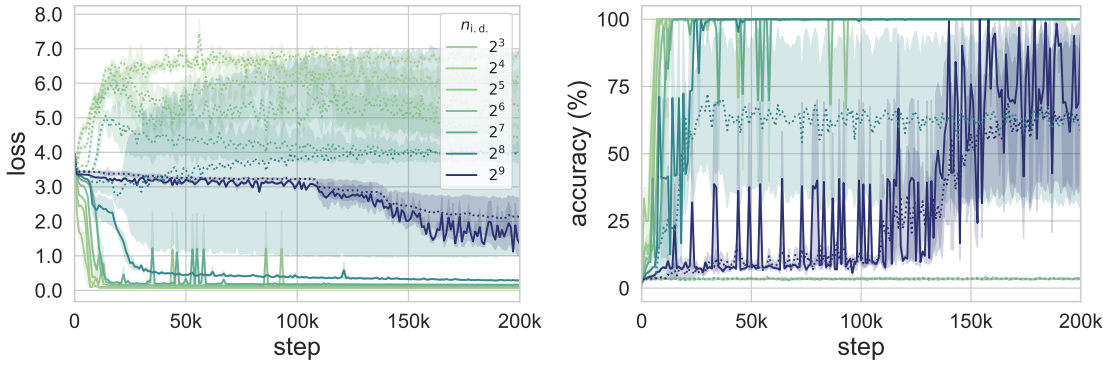


(a) y, layer 1 (b) y, layer 2 (c) z, layer 1 (d) z, layer 2

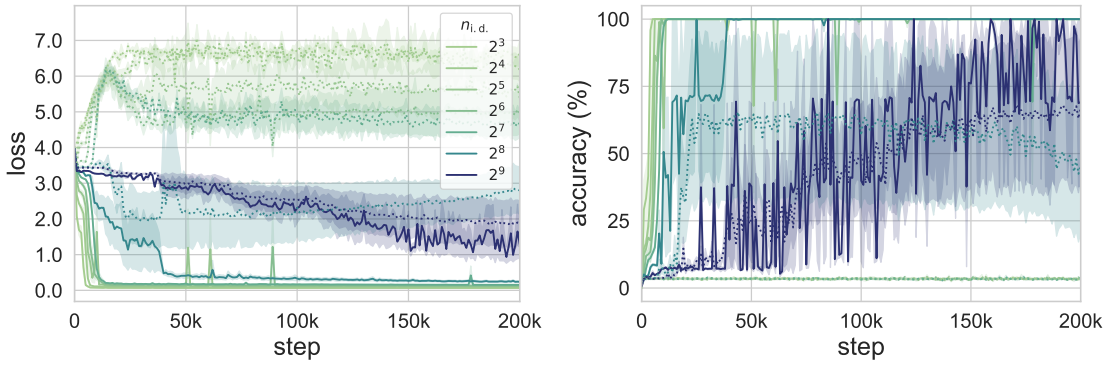
Figure D.14: Cosine-similarity of $d = 2$ model for every block, measured at 10-shot.

D.6 Additional Training Curves

We plot some selected training curves for $d = 4$ (Figure D.15) and $d = 2$ (Figure D.16) from Figure 5.7 phase diagrams. We see that even for $d = 4$, ICL can be a transient. With increased α or $n_{\text{i.d.}}$, the transient nature goes away.

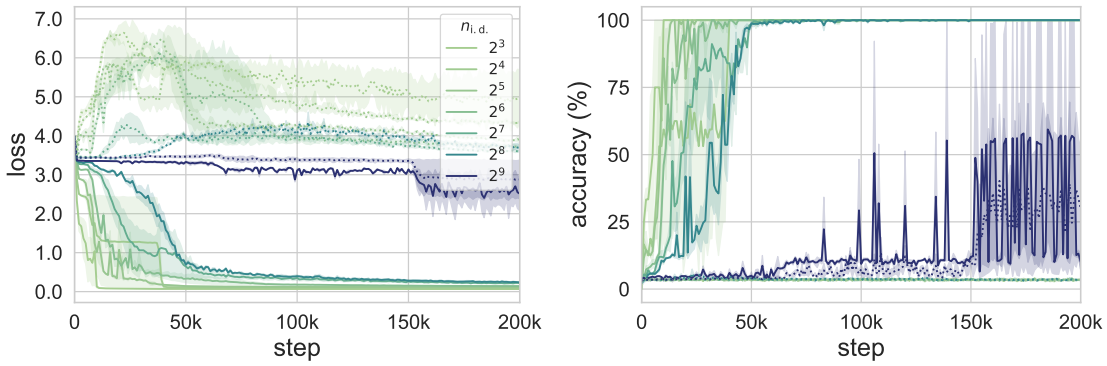


(a) $d = 4$, $\alpha = 0.7$

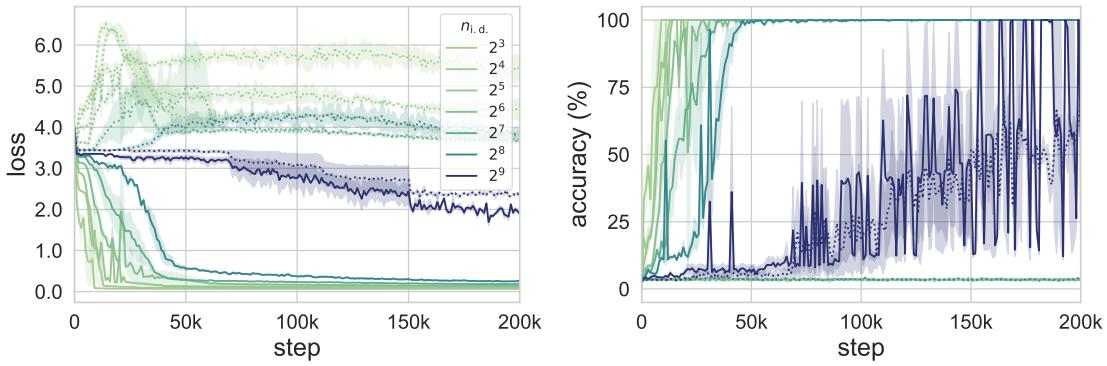


(b) $d = 4$, $\alpha = 0.6$

Figure D.15: Training curves for $d = 4$, averaged over three random seeds.



(a) $d = 2$, $\alpha = 0.7$



(b) $d = 2$, $\alpha = 0.6$

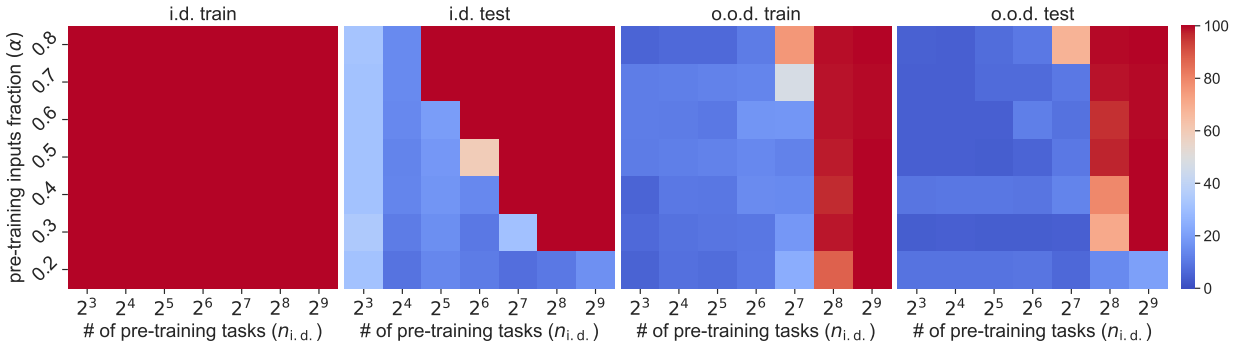
Figure D.16: Training curves for $d = 2$, averaged over three random seeds.

D.7 Additional Phase Diagrams

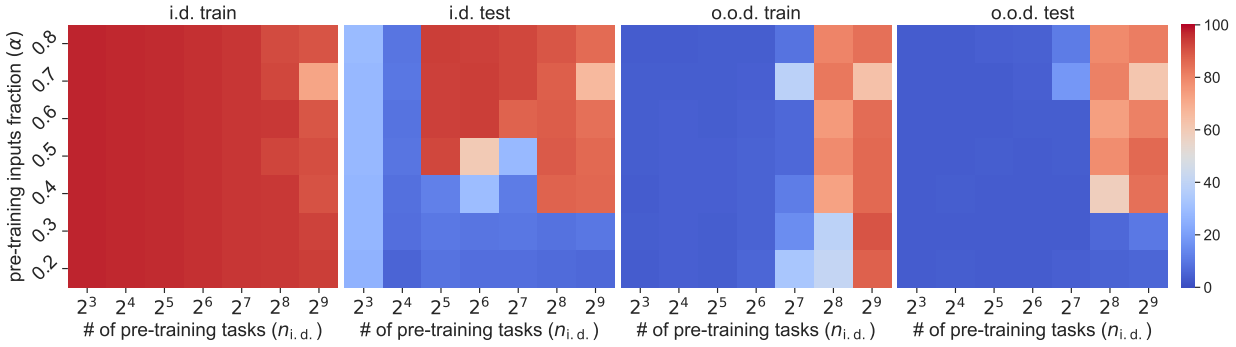
In Figure D.17, we plotted detailed/extended versions of the phase diagrams shown in Figure 5.2 and Figure 5.7. The four phases story we have shown in Figure 5.2 still holds for other depths.

D.8 Different Choice of p

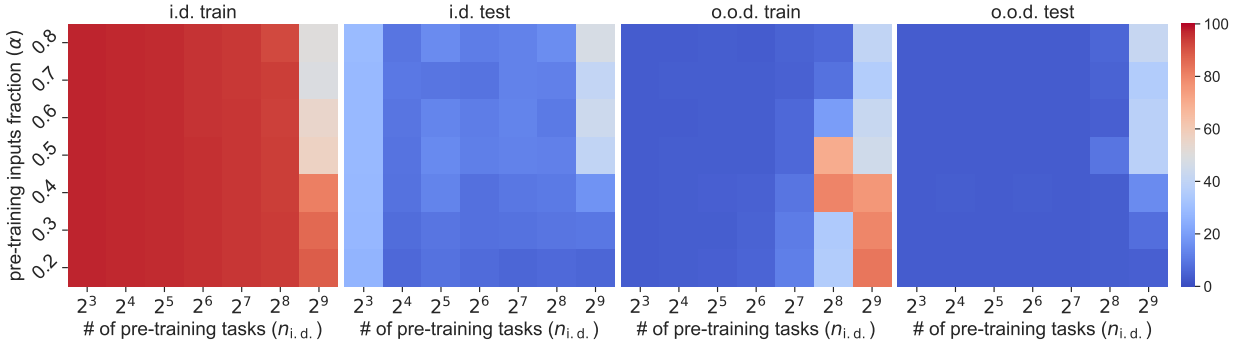
In this section, we check the effect of varying task difficulties, i.e. the value of p. In Figure D.18, we plotted o.o.d. generalization accuracy. Clearly as the task gets harder, the model



(a) $d = 6$, last shot



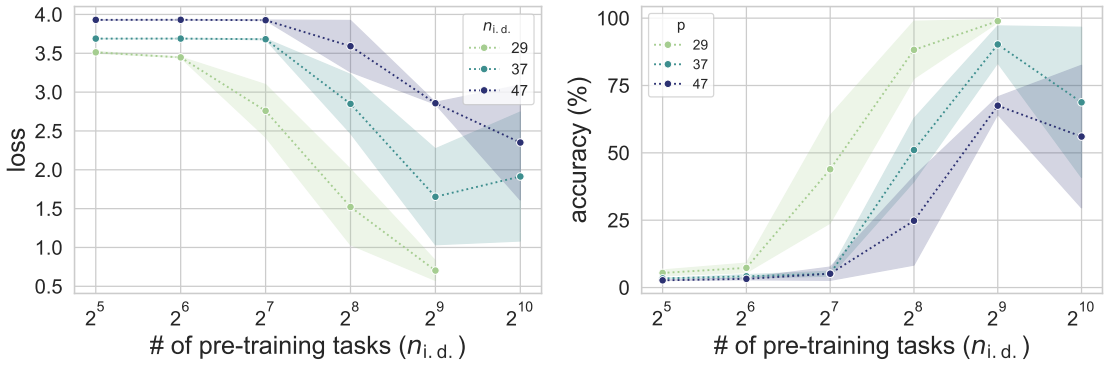
(b) $d = 4$



(c) $d = 2$

Figure D.17: Phase diagrams on all four sets for $d = 4$ and $d = 2$.

needs to see more tasks to generalize out-of-distribution.



(a)

Figure D.18: We compare the best performance for $d = 6$ models with different $n_{\text{i.d.}}$ values, averaged over three seeds. We use learning rate $\eta = 10^{-4}$ for $p = 37$ and $p = 47$, while keeping other hyperparameters the same as $p = 29$.

Appendix E: Transformers and Pseudo-Random Number Generators

E.1 Experimental Details

This section provides further details about model architecture, dataset construction, and optimization.

E.1.1 Dataset Construction

Fixed Modulus (`FM`): Given a modulus m, we apply the Hull-Dobell Theorem to determine the possible values of (a, c) that maximize the period. We then randomly select 64 values of a and c to generate the test dataset. To generate the training dataset, we *exlude* these test choices of (a, c) and uniformly sample $N = 100,000$ LCG sequences of length L (context length) with n_a values of multipliers and n_c values of increments. For each set of parameters (a, c), we sample an LCG sequence with a randomly selected initial seed x_0. Note that the training dataset includes sequences with varying periods, while the test data only contains sequences that maximize the period.

Generalization to Unseen Modulus (`UM`): For the test dataset, we first select a set of test moduli $M_{\text{test}} = \{m_{\text{test}}\}$ that would be reserved exclusively for evaluation. For each test modulus $m_{\text{test}} \in M_{\text{test}}$, we apply the Hull-Dobell Theorem to determine the values of (a, c) that maximize

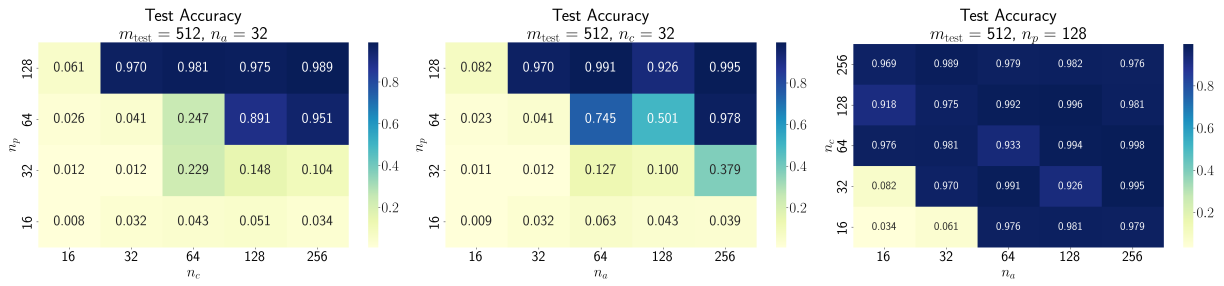


Figure E.1: The impact of training dataset parameters (n_m, n_a, n_c) on unseen modulus task performance.

the period. We then randomly select 64 values of a and c to generate the test dataset. These 64 (a, c) values are not considered while generating the training dataset.

For the training dataset generation, we sample the n_m modulus values from the range $[L, \lfloor 1.2 \max(M_{\text{test}}) \rfloor]$ while excluding all the values in M_{test}. For each modulus value m, we uniformly select n_a multipliers $(0 < a < m)$ and n_c increments $(0 \leq c < m)$, excluding the ones reserved for testing. For each parameter (a, c, m), we generate a sequence of length L using a randomly selected initial seed x_0. This results in a total of $N = n_m \times n_a \times n_c$ training sequences. We report that $N = 400,000$ served sufficient from the modulus values considered in this work.

Next, we examine the effect of training dataset composition (n_m, n_a, n_c) on the performance. Figure E.1 shows the test accuracy primarily depends on n_p and marginally on n_a and n_c. Furthermore, we found that $n_m \gtrsim m_{\text{test}}/4$ yields good generalization performance (result not shown here). Based on this relationship and our target total number of training examples N, we sample $n_a = n_c = \sqrt{\frac{N}{m_{\text{test}}/4}}$ values of multipliers and increments. Unless explicitly specified, we use these parameter settings as the default configuration for all experiments.

E.1.2 Model Architecture Details

We consider GPT-style Transformers [189] with learnable positional embeddings and weight tying [141]. The model architecture is characterized by the number of blocks (depth), embedding dimension (d_{model}), and number of attention heads (n_{heads}). For most experiments, we use GELU activations, except in Section 6.4, where we use ReLU activations for better interpretability.

E.1.3 Training Details

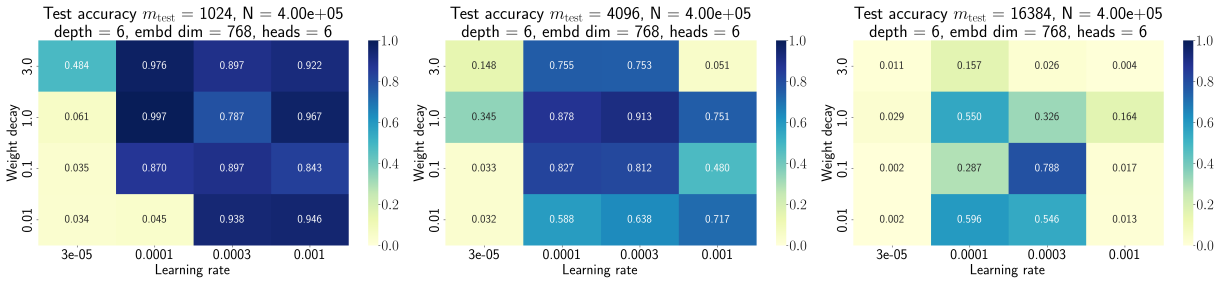


Figure E.2: Heatmap of test accuracy of a 6 layer Transformer with learning rate and weight decay as the axes. As the modulus is increased from 1024 to $16,384$ the range of hyperparameters resulting in reasonable accuracy becomes narrow.

We train the models with Cross-entropy loss using AdamW optimizer [97] with momentum hyperparameters $\beta_1 = 0.9$ and $\beta_2 = 0.99$. We implement a linear learning rate warmup over the first 2048 steps with an initial learning rate of zero and the target learning rate η. By default, all experiments employ a batch size of 256. Weight decay is only applied to non-bias parameters.

In the unseen modulus case, we observed that both the optimal target learning rate η and weight decay strength λ are highly sensitive to minute changes in training dataset properties (modulus m_{test}, number of LCG parameters n_m, n_a, n_c and total number of examples N) and architectural changes (depth and embedding dimension). To determine the optimal hyperpam-

raters, we scan the learning rates $\eta \in \{3e\text{-}05, 1e\text{-}04, 3e\text{-}04, 1e\text{-}03\}$ and weight decay strengths $\lambda \in \{0.01, 0.1, 1.0, 3.0\}$.

E.1.4 Training Cost

In Figure 6.11, the training of the $m = 2^{32}$ model was conducted using four NVIDIA A100 GPUs, requiring a total of 21.82 hours. The $m = 2^{16}$ model completed training in 4.83 hours under the same hardware setup. Despite having the same model size, the increased context length in the $m = 2^{32}$ model led to a significantly higher computational cost.

In Figure 6.12, both models were trained on a single NVIDIA H100 GPU for 85 hours.

E.1.5 Hyperparameter Space Shrinking with increasing modulus

We also report a surprising phenomenon in the unseen modulus case, which we refer to. as the 'hyperparameter space shrinking.' As we increase the test modulus m_{test} while keeping the model architecture fixed, we observe that the optimal learning rate (η) and weight decay strength (λ) shift significantly. Moreover, the range of these hyperparameter values that yield reasonable performance becomes increasingly narrow. Figure E.2 shows this result for a 6 layer Transformer.

E.2 LCG Properties

E.2.1 The period of the lower k-th bit when m is power of 2

In this section, we show that for a sequence of period $\mathcal{T}_m = m = 2^K$, the k-th lowest digit has a period of 2^k along the sequence. Consider an LCG sequence:

$$x_{t+1} = (ax_t + c) \mod m, \tag{E.1}$$

where m is a power of 2, m and c are coprime and $a - 1$ is divisible by 4. The period of sequence x_t is m [139]. The lower k-th bits of x_t is given by:

$$b_{t,k} = \frac{z_{t,k} - z_{t,k-1}}{2^{k-1}}, \tag{E.2}$$

where $z_{t,k} = x_t \mod 2^k$. Therefore, if $z_{t,k}$ has a period of 2^k, then the lower k bits also have a period of 2^k. Below, we show that $z_{t,k}$ has a period of 2^k.

For an integer M_t, we can can re-write x_t as:

$$x_t = z_{t,k} + M_t 2^k. \tag{E.3}$$

Next, we substitute Equation (E.1) into the definition of $z_{t+1,k}$:

$$z_{t+1,k} = x_{t+1} \mod 2^k,$$

$$= [(ax_t + c) \mod m] \mod 2^k.. \tag{E.4}$$

As m is divisble by 2^k, this simplifies to:

$$\begin{aligned} z_{t+1,k} &= (az_{t,k} + c) \mod 2^k, \\ &= (az_{t,k} + aM_t 2^k + c) \mod 2^k, \\ &= (az_{t,k} + c) \mod 2^k. \end{aligned} \tag{E.5}$$

Therefore, $z_{t,k}$ follows its own LCG recurrence with the same a and c but with a reduced modulus 2^k. Because 2^k and c are coprime and $a - 1$ is divisible by 4, the period of z_n is 2^k. Thus, the period of the lower k bits is 2^k.

Since $z_{n,k}$ has period 2^k and $z_{n,k-1}$ has period 2^{k-1}, the period of $b_{n,k}$ is 2^k.

E.2.2 Derivation of Equation (6.4)

We now derive Equation (6.4) of the main text:

$$x_t \mod p_i^{w_q} = \alpha_{i,0,t}\, p_i^0 + \alpha_{i,1,t}\, p_i^1 + \cdots + \alpha_{i,w_i-1,t}\, p_i^{w_i-1},$$

where $\alpha_{i,w,t} \in \{0, 1, \ldots, p_i - 1\}$ are base-p_i digits. When the period of x_t is m, each digit $\alpha_{j,w,t}$ has a period of p_i^w.

Consider an LCG sequence:

$$x_{t+1} = (ax_t + c) \mod m, \tag{E.6}$$

where m has a prime factorization $m = p_1^{w_1} p_2^{w_2} \cdots p_q^{w_q}$, m and c are coprime, $a - 1$ is divisible by all prime factors of m and $a - 1$ is divisible by 4 if m is divisible by 4 [139].

Consider the residulas $R_{i,t} = x_t \mod p_i^{w_i}$, where $i \in \{1 \dots q\}$. We have:

$$x_t = R_{i,t} + M_{i,t} p_i^{w_i}, \tag{E.7}$$

where $M_{i,t}$ is an integer.

Substituting Equations (E.6) and (E.7) into the definition of $R_{i,t+1}$:

$$\begin{aligned} R_{i,t+1} &= x_{t+1} \mod p_i^{w_i}, \\ &= [(ax_t + c) \mod m] \mod p_i^{w_i}, \\ &= (ax_t + c) \mod p_i^{w_i}, \\ &= (aR_{i,t} + aM_{i,t} p_i^{w_i} + c) \mod p_i^{w_i}, \\ &= (aR_{i,t} + c) \mod p_i^{w_i}. \end{aligned} \tag{E.8}$$

Next we consider $z_{i,k,t}$, which is the lower k base-p_i digits of $R_{i,t}$:

$$z_{i,k,t} = R_{i,t} \mod p_i^k. \tag{E.9}$$

Similarly, the recurrence of $z_{i,k,t}$ can be simplied as:

$$\begin{aligned} z_{i,k,t+1} &= R_{i,t+1} \mod p_i^k, \\ &= [(aR_{i,t} + c) \mod p_i^{w_i}] \mod p_i^k, \\ &= (aR_{i,t} + c) \mod p_i^k, \\ &= (az_{i,k,t} + c) \mod p_i^k. \end{aligned} \tag{E.10}$$

$z_{i,k,t}$ follows an LCG recurrence with the same a and c and a reduced modulus p_i^k. Because p_i^k and c are coprime and $a - 1$ is divisible by p_i, the period of $z_{i,k,t}$ is p_i^k.

$\alpha_{i,k,t}$, which is the lower k-th base-p_i digits of $y_{i,t}$, can be written as:

$$\alpha_{i,k,t} = \frac{z_{i,k,t} - z_{i,k-1,t}}{p_i^{k-1}}. \tag{E.11}$$

Since $z_{i,k,t}$ has period p_i^k, the period of $\alpha_{i,k,t}$ is p_i^k.

E.3 Fixed Modules Training Results

For the **FM** case, we find that a single attention head in one layer is sufficient to solve the task. In Figure E.3, we present the model's training loss and performance across training steps for $m = 2048$. Notably, panels (c, f) reveal a significant disparity between training and test loss, indicating a grokking transition during the training process. Similarly, we plot in Figure E.4 for similar curves for $m = 7776$, where all curves are qualitatively the same.

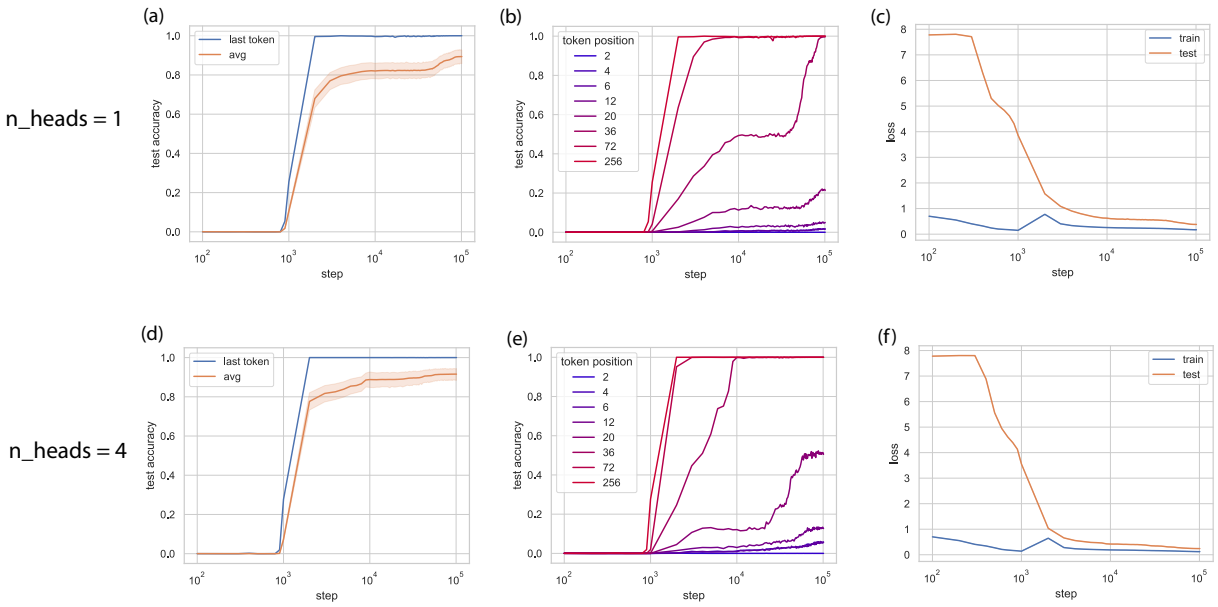


Figure E.3: Test accuracy and train/test loss for $m = 2048 = 2^{11}$, depth=1. (a,b,c) $n_{\text{heads}} = 1$ (d,e,f) $n_{\text{heads}} = 4$.

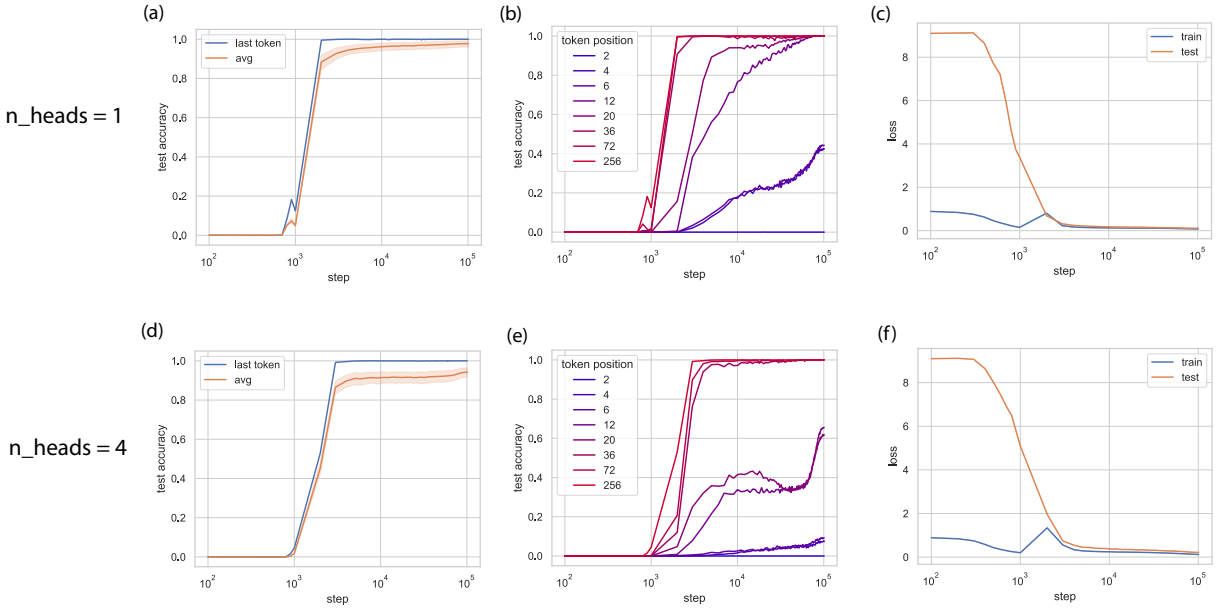


Figure E.4: Test accuracy and train/test loss for $m = 7776 = 2^5 \cdot 3^5$, depth=1. (a,b,c) $n_{\text{heads}} = 1$ (d,e,f) $n_{\text{heads}} = 4$.

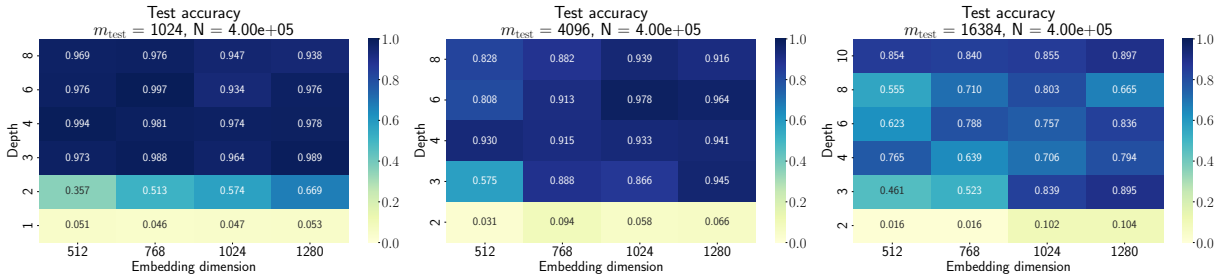


Figure E.5: Test accuracy heatmaps of with depth and embedding dimensions as the two axes.

E.4 Critical Depth for the Unseen Modulus Task

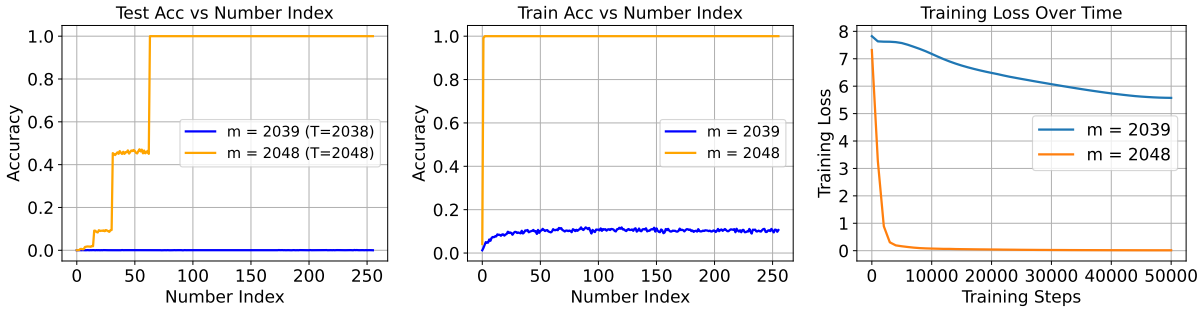
This section analyzes the depth and embedding dimension requirements for successfully training a Transformer on the unseen modulus task. The experimental details are the same as described in Appendix E.1.

We varied depths $\in \{2, 3, 4, 6, 8\}$ and embedding dimensions in $d_{\text{model}} \in \{512, 768, 1024, 1280\}$, with the head dimension fixed to $d_{\text{head}} = 128$. For each depth and width, we scanned learning rates $\eta \in \{3e\text{-}05, 1e\text{-}04, 3e\text{-}04, 1e\text{-}03\}$ and weight decay strengths $\lambda \in \{0.01, 0.1, 1.0, 3.0\}$ to identify the optimal hyperparameters. We report that the optimal learning rate and weight decay strength heavily vary with depth, embedding dimension, and training dataset. For $m_{\text{eval}} = \{1024, 4096\}$, the models were trained for $T = 100,000$ steps, while for $m_{\text{eval}} = 16,384$, the models required a longer training for $T = 200,000$ steps.

Figure E.5 shows the test accuracy heatmaps with depth and embedding dimensions as the two axes. These results demonstrate that a minimum depth of 3 is required to learn the LCG sequence prediction task, with a marginal dependence on embedding dimension. This suggests the unseen modulus task requires a minimal computational depth of three to capture the underlying structure of LCGs.

E.5 Prime Moduli

In the **FM** setting, when m is a prime number, the task becomes much harder. Since there are no digit-wise periodic patterns, the model cannot perform the algorithm described in Section 6.4.2. In Figure E.6, we trained two identical models to learn $m = 2039$ and $m = 2048$, and we observe that the task with $m = 2039$ cannot be learned within the same number of training steps. Note that to rule out potential constraints from model capability, we used depth 2 models (as oppposed to depth 1 in the main text) in both cases.



(a) Test Accuracy (b) Training Accuracy (c) Training Loss

Figure E.6: **FM:** Comparison between $m = 2039$ (prime) and $m = 2048$ (power-of-two), with both models trained for 50,000 steps. Each model has depth 2 and $d_{\text{model}} = 1024$. The test set consists of sequences with the same periods, while the training set includes arbitrary multipliers not present in the test set.

In the **UM** setting, the model exhibits similar test performance on sequences with prime m and sequences with m as a power of two, as shown in Figure E.7. We hypothesize that training on a diverse set of moduli helps the model rely less on the digit patterns and instead focus on more generalizable structure.

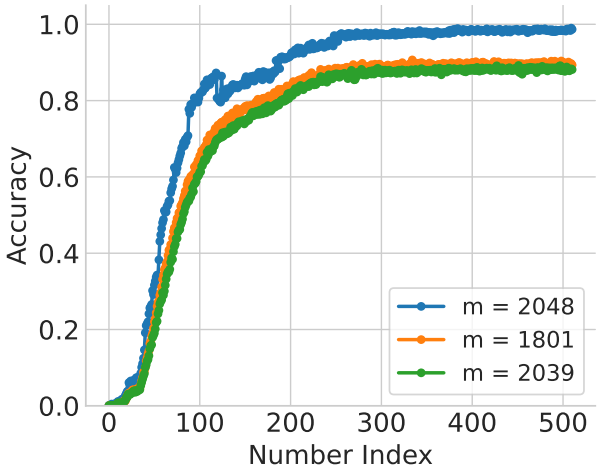


Figure E.7: **UM:** Test accuracy comparison between $m = 2039$ (prime), $m = 1801$ (prime) and $m = 2048$ (power-of-two). The model was trained for 100,000 steps on a dataset consisting of 262,144 sequences with 512 distinct training moduli not present in the test set.

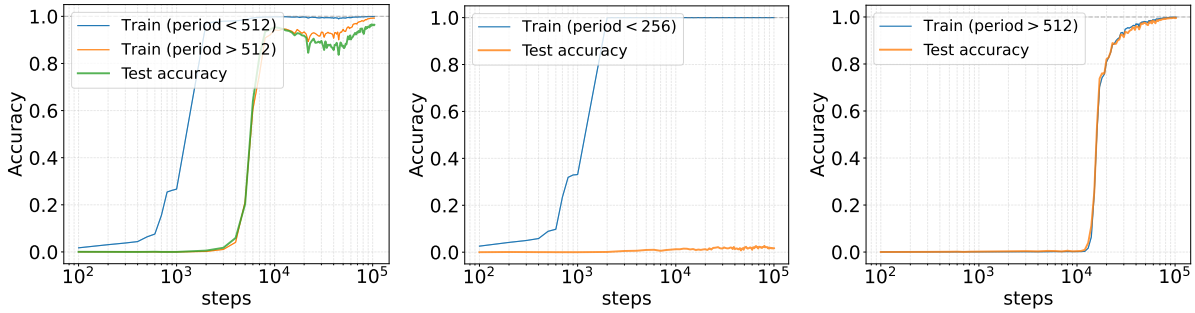


Figure E.8: (left) Comparison of the training accuracy of sequences with different periods relative to the context length 512, (center) Accuracy when the model is only trained on sequences with period < 512, (right) Accuracy when the model is trained on sequences with period > 512.

E.6 Training Time Interpretability

In this section, we examine the order in which training sequences with different periods are learned during training. For this experiment, we consider a six-layer Transformer with 4 heads and an embedding dimension of 768. For this experiment, we generate sequences of length 512 using 128 unseen moduli and 128 values of a and c each. The model is trained with Adam hyperparameters: $\eta = 3 \times 10^{-4}$ $\beta_1 = 0.9, \beta_2 = 0.99$ and weight decay strength $\lambda = 1.0$.

Figure E.8(left) compares the training accuracy of sequences with different periods relative to the context length 512. We observe that sequences with period < 512 are memorized early in training, while the accuracy of long-period sequences coincides with the test accuracy. Next, we perform two more experiments by training the model on datasets consisting of (1) sequences with period < 256, and (2) sequences with period > 512. Figure E.8(center, right) show the results of these experiments. The model fails to generalize when trained only on low-period sequences, while training only on long-period sequences eliminates grokking.

E.7 Fixed Modulus Interpretability

In this subsection, we show additional results on the model's behavior on the **FM** task.

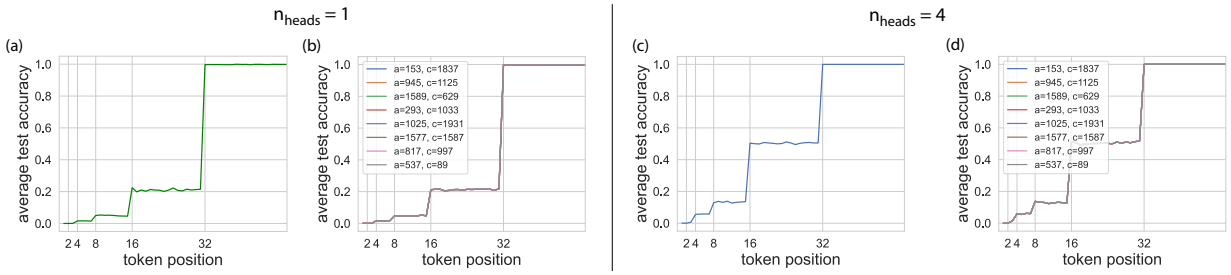


Figure E.9: Test accuracy vs token positions for $m = 2048 = 2^{11}$, depth=1, various values of a, c. (a,b) $n_{\text{heads}} = 1$ (c,d) $n_{\text{heads}} = 4$. The accuracies for different a and c are exactly on top of each other.

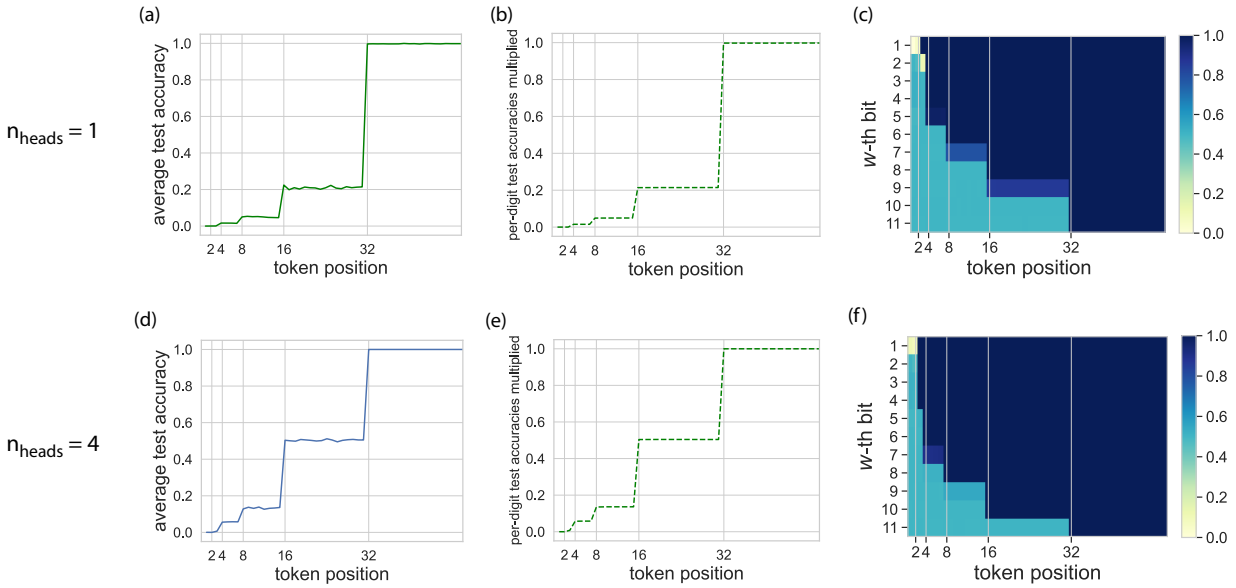


Figure E.10: Test accuracy for $m = 2048 = 2^{11}$, depth=1, $n_{\text{heads}} \in \{1, 4\}, d_{\text{model}} = 768$. (a,d) Test accuracy averaged over a, c and initial seeds. (b,e) Multiplication of per-digit test accuracies – matches exactly with the average test accuracy. (c,f) Per-digit accuracy in binary representation.

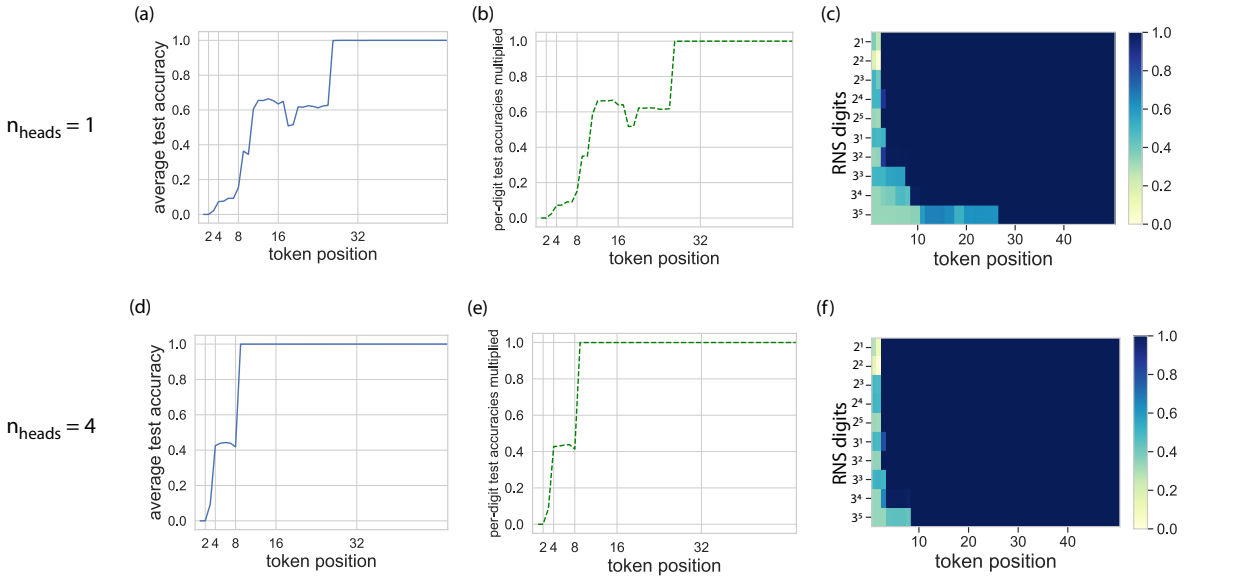


Figure E.11: Test accuracy for $m = 7776 = 2^5\, 3^5$, depth=1, $n_{\text{heads}} \in \{1, 4\}, d_{\text{model}} = 768$. (a,d) Test accuracy averaged over a, c and initial seeds. (b,e) Multiplication of per-digit test accuracies – matches exactly with the average test accuracy. (c,f) Per-digit accuracy in RNS representation.

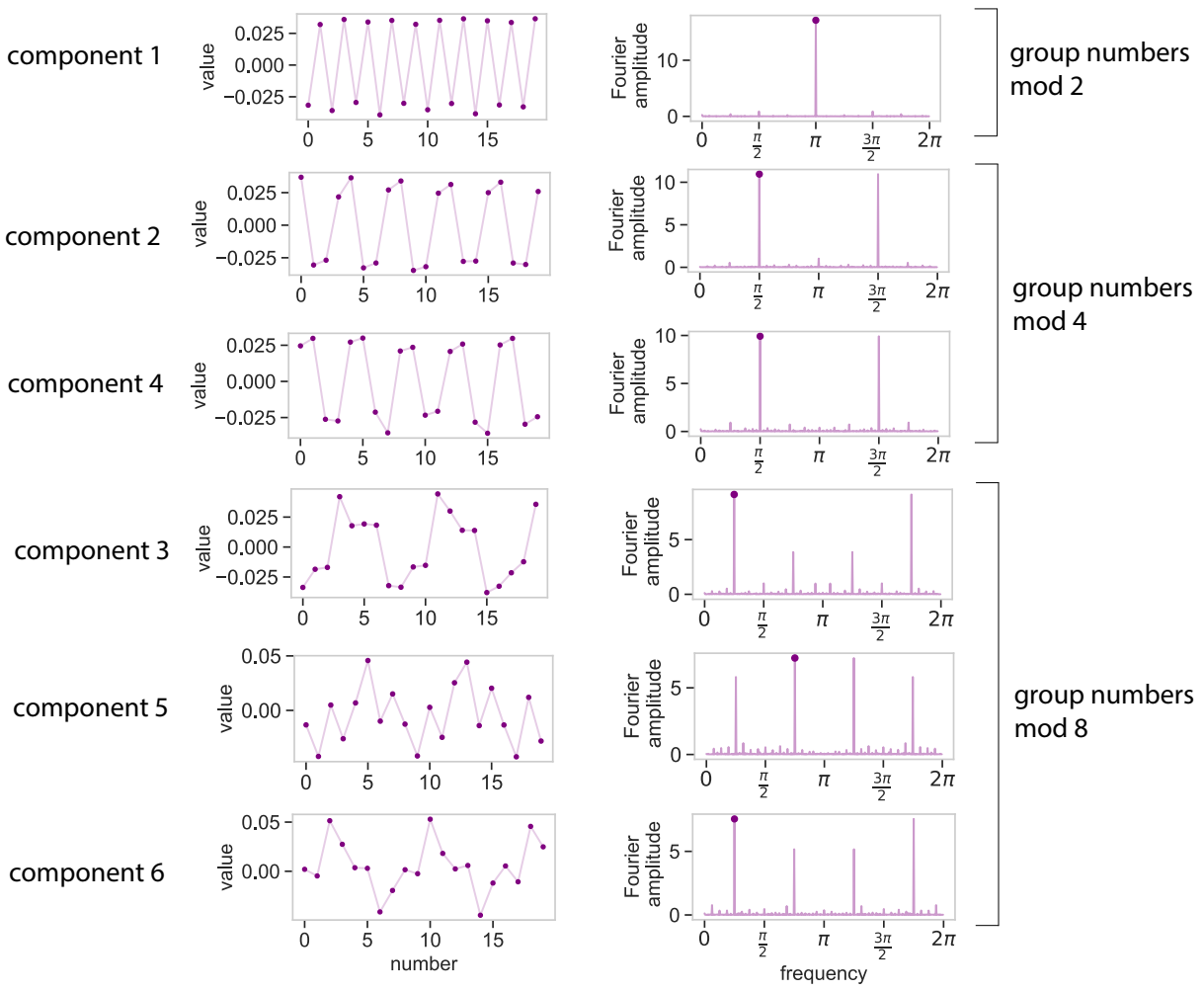


Figure E.12: Projections along Top 6 principal components of the embedding matrix, for $m = 512$, depth=1, $n_{\text{heads}} = 1, d_{\text{model}} = 768$.

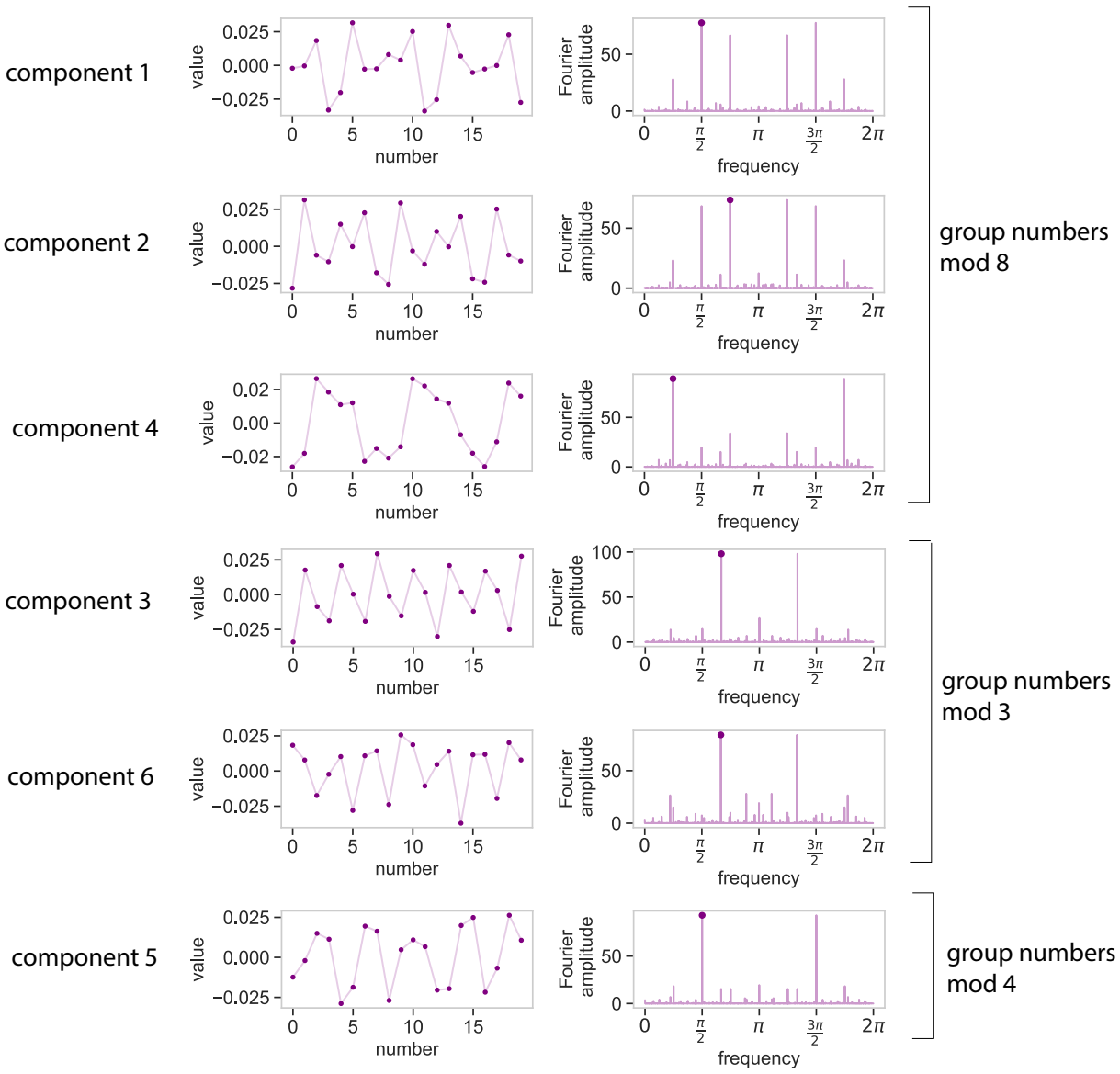


Figure E.13: Projections along Top 6 principal components of the embedding matrix, for $m = 7776$, depth=1, $n_{\text{heads}} = 1, d_{\text{model}} = 768$.

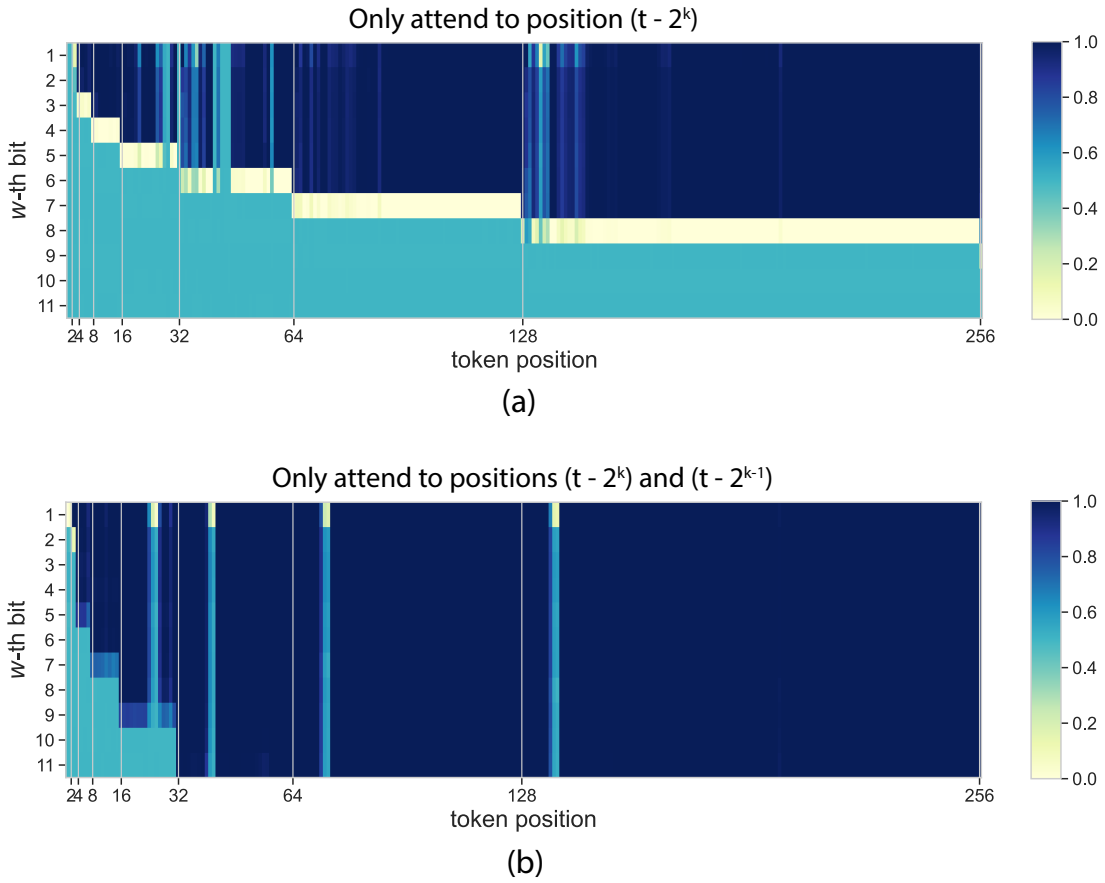


Figure E.14: Per-digit test accuracies in binary representation, with attention masking (at inference time); for $m = 2048 = 2^{11}$, depth=1, $n_{\text{heads}} = 1, d_{\text{model}} = 768$, averaged over a, c and initial seeds. (a) For each query, only the attention to key at position $t - 2^k$ (where $k = \lfloor \log_2 t \rfloor$) is kept. All other attention weights are masked out. In this case, the model can successfully copy the last k bits, but cannot predict higher bits. (b) For each query, only the attention to key at positions $t - 2^k$ and $t - 2^k$ are kept. All other attention weights are masked out. In this case, the model can predict higher bits with fairly high accuracy – this is remarkable given that majority of the attention weights are masked out.

E.8 Unseen Modulus Interpretability

E.8.1 Extra PCA plots for first layer heads

We present additional PCA analyses of attention heads shown in Figure 6.8, examining $\mathrm{PCA}(H^{(h)}[:, t, :])$ for each head h across various combinations of a, c, x_0, m_{test}, and position t. The results are depicted in Figure E.15, where we include two additional first-layer heads (heads 2 and 3) that are responsible for performing operations related to modulo 5. Notably, with different choices of a, c, m_{test} and t compared to Figure 6.8 (a1, b1, c1), the clustering behavior remains unchanged.

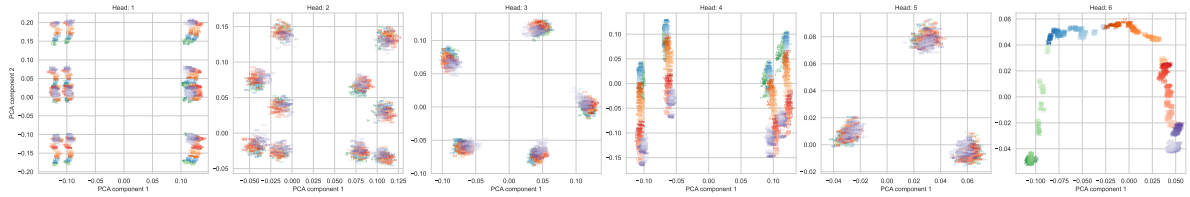
Note the emergence of attention heads dedicated to processing modulo 2, 3, 5, and 7 can likely be attributed to the prevalence of these prime factors in the training set.

E.8.2 Pruning heads corresponding to irrelevant prime factors

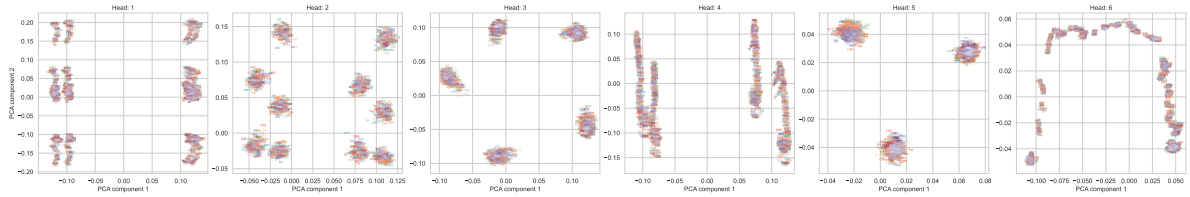
To further validate our findings from Figure 6.8 in Section 6.4.3 regarding the correlation between attention heads and digit-wise accuracy, we conduct additional pruning experiments. The results of these experiments are presented in Figures E.16 and E.17.

E.8.3 Patching other heads

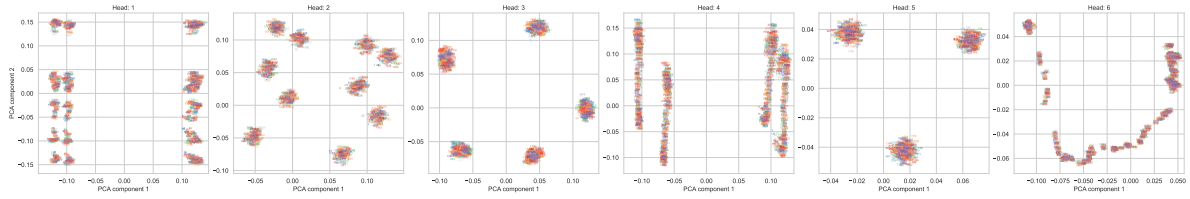
In Figure E.18, we present patching experiments for heads not previously shown in Figure 6.9. We focus only on selected heads where patching significantly affects predictions. Notably, none of these cases exhibit qualitative changes stemming from modulus alterations, further supporting our assertion that the head shown in Figure 6.9 is specifically responsible for estimat-



(a) $a = 13$, $c = 3$, $m_{\text{test}} = 2048$, $t = 0$

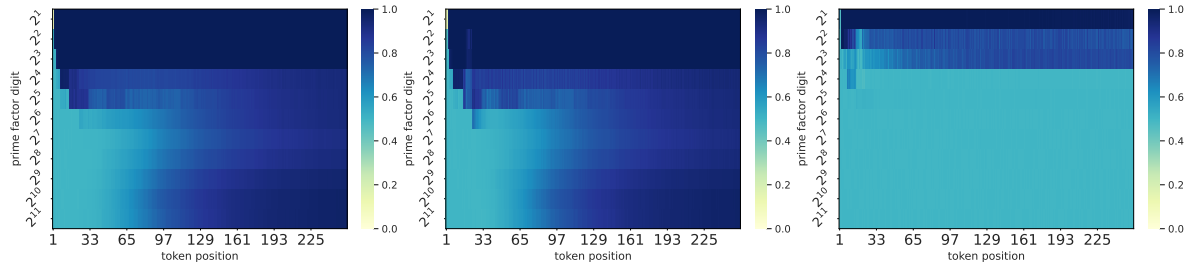


(b) $a = 9$, $c = 3$, $m_{\text{test}} = 2048$, $t = 128$



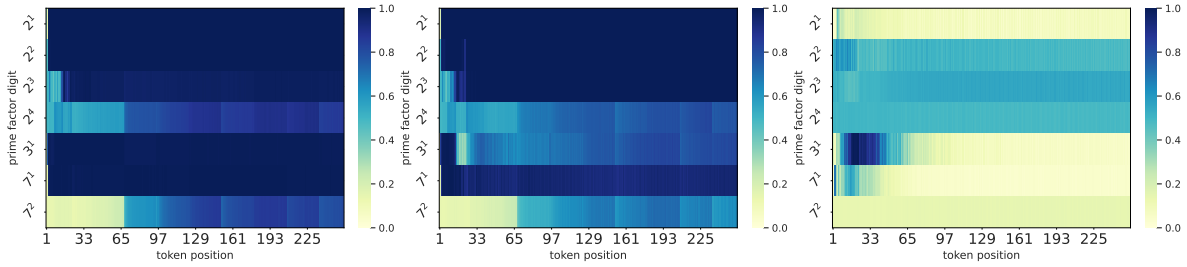
(c) $a = 61$, $c = 7$, $m_{\text{test}} = 1800$, $t = 47$

Figure E.15: PCA analysis of first-layer heads. Although the ordering of numbers varies, the grouping behavior discussed in Section 6.4.3 remains invariant to changes in a, c, m, x_0, and t. Note that the last head is the head that appears in Figure 6.9, which does not exhibit a strong grouping bias.



(a) Original Model (b) Prune modulo 3 head (c) Prune modulo 14 head

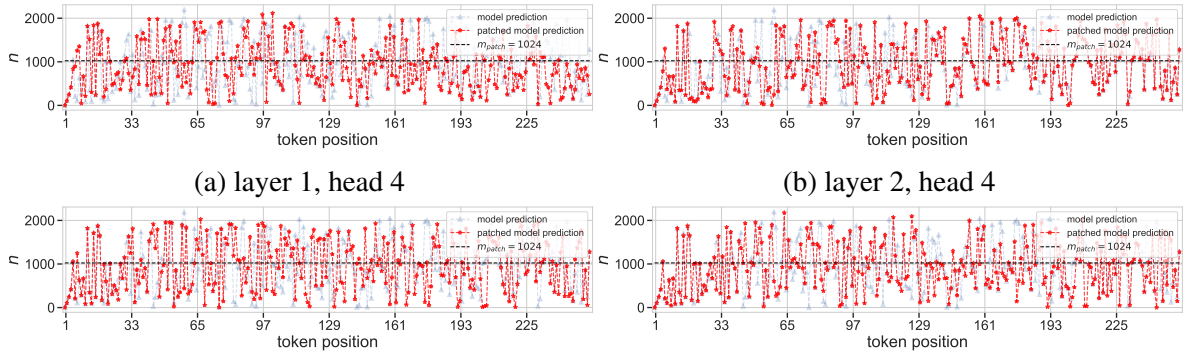
Figure E.16: Per-digit accuracy for $m_{\text{test}} = 2048$ after pruning specific attention heads. (a) Results from the same model used in Section 6.4.3, identical to Figure 6.8 (b2); (b) Performance after pruning the attention head responsible for grouping numbers by their values modulo 3, which is irrelevant for solving sequences with m_{test} (containing only the prime factor 2). After pruning, the model's performance shows marginal improvement for specific early bits at lower token positions; (c) Performance after pruning the attention head responsible for grouping numbers by their values modulo 14. The model's performance on m_{test} decreases significantly, as this head partially contributes to processing relevant prime factors. However, since the primary head responsible for binary representation remains intact, the model maintains partial functionality.



(a) Original Model (b) Prune modulo 3 head (c) Prune modulo 2 head

Figure E.17: Per-digit accuracy for $m_{\text{test}} = 2352$ after pruning specific attention heads. (a) Results from the same model used in Section 6.4.3, identical to Figure 6.8 (b2); (b) Performance after pruning the attention head responsible for grouping numbers by their values modulo 3, which is relevant for one specific digit in this case. After pruning, the model's performance shows a clear degradation, with the strongest one happening exactly at the digit corresponding to modulo 3 (c) Performance after pruning the attention head responsible for grouping numbers by their values modulo 2. The model's performance on m_{test} got obliterated, as there are many base-2 digits in the RNS representation of m_{test} in this case.

ing m_{test}.



(a) layer 1, head 4 (b) layer 2, head 4

(c) layer 3, head 6 (d) layer 4, head 1

Figure E.18: Patching experiments following the setting of Figure 6.9 in the main text. None of these heads, after patching, make the model believe that the modulus is close to m_{patch}.

E.8.4 Inaccurate estimation of m_{test}

From the cosine-similarity panel in Figure 6.9, we observe that the model's estimation approximates the target $m_{\text{test}} = 2048$. However, detailed analysis reveals that the highest cosine-similarity occurs at $m_{\text{est}} = 2033 = 19 \cdot 107$, with neighboring values exhibiting similarly high

cosine-similarity values. If we assume m_{est} represents the model's internal belief, then the prime representation would consist solely of powers of 19 and 107. Such a representation can only produce periodic structures for the k-th bit of a binary number when $r = \text{lcm}(19, 2^k)$. Consequently, patterns before reaching that period would appear random in this representation, providing a weaker signal compared to the correct representation. This explains why the model preferentially selects binary representation for lower bits when $m_{\text{test}} = 2048$.

For higher bits, the low-bit representation can be determined up to 2^k bits through copying. If the model utilizes this information in later stages, the precision of m_{est} can be drastically improved. Specifically, when $\lfloor m_{\text{est}}/2^k \rceil = \lfloor m_{\text{test}}/2^k \rceil$, the model can identify the correct representation as long as $|m_{\text{test}} - m_{\text{est}}| < 2^k$. This argument can be extended to any other composite m_{test}. Note that this argument is hypothetical; further proof of this mechanism remains in future work.

E.8.5 Evidence for Step iii

In Figure E.19, we present attention patterns and token distance statistics for a selected attention head in later layers, with the same model as the one used in Section 6.4.3. The token distances are measured by the spacing between keys with top-4 attention weights for a given query. Our analysis reveals that the model develops a head capable of dynamically adjusting their lookback distance for computations based on m_{test}, which we interpret as evidence for **step iii** of the algorithm proposed in Section 6.4.3.

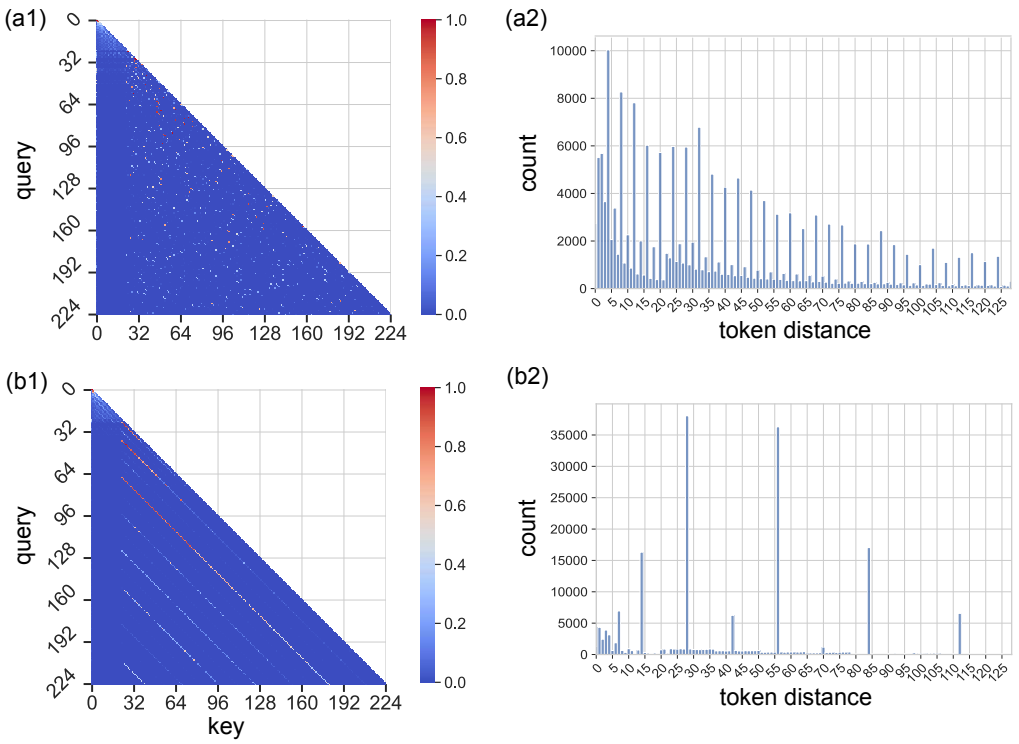


Figure E.19: Attention patterns and token distance statistics for layer 2, head 3 of the model analyzed in Section 6.4.3. (a1, b1) Results for a sequence with $m_{\text{test}} = 2048$, $a = 5$, and $c = 31$. The statistics reveal that the model consistently looks back at distances that are multiples of 4, which divides m_{test} and enables the correct copy behavior. (a2, b2) Results for a sequence with $m_{\text{test}} = 2352$, $a = 85$, and $c = 5$. In contrast to panels (a1, b1), the same attention head now consistently looks back at distances that are multiples of 14, which allows the model to copy lower digits from the context. This adaptive behavior demonstrates that the model has acquired the ability to dynamically adjust its lookback distance by r iterations to copy the lower digits and leave the higher digits to the later layers for computation.

E.9 Scaling Up the Modulus

E.9.1 Base-b tokenization

To convert an integer x to base b with the least significant digit (LSD) first, repeatedly divide x by b, storing remainders:

$$d_k = x \mod b, \quad x = \lfloor x/b \rfloor. \tag{E.12}$$

Stop when $x = 0$. The sequence $(d_0, d_1, \dots)$ is the base-b representation in LSD-first order.

Example: Converting $x = 3,214,748,365$ to base 256:

$$3,214,748,365 \div 256 = 12,557,610 \text{ remainder } 205, \quad d_0 = 205,$$

$$12,557,610 \div 256 = 49,053 \text{ remainder } 42, \quad d_1 = 42,$$

$$49,053 \div 256 = 191 \text{ remainder } 157, \quad d_2 = 157,$$

$$191 \div 256 = 0 \text{ remainder } 191, \quad d_3 = 191.$$

The final LSD-first representation consists of four tokens:

$$(205, 42, 157, 191)_{256}.$$

Each token is then one-hot encoded into a b-dimensional vector, where only the index corresponding to the token value is set to 1. These one-hot vectors are then fed into a token embedding layer with an embedding dimension of $d_{model} = 1024$.

E.9.2 Abacas Embeddings

The positional embedding for the j-th lower digit of the i-th number in the sequence is defined as:

$$\text{PosEmbed}(T_{i,j}) = E_{\text{int}}(i) + E_{\text{digit}}(j) \tag{E.13}$$

where $T_{i,j}$ represents the token corresponding to the j-th digit of the i-th integer.

- $E_{\text{int}}(i), E_{\text{digit}}(j) \in \mathbb{R}^{d_{\text{model}}}$ are learnable embeddings.
- $E_{\text{int}}(i)$ encodes the integer's position in the sequence.
- $E_{\text{digit}}(j)$ encodes the digit's relative position within the integer.

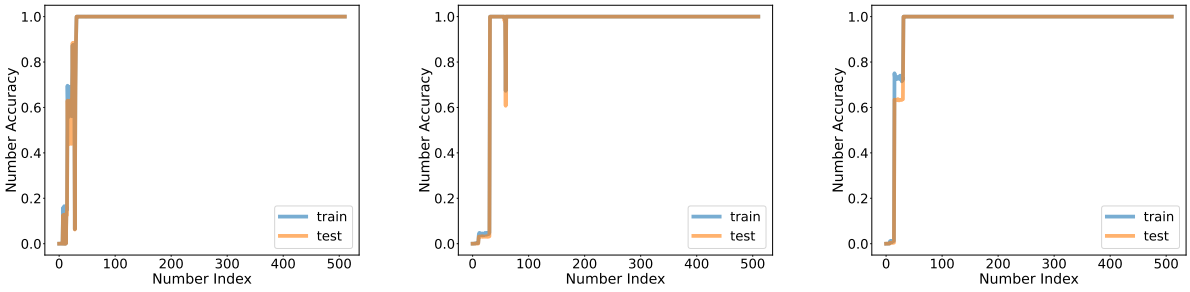
This embedding scheme ensures that each token captures both the integer's global position and the byte's local position.

E.9.3 Fixed Modulus

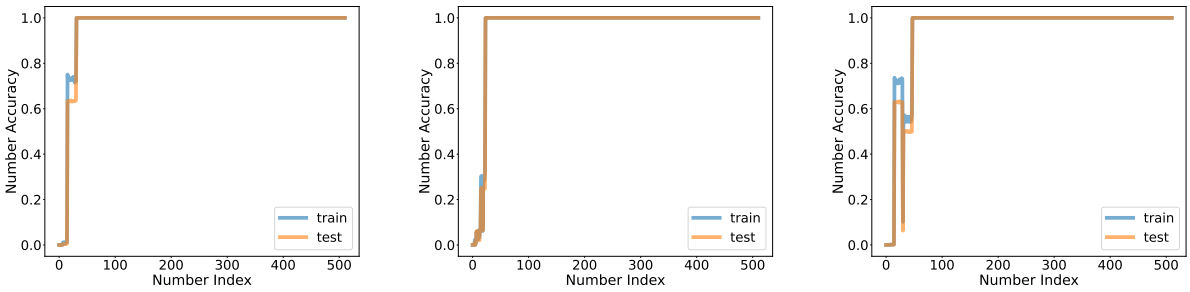
For each modulus $m = 2^k$, where $k \in [16, 32]$, we train a 2-layer GPT model with an embedding dimension of 1024 and a vocabulary size 256. The train set consists of $n_a = 1024$ multipliers and $n_c = 1024$ increments, selected via the Hull-Dobell theorem. One LCG sequence of length 512 is included in the train set for each (a, c) pair, resulting in a total training set size of $n_a \times n_c = 1,048,576$. For each modulus, the model is trained for 200,000 steps with a batch size of 512. The context length is $512\times$ the digit length of m in the byte representation -1. For $m = 2^{32}$, the digit length in byte representation is 4; therefore, the context length is 2047. Training was performed using 4 A100 GPUs over a total duration of 21.82 hours. For $m = 65536$, the digit length in byte representation is 2, resulting in a context length of 1023, with training taking 4.83 hours. For each modulus, the test set includes 512 unseen a values and 64 unseen c values selected via the Hull-Dobell theorem.

The model may converge to different solutions depending on two random seeds: one for model initialization and batch shuffling, and another for dataset generation. Figure E.22 shows the median performance across five runs, with the shaded region representing the range between

the minimum and maximum values. For larger moduli, not all models successfully find a solution that achieves 100% test accuracy.



(a) PyTorch seed = 9, dataset seed = 71 (b) PyTorch seed = 10, data seed = 71 (c) PyTorch seed = 11, data seed = 71

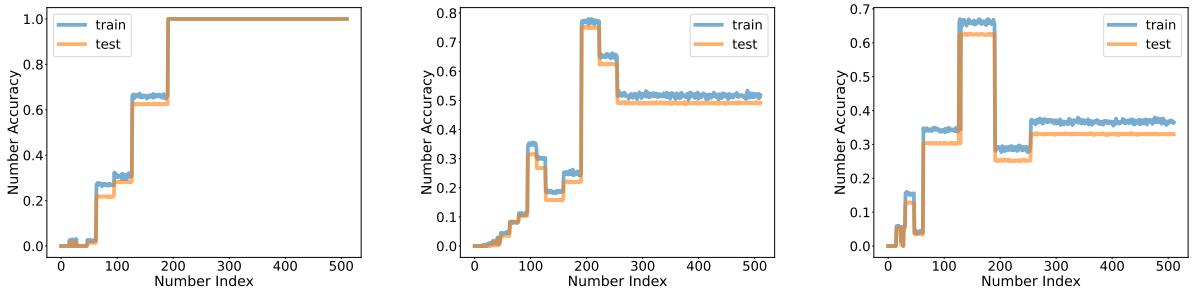


(d) PyTorch seed = 11, data seed = 71 (e) PyTorch seed = 11, data seed = 72 (f) PyTorch seed = 11, data seed = 73

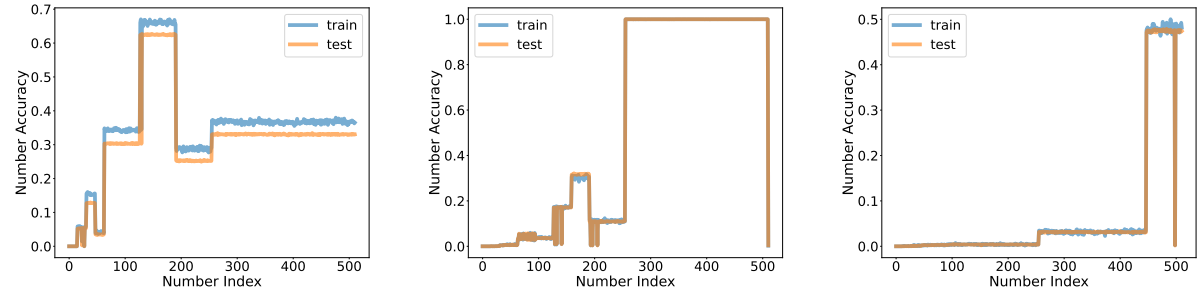
Figure E.20: Test accuracy vs. Number index for $m = 2^{20}$. First Row: Three models trained on the same dataset, each using a unique PyTorch random seed that controls model initialization and batch shuffling. Second Row: Three models trained on different datasets, with each dataset generated using a unique random seed controlling NumPy randomness for sampling a, c, and x_0. All models converged to solutions that achieved and sustained 100% test accuracy, but differed in the number of in-context examples required to reach this performance.

E.9.4 Unseen Modulus

We train a 6-layer GPT model on a dataset that comprises $n_m = 32,768$ moduli, with $n_a = 128$ training a values and $n_c = 1$ training c values per modulus. This results in a total of $n_m \times n_a \times n_c = 4,194,304$ sequences, each of length 512 in the training set. In Figure E.23a, where the tokenization base is 256, $1024 < m_{\text{train}} < 65536$. In Figure E.23b, where the tokenization



(a) PyTorch seed = 9, dataset seed = 71 (b) PyTorch seed = 10, dataset seed = 71 (c) PyTorch seed = 11, dataset seed = 71



(d) PyTorch seed = 11, dataset seed = 71 (e) PyTorch seed = 11, dataset seed = 72 (f) PyTorch seed = 11, dataset seed = 73

Figure E.21: Test accuracy vs Number index for $m = 2^{32}$. First Row: Three models trained on the same dataset, each using a unique PyTorch random seed that controls model initialization and batch shuffling. Second Row: Three models trained on different datasets, with each generated using a unique random seed controlling NumPy randomness for sampling a, c, and x_0. Only one of the five models found a solution that achieved and sustained 100% test accuracy.

base is 243, $1024 < m_{\text{train}} < 59049$. Multipliers are selected based on the Hull-Dobell theorem when sufficient qualifying a values are available; otherwise, random a values are used to ensure 128 multipliers for each modulus. The models with approximately 76M parameters were trained on 4 million sequences over 2 million steps, using a batch size of 128 on a single H100 GPU for 85.47 hours. Because LCGs are typically defined for moduli that are powers of prime numbers, the model is tested on moduli that are powers of the primes 2, 3, 5, and 7. The test set consists of 512 unseen a values and 64 unseen c values selected via the Hull-Dobell theorem for each test modulus.

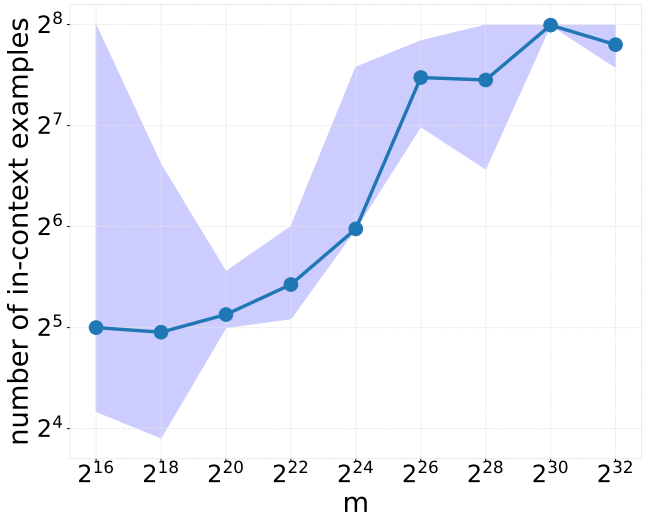
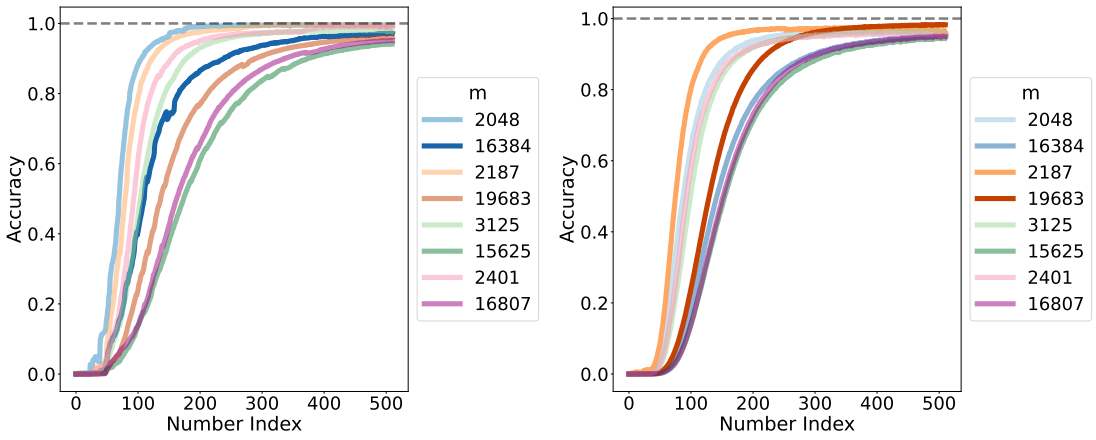


Figure E.22: Median number of in-context sequence elements required to achieve 100% test accuracy across five runs. The shaded region represents the min-max range.

Test performance is influenced by the tokenization base, exhibiting a bias toward moduli that share the same base as the tokenization method. For instance, in Figure E.23 (a) when using a byte-level representation, the model achieves better performance on moduli $m_{\text{test}} = 2^k$ compared to $m_{\text{test}} = 3^k$, 5^k, or 7^k. As contrasted with Figure E.23 (b) where the tokenization base is $243 = 3^5$, the model performs better on moduli $m = 3^k$. This behavior is likely due to the property of LCGs, where for moduli that are powers of a prime b, the lowest k-th digit exhibits a period of b^k. Tokenization in such a base highlights this periodic structure, making it more apparent and easier for the model to leverage during training and prediction.



(a) Tokenization base = $256 = 2^8$

(b) Tokenization base = $243 = 3^5$

Figure E.23: Test accuracy vs Number index. In (a), the moduli 2048 and 16384 (blue curves) have the same root 2 as the tokenization base 256. The model performs better on these two moduli. In (b), the moduli 2178 and 19683 (orange curves) have the same root 3 as the tokenization base 243. The model performs better on these two moduli.

Appendix F: Distributed Neural Architectures

F.1 Experimental details

In this section, we list all the details that the reader might find helpful.

Hardware and Precision. All experiments were conducted using a codebase based on the PyTorch [69] and xFormers [190] libraries. Training was performed in mixed precision using BFloat16 on NVIDIA A100 80GB SXM4 GPUs and NVIDIA H200 SXM5 GPUs.

Memory/Throughput. In all cases, our implementation runs slower and consumes more memory compared to its dense counterpart. This overhead arises primarily from the unoptimized handling of dynamically changing sequence lengths across batches and internal steps/routers when multiple modules are activated. Due to the highly dynamic nature of the model, it is infeasible to precompute and cache attention masks for all possible sequence length combinations. As a result, we currently generate attention masks on the fly at each step and for each module.

We also find that increasing the batch size can improve GPU utilization. However, this comes with trade-offs: for the vision model, it degrades performance, while for the language model, memory becomes the limiting factor. Part of this memory bottleneck stems from the worst-case scenario, where the model must store intermediate results for all modules across all

steps and routers.

Nevertheless, we do not view this as a fundamental limitation. In our long-term vision of a fully distributed model, module locality will be introduced, which can alleviate this bottleneck and decouple memory constraints from architectural design. A thorough investigation of this direction is left for future work.

Effective top-k. To measure the diversity of routers' decisions, we introduce the concept of *effective top-k*, defined as the inverse of $\mathrm{IPR}_\alpha^{(s)}$ for a given router at step s.

The quantity $\mathrm{IPR}_\alpha^{(s)}$ is computed from the token counts $c_i^{(s)}$ as follows:

$$\mathrm{IPR}_\alpha^{(s)} = \frac{\sum_{i \in \star} (c_i^{(s)})^{2\alpha}}{\left(\sum_{i \in \star} (c_i^{(s)})^2 \right)^\alpha} . \tag{F.1}$$

By definition, $\mathrm{IPR}_\alpha^{(s)}$ lies in the range $[N_m^{-\alpha+1}, 1]$. It attains its maximum value when only one $c_i^{(s)}$ is non-zero, and its minimum when all $c^{(s)}i$ are equal at step s. The exponent α controls the sensitivity of $\mathrm{IPR}_\alpha^{(s)}$ to the distribution of the $c_i^{(s)}$ values. Throughout this work, we use $\alpha = 1.5$.

F.1.1 Vision models

Model Architecture For all models listed in Table F.1, we use image size 224×224 with a patch size of 16×16. We are not using `[CLS]` token, but instead use global average pooling before the final output.

Initialization We initialize all weights but the patch embedding using PyTorch truncated normal with 0 mean and 0.02 standard deviation, where the patch embedding was implemented with

Model	N_b	N_m	N_r	d_{embed}	d_{MLP}	N_{head}	Active Params	Params
ViT-small	-	12	-	384	1536	6	22M	22M
top-1 DNA	1	18	11	384	1536	6	22M	34M
top-2 DNA (25% skip)	1	24	11	256	1024	4	18M	18M
top-1 DNA-n	1	18(Attn) + 18(MLP)	22	384	1536	6	18M-26M	34M
top-2 DNA-n	1	24(Attn) + 24(MLP)	22	384	1536	6	38M-40M	44M

Table F.1: Hyperparameters of all vision model used in this paper, where only the last two lines are new compared to Table 7.1. Note that i) the number of active parameters column does not take model re-use into account; ii) the backbone for DNA-n models are full transformer blocks.

`nn.Conv2d` with the default initialization. No learnable biases used in any module.

Optimization We use AdamW optimizer with $\beta_1 = 0.9$ and $\beta_2 = 0.99$, $\epsilon = 1 \times 10^{-8}$. All models were trained for 300 epochs with a 10-epoch warmup, following a $\cos$-decay. The initial learning rate is set to $\eta_{\text{init}} = 1 \times 10^{-7}$ and final learning rate is set to $\eta_{\text{final}} = 1 \times 10^{-6}$.

Augmentation We use PyTorch's default choices for hyperparameters in data augmentations. For MixUp, we set $\alpha = 0.8$. We randomly select either MixUp or CutMix with equal probability for each batch. We apply ColorJitter with parameters: brightness=0.3, contrast=0.3, and saturation=0.3.

F.1.2 Language models

Model Architecture We use models with dimension of each head, i.e. $d_{\text{embed}}/N_{\text{head}}$ fixed as 64. All other details are listed in Table F.2.

Tokenizer We use the GPT-2 Tokenizer from tiktoken library. The vocabulary size is $50,257$.

Initialization We initialize all weights using PyTorch truncated normal with 0 mean and 0.02 standard deviation. No learnable biases used in any module.

Model	N_b	N_m	N_r	d_{embed}	d_{MLP}	N_{head}	Active Params	Params
GPT-2 medium	-	24	-	1024	4096	16	406M	406M
top-1 DNA	2	36	22	1024	4096	16	406M	583M
top-2 DNA	2	72	24	768	3072	12	433M	603M
top-1 DNA-n	2	36(Attn) + 36(MLP)	44	1024	4096	16	347 − 465M	583M
top-2 DNA-n	2	64(Attn) + 64(MLP)	44	768	3072	12	350 − 455M	546M
GPT-2 (30% shallower)	-	17	-	1024	4096	16	313M	313M
top-2 DNA (30% skip)	1	72	23	768	3072	12	412M	596M

Table F.2: Hyperparameters of DNA models and GPT-2 medium (no weight-tying). Note that for DNA models, we do not take module re-use into account while counting the number of active parameters.

Optimization We use AdamW optimizer with $\beta_1 = 0.9$ and $\beta_2 = 0.95$, $\epsilon = 1 \times 10^{-15}$. All models were trained for $40,000$ steps with a $1,000$-step warmup, following a constant period then linear decay to 0.1η for last 20% of training steps. The initial learning rate is set to $\eta_{\text{init}} = 1 \times 10^{-7}$ and final learning rate is set to $\eta_{\text{final}} = 1 \times 10^{-6}$.

Downstream Evaluation Downstream Evaluations were done using torchtune recipes [191] and lm-evaluation-harness library [192]. The dataset we used for evaluation are listed here: Arc-Easy [193], BoolQ [194], HellaSwag [195], LAMBADA OpenAI version [172], PIQA [196], RACE [197] and Wikitext-2 [198].

F.2 Module Usage and Load Balancing

We plot the module usage distribution for all DNA models used in the main text in Fig. F.1. We observe that, without load balancing, although certain modules bear a heavier load, completely dead modules are rare. This may be because we are not considering a setting with high sparsity.

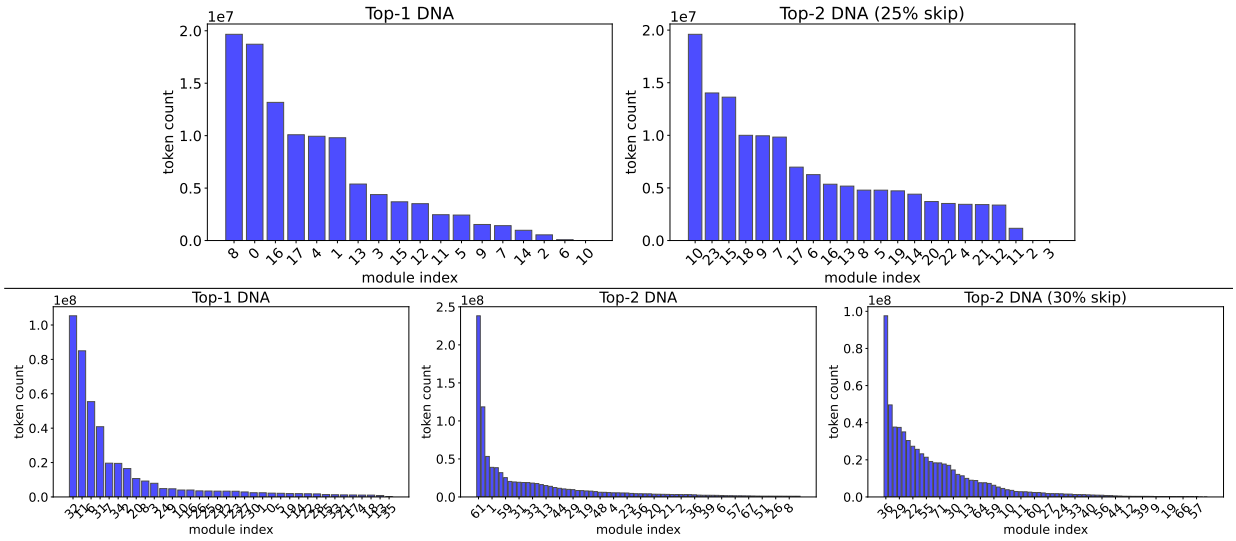


Figure F.1: Patch/Token counts for each module (exclude skipping, note that the module index does not start from 0 whenever we use skip.) for all DNA models we used in main text. **Top**: vision models. **Bottom**: language models.

F.3 Dreaming Visualizations

F.3.1 Experimental Details

Naive activation maximization can be done by straightforwardly maximizing any function of the internal (or external) variables of a network with respect to the input pixels. However, this tends to lead to poor results in particular high-frequency artifacts and images without any consistent global structure. Therefore we use several regularization methods, which serve to address these effects. These regularization methods have been drawn from several places in the literature. In particular, we follow Olah et al. [199] and their code for the parametrization and transformations and Ghiasi et al. [200] for the color shift and gaussian smoothing (additive noise) regularizations.

The first regularization method is to parametrize the image in Fourier space with a frequency-dependent coefficient, and to parametrize the color-channel dimension by the appro-

priate matrix to correct for the typical distribution of colors. Specifically, let $\theta_{\mu,c}$ be the input representation of the image in Fourier space with ω^μ corresponding to the 2d-frequency at index μ and c the color channel. We parametrize the image Θ as

$$\Theta = \text{sigmoid}\left[\text{IFFT2}\left(\sum_c W_{\omega_x^\mu \omega_y^\mu} A_{c'c} \theta_{\mu,c}\right)\right] \tag{F.2}$$

with

$$A = \begin{pmatrix} 0.26 & 0.09 & 0.02 \\ 0.27 & 0.00 & -0.05 \\ 0.27 & -0.09 & 0.03 \end{pmatrix} \text{ and } \quad W_{\omega_\mu} = \frac{1}{\max(\omega_x^\mu, \omega_y^\mu, 224^{-1})}. \tag{F.3}$$

Additionally we add regularization in three ways. First we randomly transform the image with the transform being re-drawn at every backward pass. The transformation is compsed of a random 6-pixel jitter, random rotation of up to 10 degrees, random scaling/shrinking by up to 10% followed by a random color shift and random per-pixel noise.

The color shift is given by the transformation $\Theta_c \to e^{\sigma_c}\Theta_c + \epsilon_c$ where $\sigma_c, \epsilon_c \sim \mathcal{N}(0,1)$ where the noise variables depend only on the color channel and not the position in the image. The random noise is per-pixel zero-mean, and has a linearly decaying variance, starting at 1 and ending at 0 by the end of the optimization procedure.

These procedures help regularize the high-frequency components by systematically penalizing them (unless they are robust to being moved around) as well as lowering the effective learning rate on them by placing a smaller coefficient on them. Additionally they allow global features to develop more easily because these features are more robust to noise and will therefore stabilize first, as the noise level decays through optimization.

Finally, to encourage a smoother image we explicitly add a regularizer, the total variation, which is the sum of the squares of the difference between neighboring pixels, in the horizontal, vertical, and diagonal directions. The coefficient we use for this regularization was $.01$.

To optimize the input image, we use the Adam optimizer with learning rate $.001$, and default momentum parameters. We run the optimization for 2048 steps, as mentioned with linearly decaying per-pixel noise, but with otherwise constant regularization parameters.

F.3.2 Further Dreaming Examples

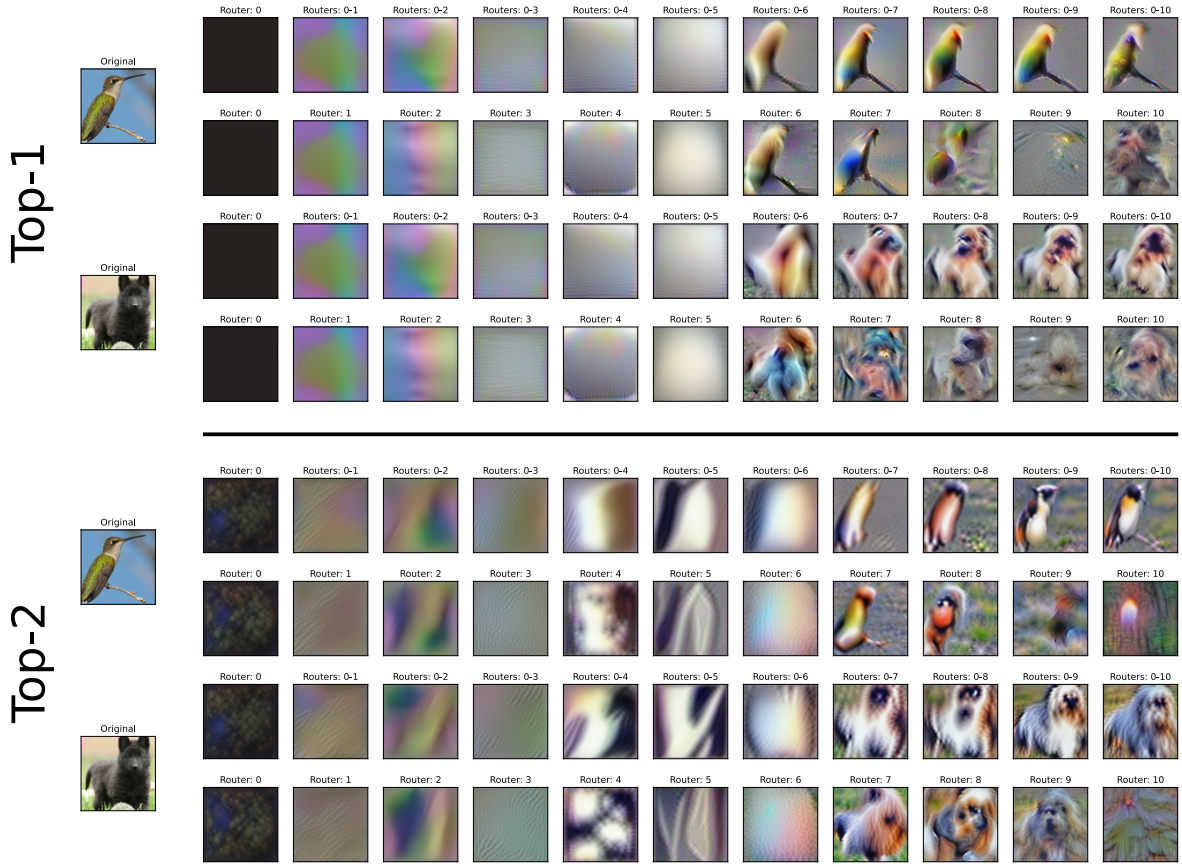


Figure F.2: Reconstruction of the hummingbird and welsh springer spaniel for the top-1 and top-2 models (some parts repeated from main text). Reconstructions are broken up into maximizing path weights from steps $0 - S$ (first and third rows) and maximizing decision weights for just step S (second and fourth rows). We see that the top-2 model exhibits interesting low level features before the sixth router, while the top-1 model does not. This is because the top-1 model has an almost input-independent selection of modules before that router. Additionally we see in both cases the same transition at router 7 where global features develop corresponding to the original image.



Figure F.3: Series of experiments where we attempt to recover the image from the paths of a group of tokens. The red boundary delineates the group of tokens whose decisions we maximized. Each recovery comes in two varieties: The upper rows have the context from the original image in the region surrounded by the red boundary, while the lower row of each example does not. The main finding is that the boundary patches contain non-local information about both the object and the background. When these patches are provided as context, the dreaming procedure generates the object and some background. The patch patterns also nicely illustrate the emergent sparse attention.

F.4 More on vision DNAs

F.4.1 Compute distribution over patches

In this subsection, we visualize the compute distribution and randomly selected patches with varying compute levels in Fig. F.4.

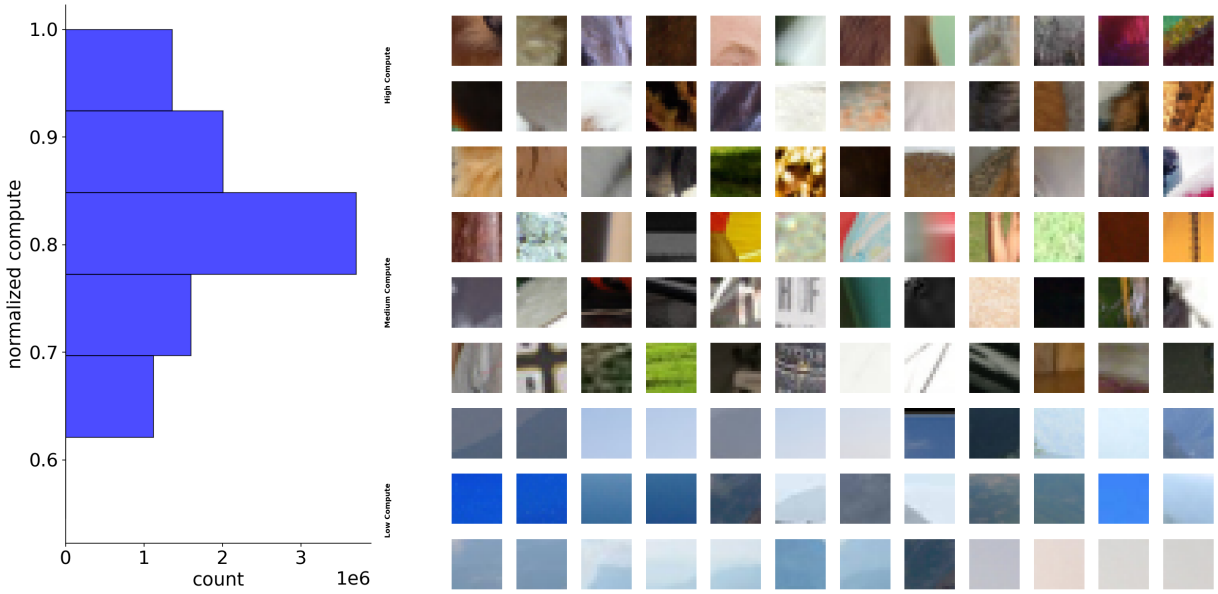


Figure F.4: **Left**: Distribution of computational cost on patches from the ImageNet validation set for the top-2 DNA (25% skip) model, which is the same model as the one in Fig. 7.8. **Right**: High compute patches generally contain more textures and boundaries, while low compute patches tend to appear visually flatter.

F.4.2 Random model paths

In this subsection, we demonstrate that a randomly initialized model can group patches based on only superficial similarities. The selected ribbons and corresponding patches are shown in Fig. F.5. Compared to Fig. 7.5, we find that the patches following the same path in a randomly initialized model share much greater visual similarities. We believe this effect arises from the

fact that our models are initialized at criticality, where the correlations between different inputs are preserved as they propagate through [28, 165, 201].

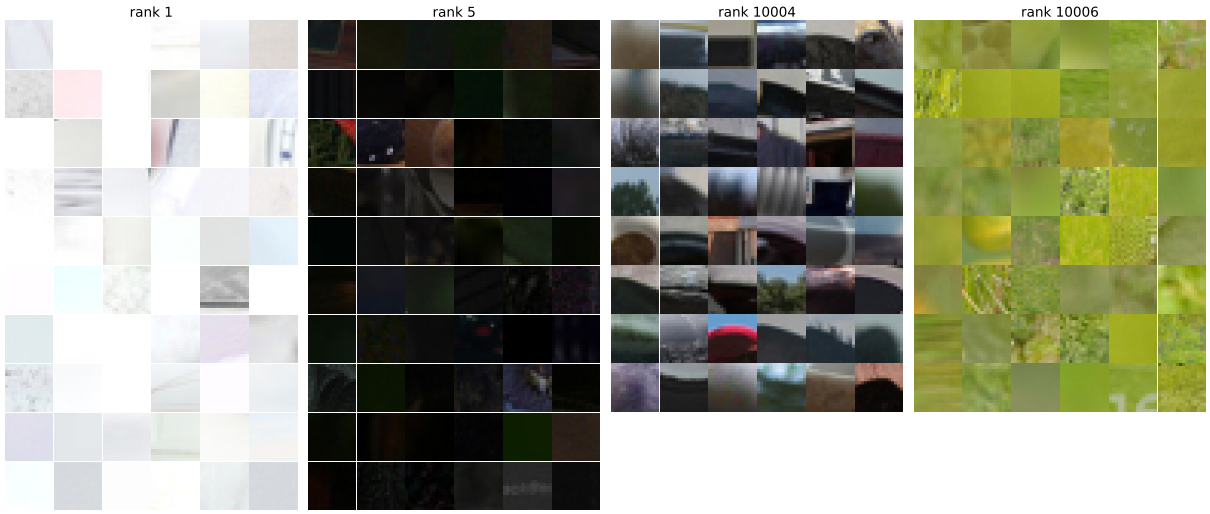


Figure F.5: Illustration of randomly selected patches processed through the same ribbon in a randomly initialized top-2 DNA model. Patches following the same ribbon display strong visual similarities, likely due to the critical initialization that preserves the similarities among different input patches.

F.5 More on language DNAs

F.5.1 Interpretability

Examples used in main text We used two paragraphs in the main text, where the first one is taken from `https://en.wikipedia.org/wiki/Neural_network_(machine_learning)` as it is a good test example; the second one is hand-crafted by putting in words with related concepts, together with intentional misspelling. The two examples:

Wiki *A neural network consists of connected units or nodes called artificial neurons, which loosely model the neurons in the brain. Artificial neuron models that mimic biological neurons more closely have also been recently investigated and shown to significantly improve performance. These are connected by edges, which model the synapses in the brain. Each artificial neuron receives signals from connected neurons, then processes them and sends a signal to other connected neurons. The 'signal' is a real number, and the output of each neuron is computed by some non-linear function of the sum of its inputs, called the activation function. The strength of the signal at each connection is determined by a weight, which adjusts during the learning process.*

Crafted *Have you had dinner? dinner? dinner? dinner? nothing? dinner? dinner? dinnar? dinner? dinner? dinner? dinner? dinner? lunch? dinner? dinner? dinner? dinner? dinner? breakfast $\sim$ dinner? dinner? dinner? Dinnar? lunch? lunch? lunch! launch! breakslow! This is an easy task.*

We present the complete token flows for these two examples using the Top-1 DNA model in Figs. F.6 and F.7. Note that the left panel of Fig. 7.11 is a zoomed-in version of those two

figures.

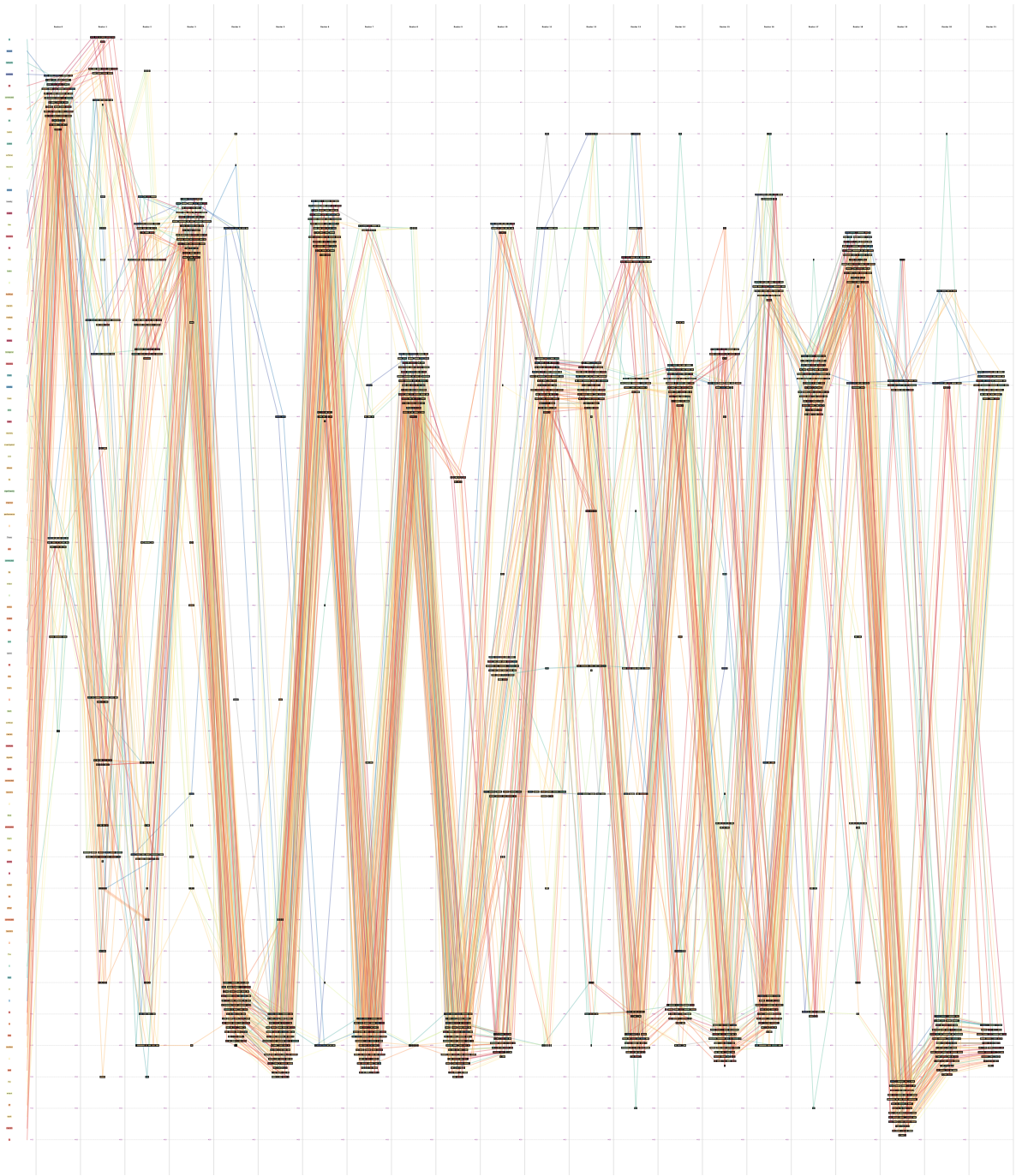


Figure F.6: Full trajectories through the top-1 DNA language model on Wiki example.

F.5.2 Compute distribution over tokens

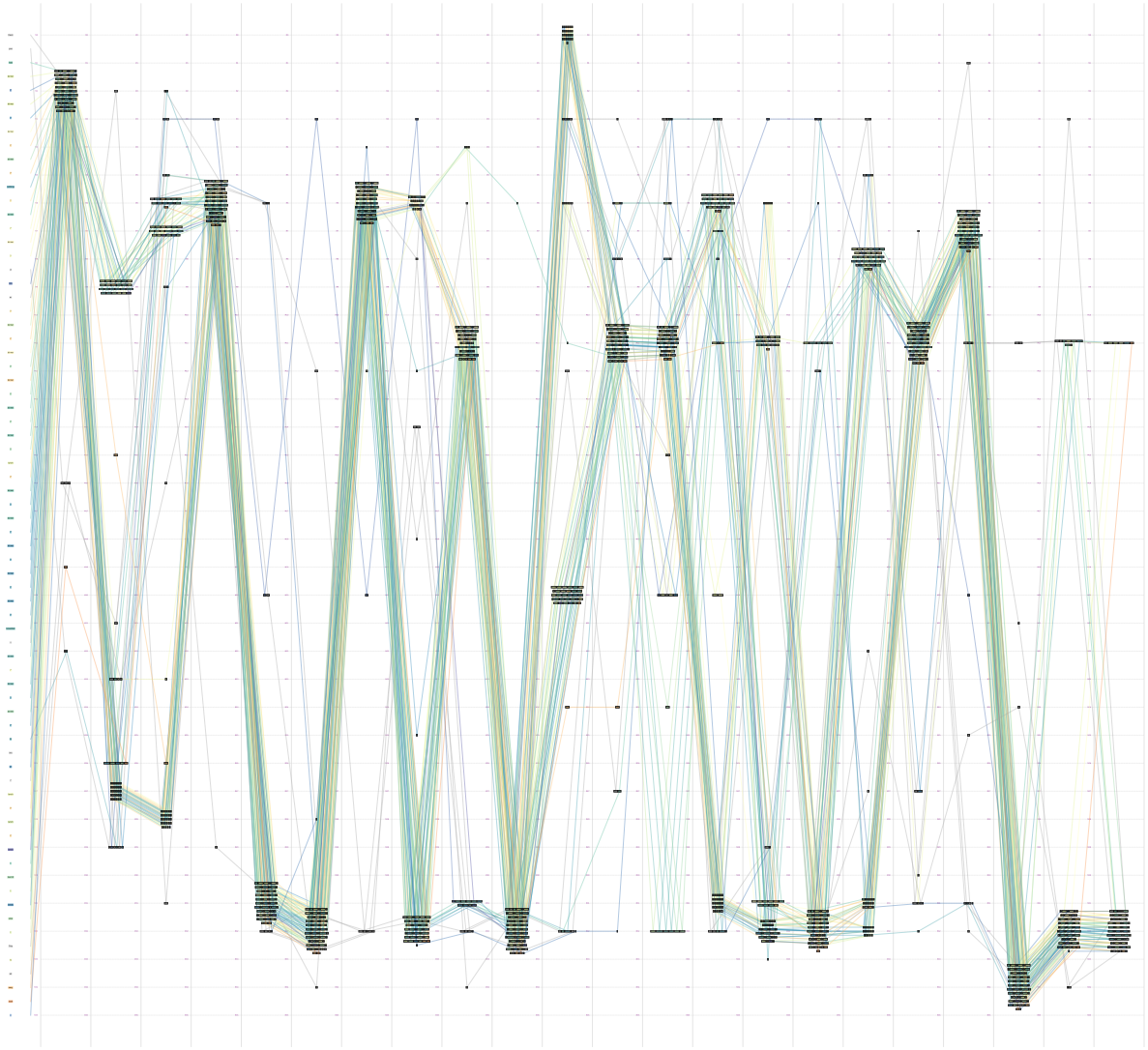


Figure F.7: Full trajectories through the top-1 DNA language model on artificial example.

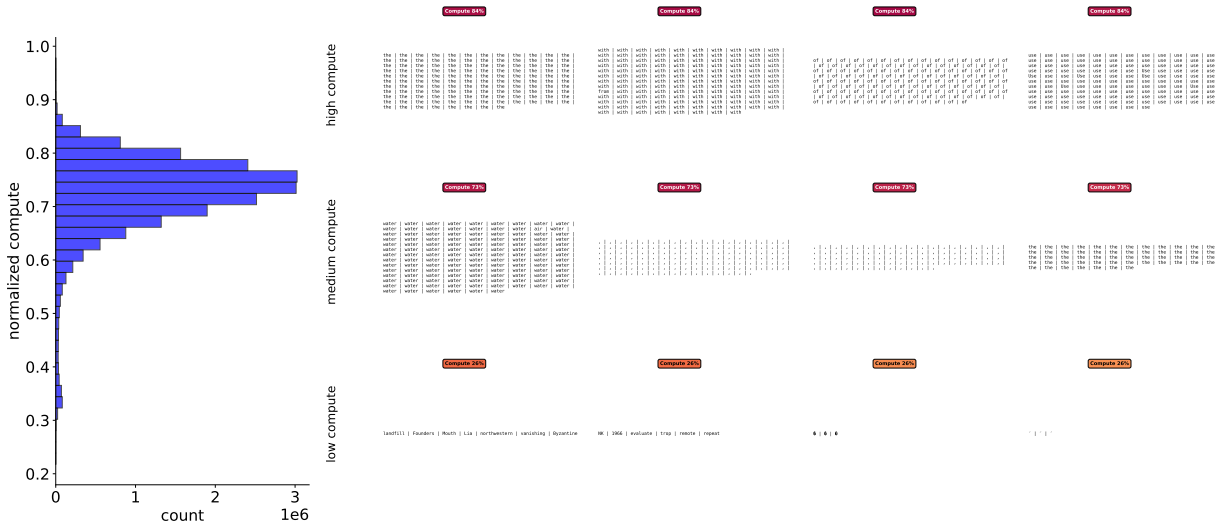


Figure F.8: **Left**: Compute distribution per token measured on the validation set of the FineWeb-Edu dataset for the top-2 DNA model with 30% skip. **Right**: Selected tokens that pass through ribbons with different compute. We find that tokens associated with very low-compute ribbons are usually particular objects, concepts, or symbols that do not play a crucial role in sequences.

F.6 Breaking Transformer Blocks

In this section, we delve deeper into architectural choices by breaking down Transformer blocks into their sub-blocks, specifically Pre-LN Attention blocks and Pre-LN MLP blocks.

F.6.1 Vision DNA

We present two DNA-n models (one top-1 and one top-2) that were trained on ImageNet. Most conclusions from the main text about paths and their interpretability still hold. However, we find that in both cases of $k = 1, 2$, the models tend to utilize more Attention blocks during the initial steps and more MLP blocks closer to the output. The patch flow patterns and path/ribbon distributions for the top-1 model are shown in Fig. F.9, with selected images and patches illustrating path specialization in Fig. F.10. Similar figures for the top-2 model are provided in Fig. F.11 and Fig. F.12, respectively. We also computed effective top-k at each step for both cases, as shown in Fig. F.13.

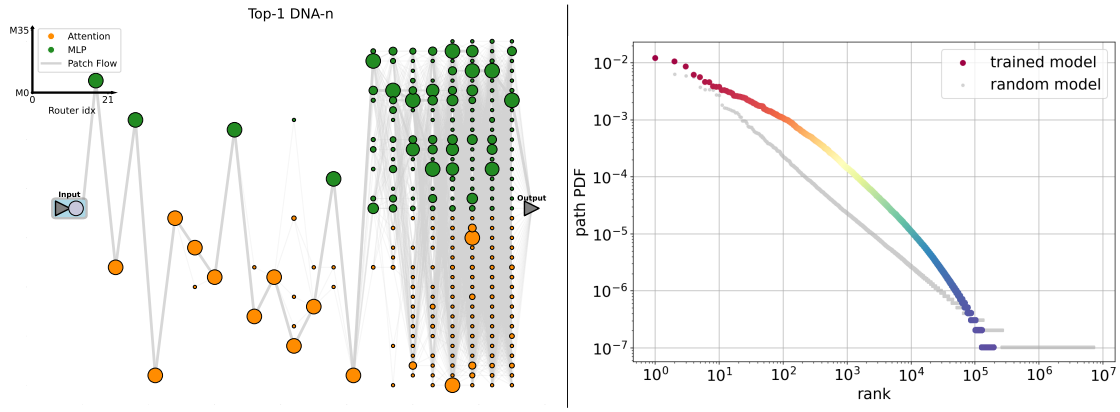


Figure F.9: A top-1 DNA-n model (76.4% validation accuracy), where the Transformer blocks are separated into Attention/MLP sub-blocks. **Left** Patch flows collected over ImageNet validation set; **Right** Path distribution.

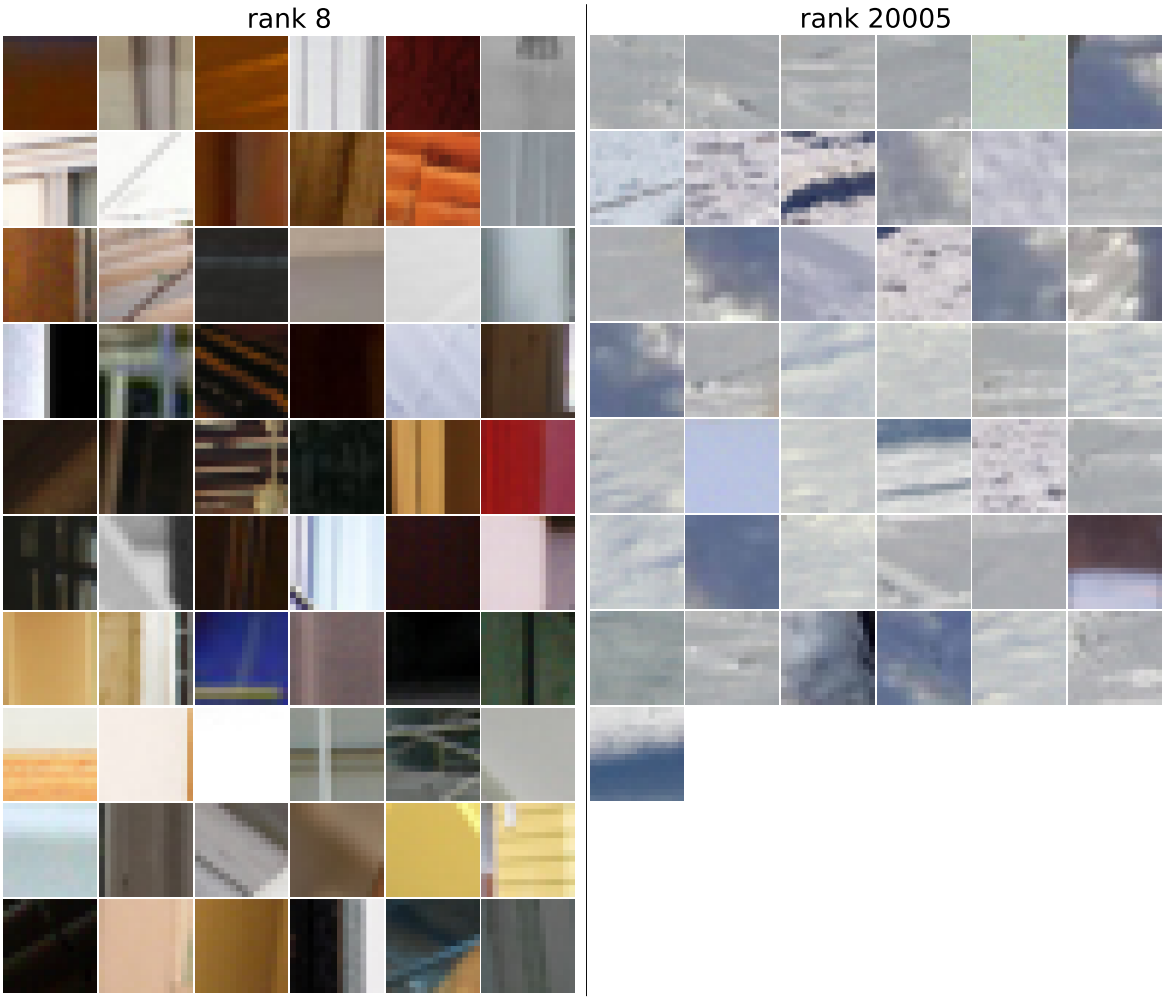


Figure F.10: Visualization patches that go through the same paths for the model shown in Fig. F.9. **Left**: A layout of single image goes through the model; **Left**: Patches that go through a low-rank path. **Right**: Patches that go through a high-rank path.

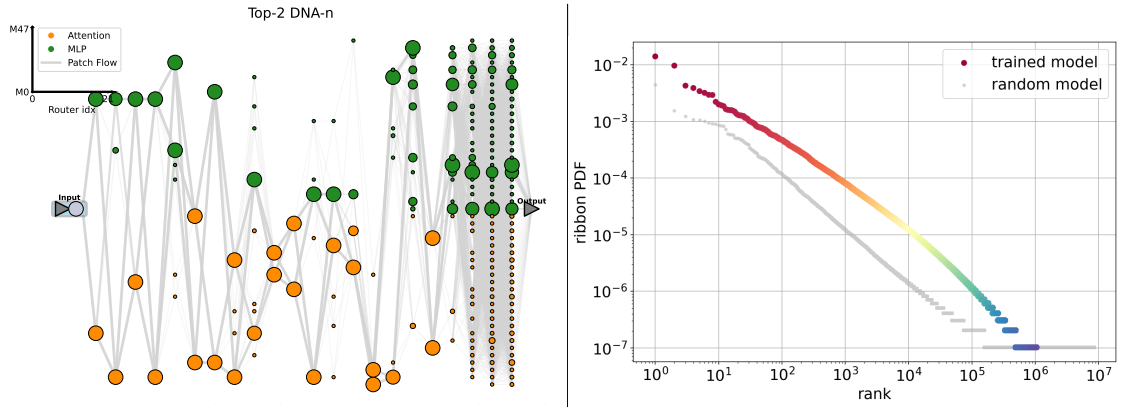


Figure F.11: A top-2 DNA-n model (78.9% validation accuracy), where the Transformer blocks are separated into Attention/MLP sub-blocks. **Left**: Patch flows collected over ImageNet validation set; **Right**: Ribbon distribution.

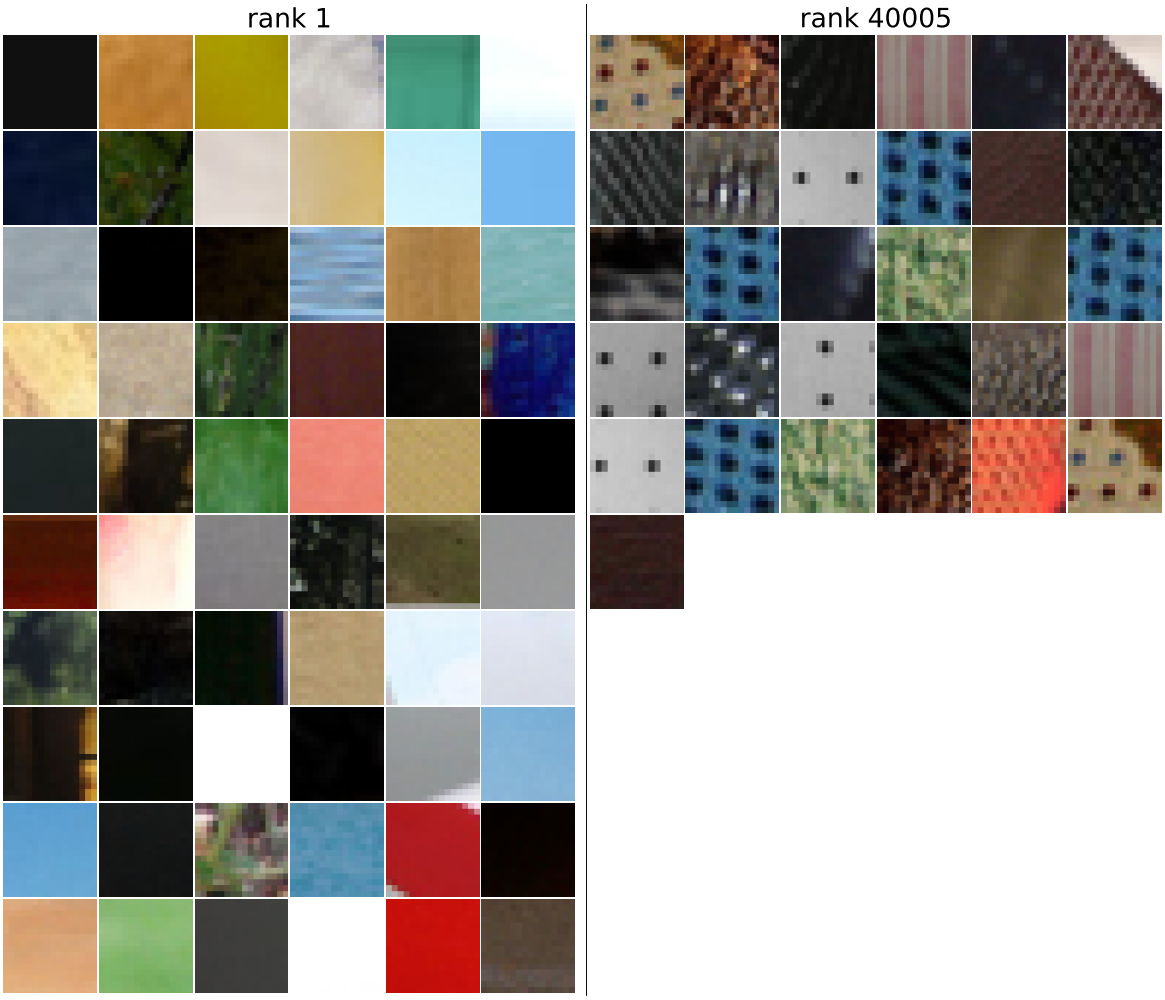


Figure F.12: Visualization patch ribbons for the model shown in Fig. F.11. **Left**: A layout of single image goes through the model; **Middle**: Patches that go through a low-rank ribbon; **Right**: Patches that go through a high-rank ribbon.

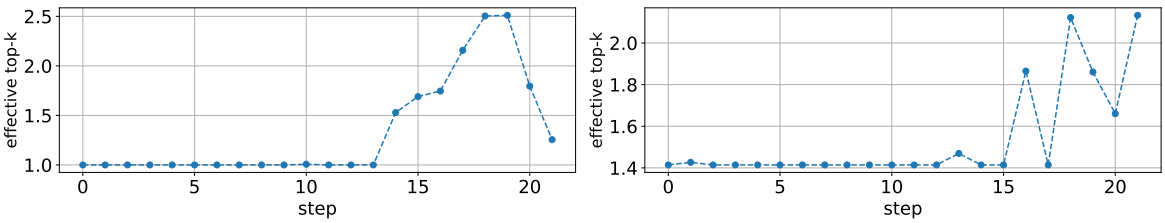


Figure F.13: Effective top-k for DNA-n models (vision), as a function of number of forward pass steps s. Early on the models are essentially dense, while later on they become more distributed. **Left**: top-1 DNA-n model. **Right**: top-2 DNA-n model.

F.6.2 Language DNA

Similarly, we trained GPT-2 Medium sized top-1 and top-2 models with separate Attention modules and MLP modules on FineWeb-Edu for 21B tokens. The token flow patterns and path/ribbon distributions are shown in Fig. F.14 and F.15, with the tokens that go through different paths/ribbons shown in Fig. F.16. We also plot effective top-k of router weights shown in Fig. F.17. Similar to the separated vision models, we find that language models also tend to use more Attention modules near the input and more MLP modules closer to the output.

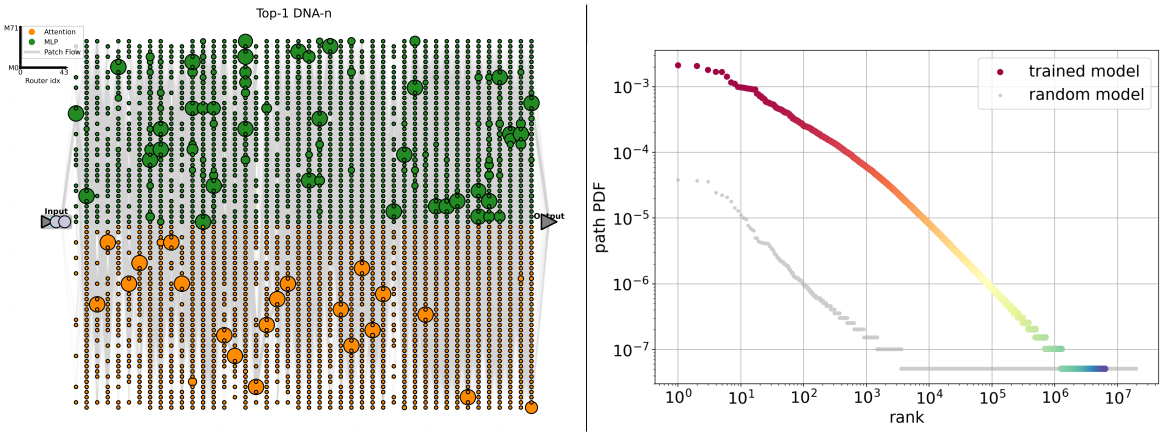


Figure F.14: A top-1 DNA-n model (final validation loss 2.708), where the Transformer blocks are separated into Attention/MLP sub-blocks. **Left** Token flows collected over FineWeb-Edu validation subset; **Right** Ribbon distribution.

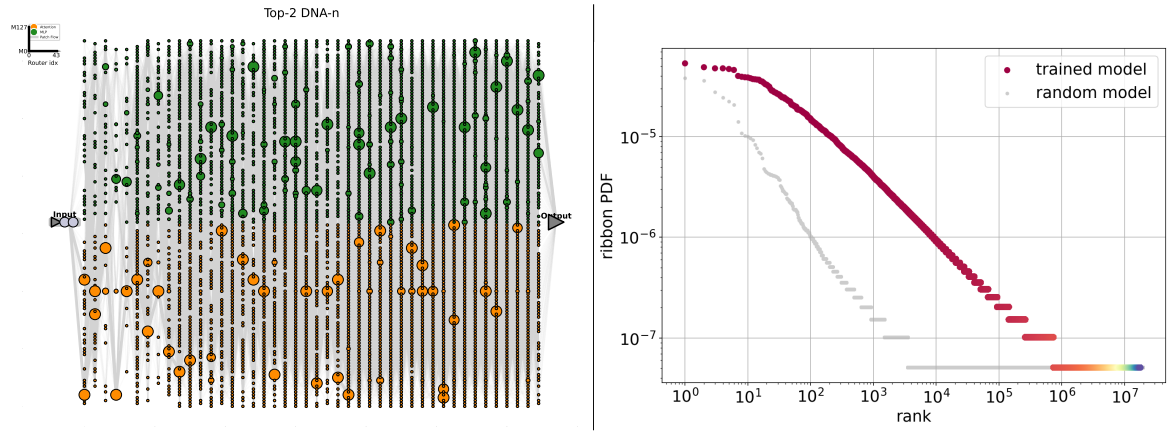


Figure F.15: A top-2 DNA-n model (final validation loss 2.688), where the Transformer blocks are separated into Attention/MLP sub-blocks. **Left**: Token flows collected over FineWeb-Edu validation subset; **Right**: Ribbon distribution.

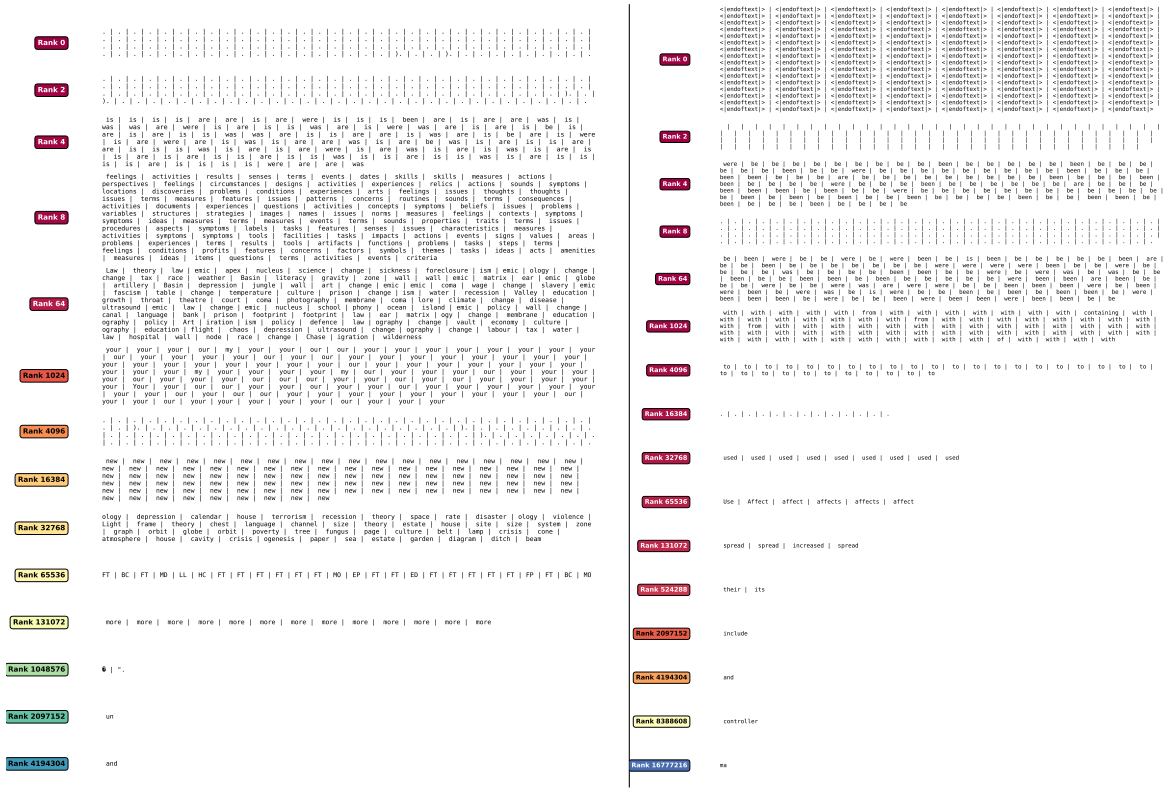


Figure F.16: Tokens that go through different rank paths/ribbons. **Left**: top-1 model; **Right**: top-2 model.

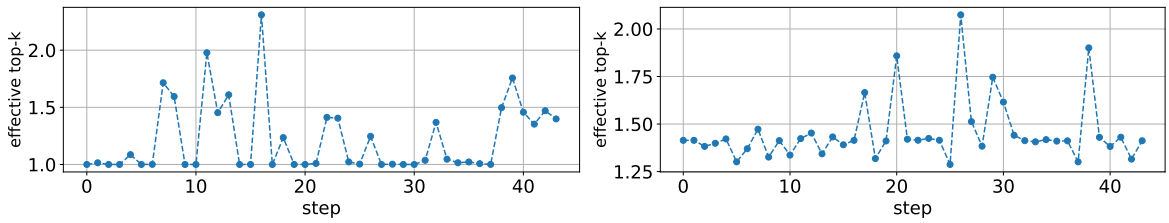
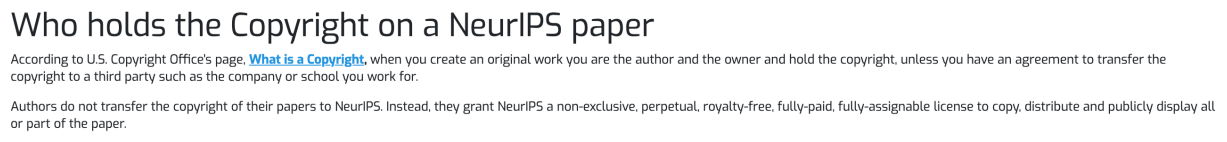


Figure F.17: Effective top-k for DNA-n models (language), as a function of number of forward pass steps s. The trend is very different from the vision models we have shown earlier. **Left**: top-1 DNA-n model. **Right**: top-2 DNA-n model.

Appendix G: Copyright Information

Portions of the work presented in this dissertation have been published, with joint authorship, in peer-reviewed conferences including NeurIPS, ICLR, and ICML [1, 2, 3, 6, 7, 8]. Verbal permission to include these materials in the dissertation has been obtained from all co-authors.

All the aforementioned conferences confirm that copyright to the published materials remains with the authors and was not transferred to the venues. Relevant links and supporting screenshots are provided below.

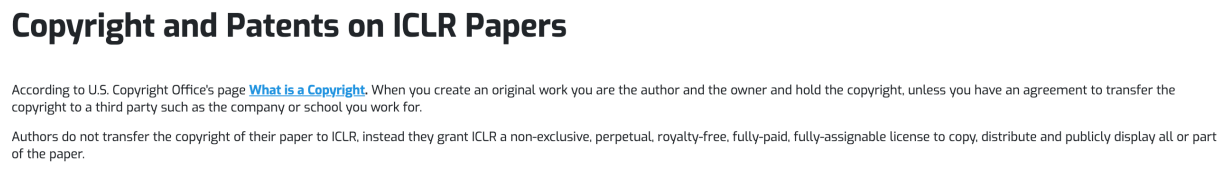


Who holds the Copyright on a NeurIPS paper

According to U.S. Copyright Office's page, **What is a Copyright**, when you create an original work you are the author and the owner and hold the copyright, unless you have an agreement to transfer the copyright to a third party such as the company or school you work for.

Authors do not transfer the copyright of their papers to NeurIPS. Instead, they grant NeurIPS a non-exclusive, perpetual, royalty-free, fully-paid, fully-assignable license to copy, distribute and publicly display all or part of the paper.

Figure G.1: Copyright information for NeurIPS: https://neurips.cc/FAQ/Copyright



Copyright and Patents on ICLR Papers

According to U.S. Copyright Office's page **What is a Copyright**. When you create an original work you are the author and the owner and hold the copyright, unless you have an agreement to transfer the copyright to a third party such as the company or school you work for.

Authors do not transfer the copyright of their paper to ICLR, instead they grant ICLR a non-exclusive, perpetual, royalty-free, fully-paid, fully-assignable license to copy, distribute and publicly display all or part of the paper.

Figure G.2: Copyright information for ICLR: https://iclr.cc/FAQ/Copyright

Who holds the Copyright on a ICML paper

According to U.S. Copyright Office's page **What is a Copyright.** When you create an original work you are the author and the owner and hold the copyright, unless you have an agreement to transfered the copyright to a third party such as the company or school you work for.

Authors do not tranfer the copyright of their paper to ICML, instead they grant ICML a non-exclusive, perpetual, royalty-free, fully-paid, fully-assignable license to copy, distribute and publicly display all or part of the paper

Figure G.3: Copyright information for ICML: https://icml.cc/FAQ/Copyright

Appendix H: AI Usage Disclosure

The author has used GenAI tools solely for checking grammatical errors or suggesting more accurate word choices.

The author has not used GenAI tools for directly generating content while writing this dissertation, nor for analyzing data within this work.

Bibliography

[1] Darshil Doshi, Tianyu He, and Andrey Gromov. Critical initialization of wide and deep neural networks using partial jacobians: General theory and applications. *Advances in Neural Information Processing Systems*, 36:40054–40095, 2023.

[2] Tianyu He, Darshil Doshi, and Andrey Gromov. Autoinit: Automatic initialization via jacobian tuning. *arXiv:2206.13568*, 2022.

[3] Dayal Singh Kalra, Tianyu He, and Maissam Barkeshli. Universal sharpness dynamics in neural network training: Fixed point analysis, edge of stability, and route to chaos. In *The Thirteenth International Conference on Learning Representations*, 2025.

[4] Darshil Doshi, Aritra Das, Tianyu He, and Andrey Gromov. To grok or not to grok: Disentangling generalization and memorization on corrupted algorithmic datasets. In *The Twelfth International Conference on Learning Representations*, 2024.

[5] Darshil Doshi, Tianyu He, Aritra Das, and Andrey Gromov. Grokking modular polynomials. *arXiv:2406.03495*, 2024.

[6] Tianyu He, Darshil Doshi, Aritra Das, and Andrey Gromov. Learning to grok: Emergence of in-context learning and skill composition in modular arithmetic tasks. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

[7] Tao Tao, Darshil Doshi, Dayal Singh Kalra, Tianyu He, and Maissam Barkeshli. (how) can transformers predict pseudo-random numbers? In *Forty-second International Conference on Machine Learning*, 2025.

[8] Aditya Cowsik, Tianyu He, and Andrey Gromov. Towards distributed neural architectures. *arXiv:2506.22389*, 2025.

[9] Darshil Doshi. *Science of Deep Learning: From Initialization to Emergent Structures*. Doctoral dissertation, University of Maryland, College Park, 2025.

[10] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[11] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv:1409.1556*, 2014.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *Computer Vision – ECCV 2016*, pages 630–645, 2016.

[15] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[16] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020.

[17] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv:2001.08361*, 2020.

[18] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Thomas Hennigan, Eric Noland, Katherine Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karén Simonyan, Erich Elsen, Oriol Vinyals, Jack Rae, and Laurent Sifre. An empirical analysis of compute-optimal large language model training. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 30016–30030. Curran Associates, Inc., 2022.

[19] Alexander Maloney, Daniel A. Roberts, and James Sully. A solvable model of neural scaling laws. *arXiv:2210.16859*, 2022.

[20] Utkarsh Sharma and Jared Kaplan. Scaling laws from the data manifold dimension. *Journal of Machine Learning Research*, 23(9):1–34, 2022. URL `http://jmlr.org/papers/v23/20-1111.html`.

[21] Yasaman Bahri, Ethan Dyer, Jared Kaplan, Jaehoon Lee, and Utkarsh Sharma. Explaining neural scaling laws. *Proceedings of the National Academy of Sciences*, 121(27): e2311878121, 2024.

[22] Blake Bordelon, Alexander Atanasov, and Cengiz Pehlevan. A dynamical model of neural scaling laws. *arXiv:2402.01092*, 2024.

[23] P. W. Anderson. More is different. *Science*, 177(4047):393–396, 1972.

[24] Andrey Gromov, Kushal Tirumala, Hassan Shapourian, Paolo Glorioso, and Dan Roberts. The unreasonable ineffectiveness of the deeper layers. In *The Thirteenth International Conference on Learning Representations*, 2025.

[25] Róbert Csordás, Christopher D Manning, and Christopher Potts. Do language models use their depth efficiently? *39th Conference on Neural Information Processing Systems*, 2025.

[26] Radford M Neal. Priors for infinite networks. In *Bayesian Learning for Neural Networks*, pages 29–53. Springer, 1996.

[27] Ben Poole, Subhaneil Lahiri, Maithra Raghu, Jascha Sohl-Dickstein, and Surya Ganguli. Exponential expressivity in deep neural networks through transient chaos. *Advances in neural information processing systems*, 29:3360–3368, 2016.

[28] Samuel S Schoenholz, Justin Gilmer, Surya Ganguli, and Jascha Sohl-Dickstein. Deep information propagation. In *International Conference on Learning Representations*, 2017.

[29] Alethea Power, Yuri Burda, Harri Edwards, Igor Babuschkin, and Vedant Misra. Grokking: Generalization beyond overfitting on small algorithmic datasets. *arXiv:2201.02177*, 2022.

[30] Andrey Gromov. Grokking modular arithmetic. *arXiv:2301.02679*, 2023. arXiv 2301.02679.

[31] Sanjeev Arora and Anirudh Goyal. A theory for emergence of complex skills in language models. *arXiv:2307.15936*, 2023.

[32] Jaehoon Lee, Yasaman Bahri, Roman Novak, Samuel S Schoenholz, Jeffrey Pennington, and Jascha Sohl-Dickstein. Deep neural networks as gaussian processes. In *International Conference on Learning Representations*, 2018.

[33] Lutz Molgedey, J Schuchhardt, and Heinz G Schuster. Suppressing chaos in neural networks by noise. *Physical review letters*, 69(26):3717, 1992.

[34] Jiri Hron, Yasaman Bahri, Jascha Sohl-Dickstein, and Roman Novak. Infinite attention: Nngp and ntk for deep attention networks. *37th International Conference on Machine Learning*, 2020.

[35] Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. *arXiv:1806.07572*, 2018.

[36] Jaehoon Lee, Lechao Xiao, Samuel Schoenholz, Yasaman Bahri, Roman Novak, Jascha Sohl-Dickstein, and Jeffrey Pennington. Wide neural networks of any depth evolve as linear models under gradient descent. *Advances in neural information processing systems*, 32:8572–8583, 2019.

[37] Roman Novak, Lechao Xiao, Yasaman Bahri, Jaehoon Lee, Greg Yang, Jiri Hron, Daniel A Abolafia, Jeffrey Pennington, and Jascha Sohl-dickstein. Bayesian deep convolutional networks with many channels are gaussian processes. In *International Conference on Learning Representations*, 2018.

[38] Lechao Xiao, Yasaman Bahri, Jascha Sohl-Dickstein, Samuel Schoenholz, and Jeffrey Pennington. Dynamical isometry and a mean field theory of cnns: How to train 10,000-layer vanilla convolutional neural networks. In *International Conference on Machine Learning*, pages 5393–5402. PMLR, 2018.

[39] Greg Yang, Jeffrey Pennington, Vinay Rao, Jascha Sohl-Dickstein, and Samuel S Schoenholz. A mean field theory of batch normalization. In *International Conference on Learning Representations*, 2018.

[40] Alexander G de G Matthews, Jiri Hron, Mark Rowland, Richard E Turner, and Zoubin Ghahramani. Gaussian process behaviour in wide deep neural networks. In *International Conference on Learning Representations*, 2018.

[41] Adrià Garriga-Alonso, Carl Edward Rasmussen, and Laurence Aitchison. Deep convolutional networks as shallow gaussian processes. In *International Conference on Learning Representations*, 2019.

[42] Ethan Dyer and Guy Gur-Ari. Asymptotics of wide networks from feynman diagrams. In *International Conference on Learning Representations*, 2019.

[43] Sanjeev Arora, Simon S Du, Zhiyuan Li, Ruslan Salakhutdinov, Ruosong Wang, and Dingli Yu. Harnessing the power of infinitely wide deep nets on small-data tasks. In *International Conference on Learning Representations*, 2019.

[44] Sanjeev Arora, Simon S Du, Wei Hu, Zhiyuan Li, Ruslan Salakhutdinov, and Ruosong Wang. On exact computation with an infinitely wide neural net. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, pages 8141–8150, 2019.

[45] Greg Yang. Wide feedforward or recurrent neural networks of any architecture are gaussian processes. In *33rd Conference on Neural Information Processing Systems*, 2019.

[46] Greg Yang. Scaling limits of wide neural networks with weight sharing: Gaussian process behavior, gradient independence, and neural tangent kernel derivation. *arXiv:1902.04760*, 2019.

[47] Zeyuan Allen-Zhu, Yuanzhi Li, and Zhao Song. A convergence theory for deep learning via over-parameterization. In *International Conference on Machine Learning*, pages 242–252. PMLR, 2019.

[48] Jiri Hron, Yasaman Bahri, Jascha Sohl-Dickstein, and Roman Novak. Infinite attention: Nngp and ntk for deep attention networks. In *International Conference on Machine Learning*, pages 4376–4386. PMLR, 2020.

[49] Anders Andreassen and Ethan Dyer. Asymptotics of wide convolutional neural networks. *arXiv:2008.08675*, 2020.

[50] Aitor Lewkowycz and Guy Gur-Ari. On the training dynamics of deep networks with l_2 regularization. In *Advances in Neural Information Processing Systems*, volume 33, pages 4790–4799. Curran Associates, Inc., 2020.

[51] Kyle Aitken and Guy Gur-Ari. On the asymptotics of wide networks with polynomial activations. In *International Conference on Learning Representations*, 2020.

[52] Mario Geiger, Arthur Jacot, Stefano Spigler, Franck Gabriel, Levent Sagun, Stéphane d'Ascoli, Giulio Biroli, Clément Hongler, and Matthieu Wyart. Scaling description of generalization with number of parameters in deep learning. *Journal of Statistical Mechanics: Theory and Experiment*, 2020(2):023401, 2020.

[53] Sho Yaida. Non-gaussian processes and neural networks at finite widths. In *Mathematical and Scientific Machine Learning*, pages 165–192. PMLR, 2020.

[54] Vaishaal Shankar, Alex Fang, Wenshuo Guo, Sara Fridovich-Keil, Jonathan Ragan-Kelley, Ludwig Schmidt, and Benjamin Recht. Neural kernels without tangents. In *International Conference on Machine Learning*, pages 8614–8623. PMLR, 2020.

[55] Jaehoon Lee, Samuel S Schoenholz, Jeffrey Pennington, Ben Adlam, Lechao Xiao, Roman Novak, and Jascha Sohl-Dickstein. Finite versus infinite neural networks: an empirical study. In *34th Conference on Neural Information Processing Systems*, 2020.

[56] Daniel A. Roberts, Sho Yaida, and Boris Hanin. *The Principles of Deep Learning Theory: An Effective Theory Approach to Understanding Neural Networks*. Cambridge University Press, 2022.

[57] Greg Yang and Edward J Hu. Tensor programs iv: Feature learning in infinite-width neural networks. In *International Conference on Machine Learning*, pages 11727–11737. PMLR, 2021.

[58] Russell Tsuchida, Tim Pearce, Chris van der Heide, Fred Roosta, and Marcus Gallagher. Avoiding kernel fixed points: Computing with elu and gelu infinite networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 9967–9977, 2021.

[59] James Martens, Andy Ballard, Guillaume Desjardins, Grzegorz Swirszcz, Valentin Dalibard, Jascha Sohl-Dickstein, and Samuel S Schoenholz. Rapid training of deep neural networks without skip connections or normalization layers using deep kernel shaping. *arXiv:2110.01765*, 2021.

[60] Rainer Engelken, Fred Wolf, and L. F. Abbott. Lyapunov spectra of chaotic recurrent neural networks. *Phys. Rev. Res.*, 5:043044, Oct 2023.

[61] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[62] Jingjing Xu, Xu Sun, Zhiyuan Zhang, Guangxiang Zhao, and Junyang Lin. Understanding and improving layer normalization. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[63] Yuxin Wu and Kaiming He. Group normalization. In *Proceedings of the European conference on computer vision (ECCV)*, pages 3–19, 2018.

[64] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16×16 words: Transformers for image recognition at scale. *International Conference on Learning Representations*, 2021.

[65] Greg Yang and Samuel S Schoenholz. Mean field residual networks: On the edge of chaos. *31st Conference on Neural Information Processing Systems*, 2017.

[66] Ilya Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Andreas Steiner, Daniel Keysers, Jakob Uszkoreit, Mario Lucic, and Alexey Dosovitskiy. Mlp-mixer: An all-mlp architecture for vision. *35th Conference on Neural Information Processing Systems*, 2021.

[67] Judy Hoffman, Daniel A. Roberts, and Sho Yaida. Robust learning with jacobian regularization. *arXiv:1908.02729*, 2019.

[68] Hugo Touvron, Piotr Bojanowski, Mathilde Caron, Matthieu Cord, Alaaeldin El-Nouby, Edouard Grave, Gautier Izacard, Armand Joulin, Gabriel Synnaeve, Jakob Verbeek, and Hervé Jégou. Resmlp: Feedforward networks for image classification with data-efficient training. *arXiv:2105.03404*, 2021.

[69] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[70] Alex Krizhevsky et al. Learning multiple layers of features from tiny images. 2009. URL `https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf`.

[71] Jeremy Cohen, Simran Kaur, Yuanzhi Li, J Zico Kolter, and Ameet Talwalkar. Gradient descent on neural networks typically occurs at the edge of stability. In *International Conference on Learning Representations*, 2021.

[72] Dayal Singh Kalra and Maissam Barkeshli. Phase diagram of early training dynamics in deep neural networks: effect of the learning rate, depth, and width. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

[73] Aitor Lewkowycz, Yasaman Bahri, Ethan Dyer, Jascha Sohl-Dickstein, and Guy Gur-Ari. The large learning rate phase of deep learning: the catapult mechanism. *ArXiv*, 2003.02218, 2020.

[74] Edward Ott. *Chaos in Dynamical Systems*. Cambridge University Press, 2 edition, 2002. doi: 10.1017/CBO9780511803260.

[75] Jascha Sohl-Dickstein, Roman Novak, Samuel S. Schoenholz, and Jaehoon Lee. On the infinite width limit of neural networks with a standard parameterization. *arXiv*, 2001.07301, 2020.

[76] Arthur Jacot, Franck Gabriel, and Clement Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[77] Greg Yang and Edward J. Hu. Tensor programs iv: Feature learning in infinite-width neural networks. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 11727–11737. PMLR, 18–24 Jul 2021.

[78] Stanislaw Jastrzebski, Maciej Szymczak, Stanislav Fort, Devansh Arpit, Jacek Tabor, Kyunghyun Cho*, and Krzysztof Geras*. The break-even point on optimization trajectories of deep neural networks. In *International Conference on Learning Representations*, 2020.

[79] Libin Zhu, Chaoyue Liu, Adityanarayanan Radhakrishnan, and Mikhail Belkin. Quadratic models for understanding neural network dynamics. *ArXiv*, 2205.11787, 2022.

[80] Atish Agarwala, Fabian Pedregosa, and Jeffrey Pennington. A second-order regression model shows edge of stability behavior. In *OPT 2022: Optimization for Machine Learning (NeurIPS 2022 Workshop)*, 2022.

[81] Lei Chen and Joan Bruna. Beyond the edge of stability via two-step gradient updates. *arXiv*, 2206.04172, 2023.

[82] ChaoKunin Ma, Lie Wu, and Lexing Ying. Beyond the quadratic approximation: The multiscale structure of neural network loss landscapes. *Journal of Machine Learning*, 1 (3):247–267, 2022. ISSN 2790-2048.

[83] Thomas D. Rogers and David C. Whitley. Chaos in the cubic mapping. *Mathematical Modelling*, 4(1):9–25, 1983. ISSN 0270-0255.

[84] Xuxing Chen, Krishnakumar Balasubramanian, Promit Ghosal, and Bhavya Agrawalla. From stability to chaos: Analyzing gradient descent dynamics in quadratic regression. 2310.01687, 2023.

[85] Zixuan Wang, Zhouzi Li, and Jian Li. Analyzing sharpness along GD trajectory: Progressive sharpening and edge of stability. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022.

[86] Sanjeev Arora, Zhiyuan Li, and Abhishek Panigrahi. Understanding gradient descent on the edge of stability in deep learning. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 948–1024. PMLR, 17–23 Jul 2022.

[87] Alex Damian, Eshaan Nichani, and Jason D. Lee. Self-stabilization: The implicit bias of gradient descent at the edge of stability. In *The Eleventh International Conference on Learning Representations*, 2023.

[88] Mihaela Rosca, Yan Wu, Chongli Qin, and Benoit Dherin. On a continuous time model of gradient descent dynamics and instability in deep learning. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856.

[89] Xingyu Zhu, Zixuan Wang, Xiang Wang, Mo Zhou, and Rong Ge. Understanding edge-of-stability training dynamics with a minimalist example. In *The Eleventh International Conference on Learning Representations*, 2023.

[90] Jingfeng Wu, Vladimir Braverman, and Jason D. Lee. Implicit bias of gradient descent for logistic regression at the edge of stability. *arXiv*, 2305.11788, 2023.

[91] Kwangjun Ahn, Sébastien Bubeck, Sinho Chewi, Yin Tat Lee, Felipe Suarez, and Yi Zhang. Learning threshold neurons via the "edge of stability". *ArXiv*, abs/2212.07469, 2022.

[92] Itai Kreisler, Mor Shpigel Nacson, Daniel Soudry, and Yair Carmon. Gradient descent monotonically decreases the sharpness of gradient flow solutions in scalar networks and beyond. *ArXiv*, 2305.13064, 2023.

[93] Minhak Song and Chulhee Yun. Trajectory alignment: Understanding the edge of stability phenomenon via bifurcation theory. *arXiv*, 2307.04204, 2023.

[94] Yuqing Wang, Zhenghao Xu, Tuo Zhao, and Molei Tao. Good regularity creates large learning rate implicit biases: edge of stability, balancing, and catapult. *arXiv*, 2310.17087, 2023.

[95] Lorenzo Noci, Alexandru Meterez, Thomas Hofmann, and Antonio Orvieto. Why do learning rates transfer? reconciling optimization and scaling limits for deep learning. *arXiv:2402.17457*, 2024.

[96] Robert A. Heinlein. *Stranger in a Strange Land*. G. P. Putnam's Sons, New York, 1961. ISBN 978-0-441-79034-8.

[97] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.

[98] Steven M. Girvin and Kun Yang. *Modern Condensed Matter Physics*. Cambridge University Press, 2019. doi: 10.1017/9781316480649.

[99] Romualdo Pastor-Satorras and Claudio Castellano. Distinct types of eigenvector localization in networks. *Scientific Reports*, 6(1):18847, 2016. doi: 10.1038/srep18847.

[100] Hwanjun Song, Minseok Kim, Dongmin Park, Yooju Shin, and Jae-Gil Lee. Learning from noisy labels with deep neural networks: A survey. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–19, 2022.

[101] Ziming Liu, Ouail Kitouni, Niklas S Nolte, Eric Michaud, Max Tegmark, and Mike Williams. Towards understanding grokking: An effective theory of representation learning. *Advances in Neural Information Processing Systems*, 35:34651–34663, 2022.

[102] Ziming Liu, Eric J Michaud, and Max Tegmark. Omnigrok: Grokking beyond algorithmic data. In *The Eleventh International Conference on Learning Representations*, 2023.

[103] Neel Nanda, Lawrence Chan, Tom Lieberum, Jess Smith, and Jacob Steinhardt. Progress measures for grokking via mechanistic interpretability. In *The Eleventh International Conference on Learning Representations*, 2023.

[104] Ziqian Zhong, Ziming Liu, Max Tegmark, and Jacob Andreas. The clock and the pizza: Two stories in mechanistic explanation of neural networks. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

[105] Boaz Barak, Benjamin L. Edelman, Surbhi Goel, Sham M. Kakade, eran malach, and Cyril Zhang. Hidden progress in deep learning: SGD learns parities near the computational limit. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022.

[106] Vikrant Varma, Rohin Shah, Zachary Kenton, János Kramár, and Ramana Kumar. Explaining grokking through circuit efficiency. *arXiv:2309.02390*, 2023.

[107] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. In *International Conference on Learning Representations*, 2017.

[108] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv:1207.0580*, 2012.

[109] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[110] Aritra Ghosh, Himanshu Kumar, and P Shanti Sastry. Robust loss functions under label noise for deep neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 31, 2017.

[111] Angelos Katharopoulos and Francois Fleuret. Not all samples are created equal: Deep learning with importance sampling. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 2525–2534. PMLR, 10–15 Jul 2018.

[112] Mansheej Paul, Surya Ganguli, and Gintare Karolina Dziugaite. Deep learning on a data diet: Finding important examples early in training. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 20596–20607. Curran Associates, Inc., 2021.

[113] Tong Xiao, Tian Xia, Yi Yang, Chang Huang, and Xiaogang Wang. Learning from massive noisy labeled data for image classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[114] Bo Han, Jiangchao Yao, Gang Niu, Mingyuan Zhou, Ivor Tsang, Ya Zhang, and Masashi Sugiyama. Masking: A new perspective of noisy supervision. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31, 2018.

[115] Jiangchao Yao, Jiajie Wang, Ivor W. Tsang, Ya Zhang, Jun Sun, Chengqi Zhang, and Rui Zhang. Deep learning from noisy image labels with quality embedding. *IEEE Transactions on Image Processing*, 28(4):1909–1922, 2019. doi: 10.1109/TIP.2018.2877939.

[116] Ari S. Morcos, David G. T. Barrett, Neil C. Rabinowitz, and Matthew Botvinick. On the importance of single directions for generalization. *The Sixth International Conference on Learning Representations*, 2018.

[117] Brian R. Bartoldson, Ari S. Morcos, Adrian Barbu, and Gordon Erlebacher. The generalization-stability tradeoff in neural network pruning. *arXiv:1906.03728*, 2020.

[118] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed H. Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy Liang, Jeff Dean, and William Fedus. Emergent abilities of large language models. *Transactions on Machine Learning Research*, 2022. ISSN 2835-8856.

[119] Deep Ganguli, Danny Hernandez, Liane Lovitt, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Nova Dassarma, Dawn Drain, Nelson Elhage, et al. Predictability and surprise in large generative models. In *Proceedings of the 2022 ACM Conference on Fairness, Accountability, and Transparency*, pages 1747–1764, 2022.

[120] Kwangjun Ahn, Xiang Cheng, Hadi Daneshmand, and Suvrit Sra. Transformers learn to implement preconditioned gradient descent for in-context learning. In A. Oh, T. Neumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 45614–45650. Curran Associates, Inc., 2023.

[121] Allan Raventos, Mansheej Paul, Feng Chen, and Surya Ganguli. Pretraining task diversity and the emergence of non-bayesian in-context learning for regression. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

[122] Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *arXiv:2104.09864*, 2023.

[123] Aaditya K Singh, Stephanie C.Y. Chan, Ted Moskovitz, Erin Grant, Andrew M Saxe, and Felix Hill. The transient nature of emergent in-context learning in transformers. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

[124] Ekin Akyürek, Dale Schuurmans, Jacob Andreas, Tengyu Ma, and Denny Zhou. What learning algorithm is in-context learning? investigations with linearmodels. 2023.

[125] Johannes von Oswald, Eyvind Niklasson, Ettore Randazzo, João Sacramento, Alexander Mordvintsev, Andrey Zhmoginov, and Max Vladymyrov. Transformers learn in-context by gradient descent. In *International Conference on Machine Learning*, 2023.

[126] Shivam Garg, Dimitris Tsipras, Percy Liang, and Gregory Valiant. What can transformers learn in-context? a case study of simple function classes. In *36th Conference on Neural Information Processing Systems*, 2023.

[127] Louis Kirsch, James Harrison, Jascha Sohl-Dickstein, and Luke Metz. General-purpose in-context learning by meta-learning transformers. *arXiv:2212.04458*, 2024.

[128] Ziqian Lin and Kangwook Lee. Dual operating modes of in-context learning. *arXiv:2402.18819*, 2024.

[129] Enric Boix-Adserà, Omid Saremi, Emmanuel Abbe, Samy Bengio, Etai Littwin, and Joshua M. Susskind. When can transformers reason with abstract symbols? In *The Twelfth International Conference on Learning Representations*, 2024.

[130] Tianyu Guo, Wei Hu, Song Mei, Huan Wang, Caiming Xiong, Silvio Savarese, and Yu Bai. How do transformers learn in-context beyond simple functions? a case study on learning with representations. In *The Twelfth International Conference on Learning Representations*, 2024.

[131] Roee Hendel, Mor Geva, and Amir Globerson. In-context learning creates task vectors. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, Singapore, December 2023. Association for Computational Linguistics.

[132] Sheng Liu, Haotian Ye, Lei Xing, and James Zou. In-context vectors: Making in context learning more effective and controllable through latent space steering. *arXiv:2311.06668*, 2024.

[133] Jiuxiang Gu, Chenyang Li, Yingyu Liang, Zhenmei Shi, Zhao Song, and Tianyi Zhou. Fourier circuits in neural networks: Unlocking the potential of large language models in mathematical reasoning and modular arithmetic. *arXiv:2402.09469*, 2024.

[134] Nelson Elhage, Neel Nanda, Catherine Olsson, Tom Henighan, Nicholas Joseph, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Nova DasSarma, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, Dario Amodei, Tom Brown, Jack

Clark, Jared Kaplan, Sam McCandlish, and Chris Olah. A mathematical framework for transformer circuits. *Transformer Circuits Thread*, 2021. https://transformer-circuits.pub/2021/framework/index.html.

[135] Catherine Olsson, Nelson Elhage, Neel Nanda, Nicholas Joseph, Nova DasSarma, Tom Henighan, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Scott Johnston, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, Dario Amodei, Tom Brown, Jack Clark, Jared Kaplan, Sam McCandlish, and Chris Olah. In-context learning and induction heads. *arXiv:2209.11895*, 2022.

[136] Eshaan Nichani, Alex Damian, and Jason D. Lee. How transformers learn causal structure with gradient descent. *arXiv:2402.14735*, 2024.

[137] W. E. Thomson. A modified congruence method of generating pseudo-random numbers. *The Computer Journal*, 1(2):83–83, 01 1958. ISSN 0010-4620. doi: 10.1093/comjnl/1.2.83.

[138] A. Rotenberg. A new pseudo-random number generator. *J. ACM*, 7(1):75–77, January 1960. ISSN 0004-5411.

[139] T. E. Hull and A. R. Dobell. Random number generators. *SIAM Review*, 4(3):230–254, 1962.

[140] Melissa E. O'Neill. Pcg : A family of simple fast space-efficient statistically good algorithms for random number generation. 2014. URL `https://api.semanticscholar.org/CorpusID:3489282`.

[141] Ofir Press and Lior Wolf. Using the output embedding to improve language models. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 157–163, 2017.

[142] Sean McLeish, Arpit Bansal, Alex Stein, Neel Jain, John Kirchenbauer, Brian R. Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, Jonas Geiping, Avi Schwarzschild, and Tom Goldstein. Transformers can do arithmetic with the right embeddings. *arXiv:2405.17399*, 2024.

[143] Catherine Olsson, Nelson Elhage, Neel Nanda, Nicholas Joseph, Nova DasSarma, Tom Henighan, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, et al. In-context learning and induction heads. *arXiv:2209.11895*, 2022.

[144] Xingwu Chen and Difan Zou. What can transformer learn with varying depth? case studies on sequence learning tasks. In *Forty-first International Conference on Machine Learning*, 2024.

[145] Donald E. Knuth. *The art of computer programming, volume 2 (3rd ed.): seminumerical algorithms*. Addison-Wesley Longman Publishing Co., Inc., USA, 1997. ISBN 0201896842.

[146] Harvey L. Garner. The residue number system. In *Papers Presented at the the March 3-5, 1959, Western Joint Computer Conference*, IRE-AIEE-ACM '59 (Western), page 146–153, New York, NY, USA, 1959. Association for Computing Machinery. ISBN 9781450378659. doi: 10.1145/1457838.1457864.

[147] Gavin McCracken, Gabriela Moisescu-Pareja, Vincent Letourneau, Doina Precup, and Jonathan Love. Uncovering a universal abstract algorithm for modular addition in neural networks. *arXiv:2505.18266*, 2025.

[148] Fred Zhang and Neel Nanda. Towards best practices of activation patching in language models: Metrics and methods. In *The Twelfth International Conference on Learning Representations*, 2024.

[149] Guy Steele and Sebastiano Vigna. Computationally easy, spectrally good multipliers for congruential pseudorandom number generators. *arXiv:2001.05304*, 2021.

[150] Lee Sharkey, Bilal Chughtai, Joshua Batson, Jack Lindsey, Jeff Wu, Lucius Bushnaq, Nicholas Goldowsky-Dill, Stefan Heimersheim, Alejandro Ortega, Joseph Bloom, Stella Biderman, Adria Garriga-Alonso, Arthur Conmy, Neel Nanda, Jessica Rumbelow, Martin Wattenberg, Nandi Schoots, Joseph Miller, Eric J. Michaud, Stephen Casper, Max Tegmark, William Saunders, David Bau, Eric Todd, Atticus Geiger, Mor Geva, Jesse Hoogland, Daniel Murfet, and Tom McGrath. Open problems in mechanistic interpretability. *arXiv:2501.16496*, 2025.

[151] Ronald L Rivest. Cryptography and machine learning. In *International Conference on the Theory and Application of Cryptology*, pages 427–439. Springer, 1991.

[152] Glauco Amigo, Liang Dong, and Robert J. Marks Ii. Forecasting pseudo random numbers using deep learning. In *2021 15th International Conference on Signal Processing and Communication Systems (ICSPCS)*, pages 1–7, 2021.

[153] Noam Chomsky. Three models for the description of language. *IRE Transactions on information theory*, 2(3):113–124, 1956.

[154] Grégoire Delétang, Anian Ruoss, Jordi Grau-Moya, Tim Genewein, Li Kevin Wenliang, Elliot Catt, Chris Cundy, Marcus Hutter, Shane Legg, Joel Veness, and Pedro A. Ortega. Neural networks and the chomsky hierarchy. *arXiv:2207.02098*, 2023.

[155] Zeyuan Allen-Zhu and Yuanzhi Li. Physics of language models: Part 1, learning hierarchical language structures. *arXiv:2305.13673*, 2024.

[156] Francesco Cagnetta, Leonardo Petrini, Umberto M Tomasini, Alessandro Favero, and Matthieu Wyart. How deep neural networks learn compositional data: The random hierarchy model. *Physical Review X*, 14(3):031001, 2024.

[157] Francesco Cagnetta and Matthieu Wyart. Towards a theory of how the structure of language is acquired by deep neural networks. *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

[158] Remi Lam, Alvaro Sanchez-Gonzalez, Matthew Willson, Peter Wirnsberger, Meire Fortunato, Ferran Alet, Suman Ravuri, Timo Ewalds, Zach Eaton-Rosen, Weihua Hu, et al. Learning skillful medium-range global weather forecasting. *Science*, 382(6677):1416–1421, 2023.

[159] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv:2303.08774*, 2023.

[160] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv:2407.21783*, 2024.

[161] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv:1308.3432*, 2013.

[162] David Raposo, Sam Ritter, Blake Richards, Timothy Lillicrap, Peter Conway Humphreys, and Adam Santoro. Mixture-of-depths: Dynamically allocating compute in transformer-based language models. *arXiv:2404.02258*, 2024.

[163] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[164] Marvin Minsky. *Society of mind*. Simon and Schuster, 1986.

[165] Daniel A Roberts, Sho Yaida, and Boris Hanin. *The principles of deep learning theory*, volume 46. Cambridge University Press Cambridge, MA, USA, 2022.

[166] Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv:2412.19437*, 2024.

[167] Ekin D. Cubuk, Barret Zoph, Dandelion Mané, Vijay Vasudevan, and Quoc V. Le. Autoaugment: Learning augmentation strategies from data. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 113–123, 2019.

[168] Zhun Zhong, Liang Zheng, Guoliang Kang, Shaozi Li, and Yi Yang. Random erasing data augmentation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 13001–13008, 2020.

[169] Hongyi Zhang, Moustapha Cisse, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *International Conference on Learning Representations*, 2018.

[170] Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, and Youngjoon Yoo. Cutmix: Regularization strategy to train strong classifiers with localizable features. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6023–6032, 2019.

[171] Carlos Riquelme, Joan Puigcerver, Basil Mustafa, Maxim Neumann, Rodolphe Jenatton, André Susano Pinto, Daniel Keysers, and Neil Houlsby. Scaling vision with sparse mixture of experts. *Advances in Neural Information Processing Systems*, 34:8583–8595, 2021.

[172] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. Technical report, OpenAI, 2019. URL `https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf`.

[173] Anton Lozhkov, Loubna Ben Allal, Leandro von Werra, and Thomas Wolf. Fineweb-edu: the finest collection of educational content. 2024. doi: 10.57967/hf/2497. URL `https://huggingface.co/datasets/HuggingFaceFW/fineweb-edu`.

[174] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

[175] Yang You, Jing Li, Sashank Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. Large batch optimization for deep learning: Training bert in 76 minutes. In *International Conference on Learning Representations*, 2020.

[176] Christopher Williams. Computing with infinite networks. In M. C. Mozer, M. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems*, volume 9. MIT Press, 1997.

[177] Roman Novak, Lechao Xiao, Jiri Hron, Jaehoon Lee, Alexander A Alemi, Jascha Sohl-Dickstein, and Samuel S Schoenholz. Neural tangents: Fast and easy infinite neural networks in python. In *International Conference on Learning Representations*, 2019.

[178] Roman Novak, Lechao Xiao, Jiri Hron, Jaehoon Lee, Alexander A. Alemi, Jascha Sohl-Dickstein, and Samuel S. Schoenholz. Neural tangents: Fast and easy infinite neural networks in python. In *International Conference on Learning Representations*, 2020.

[179] Lechao Xiao, Jeffrey Pennington, and Sam Schoenholz. Disentangling trainability and generalization in deep learning. 2020.

[180] Lingkai Kong and Molei Tao. Stochasticity of deterministic gradient descent: Large learning rate for multiscale objective function. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 2625–2638. Curran Associates, Inc., 2020.

[181] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[182] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. 2017.

[183] Alex Krizhevsky. Learning multiple layers of features from tiny images. 2009.

[184] Vaishaal Shankar, Alexander W. Fang, Wenshuo Guo, Sara Fridovich-Keil, Ludwig Schmidt, Jonathan Ragan-Kelley, and Benjamin Recht. Neural kernels without tangents. In *ICML*, 2020.

[185] Kaiming He, X. Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[186] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL `http://github.com/google/jax`.

[187] Jonathan Heek, Anselm Levskaya, Avital Oliver, Marvin Ritter, Bertrand Rondepierre, Andreas Steiner, and Marc van Zee. Flax: A neural network library and ecosystem for JAX, 2020. URL `http://github.com/google/flax`.

[188] Arthur Jacot, François Ged, Berfin Şimşek, Clément Hongler, and Franck Gabriel. Saddle-to-saddle dynamics in deep linear networks: Small initialization training, symmetry, and sparsity. *arXiv:2106.15933*, 2022.

[189] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019. URL `https://api.semanticscholar.org/CorpusID:160025533`.

[190] Benjamin Lefaudeux, Francisco Massa, Diana Liskovich, Wenhan Xiong, Vittorio Caggiano, Sean Naren, Min Xu, Jieru Hu, Marta Tintore, Susan Zhang, Patrick Labatut, Daniel Haziza, Luca Wehrstedt, Jeremy Reizenstein, and Grigory Sizov. xformers: A modular and hackable transformer modelling library. 2022.

[191] torchtune maintainers and contributors. torchtune: Pytorch's finetuning library, 2024. URL `https//github.com/pytorch/torchtune`.

[192] Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. A framework for few-shot language model evaluation. 12 2023.

[193] Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv:1803.05457v1*, 2018.

[194] Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. Boolq: Exploring the surprising difficulty of natural yes/no questions. In *NAACL*, 2019.

[195] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. HellaSwag: Can a machine really finish your sentence? In Anna Korhonen, David Traum, and Lluís Màrquez, editors, *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 2019.

[196] Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. Piqa: Reasoning about physical commonsense in natural language. In *Thirty-Fourth AAAI Conference on Artificial Intelligence*, 2020.

[197] Guokun Lai, Qizhe Xie, Hanxiao Liu, Yiming Yang, and Eduard Hovy. RACE: Large-scale ReAding comprehension dataset from examinations. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 785–794, Copenhagen, Denmark, September 2017. Association for Computational Linguistics.

[198] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *5th International Conference on Learning Representations*, 2017.

[199] Chris Olah, Alexander Mordvintsev, and Ludwig Schubert. Feature visualization. *Distill*, 2017. doi: 10.23915/distill.00007. https://distill.pub/2017/feature-visualization.

[200] Amin Ghiasi, Hamid Kazemi, Eitan Borgnia, Steven Reich, Manli Shu, Micah Goldblum, Andrew Gordon Wilson, and Tom Goldstein. What do vision transformers learn? a visual exploration. *arXiv:2212.06727*, 2022.

[201] Jaehoon Lee, Yasaman Bahri, Roman Novak, Samuel S Schoenholz, Jeffrey Pennington, and Jascha Sohl-Dickstein. Deep neural networks as gaussian processes. In *International Conference on Learning Representations*, 2018.