

ABSTRACT

Title of Dissertation: UNDERSTANDING AND ENRICHING THE
ALGORITHMIC REASONING CAPABILITIES
OF DEEP LEARNING MODELS

Jingling Li
Doctor of Philosophy, 2023

Dissertation Directed by: Professor John Dickerson
Department of Computer Science

Learning to reason is an essential step to achieving general intelligence. My research has been focusing on empowering deep learning models with the abilities to generalize efficiently, extrapolate to out-of-distribution data, learn under noisy labels, and make better sequential decisions — all of these require the models to have varying levels of reasoning capabilities. As the reasoning process can be described as a step-by-step algorithmic procedure, understanding and enriching the algorithmic reasoning capabilities has drawn increasing attention in deep learning communities. To bridge algorithms and neural networks, we propose a framework, algorithmic alignment, which connects neural networks with algorithms in a novel manner and advances our understanding of how these two fields can work together to solve complex reasoning tasks. Intuitively, the algorithmic alignment framework evaluates how well a neural network’s computation structure aligns with the algorithmic structure in a reasoning process. Utilizing algorithmic alignment, we are able to understand the limitations of machine learning models in the context of

reasoning (e.g., generalization, extrapolation, and robustness) and know where to make improvements. Beyond this framework, we also investigate how to make better sequential decisions, during which we introduce a class of efficient approaches, hindsight learning, which allows us to leverage the knowledge inside existing human-designed algorithms to make better sequential decisions under uncertainty.

First, why and when does a network structure generalize better than others, despite having equal expressive power? In studying this, we unify seemingly different reasoning tasks, such as intuitive physics, visual question answering, and shortest paths, via the lens of a powerful algorithmic paradigm, dynamic programming (DP). We then show that Graph Neural Network (GNNs)—a popular reasoning model—algorithmically align well with DP, which explains their empirical success on many reasoning tasks. At the same time, using this framework, we also suggest the limitations of GNNs and demonstrate how we can use algorithmic alignment to design a structured network given a particular task.

Second, what does a neural network learn outside the support of the training distribution? To answer this, we identify conditions under which MLPs and GNNs extrapolate well. We first quantify the observation that ReLU MLPs quickly converge to linear functions along any direction from the origin, implying that ReLU MLPs do not extrapolate most nonlinear functions. Yet, we prove that MLPs can successfully extrapolate a linear target function given sufficient training support. These results suggest a hypothesis on GNNs for which we provide theoretical and empirical evidence: the success of GNNs in extrapolating algorithmic tasks to new data (e.g., larger graphs or edge weights) relies on encoding task-specific non-linearities in the architecture or features.

Third, how does a neural network’s architecture impact its robustness to noisy labels? To

figure this out, we use the proposed framework connecting a network’s robustness to its architectural alignments with the target/noise functions. We hypothesize that a network is more robust to noisy labels if its architecture is more aligned with the target function than the noise, and we provide both theoretical and empirical evidence across various neural network architectures and different domains.

Last but not least, how can we design sample-efficient approaches for sequential decision-making problems? Many decision-making problems especially the ones in resource management have a bunch of human-designed heuristics that work well in practice. These heuristics could provide more guidance to RL agents than just the environmental reward. At the same time, we observe that in many resource management problems requiring sequential decision-making under uncertainty, the only uncertainty affecting the decision outcomes are exogenous variables outside the control of the decision-maker in the system, and we can use machine learning to learn from historical patterns of compute demands to help deal with these uncertainties. Therefore, we design a class of data-efficient algorithms termed Hindsight Learning (HL), where we utilize existing human-designed algorithms with historical data to better guide the RL agents.

UNDERSTANDING AND ENRICHING THE
ALGORITHMIC REASONING CAPABILITIES
OF DEEP LEARNING MODELS

by

Jingling Li

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
2023

Advisory Committee:

Dr. John Dickerson, Chair/Advisor
Dr. Mark Fuge, Dean's Representative
Dr. Jia-Bin Huang
Dr. Tianyi Zhou
Dr. Jimmy Ba
Dr. Petar

© Copyright by
Jinging Li
2023

Acknowledgments

I owe my gratitude to all the people who have supported me throughout my Ph.D. journey. This dissertation would not have been possible without all their help.

First and foremost, I would like to express my deepest appreciation to my advisor, Prof. John Dickerson, for his unwavering guidance, support, and encouragement throughout my research. John's expertise and insights have been invaluable in shaping my ideas and approach to my research. Beyond being my advisor, John has been a great mentor, role model, and friend. His guidance has helped me grow both personally and professionally. I have learned a lot from John's attitude toward life and research, and he has supported me through all the ups and downs. I believe that our relationship goes beyond advisor and student, as we are also lifelong friends.

I would also like to thank the members of my Ph.D. committee, Prof. Mark Fuge, Prof. Jia-Bin Huang, Prof. Tianyi Zhou, Prof. Jimmy Ba, and Dr. Petar Veličković. Their valuable time, feedback, and suggestions have helped me greatly in improving my research, dissertation, and presentation. I would also like to thank Prof. Aravind Srinivasan and Prof. Tom Goldstein for serving on the committee of my preliminary exam and providing insightful advice on my research.

Besides, I would like to express my deep appreciation to my internship mentors, Dr. Adith Swaminathan, Dr. Ching-An Cheng, Dr. Petar Veličković (also my preliminary and dissertation committee member, who has demonstrated great potentials of algorithmic alignment in many dif-

ferent tasks), Prof. Jimmy Ba (also my dissertation committee member, who has always inspired me throughout my research journey), and Prof. Taiji Suzuki. They provided me with generous guidance and support during my internships at DeepMind, Microsoft Research, Vector Institute, JP Morgan, RIKEN AIP, and the University of Tokyo. I am grateful to all my collaborators and managers from these programs for their valuable contributions and insights.

Additionally, I am also indebted to Prof. Dianna Xu, Prof. Steven Lindell, and Prof. Deepak Kumar. They taught me how to conduct great research and helped me start my research journey when I was an undergraduate student at Bryn Mawr College. Their encouragement and support have been my guiding lights along this path.

I met so many talented and bright people throughout my Ph.D., and it is hard to name them all. I really enjoyed our discussions and collaborations, and I am deeply grateful for all their contributions and inputs. I would also like to thank Dr. Qingyang Tan and Dr. Yanchao Sun for providing the LaTeX style files and formatting tips for writing this thesis.

Finally, words cannot express my heartfelt gratitude to my family and friends who have supported me throughout my Ph.D. journey with their love, encouragement, and patience. I am particularly grateful to my friends, Ziyang Yang, Xiaoyu Liu, Melissa Chen, Jiahui Liu, Ying Fan, and my family for their unwavering support and companionship during this journey. Lastly, thank you to my beloved cat, Da Miao, for bringing joy and relaxation to my life and work.

Table of Contents

Acknowledgements	ii
Table of Contents	iv
List of Tables	viii
List of Figures	ix
List of Abbreviations	xi
Chapter 1: Introduction	1
1.1 Theoretical Framework: Algorithmic Alignment	3
1.2 Graph Neural Networks (GNN)	4
1.3 Overview of Research	5
Chapter 2: Generalization: What Can Neural Networks Reason About?	7
2.1 Introduction	7
2.2 Preliminaries	10
2.3 Theoretical Framework: Algorithmic Alignment	12
2.3.1 Formalization of Algorithmic Alignment	14
2.3.2 Better Algorithmic Alignment Implies Better Generalization	17
2.4 Predicting What Neural Networks Can Reason About	18
2.4.1 Summary Statistics	19
2.4.2 Relational Argmax	20
2.4.3 Dynamic Programming	22
2.4.4 Designing Neural Networks with Algorithmic Alignment	25
2.5 Chapter Summary	26
Chapter 3: Extrapolation: How does Neural Networks Extrapolate?	28
3.1 Introduction	28
3.1.1 Background	31
3.2 Preliminaries	32
3.3 How Feedforward Neural Networks Extrapolate	33
3.3.1 Linear Extrapolation Behavior of ReLU MLPs	34
3.3.2 When ReLU MLPs Provably Extrapolate Well	36
3.3.3 MLPs with Other Activation Functions	38
3.4 How Graph Neural Networks Extrapolate	39

3.4.1	Hypothesis: Linear Algorithmic Alignment Helps Extrapolation	40
3.4.2	Theoretical and Empirical Support	43
3.5	Connections to Other Out-of-Distribution Settings	46
3.6	Chapter Summary	47
Chapter 4: Robustness: How does the Architecture Impact a Neural Network’s Robustness?		49
4.1	Introduction	49
4.2	Theoretical Framework	53
4.2.1	Problem Settings	53
4.2.2	Predictive Power in Representations	54
4.2.3	Formalization of Alignment	55
4.2.4	Better Alignment Implies Better Robustness (Better Predictive Power) . .	57
4.3	Experiments on Graph Neural Networks	58
4.3.1	Background: Graph Neural Networks	58
4.3.2	Additive Label Noise	59
4.3.3	Instance-Dependent Label Noise	62
4.4	Experiments on Vision Datasets	65
4.4.1	MLPs vs. CNN-based networks	65
4.4.2	Predictive Power in Models Trained with SOTA Methods	70
4.5	Chapter Summary	71
Chapter 5: Decision Making: Hindsight Learning for MDPs with Exogenous Inputs		73
5.1	Introduction	73
5.2	Related Work	76
5.3	Problem Setting and Definitions	78
5.3.1	MDPs with Exogenous Inputs (Exo-MDPs)	78
5.3.2	Value Decomposition in Exo-MDPs	79
5.3.3	Hindsight Planner	81
5.4	Using Hindsight In Exo-MDPs: An Example	82
5.5	Hindsight Learning	83
5.5.1	Imitating the “Bayes Selector”	84
5.6	Theoretical Guarantees	86
5.7	Experiments	89
5.7.1	Multi-Secretary Problems	90
5.7.2	Airline Revenue Management	91
5.7.3	VM Allocation	92
5.8	Chapter Summary	97
Chapter 6: Appendix: What Can Neural Networks Reason About?		98
6.1	Proofs and Additional Experimental Details	98
6.1.1	Proof of Proposition 2.3.1	98
6.1.2	Proof of Proposition 2.3.2	99
6.1.3	Proof of Theorem 2.3.5	101
6.1.4	Proof of Theorem 2.3.6	102

6.1.5	Proof of Corollary 2.3.7	105
6.1.6	Proof of Claim 2.4.1	106
6.2	Additional Experimental Details	106
6.2.1	Fantastic Treasure: Maximum Value Difference	106
6.2.2	Fantastic Treasure: Furthest Pair	107
6.2.3	Monster Trainer	108
6.2.4	Subset Sum	110
Chapter 7:	Appendix: How does Neural Networks Extrapolate?	112
7.1	Additional Experimental Details	112
7.1.1	Learning Simple Non-Linear Functions	115
7.1.2	R-squared for Out-of-distribution Directions	118
7.1.3	Learning Linear Functions	120
7.1.4	MLPs with cosine, quadratic, and tanh Activation	122
7.1.5	Max Degree	123
7.1.6	Shortest Path	126
7.1.7	N-Body Problem	129
7.2	Visualization and Additional Experimental Results	133
7.2.1	Visualization Results	133
7.2.2	Extra Experimental Results	137
7.3	Supplementary Theoretical Background	140
7.4	Proofs	144
7.4.1	Constrained optimization in NTK feature space.	151
7.4.2	Global optimum via the geometry of training data.	156
7.4.3	Proof of Theorem 3.3.4	159
7.4.4	Asymptotic convergence of data sets.	160
7.4.5	Asymptotic convergence of predictions.	163
7.4.6	Proof of Corollary 3.4.3	165
7.4.7	Proof of Theorem 3.4.4	165
7.4.8	Proof of Lemma 7.3.1	170
7.4.9	Proof of Lemma 7.3.2	172
Chapter 8:	Appendix: How does the Architecture Impact a Neural Network’s Robustness?	176
8.1	Additional Experimental Results	176
8.1.1	Predictive Power of Randomly Initialized Models	176
8.1.2	Additive Label Noise on Graph Algorithmic Datasets	177
8.1.3	Training with a Robust Loss Function	178
8.1.4	Comparing with Sophisticated Methods Using Clean Labels	179
8.2	Theoretical Results	184
Chapter 9:	Appendix: Hindsight Learning for MDPs with Exogenous Inputs	188
9.1	Table of Notation	188
9.2	Detailed Related Work	188
9.3	MDPs with Exogenous Inputs	194

9.3.1	Generality of MDPs with Exogenous Inputs	194
9.3.2	Examples of Exo-MDPs	198
9.4	Hindsight Planners	204
9.4.1	Inventory Control with Lead Times and Lost Sales	205
9.4.2	Online Stochastic Bin Packing	205
9.4.3	Online Secretary	206
9.4.4	Airline Revenue Management	207
9.4.5	Virtual Machine Allocation	207
9.5	Existing Approaches to MDPs in Exo-MDPs	210
9.5.1	Predict Then Optimize	210
9.5.2	Reinforcement Learning	211
9.6	Proofs of Main Results	212
9.7	Simulation Details	224
9.7.1	Online Bin-Packing	225
9.7.2	Multi-Secretary	226
9.7.3	Airline Revenue Management	227
9.7.4	Virtual Machine Allocation	228
9.7.5	Stylized Environment	229
9.7.6	Real-World VM Allocation	238

List of Tables

4.1	Test accuracies comparison on CIFAR-10/100	68
4.2	Test accuracies comparison on CIFAR-Easy	68
5.1	Performance of heuristics, π^+ , RL and HL algorithms on multi-secretary and ARM problems benchmarked against the optimal policy. Values are V^π , the performance of the compared policy evaluated using the Bellman equations, and error bars computed via a standard normal approximation averaging over the randomly sampled dataset. Since these are tabular problems, HINDSIGHT MAC and HINDSIGHT Q-DISTILLATION are identical, so we report the performance of both as HINDSIGHT MAC. Relative performance compared against V^{π^*} is shown in parenthesis.	89
5.2	Performance comparison on VM allocation	94
5.3	Average number of PMs saved by RL and HL policies across 5 clusters, calculated over 44 days and benchmarked against the production BestFit heuristic. $\star$ indicate significant improvement and $\circ$ indicate a significant decrease, over Best-Fit by Welch’s t -test.	96
8.1	Predictive power of random and trained GNNs on max degree task	176
8.2	Predictive power of random and trained GNNs on max node feature task	177
8.3	Test accuracy comparison on CIFAR-10 with uniform label noise	179
8.4	Test accuracy comparison on CIFAR-10 with flipped label noise	180
8.5	Test accuracy comparison on CIFAR-100 with uniform label noise	180
8.6	Test accuracy comparison on CIFAR-100 with flipped label noise	181
8.7	Test accuracy comparison on Clothing1M	181
8.8	Test accuracy comparison on WebVision and ImageNet	182
9.1	List of common notation for MDP and HL	189
9.2	Hindsight bias in online bin packing decreases as round increases	226
9.3	List of hyperparameters	237
9.4	Performance comparison on VM allocation	238
9.5	Measurement of optimization gap	241

List of Figures

2.1	Overview of reasoning tasks with increasingly complex structure	10
2.2	An illustration of algorithmic alignment	12
2.3	Test accuracies on reasoning tasks with increasingly complex structure	18
2.4	Sample efficiency with respect to algorithmic alignment	21
3.1	How ReLU MLPs extrapolate	30
3.2	How GNNs extrapolate	30
3.3	Conditions for ReLU MLPs to extrapolate well	35
3.4	Extrapolation performance of ReLU MLPs	35
3.5	Extrapolation performance of MLPs with other activation	39
3.6	Extrapolation for algorithmic tasks	43
3.7	Importance of the training graph structure	43
4.1	Max-sum GNN aligns well with the maximum degree task	60
4.2	PCA visualization of hidden representations learned with noisy labels	60
4.3	Well-aligned models can learn predictive representations under label noise	62
4.4	More aligned models learn more predictive representations under label noise	63
4.5	Illustration of our CIFAR-Easy dataset	66
4.6	Sample complexity of MLPs and CNNs on CIFAR-Easy	66
4.7	Experimental results on CIFAR-10/100 with uniform and flipped label noise	67
4.8	Experimental results on CIFAR-Easy with various types of label noise	67
5.1	Overview of different ML approaches on solving exo-MDPs	75
5.2	An Exo-MDP example for sailing in uncertain winds	82
7.1	Quadratic function	134
7.2	Cosine function	134
7.3	Cosine function	135
7.4	Square root function	135
7.5	L1 function	136
7.6	L1 function	136
7.7	Linear function	138
7.8	Density plot of the test errors in MAPE	138
7.9	Continuous vs. unlabeled node features	139
7.10	Additional results on maximum degree task	139
7.11	Experimental results of shortest path problems on random graphs	139
8.1	Additional experiments on non-zero mean label noise	177

8.2	Predictive power in representations trained with MAE loss	178
8.3	Predictive power grows as mutual information increases	179
9.1	Causal diagram for an Exo-MDP	194
9.2	Sample of a thirty-day roll-out from the Azure Public Dataset	228
9.3	Packing density for the Best Fit policy at one time-step	230
9.4	Moving average of the loss curves for various RL algorithms	240

List of Abbreviations

GNN	Graph Neural Network
DP	Dynamic Programming
MLP	Multilayer Perceptron
RL	Reinforcement Learning
MDP	Markov Decision Process
HL	Hindsight Learning

Chapter 1: Introduction

Parts of the introduction were adapted from work described in other chapters, so coauthors credited there also deserve some credit.

Having the ability to reason is an important milestone on the road to general intelligence. There are a few important criteria to mark different levels of reasoning capabilities: (a) being able to generalize sample efficiently; (b) being able to extrapolate to examples far from the training distribution; (c) being robust to incorrect information; and (d) being able to make more complex sequential decisions. For example, when learning arithmetics, humans do not learn by memorizing thousands of examples. Instead, we learn the rules of arithmetics, and once we learned, we can apply them to arbitrarily large numbers and correct calculation mistakes. Moreover, we can utilize all of these abilities to help us make better sequential decisions and solve more complex arithmetic tasks such as sudoku.

Recently, there have been many advances in building neural networks that can learn to reason. Reasoning spans a variety of tasks, for instance, visual and text-based question answering [Johnson et al. \(2017a\)](#); [Weston et al. \(2015\)](#); [Hu et al. \(2017a\)](#); [Fleuret et al. \(2011\)](#); [Antol et al. \(2015\)](#); [Guo et al. \(2022\)](#), intuitive physics, i.e., predicting the time evolution of physical objects [Battaglia et al. \(2016\)](#); [Watters et al. \(2017\)](#); [Fragkiadaki et al. \(2016\)](#); [Chang et al. \(2017a\)](#); [Sanchez-Gonzalez et al. \(2020\)](#), mathematical reasoning [Saxton et al. \(2019\)](#); [Chang et al. \(2019\)](#);

Lu et al. (2022a); Imani et al. (2023) and visual IQ tests Huang et al. (2023); Santoro et al. (2018); Zhang et al. (2019a). Behind these reasoning tasks, the underlying solutions can all be described as a step-by-step algorithmic procedure. Therefore, this dissertation starts from the algorithmic perspective to understand the essence of reasoning, building upon both theoretical and empirical evidence.

To bridge algorithms and neural networks, we propose a framework, algorithmic alignment, which connects neural networks with algorithms in a novel manner and advances our understanding of how these two fields can work together to solve complex reasoning tasks. Intuitively, the algorithmic alignment framework evaluates how well a neural network’s computation structure aligns with the algorithmic structure in a reasoning process. For example, if a neural network’s computation graph aligns well with a target algorithm, each component of the network only needs to learn *simple* steps to simulate the whole reasoning algorithm, which leads to better sample efficiency at learning. Utilizing algorithmic alignment, we are able to understand the limitations of machine learning models in the context of reasoning (e.g., generalization, extrapolation, and robustness) and know where to make improvements.

We first show important connections between *dynamic programming* (DP) Bellman (1966), a backbone algorithm that can solve a broad range of reasoning tasks, and a popular framework where many successful models in reasoning tasks follow — Graph Neural Network (GNN) framework Battaglia et al. (2016, 2018); Palm et al. (2018); Mrowca et al. (2018); Sanchez-Gonzalez et al. (2018); Janner et al. (2019) via algorithmic alignment. We prove that GNNs algorithmically align with DP, which also explains their empirical success in many reasoning tasks. Algorithmic alignment also provides a formalized way to analyze the structures in various deep learning models. It quantifies the relation between network and task structure and formally

derives the implications for learning. These theoretical ideas are the basis for characterizing what reasoning tasks a network can learn well.

To provide some background information, we include the definitions of algorithmic alignment and GNNs in section 1.1 and section 1.2, and they will be introduced again with a greater context in later chapters. The connections between algorithmic alignment and GNNs are also described formally in chapter 2.

1.1 Theoretical Framework: Algorithmic Alignment

We introduce algorithmic alignment under a PAC learning framework [Valiant \(1984\)](#). PAC learnability formalizes *simplicity* as sample complexity, i.e., the number of samples needed to ensure low test error with high probability. It refers to a learning algorithm $\mathcal{A}$ that, given training samples $\{x_i, y_i\}_{i=1}^M$, outputs a function $f = \mathcal{A}(\{x_i, y_i\}_{i=1}^M)$. The learning algorithm here is the neural network and its training method, e.g., gradient descent. A function is simple if it has low sample complexity.

Definition 1.1.1. (PAC learning and sample complexity). Fix an error parameter $\epsilon > 0$ and failure probability $\delta \in (0, 1)$. Suppose $\{x_i, y_i\}_{i=1}^M$ are i.i.d. samples from some distribution $\mathcal{D}$, and the data satisfies $y_i = g(x_i)$ for some underlying function g . Let $f = \mathcal{A}(\{x_i, y_i\}_{i=1}^M)$ be the function generated by a learning algorithm $\mathcal{A}$. Then g is (M, ϵ, δ) -learnable with $\mathcal{A}$ if

$$\mathbb{P}_{x \sim \mathcal{D}}[\|f(x) - g(x)\| \leq \epsilon] \geq 1 - \delta. \quad (1)$$

The sample complexity $\mathcal{C}_{\mathcal{A}}(g, \epsilon, \delta)$ is the minimum M so that g is (M, ϵ, δ) -learnable with $\mathcal{A}$.

With the PAC learning framework, we define a numeric measure of algorithmic alignment (Definition 2.3.4). Formally, a neural network aligns with an algorithm if it can simulate the algorithm via a limited number of modules, and each module is simple, i.e., has low sample complexity.

Definition 1.1.2. (Algorithmic alignment). *Let g be a reasoning function and $\mathcal{N}$ a neural network with n modules $\mathcal{N}_i$. The module functions $f_1, \dots, f_n$ generate g for $\mathcal{N}$ if, by replacing $\mathcal{N}_i$ with f_i , the network $\mathcal{N}$ simulates g . Then $\mathcal{N}$ (M, ϵ, δ) -algorithmically aligns with g if (1) $f_1, \dots, f_n$ generate g and (2) there are learning algorithms $\mathcal{A}_i$ for the $\mathcal{N}_i$'s such that $n \cdot \max_i C_{\mathcal{A}_i}(f_i, \epsilon, \delta) \leq M$.*

Good algorithmic alignment, i.e., small M , implies that all algorithm steps f_i to simulate the algorithm g are *easy to learn*. Therefore, the algorithm steps should not simulate complex programming constructs such as for-loops, whose sample complexity is large.

1.2 Graph Neural Networks (GNN)

GNNs are structured networks operating on graphs Battaglia et al. (2018); Scarselli et al. (2009a); Xu et al. (2019, 2018). Given a graph $G = (V, E, X)$, where X collects *node attributes* $x_v \in \mathbb{R}^d$ for all $v \in V$ and possibly *edge attributes* $x_{u,v} \in \mathbb{R}^{d'}$ for all $(u, v) \in E$, GNNs compute node representations via iterative message passing operations Xu et al. (2018); Gilmer et al. (2017). Initial node representations $h_v^{(0)} = x_v$, for all $v \in V$, are updated in iteration k by aggregating and processing the representations of the set $\mathcal{N}(v)$ of neighbors of v via learned multi-layer perceptrons (MLP):

$$h_v^{(k)} = \sum_{u \in \mathcal{N}(v)} \text{MLP}^{(k)}\left([h_v^{(k-1)}, h_u^{(k-1)}, x_{(u,v)}]\right), \quad \forall v \in V. \quad (2)$$

Here, we use $[\cdot, \cdot]$ for vector concatenation. We can optionally compute a *graph representation* h_G by aggregating the final representations of all nodes after a total of K iterations:

$$h_G = \text{MLP}^{(K+1)}\left(\sum_{v \in V} h_v^{(K)}\right). \quad (3)$$

Depending on the task, the final output is either the graph representation h_G or final node representations $h_u^{(K)}$. We refer to the sum in Eqn. 2 as *aggregation* and the sum in Eqn. 3 as *readout*. The above equations assume sum-aggregation and sum-readout, which are popular choices Battaglia et al. (2018). They can be replaced with other pooling operations such as max and min. We will discuss how aggregation and readout affect generalization and extrapolation in Chapter 2 and 3.

1.3 Overview of Research

In this dissertation, we present our analysis of how algorithmic alignment is useful for generalization, extrapolation, and robustness from three perspectives, and we propose a class of efficient approaches to make sequential decisions using knowledge from existing algorithms on MDPs where the only uncertainty affecting the decision outcomes are exogenous variables outside the control of the decision-maker (e.g., resource management problems like bin packing, virtual machine allocations, and so on).

In Chapter 2, we prove that better algorithmic alignment leads to lower sample complexity (a.k.a., easier to generalize) under simplified assumptions, and reveal how GNN is well aligned with DP. We also provide concrete examples to show how to compute the algorithmic alignment value given a task.

In Chapter 3, we show that linear algorithmic alignment (a.k.a., each component of the network only needs to learn a linear function) is needed for successful extrapolation, where we provide both theoretical and empirical support on MLPs and GNNs.

In Chapter 4, we extend algorithmic alignment to work with random process, and we demonstrate that better algorithmic alignment also leads to more robust representations learned under noisy labels. We prove theoretical support on a simplified noisy setting and provide empirical evidence across different models on various domains (including both synthetic and real-world label noise).

In Chapter 5, we observe that in many decision-making problems especially the ones in resource management, the system is often partially known and the only uncertainty is due to exogenous variables like resource requests — that is (to a first-order approximation) independent of an agent’s decisions. We model these problems as Exo-MDPs (Markov Decision Processes with Exogenous Inputs) and design a class of data-efficient algorithms for them termed Hindsight Learning (HL), where we utilize existing human-designed algorithms with historical data to better guide the solver.

Chapter 2: Generalization: What Can Neural Networks Reason About?

Joint work with Keyulu Xu, Mozhi Zhang, Simon S. Du, Ken-ichi Kawarabayashi, Stefanie Jegelka

2.1 Introduction

Recently, there have been many advances in building neural networks that can learn to reason. Reasoning spans a variety of tasks, for instance, visual and text-based question answering [Johnson et al. \(2017a\)](#); [Weston et al. \(2015\)](#); [Hu et al. \(2017a\)](#); [Fleuret et al. \(2011\)](#); [Antol et al. \(2015\)](#), intuitive physics, i.e., predicting the time evolution of physical objects [Battaglia et al. \(2016\)](#); [Watters et al. \(2017\)](#); [Fragkiadaki et al. \(2016\)](#); [Chang et al. \(2017a\)](#), mathematical reasoning [Saxton et al. \(2019\)](#); [Chang et al. \(2019\)](#) and visual IQ tests [Santoro et al. \(2018\)](#); [Zhang et al. \(2019a\)](#).

Curiously, neural networks that perform well in reasoning tasks usually possess specific *structures* [Santoro et al. \(2017\)](#). Many successful models follow the Graph Neural Network (GNN) framework [Battaglia et al. \(2016, 2018\)](#); [Palm et al. \(2018\)](#); [Mrowca et al. \(2018\)](#); [Sanchez-Gonzalez et al. \(2018\)](#); [Janner et al. \(2019\)](#). These networks explicitly model pairwise relations and recursively update each object’s representation by aggregating its relations with other objects. Other computational structures, e.g., neural symbolic programs [Yi et al. \(2018\)](#); [Mao et al.](#)

(2019a); Johnson et al. (2017b) and Deep Sets Zaheer et al. (2017), are effective on specific tasks.

However, there is limited understanding of the relation between the generalization ability and network structure for reasoning. *What tasks can a neural network (sample efficiently) learn to reason about?* Answering this question is crucial for understanding the empirical success and limitations of existing models, and for designing better models for new reasoning tasks.

This chapter is an initial work towards answering this fundamental question, by developing a theoretical framework to characterize what tasks a neural network can reason about. We build on a simple observation that reasoning processes resemble algorithms. Hence, we study how well a reasoning algorithm *aligns* with the computation graph of the network. Intuitively, if they align well, the network only needs to learn *simple* algorithm steps to simulate the reasoning process, which leads to better sample efficiency. We formalize this intuition with a numeric measure of algorithmic alignment, and show initial support for our hypothesis that algorithmic alignment facilitates learning: Under simplifying assumptions, we show a sample complexity bound that decreases with better alignment.

Our framework explains the empirical success of popular reasoning models and suggests their limitations. As concrete examples, we study four categories of increasingly complex reasoning tasks: summary statistics, relational argmax (asking about properties of the result of comparing multiple relations), dynamic programming, and NP-hard problems (Fig. 2.1). Using alignment, we characterize which architectures are expected to learn each task well: Networks inducing permutation invariance, such as Deep Sets Zaheer et al. (2017), can learn summary statistics, and one-iteration GNNs can learn relational argmax. Many other more complex tasks, such as intuitive physics, visual question answering, and shortest paths – despite seeming different – can all be solved via a powerful algorithmic paradigm: dynamic programming (DP) Bellman (1966).

Multi-iteration GNNs algorithmically align with DP and hence are expected to sample-efficiently learn these tasks. Indeed, they do. Our results offer an explanation for the popularity of GNNs in the relational reasoning literature, and also suggest limitations for tasks with even more complex structure. As an example of such a task, we consider subset sum, an NP-hard problem where GNNs indeed fail. Overall, empirical results (Fig. 2.3) agree with our theoretical analysis based on algorithmic alignment (Fig. 2.1). These findings also suggest how to take into account task structure when designing new architectures.

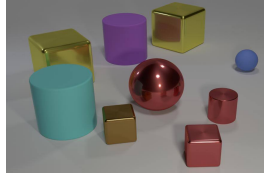
The perspective that structure in networks helps is not new. For example, in a well-known position paper, Battaglia et al. argue that GNNs are suitable for relational reasoning because they have relational inductive biases, but without formalizations Battaglia et al. (2018). Here, we take such ideas one step further, by introducing a *formal* definition (algorithmic alignment) for quantifying the relation between network and task structure, and by formally deriving implications for learning. These theoretical ideas are the basis for characterizing what reasoning tasks a network can learn well. Our algorithmic structural condition also differs from structural assumptions common in learning theory Vapnik (2013); Bartlett and Mendelson (2002); Bartlett et al. (2017); Neyshabur et al. (2015); Golowich et al. (2018) and specifically aligns with reasoning.

In summary, we introduce algorithmic alignment to analyze learning for reasoning. Our initial theoretical results suggest that algorithmic alignment is desirable for generalization. On four categories of reasoning tasks with increasingly complex structure, we apply our framework to analyze which tasks some popular networks can learn well. GNNs algorithmically align with dynamic programming, which solves a broad range of reasoning tasks. Finally, our framework implies guidelines for designing networks for new reasoning tasks. Experimental results confirm our theory.



Summary statistics

What is the maximum value difference among treasures?



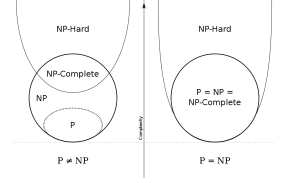
Relational argmax

What are the colors of the furthest pair of objects?



Dynamic programming

What is the cost to defeat monster X by following the optimal path?



NP-hard problem

Subset sum: Is there a subset that sums to 0?

Figure 2.1: **Overview of reasoning tasks with increasingly complex structure.** Each *task category* shows an example task on which we perform experiments in Section 2.4. Algorithmic alignment suggests that (a) Deep Sets and GNNs, but not MLP, can *sample efficiently* learn summary statistics, (b) GNNs, but not Deep Sets, can learn relational argmax, (c) GNNs can learn dynamic programming, an algorithmic paradigm that we show to unify many reasoning tasks, (d) GNNs cannot learn subset sum (NP-hard), but NES, a network we design based on exhaustive search, can generalize. Our theory agrees with empirical results (Fig. 2.3).

2.2 Preliminaries

We begin by introducing notations and summarizing common neural networks for reasoning tasks. Let S denote the universe, *i.e.*, a configuration/set of objects to reason about. Each object $s \in S$ is represented by a vector X . This vector could be state descriptions [Battaglia et al. \(2016\)](#); [Santoro et al. \(2017\)](#) or features learned from data such as images [Santoro et al. \(2017\)](#). Information about the specific question can also be included in the object representations. Given a set of universes $\{S_1, \dots, S_M\}$ and answer labels $\{y_1, \dots, y_M\} \subseteq \mathcal{Y}$, we aim to learn a function g that can answer questions about unseen universes, $y = g(S)$.

Multi-layer perceptron (MLP). For a single-object universe, applying an MLP on the object representation usually works well. But when there are multiple objects, simply applying an MLP to the concatenated object representations often does not generalize [Santoro et al. \(2017\)](#).

Deep Sets. As the input to the reasoning function is an unordered set, the function should be permutation-invariant, *i.e.*, the output is the same for all input orderings. To induce permuta-

tion invariance in a neural network, Zaheer et al. propose *Deep Sets* [Zaheer et al. \(2017\)](#), of the form

$$y = \text{MLP}_2\left(\sum_{s \in S} \text{MLP}_1(X_s)\right). \quad (1)$$

Graph Neural Networks (GNNs). GNNs are originally proposed for learning on graphs [Scarselli et al. \(2009a\)](#). Their structures follow a message passing scheme [Xu et al. \(2019, 2018\)](#); [Gilmer et al. \(2017\)](#), where the representation $h_s^{(k)}$ of each node s (in iteration k) is recursively updated by aggregating the representation of neighboring nodes. GNNs can be adopted for reasoning by considering objects as nodes and assuming all objects pairs are connected, *i.e.*, a complete graph [Battaglia et al. \(2018\)](#):

$$h_s^{(k)} = \sum_{t \in S} \text{MLP}_1^{(k)}\left(h_s^{(k-1)}, h_t^{(k-1)}\right), \quad h_S = \text{MLP}_2\left(\sum_{s \in S} h_s^{(K)}\right), \quad (2)$$

where h_S is the answer/output and K is the number of GNN layers. Each object’s representation is initialized as $h_s^{(0)} = X_s$. Although other aggregation functions are proposed, we use sum in our experiments. Similar to Deep Sets, GNNs are also permutation invariant. While Deep Sets focus on individual objects, GNNs can also focus on pairwise relations.

The GNN framework includes many reasoning models. Relation Networks [Santoro et al. \(2017\)](#) and Interaction Networks [Battaglia et al. \(2016\)](#) resemble one-layer GNNs. Recurrent Relational Networks [Palm et al. \(2018\)](#) apply LSTMs [Hochreiter and Schmidhuber \(1997\)](#) after aggregation. Transformer [Vaswani et al. \(2017\)](#) also resembles a GNN operating on a fully-connected graph with multiple parallel message passing functions.

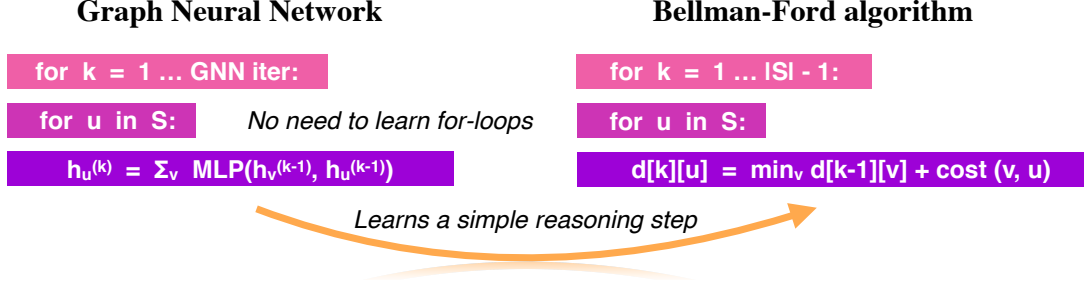


Figure 2.2: **Our framework suggests that better algorithmic alignment improves generalization.** As an example, our framework explains why GNN generalizes when learning to answer shortest paths. A correct reasoning process for the shortest paths task is the Bellman-Ford algorithm. The computation structure of a GNN (left) *aligns* well with Bellman-Ford (right): the GNN can simulate Bellman-Ford by merely learning a *simple reasoning step*, i.e., the relaxation step in the last line (a sum, and a min over neighboring nodes v) via its aggregation operation. In contrast, a giant MLP or Deep Set must learn the structure of the *entire for-loop*. Thus, the GNN is expected to generalize better when learning shortest paths, as is confirmed in experiments (Section 2.4.3).

2.3 Theoretical Framework: Algorithmic Alignment

Next, we study how the network structure and task may interact, and possible implications for generalization. Empirically, different network structures have different degrees of success in learning reasoning tasks, e.g., GNNs can learn relations well, but Deep Sets often fail (Fig. 2.3). However, all these networks are universal approximators (Propositions 2.3.1 and 2.3.2). Thus, their differences in test accuracy must come from generalization.

We observe that the answer to many reasoning tasks may be computed via a reasoning algorithm; we further illustrate the algorithms for some reasoning tasks in Section 2.4. Many neural networks can *represent* algorithms Pérez et al. (2019). For example, Deep Sets can universally represent permutation-invariant set functions Zaheer et al. (2017); Wagstaff et al. (2019). This also holds for GNNs and MLPs, as we show in Propositions 2.3.1 and 2.3.2 (our setting differs from Scarselli et al. Scarselli et al. (2009b) and Xu et al. Xu et al. (2019), who study functions

on graphs):

Proposition 2.3.1. *Let $f : \mathbb{R}^{d \times N} \rightarrow \mathbb{R}$ be any continuous function over sets S of bounded cardinality $|S| \leq N$. If f is permutation-invariant to the elements in S , and the elements are in a compact set in $\mathbb{R}^d$, then f can be approximated arbitrarily closely by a GNN (of any depth).*

Proposition 2.3.2. *For any GNN $\mathcal{N}$, there is an MLP that can represent all functions $\mathcal{N}$ can represent.*

But, empirically, not all network structures work well when *learning* these algorithms, i.e., they generalize differently. Intuitively, a network may generalize better if it can represent a function “more easily”. We formalize this idea by *algorithmic alignment*, formally defined in Definition 2.3.4. Indeed, not only the reasoning process has an algorithmic structure: the neural network’s architecture induces a computational structure on the function it computes. This corresponds to an algorithm that prescribes how the network combines computations from modules. Fig. 2.2 illustrates this idea for a GNN, where the modules are its MLPs applied to pairs of objects. In the shortest paths problem, the GNN matches the structure of the Bellman-Ford algorithm: to simulate the Bellman-Ford with a GNN, the GNN’s MLP modules only need to learn a *simple* update equation (Fig. 2.2). In contrast, if we want to represent the Bellman-Ford algorithm with a single MLP, it needs to simulate an *entire for-loop*, which is much more complex than one update step. Therefore, we expect the GNN to have better sample complexity than MLP when learning to solve shortest path problems.

This perspective suggests that a neural network which better aligns with a correct reasoning process (algorithmic solution) can more easily learn a reasoning task than a neural network that does not align well. If we look more broadly at reasoning, there may also exist solutions

which only solve a task approximately, or whose structure is obtuse. In this chapter, we focus on reasoning tasks whose underlying reasoning process is exact and has clear algorithmic structure. We leave the study of approximation algorithms and unknown structures for future work.

2.3.1 Formalization of Algorithmic Alignment

We formalize the above intuition in a PAC learning framework [Valiant \(1984\)](#). PAC learnability formalizes *simplicity* as sample complexity, i.e., the number of samples needed to ensure low test error with high probability. It refers to a learning algorithm $\mathcal{A}$ that, given training samples $\{x_i, y_i\}_{i=1}^M$, outputs a function $f = \mathcal{A}(\{x_i, y_i\}_{i=1}^M)$. The learning algorithm here is the neural network and its training method, e.g., gradient descent. A function is simple if it has low sample complexity.

Definition 2.3.3. (PAC learning and sample complexity). Fix an error parameter $\epsilon > 0$ and failure probability $\delta \in (0, 1)$. Suppose $\{x_i, y_i\}_{i=1}^M$ are i.i.d. samples from some distribution $\mathcal{D}$, and the data satisfies $y_i = g(x_i)$ for some underlying function g . Let $f = \mathcal{A}(\{x_i, y_i\}_{i=1}^M)$ be the function generated by a learning algorithm $\mathcal{A}$. Then g is (M, ϵ, δ) -learnable with $\mathcal{A}$ if

$$\mathbb{P}_{x \sim \mathcal{D}}[\|f(x) - g(x)\| \leq \epsilon] \geq 1 - \delta. \quad (3)$$

The sample complexity $\mathcal{C}_{\mathcal{A}}(g, \epsilon, \delta)$ is the minimum M so that g is (M, ϵ, δ) -learnable with $\mathcal{A}$.

With the PAC learning framework, we define a numeric measure of algorithmic alignment (Definition [2.3.4](#)), and under simplifying assumptions, we show that the sample complexity decreases with better algorithmic alignment (Theorem [2.3.6](#)).

Formally, a neural network aligns with an algorithm if it can simulate the algorithm via a limited number of modules, and each module is simple, i.e., has low sample complexity.

Definition 2.3.4. (Algorithmic alignment). *Let g be a reasoning function and $\mathcal{N}$ a neural network with n modules $\mathcal{N}_i$. The module functions $f_1, \dots, f_n$ generate g for $\mathcal{N}$ if, by replacing $\mathcal{N}_i$ with f_i , the network $\mathcal{N}$ simulates g . Then $\mathcal{N}$ (M, ϵ, δ) -algorithmically aligns with g if (1) $f_1, \dots, f_n$ generate g and (2) there are learning algorithms $\mathcal{A}_i$ for the $\mathcal{N}_i$'s such that $n \cdot \max_i C_{\mathcal{A}_i}(f_i, \epsilon, \delta) \leq M$.*

Good algorithmic alignment, i.e., small M , implies that all algorithm steps f_i to simulate the algorithm g are *easy to learn*. Therefore, the algorithm steps should not simulate complex programming constructs such as for-loops, whose sample complexity is large (Theorem 2.3.5).

Next, we show how to compute the algorithmic alignment value M . Algorithmic alignment resembles Kolmogorov complexity [Kolmogorov \(1998\)](#) for neural networks. Thus, it is generally non-trivial to obtain the optimal alignment between a neural network and an algorithm. However, one important difference to Kolmogorov complexity is that any algorithmic alignment that yields decent sample complexity is good enough (unless we want the tightest bound). In Section 2.4, we will see several examples where finding a good alignment is not hard. Then, we can compute the value of an alignment by summing the sample complexity of the algorithm steps with respect to the modules, e.g. MLPs. For illustration, we show an example of how one may compute sample complexity of MLP modules.

A line of works show one can analyze the optimization and generalization behavior of over-parameterized neural networks via neural tangent kernel (NTK) [Allen-Zhu et al. \(2019a\)](#); [Arora et al. \(2019a,b, 2020\)](#); [Du et al. \(2019a,b\)](#); [Jacot et al. \(2018\)](#); [Li and Liang \(2018\)](#). Building

upon Arora et al. [Arora et al. \(2019a\)](#), Du et al. [Du et al. \(2019c\)](#) show that infinitely-wide GNNs trained with gradient descent can provably learn certain smooth functions. This chapter studies a broader class of functions, e.g., algorithms, compared to those studied in Du et al. [Du et al. \(2019c\)](#), but with more simplifying assumptions.

Here, Theorem [2.3.5](#), proved in the Appendix, summarizes and extends Theorem 6.1 of Arora et al. [Arora et al. \(2019a\)](#) for over-parameterized MLP modules to vector-valued functions. Our framework can be used with other sample complexity bounds for other types of modules, too.

Theorem 2.3.5. (Sample complexity for overparameterized MLP modules). *Let $\mathcal{A}$ be an over-parameterized and randomly initialized two-layer MLP trained with gradient descent for a sufficient number of iterations. Suppose $g : \mathbb{R}^d \rightarrow \mathbb{R}^m$ with components $g(x)^{(i)} = \sum_j \alpha_j^{(i)} (\beta_j^{(i)\top} x)^{p_j^{(i)}}$, where $\beta_j^{(i)} \in \mathbb{R}^d$, $\alpha \in \mathbb{R}$, and $p_j^{(i)} = 1$ or $p_j^{(i)} = 2l$ ($l \in \mathbb{N}_+$). The sample complexity $\mathcal{C}_{\mathcal{A}}(g, \epsilon, \delta)$ is*

$$\mathcal{C}_{\mathcal{A}}(g, \epsilon, \delta) = O\left(\frac{\max_i \sum_{j=1}^K p_j^{(i)} |\alpha_j^{(i)}| \cdot \|\beta_j^{(i)}\|_2^{p_j^{(i)}} + \log(m/\delta)}{(\epsilon/m)^2}\right). \quad (4)$$

Theorem [2.3.5](#) suggests that functions that are “simple” when expressed as a polynomial, e.g., via a Taylor expansion, are sample efficiently learnable by an MLP module. Thus, algorithm steps that perform computation over many objects may require many samples for an MLP module to learn, since the number K of polynomials or $\|\beta_j^{(i)}\|$ can increase in Eqn. (4). “For loop” is one example of such complex algorithm steps.

2.3.2 Better Algorithmic Alignment Implies Better Generalization

We show an initial result demonstrating that algorithmic alignment is desirable for generalization. Theorem 2.3.6 states that, in a simplifying setting where we sequentially train modules of a network with auxiliary labels, the sample complexity bound increases with algorithmic alignment value M .

While we do not have auxiliary labels in practice, we observe the same pattern for end-to-end learning in experiments (Section 2.4). We leave sample complexity analysis for end-to-end learning to future work. We prove Theorem 2.3.6 in Section 6.1.4.

Theorem 2.3.6. (Algorithmic alignment improves sample complexity). Fix ϵ and δ . Suppose $\{S_i, y_i\}_{i=1}^M \sim \mathcal{D}$, where $|S_i| < N$, and $y_i = g(S_i)$ for some g . Suppose $\mathcal{N}_1, \dots, \mathcal{N}_n$ are network $\mathcal{N}$'s MLP modules in sequential order. Suppose $\mathcal{N}$ and g (M, ϵ, δ) -algorithmically align via functions $f_1, \dots, f_n$. Under the following assumptions, g is $(M, O(\epsilon), O(\delta))$ -learnable by $\mathcal{N}$.

a) Algorithm stability. Let $\mathcal{A}$ be the learning algorithm for the $\mathcal{N}_i$'s. Suppose $f = \mathcal{A}(\{x_i, y_i\}_{i=1}^M)$, and $\hat{f} = \mathcal{A}(\{\hat{x}_i, y_i\}_{i=1}^M)$. For any x , $\|f(x) - \hat{f}(x)\| \leq L_0 \cdot \max_i \|x_i - \hat{x}_i\|$, for some L_0 .

b) Sequential learning. We train $\mathcal{N}_i$'s sequentially: $\mathcal{N}_1$ has input samples $\{\hat{x}_i^{(1)}, f_1(\hat{x}_i^{(1)})\}_{i=1}^N$, with $\hat{x}_i^{(1)}$ obtained from S_i . For $j > 1$, the input $\hat{x}_i^{(j)}$ for $\mathcal{N}_j$ are the outputs from the previous modules, but labels are generated by the correct functions $f_{j-1}, \dots, f_1$ on $\hat{x}_i^{(1)}$.

c) Lipschitzness. The learned functions $\hat{f}_j$ satisfy $\|\hat{f}_j(x) - \hat{f}_j(\hat{x})\| \leq L_1 \|x - \hat{x}\|$, for some L_1 .

In our analysis, the Lipschitz constants and the universe size are constants going into $O(\epsilon)$ and $O(\delta)$. As an illustrative example, we use Theorem 2.3.6 and 2.3.5 to show that GNN has a polynomial improvement in sample complexity over MLP when learning simple relations. Indeed, GNN aligns better with summary statistics of pairwise relations than MLP does (Sec-

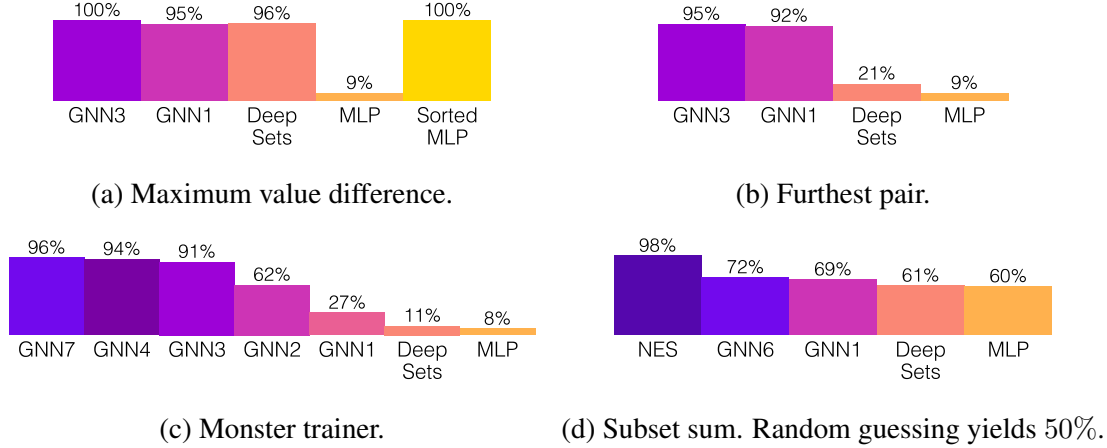


Figure 2.3: **Test accuracies on reasoning tasks with increasingly complex structure.** Fig. 2.1 shows an overview of the tasks. $\text{GNN}k$ is GNN with k iterations. (a) Summary statistics. All models except MLP generalize. (b) Relational argmax. Deep Sets fail. (c) Dynamic programming. Only GNNs with sufficient iterations generalize. (d) An NP-hard problem. Even GNNs fail, but NES generalizes.

tion 2.4.1).

Corollary 2.3.7. Suppose universe S has ℓ objects $X_1, \dots, X_\ell$, and $g(S) = \sum_{i,j} (X_i - X_j)^2$. In the setting of Theorem 2.3.6, the sample complexity bound for MLP is $O(\ell^2)$ times larger than for GNN.

2.4 Predicting What Neural Networks Can Reason About

Next, we apply our framework to analyze the neural networks for reasoning from Section 2.2: MLP, Deep Sets, and GNNs. Using algorithmic alignment, we predict whether each model can generalize on four categories of increasingly complex reasoning tasks: summary statistics, relational argmax, dynamic programming, and an NP-hard problem (Fig. 2.3). Our theoretical analysis is confirmed with experiments (Dataset and training details are in Section 6.2). To empirically compare sample complexity of different models, we make sure all models perfectly

fit training sets through extensive hyperparameter tuning. Therefore, the test accuracy reflects how well a model generalizes.

The examples in this section, together with our framework, suggest an explanation why GNNs are widely successful across reasoning tasks: Popular reasoning tasks such as visual question answering and intuitive physics can be solved by DP. GNNs align well with DP, and hence are expected to learn sample efficiently.

2.4.1 Summary Statistics

As discussed in Section 2.2, we assume each object X has a state representation $X = [h_1, h_2, \dots, h_k]$, where each $h_i \in \mathbb{R}^{d_i}$ is a feature vector. An MLP can learn simple polynomial functions of the state representation (Theorem 2.3.5). In this section, we show how Deep Sets use MLP as building blocks to learn summary statistics.

Questions about summary statistics are common in reasoning tasks. One example from CLEVR Johnson et al. (2017a) is “How many objects are either small cylinders or red things?” Deep Sets (Eqn. 1) align well with algorithms that compute summary statistics over individual objects. Suppose we want to compute the sum of a feature over all objects. To simulate the reasoning algorithm, we can use the first MLP in Deep Sets to extract the desired feature and aggregate them using the pooling layer. Under this alignment, each MLP only needs to learn simple steps, which leads to good sample complexity. Similarly, Deep Sets can learn to compute *max* or *min* of a feature by using smooth approximations like the softmax $\max_{s \in S} X_s \approx \log(\sum_{s \in X_s} \exp(X_s))$. In contrast, if we train an MLP to perform sum or max, the MLP must learn a complex for-loop and therefore needs more samples. Therefore, our frame-

work predicts that Deep Sets have better sample complexity than MLP when learning summary statistics.

Maximum value difference. We confirm our predictions by training models to compute the *maximum value difference* task. Each object in this task is a treasure $X = [h_1, h_2, h_3]$ with location h_1 , value h_2 , and color h_3 . We train models to predict the difference in value between the most and the least valuable treasure, $y(S) = \max_{s \in S} h_2(X_s) - \min_{s \in S} h_2(X_s)$.

The test accuracy follows our prediction (Fig. 2.3a). MLP does not generalize and only has 9% test accuracy, while Deep Sets has 96%. Interestingly, if we sort the treasures by value (Sorted MLP in Fig. 2.3a), MLP achieves perfect test accuracy. This observation can be explained with our theory—when the treasures are sorted, the reasoning algorithm is reduced to a simple subtraction: $y(S) = h_2(X_{|S|}) - h_2(X_1)$, which has a low sample complexity for even MLPs (Theorem 2.3.5). GNNs also have high test accuracies. This is because summary statistics are a special case of relational argmax, which GNNs can learn as shown next.

2.4.2 Relational Argmax

Next, we study *relational argmax*: tasks where we need to compare pairwise relations and answer a question about that result. For example, a question from Sort-of-CLEVR Santoro et al. (2017) asks “What is the shape of the object that is farthest from the gray object?”, which requires comparing the distance between object pairs.

One-iteration GNN aligns well with relational argmax, as it sums over all pairs of objects, and thus can compare, e.g. via softmax, pairwise information without learning the “for loops”. In contrast, Deep Sets require many samples to learn this, because most pairwise relations cannot

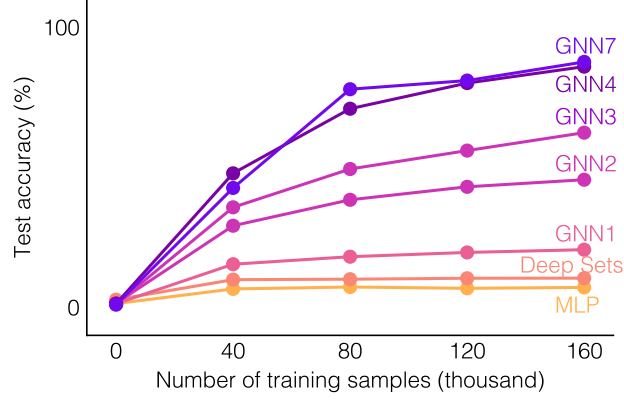


Figure 2.4: **Test accuracy vs. training set size** for models trained on sub-sampled training sets and evaluated on the same test set of monster trainer (DP task). Test accuracies increase faster when a neural network aligns well with an algorithmic solution of the task. For example, the test accuracy of GNN4 increases by 23% when the number of training samples increases from 40,000 to 80,000, which is much higher than that of Deep Sets (0.2%).

be encoded as a sum of individual objects:

Claim 2.4.1. *Suppose $g(x, y) = 0$ if and only if $x = y$. There is no f such that $g(x, y) = f(x) + f(y)$.*

Therefore, if we train a Deep Set to compare pairwise relations, one of the MLP modules has to learn a complex “for loop”, which leads to poor sample complexity. Our experiment confirms that GNNs generalize better than Deep Sets when learning relational argmax.

Furthest pair. As an example of relational argmax, we train models to identify the furthest pair among a set of objects. We use the same object settings as the maximum value difference task. We train models to find the colors of the two treasures with the largest distance. The answer is a pair of colors, encoded as an integer category:

$$y(S) = (h_3(X_{s_1}), h_3(X_{s_2})) \quad \text{s.t.} \quad \{X_{s_1}, X_{s_2}\} = \arg \max_{s_1, s_2 \in S} \|h_1(X_{s_1}) - h_1(X_{s_2})\|_{\ell_1}$$

Distance as a pairwise function satisfies the condition in Claim 2.4.1. As predicted by our frame-

work, Deep Sets has only 21% test accuracy, while GNNs have more than 90% accuracy.

2.4.3 Dynamic Programming

We observe that a broad class of relational reasoning tasks can be unified by the powerful algorithmic paradigm *dynamic programming (DP)* [Bellman \(1966\)](#). DP recursively breaks down a problem into simpler sub-problems. It has the following general form:

$$\text{Answer}[k][i] = \text{DP-Update}(\{\text{Answer}[k-1][j]\}, j = 1 \dots n), \quad (5)$$

where $\text{Answer}[k][i]$ is the solution to the sub-problem indexed by iteration k and state i , and DP-Update is an task-specific update function that computes $\text{Answer}[k][i]$ from $\text{Answer}[k-1][j]$'s.

GNNs algorithmically align with a class of DP algorithms. We can interpret GNN as a DP algorithm, where node representations $h_i^{(k)}$ are $\text{Answer}[k][i]$, and the GNN aggregation step is the DP-Update. Therefore, Theorem [2.3.6](#) suggests that a GNN with enough iterations can sample efficiently learn any DP algorithm with a simple DP-update function, e.g. sum/min/max.

Shortest paths. As an example, we experiment with GNN on Shortest paths, a standard DP problem. Shortest paths can be solved by the Bellman-Ford algorithm [Bellman \(1958\)](#), which recursively updates the minimum distance between each object u and the source s :

$$\text{distance}[1][u] = \text{cost}(s, u), \quad \text{distance}[k][u] = \min_v \{ \text{distance}[k-1][v] + \text{cost}(v, u) \}, \quad (6)$$

As discussed above, GNN aligns well with this DP algorithm. Therefore, our framework predicts that GNN has good sample complexity when learning to find shortest paths. To verify

this, we test different models on a **monster trainer** game, which is a shortest path variant with unknown cost functions that need to be learned by the models. Section 6.2.3 describes the task in details.

In Fig. 2.3c, only GNNs with at least four iterations generalize well. The empirical result confirms our theory: a neural network can sample efficiently learn a task if it aligns with a correct algorithm. Interestingly, GNN does not need as many iterations as Bellman-Ford. While Bellman-Ford needs $N = 7$ iterations, GNNs with four iterations have almost identical test accuracy as GNNs with seven iterations (94% vs 95%). This can also be explained through algorithmic alignment, as GNN aligns with an optimized version of Bellman-Ford, which we explain in Section 6.2.3.

Fig. 2.4 shows how the test accuracies of different models vary with the number of subsampled training points. Indeed, the test accuracy increases more slowly for models that align worse with the task, which implies they need more training samples to achieve similar generalization performance. Again, this confirms our theory.

After verifying that GNNs can sample-efficiently learn DP, we show that two popular families of reasoning tasks, visual question answering and intuitive physics, can be formulated as DP. Therefore, our framework explains why GNNs are effective in these tasks.

Visual question answering. The Pretty-CLEVR dataset Palm et al. (2018) is an extension of Sort-of-CLEVR Santoro et al. (2017) and CLEVR Johnson et al. (2017a). GNNs work well on these datasets. Each question in Pretty-CLEVR has state representations and asks “Starting at object X , if each time we jump to the closest object, which object is K jumps away?”. This problem can be solved by DP, which computes the answers for k jumps from the answers for

$(k - 1)$ jumps.

$$\text{closest}[1][i] = \arg \min_j d(i, j), \quad \text{closest}[k][i] = \text{closest}[k - 1][\text{closest}[1][i]] \text{ for } k > 1, \quad (7)$$

where $\text{closest}[k][i]$ is the answer for jumping k times from object i , and $d(i, j)$ is the distance between the i -th and the j -th object.

Intuitive physics. Battaglia et al. [Battaglia et al. \(2016\)](#) and Watters et al. [Watters et al. \(2017\)](#) train neural networks to predict object dynamics in rigid body scenes and n-body systems. Chang et al. [Chang et al. \(2017a\)](#) and Janner et al. [Janner et al. \(2019\)](#) study other rigid body scenes. If the force acting on a physical object stays constant, we can compute the object’s trajectory with simple functions (physics laws) based on its initial position and force. Physical interactions, however, make the force change, which means the function to compute the object’s dynamics has to change too. Thus, a DP algorithm would *recursively* compute the next force changes in the system and update DP states (velocity, momentum, position etc of objects) according to the (learned) forces and physics laws [Thijssen \(2007\)](#).

$$\text{for } k = 1..K : \quad \text{time} = \min_{i,j} \text{Force-change-time}(\text{state}[k - 1, i], \text{state}[k - 1, j]), \quad (8)$$

$$\text{for } i = 1..N : \quad \text{state}[k][i] = \text{Update-by-forces}(\text{state}[k - 1][j], \text{time}), \quad j = 1..N, \quad (9)$$

Force-change-time computes the time at which the force between object i and j will change. Update-by-forces updates the state of each object at the next force change time. In rigid body systems, force changes only at collision. In datasets where no object collides more than once between time frames, one-iteration algorithm/GNN can work [Battaglia et al. \(2016\)](#). More itera-

tions are needed if multiple collisions occur between two consecutive frames [Li and Liang \(2018\)](#). In n-body systems, forces change continuously but smoothly. Thus, finite-iteration DP/GNN can be viewed as a form of Runge-Kutta method [DeVries and Hamill \(1995\)](#).

2.4.4 Designing Neural Networks with Algorithmic Alignment

While DP solves many reasoning tasks, it has limitations. For example, NP-hard problems cannot be solved by DP. It follows that GNN also cannot sample-efficiently learn these hard problems. Our framework, however, goes beyond GNNs. If we know the structure of a suitable underlying reasoning algorithm, we can design a network with a similar structure to learn it. If we have no prior knowledge about the structure, then neural architecture search over algorithmic structures will be needed.

Subset Sum. As an example, we design a new architecture that can learn to solve the subset sum problem: Given a set of numbers, does there exist a subset that sums to 0? Subset sum is NP-hard [Karp \(1972\)](#) and cannot be solved by DP. Therefore, our framework predicts that GNN cannot generalize on this task. One subset sum algorithm is exhaustive search, where we enumerate all $2^{|S|}$ possible subsets τ and check whether τ has zero-sum. Following this algorithm, we design a similarly structured neural network which we call **Neural Exhaustive Search (NES)**. Given a universe, NES enumerates all subsets of objects and passes each subset through an LSTM followed by a MLP. The results are aggregated with a max-pooling layer and MLP:

$$\text{MLP}_2(\max_{\tau \subseteq S} \text{MLP}_1 \circ \text{LSTM}(X_1, \dots, X_{|\tau|} : X_1, \dots, X_{|\tau|} \in \tau)). \quad (10)$$

This architecture aligns well with subset-sum, since the first MLP and LSTM only need to learn a simple step, checking whether a subset has zero sum. Therefore, we expect NES to generalize well in this task. Indeed, NES has 98% test accuracy, while other models perform much worse (Fig. 2.3d).

2.5 Chapter Summary

This chapter is an initial step towards formally understanding how neural networks can learn to reason. In particular, we answer what tasks a neural network can learn to reason about well, by studying the generalization ability of learning the underlying reasoning processes for a task. To this end, we introduce an algorithmic alignment framework to formalize the interaction between the structure of a neural network and a reasoning process, and provide preliminary results on sample complexity. Our results explain the success and suggest the limits of current neural architectures: Graph Neural Networks generalize in many popular reasoning tasks because the underlying reasoning processes for those tasks resemble dynamic programming.

Our algorithmic alignment perspective may inspire neural network design and opens up theoretical avenues. An interesting direction for future work is to design, e.g. via algorithmic alignment, neural networks that can learn other reasoning paradigms beyond dynamic programming, and to explore the neural architecture search space of algorithmic structures.

From a broader standpoint, reasoning assumes a good representation of the concepts and objects in the world. To complete the picture, it would also be interesting to understand how to better disentangle and eventually integrate “representation” and “reasoning”.

My main contributions to this chapter include the initial exploration experiments, sec-

tion 2.3.1, section 2.4, and additional experiments in (Appendix) Chapter 6. The research findings from this chapter have been published and have since inspired numerous follow-up works. Subsequent studies have leveraged the insights that GNNs can efficiently learn DP algorithms to investigate the generalization and representation capabilities of GNNs [Garg et al. \(2020\)](#), and to apply GNNs for combinatorial optimization and reasoning tasks [Cappart et al. \(2023\)](#). Moreover, algorithmic alignment has been applied for implicit planning in RL [Deac et al. \(2021\)](#), and the alignments between GNNs and DP have been quantified in [Dudzik and Veličković \(2022\)](#).

Chapter 3: Extrapolation: How does Neural Networks Extrapolate?

Joint work with Keyulu Xu, Mozhi Zhang, Simon S. Du, Ken-ichi Kawarabayashi, Stefanie Jegelka

3.1 Introduction

Humans extrapolate well in many tasks. For example, we can apply arithmetics to arbitrarily large numbers. One may wonder whether a neural network can do the same and generalize to examples arbitrarily far from the training data [Lake et al. \(2017\)](#). Curiously, previous works report mixed extrapolation results with neural networks. Early works demonstrate feedforward neural networks, a.k.a. multilayer perceptrons (MLPs), fail to extrapolate well when learning simple polynomial functions [Barnard and Wessels \(1992\)](#); [Haley and Soloway \(1992\)](#). However, recent works show Graph Neural Networks (GNNs) [Scarselli et al. \(2009a\)](#), a class of structured networks with MLP building blocks, can generalize to graphs much larger than training graphs in challenging algorithmic tasks, such as predicting the time evolution of physical systems [Battaglia et al. \(2016\)](#), learning graph algorithms [Velickovic et al. \(2020\)](#), and solving mathematical equations [Lample and Charton \(2020\)](#).

To explain this puzzle, we formally study how neural networks trained by gradient descent (GD) extrapolate, i.e., what they learn outside the support of training distribution. We say a neural

network extrapolates well if it learns a task outside the training distribution. At first glance, it may seem that neural networks can behave arbitrarily outside the training distribution since they have high capacity [Zhang et al. \(2017a\)](#) and are universal approximators [Cybenko \(1989\)](#); [Funahashi \(1989\)](#); [Hornik et al. \(1989\)](#); [Kůrková \(1992\)](#). However, neural networks are constrained by gradient descent training [Hardt et al. \(2016\)](#); [Soudry et al. \(2018\)](#). In our analysis, we explicitly consider such implicit bias through the analogy of the training dynamics of over-parameterized neural networks and kernel regression via the *neural tangent kernel (NTK)* [Jacot et al. \(2018\)](#).

Starting with feedforward networks, the simplest neural networks and building blocks of more complex architectures such as GNNs, we establish that the predictions of over-parameterized MLPs with ReLU activation trained by GD converge to linear functions along any direction from the origin. We prove a convergence rate for two-layer networks and empirically observe that convergence often occurs *close to* the training data (Figure 3.1), which suggests ReLU MLPs cannot extrapolate well for most nonlinear tasks. We emphasize that our results do not follow from the fact that ReLU networks have finitely many linear regions [Arora et al. \(2018\)](#); [Hanin and Rolnick \(2019\)](#); [Hein et al. \(2019\)](#). While having finitely many linear regions implies ReLU MLPs *eventually* become linear, it does not say whether MLPs will learn the correct target function close to the training distribution. In contrast, our results are non-asymptotic and quantify what kind of functions MLPs will learn close to the training distribution. Second, we identify a condition when MLPs extrapolate well: the task is linear and the geometry of the training distribution is sufficiently “diverse”. To our knowledge, our results are the first extrapolation results of this kind for feedforward neural networks.

We then relate our insights into feedforward neural networks to GNNs, to explain why GNNs extrapolate well in some algorithmic tasks. In chapter 2, we see successful extrapolation

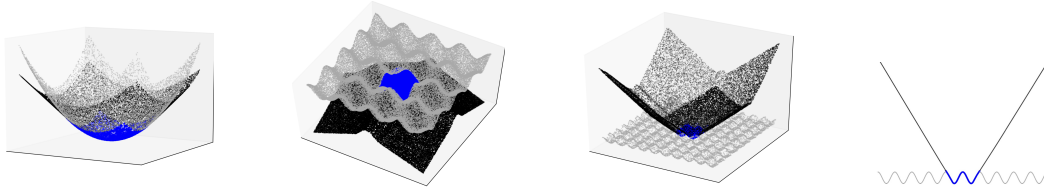


Figure 3.1: **How ReLU MLPs extrapolate.** We train MLPs to learn nonlinear functions (grey) and plot their predictions both within (blue) and outside (black) the training distribution. MLPs converge quickly to linear functions outside the training data range along directions from the origin (Theorem 3.3.2). Hence, MLPs do not extrapolate well in most nonlinear tasks. But, with appropriate training data, MLPs can provably extrapolate linear target functions (Theorem 3.3.4).

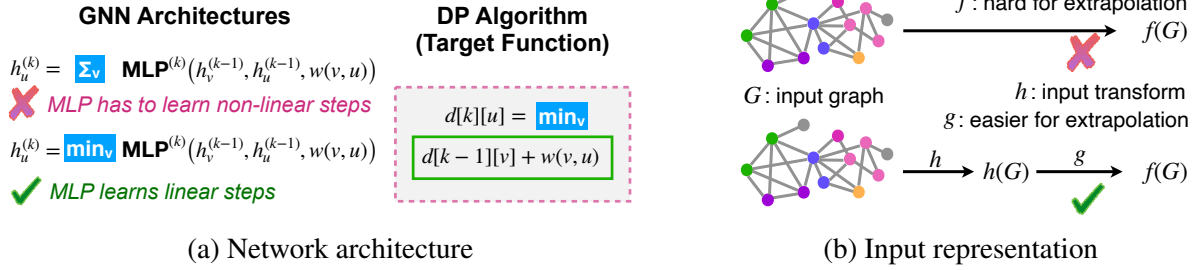


Figure 3.2: **How GNNs extrapolate.** Since MLPs can extrapolate well when learning linear functions, we hypothesize that GNNs can extrapolate well in dynamic programming (DP) tasks if we encode appropriate non-linearities in the architecture (left) and input representation (right; through domain knowledge or representation learning). The encoded non-linearities may not be necessary for interpolation, as they can be approximated by MLP modules, but they help extrapolation. We support the hypothesis theoretically (Theorem 3.4.4) and empirically (Figure 3.6).

for tasks that can be solved by dynamic programming (DP) [Bellman \(1966\)](#), which has a computation structure aligned with GNNs. DP updates can often be decomposed into nonlinear and linear steps. Hence, we hypothesize that GNNs trained by GD can extrapolate well in a DP task, if we encode appropriate non-linearities in the *architecture* and *input representation* (Figure 3.2). Importantly, encoding non-linearities may be unnecessary for GNNs to *interpolate*, because the MLP modules can easily learn many nonlinear functions inside the training distribution [Cybenko \(1989\)](#); [Hornik et al. \(1989\)](#), but it is crucial for GNNs to *extrapolate* correctly. We prove this hypothesis for a simplified case using *Graph NTK* [Du et al. \(2019c\)](#). Empirically, we validate the hypothesis on three DP tasks: max degree, shortest paths, and n -body problem. We show GNNs

with appropriate architecture, input representation, and training distribution can predict well on graphs with unseen sizes, structures, edge weights, and node features. Our theory explains the empirical success in previous works and suggests their limitations: successful extrapolation relies on encoding task-specific non-linearities, which requires domain knowledge or extensive model search. From a broader standpoint, our insights go beyond GNNs and apply broadly to other neural networks.

To summarize, we study how neural networks extrapolate. First, ReLU MLPs trained by GD converge to linear functions along directions from the origin with a rate of $O(1/t)$. Second, to explain why GNNs extrapolate well in some algorithmic tasks, we prove that ReLU MLPs can extrapolate well in linear tasks, leading to a hypothesis: a neural network can extrapolate well when appropriate non-linearities are encoded into the architecture and features. We prove this hypothesis for a simplified case and provide empirical support for more general settings.

3.1.1 Background

Early works show example tasks where MLPs do not extrapolate well, e.g. learning simple polynomials [Barnard and Wessels \(1992\)](#); [Haley and Soloway \(1992\)](#). We instead show a *general* pattern of how ReLU MLPs extrapolate and identify conditions for MLPs to extrapolate well. More recent works study the implicit biases induced on MLPs by gradient descent, for both the NTK and mean field regimes [Bietti and Mairal \(2019\)](#); [Chizat and Bach \(2018\)](#); [Song et al. \(2018a\)](#). Related to our results, some works show MLP predictions converge to “simple” piecewise linear functions, e.g., with few linear regions [Hanin and Rolnick \(2019\)](#); [Maennel et al. \(2018\)](#); [Savarese et al. \(2019\)](#); [Williams et al. \(2019\)](#). This chapter differs in that none of these

works explicitly studies extrapolation, and some focus only on one-dimensional inputs. Recent works also show that in high-dimensional settings of the NTK regime, MLP is asymptotically at most a linear predictor in certain scaling limits [Ba et al. \(2020\)](#); [Ghorbani et al. \(2019\)](#). We study a different setting (extrapolation), and our analysis is non-asymptotic in nature and does not rely on random matrix theory.

Prior works explore GNN extrapolation by testing on larger graphs [Saxton et al. \(2019\)](#); [Santoro et al. \(2018\)](#); [Battaglia et al. \(2018\)](#); [Velickovic et al. \(2020\)](#). We are the first to theoretically study GNN extrapolation, and we complete the notion of extrapolation to include unseen features and structures.

3.2 Preliminaries

We begin by introducing our setting. Let $\mathcal{X}$ be the domain of interest, e.g., vectors or graphs. The task is to learn an underlying function $g : \mathcal{X} \rightarrow \mathbb{R}$ with a training set $\{(\mathbf{x}_i, y_i)\}_{i=1}^n \subset \mathcal{D}$, where $y_i = g(\mathbf{x}_i)$ and $\mathcal{D}$ is the support of training distribution. Previous works have extensively studied in-distribution generalization where the training and the test distributions are identical [Valiant \(1984\)](#); [Vapnik \(2013\)](#); i.e., $\mathcal{D} = \mathcal{X}$. In contrast, extrapolation addresses predictions on a domain $\mathcal{X}$ that is larger than the support of the training distribution $\mathcal{D}$. We will say a model *extrapolates well* if it has a small *extrapolation error*.

Definition 3.2.1. (*Extrapolation error*). Let $f : \mathcal{X} \rightarrow \mathbb{R}$ be a model trained on $\{(\mathbf{x}_i, y_i)\}_{i=1}^n \subset \mathcal{D}$ with underlying function $g : \mathcal{X} \rightarrow \mathbb{R}$. Let $\mathcal{P}$ be a distribution over $\mathcal{X} \setminus \mathcal{D}$ and let $\ell : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$ be a loss function. We define the extrapolation error of f as $\mathbb{E}_{\mathbf{x} \sim \mathcal{P}}[\ell(f(\mathbf{x}), g(\mathbf{x}))]$.

We focus on neural networks trained by *gradient descent* (GD) or its variants with *squared*

loss. We study two network architectures: feedforward and graph neural networks.

Graph Neural Networks. GNNs are structured networks operating on graphs with MLP modules [Battaglia et al. \(2018\)](#); [Xu et al. \(2019\)](#). Let $G = (V, E)$ be a graph. Each node $u \in V$ has a feature vector $\mathbf{x}_u$, and each edge $(u, v) \in E$ has a feature vector $\mathbf{w}_{(u,v)}$. GNNs recursively compute node representations $\mathbf{h}_u^{(k)}$ at iteration k [Xu et al. \(2018\)](#); [Gilmer et al. \(2017\)](#). Initially, $\mathbf{h}_u^{(0)} = \mathbf{x}_u$. For $k = 1..K$, GNNs update $\mathbf{h}_u^{(k)}$ by aggregating the neighbor representations. We can optionally compute a graph representation $\mathbf{h}_G$ by aggregating the final node representations. That is,

$$\mathbf{h}_u^{(k)} = \sum_{v \in \mathcal{N}(u)} \text{MLP}^{(k)}\left(\mathbf{h}_u^{(k-1)}, \mathbf{h}_v^{(k-1)}, \mathbf{w}_{(v,u)}\right), \quad \mathbf{h}_G = \text{MLP}^{(K+1)}\left(\sum_{u \in G} \mathbf{h}_u^{(K)}\right). \quad (1)$$

The final output is the graph representation $\mathbf{h}_G$ or final node representations $\mathbf{h}_u^{(K)}$ depending on the task. We refer to the neighbor aggregation step for $\mathbf{h}_u^{(k)}$ as *aggregation* and the pooling step in $\mathbf{h}_G$ as *readout*. Previous works typically use sum-aggregation and sum-readout [Battaglia et al. \(2018\)](#). Our results indicate why replacing them may help extrapolation (Section 3.4).

3.3 How Feedforward Neural Networks Extrapolate

Feedforward networks are the simplest neural networks and building blocks of more complex architectures such as GNNs, so we first study how they extrapolate when trained by GD. Throughout the chapter, we assume ReLU activation. Section 3.3.3 contains preliminary results for other activations.

3.3.1 Linear Extrapolation Behavior of ReLU MLPs

By architecture, ReLU networks learn piecewise linear functions, but what do these regions precisely look like outside the support of the training data? Figure 3.1 illustrates examples of how ReLU MLPs extrapolate when trained by GD on various nonlinear functions. These examples suggest that outside the training support, the predictions quickly become linear along directions from the origin. We systematically verify this pattern by linear regression on MLPs’ predictions: the coefficient of determination (R^2) is always greater than 0.99 (Section 7.1.2). That is, ReLU MLPs “linearize” almost immediately outside the training data range.

We formalize this observation using the implicit biases of neural networks trained by GD via the *neural tangent kernel* (NTK): optimization trajectories of over-parameterized networks trained by GD are equivalent to those of kernel regression with a specific neural tangent kernel, under a set of assumptions called the “NTK regime” Jacot et al. (2018). We provide an informal definition here; for further details, we refer the readers to Jacot et al. Jacot et al. (2018) and Section 7.3.

Definition 3.3.1. (Informal) A neural network trained in the NTK regime is infinitely wide, randomly initialized with certain scaling, and trained by GD with infinitesimal steps.

Prior works analyze optimization and in-distribution generalization of over-parameterized neural networks via NTK Allen-Zhu et al. (2019a); Arora et al. (2019a,b); Du et al. (2019a,b); Li and Liang (2018); Allen-Zhu et al. (2019b); Cao and Gu (2019); Nitanda and Suzuki (2021). We instead analyze extrapolation.

Theorem 3.3.2 formalizes our observation from Figure 3.1: outside the training data range, along any direction tv from the origin, the prediction of a two-layer ReLU MLP quickly con-



Figure 3.3: **Conditions for ReLU MLPs to extrapolate well.** We train MLPs to learn linear functions (grey) with different training distributions (blue) and plot out-of-distribution predictions (black). Following Theorem 3.3.4, MLPs extrapolate well when the training distribution (blue) has support in all directions (first panel), but not otherwise: in the two middle panels, some dimensions of the training data are constrained to be positive (red arrows); in the last panel, one dimension is a fixed constant.

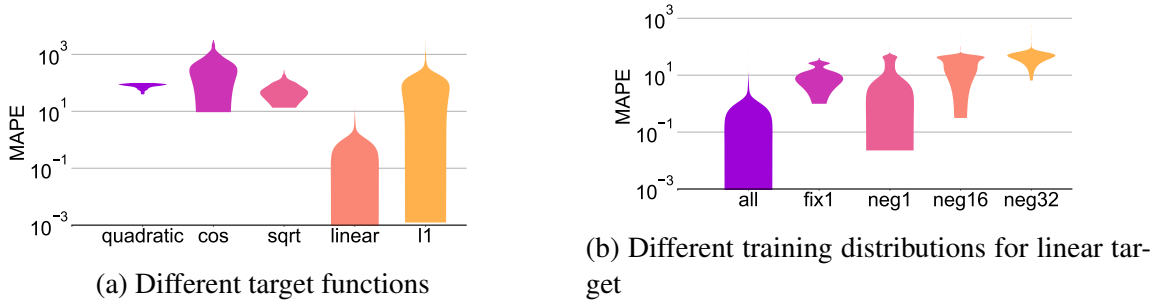


Figure 3.4: **Extrapolation performance of ReLU MLPs.** We plot the distributions of MAPE (mean absolute percentage error) of MLPs trained with various hyperparameters (depth, width, learning rate, batch size). (a) Learning different target functions; (b) Different training distributions for learning linear target functions: “all” covers all directions, “fix1” has one dimension fixed to a constant, and “neg d ” has d dimensions constrained to negative values. ReLU MLPs generally do not extrapolate well unless the target function is linear along each direction (Figure 3.4a), and extrapolate linear target functions if the training distribution covers sufficiently many directions (Figure 3.4b).

verges to a linear function with rate $O(\frac{1}{t})$. The linear coefficients β_v and the constant terms in the convergence rate depend on the training data and direction v . The proof is in Section 7.4.

Theorem 3.3.2. (Linear extrapolation). Suppose we train a two-layer ReLU MLP $f : \mathbb{R}^d \rightarrow \mathbb{R}$ with squared loss in the NTK regime. For any direction $v \in \mathbb{R}^d$, let $x_0 = tv$. As $t \rightarrow \infty$, $f(x_0 + hv) - f(x_0) \rightarrow \beta_v \cdot h$ for any $h > 0$, where β_v is a constant linear coefficient. Moreover, given $\epsilon > 0$, for $t = O(\frac{1}{\epsilon})$, we have $|\frac{f(x_0 + hv) - f(x_0)}{h} - \beta_v| < \epsilon$.

ReLU networks have finitely many linear regions [Arora et al. \(2018\)](#); [Hanin and Rolnick](#)

(2019), hence their predictions *eventually* become linear. In contrast, Theorem 3.3.2 is a more fine-grained analysis of *how* MLPs extrapolate and provides a convergence rate. While Theorem 3.3.2 assumes two-layer networks in the NTK regime, experiments confirm that the linear extrapolation behavior happens across networks with different depths, widths, learning rates, and batch sizes (Section 7.1.1 and 7.1.2). Our proof technique potentially also extends to deeper networks.

Theorem 3.3.2 implies which target functions a ReLU MLP may be able to match outside the training data: only functions that are almost-linear along the directions away from the origin. Indeed, Figure 3.4a shows ReLU MLPs do not extrapolate target functions such as $\mathbf{x}^\top A \mathbf{x}$ (quadratic), $\sum_{i=1}^d \cos(2\pi \cdot \mathbf{x}^{(i)})$ (cos), and $\sum_{i=1}^d \sqrt{\mathbf{x}^{(i)}}$ (sqrt), where $\mathbf{x}^{(i)}$ is the i -th dimension of $\mathbf{x}$. With suitable hyperparameters, MLPs extrapolate the L1 norm correctly, which satisfies the directional linearity condition.

Figure 3.4a provides one more positive result: MLPs extrapolate linear target functions well, across many different hyperparameters. While learning linear functions may seem very limited at first, in Section 3.4 this insight will help explain extrapolation properties of GNNs in non-linear practical tasks. Before that, we first theoretically analyze when MLPs extrapolate well.

3.3.2 When ReLU MLPs Provably Extrapolate Well

Figure 3.4a shows that MLPs can extrapolate well when the target function is linear. However, this is not always true. In this section, we show that successful extrapolation depends on the *geometry* of training data. Intuitively, the training distribution must be “diverse” enough for

correct extrapolation.

We provide two conditions that relate the geometry of the training data to extrapolation. Lemma 3.3.3 states that over-parameterized MLPs can learn a linear target function with only $2d$ examples.

Lemma 3.3.3. *Let $g(\mathbf{x}) = \boldsymbol{\beta}^\top \mathbf{x}$ be the target function for $\boldsymbol{\beta} \in \mathbb{R}^d$. Suppose $\{\mathbf{x}_i\}_{i=1}^n$ contains an orthogonal basis $\{\hat{\mathbf{x}}_i\}_{i=1}^d$ and $\{-\hat{\mathbf{x}}_i\}_{i=1}^d$. If we train a two-layer ReLU MLP f on $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$ with squared loss in the NTK regime, then $f(\mathbf{x}) = \boldsymbol{\beta}^\top \mathbf{x}$ for all $\mathbf{x} \in \mathbb{R}^d$.*

Lemma 3.3.3 is mainly of theoretical interest, as the $2d$ examples need to be carefully chosen. Theorem 3.3.4 builds on Lemma 3.3.3 and identifies a more practical condition for successful extrapolation: if the support of the training distribution covers all directions (e.g., a hypercube that covers the origin), the MLP converges to a linear target function with sufficient training data.

Theorem 3.3.4. *(Conditions for extrapolation). Let $g(\mathbf{x}) = \boldsymbol{\beta}^\top \mathbf{x}$ be the target function for $\boldsymbol{\beta} \in \mathbb{R}^d$. Suppose $\{\mathbf{x}_i\}_{i=1}^n$ is sampled from a distribution whose support $\mathcal{D}$ contains a connected subset $\mathcal{S}$, where for any non-zero $\mathbf{w} \in \mathbb{R}^d$, there exists $k > 0$ so that $k\mathbf{w} \in \mathcal{S}$. If we train a two-layer ReLU MLP $f : \mathbb{R}^d \rightarrow \mathbb{R}$ on $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$ with squared loss in the NTK regime, $f(\mathbf{x}) \xrightarrow{p} \boldsymbol{\beta}^\top \mathbf{x}$ as $n \rightarrow \infty$.*

Experiments: geometry of training data affects extrapolation. The condition in Theorem 3.3.4 formalizes the intuition that the training distribution must be “diverse” for successful extrapolation, e.g., $\mathcal{D}$ includes all directions. Empirically, the extrapolation error is indeed small when the condition of Theorem 3.3.4 is satisfied (“all” in Figure 3.4b). In contrast, the extrapolation error is much larger when the training examples are restricted to only some directions

(Figure 3.4b and Figure 3.3).

Relating to previous works, Theorem 3.3.4 suggests why spurious correlations may hurt extrapolation, complementing the causality arguments Arjovsky et al. (2019); Peters et al. (2016); Rojas-Carulla et al. (2018). When the training data has spurious correlations, some combinations of features are missing; e.g., camels might only appear in deserts in an image collection. Therefore, the condition for Theorem 3.3.4 no longer holds, and the model may extrapolate incorrectly. Theorem 3.3.4 is also analogous to an identifiability condition for linear models, but stricter. We can uniquely identify a linear function if the training data has full (feature) rank. MLPs are more expressive, so identifying the linear target function requires additional constraints.

To summarize, we analyze how ReLU MLPs extrapolate and provide two insights: (1) MLPs cannot extrapolate most nonlinear tasks due to their linear extrapolation (Theorem 3.3.2); and (2) MLPs extrapolate well when the target function is linear, if the training distribution is “diverse” (Theorem 3.3.4). In the next section, these results help us understand how more complex networks extrapolate.

3.3.3 MLPs with Other Activation Functions

Before moving on to GNNs, we complete the picture of MLPs with experiments on other activation functions: $\tanh \sigma(x) = \tanh(x)$, cosine $\sigma(x) = \cos(x)$ Lapedes and Farber (1987); McCaughan (1997); Sopena and Alquezar (1994), and quadratic $\sigma(x) = x^2$ Du and Lee (2018); Livni et al. (2014). Details are in Appendix 7.1.4. MLPs extrapolate well when the activation and target function are similar; e.g., $\tanh$ activation extrapolates well when learning $\tanh$, but not other functions (Figure 3.5). Moreover, each activation function has different limitations. To

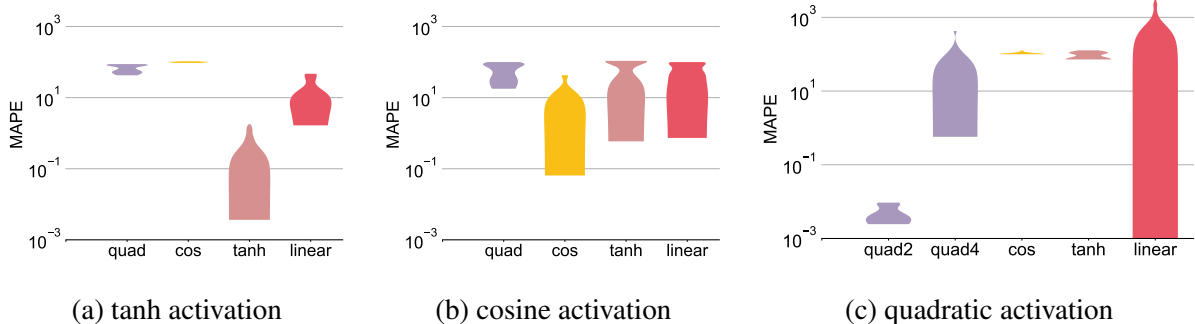


Figure 3.5: **Extrapolation performance of MLPs with other activation.** MLPs can extrapolate well when the activation is “similar” to the target function. When learning quadratic with quadratic activation, 2-layer networks (quad-2) extrapolate well, but 4-layer networks (quad-4) do not.

extrapolate the tanh function with tanh activation, the training data range has to be sufficiently wide. When learning a quadratic function with quadratic activation, only two-layer networks extrapolate well as more layers lead to higher-order polynomials. Cosine activations are hard to optimize for high-dimensional data, so we only consider one/two dimensional cosine target functions.

3.4 How Graph Neural Networks Extrapolate

Above, we saw that extrapolation in nonlinear tasks is hard for MLPs. Despite this limitation, GNNs have been shown to extrapolate well in some nonlinear algorithmic tasks, such as intuitive physics [Battaglia et al. \(2016\)](#); [Janner et al. \(2019\)](#), graph algorithms [Battaglia et al. \(2018\)](#); [Velickovic et al. \(2020\)](#), and symbolic mathematics [Lample and Charton \(2020\)](#). To address this discrepancy, we build on our MLP results and study how GNNs trained by GD extrapolate.

3.4.1 Hypothesis: Linear Algorithmic Alignment Helps Extrapolation

We start with an example: training GNNs to solve the shortest path problem. For this task, prior works observe that a modified GNN architecture with min-aggregation can generalize to graphs larger than those in the training set [Battaglia et al. \(2018\)](#); [Velickovic et al. \(2020\)](#):

$$\mathbf{h}_u^{(k)} = \min_{v \in \mathcal{N}(u)} \text{MLP}^{(k)}(\mathbf{h}_u^{(k-1)}, \mathbf{h}_v^{(k-1)}, \mathbf{w}_{(v,u)}). \quad (2)$$

We first provide an intuitive explanation (Figure [3.2a](#)). Shortest path can be solved by the Bellman-Ford (BF) algorithm [Bellman \(1958\)](#) with the following update:

$$d[k][u] = \min_{v \in \mathcal{N}(u)} d[k-1][v] + \mathbf{w}(v, u), \quad (3)$$

where $\mathbf{w}(v, u)$ is the weight of edge (v, u) , and $d[k][u]$ is the shortest distance to node u within k steps. The two equations can be easily aligned: GNNs simulate the BF algorithm if its MLP modules learn a linear function $d[k-1][v] + \mathbf{w}(v, u)$. Since MLPs can extrapolate linear tasks, this “alignment” may explain why min-aggregation GNNs can extrapolate well in this task.

For comparison, we can reason why we would not expect GNNs with the more commonly used sum-aggregation (Eqn. [1](#)) to extrapolate well in this task. With sum-aggregation, the MLP modules need to learn a nonlinear function to simulate the BF algorithm, but Theorem [3.3.2](#) suggests that they will not extrapolate most nonlinear functions outside the training support.

We can generalize the above intuition to other algorithmic tasks. Many tasks where GNNs extrapolate well can be solved by dynamic programming (DP) [Bellman \(1966\)](#), an algorithmic

paradigm with a recursive structure similar to GNNs’ (Eqn. 1 in chapter 2).

Definition 3.4.1. Dynamic programming (*DP*) is a recursive procedure with updates

$$\text{Answer}[k][s] = \text{DP-Update}(\{\text{Answer}[k-1][s']\}, s' = 1 \dots n), \quad (4)$$

where $\text{Answer}[k][s]$ is the solution to a sub-problem indexed by iteration k and state s , and *DP-Update* is a task-specific update function that solves the sub-problem based on the previous iteration.

From a broader standpoint, we hypothesize that: if we encode appropriate non-linearities into the model architecture and input representations so that the MLP modules only need to learn nearly linear steps, then the resulting neural network can extrapolate well.

Hypothesis 3.4.2. (*Linear algorithmic alignment*). Let $f : \mathcal{X} \rightarrow \mathbb{R}$ be the underlying function and $\mathcal{N}$ a neural network with m MLP modules. Suppose there exist m linear functions $\{g_i\}_{i=1}^m$ so that by replacing $\mathcal{N}$ ’s MLP modules with g_i ’s, $\mathcal{N}$ simulates f . Given $\epsilon > 0$, there exists $\{(x_i, f(x_i))\}_{i=1}^n \subset \mathcal{D} \subsetneq \mathcal{X}$ so that $\mathcal{N}$ trained on $\{(x_i, f(x_i))\}_{i=1}^n$ by GD with squared loss learns $\hat{f}$ with $\|\hat{f} - f\| < \epsilon$.

Our hypothesis builds on the algorithmic alignment framework stated in chapter 2, which states that a neural network *interpolates* well if the modules are “aligned” to easy-to-learn (possibly nonlinear) functions. Successful extrapolation is harder: the modules need to align with linear functions.

Applications of linear algorithmic alignment. In general, linear algorithmic alignment is not restricted to GNNs and applies broadly to neural networks. To satisfy the condition, we can

encode appropriate nonlinear operations in the *architecture* or *input representation* (Figure 2.1). Learning DP algorithms with GNNs is one example of encoding non-linearity in the architecture Battaglia et al. (2018); Corso et al. (2020). Another example is to encode log-and-exp transforms in the architecture to help extrapolate multiplication in arithmetic tasks Trask et al. (2018); Madsen and Johansen (2020). Neural symbolic programs take a step further and encode a library of symbolic operations to help extrapolation Yi et al. (2018); Mao et al. (2019a); Johnson et al. (2017b).

For some tasks, it may be easier to change the input representation (Figure 3.2b). Sometimes, we can decompose the target function f as $f = g \circ h$ into a feature embedding h and a “simpler” target function g that our model can extrapolate well. We can obtain h via specialized features or feature transforms using *domain knowledge* Lample and Charton (2020); Webb et al. (2020), or via *representation learning* (e.g., BERT) with unlabeled out-of-distribution data in $\mathcal{X} \setminus \mathcal{D}$ Chen et al. (2020); Devlin et al. (2019); Hu et al. (2020a); Mikolov et al. (2013a); Peters et al. (2018). This brings a new perspective of how representations help extrapolation in various application areas. For example, in natural language processing, pretrained representations Mikolov et al. (2013b); Wu and Dredze (2019) and feature transformation using domain knowledge Yuan et al. (2020); Zhang et al. (2019b) help models generalize across languages, a special type of extrapolation. In quantitative finance, identifying the right “factors” or features is crucial for deep learning models as the financial markets may frequently be in extrapolation regimes Banz (1981); Fama and French (1993); Ross (1976).

Linear algorithmic alignment explains successful extrapolation in the literature and suggests that extrapolation is harder in general: encoding appropriate non-linearity often requires domain expertise or model search. Next, we provide theoretical and empirical support for our

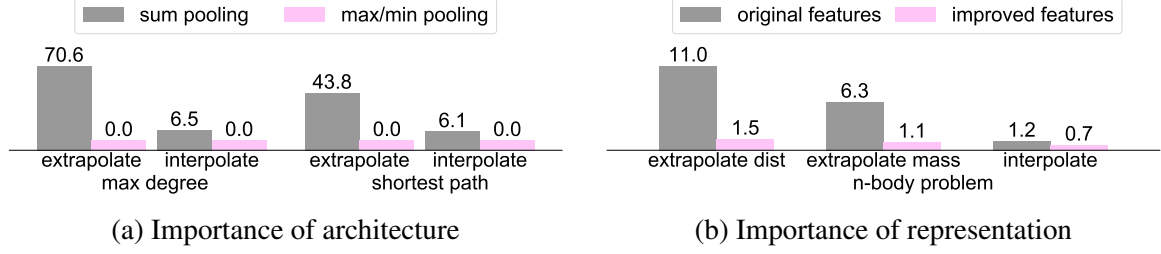


Figure 3.6: **Extrapolation for algorithmic tasks.** Each column indicates the task and mean average percentage error (MAPE). Encoding appropriate non-linearity in the architecture or representation is less helpful for *interpolation*, but significantly improves *extrapolation*. Left: In max degree and shortest path, GNNs that appropriately encode max/min extrapolate well, but GNNs with sum-pooling do not. Right: With improved input representation, GNNs extrapolate better for the n -body problem.

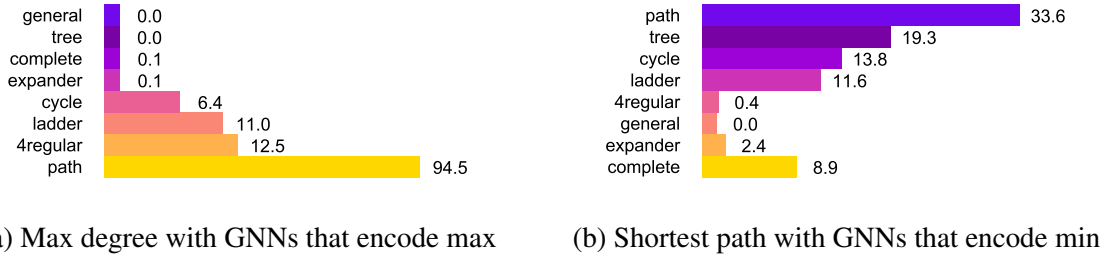


Figure 3.7: **Importance of the training graph structure.** Rows indicate the graph structure covered by the training set and the extrapolation error (MAPE). In max degree, GNNs with max readout extrapolate well if the max/min degrees of the training graphs are not restricted (Theorem 3.4.4). In shortest path, the extrapolation errors of min GNNs follow a U-shape in the sparsity of the training graphs. More results may be found in Appendix 7.2.2.

hypothesis.

3.4.2 Theoretical and Empirical Support

We validate our hypothesis on three DP tasks: max degree, shortest path, and n -body problem, and prove the hypothesis for max degree. We highlight the role of graph structures in extrapolation.

Theoretical analysis. We start with a simple yet fundamental task: learning the max degree of a graph, a special case of DP with one iteration. As a corollary of Theorem 3.3.2,

the commonly used sum-based GNN (Eqn. 1) cannot extrapolate well (proof in Appendix 7.4.6).

Corollary 3.4.3. *GNNs with sum-aggregation and sum-readout do not extrapolate well in Max Degree.*

To achieve linear algorithmic alignment, we can encode the only non-linearity, the max function, in the readout. Theorem 3.4.4 confirms that a GNN with max-readout can extrapolate well in this task.

Theorem 3.4.4. *(Extrapolation with GNNs). Assume all nodes have the same feature. Let g and g' be the max/min degree function, respectively. Let $\{(G_i, g(G_i))\}_{i=1}^n$ be the training set. If $\{(g(G_i), g'(G_i), g(G_i) \cdot N_i^{\max}, g'(G_i) \cdot N_i^{\min})\}_{i=1}^n$ spans $\mathbb{R}^4$, where $N_i^{\max}$ and $N_i^{\min}$ are the number of nodes that have max/min degree on G_i , then one-layer max-readout GNNs trained on $\{(G_i, g(G_i))\}_{i=1}^n$ with squared loss in the NTK regime learn g .*

Theorem 3.4.4 does not follow immediately from Theorem 3.3.4, because MLP modules in GNNs only receive indirect supervision. We analyze the *Graph NTK* Du et al. (2019c) to prove Theorem 3.4.4 in Appendix 7.4.7. While Theorem 3.4.4 assumes identical node features, we empirically observe similar results for both identical and non-identical features (Figure 7.9 in Appendix).

Interpretation of conditions. The condition in Theorem 3.4.4 is analogous to that in Theorem 3.3.4. Both theorems require diverse training data, measured by *graph structure* in Theorem 3.4.4 or *directions* in Theorem 3.3.4. In Theorem 3.4.4, the condition is violated if all training graphs have the same max or min node degrees, e.g., when training data are from one of the following families: path, C -regular graphs (regular graphs with degree C), cycle, and ladder.

Experiments: architectures that help extrapolation. We validate our theoretical analysis with two DP tasks: max degree and shortest path (details in Appendix 7.1.5 and 7.1.6). While previous works only test on graphs with different sizes Battaglia et al. (2018); Velickovic et al. (2020), we also test on graphs with unseen structure, edge weights and node features. The results support our theory. For max degree, GNNs with max-readout are better than GNNs with sum-readout (Figure 3.6a), confirming Corollary 3.4.3 and Theorem 3.4.4. For shortest path, GNNs with min-readout and min-aggregation are better than GNNs with sum-readout (Figure 3.6a).

Experiments confirm the importance of training graphs structure (Figure 3.7). Interestingly, the two tasks favor different graph structure. For max degree, as Theorem 3.4.4 predicts, GNNs extrapolate well when trained on trees, complete graphs, expanders, and general graphs, and extrapolation errors are higher when trained on 4-regular, cycles, or ladder graphs. For shortest path, extrapolation errors follow a U-shaped curve as we change the *sparsity* of training graphs (Figure 3.7b and Figure 7.11 in Appendix). Intuitively, models trained on sparse or dense graphs likely learn degenerative solutions.

Experiments: representations that help extrapolation. Finally, we show a good input representation helps extrapolation. We study the n -body problem Battaglia et al. (2016); Watters et al. (2017) (Appendix 7.1.7), that is, predicting the time evolution of n objects in a gravitational system. Following previous work, the input is a complete graph where the nodes are the objects Battaglia et al. (2016). The node feature for u is the concatenation of the object’s mass m_u , position $\mathbf{x}_u^{(t)}$, and velocity $\mathbf{v}_u^{(t)}$ at time t . The edge features are set to zero. We train GNNs to predict the velocity of each object u at time $t + 1$. The true velocity $f(G; u)$ for object u is

approximately

$$f(G; u) \approx \mathbf{v}_u^t + \mathbf{a}_u^t \cdot dt, \quad \mathbf{a}_u^t = C \cdot \sum_{v \neq u} \frac{m_v}{\|\mathbf{x}_u^t - \mathbf{x}_v^t\|_2^3} \cdot (\mathbf{x}_v^t - \mathbf{x}_u^t), \quad (5)$$

where C is a constant. To learn f , the MLP modules need to learn a nonlinear function. Therefore, GNNs do not extrapolate well to unseen masses or distances (“original features” in Figure 3.6b). We instead use an improved representation $h(G)$ to encode non-linearity. At time t , we transform the edge features of (u, v) from zero to $\mathbf{w}_{(u,v)}^{(t)} = m_v \cdot (\mathbf{x}_v^{(t)} - \mathbf{x}_u^{(t)}) / \|\mathbf{x}_u^{(t)} - \mathbf{x}_v^{(t)}\|_2^3$. The new edge features do not add information, but the MLP modules now only need to learn linear functions, which helps extrapolation (“improved features” in Figure 3.6b).

3.5 Connections to Other Out-of-Distribution Settings

We discuss several related settings. Intuitively, from the viewpoint of our results above, methods in related settings may improve extrapolation by 1) learning useful non-linearities beyond the training data range and 2) mapping relevant test data to the training data range.

Domain adaptation studies generalization to a specific target domain [Ben-David et al. \(2010\)](#); [Blitzer et al. \(2008\)](#); [Mansour et al. \(2009\)](#). Typical strategies adjust the training process: for instance, use unlabeled samples from the target domain to align the target and source distributions [Ganin et al. \(2016\)](#); [Zhao et al. \(2018\)](#). Using target domain data during training may induce useful non-linearities and may mitigate extrapolation by matching the target and source distributions, though the correctness of the learned mapping depends on the label distribution [Zhao et al. \(2019\)](#).

Self-supervised learning on a large amount of unlabeled data can learn useful non-linearities

beyond the labeled training data range [Chen et al. \(2020\)](#); [Devlin et al. \(2019\)](#); [Peters et al. \(2018\)](#); [He et al. \(2020\)](#). Hence, our results suggest an explanation why pre-trained representations such as BERT improve out-of-distribution robustness [Hendrycks et al. \(2020\)](#). In addition, self-supervised learning could map semantically similar data to similar representations, so some out-of-domain examples might fall inside the training distribution after the mapping.

Invariant models aim to learn features that respect specific invariances across multiple training distributions [Arjovsky et al. \(2019\)](#); [Rojas-Carulla et al. \(2018\)](#); [Zhou et al. \(2021\)](#). If the model indeed learns these invariances, which can happen in the linear case and when there are confounders or anti-causal variables [Ahuja et al. \(2021\)](#); [Rosenfeld et al. \(2021\)](#), this may essentially increase the training data range, since variations in the invariant features may be ignored by the model.

Distributional robustness considers small adversarial perturbations of the data distribution, and ensures that the model performs well under these [Goh and Sim \(2010\)](#); [Sagawa et al. \(2020\)](#); [Sinha et al. \(2018\)](#); [Staib and Jegelka \(2019\)](#). We instead look at more global perturbations. Still, one would expect that modifications that help extrapolation in general also improve robustness to local perturbations.

3.6 Chapter Summary

This chapter is an initial step towards formally understanding how neural networks trained by gradient descent extrapolate. We identify conditions under which MLPs and GNNs extrapolate as desired. We also suggest an explanation how GNNs have been able to extrapolate well in complex algorithmic tasks: encoding appropriate non-linearity in architecture and features can

help extrapolation. Our results and hypothesis agree with empirical results, in this chapter and in the literature.

My main contribution to this chapter includes initial exploration experiments which discovered our main finding (a.k.a., linear alignment), experiments (design and execution) in section 3.3 and section 3.4, interpretation of Theorem 3.4.4, and additional experiments in (Appendix) Chapter 7. Since the publication of this chapter, numerous follow-up studies have emerged, employing the key insight of encoding non-linearities in the data representations [Ibarz et al. \(2022\)](#); [Na and Park \(2022\)](#); [Li et al. \(2023\)](#) or neural network architectures [Ko et al. \(2021\)](#) for better out-of-distribution generalization.

Chapter 4: Robustness: How does the Architecture Impact a Neural Network’s Robustness?

Joint work with Keyulu Xu, Mozhi Zhang, John P Dickerson, Jimmy Ba

4.1 Introduction

Supervised learning starts with collecting labeled data. Yet, high-quality labels are often expensive. To reduce annotation cost, we collect labels from non-experts [Snow et al. \(2008\)](#); [Welinder et al. \(2010\)](#); [Yan et al. \(2014\)](#); [Yu et al. \(2018\)](#) or online queries [Blum et al. \(2003\)](#); [Jiang et al. \(2020a\)](#); [Liu et al. \(2011\)](#), which are inevitably noisy. To learn from these noisy labels, previous works propose many techniques, including modeling the label noise [Natarajan et al. \(2013\)](#); [Liu and Tao \(2015\)](#); [Yao et al. \(2020\)](#), designing robust losses [Ghosh et al. \(2017\)](#); [Lyu and Tsang \(2019\)](#); [Wang et al. \(2019\)](#); [Zhang and Sabuncu \(2018\)](#), adjusting loss before gradient updates [Arazo et al. \(2019\)](#); [Chang et al. \(2017b\)](#); [Han et al. \(2020\)](#); [Hendrycks et al. \(2018\)](#); [Ma et al. \(2018\)](#); [Patrini et al. \(2017\)](#); [Reed et al. \(2014\)](#); [Song et al. \(2019\)](#); [Wang et al. \(2017\)](#), selecting trust-worthy samples [Lyu and Tsang \(2019\)](#); [Song et al. \(2019\)](#); [Chen et al. \(2019\)](#); [Han et al. \(2018a\)](#); [Jiang et al. \(2018\)](#); [Malach and Shalev-Shwartz \(2017\)](#); [Nguyen et al. \(2019\)](#); [Shen and Sanghavi \(2019\)](#); [Wang et al. \(2018\)](#); [Yu et al. \(2019\)](#), designing robust architectures [Bekker and Goldberger \(2016\)](#); [Chen and Gupta \(2015\)](#); [Goldberger and Ben-Reuven \(2017\)](#); [Han et al.](#)

(2018b); Jindal et al. (2016); Li et al. (2020a); Sukhbaatar et al. (2014); Yao et al. (2018), applying robust regularization in training Goodfellow et al. (2015); Hendrycks et al. (2019); Jenni and Favaro (2018); Pereyra et al. (2017); Tanno et al. (2019); Zhang et al. (2017b), using meta-learning to avoid over-fitting Garcia et al. (2016); Li et al. (2019), and applying semi-supervised learning Nguyen et al. (2019); Ding et al. (2018); Li et al. (2020b); Liu et al. (2020a); Yan et al. (2016) to learn better representations.

While these methods improve some networks’ robustness to noisy labels, we observe that their effectiveness depends on how well the network’s architecture aligns with the target/noise functions, and they are less effective when encountering more realistic label noise that is class-dependent or instance-dependent. This motivates us to investigate an understudied topic: how the network’s architecture impacts its robustness to noisy labels.

We formally answer this question by analyzing how a network’s architecture aligns with the target function and the noise. To start, we measure the robustness of a network via the predictive power in its learned representations (Definition 4.2.1), as models with large test errors may still learn useful predictive hidden representations Arpit et al. (2017); Maennel et al. (2020). Intuitively, the predictive power measures how well the representations can predict the target function. In practice, we measure it by training a linear model on top of the learned representations using a small set of clean labels and evaluate the linear model’s test performance Alain and Bengio (2016).

We find that a network having a more aligned architecture with the target function is more robust to noisy labels due to its more predictive representations, whereas a network having an architecture more aligned with the noise function is less robust. Intuitively, a *good* alignment between a network’s architecture and a function exists if the architecture can be decomposed into

several modules such that each module can simulate one part of the function with a *small* sample complexity. The formal definition of alignment is in Section 4.2.3, adapted from chapter 2.

Our proposed framework provides initial theoretical support for our findings on a simplified noisy setting (Theorem 4.2.6). Empirically, we validate our findings on synthetic graph algorithmic tasks by designing several variants of Graph Neural Networks (GNNs), whose theoretical properties and alignment with algorithmic functions have been well-studied in Du et al. (2019c) and both chapters 2 and 3. Many noisy label training methods are applied to image classification datasets, so we also validate our findings on image domains using different architectures.

Most of our analysis and experiments use standard neural network training. Interestingly, we find similar results when using DivideMix Li et al. (2020b), a SOTA method for learning with noisy labels: for networks less aligned with the target function, the SOTA method barely helps and sometimes even hurts test accuracy; whereas for more aligned networks, it helps greatly.

For well-aligned networks, the predictive power of their learned representation could further improve the test performance of SOTA methods, especially on class-dependent or instance-dependent label noise where current methods on noisy label training are less effective. Moreover, on Clothing1M Xiao et al. (2015), a large-scale dataset with real-world label noise, the predictive power of a well-aligned network’s learned representations could even outperform some sophisticated methods that use clean labels.

In summary, we investigate how an architecture’s alignments with different (target and noise) functions affect the network’s robustness to noisy labels, in which we discover that despite having large test errors, networks well-aligned with the target function can still be robust to noisy labels when evaluating their predictive power in learned representations. To formalize our finding, we provide a theoretical framework to illustrate the above connections. At the same

time, we conduct empirical experiments on various datasets with various network architectures to validate this finding. Besides, this finding further leads to improvements over SOTA noisy-label-training methods on various datasets and under various kinds of noisy labels (Tables 8.3-8.8 in Appendix 8.1).

Related Work A commonly studied type of noisy label is the random label noise, where the noisy labels are drawn i.i.d. from a uniform distribution. While neural networks trained with random labels easily overfit [Zhang et al. \(2017a\)](#), it has been observed that networks learn simple patterns first [Arpit et al. \(2017\)](#), converge faster on downstream tasks [Maennel et al. \(2020\)](#), and benefit from memorizing atypical training samples [Feldman and Zhang \(2020\)](#).

Accordingly, many recent works on noisy label training are based on the assumption that when trained with noisy labels, neural networks would first fit to clean labels [Lyu and Tsang \(2019\)](#); [Han et al. \(2018a\)](#); [Jiang et al. \(2018\)](#); [Li et al. \(2020b\)](#); [Liu et al. \(2020a\)](#) and learn useful feature patterns [Hendrycks et al. \(2018\)](#); [Lee et al. \(2019\)](#); [Bahri et al. \(2020\)](#); [Wu et al. \(2020\)](#). Yet, these methods are often more effective on random label noise than on more realistic label noise (i.e., class-dependent and instance-dependent label noise).

Many works on representation learning have investigated the features preferred by a network during training [Arpit et al. \(2017\)](#); [Hermann and Lampinen \(2020\)](#); [Shah et al. \(2020\)](#); [Sanyal et al. \(2020\)](#), and how to interpret or control the learned representations on clean data [Yuan et al. \(2020\)](#); [Alain and Bengio \(2016\)](#); [Hermann and Lampinen \(2020\)](#); [Hermann et al. \(2019\)](#); [Montavon et al. \(2018\)](#). This chapter focuses more on the predictive power rather than the explanatory power in the learned representations. We adapt the method in [Alain and Bengio \(2016\)](#) to measure the predictive power in representations, and we study learning from noisy labels rather

than from a clean distribution.

On noiseless settings, prior works show that neural networks have the inductive bias to learn simple patterns [Arpit et al. \(2017\)](#); [Hermann and Lampinen \(2020\)](#); [Shah et al. \(2020\)](#); [Sanyal et al. \(2020\)](#). This chapter formalizes what a simple pattern is to a given network via architectural alignments, and we extend the definition of alignment in chapter 2 to noisy settings.

4.2 Theoretical Framework

In this section, we introduce our problem settings, give formal definitions for “predictive power” and “alignment,” and present our main hypothesis as well as our main theorem.

4.2.1 Problem Settings

Let $\mathcal{X}$ denote the input domain, which can be vectors, images, or graphs. The task is to learn an underlying target function $f : \mathcal{X} \rightarrow \mathcal{Y}$ on a noisy training dataset $S := \{(\mathbf{x}_i, y_i)\}_{i \in \mathcal{I}} \cup \{(\mathbf{x}_i, \hat{y}_i)\}_{i \in \mathcal{I}'}$, where $y := f(\mathbf{x})$ denotes the true label for an input $\mathbf{x}$, and $\hat{y}$ denotes the noisy label. Here, the set $\mathcal{I}$ contains indices with clean labels, and $\mathcal{I}'$ contains indices with noisy labels. We denote $\frac{|\mathcal{I}'|}{|S|}$ as the *noise ratio* in the dataset S . We consider both regression and classification problems.

Regression settings. We consider a label space $\mathcal{Y} \subseteq \mathbb{R}$ and two types of label noise: a) **additive label noise** [Hu et al. \(2020b\)](#): $\hat{y} := y + \epsilon$, where ϵ is a random variable independent from $\mathbf{x}$; b) **instance-dependent label noise**: $\hat{y} := g(\mathbf{x})$ where $g : \mathcal{X} \rightarrow \mathcal{Y}$ is a noise function dependent on the input.

Classification settings. We consider a discrete label space with C classes: $\mathcal{Y} = \{1, 2, \dots, C\}$, and three types of label noise: a) **uniform label noise**: $\hat{y} \sim \text{Unif}(1, C)$, where the noisy label is

drawn from a discrete uniform distribution with values between 1 and C , and thus is independent of the true label; b) **flipped label noise**: $\hat{y}$ is generated based on the value of the true label y and does not consider other input structures; c) **instance-dependent label noise**: $\hat{y} := g(\mathbf{x})$ where $g : \mathcal{X} \rightarrow \mathcal{Y}$ is a function dependent on the input $\mathbf{x}$'s internal structures. Previous works on noisy label learning commonly study uniform and flipped label noise. A few recent works [Cheng et al. \(2020\)](#); [Wang et al. \(2020\)](#) explore the instance-dependent label noise as it is more realistic.

4.2.2 Predictive Power in Representations

A network's robustness is often measured by its test performance after being trained with noisy labels. Yet, since models with large test errors may still learn useful representations, we measure the robustness of a network by how good the learned representations are at predicting the target function — the predictive power in representations. To formalize this definition, we decompose a neural network $\mathcal{N}$ into different modules $\mathcal{N}_1, \mathcal{N}_2, \dots$, where each module can be a single layer (e.g., a convolutional layer) or a block of layers (e.g., a residual block).

Definition 4.2.1 (Predictive power). *Let $f : \mathcal{X} \rightarrow \mathcal{Y}$ denote the underlying target function where the input $\mathbf{x} \in \mathcal{X}$ is drawn from a distribution $\mathcal{D}$. Let $\mathcal{C} := \{(\mathbf{x}_i, y_i)\}_{i=1}^m$ denote a small set of clean data (i.e., $y_i = f(\mathbf{x}_i)$). Given a network $\mathcal{N}$ with n modules $\mathcal{N}_j$, let $h^{(j)}(\mathbf{x})$ denote the representation from module $\mathcal{N}_j$ on the input $\mathbf{x}$ (i.e., the output of $\mathcal{N}_j$). Let L denote the linear model trained with the clean set $\mathcal{C}$ where we use $h^{(j)}(\mathbf{x})$ as the input, and y_i as the target value during training. Then the predictive power of representations from the module $\mathcal{N}_i$ is defined as*

$$P_j(f, \mathcal{N}, \mathcal{C}) := \mathbb{E}_{\mathbf{x} \sim \mathcal{D}} [l(f(\mathbf{x}), L(h^{(j)}(\mathbf{x})))], \quad (1)$$

where l is a loss function used to evaluate the test performance on the learning task.

Remark. Notice that smaller $P_j(f, \mathcal{N}, C)$ indicates better predictive power; i.e., the representations are better at predicting the target function. We empirically evaluate the predictive power using linear regression to obtain a trained linear model L , which avoids the issue of local minima as we are solving a convex problem; then we evaluate L on the test set.

4.2.3 Formalization of Alignment

Our analysis stems from the intuition that a network would be more robust to noisy labels if it could learn the target function more easily than the noise function. Thus, we use architectural alignment to formalize what is easy to learn by a given network. Chapter 2 defines the alignment between a network and a deterministic function via a sample complexity measure in a PAC learning framework. Intuitively, a network aligns well with a function if each network module can easily learn one part of the function with a small sample complexity.

Definition 4.2.2 (Alignment, simplified according on chapter 2). Let $\mathcal{N}$ denote a neural network with n modules $\mathcal{N}_j$. Given a function $f : \mathcal{X} \rightarrow \mathcal{Y}$ which can be decomposed into n functions f_j (e.g., $f(\mathbf{x}) = f_1(f_2(\dots f_n(\mathbf{x})))$), the alignment between the network $\mathcal{N}$ and f is defined via

$$\text{Alignment}(\mathcal{N}, f, \epsilon, \delta) := \max_j \mathcal{M}_{A_j}(f_j, \mathcal{N}_j, \epsilon, \delta), \quad (2)$$

where $\mathcal{M}_{A_j}(f_j, \mathcal{N}_j, \epsilon, \delta)$ denotes the sample complexity measure for $\mathcal{N}_j$ to learn f_j with ϵ precision at a failure probability δ under a learning algorithm A_j .

Remark. Notice that smaller $\text{Alignment}(\mathcal{N}, f, \epsilon, \delta)$ indicates better alignment between network $\mathcal{N}$ and function f . If f is obtuse or does not have a structural decomposition, we can

choose $n = 1$, and the definition of alignment degenerates into the sample complexity measure for $\mathcal{N}$ to learn f . Although it is sometimes non-trivial to compute the exact alignment for a task without clear algorithmic structures, we could break this complicated task into sub-tasks, and it would be easier to measure the sample complexity of learning each sub-task.

Chapter 2 further proves that better alignment implies better sample complexity and vice versa.

Theorem 4.2.3 (Informal; Chapter 2). *Fix ϵ and δ . Given a target function $f : \mathcal{X} \rightarrow \mathcal{Y}$ and a network $\mathcal{N}$, suppose $\{x_i\}_{i=1}^M$ are i.i.d. samples drawn from a distribution $\mathcal{D}$, and let $y_i := f(x_i)$. Then $\text{Alignment}(\mathcal{N}, f, \epsilon, \delta) \leq M$ if and only if there exists a learning algorithm A such that*

$$\mathbb{P}_{x \sim \mathcal{D}}[\|f_{\mathcal{N}, A}(x) - f(x)\| \leq \epsilon] \geq 1 - \delta, \quad (3)$$

where $f_{\mathcal{N}, A}$ is the function generated by A on the training data $\{x_i, y_i\}_{i=1}^M$.

Remark. *Intuitively, a function f (with a decomposition $\{f_j\}_j$) can be efficiently learned by a network $\mathcal{N}$ (with modules $\{\mathcal{N}_j\}_j$) iff each f_j can be efficiently learned by $\mathcal{N}_j$.*

We further extend Definition 4.2.2 to work with a random process $\mathcal{F}$ (i.e., a set of all possible sample functions that describes the noisy label distribution).

Definition 4.2.4 (Alignment, extended to various noise functions). *Given a neural network $\mathcal{N}$ and a random process $\mathcal{F}$, for each $f \in \mathcal{F}$, the alignment between $\mathcal{N}$ and f is measured via $\max_j \mathcal{M}_{A_j}(f_j, \mathcal{N}_j, \epsilon, \delta)$ based on Definition 4.2.2. Then the alignment between $\mathcal{N}$ and $\mathcal{F}$ is defined as*

$$\text{Alignment}^*(\mathcal{N}, \mathcal{F}, \epsilon, \delta) := \sup_{f \in \mathcal{F}} \max_j \mathcal{M}_{A_j}(f_j, \mathcal{N}_j, \epsilon, \delta),$$

where $\mathcal{N}$ can be decomposed differently for various f .

4.2.4 Better Alignment Implies Better Robustness (Better Predictive Power)

Building on the definitions of *predictive power* and *alignment*, we hypothesize that a network better-aligned with the target function (smaller $\text{Alignment}(\mathcal{N}, f, \epsilon, \delta)$) would learn more predictive representations (smaller $P_j(f, \mathcal{N}, \mathcal{C})$) when trained on a given noisy dataset.

Hypothesis 4.2.5 (Better alignment leads to more robustness). *Let $f : \mathcal{X} \rightarrow \mathcal{Y}$ denote the target function. Fix ϵ, δ , a learning algorithm A , a noise ratio, and a noise function $g : \mathcal{X} \rightarrow \mathcal{Y}$ (which may be drawn from a random process). Let S denote a noisy training dataset and $\mathcal{C}$ denote a small set of clean data. Then for a network $\mathcal{N}$ trained on S with the learning algorithm A ,*

$$\text{Alignment}(\mathcal{N}, f, \epsilon, \delta) \downarrow \implies P_j(f, \mathcal{N}, \mathcal{C}) \downarrow, \quad (4)$$

where j is selected based on the network's architectural alignment with the target function (for simplicity, we consider $j = n - 1$ in this chapter).

We prove this hypothesis for a simplified case where the target function shares some common structures with the noise function (e.g., class-dependent label noise). We refer the readers to Appendix 8.2 for a full statement of our main theorem with detailed assumptions.

Theorem 4.2.6 (Better alignment leads to more robustness, specific case). *For a target function $f : \mathcal{X} \rightarrow \mathcal{Y}$ and a noise function $g : \mathcal{X} \rightarrow \mathcal{Y}$, consider a neural network $\mathcal{N}$ well-aligned with f such that $P_j(f, \mathcal{N}, \mathcal{C})$ is small when training $\mathcal{N}$ on clean data (i.e., $P_j(f, \mathcal{N}, \mathcal{C}) < c$ for some small constant c). If there exists a function h on the input domain $\mathcal{X}$ such that f and g*

can be decomposed as follows: $\forall x \in \mathcal{X}, f(x) = f_r(h(x))$ with f_r being a linear function, and $g(x) = g_r(h(x))$ for some function g_r , then the representations learned by $\mathcal{N}$ on the noisy dataset still have a good predictive power with $P_j(f, \mathcal{N}, \mathcal{C}) < c$.

We further provide empirical support for our hypothesis via systematic experiments on various architectures, target and noise functions on both regression and classification settings.

4.3 Experiments on Graph Neural Networks

We first validate our hypothesis on synthetic graph algorithmic tasks by designing GNNs with different levels of alignments to the underlying target/noise functions. We consider regression tasks. The theoretical properties of GNNs and their alignment with algorithmic regression tasks are well-studied in [Du et al. \(2019c\)](#); [Sato et al. \(2019\)](#) and in both chapters 2 and 3. To start, we conduct experiments on different types of additive label noise and extend our experiments to instance-dependent label noise, which is closer to real-life noisy labels.

Common Experimental Settings. The training and validation sets always have the same noise ratio, the percentage of data with noisy labels. We choose mean squared error (MSE) and Mean Absolute Error (MAE) as our loss functions. Due to space limitations, the results using MAE are in Appendix 8.1.3. All training details are in Appendix ???. The test error is measured by mean absolute percentage error (MAPE), a relative error metric.

4.3.1 Background: Graph Neural Networks

GNNs are structured networks operating on graphs with MLP modules [Battaglia et al. \(2018\)](#); [Scarselli et al. \(2009a\)](#); [Xu et al. \(2019, 2018\)](#). The input is a graph $\mathcal{G} = (V, E)$ where

each node $u \in V$ has a feature vector $\mathbf{x}_u$, and we use $\mathcal{N}(u)$ to denote the set of neighbors of u . GNNs iteratively compute the node representations via message passing: (1) the node representation $\mathbf{h}_u$ is initialized as the node feature: $\mathbf{h}_u^{(0)} = \mathbf{x}_u$; (2) in iteration $k = 1..K$, the node representations $\mathbf{h}_u^{(k)}$ are updated by aggregating the neighboring nodes' representations with MLP modules [Gilmer et al. \(2017\)](#). We can optionally compute a graph representation $\mathbf{h}_{\mathcal{G}}$ by aggregating the final node representations with another MLP module. Formally,

$$\mathbf{h}_u^{(k)} := \sum_{v \in \mathcal{N}(u)} \text{MLP}^{(k)}\left(\mathbf{h}_u^{(k-1)}, \mathbf{h}_v^{(k-1)}\right), \quad (5)$$

$$\mathbf{h}_{\mathcal{G}} := \text{MLP}^{(K+1)}\left(\sum_{u \in \mathcal{G}} \mathbf{h}_u^{(K)}\right). \quad (6)$$

Depending on the task, the output is either the graph representation $\mathbf{h}_{\mathcal{G}}$ or the final node representations $\mathbf{h}_u^{(K)}$. We refer to the neighbor aggregation step for $\mathbf{h}_u^{(k)}$ as *aggregation* and the pooling step for $\mathbf{h}_{\mathcal{G}}$ as *readout*. Different tasks require different aggregation and readout functions.

4.3.2 Additive Label Noise

[Hu et al. \(2020b\)](#) prove that MLPs are robust to zero-mean additive label noise if the labels are drawn i.i.d. from a Sub-Gaussian distribution. [Wu and Xu \(2020\)](#) also show that linear models are robust to zero-mean additive label noise even in the absence of explicit regularization. In this section, we show that a GNN *well-aligned* to the target function not only achieves low test errors on additive label noise with zero-mean, but also learns *predictive* representations on noisy labels that are drawn from non-zero-mean distributions despite having large test error.

$$\begin{aligned}
h_{\mathcal{G}} &:= \text{Module}^{(2)} \left[\max_{u \in \mathcal{G}} \left(\sum_{v \in \mathcal{N}(u)} \text{Module}^{(1)}(h_u, h_v) \right) \right] \\
f(\mathcal{G}) &:= \underbrace{\max_{u \in \mathcal{G}}}_{f_2(\cdot)} \left[\underbrace{\sum_{v \in \mathcal{N}(u)} 1}_{f_1(\cdot)} \right]
\end{aligned}$$

Figure 4.1: **Max-sum GNN aligns well with the task maximum degree.** Max-sum GNN $h_{\mathcal{G}}$ can be decomposed into two modules: $\text{Module}^{(1)}$ and $\text{Module}^{(2)}$, and the target function $f(\mathcal{G})$ can be similarly divided as $f(\mathcal{G}) = f_2(f_1(\mathcal{G}))$. As the nonlinearities of the target function have been encoded in the GNN’s architecture, $f(\mathcal{G})$ can be easily learned by $h_{\mathcal{G}}$: $f_1(\cdot)$ can be easily learned by $\text{Module}^{(1)}$, and $f_2(\cdot)$ is the same as $\text{Module}^{(2)}$.

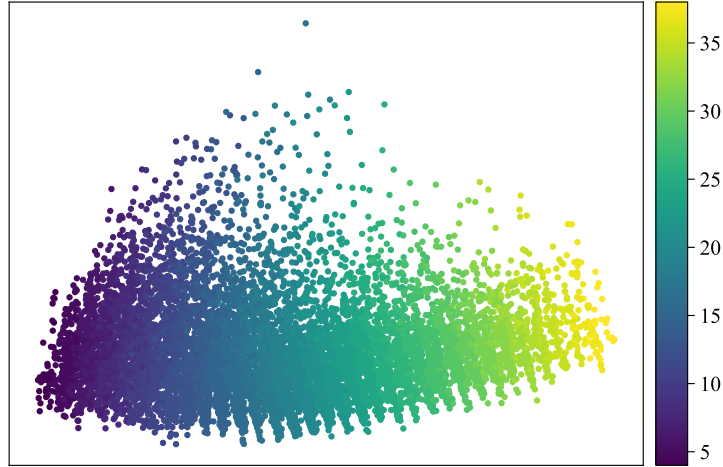


Figure 4.2: **PCA visualization of hidden representations** from a max-sum GNN trained with additive label noise drawn from $\mathcal{N}(10, 15)$ at 100% noise ratio. Each dot denotes a single training example and is colored with its true label. The x-axis and y-axis denote the projected values at the first and second principal components. As the colors change gradually from left to right, the largest principal component of the representations has a clear linear relationship with the true labels.

Task and Architecture. The task is to compute the maximum node degree:

$$f(\mathcal{G}) := \max_{u \in \mathcal{G}} \sum_{v \in \mathcal{N}(u)} 1. \quad (7)$$

We choose this task as we know which GNN architecture aligns well with this target function—a 2-layer GNN with max-aggregation and sum-readout (max-sum GNN):

$$\mathbf{h}_{\mathcal{G}} := \text{MLP}^{(2)} \left(\max_{u \in \mathcal{G}} \sum_{v \in \mathcal{N}(u)} \text{MLP}^{(1)}(\mathbf{h}_u, \mathbf{h}_v) \right), \quad (8)$$

$$\mathbf{h}_u := \sum_{v \in \mathcal{N}(u)} \text{MLP}^{(0)}(\mathbf{x}_u, \mathbf{x}_v). \quad (9)$$

Figure 4.1 demonstrates how exactly the max-sum GNN aligns with $f(\mathcal{G})$. Intuitively, they are well-aligned as the MLP modules of max-sum GNN only need to learn simple constant functions to simulate $f(\mathcal{G})$. Based on Figure 4.1, we take the output of Module⁽²⁾ as the learned representations for max-sum GNNs when evaluating the predictive power.

Label Noise. We corrupt labels by adding independent noise ϵ drawn from three distributions: Gaussian distributions with zero mean $\mathcal{N}(0, 40)$ and non-zero mean $\mathcal{N}(10, 15)$, and a long-tailed Gamma distribution with zero-mean $\Gamma(2, 1/15) - 30$. We also consider more distributions with non-zero mean in Appendix 8.1.2.

Findings. In Figure 4.3, while the max-sum GNN is robust to *zero-mean* additive label noise (dotted yellow and purple lines), its test error is much higher under non-zero-mean noise $\mathcal{N}(10, 15)$ (dotted red line) as the learned signal may be “shifted” by the non-centered label noise. Yet, max-sum GNNs’ learned representations under these three types of label noise all predict the target function well when evaluating their predictive powers with 10% clean labels

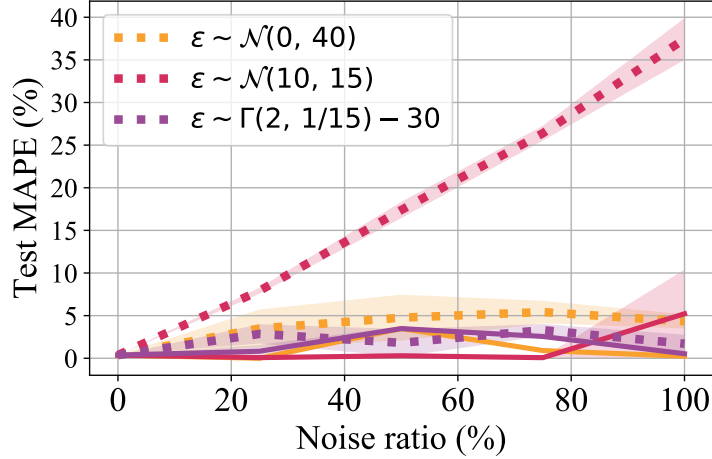


Figure 4.3: **Representations are very predictive for a GNN well-aligned with the target function under additive label noise.** On the maximum degree task, the representations’ predictive power (solid lines) achieves low test MAPE ($< 5\%$) across all three types of noise for the max-sum GNN, despite that the model’s test MAPE (dotted lines) may be quite large (for non-zero-mean noise). We average the statistics over 3 runs using different random seeds.

(solid lines in Figure 4.3).

Moreover, when we plot the representations (using PCA) from a max-sum GNN trained under 100% noise ratio with $\epsilon \sim \mathcal{N}(10, 15)$, the representations indeed correlate well with true labels (Figure 4.2). This explains why the representation learned under noisy labels can recover surprisingly good test performance despite that the original model has large test errors.

The predictive power of randomly-initialized max-sum GNNs is in Table 8.1 (Appendix 8.1.1).

4.3.3 Instance-Dependent Label Noise

Realistic label noise is often instance-dependent. For example, an option is often incorrectly priced in the market, but its incorrect price (i.e., the noisy label) should depend on the properties of the underlying stock. Such instance-dependent label noise is more challenging, as it may contain *spurious signals* that are easy to learn by certain architectures. In this section,

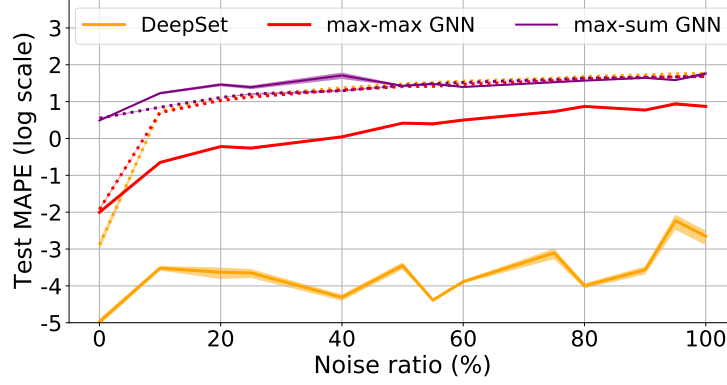


Figure 4.4: **Representations are more predictive for GNNs more aligned with the target function, and less predictive for GNNs more aligned with the noise function.** On the maximum node feature task, while all three GNNs have large test errors under high noise ratios (dotted lines), the predictive power (solid lines) in representations from Deepset (yellow) and max-max GNN (red) greatly reduces the test MAPE. In contrast, the representation’s predictive power for max-sum GNN barely reduces the model’s test MAPE (tiny gaps between dotted and solid purple lines).

we evaluate the representation’ predictive power for three different GNNs trained with instance-dependent label noise.

Task and Label Noise. We experiment with a new task—computing the maximum node feature:

$$f(\mathcal{G}) := \max_{u \in \mathcal{G}} \|\mathbf{x}_u\|_{\infty}. \quad (10)$$

To create an instance-dependent noise, we randomly replace the label with the maximum degree:

$$g(\mathcal{G}) := \max_{u \in \mathcal{G}} \sum_{v \in \mathcal{N}(u)} 1. \quad (11)$$

Architecture. We consider three GNNs: DeepSet [Zaheer et al. \(2017\)](#), max-max GNN, and max-sum GNN. DeepSet can be interpreted as a special GNN that does not use neighborhood

information:

$$h_{\mathcal{G}} = \text{MLP}^{(1)}\left(\max_{u \in \mathcal{G}} \text{MLP}^{(0)}\left(\mathbf{x}_u\right)\right). \quad (12)$$

Max-max GNN is a 2-layer GNN with max-aggregation and max-readout. Max-sum GNN is the same as the one in the previous section.

DeepSet and max-max GNN are well-aligned with the target function $f(\mathcal{G})$, as their MLP modules only need to learn simple linear functions. In contrast, max-sum GNN is more aligned with $g(\mathcal{G})$ than $f(\mathcal{G})$ since neither its MLP modules or sum-aggregation module can efficiently learn the max-operation in $f(\mathcal{G})$, which is shown in both chapter 2 and chapter 3.

Moreover, DeepSet cannot learn $g(\mathcal{G})$ as the model ignores *edge information*. We take the hidden representations before the last MLP modules in all three GNNs and compare their predictive power.

Findings. While all three GNNs have large test errors under high noise ratios (dotted lines in Figure 4.4), the predictive power in representations from GNNs more aligned with the target function — DeepSet (solid yellow line) and max-max GNN (solid red line) — significantly reduces the original models’ test errors by 10 and 1000 times respectively. Yet, for the max-sum GNN, which is more aligned with the noise function, training with noisy labels indeed destroy the internal representations such that they are no longer to predict the target function — its representations’ predictive power (solid purple line) barely decreases test error. We also evaluate the predictive power of these three types of randomly-initialized GNNs, and the results are in Table 8.2 (Appendix 8.1.1).

4.4 Experiments on Vision Datasets

Many noisy label training methods are benchmarked on image classification; thus, we also validate our hypothesis on image domains. We compare the representations’ predictive power between MLPs and CNN-based networks using 10% clean labels (all models are trained until they could perfectly fit the noisy labels, a.k.a., achieving close to 100% training accuracy). We further evaluate the predictive power in representations learned with SOTA methods. Predictive power on networks that aligned well with the target function could further improve the SOTA method’s test performance (Section 4.4.2). The final model also outperforms some sophisticated methods on noisy label training which also use clean labels (Appendix 8.1.4). All our experiment details are in Appendix ??.

4.4.1 MLPs vs. CNN-based networks

To validate our hypothesis, we consider several target functions with different levels of alignment to MLPs and CNN-based networks. All models in this section are trained with standard procedures without any robust training methods or robust losses.

Datasets and Label Noise. We consider two types of target functions: one aligns better with CNN-based models than MLPs, and the other aligns better with MLPs than CNN-based networks.

1). CIFAR-10 and CIFAR-100 [Krizhevsky \(2009\)](#) come with clean labels. Therefore, we generate two types of noisy labels following existing works: (1) **uniform label noise** randomly replaces the true labels with all possible labels, and (2) **flipped label noise** swaps the labels between similar classes (e.g., deer $\leftrightarrow$ horse, dog $\leftrightarrow$ cat) on CIFAR-10 [Li et al. \(2020b\)](#), or flips the

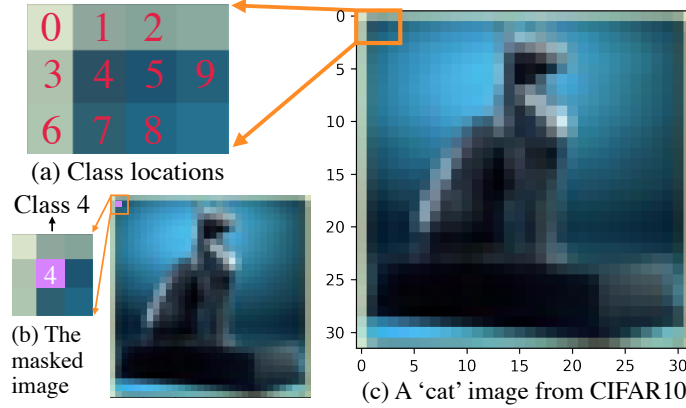


Figure 4.5: **Synthetic Labels on CIFAR-Easy.** For each image, we mask a pixel at the top left corner with pink color. Then the synthetic label for this image is the location of the pink pixel/mask (i.e., the cat image in the above example has label 4).

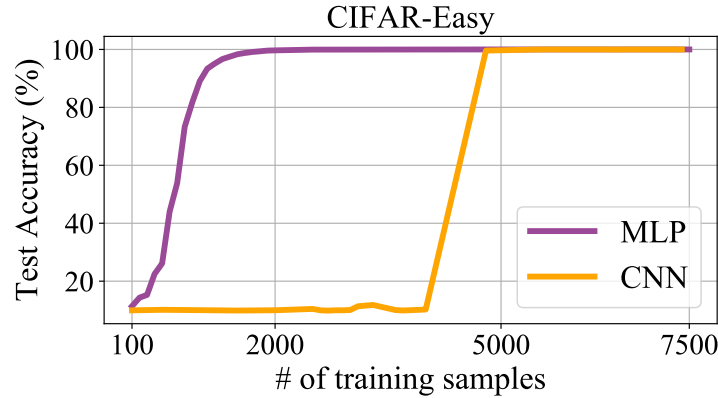


Figure 4.6: **Sample complexity of MLPs and CNNs on CIFAR-Easy.** Both MLPs and CNNs can achieve 100% test accuracy given sufficient training examples, but MLPs need far fewer examples than CNNs and thus are more sample-efficient on CIFAR-Easy.

labels to the next class on CIFAR-100 [Natarajan et al. \(2013\)](#).

2). CIFAR-Easy is a dataset modified on CIFAR-10 with labels generated by procedures in Figure 4.5 — the class/label of each image depends on the location of a special pixel. We consider three types of noisy labels on CIFAR-Easy: (1) **uniform label noise** and (2) **flipped label noise** (described as above); and (3) **instance-dependent label noise** which takes the original image classification label as the noisy label.

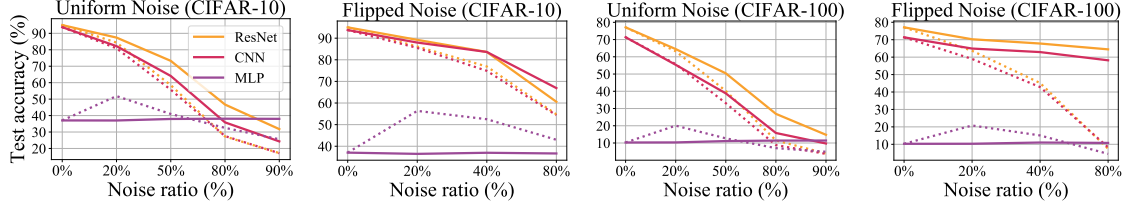


Figure 4.7: **CIFAR-10/100 with uniform and flipped label noise.** Each line indicates the raw test accuracy (dotted) and the predictive power in representations (solid) learned by a model trained across various noise ratios. As CNN-based networks align better with image classification tasks than MLPs, their representations’ predictive power (solid yellow and red lines) are higher than that of MLPs (solid purple lines) on most noise ratios.

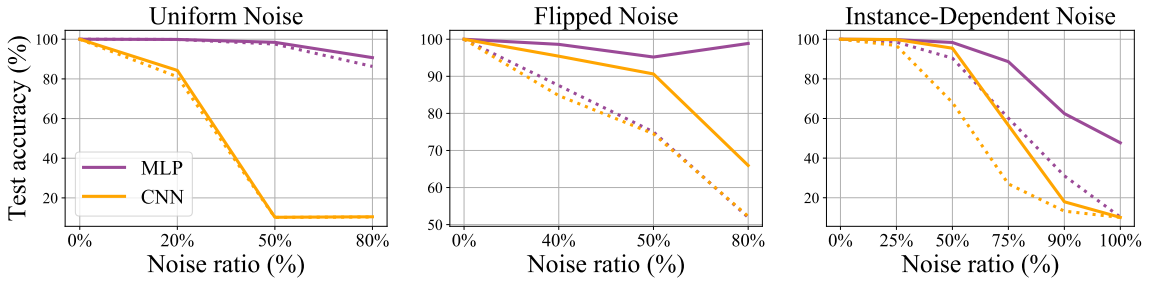


Figure 4.8: **CIFAR-Easy with uniform, flipped, and instance-dependent label noise.** Each line indicates the raw test accuracy (dotted) and the predictive power in representations (solid) learned by a model trained across various noise ratios. As MLPs align better with the target function than CNN-based networks on CIFAR-Easy, their representations’ predictive power on MLPs (solid purple lines) are consistently better than that of CNNs (solid yellow lines) across various noise ratios and noise types.

Architectures. On CIFAR-10/100, we evaluate the predictive power in representations for three architectures: 4-layer MLPs, 9-layer CNNs, and 18-layer PreAct ResNets [He et al. \(2016\)](#). On CIFAR-Easy, we compare between MLPs and CNNs. We take the representations before the penultimate layer when evaluating the predictive power for these networks.

As the designs of CNN-based networks (e.g., CNNs and ResNets) are similar to human perception system because of the receptive fields in convolutional layers and a hierarchical extraction of more and more abstracted features [LeCun et al. \(1995\)](#); [Kheradpisheh et al. \(2016\)](#), CNN-based networks are expected to *align better* with the target functions than MLPs on image classification datasets (e.g., CIFAR-10/100).

Table 4.1: **Comparison of different networks’ test accuracies (%) on CIFAR-10/100.** We color a test accuracy in **red** if it is lower than the test accuracy from vanilla training, and we color it in **green** if it is higher.

Model	Setting	CIFAR10								CIFAR100							
		Uniform noise				Flipped noise				Uniform noise				Flipped noise			
		20%	50%	80%	90%	20%	40%	80%		20%	50%	80%	90%	20%	40%	80%	
4-layer FC (MLP)	Vanilla training	51.8	41.0	32.5	25.6	56.4	52.5	43.0		20.1	12.7	7.0	4.9	20.8	15.1	4.5	
	DivideMix Li et al. (2020b)	62.2	55.2	34.4	28.1	60.2	56.8	44.0		32.8	28.0	13.9	7.2	31.5	22.3	1.3	
	DivideMix's Predictive Power	38.6	38.6	38.8	38.2	38.5	39.0	38.8		11.1	11.8	12.4	11.6	11.1	12.0	11.9	
PreAct ResNet18	Vanilla training	84.4	58.5	27.3	17.2	86.1	76.9	54.7		63.2	40.2	11.5	3.9	63.6	45.2	7.4	
	DivideMix Li et al. (2020b)	95.7	94.4	92.9	75.4	94.0	92.1	56.2		76.9	74.2	59.6	31.0	77.0	55.2	0.2	
	DivideMix's Predictive Power	96.0	94.8	93.5	83.8	94.9	94.0	93.6		76.6	73.9	60.9	39.3	76.8	74.8	76.1	
9-layer CNN	Vanilla training	80.9	55.7	27.5	17.1	85.5	74.9	54.4		55.8	33.1	8.8	3.7	59.0	42.8	8.3	
	DivideMix Li et al. (2020b)	94.5	93.4	91.2	78.2	92.9	89.8	55.3		71.4	69.0	51.8	22.9	71.3	53.0	0.3	
	DivideMix's Predictive Power	94.5	93.6	91.4	81.8	93.6	92.1	90.1		69.9	67.1	50.4	26.3	70.2	69.0	68.8	

Table 4.2: **Comparison of different networks’ test accuracies (%) on CIFAR-Easy.** We color a test accuracy in **red** if it is lower than the test accuracy from vanilla training, and we color it in **green** if it is higher.

Model	Setting	Uniform noise				Flipped noise			Spurious noise				
		0%	50%	80%	90%	40%	50%	80%	25%	50%	75%	90%	100%
4-layer FC (MLP)	Vanilla training	100.00	99.88	97.57	86.29	87.52	75.01	51.94	98.55	90.46	60.16	31.08	10.25
	DivideMix Li et al. (2020b)	98.35	10.00*	99.99	16.22	100.00	88.36	50.04	100.00	100.00	45.59	14.16	10.10
	DivideMix’s Predictive Power	100.00	100.00	100.00	99.94	100.00	100.00	100.00	100.00	100.00	100.00	98.66	99.65
9-layer CNN	Vanilla training	100.00	81.08	10.15	10.36	84.80	74.50	52.24	96.84	68.07	26.97	13.24	10.07
	DivideMix Li et al. (2020b)	100.00	100.00	100.00	10.00	92.75	86.33	50.60	99.96	99.82	10.45	10.09	10.14
	DivideMix’s Predictive Power	100.00	100.00	100.00	10.09	99.33	98.42	96.76	100.00	99.99	14.99	10.70	10.15

* The phenomenon that DivideMix fails under 50% uniform noise but succeeds under 80% uniform noise is due to the unstable behaviors of DivideMix’s division process, indicating that the predictive power in representations could be a more stable measure of a model’s robustness, as the model can fail miserably (a.k.a., performance close to random guessing), but its learned representation can still predict the target function well.

On the other hand, on CIFAR-Easy, while both CNNs and MLPs can generalize perfectly given sufficient training examples, MLPs have a much smaller sample complexity than CNNs (Figure 4.6). Thus, both MLP and CNN are *well-aligned* with the target function on CIFAR-Easy, but MLP is *better-aligned* than CNN according to Theorem 4.2.6. Moreover, since the instance-dependent label on CIFAR-Easy is the original image classification label, CNN is also *aligned* with this instance-dependent noise function on CIFAR-Easy.

Experimental Results. First, we empirically verify our hypothesis that *networks better aligned with the target function have more predictive representations*. As expected, across most noise ratios on CIFAR-10/100, the representations in CNN-based networks (i.e., CNN and

ResNet) are more predictive than those in MLPs (Figure 4.7) under both types of label noise. Moreover, the predictive power in representations learned by less aligned networks (i.e., MLPs) sometimes is even worse than the vanilla-trained models’ test performance, suggesting that the noisy representations on less aligned networks may be more corrupted and less linearly separable. On the other hand, across all three types of label noise on CIFAR-Easy, MLPs, which align better with the target function, have more predictive representations than CNNs (Figure 4.8).

We also observe that *models with similar test performance could have various levels of predictive powers in their learned representations*. For example, in Figure 4.8, while the test accuracies of MLPs and CNNs are very similar on CIFAR-Easy under flipped label noise (i.e., dotted purple and yellow lines overlap), the predictive power in representations from MLPs is much stronger than the one from CNNs (i.e., solid purple lines are much higher than yellow lines). This also suggests that when trained with noisy labels, if we do not know which architecture is more aligned with the underlying target function, we can evaluate the predictive power in their representations to test alignment.

We further discover that *for networks well-aligned with the target function, its learned representations are more predictive when the noise function shares more mutual information with the target function*. We compute the empirical mutual information between the noisy training labels and the original clean labels across different noise ratios on various types of label noise. The predictive power in representations improves as the mutual information increases (Figure 8.3 in Appendix 8.1). This explains why the predictive power for a network is often higher under flipped noise than uniform noise: at the same noise ratio, flipped noise has higher mutual information than uniform noise. Moreover, comparing the three datasets in Figure 8.3, we observe the growth rate of a network’s predictive power w.r.t. the mutual information depends on both the intrinsic

difficulties of the learning task and the alignment between the network and the target function.

4.4.2 Predictive Power in Models Trained with SOTA Methods

As previous experiments are on standard training procedures, we also validate our hypothesis on models learned with SOTA methods on noisy label training. We evaluate the representations’ predictive power for models trained with the SOTA method, DivideMix [Li et al. \(2020b\)](#), which leverages techniques from semi-supervised learning to treat examples with unreliable labels as unlabeled data.

We compare (1) the test performance for models trained with standard procedures on noisy labels (denoted as **Vanilla training**), (2) the SOTA method’s test performance (denoted as **DivideMix**), and (3) the predictive power in representations from models trained with DivideMix in (2) (denoted as **DivideMix’s Predictive Power**).

We discover that *the effectiveness of DivideMix also depends on the alignment between the network and the target/noise functions*. DivideMix only slightly improves the test accuracy of MLPs on CIFAR-10/100 (Table [4.1](#)), and DivideMix’s predictive power does not improve the test performance of MLPs, either. In Table [4.2](#), DivideMix also barely helps CNNs as they are well-aligned with the instance-dependent noise, where the noisy label is the original image classification label.

Moreover, we observe that *even for networks well-aligned with the target function, DivideMix may only slightly improve or do not improve its test performance at all* (e.g., red entries of DivideMix on MLPs in Table [4.2](#)). Yet, the representations learned with DivideMix can still be very predictive: the predictive power can achieve over 50% improvements over DivideMix for

CNN-based models on CIFAR-10/100 (e.g., 80% flipped noise), and the improvements can be over 80% for MLPs on CIFAR-Easy (e.g., 90% uniform noise).

Tables 4.1 and 4.2 show that the representations’ predictive power on networks well aligned with the target function could further improve SOTA test performance. Appendix 8.1.4 further demonstrates that on large-scale datasets with real-world noisy labels, the predictive power in well-aligned networks could outperform sophisticated methods that also use clean labels (Table 8.7 and Table 8.8).

4.5 Chapter Summary

This chapter is an initial step towards formally understanding how a network’s architecture impacts its robustness to noisy labels. We formalize our intuitions and hypothesize that a network better aligned with the target function would learn more predictive representations under noisy label training. We prove our hypothesis in a simplified noisy setting and conduct systematic experiments across various noisy settings to further validate our hypothesis.

Our empirical results along with Theorem 4.2.6 suggest that knowing more structures of the target function can help design more robust architectures. In practice, although an exact mathematical formula for a decomposition of a given target function is often hard to obtain, a high-level decomposition of the target function often exists for real-world tasks and will be helpful in designing robust architectures — a direction undervalued by existing works on learning with noisy labels.

My main contribution to this chapter includes the main idea, initial exploration experiments, the method that helps with noisy label learning, the theoretical results, all experimental

results, and additional experiments in (Appendix) Chapter 8. Since the publication of this chapter, many follow-up studies have been inspired by our key observation: the learned representations of aligned models under noisy label training can still be useful to predict the clean labels. These subsequent works have proposed improved methods to mitigate the effects of noisy labels [Miao et al. \(2023\)](#); [Chen et al. \(2022\)](#); [Forouzesh et al. \(2022\)](#), and used our insights to better explain the success of their approaches [Bai et al. \(2021\)](#); [Zhu et al. \(2022\)](#).

Chapter 5: Decision Making: Hindsight Learning for MDPs with Exogenous Inputs

Joint work with Sean R. Sinclair, Felipe Frujeri, Ching-An Cheng, Luke Marshall, Hugo Barbalho, Jennifer Neville, Ishai Menache, and Adith Swaminathan

5.1 Introduction

Many aspects of our physical and digital infrastructure — like data centers, power grids, and supply chains — can become more adaptive and efficient through data-driven decision-making. For instance, in a world-wide cloud service, even 1% lower resource fragmentation can reduce energy use and save approximately \$100M per year ([Hadary et al., 2020](#)). This type of improvement could be achieved by examining historical patterns of compute demands and using machine learning (ML) to allocate future demands more efficiently.

In this work, we make the key observation that in resource management applications the system is often partially known and the only uncertainty is due to *exogenous* variables like resource requests — that are (to a first-order approximation) *independent* of an agent’s decisions ([Powell, 2022](#)). For example, a cloud operator deciding to place a virtual machine (VM) on a specific server rack does not directly affect future VM requests, but future demands can strongly affect the eventual quality of their allocation decisions ([Hadary et al., 2020](#)). We define these

problems as *Exo-MDPs* (Markov Decision Processes with Exogenous Inputs), which are a subclass of *Input-Driven MDPs* (Mao et al., 2019b). In Input-Driven MDPs the minimal state describing the system dynamics decompose into (i) *exogenous* inputs, which evolve independently of the agent’s actions, and (ii) *endogenous* factors that are impacted by the agent’s actions and the exogenous inputs. Exo-MDPs make the additional assumption that the only unknowns are the distribution of future exogenous inputs (see 5.3).

ML has been applied in several resource management applications, which we will show are Exo-MDPs, and found to outperform domain-specific heuristics (Lykouris and Vassilvitskii, 2021; Kumar et al., 2018; Gollapudi and Panigrahi, 2019). The ML approaches often follow the Predict-Then-Optimize (PTO) paradigm (Elmachtoub and Grigas, 2022), using ML to forecast the future exogenous inputs (e.g., demands). However, when the future is highly stochastic, forecasting is challenging. For example, VM requests from real-world data centers (Hadary et al., 2020) show long-term regularity of diurnal and weekly patterns but extreme short-term fluctuations (see Figure 9.2).

Reinforcement Learning (RL) is an alternative to PTO. RL directly optimizes decision quality (Chen et al., 2021; Fang et al., 2019; Mao et al., 2016) by replaying historical samples of the exogenous inputs through the known dynamics of endogenous factors to learn good policies (Madeka et al., 2022). RL methods applied to Exo-MDPs must, however, learn by trial-and-error that their actions can never affect the exogenous inputs. RL is thus sensitive to variance in the outcomes introduced by the exogenous inputs and requires more data to learn an optimal policy when the variance is high (Foster et al., 2021) (see 5.4).

Recent works have proposed to use *hindsight* control variates to reduce the variance of RL for input-driven MDPs (which include Exo-MDPs) (Mao et al., 2019b; Mesnard et al., 2021).

They derive unbiased policy gradients by subtracting from the observed outcomes a function that depends additionally on hindsight information. However, we find that for many resource management scenarios, the variance reduction from hindsight control variates is not enough for data-efficient learning in practical regimes (see Table 5.2).

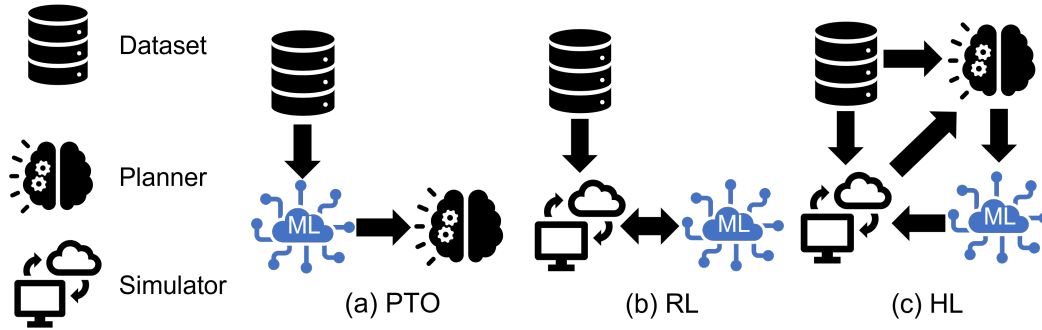


Figure 5.1: Conceptual view of different ML approaches to solving exo-MDPs ((a) and (b) are detailed in 9.5). Arrows in the figure indicate the flow of information from one component to the other. In (a) **Predict-Then-Optimize** (PTO) uses the dataset to train an ML forecasting model, uses the model to predict future exogenous inputs, and uses the forecasted inputs online for planning optimal actions. In (b) **RL** replays the dataset through the simulator to evaluate an ML policy’s performance, and tunes policy parameters using the collected rewards as the training signal. In (c) our **Hindsight Learning** (HL) approach uses the dataset directly with the planner (top arrow) to identify hindsight-optimal values, and trains the ML policy using state trajectories from the simulator annotated with the hindsight optimal values.

We argue that hindsight information can be more effectively used to improve learning, as it can largely reduce the variance of exogenous inputs at the cost of a small amount of asymptotic bias in many cases. Based on this insight, we develop a family of algorithms called Hindsight Learning (HL), which uses hindsight planners during learning to lessen the variance from exogenous inputs.

A hindsight planner (Chong et al., 2000; Gopalan et al., 2010; Conforti et al., 2014) is an optimization algorithm that provides computationally tractable approximations to hindsight-optimal decisions, which is the optimal action specialized to a fixed sequence of future demands and thus is not affected by the exogenous variability. Therefore, by using hindsight planners

during learning, HL can more efficiently identify decisions critical to future performance under high-variance exogenous inputs. Figure 5.1 contrasts the HL algorithm schematic with PTO and RL.

This chapter theoretically characterizes when HL succeeds using a novel quantity called *hindsight bias* (which arises due to the mismatch between truly optimal and hindsight-optimal decisions). Remarkably, we find that hindsight bias is small for many resource management problems, so HL can learn with extremely limited data for them. To prove this, we use recent advances in prophet inequalities (Dutting et al., 2020; Vera and Banerjee, 2021) with novel adaptations for the Exo-MDP setting. We empirically test HL in several domains: Multi-Secretary problems, Airline Revenue Management (ARM) benchmarks, and Virtual Machine (VM) allocation. We find that HL is significantly better than both domain heuristics and RL (with and without hindsight control variates), illustrating that HL indeed strikes a better bias-variance trade-off. Notably, our VM allocation experiments use real historical traces from a large public cloud provider, where HL is the only approach that consistently beats the currently used heuristics (0.1% – 5% better). Recall that even a 1% better allocator can yield massive savings in practice (Hadary et al., 2020).

5.2 Related Work

Here we include a brief discussion of salient related work. Please see 9.2 for more details.

Recent studies have exploited causal structure in MDPs for better decision-making (Lattimore et al., 2016; Lu et al., 2022b). Their causal graphs however do not capture Exo-MDPs (e.g., Figure 5.2). When the exogenous process is only partially observed, HL may additionally need

causal RL techniques (Zhang et al., 2020a); this is left for future work.

Input-Driven MDPs have been specialized before with additional assumptions that either the rewards or the transitions factorize so that the exogenous process can be filtered out (Dietterich et al., 2018; Efroni et al., 2022). However, they are not suited for Exo-MDPs because filtering out the exogenous process yields demand-agnostic policies, which are highly sub-optimal for resource management problems.

Hindsight optimization has been previously attempted Chong et al. (2000); Feng et al. (2021), and it is well known that these values are over-optimistic. Mercier and Van Hentenryck (2007) show that despite the over-optimism, the regret for hindsight optimization policies in several network scheduling and caching problems is a constant. The Progressive Hedging (PH) algorithm in Rockafellar and Wets (1991) attempts to eliminate the over-optimism by iteratively refining the hindsight optimization solution by adding non-anticipative constraints. PH has weak guarantees for convergence in non-convex problems (like our settings) and is intractable for the large problem sizes that we consider. Information relaxation (Brown and Smith, 2022) extends PH to non-convex rewards and arbitrary action spaces and allows for imperfect penalization of the non-anticipative constraint violations. These schemes require hand-crafted penalties as well as tractable hindsight planning with those penalized objectives. Instead, Mercier and Van Hentenryck (2007) foregoes solving for the optimal policy of the MDP and instead produces a potentially sub-optimal non-anticipatory policy. We provide a tighter regret analysis for their surrogate policy (9.6.6) and additionally describe an imitation learning algorithm to avoid unnecessary computation in large-scale problems.

5.3 Problem Setting and Definitions

5.3.1 MDPs with Exogenous Inputs (Exo-MDPs)

We consider a subclass of finite horizon ¹Markov Decision Processes (MDPs). An MDP is defined by $(\mathcal{S}, \mathcal{A}, T, P, R, s_1)$ with horizon T , state space $\mathcal{S}$, action space $\mathcal{A}$, reward distribution R , state transition distribution P , and starting state s_1 (Puterman, 2014). An Exo-MDP further specializes an MDP by separating the process into *endogenous* and *exogenous* parts: a state $s \in \mathcal{S}$ factorizes into endogenous/system state $x \in \mathcal{X}$ and exogenous inputs $\xi := (\xi_1, \dots, \xi_T) \in \Xi^T$ (namely, $\mathcal{S} := \mathcal{X} \times \Xi^T$). The state transition distribution P also factors into an endogenous part f and exogenous part $\mathcal{P}_\Xi$ as follows. At time t , the agent selects action $a_t \in \mathcal{A}$ based on the current state $s_t := (x_t, \xi_{<t})$ where $\xi_{<t} := (\xi_1, \dots, \xi_{t-1})$ is the observed exogenous inputs thus far, *and then* ξ_t is sampled from an unknown distribution $\mathcal{P}_\Xi(\cdot \mid \xi_{<t})$, independently of the agent's action a_t . With ξ_t , the endogenous state evolves according to $x_{t+1} = f(s_t, a_t, \xi_t)$ and the reward earned is $r(s_t, a_t, \xi_t) \in [0, 1]$. Note ξ_t is only observed when the agent makes the decision at time $t + 1$, *not* at time t . We restrict our attention to policies $\pi_t : \mathcal{X} \times \Xi^{t-1} \rightarrow \Delta(\mathcal{A})$ and let Π denote the policy class of the agent. The endogenous dynamics f and reward function r are assumed to be known to the agent²; the only unknown in the Exo-MDP is $\mathcal{P}_\Xi$. For notational convenience, we also assume that f and r are deterministic; all of our insights carry over to stochastic rewards and transitions. These assumptions of Exo-MDPs are well-motivated for resource management problems, and we list the state decomposition along with f and r for several examples in 9.3.

¹All the algorithms and analysis also apply analogously to infinite horizon problems with a discount factor.

²Thus, Exo-MDPs are a subclass of Input-Driven MDPs (Mao et al., 2019b) which more generally have unknown f, r . 9.3 shows that even though f and r are known in Exo-MDPs, they can be as difficult as arbitrary MDPs with state space $\mathcal{X}$.

5.3.2 Value Decomposition in Exo-MDPs

Since the only unknown in an Exo-MDP is $\mathcal{P}_\Xi$, a policy's performance can be written as expectations over $\mathcal{P}_\Xi$. This motivates the use of historical samples $\xi \sim \mathcal{P}_\Xi$ to evaluate any policy and find optimal policies for the Exo-MDP.

For $\pi \in \Pi$, the *values* and *action-values* are defined as:

$$V_t^\pi(s) := \mathbb{E}_{\xi_{\geq t}, \pi} \left[\sum_{\tau \geq t} r(s_\tau, a_\tau, \xi_\tau) \mid s_t = s \right],$$

$$Q_t^\pi(s, a) := \mathbb{E}_{\xi_{\geq t}, \pi} \left[\sum_{\tau \geq t} r(s_\tau, a_\tau, \xi_\tau) \mid s_t = s, a_t = a \right],$$

where the expectation is taken over the randomness in π and the exogenous inputs ξ . We denote π^* as the optimal policy, i.e. the policy that maximizes $V_t^\pi(s)$ in each state s , and denote Q^*, V^* for Q^{π^*}, V^{π^*} respectively. Our goal is to find a policy with near-optimal returns, $\arg \max_{\pi \in \Pi} \{V_1^\pi := V_1^\pi(s_1)\}$. Or equivalently, minimize $\text{REGRET}(\pi)$ where $\text{REGRET}(\pi) := V_1^* - V_1^\pi$. For convenience we assume ³that $\pi^* \in \Pi$. We introduce value functions for fixed $\xi = \{\xi_1, \dots, \xi_T\}$ as

$$Q_t^\pi(s, a, \xi_{\geq t}) := \mathbb{E}_\pi \left[\sum_{\tau \geq t} r(s_\tau, a_\tau, \xi_\tau) \mid s_t = s, a_t = a \right], \quad (1)$$

$$V_t^\pi(s, \xi_{\geq t}) := \sum_a \pi(a|s) Q_t^\pi(s, a, \xi_{\geq t}). \quad (2)$$

Note that the expectation is not over $\mathcal{P}_\Xi$ because ξ is fixed. The ξ -specific values are related to policy values as follows.

³If not, our theoretical results need an additional term for the difference in returns between the best policy in Π and π^* .

Lemma 5.3.1. *For every $t \in [T]$, $(s, a) \in \mathcal{S} \times \mathcal{A}$, policy $\pi \in \Pi$ we have that*

$$Q_t^\pi(s, a) = \mathbb{E}_{\boldsymbol{\xi}_{\geq t}}[Q_t^\pi(s, a, \boldsymbol{\xi}_{\geq t})], \quad (3)$$

$$V_t^\pi(s) = \mathbb{E}_{\boldsymbol{\xi}_{\geq t}}[V_t^\pi(s, \boldsymbol{\xi}_{\geq t})]. \quad (4)$$

In particular $V_1^\pi = \mathbb{E}_{\boldsymbol{\xi}}[V_1^\pi(s_1, \boldsymbol{\xi})]$.

We relegate complete proofs to 9.6. Since the transition dynamics f and reward function r are known and the unknown $\mathcal{P}_\Xi$ does not depend on the agent’s actions, an immediate consequence is that *action exploration is not needed*. Given any policy π and exogenous trace $\boldsymbol{\xi}$ we can simulate with f and r to calculate its return in Equation 1.

Suppose we collected traces $\mathcal{D} = \{\boldsymbol{\xi}^1, \dots, \boldsymbol{\xi}^N\}$ where each trace $\boldsymbol{\xi}^n = \{\xi_1^n \dots, \xi_T^n\}$ is sampled independently from $\mathcal{P}_\Xi$. Finding a near-optimal policy using this historical dataset is known as the *offline RL* problem (Fujimoto et al., 2019; Liu et al., 2020b; Rashidinejad et al., 2021; Cheng et al., 2022), but this is much simpler in Exo-MDPs. We do not face *support mismatch* wherein trajectories from a data-collection policy may not cover the scenarios that the learner policy would encounter. Here $\mathcal{D}$ (collected by a behavior policy) can be safely replayed to evaluate any learner policy. This fact also implies that model selection and hyper-parameter tuning can be safely done using a held-out $\mathcal{D}$ akin to supervised learning. Our goal finally is to learn policies that generalize from $\mathcal{D}$ to the unknown $\mathcal{P}_\Xi$, which can be challenging because the exogenous inputs $\boldsymbol{\xi}$ introduce variance in a policy’s return estimation.

5.3.3 Hindsight Planner

Exo-MDPs not only allow easy policy evaluation using a dataset of traces, but they also allow computing valuable hindsight information like the hindsight-optimal decisions for a trace ξ . This hindsight information can be stable even when $\mathcal{P}_\Xi$ is highly stochastic. We now make a computational assumption for calculating hindsight-optimal decisions that will enable tractable algorithms for Exo-MDPs.

Assumption 5.3.2. *Given any trace $\xi_{\geq t} = (\xi_t, \dots, \xi_T)$ and state $s = (x_t, \xi_{<t})$ we can tractably solve:*

$$\begin{aligned} & \max_{a_t, \dots, a_T} \sum_{\tau=t}^T r(s_\tau, a_\tau, \xi_\tau) \\ & s.t. \ x_{\tau+1} = f(s_\tau, a_\tau, \xi_\tau), \text{ for } \tau = t, \dots, T \\ & \quad s_\tau = (x_\tau, \xi_{<\tau}), \text{ for } \tau = t, \dots, T. \end{aligned} \tag{5}$$

We denote the optimal objective value to this problem as $\text{HINDSIGHT}(t, \xi_{\geq t}, s)$.

The optimization community has developed computationally efficient implementations for the $\text{HINDSIGHT}(t, \xi_{\geq t}, s)$ oracle; with tight bounds on the optimal value even when 5 is intractable. For example, online knapsack for a fixed input sequence can be solved in pseudo-polynomial time (Gopalan et al., 2010). In many instances, Equation 5 can be represented as an integer program and solved via heuristics (Conforti et al., 2014). Recently RLCO (RL for Combinatorial Optimization) has proved to be an effective heuristic for hindsight planning (Anthony et al., 2017; Fang et al., 2021). Note that 5.3.2 or RLCO cannot be used directly as a non-anticipatory policy for the Exo-MDP, since the whole sequence $\xi \sim \mathcal{P}_\Xi$ is not observed

upfront when making decisions. We discuss several examples of hindsight planners in 9.4 and assess the impact of approximate planning empirically in 5.7.3.2.

5.4 Using Hindsight In Exo-MDPs: An Example

In an Exo-MDP, the only uncertainty is due to unknown $\mathcal{P}_{\Xi}$. When $\mathcal{P}_{\Xi}$ introduces substantial variability in outcomes, the natural question is whether generic MDP algorithms can learn effectively? When $T = 1$, Exo-MDPs are isomorphic to a multi-armed bandit and so general bandit algorithms are optimal for the Exo-MDP. However, we will see in the next example with $T > 1$ that the answer is in general no.

Consider the sailing example in Figure 5.2 which is an Exo-MDP where the decision is to pick between one of two routes *prior to observing wind* with hopes of minimizing the trip duration. By direct calculation, $Q^*(\text{route2}) - Q^*(\text{route1}) = -48$. Hence, the optimal non-anticipative policy will always pick route2.

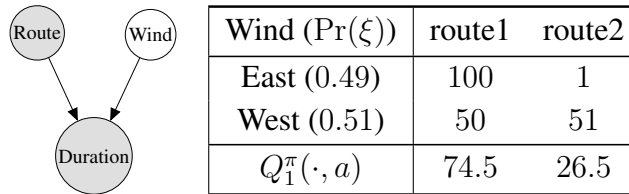


Figure 5.2: An Exo-MDP for sailing in uncertain winds: $\xi = \{\text{Wind}\}$, $\mathcal{A} = \{\text{Route}\}$, $r = \{\text{Duration}\}$ and $\mathcal{X} = \emptyset$. First, the agent picks a *Route*. Then *Wind* conditions are observed during the trip and the agent receives a cost with respect to *Duration*. Values in the table denote average trip duration $r(a)$ (accounting for random fluctuations in wind).

RL: If a classic RL approach was applied to this problem, it would estimate the average duration for each route using observed samples, include exploration bonuses to account for uncertainty, and compare the averages to pick future routes. This requires many ξ samples because wind

introduces large variance in the Q -estimates, and requires sufficient data to be collected across both the routes.

Hindsight Learning: In hindsight, we can instead use *all* observed samples (leveraging known f and r), with *no* exploration bonuses, and use paired comparisons to identify the optimal route. Variability due to wind means the routes’ durations are typically positively correlated and thus a paired sample between routes will be more statistically efficient.

5.5 Hindsight Learning

We introduce Hindsight Learning (HL) to incorporate hindsight information in a principled manner, so as to reduce the exogenous variability and thereby speed-up learning. HL first uses a hindsight planner (5.3.2) to derive a surrogate policy π^+ :

$$\pi_t^+(s) := \arg \max_{a \in \mathcal{A}} Q_t^+(s, a), \quad (6)$$

$$Q_t^+(s, a) := \mathbb{E}_{\xi_{\geq t}}[r(s, a, \xi_t) + \text{HINDSIGHT}(t+1, \xi_{>t}, f(s, a, \xi_t))], \quad (7)$$

$$V_t^+(s) := \mathbb{E}_{\xi_{\geq t}}[\text{HINDSIGHT}(t, \xi_{\geq t}, s)]. \quad (8)$$

We define $Q_t^+(s, a, \xi_{\geq t})$ and $V_t^+(s, \xi_{\geq t})$ as the terms inside of the respective expectations. Note that π^+ is a non-anticipatory policy, which considers expectation over future exogenous $\xi_{\geq t}$ rather than being defined for a fixed trace. π^+ is called “Bayes Selector” in the literature (Vera and Banerjee, 2021; Mercier and Van Hentenryck, 2007) and has been used for applications like bin packing and refugee resettlement (Bansak and Paulson, 2022; Ahani et al., 2021; Banerjee and

Freund, 2020). Intuitively $\pi^\dagger$ uses the returns accumulated by hindsight-optimal actions to score and rank good decisions, instead of mimicking the hindsight-optimal actions directly. However, it is always the case that $V_t^\dagger(s) \geq V^*(s)$ and $Q_t^\dagger(s, a) \geq Q^*(s, a)$ (Chong et al., 2000), and so $\pi^\dagger$ can be a sub-optimal surrogate for π^* . In 5.6 we will bound its gap to π^* with a novel quantity called *hindsight bias* and discuss the implications for learning.

5.5.1 Imitating the “Bayes Selector”

Executing $\pi^\dagger$ requires frequent calls to the hindsight planner online to evaluate $Q_t^\dagger(S_t, a)$ on every observed state. Therefore running this *tabular* policy (i.e., considering every state separately) can be prohibitively costly when policy execution must also satisfy latency constraints (e.g., in VM allocation). Additionally, in resource allocation domains it is infeasible to enumerate all possible states. We describe a family of algorithms in 1 that offloads the hindsight planner invocations to an offline training phase and distills $\pi^\dagger$ into a computationally feasible policy of neural networks that can extrapolate to unseen states.

Algorithm 1 Hindsight Learning

- 1: **Input:** An empty buffer $\mathcal{B}$, simulator for f and r , initial policy π , dataset $\mathcal{D}$, number of epochs K .
 - 2: **for** $k = 1, 2, \dots, K$ **do**
 - 3: Sample a trace ξ from $\mathcal{D}$
 - 4: Sample trajectory $\{s_1 \dots s_T\}$ from π using the f, r, ξ
 - 5: Label sampled states from the trajectory with $\{Q_t^\dagger(s_t, a, \xi_{\geq t}) : a \in \mathcal{A}, t \in [T]\}$
 - 6: Aggregate the labeled data into the buffer $\mathcal{B}$
 - 7: Optimize π on $\mathcal{B}$ either with Hindsight MAC (Equation 9) or Hindsight Q-Distillation (Equation 10)
 - 8: **end for**
-

We use online imitation learning (IL) with $\pi^\dagger$ as the expert policy, specifically, the AggreVaTE algorithm (Ross and Bagnell, 2014) with expert $Q^\dagger(s, a)$ values from the hindsight

planner. By interleaving the trajectory sampling and policy updates we avoid querying $Q^+(s, a)$ (and hence solving hindsight planning problems) in uninformative states. We note that all of this interactive querying of the expert occurs during *offline* training, and hence the planning is not needed online during test time. Since we allow ξ to be arbitrarily correlated across t , in 1 we sample an entire trace ξ^i from $\mathcal{D}$. However, if ξ is iid across time-steps t we can enhance step 3 by resampling a single ξ_t at each t .

Many popular RL algorithms (Konda and Tsitsiklis, 2000; Van Hasselt et al., 2016; Schulman et al., 2017) compute Q -values or advantages via Monte Carlo estimation. HL can be easily incorporated in them by replacing the Monte Carlo estimates (e.g., $Q^\pi(s, a)$) with Step 5 of 1 (i.e., $Q^+(s, a)$). We outline two such modifications below, one using a policy network and the other using a critic network. Since we can simulate $Q_t^+(s, a, \xi_{\geq t})$ for any action, we use the *common random numbers* insight (Ng and Jordan, 2000) for variance reduction and sum across all actions in both instantiations. This additional sum over actions trick is not critical, and is used in our experiments only because the action spaces are relatively small.

Hindsight MAC: We modify Mean Actor Critic (Allen et al., 2017) by incorporating Q^+ into differentiable imitation learning (Sun et al., 2017). For the policy represented with a neural network π_θ , consider the loss function:

$$\ell(\pi_\theta) = \mathbb{E}_\xi \left[\sum_{t=1}^T \mathbb{E}_{S_t \sim \text{Pr}_t^\pi} \sum_{a \in \mathcal{A}} \pi_\theta(a | S_t) Q_t^+(S_t, a, \xi_{\geq t}) \right]. \quad (9)$$

Hindsight Q-Distillation: We can represent Q^+ values directly using a neural network critic Q^θ . The policy is defined implicitly w.r.t. Q^θ as $\pi_\theta = \arg \max_{a \in \mathcal{A}} Q^\theta(s, a)$. The loss function $\ell(\pi_\theta)$ to fit Q^θ is:

$$\ell(\pi_\theta) = \mathbb{E}_\xi \left[\sum_{t=1}^T \mathbb{E}_{S_t \sim \text{Pr}_t^\pi} \left[\sum_{a \in \mathcal{A}} (Q_t^\theta(S_t, a) - Q_t^+(S_t, a, \xi_{\geq t}))^2 \right] \right]. \quad (10)$$

We optimize a sample approximation of Equation 9 or 10 using the dataset $\mathcal{D}$. The current policy defines the state sampling distribution, while $Q^\dagger$ gives the long-term reward signal for each action.

5.6 Theoretical Guarantees

Compared with RL, HL uses the known dynamics f and rewards r of an Exo-MDP, the hindsight planner of 5.3.2, and the dataset $\mathcal{D}$ to trade-off a small asymptotic bias (which we define in 11) for a large reduction in the variance from exogenous inputs. To prove this, we first characterize the regret of $\pi^\dagger$ in terms of a novel quantity, called the *hindsight bias*, and show that it is negligible in many resource management problems. Next we show that HL is sample efficient, and can imitate the $\pi^\dagger$ policy with faster optimization than RL. Finally, even if hindsight bias is large in an application, there are techniques to reduce it, including combinations of HL and RL in the future.

Definition 5.6.1. *The hindsight bias of $\pi^\star$ versus $\pi^\dagger$ at time t in state s is defined as*

$$\begin{aligned} \Delta_t^\dagger(s) := & Q_t^\dagger(s, \pi_t^\dagger(s)) - Q_t^\star(s, \pi_t^\dagger(s)) + \\ & Q_t^\star(s, \pi_t^\star(s)) - Q_t^\dagger(s, \pi_t^\star(s)). \end{aligned} \tag{11}$$

Consider the over-estimation error of the hindsight planner $\Omega_t(s, a) := Q_t^\dagger(s, a) - Q_t^\star(s, a)$ (referred to as *local loss* in Mercier and Van Hentenryck (2007)). 11 subtracts the over-estimation of $\pi^\star(s)$ from the over-estimation of $\pi^\dagger(s)$ and so the hindsight bias can be small even if the over-estimation is large; as an extreme example consider the case when the argmax of $Q^\dagger$ and $Q^\star$

coincide (see 9.6.5). We show that $\Delta_t^\dagger(s)$ bounds the regret of $\pi^\dagger$.

Theorem 5.6.2. $\text{REGRET}(\pi^\dagger) \leq \sum_{t=1}^T \mathbb{E}_{S_t \sim \text{Pr}_t^{\pi^\dagger}} [\Delta_t^\dagger(S_t)]$, where $\text{Pr}_t^{\pi^\dagger}$ denotes the state distribution of $\pi^\dagger$ at step t induced by the exogenous process. In particular, if $\Delta_t^\dagger(s) \leq \Delta$ for some constant Δ then we have: $\text{REGRET}(\pi^\dagger) \leq \Delta \sum_{t=1}^T \mathbb{E}_{S_t \sim \text{Pr}_t^{\pi^\dagger}} [\Pr(\pi_t^\dagger(S_t) \neq \pi^*(S_t))]$.

Regret bounds of this form appear in the prophet inequality literature (Dutting et al., 2020; Vera et al., 2021) however against a much stronger benchmark of the hindsight planner, where the regret is defined as $V^\dagger(s_1) - V_1^\pi(s_1)$. Mercier and Van Hentenryck (2007) (Theorem 1) show that regret is bounded by the worst-case hindsight bias on the states visited by *any* decision policy. However, there are examples (see 9.6.6) where their bound is large and one could incorrectly conclude that hindsight optimization should not be applied. In contrast, 5.6.2 is tighter, requiring that the hindsight bias be small only on states visited by $\pi^\dagger$.

As corollaries of 5.6.2, the regret of $\pi^\dagger$ is *constant* for many resource management Exo-MDPs, such as stochastic online bin packing with i.i.d. arrivals. See 9.7.1 (9.7.1) for a formal statement of the result, and a discussion of related results from the literature.

Finally, we show that the performance of the best policy produced by 1 will converge to that of $\pi^\dagger$ under standard assumptions for online imitation learning (Sun et al., 2017; Ross et al., 2011; Yan et al., 2021). We use the overline notation to denote quantities for an empirical MDP whose exogenous distribution $\mathcal{P}_\Xi$ is replaced with the empirical one $\overline{\mathcal{P}}_\Xi \sim \mathcal{D}$.

Theorem 5.6.3. Let $\overline{\pi}^\dagger$ denote the hindsight planning surrogate policy for the empirical Exo-MDP w.r.t. $\mathcal{D}$. Assume $\overline{\pi}^\dagger \in \Pi$ and 1 achieves no-regret in the optimization problem of Equa-

tion 9. Let π be the best policy from 1. Then, for any $\delta \in (0, 1)$, with probability $1 - \delta$,

$$\begin{aligned} \text{REGRET}(\pi) &\leq 2T \sqrt{\frac{2 \log(2|\Pi|/\delta)}{N}} + \\ &\quad \sum_{t=1}^T \mathbb{E}_{S_t \sim \overline{Pr}_t^{\bar{\pi}^\dagger}} [\bar{\Delta}_t^\dagger(S_t)] + o(1), \end{aligned}$$

for $\bar{\Delta}_t^\dagger$ the sample average of Eqn. 11, and $\overline{Pr}_t^{\bar{\pi}^\dagger}$ is the state probability of $\bar{\pi}^\dagger$ in the empirical MDP.

In 9.5 we also derive sample complexity results for PTO and RL when applied to Exo-MDPs. In PTO (see 9.5.1) the guarantees scale quadratically in T and depend on the complexity of $\mathcal{P}_\Xi$ (which in the worst case can scale by $|\Xi|^T$ if the ξ_t are strongly correlated across t). In contrast, 5.6.3 scales linearly in T and is only affected by the randomness over the induced Q^+ values and not directly by the complexity of $\mathcal{P}_\Xi$. Unlike guarantees for RL (see 9.5.2), 5.6.3 is not asymptotically consistent due to the hindsight bias. However, RL methods have asymptotic consistency *only if* they converge to the optimal policy in the empirical MDP. This convergence is an idealized computation assumption that hides optimization issues when studying statistical guarantees and is incomparable to 5.3.2 for the hindsight planner (for which we show several examples in 9.4).

Although hindsight bias is small for many practical Exo-MDPs, this is not universally true as we now show. Since hindsight bias bounds the regret of $\pi^\dagger$ (5.6.2), Exo-MDPs with large regret must also have large hindsight bias.

Theorem 5.6.4. *There exists a set of Exo-MDPs such that $\text{REGRET}(\pi^\dagger) \geq \Omega(T)$.*

Hence, for an arbitrary Exo-MDP the hindsight bias needs to be properly controlled to suc-

Table 5.1: Performance of heuristics, $\pi^\dagger$, RL and HL algorithms on multi-secretary and ARM problems benchmarked against the optimal policy. Values are V^π , the performance of the compared policy evaluated using the Bellman equations, and error bars computed via a standard normal approximation averaging over the randomly sampled dataset. Since these are tabular problems, HINDSIGHT MAC and HINDSIGHT Q-DISTILLATION are identical, so we report the performance of both as HINDSIGHT MAC. Relative performance compared against V^{π^*} is shown in parenthesis.

Multi-Secretary	$T = 5$			T
π^*	2.22		5.09	
$\pi^\dagger$	2.21	(−0.5%)	4.95	
Greedy	1.67	(−24.8%)	3.81	
Tabular Q-learning	1.67 ± 0.0032	(−24.8%)	3.81 ± 0.0037	
Hindsight MAC	2.17 ± 0.0040	(−2.4%)	4.98 ± 0.0035	
ARM	$T = 5$			T
π^*	1.89		3.72	
$\pi^\dagger$	1.88	(−0.3%)	3.61	
Greedy	1.39	(−26.5%)	2.50	
Tabular Q-learning	1.28 ± 0.015	(−32.2%)	2.75 ± 0.064	
Hindsight MAC	1.81 ± 0.032	(−4.0%)	3.30 ± 0.095	

successfully leverage hindsight planning (Chong et al., 2000). The information relaxation literature (Brown and Smith, 2014; El Shar and Jiang, 2020) subtracts a carefully chosen baseline $b(s, a, \xi)$ in special cases of 7; viewed through our results, this procedure essentially reduces the hindsight bias of the eventual $\pi^\dagger$. Building on this technique, we anticipate future works to design HL variants that are more robust to hindsight bias.

5.7 Experiments

We evaluate Hindsight Learning on three resource management domains with different characteristics (our code is available at <https://github.com/seanrsinclair/hindsight-learning>). First, **Multi-Secretary**, where the exogenous inputs are the arriving candidates’ qualities and the hindsight bias is negligible (see Theorem 4.2 of Banerjee et al. (2020)). Next we consider **Airline**

Revenue Management where the exogenous inputs are the current request’s (resource demands, revenue) and the hindsight bias is small (see 9.7.1). Lastly, we consider **VM Allocation** where the exogenous inputs are VM requests and the hindsight bias is unknown. In section 9.3.2 and section 9.4, we show explicit constructions of the Exo-MDP and hindsight planner for each domain. For the first two domains, we use traces drawn from benchmark distributions and evaluate π^+ using Monte-Carlo rollouts. For the VM allocation domain we use real-world historical traces extracted from a large public cloud provider.

5.7.1 Multi-Secretary Problems

Multi-secretary is the generalization of the classic secretary problem (Buchbinder et al., 2009), where T candidates arrive sequentially but only B can be selected. An arriving candidate at time t has ability $r_t \in (0, 1]$ drawn i.i.d. from a finite set of K levels of expertise. At each round, if the decision-maker has remaining budget (i.e., has chosen less than B candidates thus far), they can *accept* a candidate and collect the reward r_t , or *reject* the candidate. The goal is to maximize the expected cumulative reward.

When T is large relative to N , we can expect historical traces to provide sufficient information about $\mathcal{P}_\Xi$. Recent results in Banerjee et al. (2020) use the “Bayes Selector” with a single exogenous trace to derive a policy with constant regret for a sufficiently large T . This suggests that the hindsight bias is negligible in this regime. Our experiment setup is identical to Banerjee et al. (2020) and is included in the supplementary material. We use $T = \{5, 10, 100\}$, $B = \frac{3}{5}T$, $K = 4$ and $N = 1$. The Greedy heuristic accepts the first B candidates regardless of their quality. ML methods use a single trace sampled from the non-stationary candidate arrival process, and use a

policy that maps a 3-dim state (the rounds and budget remaining, and the current candidate ability) to an *accept* probability. For the hindsight planner, we use Equation 2 from Banerjee et al. (2020) which implements a linear program with $2K$ variables. The Bayes Selector π^+ solves the LP with the historical trace in every possible state, and is only feasible for problems with small LPs and state spaces. We evaluate each policy using dynamic programming with the true arrivals distribution.

In 5.1 (Top) we see that the HL algorithm (Hindsight MAC) is competitive with the optimal policy (which depends on the unknown $\mathcal{P}_\Xi$ distribution) using just a *single exogenous trace*. RL (implemented via Tabular Q-learning) however is very sample inefficient; for small $T \leq 10$ it performs no better than the sub-optimal Greedy heuristic.

5.7.2 Airline Revenue Management

Airline Revenue Management (Littlewood, 1972) is a special case of the multi-dimensional Online Bin Packing (OBP) problem (OBP exhibits vanishing hindsight bias via 9.7.1). The agent has capacity B_k for K different resources. At each round, the decision-maker observes a request $A_t \in \mathbb{R}_+^K$ (the consumed capacity in each resource dimension), alongside a revenue f_t . The algorithm can either *accept* the request (obtaining revenue f_t and updating remaining capacity according to A_t), or *reject* it (note that partial acceptance is not allowed). The goal of the decision-maker is to maximize the expected revenue.

We use ORSuite (Archer et al., 2022) as an ARM simulator with fixed capacity, i.i.d. request types and job distribution, using a setting from Vera and Banerjee (2021) which shows large regret for existing heuristics. We vary T from 5 to 100, and compute π^* through dynamic

programming. Both RL (Tabular Q-learning) and HL (Hindsight MAC) were trained on the same dataset of $N = 100$ traces.

In 5.1 (Bottom) we see that HL outperforms RL but is not as good as the Bayes Selector π^+ . Since the state space is much larger in ARM, HL has not sampled all the relevant states for imitating π^+ and so its performance suffers. Moreover, as T increases the performance of RL again approaches HL, highlighting that HL strikes a better bias-variance trade-off to perform better with limited data.

5.7.3 VM Allocation

Arguably, our experiments thus far have been advantageous to HL because the hindsight bias is known to be small. We next examine HL on a large-scale allocation problem: allocating virtual machines (VMs) to physical servers. In contrast to previous experiments, in this problem, the bias can be arbitrary, and enumerating all states or solving the Bellman equations is infeasible. From an algorithmic perspective, the allocation problem is a multi-dimensional variant of OBP with stochastic (not i.i.d.) arrivals and departures (hence, the bound on hindsight bias does not apply). Due to problem scale we cannot compute π^* exactly with dynamic programming, so we instead benchmark a policy’s performance with respect to a **BestFit** heuristic.

In the VM allocation problem we have K physical machines (PM), each with a fixed capacity limit for both CPU and memory. VM requests arrive over time, each with an associated CPU, memory requirement, and a lifetime (or duration); the lifetime is in principle unknown to the provider, but it can be predicted (Cortez et al., 2017). Accordingly, we study below two variants where lifetime information is either available (5.7.3.1) or not (5.7.3.2). The decision-maker

must assign a feasible PM for the VM or reject the request (incurring a large penalty). A PM is considered active when one or more VMs are assigned to it. The objective is to minimize the total time that the machines remain active, normalized by the time horizon T (i.e., the average number of active PMs per unit time). This objective is critical for cloud efficiency; see [Buchbinder et al. \(2021\)](#) for a longer discussion.

5.7.3.1 Stylized environment

To gain insights into the problem domain, we consider first a stylized setting where the VMs arrive at discrete time steps (in practice, time is continuous, and VMs may arrive at any point in time); furthermore, the VM lifetime is perfectly predicted upon arrival. To carry out the experiments, we use the MARO simulator ([Jiang et al., 2020b](#)) with $K = 80$ PMs and $T = 288$ (reflecting one day period discretized into time steps, each of which represents 5 minutes of actual time). MARO replays the VM requests in the Azure Public Dataset ([Cortez et al., 2017](#))⁴ as follows: all VM requests arriving within a time step (i.e., 5 minutes of actual time) are buffered and instead arrive simultaneously at the next discrete time step. The first half of the resulting trace is used for training and the remaining trace for testing. To evaluate any policy, we sampled 50 different one-day traces from the held-out portion and report the average value of the objective function. For the hindsight planner, we implemented the integer program of 9.4.5 in Gurobi and solved its linear relaxation. We used a modified objective function (inverse packing density) which is linear in the decision variables for computational feasibility (see discussion in 9.7).

We compare four allocation approaches. (1) **Heuristics:** We consider several heuristics

⁴This dataset contains a uniform sample of VM requests received in a real data center over a one month period in 2019.

Table 5.2: Performance of heuristics, RL, and HL algorithms on VM allocation benchmarked against the Best Fit baseline. $\star$ indicate significant improvement and $\circ$ indicate significant decrease, over BestFit by Welch’s t -test.

Algorithm	PMs Saved
Performance Upper Bound (Oracle)	4.96 $\star$
Best Fit	0.0
Bin Packing	−1.05 $^\circ$
DQN	−0.64
MAC	−0.51 $^\circ$
PPO	−0.50
PG with Hindsight Baseline (Mao et al., 2019b)	−0.057
HINDSIGHT MAC	4.33$\star$
HINDSIGHT Q-DISTILLATION	3.71 $\star$

that have been widely used for different bin packing problems (round robin, first fit, load balance, etc.). We report here the results for the best performing heuristic *BestFit*, which has been widely applied in practice (Panigrahy et al., 2011), in particular for VM allocation (Hadary et al., 2020); in a nutshell BestFit chooses the machine which leaves less amount of unused resources (see (Panigrahy et al., 2011) for details). (2) **RL**: We benchmark several popular RL algorithms including Double-DQN (Van Hasselt et al., 2016), MAC (Allen et al., 2017) and PPO (Schulman et al., 2017). (3) **Hindsight approaches**: We test Hindsight MAC (Equation 9) and Hindsight Q-Distillation (Equation 10). In addition, we test Mao et al. (2019b) which uses hindsight-aware control variates to reduce the variance of policy gradient (PG) methods. (4) **Oracle**: We report the $\text{HINDSIGHT}(1, \xi, s_1)$ (objective of the relaxed IP) evaluated on the test traces, and use the experiment outcome as a performance upper bound.

All the ML methods use a 4-layer neural net to map features describing a PM and the VM request to a score. In 9.7, we detail the network design, state features and the hyper-parameter ranges we used. 5.2 reports the *PMs Saved* which is the regret for the objective function relative

to *BestFit*, averaged across the evaluation traces. We created realistic starting state distributions by executing the *BestFit* heuristic for a random duration (greater than one day). Error bars are computed by (i) training each algorithm over 20 random seeds (neural network parameters and the offline dataset) and (ii) evaluating each algorithm on 50 one-day traces sampled from the hold-out set. We then compared its performance to Best Fit on each evaluation trace with a paired t -test of value $p = 0.05$. We observe that HL outperforms all the heuristics and RL methods, requiring 4 fewer PMs on average (or a 5% improvement in relative terms, since $K = 80$).

5.7.3.2 Real-World Resource Allocation

We now consider a more realistic setting where VM arrivals are in continuous time and the allocation agent has no information about VM lifetimes. In real-world settings, the scale of clusters can be much larger than the one considered in 5.7.3.1, see Hadary et al. (2020); scaling ML algorithms to larger inventory sizes is an ongoing research direction. Furthermore, each VM arrival or departure is modeled as a step in the Exo-MDP, resulting in much larger time horizon T (order of 100k). Our total trace period was 88 days, and we used the exact methodology as in 5.7.3.1 to obtain the training and test datasets. Due to the large scale, even the linear relaxation of the integer program was not tractable. Consequently, we carefully designed a *hindsight heuristic* (3) to derive $\text{HINDSIGHT}(t, \xi, s)$. The heuristic prioritizes VMs according to both their size and lifetime (see 9.7.6.3).

We adapt HINDSIGHT MAC (HL) and compare it with MAC Allen et al. (2017) (RL), where both used the same network architecture, which embeds VM-specific and PM-specific features using a 6-layer GNN. The resulting architecture is rich enough to represent the **BestFit**

Table 5.3: Average number of PMs saved by RL and HL policies across 5 clusters, calculated over 44 days and benchmarked against the production **BestFit** heuristic. $\star$ indicate significant improvement and $\circ$ indicate a significant decrease, over BestFit by Welch’s t -test.

Cluster	A	B	C	D	E
RL	−0.14	−0.35	−0.27 $\circ$	1.34 $\star$	−0.37
HL	3.20$\star$	1.35	1.13$\star$	2.27$\star$	0.02

heuristic, but can also express more flexible policies. The Bayes Selector $\pi^\dagger$ is infeasible to run within the latency requirements for VM allocation, and so is not compared.

5.3 summarizes the results over five different clusters. Unlike 5.7.3.1 where we sampled many 1-day periods from the test trace, the demands on the real clusters were non-stationary throughout the test period. Hence we report results on the entire 44-day test trace. We trained each algorithm over 3 random seeds and evaluated 5 rollouts to capture the variation in the cluster state at the start of the evaluation trace. Unlike the other experiments, we cannot account for the randomness in exogenous samples because we only have one evaluation trace for each cluster. Error metrics are computed with a paired t -test of value $p = 0.05$.

We observe that RL exhibits unreliable performance: in fact, it is sometimes worse than **BestFit**, intuitively this can happen because it overfits to the request patterns seen during training. In contrast, HL always improved over **BestFit**, with relative improvements of 0.1% – 1.6% over RL and 0.1% – 0.7% over BestFit (note cluster sizes are much larger here than 5.7.3.1). As noted earlier, any percent-point (or even fractions of a percent) improvement implies millions of dollars in savings. The relative gains obtained here are more modest than in the stylized setting due to a combination of reasons. First, intuitively, every packing “mistake” is more costly in a smaller cluster, meaning that algorithms have more room to shine in smaller-scale problems. Second, using a heuristic for hindsight learning is inherently sub-optimal. Lastly, we have not used *any*

information about VM lifetime; an interesting direction for future work is incorporating lifetime predictions to HL.

5.8 Chapter Summary

In this chapter, we introduced Hindsight Learning (HL) as a family of algorithms that solve a subclass of MDPs with exogenous inputs, termed Exo-MDPs. Exo-MDPs capture a variety of important resource management problems, such as VM allocation. We show that the HL algorithms outperform both heuristics and RL methods. One direction for future work is to blend RL with HL using reward shaping [Cheng et al. \(2021\)](#) for solving Exo-MDPs with large hindsight bias. Intuitively, combining pessimistic value estimates from RL with optimistic estimates from HL can provide finer-grained control for trading-off hindsight bias and variance from exogenous inputs. Another direction is designing hindsight learning algorithms in “nearly Exo-MDP” environments where the action can have a limited impact on the exogenous variables, such as using recent results from [Liu et al. \(2021\)](#).

My main contribution to this chapter includes designing and implementing a method for stabilizing the performance of Hindsight Learning algorithms in practice, and part of the empirical experiments for real-world resource allocation problems. The publication of this chapter has drawn people’s attention to investigating problems in *Exo-MDPs* [Madeka et al. \(2022\)](#); [Lee et al. \(2023\)](#), and inspired follow-up works utilizing the hindsight information for resource management problems [Alvo et al. \(2023\)](#).

Appendix 6: Appendix: What Can Neural Networks Reason About?

6.1 Proofs and Additional Experimental Details

6.1.1 Proof of Proposition 2.3.1

We will prove the universal approximation of GNNs by showing that GNNs have at least the same expressive power as Deep Sets, and then apply the universal approximation of Deep Sets for permutation invariant continuous functions.

Zaheer et al. [Zaheer et al. \(2017\)](#) prove the universal approximation of Deep Sets under the restriction that the set size is fixed and the hidden dimension is equal to the set size plus one. Wagstaff et al. [Wagstaff et al. \(2019\)](#) extend the universal approximation result for Deep Sets by showing that the set size does not have to be fixed and the hidden dimension is only required to be at least as large as the set size. The results for our purposes can be summarized as follows.

Universal approximation of Deep Sets. Assume the elements are from a compact set in $\mathbb{R}^d$. Any continuous function on a set S of size bounded by N , *i.e.*, $f : \mathbb{R}^{d \times N} \rightarrow \mathbb{R}$, that is permutation invariant to the elements in S can be approximated arbitrarily close by some Deep Sets model with sufficiently large width and output dimension for its MLPs.

Next we show any Deep Sets can be expressed by some GNN with one message passing

iteration. The computation structure of one-layer GNNs is shown below.

$$h_s = \sum_{t \in S} \phi(X_s, X_t), \quad h_S = g\left(\sum_{s \in S} h_s\right), \quad (1)$$

where ϕ and g are parameterized by MLPs. If ϕ is a function that ignores X_t so that $\phi(X_s, X_t) = \rho(X_s)$ for some ρ , *e.g.*, by letting part of the weight matrices in ϕ be 0, then we essentially get a Deep Sets in the following form.

$$h_s = \rho(X_s), \quad h_S = g\left(\sum_{s \in S} h_s\right). \quad (2)$$

For any such ρ , we can get the corresponding ϕ via the construction above. Hence for any Deep Sets, we can express it with an one-layer GNN. The same result applies to GNNs with multiple layers (message passing iterations), because we can express a function $\rho(X_s)$ by the composition of multiple $\rho^{(k)}$'s, which we can express with a GNN layer via our construction above. It then follows that GNNs are universal approximators for permutation invariant continuous functions.

6.1.2 Proof of Proposition [2.3.2](#)

For any GNN $\mathcal{N}$, we construct an MLP that is able to do the exact same computation as $\mathcal{N}$. It will then follow that the MLP can represent any function $\mathcal{N}$ can represent. Suppose the computation structure of $\mathcal{N}$ is the following.

$$h_s^{(k)} = \sum_{t \in S} f^{(k)}\left(h_s^{(k-1)}, h_t^{(k-1)}\right), \quad h_S = g\left(\sum_{s \in S} h_s^{(K)}\right), \quad (3)$$

where f and g are parameterized by MLPs. Suppose the set size is bounded by M (the expressive power of GNNs also depend on M [Wagstaff et al. \(2019\)](#)). We first show the result for a fixed size input, *i.e.*, MLPs can simulate GNNs if the input set has a fixed size, and then apply an ensemble approach to deal with variable sized input.

Let the input to the MLP be a vector concatenated by $h_s^{(0)}$'s, in some arbitrary ordering. For each message passing iteration of $\mathcal{N}$, any $f^{(k)}$ can be represented by an MLP. Thus, for each pair of $(h_t^{(k-1)}, h_s^{(k-1)})$, we can set weights in the MLP so that the the concatenation of all $f(h_t^{(k-1)}, h_s^{(k-1)})$ become the hidden vector after some layers of the MLP. With the vector of $f(h_t^{(k-1)}, h_s^{(k-1)})$ as input, in the next few layers of the MLP we can construct weights so that we have the concatenation of $h_s^{(k)} = \sum_{t \in S} f^{(k)}(h_s^{(k-1)}, h_t^{(k-1)})$ as the result of the hidden dimension, because we can encode summation with weights in MLPs. So far, we can simulate an iteration of GNN $\mathcal{N}$ with layers of MLP. We can repeat the process for K times by stacking the similar layers. Finally, with a concatenation of $h_s^{(K)}$ as our hidden dimension in the MLP, similarly, we can simulate $h_S = g\left(\sum_{s \in S} h_s^{(K)}\right)$ with layers of MLP. Stacking all layers together, we have obtained an MLP that can simulate $\mathcal{N}$.

To deal with variable sized inputs, we construct M MLPs that can simulate the GNN for each input set size $1, \dots, M$. Then we construct a meta-layer, whose weights represent (universally approximate) the summation of the output of M MLPs multiplied by an indicator function of whether each MLPs has the same size as the set input (these need to be input information). The meta layer weights on top can then essentially select the output from of MLP that has the same size as the set input and then exactly simulate the GNN. Note that the MLP we construct here has the requirement for how we input the data and the information of set sizes etc. In practice, we can have M MLPs and decide which MLP to use depending on the input set size.

6.1.3 Proof of Theorem 2.3.5

Theorem 2.3.5 is a generalization of Theorem 6.1 in Arora et al. (2019a), which addresses the scalar case. See Arora et al. (2019a) for a complete list of assumptions.

Theorem 6.1.1. *Arora et al. (2019a) Suppose we have $g : \mathbb{R}^d \rightarrow \mathbb{R}$, $g(x) = \sum_j \alpha_j (\beta_j^\top x)^{p_j}$, where $\beta_j \in \mathbb{R}^d$, $\alpha \in \mathbb{R}$, and $p_j = 1$ or $p_j = 2l$ ($l \in \mathbb{N}_+$). Let $\mathcal{A}$ be an overparameterized two-layer MLP that is randomly initialized and trained with gradient descent for a sufficient number of iterations. The sample complexity $\mathcal{C}_{\mathcal{A}}(g, \epsilon, \delta)$ is $O\left(\frac{\sum_j p_j |\alpha_j| \cdot \|\beta_j\|_2^{p_j} + \log(1/\delta)}{\epsilon^2}\right)$.*

To extend the sample complexity bound to vector-valued functions, we view each entry/component of the output vector as an independent scalar-valued output. We can then apply a union bound to bound the error rate and failure probability for the output vector, and thus, bound the overall sample complexity.

Let ϵ and δ be the given error rate and failure probability. Moreover, suppose we choose some error rate ϵ_0 and failure probability δ_0 for the output/function of each entry. Applying Theorem 6.1.1 to each component

$$g(x)^{(i)} = \sum_j \alpha_j^{(i)} \left(\beta_j^{(i)\top} x \right)^{p_j^{(i)}} =: g_i(x) \quad (4)$$

yields a sample complexity bound of

$$\mathcal{C}_{\mathcal{A}}(g_i, \epsilon_0, \delta_0) = O\left(\frac{\sum_j p_j^{(i)} |\alpha_j^{(i)}| \cdot \|\beta_j^{(i)}\|_2^{p_j^{(i)}} + \log(1/\delta_0)}{\epsilon_0^2}\right) \quad (5)$$

for each $g_i(x)$. Now let us bound the overall error rate and failure probability given ϵ_0 and δ_0 for

each entry. The probability that we fail to learn each of the g_i is at most δ_0 . Hence, by a union bound, the probability that we fail to learn any of the g_i is at most $m \cdot \delta_0$. Thus, with probability at least $1 - m\delta_0$, we successfully learn all g_i for $i = 1, \dots, m$, so the error for every entry is bounded by ϵ_0 . The error for the vector output is then at most $\sum_{i=1}^m \epsilon_0 = m\epsilon_0$.

Setting $m\delta_0 = \delta$ and $m\epsilon_0 = \epsilon$ gives us $\delta_0 = \frac{\delta}{m}$ and $\epsilon_0 = \frac{\epsilon}{m}$. Thus, if we can successfully learn the function for each output entry independently with error ϵ/m and failure rate δ/m , we can successfully learn the entire vector-valued function with rate ϵ and δ . This yields the following overall sample complexity bound:

$$\mathcal{C}_{\mathcal{A}}(g, \epsilon, \delta) = O\left(\frac{\max_i \sum_j p_j^{(i)} |\alpha_j^{(i)}| \cdot \|\beta_j^{(i)}\|_2^{p_j^{(i)}} + \log(m/\delta)}{(\epsilon/m)^2}\right) \quad (6)$$

Regarding m as a constant, we can further simplify the sample complexity to

$$\mathcal{C}_{\mathcal{A}}(g, \epsilon, \delta) = O\left(\frac{\max_i \sum_j p_j^{(i)} |\alpha_j^{(i)}| \cdot \|\beta_j^{(i)}\|_2^{p_j^{(i)}} + \log(1/\delta)}{\epsilon^2}\right). \quad (7)$$

6.1.4 Proof of Theorem 2.3.6

We will show the learnability result by an inductive argument. Specifically, we will show that under our setting and assumptions, the error between the learned function and correct function on the test set will not blow up after the transform of another learned function $\hat{f}_j$, assuming learnability on previous $\hat{f}_1, \dots, \hat{f}_{j-1}$ by induction. Thus, we can essentially provably learn at all layers/iterations and eventually learn g .

Suppose we have performed the sequential learning. Let us consider what happens at the

test time. Let f_j be the *correct functions* as defined in the algorithmic alignment. Let $\hat{f}_j$ be the functions learned by algorithm $\mathcal{A}_j$ and MLP $\mathcal{N}_j$. We have input $S \sim \mathcal{D}$, and our goal is to bound $\|g(S) - \hat{g}(S)\|$ with high probability. To show this, we bound the error of the intermediate representation vectors, *i.e.*, the output of $\hat{f}_j$ and f_j , and thus, the input to $\hat{f}_{j+1}$ and f_{j+1} .

Let us first consider what happens for the first module $\mathcal{N}_1$. f_1 and $\hat{f}_1$ have the same input distribution $x \sim \mathcal{D}$, where x are obtained from S , *e.g.*, the pairwise object representations as in Eqn. 2. Hence, by the learnability assumption on $\mathcal{A}_1$, $\|f_1(x) - \hat{f}_1(x)\| < \epsilon$ with probability at least $1 - \delta$. The error for the input of $\mathcal{N}_2$ is then $O(\epsilon)$ with failure probability $O(\delta)$, because there are a constant number of terms of aggregation of f_1 's output, and we can apply union bound to upper bound the failure probability.

Next, we proceed by induction. Let us fix a k . Let z denote the input for f_k , which are generated by the previous f_j 's, and let $\hat{z}$ denote the input for $\hat{f}_k$, which are generated by the previous $\hat{f}_j$'s. Assume $\|z - \hat{z}\| \leq O(\epsilon)$ with failure probability at most $O(\delta)$. We aim to show that this holds for $k + 1$. For the simplicity of notation, let f denote the correct function f_k and let $\hat{f}$ denote the learned function $\hat{f}_k$. Since there are a constant number of terms for aggregation, our goal is then to bound $\|\hat{f}(\hat{z}) - f(z)\|$. By triangle inequality, we have

$$\|\hat{f}(\hat{z}) - f(z)\| = \|\hat{f}(\hat{z}) - \hat{f}(z) + \hat{f}(z) - f(z)\| \quad (8)$$

$$\leq \|\hat{f}(\hat{z}) - \hat{f}(z)\| + \|\hat{f}(z) - f(z)\| \quad (9)$$

We can bound the first term with the Lipschitzness assumption of $\hat{f}$ as the following.

$$\|\hat{f}(\hat{z}) - \hat{f}(z)\| \leq L_1 \|\hat{z} - z\| \quad (10)$$

To bound the second term, our key insight is that f is a *learnable correct function*, so by the learnability coefficients in algorithmic alignment, it is close to the function $\tilde{f}$ learned by the learning algorithm $\mathcal{A}$ on the *correct samples*, i.e., f is close to $\tilde{f} = \mathcal{A}(\{z_i, y_i\})$. Moreover, $\hat{f}$ is generated by the learning algorithm $\mathcal{A}$ on the perturbed samples, i.e., $\hat{f} = \mathcal{A}(\{\hat{z}_i, y_i\})$. By the algorithm stability assumption, $\hat{f}$ and $\tilde{f}$ should be close if the input samples are only slightly perturbed. It then follows that

$$\|\hat{f}(z) - f(z)\| = \|\hat{f}(z) - \tilde{f}(z) + \tilde{f}(z) - f(z)\| \quad (11)$$

$$\leq \|\hat{f}(z) - \tilde{f}(z)\| + \|\tilde{f}(z) - f(z)\| \quad (12)$$

$$\leq L_0 \max_i \|z_i - \hat{z}_i\| + \epsilon \quad \text{w.p.} \geq 1 - \delta \quad (13)$$

where z_i and $\hat{z}_i$ are the training samples at the same layer k . Here, we apply the same induction condition as what we had for z and $\hat{z}$: $\|z_i - \hat{z}_i\| \leq O(\epsilon)$ with failure probability at most $O(\delta)$. We can then apply union bound to bound the probability of any bad event happening. Here, we have 3 bad events each happening with probability at most $O(\delta)$. Thus, with probability at least $1 - O(\delta)$, we have

$$\|\hat{f}(\hat{z}) - f(z)\| \leq L_1 O(\epsilon) + L_0 O(\epsilon) + \epsilon = O(\epsilon) \quad (14)$$

This completes the proof.

6.1.5 Proof of Corollary 2.3.7

Our main insight is that a giant MLP learns the same function $(X_i - X_j)^2$ for ℓ^2 times and encode them in the weights. This leads to the $O(\ell^2)$ extra sample complexity through Theorem 2.3.5, because the number of polynomial terms $(X_i - X_j)^2$ is of order ℓ^2 .

First of all, the function $f(x, y) = (x - y)^2$ can be expressed as the following polynomial.

$$(x - y)^2 = \left([1 \ -1]^\top [x \ y] \right)^2 \quad (15)$$

We have $\beta = [1 \ -1]$, so $p \cdot \|\beta\|^p = 4$. Hence, by Theorem 2.3.5, it takes $O(\frac{\log(1/\delta)}{\epsilon^2})$ samples for an MLP to learn $f(x, y) = (x - y)^2$. Under the sequential training setting, an one-layer GNN applies an MLP to learn f , and then sums up the outcome of $f(X_i, X_j)$ for all pairs X_i, X_j . Here, we essentially get the aggregation error $O(\ell^2 \cdot \epsilon)$ from ℓ^2 pairs. However, we will see that applying an MLP to learn g will also incur the same aggregation error. Hence, we do not need to consider the aggregation error effect when we compare the sample complexities.

Now we consider using MLP to learn the function g . No matter in what order the objects X_i are concatenated, we can express g with the sum of polynomials as the following.

$$g(S) = \sum_{ij} (\beta_{ij}^\top [X_1, \dots, X_n])^2, \quad (16)$$

where β_{ij} has 1 at the i -th entry, -1 at the j -th entry and 0 elsewhere. Hence $\|\beta_{ij}\|^p \cdot p = 4$. It then follows from Theorem 2.3.5 and union bound that it takes $O((\ell^2 + \log(1/\hat{\delta}))/\hat{\epsilon}^2)$ to learn g , where $\hat{\epsilon} = \ell^2 \epsilon$ and $\hat{\delta} = \ell^2 \delta$. Here, as we have discussed above, the same aggregation error $\hat{\epsilon}$

occurs in the aggregation process of f , so we can simply consider $\hat{\epsilon}$ for both. Thus, comparing $O(\log(1/\hat{\delta})/\hat{\epsilon}^2)$ and $O((\ell^2 + \log(1/\hat{\delta}))/\hat{\epsilon}^2)$ gives us the $O(\ell^2)$ difference.

6.1.6 Proof of Claim 2.4.1

We prove the claim by contradiction. Suppose there exists f such that $f(x) + f(y) = g(x, y)$ for any x and y . This implies that for any x , we have $f(x) + f(x) = g(x, x) = 0$. It follows that $f(x) = 0$ for any x . Now consider some x and y so that $x \neq y$. We must have $f(x) + f(y) = 0 + 0 = 0$. However, $g(x, y) \neq 0$ because $x \neq y$. Hence, there exists x and y so that $f(x) + f(y) \neq g(x, y)$. We have reached a contradiction.

6.2 Additional Experimental Details

6.2.1 Fantastic Treasure: Maximum Value Difference

Dataset generation. In the dataset, we sample 50,000 training data, 5,000 validation data, and 5,000 test data. For each model, we report the test accuracy with the hyperparameter setting that achieves the best validation accuracy. In each training sample, the input universe consists of 25 treasures $X_1, \dots, X_{25}$. For each treasure X_i , we have $X_i = [h_1, h_2, h_3]$, where the location h_1 is sampled uniformly from $[0..20]^8$, the value h_2 is sample uniformly form $[0..100]$, and the color h_3 is sampled uniformly from $[1..6]$. The task is to answer what the difference is in value between the most and least valuable treasure. We generate the answer label y for a universe S as follows: we find the the maximum difference in value among all treasures and set it to y . Then we make the label y into one-hot encoding with $100 + 1 = 101$ classes.

Hyperparameter setting. We train all models with the Adam optimizer, with learning

rate from $1e-3$, $5e-4$, and $1e-4$, and we decay the learning rate by 0.5 every 50 steps. We use cross-entropy loss. We train all models for 150 epochs. We tune batch size of 128 and 64.

For GNNs and HRN, we choose the hidden dimension of MLP modules from 128 and 256. For DeepSet and MLP, we choose the hidden dimension of MLP modules from 128, 256, 2500, 5000. For the MLP and DeepSet model, we choose the number of hidden layers for MLP modules from 4 and 8, 16. For GNN and HRN, we set the number of hidden layers of the MLP modules to 3, 4. Moreover, dropout with rate 0.5 is applied before the last two hidden layers of MLP_1 , *i.e.*, the last MLP module in all models.

6.2.2 Fantastic Treasure: Furthest Pair

Dataset generation. In the dataset, we sample 60,000 training data, 6,000 validation data, and 6,000 test data. For each model, we report the test accuracy with the hyperparameter setting that achieves the best validation accuracy. In each training sample, the input universe consists of 25 treasures $X_1, \dots, X_{25}$. For each treasure X_i , we have $X_i = [h_1, h_2, h_3]$, where the location h_1 is sampled uniformly from $[0..20]^8$, the value h_2 is sample uniformly form $[0..100]$, and the color h_3 is sampled uniformly from $[1..6]$. The task is to answer what are the colors of the two treasure that are the most distant from each other. We generate the answer label y for a universe S as follows: we find the pair of treasures that are the most distant from each other, say (X_i, X_j) . Then we order the pair $(h_3(X_i), h_3(X_j))$ to obtain an ordered pair (a, b) with $a \leq b$ (aka. $a = \min\{h_3(X_i), h_3(X_j)\}$ and $b = \max\{h_3(X_i), h_3(X_j)\}$), where $h_3(X_i)$ denotes the color of X_i . Then we compute the label y from (a, b) by counting how many valid pairs of colors are smaller than (a, b) (a pair (k, l) is smaller than (a, b) iff i). $k < a$ or ii). $k = a$ and $l < b$).

The label y is one-hot encoding of the minimum cost with $6 \times (6 - 1)/2 + 6 = 21$ classes.

Hyperparameter setting. We train all models with the Adam optimizer, with learning rate from $1e - 3$, $5e - 4$, and $1e - 4$, and we decay the learning rate by 0.5 every 50 steps. We use cross-entropy loss. We train all models for 150 epochs. We tune batch size of 128 and 64.

For the MLP and DeepSet model, we choose the number of hidden layers of MLP modules from 4 and 8, 16. For GNN and HRN models, we set the number of hidden layers of the MLP modules from 3 and 4. For DeepSet and MLP models, we choose the hidden dimension of MLP modules from 128, 256, 2500, 5000. For GNNs and HRN, we choose the hidden dimension of MLP modules from 128 and 256. Moreover, dropout with rate 0.5 is applied before the last two hidden layers of MLP_1 , *i.e.*, the last MLP module in all models.

6.2.3 Monster Trainer

Task description. We are a monster trainer who lives in a world S with 10 monsters. Each monster $X = [h_1, h_2]$ has a location $h_1 \in [0..10]^2$ and a unique combat level $h_2 \in [1..10]$. In each game, the trainer starts at a random location with level zero, $X_{\text{trainer}} = [p_0, 0]$, and receives a quest to defeat the level- k monster. At each time step, the trainer can challenge any *more powerful* monster X , with a cost equal to the product of the travel distance and the level difference $c(X_{\text{trainer}}, X) = \|h_1(X_{\text{trainer}}) - h_1(X)\|_{\ell_1} \times (h_2(X) - h_2(X_{\text{trainer}}))$. After defeating monster X , the trainer’s level upgrades to $h_2(X)$, and the trainer moves to $h_1(X)$. We ask the minimum cost of completing the quest, *i.e.*, defeating the level- k monster. The range of cost (number of classes for prediction) is 200. To make games even more challenging, we sample games whose optimal solution involves defeating three to seven non-quest monsters.

A DP algorithm for shortest paths that needs half of the iterations of Bellman-Ford.

We provide a DP algorithm as the following. To compute a shortest-path from a source object s to a target object t with at most seven stops, we run the following updates for four iterations:

$$\text{distance}_s[1][u] = \text{cost}(s, u), \quad \text{distance}_s[k][u] = \min_v \{ \text{distance}_s[k-1][v] + \text{cost}(v, u) \}, \quad (17)$$

$$\text{distance}_t[1][u] = \text{cost}(u, t), \quad \text{distance}_t[k][u] = \min_v \{ \text{distance}_t[k-1][v] + \text{cost}(u, v) \}. \quad (18)$$

Update Eqn. 17 is identical to the Bellman-Ford algorithm Eqn. 6, and $\text{distance}_s[k][u]$ is the shortest distance from s to u with at most k stops. Update Eqn. 18 is a *reverse* Bellman-Ford algorithm, and $\text{distance}_t[k][u]$ is the shortest distance from u to t with at most k stops. After running Eqn. 17 and Eqn. 18 for k iterations, we can compute a shortest path with at most $2k$ stops by enumerating a mid-point and aggregating the results of the two Bellman-Ford algorithms:

$$\min_u \{ \text{distance}_s[k][u] + \text{distance}_t[k][u] \}. \quad (19)$$

Thus, this algorithm needs *half of the iterations of Bellman-Ford*.

Dataset generation. In the dataset, we sample 200,000 training data, 6,000 validation data, and 6,000 test data. For each model, we report the test accuracy with the hyperparameter setting that achieves the best validation accuracy. In each training sample, the input universe consists of the trainer and 10 monsters $X_0, \dots, X_{10}$, and the request level k , *i.e.*, we need to challenge monster k . We have $X_i = [h_1, h_2]$, where $h_1 = i$ indicates the combat level, and the

location $h_2 \in [0..10]^2$ is sampled uniformly from $[0..10]^2$. We generate the answer label y for a universe S as follows. We implement a shortest path algorithm to compute the minimum cost from the trainer to monster k , where the cost is defined in task description. Then the label y is a one-hot encoding of minimum cost with 200 classes. Moreover, when we sample the data, we apply rejection sampling to ensure that the minimum cost’s shortest path is of length 3, 4, 5, 6, 7 with equal probability. That is, we eliminate the trivial questions.

Hyperparameter setting. We train all models with the Adam optimizer, with learning rate from $2e - 4$ and $5e - 4$, and we decay the learning rate by 0.5 every 50 steps. We use cross-entropy loss. We train all models for 300 epochs. We tune batch size of 128 and 64.

For the MLP model, we choose the number of layers from 4 and 8, 16. For other models, we choose the number of hidden layers of MLP modules from 3 and 4. For GNN models, we choose the hidden dimension of MLP modules from 128 and 256. For DeepSet and MLP models, we choose the hidden dimension of MLP modules from 128, 256, 2500. Moreover, dropout with rate 0.5 is applied before the last two hidden layers of MLP_1 , *i.e.*, the last MLP module in all models.

6.2.4 Subset Sum

Dataset generation. In the dataset, we sample 40,000 training data, 4,000 validation data, and 4,000 test data. For each model, we report the test accuracy with the hyperparameter setting that achieves the best validation accuracy. In each training sample, the input universe S consists of 6 numbers $X_1, \dots, X_6$, where each X_i is uniformly sampled from $[-200..200]$. The goal is to decide if there exists a subset that sums up to 0. In the data generation, we carefully

decrease the number of questions that have trivial answers: 1) we control the number of samples where $0 \in \{X_1, \dots, X_6\}$ to be around 1% of the total training data; 2) we further control the number of samples where $X_1 + \dots + X_6 = 0$ or $\exists i, j \in [1..6]$ so that $X_i = -X_j$ to be around 1.5% of the total training data. In addition, we apply rejection sampling to make sure that the questions with answer yes (aka. such subset exists) and answer no (aka. no such subset exists) are balanced (*i.e.*, 20,000 samples for each class in the training data).

Hyperparameter setting. We train all models with the Adam optimizer, with learning rate from $1e-3$, $5e-4$, and $1e-4$, and we decay the learning rate by 0.5 every 50 steps. We use cross-entropy loss. We train all models for 300 epochs. The batch size we use for all models is 64.

For DeepSets and MLP models, we choose the number of hidden layers of the MLP modules from 4, 8, 16. For GNN and HRN models, we set the number of hidden layers of the last MLP modules to 4. For DeepSets and MLP, we choose the hidden dimension of MLP modules from 128, 256, 2500, 5000. For GNN and HRN models, we choose the hidden dimension of MLP modules from 128 and 256. Moreover, dropout with rate 0.5 is applied before the last two hidden layers of MLP_1 , *i.e.*, the last MLP module in all models.

The model Neural Exhaustive Search (NES) enumerates all possible non-empty subsets τ of S , and passes the numbers of τ to an MLP, in a random order, to obtain the hidden feature. The hidden feature is then passed to a single-direction one-layer LSTM of hidden dimension 128. Afterwards, NES applies an aggregation function to these $2^6 - 1$ hidden states obtained by the LSTM to obtain the final output. For NES, we set the number of hidden layers of the last MLP, *i.e.*, MLP_2 , to 4, the number of hidden layers of the MLPs prior to the last MLP, *i.e.*, MLP_1 , to 3, and we choose the hidden dimension of all MLP modules from 128 and 256.

Appendix 7: Appendix: How does Neural Networks Extrapolate?

7.1 Additional Experimental Details

In this section, we describe the model, data and training details for reproducing our experiments. Our experiments support all of our theoretical claims and insights.

Overview. We classify our experiments into the following major categories, each of which includes several ablation studies:

- 1) Learning tasks where the target functions are **simple nonlinear functions** in various dimensions and training/test distributions: quadratic, cosine, square root, and l1 norm functions, with MLPs with a wide range of hyper-parameters.

This validates our implications on MLPs generally cannot extrapolate in tasks with nonlinear target functions, unless the nonlinear function is directionally linear out-of-distribution.

In the latter case, the extrapolation error is more sensitive to the hyper-parameters.

- 2) Computation of the **R-Squared** of MLP’s learned functions along (thousands of) randomly sampled directions in out-of-distribution domain.

This validates Theorem [3.3.2](#) and shows the convergence rate is very high in practice, and often happens immediately out of training range.

- 3) Learning tasks where the target functions are **linear functions** with MLPs. These validate Theorem 3.3.4 and Lemma 3.3.3, i.e., MLPs can extrapolate if the underlying function is linear under conditions on training distribution. This section includes four ablation studies:
 - a) Training distribution satisfy the conditions in Theorem 3.3.4 and cover all directions, and hence, MLPs extrapolate.
 - b) Training data distribution is **restricted** in some **directions**, e.g., restricted to be positive/negative/constant in some feature dimensions. This shows when training distribution is restrictive in directions, MLPs may fail to extrapolate.
 - c) Exact extrapolation with **infinitely-wide neural networks**, i.e., exact computation with **neural tangent kernel** (NTK) on the data regime in Lemma 3.3.3. This is mainly for theoretical understanding.
- 4) MLPs with cosine, quadratic, and tanh activation functions.
- 5) Learning **maximum degree of graphs** with Graph Neural Networks. Extrapolation on graph structure, number of nodes, and node features. To show the role of architecture for extrapolation, we study the following GNN architecture regimes.
 - a) GNN with graph-level max-pooling and neighbor-level sum-pooling. By Theorem 3.4.4, this GNN architecture extrapolates in max degree with appropriate training data.
 - b) GNN with graph-level and neighbor-level sum-pooling. By Corollary 3.4.3, this default GNN architecture cannot extrapolate in max degree.

To show the importance of training distribution, i.e., graph structure in training set, we study the following training data regimes.

- a) Node features are **identical**, e.g., 1. In such regimes, our learning tasks only consider graph structure. We consider training sets sampled from various graph structure, and find only those satisfy conditions in Theorem 3.4.4 enables GNNs with graph-level max-pooling to extrapolate.
 - b) Node features are **spurious** and continuous. This also requires extrapolation on OOD node features. GNNs with graph-level max-pooling with appropriate training sets also extrapolate to OOD spurious node features.
- 6) Learning the length of the **shortest path** between given source and target nodes, with Graph Neural Networks. Extrapolation on graph structure, number of nodes, and edge weights. We study the following regimes.

- a) Continuous features. Edge and node features are real values. This regime requires extrapolating to graphs with edge weights out of training range.

Test graphs are all sampled from the “general graphs” family with a diverse range of structure. Regarding the type of training graph structure, we consider two schemes. Both schemes show a U-shape curve of extrapolation error with respect to the sparsity of training graphs.

- a) Specific graph structure: path, cycle, tree, expander, ladder, complete graphs, general graphs, 4-regular graphs.
- b) Random graphs with a range of probability p of an edge between any two nodes. Smaller p samples sparse graphs and large p samples dense graphs.

- 7) **Physical reasoning** of the n -**Body problem** in the orbit setting with Graph Neural Net-

works. We show that GNNs on the original features from previous works fail to extrapolate to unseen masses and distances. On the other hand, we show extrapolation can be achieved via an improved representation of the input edge features. We consider the following extrapolation regimes.

- a) Extrapolation on the masses of the objects.
- b) Extrapolation on the distances between objects.

We consider the following two *input representation* schemes to compare the effects of how representation helps extrapolation.

- a) Original features. Following previous works on solving n -body problem with GNNs, the edge features are simply set to 0.
- b) Improved features. We show although our edge features do not bring in new information, it helps extrapolation.

7.1.1 Learning Simple Non-Linear Functions

Dataset details. We consider four tasks where the underlying functions are simple non-linear functions $g : \mathbb{R}^d \rightarrow \mathbb{R}$. Given an input $\mathbf{x} \in \mathbb{R}^d$, the label is computed by $y = g(\mathbf{x})$ for all $\mathbf{x}$. We consider the following four families of simple functions g .

- a) Quadratic functions $g(\mathbf{x}) = \mathbf{x}^\top A \mathbf{x}$. In each dataset, we randomly sample A . In the simplest case where $A = I$, $g(\mathbf{x}) = \sum_{i=1}^d x_i^2$.
- a) Cosine functions $g(\mathbf{x}) = \sum_{i=1}^d \cos(2\pi \cdot \mathbf{x}_i)$.

- c) Square root functions $g(\mathbf{x}) = \sum_{i=1}^d \sqrt{x_i}$. Here, the domain $\mathcal{X}$ of $\mathbf{x}$ is restricted to the space in $\mathbb{R}^d$ with non-negative value in each dimension.
- d) L1 norm functions $g(\mathbf{x}) = |\mathbf{x}|_1 = \sum_{i=1}^d |x_i|$.

We sample each dataset of a task by considering the following parameters

- a) The shape and support of training, validation, and test data distributions.
 - i) Training, validation, and test data are uniformly sampled from a hyper-cube. Training and validation data are sampled from $[-a, a]^d$ with $a \in \{0.5, 1.0\}$, i.e., each dimension of $\mathbf{x} \in \mathbb{R}^d$ is uniformly sampled from $[-a, a]$. Test data are sampled from $[-a, a]^d$ with $a \in \{2.0, 5.0, 10.0\}$.
 - ii) Training and validation data are uniformly sampled from a sphere, where every point has $L2$ distance r from the origin. We sample r from $r \in \{0.5, 1.0\}$. Then, we sample a random Gaussian vector $\mathbf{q}$ in $\mathbb{R}^d$. We obtain the training or validation data $\mathbf{x} = \mathbf{q} / \|\mathbf{q}\|_2 \cdot r$. This corresponds to uniform sampling from the sphere.

Test data are sampled (non-uniformly) from a hyper-ball. We first sample r uniformly from $[0.0, 2.0]$, $[0.0, 5.0]$, and $[0.0, 10.0]$. Then, we sample a random Gaussian vector $\mathbf{q}$ in $\mathbb{R}^d$. We obtain the test data $\mathbf{x} = \mathbf{q} / \|\mathbf{q}\|_2 \cdot r$. This corresponds to (non-uniform) sampling from a hyper-ball in $\mathbb{R}^d$.

- b) We sample 20,000 training data, 1,000 validation data, and 20,000 test data.
- c) We sample input dimension d from $\{1, 2, 8\}$.
- d) For quadratic functions, we sample the entries of A uniformly from $[-1, 1]$.

Model and hyperparameter settings. We consider the multi-layer perceptron (MLP) architecture.

$$\text{MLP}(\mathbf{x}) = \mathbf{W}^{(d)} \cdot \sigma(\mathbf{W}^{(d-1)} \sigma(\dots \sigma(\mathbf{W}^{(1)} \mathbf{x})))$$

We search the following hyper-parameters for MLPs

- a) Number of layers d from $\{2, 4\}$.
- b) Width of each $\mathbf{W}^{(k)}$ from $\{64, 128, 512\}$.
- c) Initialization schemes.
 - i) The default initialization in PyTorch.
 - ii) The initialization scheme in neural tangent kernel theory, i.e., we sample entries of $\mathbf{W}^k$ from $\mathcal{N}(0, 1)$ and scale the output after each $\mathbf{W}^{(k)}$ by $\sqrt{\frac{2}{d_k}}$, where d_k is the output dimension of $\mathbf{W}^{(k)}$.
- d) Activation function σ is set to ReLU.

We train the MLP with the mean squared error (MSE) loss, and Adam and SGD optimizer.

We consider the following hyper-parameters for training

- a) Initial learning rate from $\{5e - 2, 1e - 2, 5e - 3, 1e - 3\}$. Learning rate decays 0.5 for every 50 epochs
- b) Batch size from $\{32, 64, 128\}$.
- c) Weight decay is set to $1e - 5$.

d) Number of epochs is set to 250.

Test error and model selection. For each dataset and architecture, training hyper-parameter setting, we perform model selection via validation set, i.e., we report the test error by selecting the epoch where the model achieves the best validation error. Note that our validation sets always have the same distribution as the training sets.

We train our models with the MSE loss. Because we sample test data from different ranges, the mean absolute percentage error (MAPE) loss, which scales the error by the actual value, better measures the extrapolation performance

$$\text{MAPE} = \frac{1}{n} \left| \frac{A_i - F_i}{A_i} \right|,$$

where A_i is the actual value and F_i is the predicted value. Hence, in our experiments, we also report the MAPE.

7.1.2 R-squared for Out-of-distribution Directions

We perform linear regression to fit the predictions of MLPs along randomly sampled directions in out-of-distribution regions, and compute the R-squared (or R^2) for these directions. This experiment is to validate Theorem 3.3.2 and show that the convergence rate (to a linear function) is very high in practice.

Definition. R-squared, also known as coefficient of determination, assesses how strong the linear relationship is between input and output variables. The closer R-squared is to 1, the stronger the linear relationship is, with 1 being perfectly linear.

Datasets and models. We perform the R-squared computation on over 2,000 combinations of datasets, test/train distributions, and hyper-parameters, e.g., learning rate, batch size, MLP layer, width, initialization. These are described in Appendix 7.1.1.

Computation. For each combination of dataset and model hyper-parameters as described in Section 7.1.1, we save the trained MLP model $f : \mathbb{R}^d \rightarrow \mathbb{R}$. For each dataset and model combination, we then randomly sample 5,000 directions via Gaussian vectors $\mathcal{N}(\mathbf{0}, \mathbf{I})$. For each of these directions $\mathbf{w}$, we compute the intersection point $\mathbf{x}_w$ of direction $\mathbf{w}$ and the training data distribution support (specified by a hyper-sphere or hyper-cube; see Section 7.1.1 for details).

We then collect 100 predictions of the trained MLP f along direction $\mathbf{w}$ (assume $\mathbf{w}$ is normalized) with

$$\left\{ \left(\mathbf{x}_w + k \cdot \frac{r}{10} \cdot \mathbf{w} \right), f \left(\mathbf{x}_w + k \cdot \frac{r}{10} \cdot \mathbf{w} \right) \right\}_{k=0}^{100}, \quad (1)$$

where r is the range of training data distribution support (see Section 7.1.1). We perform linear regression on these predictions in Eqn. 1, and obtain the R-squared.

Results. We obtain the R-squared for each combination of dataset, model and training setting, and randomly sampled direction. For the tasks of learning the simple non-linear functions, we confirm that more than 96% of the R-squared results are above 0.99. This empirically confirms Theorem 3.3.2 and shows that the convergence rate is in fact fast in practice. Along most directions, MLP’s learned function becomes linear immediately out of the training data support.

7.1.3 Learning Linear Functions

Dataset details. We consider the tasks where the underlying functions are linear $g : \mathbb{R}^d \rightarrow \mathbb{R}$. Given an input $\mathbf{x} \in \mathbb{R}^d$, the label is computed by $y = g(\mathbf{x}) = A\mathbf{x}$ for all $\mathbf{x}$. For each dataset, we sample the following parameters

- a) We sample 10,000 training data, 1,000 validation data, and 2,000 test data.
- b) We sample input dimension d from $\{1, 2, 32\}$.
- c) We sample entries of A uniformly from $[-a, a]$, where we sample $a \in \{5.0, 10.0\}$.
- d) The shape and support of training, validation, and test data distributions.
 - i) Training, validation, and test data are uniformly sampled from a hyper-cube. Training and validation data are sampled from $[-a, a]^d$ with $a \in \{5.0, 10.0\}$, i.e., each dimension of $\mathbf{x} \in \mathbb{R}^d$ is uniformly sampled from $[-a, a]$. Test data are sampled from $[-a, a]^d$ with $a \in \{20.0, 50.0\}$.
 - ii) Training and validation data are uniformly sampled from a sphere, where every point has $L2$ distance r from the origin. We sample r from $r \in \{5.0, 10.0\}$. Then, we sample a random Gaussian vector $\mathbf{q}$ in $\mathbb{R}^d$. We obtain the training or validation data $\mathbf{x} = \mathbf{q} / \|\mathbf{q}\|_2 \cdot r$. This corresponds to uniform sampling from the sphere.

Test data are sampled (non-uniformly) from a hyper-ball. We first sample r uniformly from $[0.0, 20.0]$ and $[0.0, 50.0]$. Then, we sample a random Gaussian vector $\mathbf{q}$ in $\mathbb{R}^d$. We obtain the test data $\mathbf{x} = \mathbf{q} / \|\mathbf{q}\|_2 \cdot r$. This corresponds to (non-uniform) sampling

from a hyper-ball in $\mathbb{R}^d$.

e) We perform ablation study on how the training distribution support misses directions. The test distributions remain the same as in d).

i) We restrict the first dimension of any training data x_i to a fixed number 0.1, and randomly sample the remaining dimensions according to d).

ii) We restrict the first k dimensions of any training data x_i to be positive. For input dimension 32, we only consider the hyper-cube training distribution, where we sample the first k dimensions from $[0, a]$ and sample the remaining dimensions from $[-a, a]$. For input dimensions 1 and 2, we consider both hyper-cube and hyper-sphere training distribution by performing rejection sampling. For input dimension 2, we consider k from $\{1, 2\}$. For input dimension 32, we consider k from $\{1, 16, 32\}$.

iii) We restrict the first k dimensions of any training data x_i to be negative. For input dimension 32, we only consider the hyper-cube training distribution, where we sample the first k dimensions from $[-a, 0]$ and sample the remaining dimensions from $[-a, a]$. For input dimensions 1 and 2, we consider both hyper-cube and hyper-sphere training distribution by performing rejection sampling. For input dimension 2, we consider k from $\{1, 2\}$. For input dimension 32, we consider k from $\{1, 16, 32\}$.

Model and hyperparameter settings. For the regression task, we search the same set of hyperparameters as those in simple non-linear functions (Section 7.1.1). We report the test error with the same validation procedure as in Section 7.1.1.

Exact computation with neural tangent kernel Our experiments with MLPs validate Theorem 3.3.4 asymptotic extrapolation for neural networks trained in regular regimes. Here, we also validate Lemma 3.3.3, exact extrapolation with finite data regime, by training an infinitely-wide neural network. That is, we directly perform the kernel regression with the neural tangent kernel (NTK). This experiment is mainly of theoretical interest.

We sample the same test set as in our experiments with MLPs. For training set, we sample $2d$ training examples according to the conditions in Lemma 3.3.3. Specifically, we first sample an orthogonal basis and their opposite vectors $\mathbf{X} = \{\mathbf{e}_i, -\mathbf{e}_i\}_{i=1}^d$. We then randomly sample 100 orthogonal transform matrices Q via the QR decomposition. Our training samples are $Q\mathbf{X}$, i.e., multiply each point in $\mathbf{X}$ by Q . This gives 100 training sets with $2d$ data points satisfying the condition in Lemma 3.3.3.

We perform kernel regression on these training sets using a two-layer neural tangent kernel (NTK). Our code for exact computation of NTK is adapted from Arora et al. (2020); Novak et al. (2020). We verify that the test losses are all precisely 0, up to machine precision. This empirically confirms Lemma 3.3.3.

Note that due to the difference of hyper-parameter settings in different implementations of NTK, to reproduce our experiments and achieve zero test error, the implementation by Arora et al. (2020) is assumed.

7.1.4 MLPs with cosine, quadratic, and tanh Activation

This section describes the experimental settings for extrapolation experiments for MLPs with cosine, quadratic, and tanh activation functions. We train MLPs to learn the following

functions:

- a) Quadratic function $g(\mathbf{x}) = \mathbf{x}^\top A \mathbf{x}$, where A is a randomly sampled matrix.
- b) Cosine function $g(\mathbf{x}) = \sum_{i=1}^d \cos(2\pi \cdot \mathbf{x}_i)$.
- c) Hyperbolic tangent function $g(\mathbf{x}) = \sum_{i=1}^d \tanh(\mathbf{x}_i)$.
- d) Linear function $g(\mathbf{x}) = W\mathbf{x} + b$.

Dataset details. We use 20,000 training, 1,000 validation, and 20,000 test data. For quadratic, we sample input dimension d from $\{1, 8\}$, training and validation data from $[-1, 1]^d$, and test data from $[-5, 5]^d$. For cosine, we sample input dimension d from $\{1, 2\}$, training and validation data from $[-100, 100]^d$, and test data from $[-200, 200]^d$. For tanh, we sample input dimension d from $\{1, 8\}$, training and validation data from $[-100, 100]^d$, and test data from $[-200, 200]^d$. For linear, we use a subset of datasets from Appendix 7.1.3: 1 and 8 input dimensions with hyper-cube training distributions.

Model and hyperparameter settings. We use the same hyperparameters from Appendix 7.1.1, except we fix the batch size to 128, as the batch size has minimal impact on models. MLPs with cos activation is hard to optimize, so we only report models with training MAPE less than 1.

7.1.5 Max Degree

Dataset details. We consider the task of finding the maximum degree on a graph. Given any input graph $G = (V, E)$, the label is computed by the underlying function $y = g(G) =$

$\max_{u \in G} \sum_{v \in \mathcal{N}(u)} 1$. For each dataset, we sample the graphs and node features with the following parameters

- a) Graph structure for training and validation sets. For each dataset, we consider one of the following graph structure: path graphs, cycles, ladder graphs, 4-regular random graphs, complete graphs, random trees, expanders (here we use random graphs with $p = 0.8$ as they are expanders with high probability), and general graphs (random graphs with $p = 0.1$ to 0.9 with equal probability for a broad range of graph structure). We use the networkx library for sampling graphs.
- b) Graph structure for test set. We consider the general graphs (random graphs with $p = 0.1$ to 0.9 with equal probability).
- c) The number of vertices of graphs $|V|$ for training and validation sets are sampled uniformly from $[20...30]$. The number of vertices of graphs $|V|$ for test set is sampled uniformly from $[50..100]$.
- d) We consider two schemes for node features.
 - i) Identical features. All nodes in training, validation and set sets have uniform feature 1.
 - ii) Spurious (continuous) features. Node features in training and validation sets are sampled uniformly from $[-5.0, 5.0]^3$, i.e., a three-dimensional vector where each dimension is sampled from $[-5.0, 5.0]$. There are two schemes for test sets, in the first case we do not extrapolate node features, so we sample node features uniformly from

$[-5.0, 5.0]^3$. In the second case we extrapolate node features, we sample node features uniformly from $[-10.0, 10.0]^3$.

- e) We sample 5,000 graphs for training, 1,000 graphs for validation, and 2,500 graphs for testing.

Model and hyperparameter settings. We consider the following Graph Neural Network (GNN) architecture. Given an input graph G , GNN learns the output h_G by first iteratively aggregating and transforming the neighbors of all node vectors $h_u^{(k)}$ (vector for node u in layer k), and perform a max or sum-pooling over all node features h_u to obtain h_G . Formally, we have

$$h_u^{(k)} = \sum_{v \in \mathcal{N}(u)} \text{MLP}^{(k)}(h_v^{(k-1)}, h_u^{(k-1)}), \quad h_G = \text{MLP}^{(K+1)}(\text{graph-pooling}\{h_u^{(K)} : u \in G\}). \quad (2)$$

Here, $\mathcal{N}(u)$ denotes the neighbors of u , K is the number of GNN iterations, and graph-pooling is a hyper-parameter with choices as max or sum. $h_u^{(0)}$ is the input node feature of node u . We search the following hyper-parameters for GNNs

- a) Number of GNN iterations K is 1.
- b) Graph pooling is from max or sum.
- c) Width of all MLPs are set to 256.
- d) The number of layers for $\text{MLP}^{(k)}$ with $k = 1..K$ are set to 2. The number of layers for $\text{MLP}^{(K+1)}$ is set to 1.

We train the GNNs with the mean squared error (MSE) loss, and Adam and SGD optimizer.

We search the following hyper-parameters for training

- a) Initial learning rate is set to 0.01.
- b) Batch size is set to 64.
- c) Weight decay is set to $1e - 5$.
- d) Number of epochs is set to 300 for graphs with continuous node features, and 100 for graphs with uniform node features.

Test error and model selection. For each dataset and architecture, training hyper-parameter setting, we perform model selection via validation set, i.e., we report the test error by selecting the epoch where the model achieves the best validation error. Note that our validation sets always have the same distribution as the training sets. Again, we report the MAPE for test error as in MLPs.

7.1.6 Shortest Path

Dataset details. We consider the task of finding the length of the shortest path on a graph, from a given source to target nodes. Given any graph $G = (V, E)$, the node features, besides regular node features, encode whether a node is source s , and whether a node is target t . The edge features are a scalar representing the edge weight. For unweighted graphs, all edge weights are 1. Then the label $y = g(G)$ is the length of the shortest path from s to t on G .

For each dataset, we sample the graphs and node, edge features with the following parameters

- a) Graph structure for training and validation sets. For each dataset, we consider one of the following graph structure: path graphs, cycles, ladder graphs, 4-regular random graphs,

complete graphs, random trees, expanders (here we use random graphs with $p = 0.6$ which are expanders with high probability), and general graphs (random graphs with $p = 0.1$ to 0.9 with equal probability for a broad range of graph structure). We use the `networkx` library for sampling graphs.

- b) Graph structure for test set. We consider the general graphs (random graphs with $p = 0.1$ to 0.9 with equal probability).
- c) The number of vertices of graphs $|V|$ for training and validation sets are sampled uniformly from $[20...40]$. The number of vertices of graphs $|V|$ for test set is sampled uniformly from $[50..70]$.
- d) We consider the following scheme for node and edge features. All edges have continuous weights. Edge weights for training and validation graphs are sampled from $[1.0, 5.0]$. There are two schemes for test sets, in the first case we do not extrapolate edge weights, so we sample edge weights uniformly from $[1.0, 5.0]$. In the second case we extrapolate edge weights, we sample edge weights uniformly from $[1.0, 10.0]$. All node features are $[h, \mathbb{I}(v = s), \mathbb{I}(v = t)]$ with h sampled from $[-5.0, 5.0]$.
- e) After sampling a graph and edge weights, we sample source s and t by randomly sampling s, t and selecting the first pair s, t whose shortest path involves at most 3 hops. This enables us to solve the task using GNNs with 3 iterations.
- f) We sample 10,000 graphs for training, 1,000 graphs for validation, and 2,500 graphs for testing.

We also consider the ablation study of training on random graphs with different p . We

consider $p = 0.05..1.0$ and report the test error curve. The other parameters are the same as described above.

Model and hyperparameter settings. We consider the following Graph Neural Network (GNN) architecture. Given an input graph G , GNN learns the output h_G by first iteratively aggregating and transforming the neighbors of all node vectors $h_u^{(k)}$ (vector for node u in layer k), and perform a max or sum-pooling over all node features h_u to obtain h_G . Formally, we have

$$h_u^{(k)} = \min_{v \in \mathcal{N}(u)} \text{MLP}^{(k)}(h_v^{(k-1)}, h_u^{(k-1)}, w_{(u,v)}), \quad h_G = \text{MLP}^{(K+1)}\left(\min_{u \in G} h_u\right). \quad (3)$$

Here, $\mathcal{N}(u)$ denotes the neighbors of u , K is the number of GNN iterations, and for neighbor aggregation we run both min and sum. $h_u^{(0)}$ is the input node feature of node u . $w_{(u,v)}$ is the input edge feature of edge (u, v) . We search the following hyper-parameters for GNNs

- a) Number of GNN iterations K is set to 3.
- b) Graph pooling is set to min.
- c) Neighobr aggregation is selected from min and sum.
- d) Width of all MLPs are set to 256.
- e) The number of layers for $\text{MLP}^{(k)}$ with $k = 1..K$ are set to 2. The number of layers for $\text{MLP}^{(K+1)}$ is set to 1.

We train the GNNs with the mean squared error (MSE) loss, and Adam and SGD optimizer.

We consider the following hyper-parameters for training

- a) Initial learning rate is set to 0.01.
- b) Batch size is set to 64.
- c) Weight decay is set to $1e - 5$.
- d) Number of epochs is set to 250.

We perform the same model selection and validation as in Section 7.1.5.

7.1.7 N-Body Problem

Task description. The n-body problem asks a neural network to predict how n stars in a physical system evolves according to physics laws. That is, we train neural networks to predict properties of future states of each star in terms of next frames, e.g., 0.001 seconds.

Mathematically, in an n-body system $S = \{X_i\}_{i=1}^n$, such as solar systems, all n stars $\{X_i\}_{i=1}^n$ exert distance and mass-dependent gravitational forces on each other, so there were $n(n-1)$ relations or forces in the system. Suppose X_i at time t is at position $\mathbf{x}_i^t$ and has velocity $\mathbf{v}_i^t$. The overall forces a star X_i receives from other stars is determined by physics laws as the following

$$\mathbf{F}_i^t = G \cdot \sum_{j \neq i} \frac{m_i \times m_j}{\|\mathbf{x}_i^t - \mathbf{x}_j^t\|_2^3} \cdot (\mathbf{x}_j^t - \mathbf{x}_i^t), \quad (4)$$

where G is the gravitational constant, and m_i is the mass of star X_i . Then acceralation $\mathbf{a}_i^t$ is determined by the net force $\mathbf{F}_i^t$ and the mass of star m_i

$$\mathbf{a}_i^t = \mathbf{F}_i^t / m_i \quad (5)$$

Suppose the velocity of star X_i at time t is $\mathbf{v}_i^t$. Then assuming the time steps dt , i.e., difference between time frames, are sufficiently small, the velocity at the next time frame $t + 1$ can be approximated by

$$\mathbf{v}_i^{t+1} = \mathbf{v}_i^t + \mathbf{a}_i^t \cdot dt. \quad (6)$$

Given m_i , $\mathbf{x}_i^t$, and $\mathbf{v}_i^t$, our task asks the neural network to predict $\mathbf{v}_i^{t+1}$ for all stars X_i . In our task, we consider two extrapolation schemes

- a) The distances between stars $\|\mathbf{x}_i^t - \mathbf{x}_j^t\|_2$ are out-of-distribution for test set, i.e., different sampling ranges from the training set.
- b) The masses of stars m_i are out-of-distribution for test set, i.e., different sampling ranges from the training set.

Here, we use a physics engine that we code in Python to simulate and sample the inputs and labels. We describe the dataset details next.

Dataset details. We first describe the simulation and sampling of our training set. We sample 100 videos of n-body system evolution, each with 500 rollout, i.e., time steps. We consider the orbit situation: there exists a huge center star and several other stars. We sample the initial states, i.e., position, velocity, masses, acceleration etc according to the following parameters.

- a) The mass of the center star is $100kg$.
- b) The masses of other stars are sampled from $[0.02, 9.0]kg$.
- c) The number of stars is 3.

- d) The initial position of the center star is $(0.0, 0.0)$.
- d) The initial positions x_i^t of other objects are randomly sampled from all angles, with a distance in $[10.0, 100.0]m$.
- e) The velocity of the center star is 0 .
- f) The velocities of other stars are perpendicular to the gravitational force between the center star and itself. The scale is precisely determined by physics laws to ensure the initial state is an orbit system.

For each video, after we get the initial states, we continue to rollout the next frames according the physics engine described above. We perform rejection sampling of the frames to ensure that all pairwise distances of stars in a frame are at least $30m$. We guarantee that there are 10,000 data points in the training set.

The validation set has the same sampling and simulation parameters as the training set. We have 2,500 data points in the validation set.

For test set, we consider two datasets, where we respectively have OOD distances and masses. We have 5,000 data points for each dataset.

- a) We sample the distance OOD test set to ensure all pairwise distances of stars in a frame are from $[1..20]m$, but have in-distribution masses.
- b) We sample the mass OOD test set as follows
 - i) The mass of the center star is $200kg$, i.e., twice of that in the training set.
 - ii) The masses of other stars are sampled from $[0.04, 18.0]kg$, compared to $[0.02, 9.0]kg$ in the training set.

iii) The distances are in-distribution, i.e., same sampling process as training set.

Model and hyperparameter settings. We consider the following one-iteration Graph Neural Network (GNN) architecture, a.k.a. Interaction Networks. Given a collection of stars $S = \{X_i\}_{i=1}^n$, our GNN runs on a complete graph with nodes being the stars X_i . GNN learns the star (node) representations by aggregating and transforming the interactions (forces) of all other node vectors

$$o_u = \text{MLP}^{(2)} \left(\sum_{v \in S \setminus \{u\}} \text{MLP}^{(1)}(h_v, h_u, w_{(u,v)}) \right). \quad (7)$$

Here, h_v is the input feature of node v , including mass, position and velocity

$$h_v = (m_v, \mathbf{x}_v, \mathbf{v}_v)$$

$w_{(u,v)}$ is the input edge feature of edge (u, v) . The loss is computed and backpropagated via the MSE loss of

$$\|[o_1, \dots, o_n] - [ans_1, \dots, ans_n]\|_2,$$

where o_i denotes the output of GNN for node i , and ans_i denotes the true label for node i in the next frame.

We search the following hyper-parameters for GNNs

- a) Number of GNN iterations is set to 1.
- b) Width of all MLPs are set to 128.

- c) The number of layers for $\text{MLP}^{(1)}$ is set to 4. The number of layers for $\text{MLP}^{(2)}$ is set to 2.
- d) We consider *two representations* of edge/relations $w_{(i,j)}$.
 - i) The first one is simply 0.
 - ii) The better representation, which makes the underlying target function more linear, is

$$w_{(i,j)} = \frac{m_j}{\|\mathbf{x}_i^t - \mathbf{x}_j^t\|_2^3} \cdot (\mathbf{x}_j^t - \mathbf{x}_i^t)$$

We train the GNN with the mean squared error (MSE) loss, and Adam optimizer. We search the following hyper-parameters for training

- a) Initial learning rate is set to 0.005. learning rate decays 0.5 for every 50 epochs
- b) Batch size is set to 32.
- c) Weight decay is set to $1e - 5$.
- d) Number of epochs is set to 2,000.

7.2 Visualization and Additional Experimental Results

7.2.1 Visualization Results

In this section, we show additional visualization results of the MLP’s learned function out of training distribution (in **black color**) v.s. the underlying true function (in **grey color**). We color the predictions in training distribution in **blue color**.

In general, MLP’s learned functions agree with the underlying true functions in training range (blue). This is explained by in-distribution generalization arguments. When out of distribution, the MLP’s learned functions become linear along directions from the origin. We explain this OOD directional linearity behavior in Theorem 3.3.2.

Finally, we show additional experimental results for graph-based reasoning tasks.



Figure 7.1: **(Quadratic function)**. Both panels show the learned v.s. true $y = x_1^2 + x_2^2$. In each figure, we color OOD predictions by MLPs in black, underlying function in grey, and in-distribution predictions in blue. The support of training distribution is a square (cube) for the top panel, and is a circle (sphere) for the bottom panel.

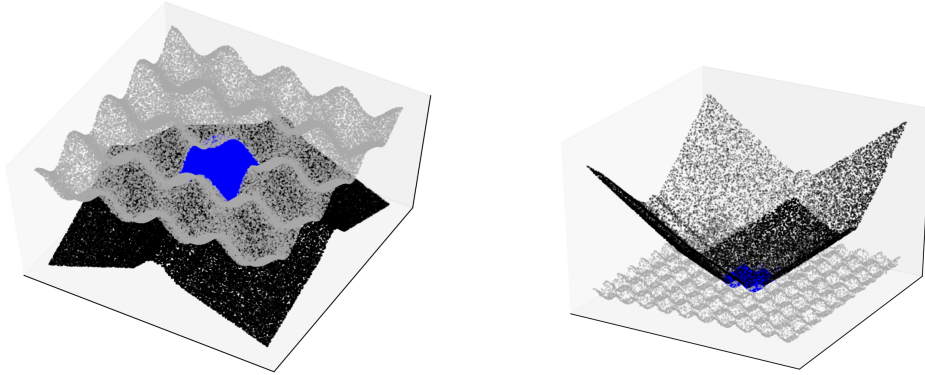


Figure 7.2: **(Cos function)**. Both panels show the learned v.s. true $y = \cos(2\pi \cdot x_1) + \cos(2\pi \cdot x_2)$. In each figure, we color OOD predictions by MLPs in black, underlying function in grey, and in-distribution predictions in blue. The support of training distribution is a square (cube) for both top and bottom panels, but with different ranges.

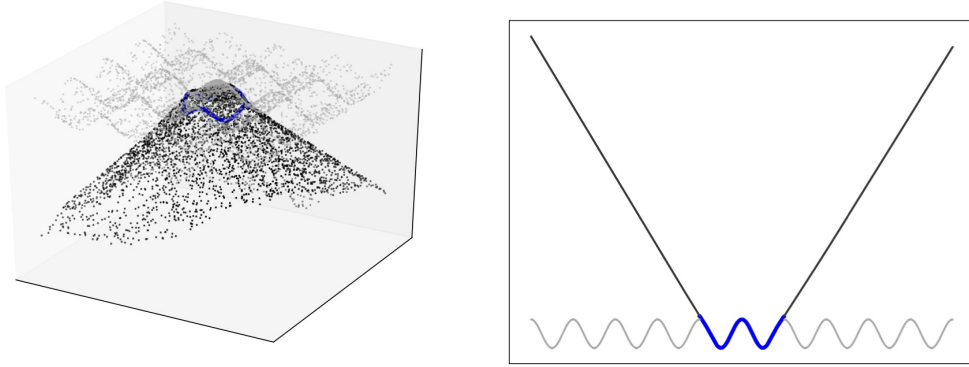


Figure 7.3: **(Cos function)**. Top panel shows the learned v.s. true $y = \cos(2\pi \cdot x_1) + \cos(2\pi \cdot x_2)$ where the support of training distribution is a circle (sphere). Bottom panel shows results for cosine in 1D, i.e. $y = \cos(2\pi \cdot x)$. In each figure, we color OOD predictions by MLPs in black, underlying function in grey, and in-distribution predictions in blue.

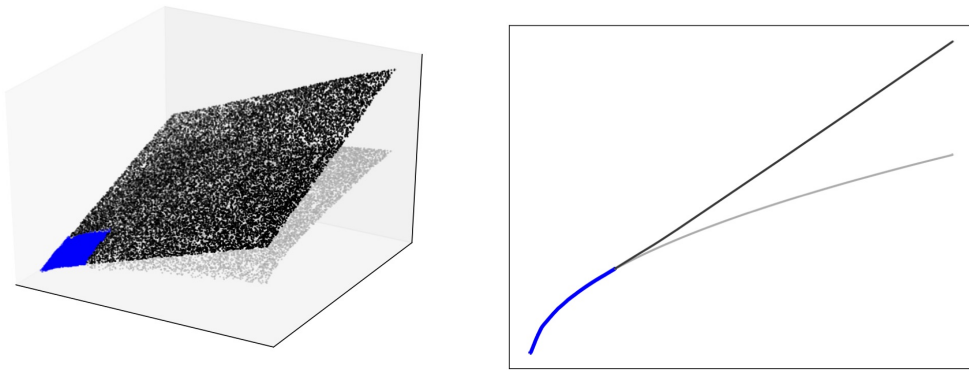


Figure 7.4: **(Sqrt function)**. Top panel shows the learned v.s. true $y = \sqrt{x_1} + \sqrt{x_2}$ where the support of training distribution is a square (cube). Bottom panel shows the results for the square root function in 1D, i.e. $y = \sqrt{x}$. In each figure, we color OOD predictions by MLPs in black, underlying function in grey, and in-distribution predictions in blue.

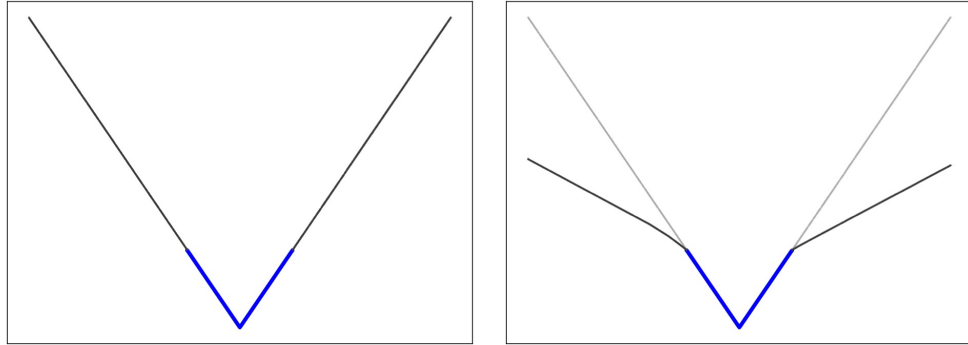


Figure 7.5: **(L1 function)**. Both panels show the learned v.s. true $y = |x|$. In the top panel, the MLP successfully learns to extrapolate the absolute function. In the bottom panel, an MLP with different hyper-parameters fails to extrapolate. In each figure, we color OOD predictions by MLPs in black, underlying function in grey, and in-distribution predictions in blue.



Figure 7.6: **(L1 function)**. Both panels show the learned v.s. true $y = |x_1| + |x_2|$. In the top panel, the MLP successfully learns to extrapolate the l1 norm function. In the bottom panel, an MLP with different hyper-parameters fails to extrapolate. In each figure, we color OOD predictions by MLPs in black, underlying function in grey, and in-distribution predictions in blue.

7.2.2 Extra Experimental Results

In this section, we show additional experimental results.

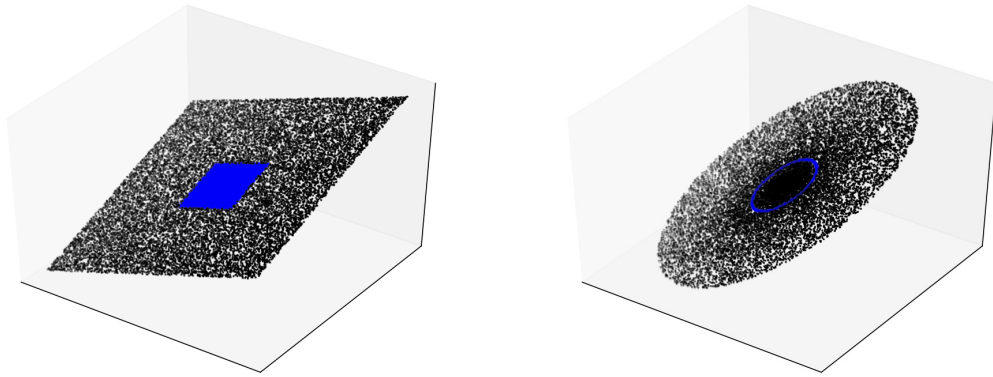


Figure 7.7: **(Linear function).** Both panels show the learned v.s. true $y = x_1 + x_2$, with the support of training distributions being square (cube) for top panel, and circle (sphere) for bottom panel. MLPs successfully extrapolate the linear function with both training distributions. This is explained by Theorem 3.3.4: both sphere and cube intersect all directions. In each figure, we color OOD predictions by MLPs in black, underlying function in grey, and in-distribution predictions in blue.



Figure 7.8: **Density plot of the test errors in MAPE.** The underlying functions are linear, but we train MLPs on different distributions, whose support potentially miss some directions. The training support for “all” are hyper-cubes that intersect all directions. In “fix1”, we set the first dimension of training data to a fixed number. In “posX”, we restrict the first X dimensions of training data to be positive. We can see that MLPs trained on “all” extrapolate the underlying linear functions, but MLPs trained on datasets with missing directions, i.e., “fix1” and “posX”, often cannot extrapolate well.

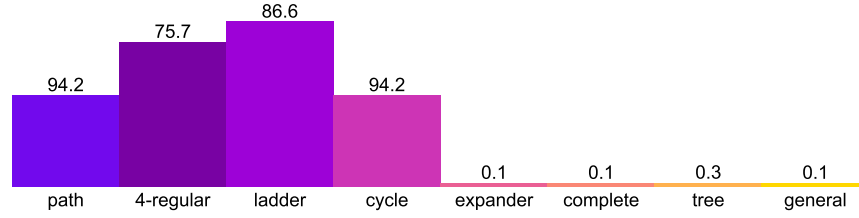


Figure 7.9: **Maximum degree: continuous and “spurious” node features.** Here, each node has a node feature in $\mathbb{R}^3$ that shall not contribute to the answer of maximum degree. GNNs with graph-level max-pooling extrapolate to graphs with OOD node features and graph structure, graph sizes, if trained on graphs that satisfy the condition in Theorem 3.4.4.

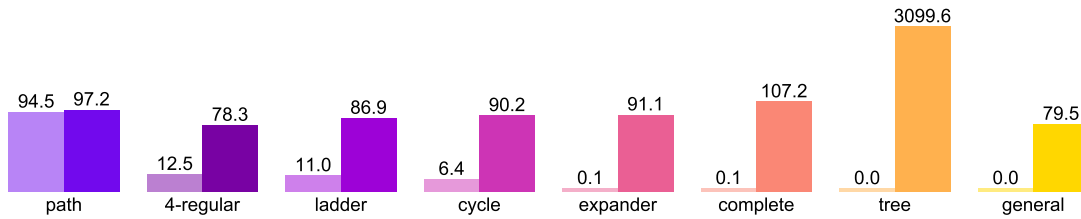


Figure 7.10: **Maximum degree: max-pooling v.s. sum-pooling.** In each sub-figure, left column shows test errors for GNNs with graph-level max-pooling; right column shows test errors for GNNs with graph-level sum-pooling. x-axis shows the graph structure covered in the training set. GNNs with sum-pooling fail to extrapolate, validating Corollary 3.4.3. GNNs with max-pooling encode appropriate non-linear operations, and thus extrapolates under appropriate training sets (Theorem 3.4.4).

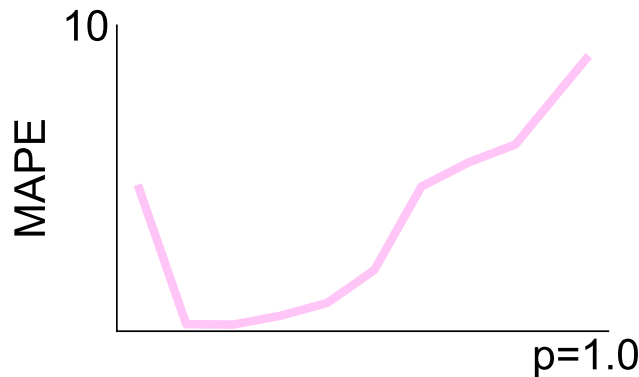


Figure 7.11: **Shortest path: random graphs.** We train GNNs with neighbor and graph-level min on random graphs with probability p of an edge between any two vertices. x-axis denotes the p for the training set, and y-axis denotes the test/extrapolation error on unseen graphs. The test errors follow a U-shape: errors are high if the training graphs are very sparse (small p) or dense (large p). The same pattern is obtained if we train on specific graph structure.

7.3 Supplementary Theoretical Background

In this section, we introduce the theoretical background on neural tangent kernel (NTK), which draws an equivalence between the training dynamics of infinitely wide (or ultra-wide) neural networks and that of kernel regression with respect to the neural tangent kernel.

Consider a general neural network $f(\boldsymbol{\theta}, \mathbf{x}) : \mathcal{X} \rightarrow \mathbb{R}$ where $\boldsymbol{\theta} \in \mathbb{R}^m$ is the parameters in the network and $\mathbf{x} \in \mathcal{X}$ is the input. Suppose we train the neural network by minimizing the squared loss over training data, $\ell(\boldsymbol{\theta}) = \frac{1}{2} \sum_{i=1}^n (f(\boldsymbol{\theta}, \mathbf{x}_i) - y_i)^2$, by gradient descent with infinitesimally small learning rate, i.e., $\frac{d\boldsymbol{\theta}(t)}{dt} = -\nabla \ell(\boldsymbol{\theta}(t))$. Let $\mathbf{u}(t) = (f(\boldsymbol{\theta}(t), \mathbf{x}_i))_{i=1}^n$ be the network outputs. $\mathbf{u}(t)$ follows the dynamics

$$\frac{d\mathbf{u}(t)}{dt} = -\mathbf{H}(t)(\mathbf{u}(t) - \mathbf{y}), \quad (8)$$

where $\mathbf{H}(t)$ is an $n \times n$ matrix whose (i, j) -th entry is

$$\mathbf{H}(t)_{ij} = \left\langle \frac{\partial f(\boldsymbol{\theta}(t), \mathbf{x}_i)}{\partial \boldsymbol{\theta}}, \frac{\partial f(\boldsymbol{\theta}(t), \mathbf{x}_j)}{\partial \boldsymbol{\theta}} \right\rangle. \quad (9)$$

A line of works shows that for sufficiently wide networks, $\mathbf{H}(t)$ stays almost constant during training, i.e., $\mathbf{H}(t) = \mathbf{H}(0)$ in the limit [Allen-Zhu et al. \(2019a\)](#); [Arora et al. \(2019a,b\)](#); [Du et al. \(2019a,b\)](#); [Jacot et al. \(2018\)](#); [Li and Liang \(2018\)](#). Suppose network parameters are randomly initialized with certain scaling, as network width goes to infinity, $\mathbf{H}(0)$ converges to a fixed

matrix, the neural tangent kernel (NTK) [Jacot et al. \(2018\)](#):

$$\text{NTK}(\mathbf{x}, \mathbf{x}') = \mathbb{E}_{\boldsymbol{\theta} \sim \mathcal{W}} \left\langle \frac{\partial f(\boldsymbol{\theta}(t), \mathbf{x})}{\partial \boldsymbol{\theta}}, \frac{\partial f(\boldsymbol{\theta}(t), \mathbf{x}')}{\partial \boldsymbol{\theta}} \right\rangle, \quad (10)$$

where $\mathcal{W}$ is Gaussian.

Therefore, the learning dynamics of sufficiently wide neural networks in this regime is equivalent to that of kernel gradient descent with respect to the NTK. This implies the function learned by a neural network at convergence on any specific training set, denoted by $f_{\text{NTK}}(\mathbf{x})$, can be precisely characterized, and is equivalent to the following kernel regression solution

$$f_{\text{NTK}}(\mathbf{x}) = (\text{NTK}(\mathbf{x}, \mathbf{x}_1), \dots, \text{NTK}(\mathbf{x}, \mathbf{x}_n)) \cdot \text{NTK}_{\text{train}}^{-1} \mathbf{Y}, \quad (11)$$

where $\text{NTK}_{\text{train}}$ is the $n \times n$ kernel for training data, $\text{NTK}(\mathbf{x}, \mathbf{x}_i)$ is the kernel value between test data $\mathbf{x}$ and training data $\mathbf{x}_i$, and $\mathbf{Y}$ is the training labels.

We can in fact exactly calculate the neural tangent kernel matrix for certain architectures and activation functions. The exact formula of NTK with ReLU activation has been derived for feedforward neural networks [Jacot et al. \(2018\)](#), convolutional neural networks [Arora et al. \(2019b\)](#), and Graph Neural Networks [Du et al. \(2019c\)](#).

Our theory builds upon this equivalence of network learning and kernel regression to more precisely characterize the function learned by a sufficiently-wide neural network given any specific training set. In particular, the difference between the learned function and true function over the domain of $\mathcal{X}$ determines the extrapolation error.

However, in general it is non-trivial to compute or analyze the *functional form* of what a

neural network learns using Eqn. 11, because the kernel regression solution using neural tangent kernel only gives point-wise evaluation. Thus, we instead analyze the function learned by a network in the NTK’s induced *feature space*, because representations in the feature space would give a functional form.

Lemma 7.3.1 makes this connection more precise: the solution to the kernel regression using neural tangent kernel, which also equals over-parameterized network learning, is equivalent to a min-norm solution among functions in the NTK’s induced feature space that fits all training data. Here the min-norm refers to the RKHS norm.

Lemma 7.3.1. *Let $\phi(\mathbf{x})$ be a feature map induced by a neural tangent kernel, for any $\mathbf{x} \in \mathbb{R}^d$. The solution to kernel regression Eqn. 11 is equivalent to $f_{NTK}(\mathbf{x}) = \phi(\mathbf{x})^\top \boldsymbol{\beta}_{NTK}$, where $\boldsymbol{\beta}_{NTK}$ is*

$$\begin{aligned} & \min_{\boldsymbol{\beta}} \|\boldsymbol{\beta}\|_2 \\ \text{s.t. } & \phi(\mathbf{x}_i)^\top \boldsymbol{\beta} = y_i, \text{ for } i = 1, \dots, n. \end{aligned}$$

We prove Lemma 7.3.1 in Appendix 7.4.8. To analyze the learned functions as the min-norm solution in feature space, we also need the explicit formula of an induced feature map of the corresponding neural tangent kernel. The following lemma gives a NTK feature space for two-layer MLPs with ReLU activation. It follows easily from the kernel formula described in Arora et al. (2019b); Jacot et al. (2018); Bietti and Mairal (2019).

Lemma 7.3.2. *An infinite-dimensional feature map $\phi(\mathbf{x})$ induced by the neural tangent kernel of*

a two-layer multi-layer perceptron with ReLU activation function is

$$\phi(\mathbf{x}) = c \left(\mathbf{x} \cdot \mathbb{I} \left(\mathbf{w}^{(k)\top} \mathbf{x} \geq 0 \right), \mathbf{w}^{(k)\top} \mathbf{x} \cdot \mathbb{I} \left(\mathbf{w}^{(k)\top} \mathbf{x} \geq 0 \right), \dots \right), \quad (12)$$

where $\mathbf{w}^{(k)} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, with k going to infinity. c is a constant, and $\mathbb{I}$ is the indicator function.

We prove Lemma 7.3.2 in Appendix 7.4.9. The feature maps for other architectures, e.g., Graph Neural Networks (GNNs) can be derived similarly. We analyze the Graph Neural Tangent Kernel (GNTK) for a simple GNN architecture in Theorem 3.4.4.

We then use Lemma 7.3.1 and 7.3.2 to characterize the properties of functions learned by an over-parameterized neural network. We precisely characterize the neural networks' learned functions in the NTK regime via solving the constrained optimization problem corresponding to the min-norm function in NTK feature space with the constraint of fitting the training data.

However, there still remains many technical challenges. For example, provable extrapolation (exact or asymptotic) is often *not* achieved with most training data distribution. Understanding the desirable condition requires significant insights into the geometry properties of training data distribution, and how they interact with the solution learned by neural networks. Our insights and refined analysis shows in $\mathbb{R}^d$ space, we need to consider the directions of training data. In graphs, we need to consider, in addition, the graph structure of training data. We refer readers to detailed proofs for the intuition of data conditions. Moreover, since NTK corresponds to infinitely wide neural networks, the feature space is of infinite dimension. The analysis of infinite dimensional spaces poses non-trivial technical challenges too.

Since different theorems have their respective challenges and insights/techniques, we refer the interested readers to the respective proofs for details. In Lemma 3.3.3 (proof in Appendix 7.4),

Theorem 3.3.4 (proof in Appendix 7.4.3), and Theorem 3.3.2 (proof in Appendix 7.4) we analyze over-parameterized MLPs. The proof of Corollary 3.4.3 is in Appendix 7.4.6. In Theorem 3.4.4 we analyze Graph Neural Networks (proof in Appendix 7.4.7).

7.4 Proofs

Proof of Theorem 3.3.2 To show neural network outputs $f(\mathbf{x})$ converge to a linear function along all directions $\mathbf{v}$, we will analyze the function learned by a neural network on the training set $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$, by studying the functional representation in the network's neural tangent kernel RKHS space.

Recall from Section 7.3 that in the NTK regime, i.e., networks are infinitely wide, randomly initialized, and trained by gradient descent with infinitesimally small learning rate, the learning dynamics of the neural network is equivalent to that of a kernel regression with respect to its neural tangent kernel.

For any $\mathbf{x} \in \mathbb{R}^d$, the network output is given by

$$f(\mathbf{x}) = (\langle \phi(\mathbf{x}), \phi(\mathbf{x}_1) \rangle, \dots, \langle \phi(\mathbf{x}), \phi(\mathbf{x}_n) \rangle) \cdot \text{NTK}_{\text{train}}^{-1} \mathbf{Y},$$

where $\text{NTK}_{\text{train}}$ is the $n \times n$ kernel for training data, $\langle \phi(\mathbf{x}), \phi(\mathbf{x}_i) \rangle$ is the kernel value between test data $\mathbf{x}$ and training data $\mathbf{x}_i$, and $\mathbf{Y}$ is training labels. By Lemma 7.3.1, the kernel regression solution is also equivalent to the min-norm solution in the NTK RKHS space that fits all training data

$$f(\mathbf{x}) = \phi(\mathbf{x})^\top \boldsymbol{\beta}_{\text{NTK}}, \tag{13}$$

where the representation coefficient β_{NTK} is

$$\begin{aligned} \min_{\beta} \quad & \|\beta\|_2 \\ \text{s.t.} \quad & \phi(\mathbf{x}_i)^\top \beta = y_i, \quad \text{for } i = 1, \dots, n. \end{aligned}$$

The feature map $\phi(\mathbf{x})$ for a two-layer MLP with ReLU activation is given by Lemma 7.3.2

$$\phi(\mathbf{x}) = c' \left(\mathbf{x} \cdot \mathbb{I}(\mathbf{w}^{(k)\top} \mathbf{x} \geq 0), \mathbf{w}^{(k)\top} \mathbf{x} \cdot \mathbb{I}(\mathbf{w}^{(k)\top} \mathbf{x} \geq 0), \dots \right), \quad (14)$$

where $\mathbf{w}^{(k)} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, with k going to infinity. c' is a constant, and $\mathbb{I}$ is the indicator function.

Without loss of generality, we assume the bias term to be 1. For simplicity of notations, we denote each data $\mathbf{x}$ plus bias term by, i.e., $\hat{\mathbf{x}} = [\mathbf{x}|1]$ Bietti and Mairal (2019), and assume constant term is 1.

Given any direction $\mathbf{v}$ on the unit sphere, the network outputs for out-of-distribution data $\mathbf{x}_0 = t\mathbf{v}$ and $\mathbf{x} = \mathbf{x}_0 + h\mathbf{v} = (1 + \lambda)\mathbf{x}_0$, where we introduce the notation of $\mathbf{x}$ and λ for convenience, are given by Eqn. 13 and Eqn. 14

$$\begin{aligned} f(\hat{\mathbf{x}}_0) &= \beta_{\text{NTK}}^\top \left(\hat{\mathbf{x}}_0 \cdot \mathbb{I}(\mathbf{w}^{(k)\top} \hat{\mathbf{x}}_0 \geq 0), \mathbf{w}^{(k)\top} \hat{\mathbf{x}}_0 \cdot \mathbb{I}(\mathbf{w}^{(k)\top} \hat{\mathbf{x}}_0 \geq 0), \dots \right), \\ f(\hat{\mathbf{x}}) &= \beta_{\text{NTK}}^\top \left(\hat{\mathbf{x}} \cdot \mathbb{I}(\mathbf{w}^{(k)\top} \hat{\mathbf{x}} \geq 0), \mathbf{w}^{(k)\top} \hat{\mathbf{x}} \cdot \mathbb{I}(\mathbf{w}^{(k)\top} \hat{\mathbf{x}} \geq 0), \dots \right), \end{aligned}$$

where we have $\hat{\mathbf{x}}_0 = [\mathbf{x}_0|1]$ and $\hat{\mathbf{x}} = [(1 + \lambda)\mathbf{x}_0|1]$. It follows that

$$f(\hat{\mathbf{x}}) - f(\hat{\mathbf{x}}_0) = \beta_{\text{NTK}}^\top \left(\hat{\mathbf{x}} \cdot \mathbb{I}(\mathbf{w}^{(k)\top} \hat{\mathbf{x}} \geq 0) - \hat{\mathbf{x}}_0 \cdot \mathbb{I}(\mathbf{w}^{(k)\top} \hat{\mathbf{x}}_0 \geq 0), \dots \right), \quad (15)$$

$$\mathbf{w}^{(k)\top} \hat{\mathbf{x}} \cdot \mathbb{I}(\mathbf{w}^{(k)\top} \hat{\mathbf{x}} \geq 0) - \mathbf{w}^{(k)\top} \hat{\mathbf{x}}_0 \cdot \mathbb{I}(\mathbf{w}^{(k)\top} \hat{\mathbf{x}}_0 \geq 0), \dots \quad (16)$$

By re-arranging the terms, we get the following equivalent form of the entries:

$$\hat{\mathbf{x}} \cdot \mathbb{I}(\mathbf{w}^\top \hat{\mathbf{x}} \geq 0) - \hat{\mathbf{x}}_0 \cdot \mathbb{I}(\mathbf{w}^\top \hat{\mathbf{x}}_0 \geq 0) \quad (17)$$

$$= \hat{\mathbf{x}} \cdot (\mathbb{I}(\mathbf{w}^\top \hat{\mathbf{x}} \geq 0) - \mathbb{I}(\mathbf{w}^\top \hat{\mathbf{x}}_0 \geq 0) + \mathbb{I}(\mathbf{w}^\top \hat{\mathbf{x}}_0 \geq 0)) - \hat{\mathbf{x}}_0 \cdot \mathbb{I}(\mathbf{w}^\top \hat{\mathbf{x}}_0 \geq 0) \quad (18)$$

$$= \hat{\mathbf{x}} \cdot (\mathbb{I}(\mathbf{w}^\top \hat{\mathbf{x}} \geq 0) - \mathbb{I}(\mathbf{w}^\top \hat{\mathbf{x}}_0 \geq 0)) + (\hat{\mathbf{x}} - \hat{\mathbf{x}}_0) \cdot \mathbb{I}(\mathbf{w}^\top \hat{\mathbf{x}}_0 \geq 0) \quad (19)$$

$$= [x|1] \cdot (\mathbb{I}(\mathbf{w}^\top \hat{\mathbf{x}} \geq 0) - \mathbb{I}(\mathbf{w}^\top \hat{\mathbf{x}}_0 \geq 0)) + [h\mathbf{v}|0] \cdot \mathbb{I}(\mathbf{w}^\top \hat{\mathbf{x}}_0 \geq 0) \quad (20)$$

Similarly, we have

$$\mathbf{w}^\top \hat{\mathbf{x}} \cdot \mathbb{I}(\mathbf{w}^\top \hat{\mathbf{x}} \geq 0) - \mathbf{w}^\top \hat{\mathbf{x}}_0 \cdot \mathbb{I}(\mathbf{w}^\top \hat{\mathbf{x}}_0 \geq 0) \quad (21)$$

$$= \mathbf{w}^\top \hat{\mathbf{x}} \cdot (\mathbb{I}(\mathbf{w}^\top \hat{\mathbf{x}} \geq 0) - \mathbb{I}(\mathbf{w}^\top \hat{\mathbf{x}}_0 \geq 0) + \mathbb{I}(\mathbf{w}^\top \hat{\mathbf{x}}_0 \geq 0)) - \mathbf{w}^\top \hat{\mathbf{x}}_0 \cdot \mathbb{I}(\mathbf{w}^\top \hat{\mathbf{x}}_0 \geq 0) \quad (22)$$

$$= \mathbf{w}^\top \hat{\mathbf{x}} \cdot (\mathbb{I}(\mathbf{w}^\top \hat{\mathbf{x}} \geq 0) - \mathbb{I}(\mathbf{w}^\top \hat{\mathbf{x}}_0 \geq 0)) + \mathbf{w}^\top (\hat{\mathbf{x}} - \hat{\mathbf{x}}_0) \cdot \mathbb{I}(\mathbf{w}^\top \hat{\mathbf{x}}_0 \geq 0) \quad (23)$$

$$= \mathbf{w}^\top [x|1] \cdot (\mathbb{I}(\mathbf{w}^\top \hat{\mathbf{x}} \geq 0) - \mathbb{I}(\mathbf{w}^\top \hat{\mathbf{x}}_0 \geq 0)) + \mathbf{w}^\top [h\mathbf{v}|0] \cdot \mathbb{I}(\mathbf{w}^\top \hat{\mathbf{x}}_0 \geq 0) \quad (24)$$

Again, let us denote the part of β_{NTK} corresponding to each $\mathbf{w}$ by $\beta_{\mathbf{w}}$. Moreover, let us denote the part corresponding to Eqn. 20 by $\beta_{\mathbf{w}}^1$ and the part corresponding to Eqn. 24 by $\beta_{\mathbf{w}}^2$. Then we have

$$\frac{f(\hat{\mathbf{x}}) - f(\hat{\mathbf{x}}_0)}{h} \quad (25)$$

$$= \int \beta_{\mathbf{w}}^{1\top} [x/h|1/h] \cdot (\mathbb{I}(\mathbf{w}^\top \hat{\mathbf{x}} \geq 0) - \mathbb{I}(\mathbf{w}^\top \hat{\mathbf{x}}_0 \geq 0)) d\mathbb{P}(\mathbf{w}) \quad (26)$$

$$+ \int \beta_w^{1^\top} [\mathbf{v}|0] \cdot \mathbb{I}(\mathbf{w}^\top \hat{\mathbf{x}}_0 \geq 0) d\mathbb{P}(\mathbf{w}) \quad (27)$$

$$+ \int \beta_w^2 \cdot \mathbf{w}^\top [\mathbf{x}/h|1/h] \cdot (\mathbb{I}(\mathbf{w}^\top \hat{\mathbf{x}} \geq 0) - \mathbb{I}(\mathbf{w}^\top \hat{\mathbf{x}}_0 \geq 0)) d\mathbb{P}(\mathbf{w}) \quad (28)$$

$$+ \int \beta_w^2 \cdot \mathbf{w}^\top [\mathbf{v}|0] \cdot \mathbb{I}(\mathbf{w}^\top \hat{\mathbf{x}}_0 \geq 0) d\mathbb{P}(\mathbf{w}) \quad (29)$$

Note that all β_w are finite constants that depend on the training data. Next, we show that as $t \rightarrow \infty$, each of the terms above converges in $O(1/\epsilon)$ to some constant coefficient β_v that depend on the training data and the direction $\mathbf{v}$. Let us first consider Eqn. 27. We have

$$\int \mathbb{I}(\mathbf{w}^\top \hat{\mathbf{x}}_0 \geq 0) d\mathbb{P}(\mathbf{w}) = \int \mathbb{I}(\mathbf{w}^\top [\mathbf{x}_0|1] \geq 0) d\mathbb{P}(\mathbf{w}) \quad (30)$$

$$= \int \mathbb{I}(\mathbf{w}^\top [\mathbf{x}_0/t|1/t] \geq 0) d\mathbb{P}(\mathbf{w}) \quad (31)$$

$$\rightarrow \int \mathbb{I}(\mathbf{w}^\top [\mathbf{v}|0] \geq 0) d\mathbb{P}(\mathbf{w}) \quad \text{as } t \rightarrow \infty \quad (32)$$

Because β_w^1 are finite constants, it follows that

$$\int \beta_w^{1^\top} [\mathbf{v}|0] \cdot \mathbb{I}(\mathbf{w}^\top \hat{\mathbf{x}}_0 \geq 0) d\mathbb{P}(\mathbf{w}) \rightarrow \int \beta_w^{1^\top} [\mathbf{v}|0] \cdot \mathbb{I}(\mathbf{w}^\top [\mathbf{v}|0] \geq 0) d\mathbb{P}(\mathbf{w}), \quad (33)$$

where the right hand side is a constant that depends on training data and direction $\mathbf{v}$. Next, we show the convergence rate for Eqn. 33. Given error $\epsilon > 0$, because $\beta_w^{1^\top} [\mathbf{v}|0]$ are finite constants, we need to bound the following by $C \cdot \epsilon$ for some constant C ,

$$| \int \mathbb{I}(\mathbf{w}^\top \hat{\mathbf{x}}_0 \geq 0) - \mathbb{I}(\mathbf{w}^\top [\mathbf{v}|0] \geq 0) d\mathbb{P}(\mathbf{w}) | \quad (34)$$

$$= | \int \mathbb{I}(\mathbf{w}^\top [\mathbf{x}_0|1] \geq 0) - \mathbb{I}(\mathbf{w}^\top [\mathbf{x}_0|0] \geq 0) d\mathbb{P}(\mathbf{w}) | \quad (35)$$

Observe that the two terms in Eqn. 35 represent the volume of half-(balls) that are orthogonal to vectors $[\mathbf{x}_0|1]$ and $[\mathbf{x}_0|0]$. Hence, Eqn. 35 is the volume of the non-overlapping part of the two (half)balls, which is created by rotating an angle θ along the last coordinate. By symmetry, Eqn. 35 is linear in θ . Moreover, the angle $\theta = \arctan(C/t)$ for some constant C . Hence, it follows that

$$|\int \mathbb{I}(\mathbf{w}^\top [\mathbf{x}_0|1] \geq 0) - \mathbb{I}(\mathbf{w}^\top [\mathbf{x}_0|0] \geq 0) d\mathbb{P}(\mathbf{w})| = C_1 \cdot \arctan(C_2/t) \quad (36)$$

$$\leq C_1 \cdot C_2/t \quad (37)$$

$$= O(1/t) \quad (38)$$

In the last inequality, we used the fact that $\arctan x < x$ for $x > 0$. Hence, $O(1/t) < \epsilon$ implies $t = O(1/\epsilon)$ as desired. Next, we consider Eqn. 26.

$$\int \beta_w^{1^\top} [\mathbf{x}/h|1/h] \cdot (\mathbb{I}(\mathbf{w}^\top \hat{\mathbf{x}} \geq 0) - \mathbb{I}(\mathbf{w}^\top \hat{\mathbf{x}}_0 \geq 0)) d\mathbb{P}(\mathbf{w}) \quad (39)$$

Let us first analyze the convergence of the following:

$$|\int \mathbb{I}(\mathbf{w}^\top \hat{\mathbf{x}} \geq 0) - \mathbb{I}(\mathbf{w}^\top \hat{\mathbf{x}}_0 \geq 0) d\mathbb{P}(\mathbf{w})| \quad (40)$$

$$= |\int \mathbb{I}(\mathbf{w}^\top [(1+\lambda)\mathbf{x}_0|1] \geq 0) - \mathbb{I}(\mathbf{w}^\top [\mathbf{x}_0|1] \geq 0) d\mathbb{P}(\mathbf{w}) d\mathbb{P}(\mathbf{w})| \quad (41)$$

$$= |\int \mathbb{I}\left(\mathbf{w}^\top [\mathbf{x}_0|\frac{1}{1+\lambda}] \geq 0\right) - \mathbb{I}(\mathbf{w}^\top [\mathbf{x}_0|1] \geq 0) d\mathbb{P}(\mathbf{w}) d\mathbb{P}(\mathbf{w})| \rightarrow 0 \quad (42)$$

The convergence to 0 follows from Eqn. 36. Now we consider the convergence rate. The angle θ

is at most $1 - \frac{1}{1+\lambda}$ times of that in Eqn. 36. Hence, the rate is as follows

$$\left(1 - \frac{1}{1+\lambda}\right) \cdot O\left(\frac{1}{t}\right) = \frac{\lambda}{1+\lambda} \cdot O\left(\frac{1}{t}\right) = \frac{h/t}{1+h/t} \cdot O\left(\frac{1}{t}\right) = O\left(\frac{h}{(h+t)t}\right) \quad (43)$$

Now we get back to Eqn. 26, which simplifies as the following.

$$\int \beta_w^{1^\top} \left[\mathbf{v} + \frac{t\mathbf{v}}{h} \middle| \frac{1}{h} \right] \cdot (\mathbb{I}(\mathbf{w}^\top \hat{\mathbf{x}} \geq 0) - \mathbb{I}(\mathbf{w}^\top \hat{\mathbf{x}}_0 \geq 0)) d\mathbb{P}(\mathbf{w}) \quad (44)$$

We compare the rate of growth of left hand side and the rate of decrease of right hand side (indicators).

$$\frac{t}{h} \cdot \frac{h}{(h+t)t} = \frac{1}{h+t} \rightarrow 0 \quad \text{as } t \rightarrow \infty \quad (45)$$

$$\frac{1}{h} \cdot \frac{h}{(h+t)t} = \frac{1}{(h+t)t} \rightarrow 0 \quad \text{as } t \rightarrow \infty \quad (46)$$

Hence, the indicators decrease faster, and it follows that Eqn. 26 converges to 0 with rate $O(\frac{1}{\epsilon})$.

Moreover, we can bound $\mathbf{w}$ with standard concentration techniques. Then the proofs for Eqn. 28 and Eqn. 29 follow similarly. This completes the proof.

Proof of Lemma 3.3.3 Overview of proof. To prove exact extrapolation given the conditions on training data, we analyze the function learned by the neural network in a functional form. The network's learned function can be precisely characterized by a solution in the network's neural tangent kernel feature space which has a minimum RKHS norm among functions that can fit all training data, i.e., it corresponds to the optimum of a constrained optimization problem. We show that the global optimum of this constrained optimization problem, given the conditions on

training data, is precisely the same function as the underlying true function.

Setup and preparation. Let $\mathbf{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_n\}$ and $\mathbf{Y} = \{y_1, \dots, y_n\}$ denote the training set input features and their labels. Let $\beta_g \in \mathbb{R}^d$ denote the true parameters/weights for the underlying linear function g , i.e.,

$$g(\mathbf{x}) = \beta_g^\top \mathbf{x} \quad \text{for all } \mathbf{x} \in \mathbb{R}^d$$

Recall from Section 7.3 that in the NTK regime, where networks are infinitely wide, randomly initialized, and trained by gradient descent with infinitesimally small learning rate, the learning dynamics of a neural network is equivalent to that of a kernel regression with respect to its neural tangent kernel. Moreover, Lemma 7.3.1 tells us that this kernel regression solution can be expressed in the functional form in the neural tangent kernel's feature space. That is, the function learned by the neural network (in the ntk regime) can be precisely characterized as

$$f(\mathbf{x}) = \phi(\mathbf{x})^\top \beta_{\text{NTK}},$$

where the representation coefficient β_{NTK} is

$$\min_{\beta} \|\beta\|_2 \tag{47}$$

$$\text{s.t. } \phi(\mathbf{x}_i)^\top \beta = y_i, \quad \text{for } i = 1, \dots, n. \tag{48}$$

An infinite-dimensional feature map $\phi(\mathbf{x})$ for a two-layer ReLU network is described in Lemma 7.3.2

$$\phi(\mathbf{x}) = c' \left(\mathbf{x} \cdot \mathbb{I}(\mathbf{w}^{(k)\top} \mathbf{x} \geq 0), \mathbf{w}^{(k)\top} \mathbf{x} \cdot \mathbb{I}(\mathbf{w}^{(k)\top} \mathbf{x} \geq 0), \dots \right),$$

where $\mathbf{w}^{(k)} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, with k going to infinity. c' is a constant, and $\mathbb{I}$ is the indicator function. That is, there are infinitely many directions $\mathbf{w}$ with Gaussian density, and each direction comes with two features. Without loss of generality, we can assume the scaling constant to be 1.

7.4.1 Constrained optimization in NTK feature space.

The representation or weight of the neural network's learned function in the neural tangent kernel feature space, β_{NTK} , consists of weight vectors for each $\mathbf{x} \cdot \mathbb{I}(\mathbf{w}^{(k)\top} \mathbf{x} \geq 0) \in \mathbb{R}^d$ and $\mathbf{w}^{(k)\top} \mathbf{x} \cdot \mathbb{I}(\mathbf{w}^{(k)\top} \mathbf{x} \geq 0) \in \mathbb{R}$. For simplicity of notation, we will use $\mathbf{w}$ to refer to a particular $\mathbf{w}$, without considering the index (k) , which does not matter for our purposes. For any $\mathbf{w} \in \mathbb{R}^d$, we denote by $\hat{\beta}_{\mathbf{w}} = (\hat{\beta}_{\mathbf{w}}^{(1)}, \dots, \hat{\beta}_{\mathbf{w}}^{(d)}) \in \mathbb{R}^d$ the weight vectors corresponding to $\mathbf{x} \cdot \mathbb{I}(\mathbf{w}^\top \mathbf{x} \geq 0)$, and denote by $\hat{\beta}'_{\mathbf{w}} \in \mathbb{R}^d$ the weight for $\mathbf{w}^\top \mathbf{x} \cdot \mathbb{I}(\mathbf{w}^\top \mathbf{x} \geq 0)$.

Observe that for any $\mathbf{w} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \in \mathbb{R}^d$, any other vectors in the same direction will activate the same set of $\mathbf{x}_i \in \mathbb{R}^d$. That is, if $\mathbf{w}^\top \mathbf{x}_i \geq 0$ for any $\mathbf{w} \in \mathbb{R}^d$, then $(k \cdot \mathbf{w})^\top \mathbf{x}_i \geq 0$ for any $k > 0$. Hence, we can reload our notation to combine the effect of weights for $\mathbf{w}$'s in the same direction. This enables simpler notations and allows us to change the distribution of $\mathbf{w}$ in NTK features from Gaussian distribution to uniform distribution on the unit sphere.

More precisely, we reload our notation by using $\beta_{\mathbf{w}}$ and $\beta'_{\mathbf{w}}$ to denote the combined effect of all weights $(\hat{\beta}_{k\mathbf{w}}^{(1)}, \dots, \hat{\beta}_{k\mathbf{w}}^{(d)}) \in \mathbb{R}^d$ and $\hat{\beta}'_{k\mathbf{w}} \in \mathbb{R}$ for all $k\mathbf{w}$ with $k > 0$ in the same direction of $\mathbf{w}$. That is, for each $\mathbf{w} \sim \text{Uni}(\text{unit sphere}) \in \mathbb{R}^d$, we define $\beta_{\mathbf{w}}^{(j)}$ as the total effect of weights in the same direction

$$\beta_{\mathbf{w}}^{(j)} = \int \hat{\beta}_{\mathbf{u}}^{(j)} \mathbb{I}\left(\frac{\mathbf{w}^\top \mathbf{u}}{\|\mathbf{w}\| \cdot \|\mathbf{u}\|} = 1\right) d\mathbb{P}(\mathbf{u}), \quad \text{for } j = [d] \quad (49)$$

where $\mathbf{u} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. Note that to ensure the β_w is a well-defined number, here we can work with the polar representation and integrate with respect to an angle. Then β_w is well-defined. But for simplicity of exposition, we use the plain notation of integral. Similarly, we define β'_w as reloading the notation of

$$\beta'_w = \int \hat{\beta}_u \mathbb{I}\left(\frac{\mathbf{w}^\top \mathbf{u}}{\|\mathbf{w}\| \cdot \|\mathbf{u}\|} = 1\right) \cdot \frac{\|\mathbf{u}\|}{\|\mathbf{w}\|} d\mathbb{P}(\mathbf{u}) \quad (50)$$

Here, in Eqn. 50 we have an extra term of $\frac{\|\mathbf{u}\|}{\|\mathbf{w}\|}$ compared to Eqn. 49 because the NTK features that Eqn. 50 corresponds to, $\mathbf{w}^\top \mathbf{x} \cdot \mathbb{I}(\mathbf{w}^\top \mathbf{x} \geq 0)$, has an extra $\mathbf{w}^\top$ term. So we need to take into account the scaling. This abstraction enables us to make claims on the high-level parameters β_w and β'_w only, which we will show to be sufficient to determine the learned function.

Then we can formulate the constrained optimization problem whose solution gives a functional form of the neural network's learned function. We rewrite the min-norm solution in Eqn. 47 as

$$\min_{\beta} \int (\beta_w^{(1)})^2 + (\beta_w^{(2)})^2 + \dots + (\beta_w^{(d)})^2 + (\beta'_w)^2 d\mathbb{P}(\mathbf{w}) \quad (51)$$

$$\text{s.t.} \quad \int_{\mathbf{w}^\top \mathbf{x}_i \geq 0} \beta_w^\top \mathbf{x}_i + \beta'_w \cdot \mathbf{w}^\top \mathbf{x}_i d\mathbb{P}(\mathbf{w}) = \beta_g^\top \mathbf{x}_i \quad \forall i \in [n], \quad (52)$$

where the density of $\mathbf{w}$ is now uniform on the unit sphere of $\mathbb{R}^d$. Observe that since $\mathbf{w}$ is from a uniform distribution, the probability density function $\mathbb{P}(\mathbf{w})$ is a constant. This means every $\mathbf{x}_i$ is activated by half of the $\mathbf{w}$ on the unit sphere, which implies we can now write the right hand side of Eqn. 52 in the form of left hand side, i.e., integral form. This allows us to further simplify

Eqn. 52 as

$$\int_{\mathbf{w}^\top \mathbf{x}_i \geq 0} (\beta_{\mathbf{w}}^\top + \beta'_{\mathbf{w}} \cdot \mathbf{w}^\top - 2 \cdot \beta_g^\top) \mathbf{x}_i \, d\mathbb{P}(\mathbf{w}) = 0 \quad \forall i \in [n], \quad (53)$$

where Eqn. 53 follows from the following steps of simplification

$$\begin{aligned} & \int_{\mathbf{w}^\top \mathbf{x}_i \geq 0} \beta_{\mathbf{w}}^{(1)} \mathbf{x}_i^{(1)} + \dots \beta_{\mathbf{w}}^{(d)} \mathbf{x}_i^{(d)} + \beta'_{\mathbf{w}} \cdot \mathbf{w}^\top \mathbf{x}_i \, d\mathbb{P}(\mathbf{w}) = \beta_g^{(1)} \mathbf{x}_i^{(1)} + \dots \beta_g^{(d)} \mathbf{x}_i^{(d)} \quad \forall i \in [n], \\ \iff & \int_{\mathbf{w}^\top \mathbf{x}_i \geq 0} \beta_{\mathbf{w}}^{(1)} \mathbf{x}_i^{(1)} + \dots + \beta_{\mathbf{w}}^{(d)} \mathbf{x}_i^{(d)} + \beta'_{\mathbf{w}} \cdot \mathbf{w}^\top \mathbf{x}_i \, d\mathbb{P}(\mathbf{w}) \\ &= \frac{1}{\int_{\mathbf{w}^\top \mathbf{x}_i \geq 0} d\mathbb{P}(\mathbf{w})} \cdot \int_{\mathbf{w}^\top \mathbf{x}_i \geq 0} d\mathbb{P}(\mathbf{w}) \cdot \left(\beta_g^{(1)} \mathbf{x}_i^{(1)} + \dots + \beta_g^{(d)} \mathbf{x}_i^{(d)} \right) \quad \forall i \in [n], \\ \iff & \int_{\mathbf{w}^\top \mathbf{x}_i \geq 0} \beta_{\mathbf{w}}^{(1)} \mathbf{x}_i^{(1)} + \dots + \beta_{\mathbf{w}}^{(d)} \mathbf{x}_i^{(d)} + \beta'_{\mathbf{w}} \cdot \mathbf{w}^\top \mathbf{x}_i \, d\mathbb{P}(\mathbf{w}) \\ &= 2 \cdot \int_{\mathbf{w}^\top \mathbf{x}_i \geq 0} \beta_g^{(1)} \mathbf{x}_i^{(1)} + \dots + \beta_g^{(d)} \mathbf{x}_i^{(d)} \, d\mathbb{P}(\mathbf{w}) \quad \forall i \in [n], \\ \iff & \int_{\mathbf{w}^\top \mathbf{x}_i \geq 0} (\beta_{\mathbf{w}}^\top + \beta'_{\mathbf{w}} \cdot \mathbf{w}^\top - 2 \cdot \beta_g^\top) \mathbf{x}_i \, d\mathbb{P}(\mathbf{w}) = 0 \quad \forall i \in [n]. \end{aligned}$$

Claim 7.4.1. *Without loss of generality, assume the scaling factor c in NTK feature map $\phi(\mathbf{x})$ is*

1. *Then the global optimum to the constraint optimization problem Eqn. 51 subject to Eqn. 53,*

i.e.,

$$\min_{\beta} \int (\beta_{\mathbf{w}}^{(1)})^2 + (\beta_{\mathbf{w}}^{(2)})^2 + \dots + (\beta_{\mathbf{w}}^{(d)})^2 + (\beta'_{\mathbf{w}})^2 \, d\mathbb{P}(\mathbf{w}) \quad (54)$$

$$\text{s.t.} \quad \int_{\mathbf{w}^\top \mathbf{x}_i \geq 0} (\beta_{\mathbf{w}}^\top + \beta'_{\mathbf{w}} \cdot \mathbf{w}^\top - 2 \cdot \beta_g^\top) \mathbf{x}_i \, d\mathbb{P}(\mathbf{w}) = 0 \quad \forall i \in [n]. \quad (55)$$

satisfies $\beta_w + \beta'_w \cdot w = 2\beta_g$ for all w .

This claim implies the exact extrapolation we want to prove, i.e., $f_{\text{NTK}}(x) = g(x)$. This is because, if our claim holds, then for any $x \in \mathbb{R}^d$

$$\begin{aligned} f_{\text{NTK}}(x) &= \int_{w^\top x \geq 0} \beta_w^\top x + \beta'_w \cdot w^\top x \, d\mathbb{P}(w) \\ &= \int_{w^\top x \geq 0} 2 \cdot \beta_g^\top x \, d\mathbb{P}(w) \\ &= \int_{w^\top x \geq 0} d\mathbb{P}(w) \cdot 2\beta_g^\top x \\ &= \frac{1}{2} \cdot 2\beta_g^\top x = g(x) \end{aligned}$$

Thus, it remains to prove Claim 7.4.1. To compute the optimum to the constrained optimization problem Eqn. 54, we consider the Lagrange multipliers. It is clear that the objective Eqn. 54 is convex. Moreover, the constraint Eqn. 55 is affine. Hence, by KKT, solution that satisfies the Lagrange condition will be the global optimum. We compute the Lagrange multiplier as

$$\mathcal{L}(\beta, \lambda) = \int (\beta_w^{(1)})^2 + (\beta_w^{(2)})^2 + \dots + (\beta_w^{(d)})^2 + (\beta'_w)^2 d\mathbb{P}(w) \quad (56)$$

$$- \sum_{i=1}^n \lambda_i \cdot \left(\int_{w^\top x_i \geq 0} (\beta_w^\top + \beta'_w \cdot w^\top - 2 \cdot \beta_g^\top) x_i \, d\mathbb{P}(w) \right) \quad (57)$$

Setting the partial derivative of $\mathcal{L}(\beta, \lambda)$ with respect to each variable to zero gives

$$\frac{\partial \mathcal{L}}{\partial \beta_w^{(k)}} = 2\beta_w^{(k)} \mathbb{P}(w) + \sum_{i=1}^n \lambda_i \cdot x_i^{(k)} \cdot \mathbb{I}(w^\top x_i \geq 0) = 0 \quad (58)$$

$$\frac{\partial \mathcal{L}}{\partial \beta'_w} = 2\beta'_w \mathbb{P}(w) + \sum_{i=1}^n \lambda_i \cdot w^\top x_i \cdot \mathbb{I}(w^\top x_i \geq 0) = 0 \quad (59)$$

$$\frac{\partial \mathcal{L}}{\partial \lambda_i} = \int_{w^\top x_i \geq 0} (\beta_w^\top + \beta'_w \cdot w^\top - 2 \cdot \beta_g^\top) x_i \, d\mathbb{P}(w) = 0 \quad (60)$$

It is clear that the solution in Claim 7.4.1 immediately satisfies Eqn. 60. Hence, it remains to show there exist a set of λ_i for $i \in [n]$ that satisfies Eqn. 58 and Eqn. 59. We can simplify Eqn. 58 as

$$\beta_w^{(k)} = c \cdot \sum_{i=1}^n \lambda_i \cdot x_i^{(k)} \cdot \mathbb{I}(w^\top x_i \geq 0), \quad (61)$$

where c is a constant. Similarly, we can simplify Eqn. 59 as

$$\beta'_w = c \cdot \sum_{i=1}^n \lambda_i \cdot w^\top x_i \cdot \mathbb{I}(w^\top x_i \geq 0) \quad (62)$$

Observe that combining Eqn. 61 and Eqn. 62 implies that the constraint Eqn. 62 can be further simplified as

$$\beta'_w = w^\top \beta_w \quad (63)$$

It remains to show that given the condition on training data, there exists a set of λ_i so that Eqn. 61 and Eqn. 63 are satisfied.

7.4.2 Global optimum via the geometry of training data.

Recall that we assume our training data $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$ satisfies for any $\mathbf{w} \in \mathbb{R}^d$, there exist d linearly independent $\{\mathbf{x}_i^{\mathbf{w}}\}_{i=1}^d \subset \mathbf{X}$, where $\mathbf{X} = \{\mathbf{x}_i\}_{i=1}^n$, so that $\mathbf{w}^\top \mathbf{x}_i^{\mathbf{w}} \geq 0$ and $-\mathbf{x}_i^{\mathbf{w}} \in \mathbf{X}$ for $i = 1..d$, e.g., an orthogonal basis of $\mathbb{R}^d$ and their opposite vectors. We will show that under this data regime, we have

(a) for any particular $\mathbf{w}$, there indeed exist a set of λ_i that can satisfy the constraints Eqn. 61 and Eqn. 63 for this particular $\mathbf{w}$.

(b) For any $\mathbf{w}_1$ and $\mathbf{w}_2$ that activate the exact same set of $\{\mathbf{x}_i\}$, the same set of λ_i can satisfy the constraints Eqn. 61 and Eqn. 63 of both $\mathbf{w}_1$ and $\mathbf{w}_2$.

(c) Whenever we rotate a $\mathbf{w}_1$ to a $\mathbf{w}_2$ so that the set of $\mathbf{x}_i$ being activated changed, we can still find λ_i that satisfy constraint of both $\mathbf{w}_1$ and $\mathbf{w}_2$.

Combining (a), (b) and (c) implies there exists a set of λ that satisfy the constraints for all $\mathbf{w}$. Hence, it remains to show these three claims.

We first prove Claim (a). For each $\mathbf{w}$, we must find a set of λ_i so that the following hold.

$$\beta_{\mathbf{w}}^{(k)} = c \cdot \sum_{i=1}^n \lambda_i \cdot \mathbf{x}_i^{(k)} \cdot \mathbb{I}(\mathbf{w}^\top \mathbf{x}_i \geq 0),$$

$$\beta'_{\mathbf{w}} = \mathbf{w}^\top \beta_{\mathbf{w}}$$

$$\beta_{\mathbf{w}} + \beta'_{\mathbf{w}} \cdot \mathbf{w} = 2\beta_g$$

Here, β_g and $\mathbf{w}$ are fixed, and $\mathbf{w}$ is a vector on the unit sphere. It is easy to see that $\beta_{\mathbf{w}}$ is then determined by β_g and $\mathbf{w}$, and there indeed exists a solution (solving a consistent linear system).

Hence we are left with a linear system with d linear equations

$$\beta_{\mathbf{w}}^{(k)} = c \cdot \sum_{i=1}^n \lambda_i \cdot \mathbf{x}_i^{(k)} \cdot \mathbb{I}(\mathbf{w}^\top \mathbf{x}_i \geq 0) \quad \forall k \in [d]$$

to solve with free variables being λ_i so that $\mathbf{w}$ activates $\mathbf{x}_i$, i.e., $\mathbf{w}^\top \mathbf{x}_i \geq 0$. Because the training data $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$ satisfies for any $\mathbf{w}$, there exist at least d linearly independent $\mathbf{x}_i$ that activate $\mathbf{w}$. This guarantees for any $\mathbf{w}$ we must have at least d free variables. It follows that there must exist solutions λ_i to the linear system. This proves Claim (a).

Next, we show that (b) for any $\mathbf{w}_1$ and $\mathbf{w}_2$ that activate the exact same set of $\{\mathbf{x}_i\}$, the same set of λ_i can satisfy the constraints Eqn. 61 and Eqn. 63 of both $\mathbf{w}_1$ and $\mathbf{w}_2$. Because $\mathbf{w}_1$ and $\mathbf{w}_2$ are activated by the same set of $\mathbf{x}_i$, this implies

$$\beta_{\mathbf{w}_1} = c \cdot \sum_{i=1}^n \lambda_i \cdot \mathbf{x}_i \cdot \mathbb{I}(\mathbf{w}_1^\top \mathbf{x}_i \geq 0) = c \cdot \sum_{i=1}^n \lambda_i \cdot \mathbf{x}_i \cdot \mathbb{I}(\mathbf{w}_2^\top \mathbf{x}_i \geq 0) = \beta_{\mathbf{w}_2}$$

Since λ_i already satisfy constraint Eqn. 61 for $\mathbf{w}_1$, they also satisfy that for $\mathbf{w}_2$. Thus, it remains to show that $\beta_{\mathbf{w}_1} + \beta'_{\mathbf{w}_1} \cdot \mathbf{w}_1 = \beta_{\mathbf{w}_2} + \beta'_{\mathbf{w}_2} \cdot \mathbf{w}_1$ assuming $\beta_{\mathbf{w}_1} = \beta_{\mathbf{w}_2}$, $\beta'_{\mathbf{w}_1} = \mathbf{w}_1^\top \beta_{\mathbf{w}_1}$, and $\beta'_{\mathbf{w}_2} = \mathbf{w}_2^\top \beta_{\mathbf{w}_2}$. This indeed holds because

$$\begin{aligned} & \beta_{\mathbf{w}_1} + \beta'_{\mathbf{w}_1} \cdot \mathbf{w}_1 = \beta_{\mathbf{w}_2} + \beta'_{\mathbf{w}_2} \cdot \mathbf{w}_2 \\ \iff & \beta'_{\mathbf{w}_1} \cdot \mathbf{w}_1^\top = \beta'_{\mathbf{w}_2} \cdot \mathbf{w}_2^\top \\ \iff & \mathbf{w}_1^\top \beta_{\mathbf{w}_1} \mathbf{w}_1^\top = \mathbf{w}_2^\top \beta_{\mathbf{w}_2} \mathbf{w}_2^\top \\ \iff & \mathbf{w}_1^\top \mathbf{w}_1 \beta_{\mathbf{w}_1}^\top = \mathbf{w}_2^\top \mathbf{w}_2 \beta_{\mathbf{w}_2}^\top \\ \iff & 1 \cdot \beta_{\mathbf{w}_1}^\top = 1 \cdot \beta_{\mathbf{w}_2}^\top \end{aligned}$$

$$\Longleftrightarrow \beta_{w_1} = \beta_{w_1}$$

Here, we used the fact that w_1 and w_2 are vectors on the unit sphere. This proves Claim (b).

Finally, we show (c) that Whenever we rotate a w_1 to a w_2 so that the set of x_i being activated changed, we can still find λ_i that satisfy constraint of both w_1 and w_2 . Suppose we rotate w_1 to w_2 so that w_2 lost activation with $x_1, x_2, \dots, x_p$ which in the set of linearly independent x_i 's being activated by w_1 and their opposite vectors $-x_i$ are also in the training set (without loss of generality). Then w_2 must now also get activated by $-x_1, -x_2, \dots, -x_p$. This is because if $w_2^\top x_i < 0$, we must have $w_2^\top (-x_i) > 0$.

Recall that in the proof of Claim (a), we only needed the λ_i from linearly independent x_i that we used to solve the linear systems, and their opposite as the free variables to solve the linear system of d equations. Hence, we can set λ to 0 for the other x_i while still satisfying the linear system. Then, suppose there exists λ_i that satisfy

$$\beta_{w_1}^{(k)} = c \cdot \sum_{i=1}^d \lambda_i \cdot x_i^{(k)}$$

where the x_i are the linearly independent vectors that activate w_1 with opposite vectors in the training set, which we have proved in (a). Then we can satisfy the constraint for β_{w_2} below

$$\beta_{w_2}^{(k)} = c \cdot \sum_{i=1}^p \hat{\lambda}_i \cdot (-x_i)^{(k)} + \sum_{i=p+1}^d \lambda_i \cdot x_i^{(k)}$$

by setting $\hat{\lambda}_i = -\lambda_i$ for $i = 1 \dots p$. Indeed, this gives

$$\begin{aligned}\beta_{w_2}^{(k)} &= c \cdot \sum_{i=1}^p (-\lambda_i) \cdot (-x_i)^{(k)} + \sum_{i=p+1}^d \lambda_i \cdot x_i^{(k)} \\ &= c \cdot \sum_{i=1}^d \lambda_i \cdot x_i^{(k)}\end{aligned}$$

Thus, we can also find λ_i that satisfy the constraint for β_{w_2} . Here, we do not consider the case where w_2 is parallel with an x_i because such w_2 has measure zero. Note that we can apply this argument iteratively because the flipping the sign always works and will not create any inconsistency.

Moreover, we can show that the constraint for β'_{w_2} is satisfied by a similar argument as in proof of Claim (b). This follows from the fact that our construction makes $\beta_{w_1} = \beta_{w_2}$. Then we can follow the same argument as in (b) to show that $\beta_{w_1} + \beta'_{w_1} \cdot w_1 = \beta_{w_2} + \beta'_{w_2} \cdot w_1$. This completes the proof of Claim (c).

In summary, combining Claim (a), (b) and (c) gives that Claim 7.4.1 holds. That is, given our training data, the global optimum to the constrained optimization problem of finding the min-norm solution among functions that fit the training data satisfies $\beta_w + \beta'_w \cdot w = 2\beta_g$. We also showed that this claim implies exact extrapolation, i.e., the network's learned function $f(x)$ is equal to the true underlying function $g(x)$ for all $x \in \mathbb{R}^d$. This completes the proof.

7.4.3 Proof of Theorem 3.3.4

Proof of the asymptotic convergence to extrapolation builds upon our proof of exact extrapolation, i.e., Lemma 3.3.3. The proof idea is that if the training data distribution has support at all

directions, when the number of samples $n \rightarrow \infty$, asymptotically the training set will converge to some imaginary training set that satisfies the condition for exact extrapolation. Since if training data are close the neural tangent kernels are also close, the predictions or learned function will converge to a function that achieves perfect extrapolation, that is, the true underlying function.

7.4.4 Asymptotic convergence of data sets.

We first show the training data converge to a data set that satisfies the exact extrapolation condition in Lemma 3.3.3. Suppose training data $\{\mathbf{x}_i\}_{i=1}^n$ are sampled from a distribution whose support contains a connected set $\mathcal{S}$ that intersects all directions, i.e., for any non-zero $\mathbf{w} \in \mathbb{R}^d$, there exists $k > 0$ so that $k\mathbf{w} \in \mathcal{S}$.

Let us denote by $\mathcal{S}$ the set of datasets that satisfy the condition in Lemma 3.3.3. In fact, we will use a relaxed condition in the proof of Lemma 3.3.3 (Lemma 3.3.3 in the main text uses a stricter condition for simplicity of exposition). Given a general dataset $\mathbf{X}$ and a dataset $\mathbf{S} \in \mathcal{S}$ of the same size n , let $\sigma(\mathbf{X}, \mathbf{S})$ denote a matching of their data points, i.e., σ outputs a sequence of pairs

$$\sigma(\mathbf{X}, \mathbf{S})_i = (\mathbf{x}_i, \mathbf{s}_i) \quad \text{for } i \in [n]$$

$$s.t. \quad \mathbf{X} = \{\mathbf{x}_i\}_{i=1}^n$$

$$\mathbf{S} = \{\mathbf{s}_i\}_{i=1}^n$$

Let $\ell : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$ be the l_2 distance that takes in a pair of points. We then define the distance between the datasets $d(\mathbf{X}, \mathbf{S})$ as the minimum sum of l_2 distances of their data points

over all possible matching.

$$d(\mathbf{X}, \mathbf{S}) = \begin{cases} \min_{\sigma} \sum_{i=1}^n \ell(\sigma(\mathbf{X}, \mathbf{S})_i) & |\mathbf{X}| = |\mathbf{S}| = n \\ \infty & |\mathbf{X}| \neq |\mathbf{S}| \end{cases}$$

We can then define a “closest distance to perfect dataset” function $\mathcal{D}^* : \mathcal{X} \rightarrow \mathbb{R}$ which maps a dataset $\mathbf{X}$ to the minimum distance of $\mathbf{X}$ to any dataset in $\mathcal{S}$

$$\mathcal{D}^*(\mathbf{X}) = \min_{\mathbf{S} \in \mathcal{S}} d(\mathbf{X}, \mathbf{S})$$

It is easy to see that for any dataset $\mathbf{X} = \{\mathbf{x}_i\}_{i=1}^n$, $\mathcal{D}^*(\mathbf{X})$ can be bounded by the minimum of the closest distance to perfect dataset $\mathcal{D}^*$ of sub-datasets of $\mathbf{X}$ of size $2d$.

$$\mathcal{D}^*(\{\mathbf{x}_i\}_{i=1}^n) \leq \min_{k=1}^{\lfloor n/2d \rfloor} \mathcal{D}^*(\{\mathbf{x}_j\}_{j=(k-1)*2d+1}^{k*2d}) \quad (64)$$

This is because for any $\mathbf{S} \in \mathcal{S}$, and any $\mathbf{S} \subseteq \mathbf{S}'$, we must have $\mathbf{S}' \in \mathcal{S}$ because a dataset satisfies exact extrapolation condition as long as it contains some key points. Thus, adding more data will not hurt, i.e., for any $\mathbf{X}_1 \subseteq \mathbf{X}_2$, we always have

$$\mathcal{D}^*(\mathbf{X}_1) \leq \mathcal{D}^*(\mathbf{X}_2)$$

Now let us denote by $\mathbf{X}_n$ a random dataset of size n where each $\mathbf{x}_i \in \mathbf{X}_n$ is sampled from the training distribution. Recall that our training data $\{\mathbf{x}_i\}_{i=1}^n$ are sampled from a distribution whose support contains a connected set $\mathcal{S}^*$ that intersects all directions, i.e., for any non-zero

$\mathbf{w} \in \mathbb{R}^d$, there exists $k > 0$ so that $k\mathbf{w} \in \mathcal{S}^*$. It follows that for a random dataset $\mathbf{X}_{2d}$ of size $2d$, the probability that $\mathcal{D}^*(\mathbf{X}_{2d}) > \epsilon$ happens is less than 1 for any $\epsilon > 0$.

First there must exist $\mathcal{S}_0 = \{\mathbf{s}_i\}_{i=1}^{2d} \in \mathcal{S}$ of size $2d$, e.g., orthogonal basis and their opposite vectors. Observe that if we scale any $\mathbf{s}_i$ by $k > 0$, the resulting dataset is still in $\mathcal{S}$ by the definition of $\mathcal{S}$. We denote the set of datasets where we are allowed to scale elements of $\mathcal{S}_0$ by $\mathcal{S}_0$. It follows that

$$\begin{aligned}
\mathbb{P}(\mathcal{D}^*(\mathbf{X}_{2d}) > \epsilon) &= \mathbb{P}\left(\min_{\mathbf{S} \in \mathcal{S}} d(\mathbf{X}_{2d}, \mathbf{S}) > \epsilon\right) \\
&\leq \mathbb{P}\left(\min_{\mathbf{S} \in \mathcal{S}_0} d(\mathbf{X}_{2d}, \mathbf{S}) > \epsilon\right) \\
&= \mathbb{P}\left(\min_{\mathbf{S} \in \mathcal{S}_0} \min_{\sigma} \sum_{i=1}^n \ell(\sigma(\mathbf{X}_{2d}, \mathbf{S})_i) > \epsilon\right) \\
&= 1 - \mathbb{P}\left(\min_{\mathbf{S} \in \mathcal{S}_0} \min_{\sigma} \sum_{i=1}^n \ell(\sigma(\mathbf{X}_{2d}, \mathbf{S})_i) \leq \epsilon\right) \\
&\leq 1 - \mathbb{P}\left(\min_{\mathbf{S} \in \mathcal{S}_0} \min_{\sigma} \max_{i=1}^n \ell(\sigma(\mathbf{X}_{2d}, \mathbf{S})_i) \leq \epsilon\right) \\
&\leq \delta < 1
\end{aligned}$$

where we denote the bound of $\mathbb{P}(\mathcal{D}^*(\mathbf{X}_{2d}) > \epsilon)$ by $\delta < 1$, and the last step follows from

$$\mathbb{P}\left(\min_{\mathbf{S} \in \mathcal{S}_0} \min_{\sigma} \max_{i=1}^n \ell(\sigma(\mathbf{X}_{2d}, \mathbf{S})_i) \leq \epsilon\right) > 0$$

which further follows from the fact that for any $\mathbf{s}_i \in \mathcal{S}_0$, by the assumption on training distribution, we can always find $k > 0$ so that $k\mathbf{s}_i \in \mathcal{S}^*$, a connected set in the support of training distribution. By the connectivity of support $\mathcal{S}^*$, $k\mathbf{s}_i$ cannot be an isolated point in $\mathcal{S}^*$, so for any

$\epsilon > 0$, we must have

$$\int_{\|\mathbf{x} - k\mathbf{s}_i\| \leq \epsilon, \mathbf{x} \in \mathcal{S}^*} f_{\mathbf{X}}(\mathbf{x}) d\mathbf{x} > 0$$

Hence, we can now apply Eqn. 64 to bound $\mathcal{D}^*(\mathbf{X}_n)$. Given any $\epsilon > 0$, we have

$$\begin{aligned} \mathbb{P}(\mathcal{D}^*(\mathbf{X}_n) > \epsilon) &= 1 - \mathbb{P}(\mathcal{D}^*(\mathbf{X}_n) \leq \epsilon) \\ &\leq 1 - \mathbb{P}\left(\min_{k=1}^{\lfloor n/2d \rfloor} \mathcal{D}^*(\{\mathbf{x}_j\}_{j=(k-1)*2d+1}^{k*2d}) \leq \epsilon\right) \\ &\leq 1 - \left(1 - \prod_{k=1}^{\lfloor n/2d \rfloor} \mathbb{P}(\mathcal{D}^*(\{\mathbf{x}_j\}_{j=(k-1)*2d+1}^{k*2d}) > \epsilon)\right) \\ &= \prod_{k=1}^{\lfloor n/2d \rfloor} \mathbb{P}(\mathcal{D}^*(\{\mathbf{x}_j\}_{j=(k-1)*2d+1}^{k*2d}) > \epsilon) \\ &\leq \delta^{\lfloor n/2d \rfloor} \end{aligned}$$

Here $\delta < 1$. This implies $\mathcal{D}^*(\mathbf{X}_n) \xrightarrow{p} 0$, i.e.,

$$\lim_{n \rightarrow \infty} \mathbb{P}(\mathcal{D}^*(\mathbf{X}_n) > \epsilon) = 0 \quad \forall \epsilon > 0 \quad (65)$$

Eqn. 65 says as the number of training samples $n \rightarrow \infty$, our training set will converge in probability to a dataset that satisfies the requirement for exact extrapolation.

7.4.5 Asymptotic convergence of predictions.

Let $\text{NTK}(\mathbf{x}, \mathbf{x}') : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$ denote the neural tangent kernel for a two-layer ReLU MLP. It is easy to see that if $\mathbf{x} \rightarrow \mathbf{x}^*$, then $\text{NTK}(\mathbf{x}, \cdot) \rightarrow \text{NTK}(\mathbf{x}^*, \cdot)$ [Arora et al. \(2019b\)](#). Let

$\text{NTK}_{\text{train}}$ denote the $n \times n$ kernel matrix for training data.

We have shown that our training set converges to a perfect data set that satisfies conditions of exact extrapolation. Moreover, note that our training set will only have a finite number of (not increase with n) $\mathbf{x}_i$ that are not precisely the same as those in a perfect dataset. This is because a perfect data only contains a finite number of key points and the other points can be replaced by any other points while still being a perfect data set. Thus, we have $\text{NTK}_{\text{train}} \rightarrow N^*$, where N^* is the $n \times n$ NTK matrix for some perfect data set.

Because neural tangent kernel is positive definite, we have $\text{NTK}_{\text{train}}^{-1} \rightarrow N^{*-1}$. Recall that for any $\mathbf{x} \in \mathbb{R}^d$, the prediction of NTK is

$$f_{\text{NTK}}(\mathbf{x}) = (\text{NTK}(\mathbf{x}, \mathbf{x}_1), \dots, \text{NTK}(\mathbf{x}, \mathbf{x}_n)) \cdot \text{NTK}_{\text{train}}^{-1} \mathbf{Y},$$

where $\text{NTK}_{\text{train}}$ is the $n \times n$ kernel for training data, $\text{NTK}(\mathbf{x}, \mathbf{x}_i)$ is the kernel value between test data $\mathbf{x}$ and training data $\mathbf{x}_i$, and $\mathbf{Y}$ is training labels.

Similarly, we have $(\text{NTK}(\mathbf{x}, \mathbf{x}_1), \dots, \text{NTK}(\mathbf{x}, \mathbf{x}_n)) \rightarrow (\text{NTK}(\mathbf{x}, \mathbf{x}_1^*), \dots, \text{NTK}(\mathbf{x}, \mathbf{x}_n^*))$, where $\mathbf{x}_i^*$ is a perfect data set that our training set converges to. Combining this with $\text{NTK}_{\text{train}}^{-1} \rightarrow N^{*-1}$ gives

$$f_{\text{NTK}} \xrightarrow{p} f_{\text{NTK}}^* = g,$$

where f_{NTK} is the function learned using our training set, and f_{NTK}^* is that learned using a perfect data set, which is equal to the true underlying function g . This completes the proof.

7.4.6 Proof of Corollary 3.4.3

In order for GNN with linear aggregations

$$h_u^{(k)} = \sum_{v \in \mathcal{N}(u)} \text{MLP}^{(k)}(h_u^{(k)}, h_v^{(k)}, \mathbf{x}_{(u,v)}),$$

$$h_G = \text{MLP}^{(K+1)}\left(\sum_{u \in G} h_u^{(K)}\right),$$

to extrapolate in the maximum degree task, it must be able to simulate the underlying function

$$h_G = \max_{u \in G} \sum_{v \in \mathcal{N}(u)} 1$$

Because the max function cannot be decomposed as the composition of piece-wise linear functions, the $\text{MLP}^{(K+1)}$ module in GNN must learn a function that is not piece-wise linear over domains outside the training data range. Since Theorem 3.3.2 proves for two-layer overparameterized MLPs, here we also assume $\text{MLP}^{(K+1)}$ is a two-layer overparameterized MLP, although the result can be extended to more layers. It then follows from Theorem 3.3.2 that for any input and label (and thus gradient), $\text{MLP}^{(K+1)}$ will converge to linear functions along directions from the origin. Hence, there are always domains where the GNN cannot learn a correct target function.

7.4.7 Proof of Theorem 3.4.4

Our proof applies the similar proof techniques for Lemma 3.3.3 and 3.3.4 to Graph Neural Networks (GNNs). This is essentially an analysis of Graph Neural Tangent Kernel (GNTK), i.e.,

neural tangent kernel of GNNs.

We first define the simple GNN architecture we will be analyzing, and then present the GNTK for this architecture. Suppose $G = (V, E)$ is an input graph without edge feature, and $\mathbf{x}_u \in \mathbb{R}^d$ is the node feature of any node $u \in V$. Let us consider the simple one-layer GNN whose input is G and output is h_G

$$h_G = W^{(2)} \max_{u \in G} \sum_{v \in \mathcal{N}(u)} W^{(1)} \mathbf{x}_v \quad (66)$$

Note that our analysis can be extended to other variants of GNNs, e.g., with non-empty edge features, ReLU activation, different neighbor aggregation and graph-level pooling architectures. We analyze this GNN for simplicity of exposition.

Next, let us calculate the feature map of the neural tangent kernel for this GNN. Recall from Section 7.3 that consider a graph neural network $f(\boldsymbol{\theta}, G) : \mathcal{G} \rightarrow \mathbb{R}$ where $\boldsymbol{\theta} \in \mathbb{R}^m$ is the parameters in the network and $G \in \mathcal{G}$ is the input graph. Then the neural tangent kernel is

$$\mathbf{H}_{ij} = \left\langle \frac{\partial f(\boldsymbol{\theta}, G_i)}{\partial \boldsymbol{\theta}}, \frac{\partial f(\boldsymbol{\theta}, G_j)}{\partial \boldsymbol{\theta}} \right\rangle,$$

where $\boldsymbol{\theta}$ are the infinite-dimensional parameters. Hence, the gradients with respect to all parameters give a natural feature map. Let us denote, for any node u , the degree of u by

$$\mathbf{h}_u = \sum_{v \in \mathcal{N}(u)} \mathbf{x}_v \quad (67)$$

It then follows from simple computation of derivative that the following is a feature map of the

GNTK for Eqn. 66

$$\phi(G) = c \cdot \left(\max_{u \in G} (\mathbf{w}^{(k)\top} \mathbf{h}_u), \sum_{u \in G} \mathbb{I} \left(u = \arg \max_{v \in G} \mathbf{w}^{(k)\top} \mathbf{h}_v \right) \cdot \mathbf{h}_u, \dots \right), \quad (68)$$

where $\mathbf{w}^{(k)} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, with k going to infinity. c is a constant, and $\mathbb{I}$ is the indicator function.

Next, given training data $\{(G_i, y_i)\}_{i=1}^n$, let us analyze the function learned by GNN through the min-norm solution in the GNTK feature space. The same proof technique is also used in Lemma 3.3.3 and 3.3.4.

Recall the assumption that all graphs have uniform node feature, i.e., the learning task only considers graph structure, but not node feature. We assume $x_v = 1$ without loss of generality. Observe that in this case, there are two directions, positive or negative, for one-dimensional Gaussian distribution. Hence, we can simplify our analysis by combining the effect of linear coefficients for $\mathbf{w}$ in the same direction as in Lemma 3.3.3 and 3.3.4.

Similarly, for any $\mathbf{w}$, let us define $\hat{\beta}_{\mathbf{w}} \in \mathbb{R}$ as the linear coefficient corresponding to $\sum_{u \in G} \mathbb{I} \left(u = \arg \max_{v \in G} \mathbf{w}^\top \mathbf{h}_v \right) \cdot \mathbf{h}_u$ in RKHS space, and denote by $\hat{\beta}'_{\mathbf{w}} \in \mathbb{R}$ the weight for $\max_{u \in G} (\mathbf{w}^\top \mathbf{h}_u)$. Similarly, we can combine the effect of all $\hat{\beta}$ in the same direction as in Lemma 3.3.3 and 3.3.4.

We define the combined effect with $\beta_{\mathbf{w}}$ and $\beta'_{\mathbf{w}}$. This allows us to reason about $\mathbf{w}$ with two directions, $+$ and $-$.

Recall that the underlying reasoning function, maximum degree, is

$$g(G) = \max_{u \in G} \mathbf{h}_u.$$

We formulate the constrained optimization problem, i.e., min-norm solution in GNTK fea-

ture space that fits all training data, as

$$\begin{aligned} & \min_{\hat{\beta}, \hat{\beta}'} \int \hat{\beta}_{\mathbf{w}}^2 + \hat{\beta}'_{\mathbf{w}}^2 d\mathbb{P}(\mathbf{w}) \\ \text{s.t. } & \int \sum_{u \in G_i} \mathbb{I}\left(u = \arg \max_{v \in G_i} \mathbf{w} \cdot \mathbf{h}_v\right) \cdot \hat{\beta}_{\mathbf{w}} \cdot \mathbf{h}_u + \max_{u \in G_i} (\mathbf{w} \cdot \mathbf{h}_u) \cdot \hat{\beta}'_{\mathbf{w}} d\mathbb{P}(\mathbf{w}) = \max_{u \in G_i} \mathbf{h}_u \quad \forall i \in [n], \end{aligned}$$

where G_i is the i -th training graph and $\mathbf{w} \sim \mathcal{N}(0, 1)$. By combining the effect of $\hat{\beta}$, and taking the derivative of the Lagrange for the constrained optimization problem and setting to zero, we get the global optimum solution satisfy the following constraints.

$$\beta_+ = c \cdot \sum_{i=1}^n \lambda_i \cdot \sum_{u \in G_i} \mathbf{h}_u \cdot \mathbb{I}\left(u = \arg \max_{v \in G_i} \mathbf{h}_v\right) \quad (69)$$

$$\beta_- = c \cdot \sum_{i=1}^n \lambda_i \cdot \sum_{u \in G_i} \mathbf{h}_u \cdot \mathbb{I}\left(u = \arg \min_{v \in G_i} \mathbf{h}_v\right) \quad (70)$$

$$\beta'_+ = c \cdot \sum_{i=1}^n \lambda_i \cdot \max_{u \in G_i} \mathbf{h}_u \quad (71)$$

$$\beta'_- = c \cdot \sum_{i=1}^n \lambda_i \cdot \min_{u \in G_i} \mathbf{h}_u \quad (72)$$

$$\max_{u \in G_i} \mathbf{h}_u = \beta_+ \cdot \sum_{u \in G_i} \mathbb{I}\left(u = \arg \max_{v \in G_i} \mathbf{h}_v\right) \cdot \mathbf{h}_u + \beta'_+ \cdot \max_{u \in G_i} \mathbf{h}_u \quad (73)$$

$$+ \beta_- \cdot \sum_{u \in G_i} \mathbb{I}\left(u = \arg \min_{v \in G_i} \mathbf{h}_v\right) \cdot \mathbf{h}_u + \beta'_- \cdot \min_{u \in G_i} \mathbf{h}_u \quad \forall i \in [n] \quad (74)$$

where c is some constant, λ_i are the Lagrange parameters. Note that here we used the fact that there are two directions $+1$ and -1 . This enables the simplification of Lagrange derivative. For a similar step-by-step derivation of Lagrange, refer to the proof of Lemma 3.3.3.

Let us consider the solution $\beta'_+ = 1$ and $\beta_+ = \beta_- = \beta'_- = 0$. It is clear that this solution can fit the training data, and thus satisfies Eqn. 73. Moreover, this solution is equivalent to the

underlying reasoning function, maximum degree, $g(G) = \max_{u \in G} \mathbf{h}_u$.

Hence, it remains to show that, given our training data, there exist λ_i so that the remaining four constraints are satisfies for this solution. Let us rewrite these constraints as a linear systems where the variables are λ_i

$$\begin{pmatrix} \beta_+ \\ \beta_- \\ \beta'_+ \\ \beta'_- \end{pmatrix} = c \cdot \sum_{i=1}^n \lambda_i \cdot \begin{pmatrix} \sum_{u \in G_i} \mathbf{h}_u \cdot \mathbb{I}\left(u = \arg \max_{v \in G_i} \mathbf{h}_v\right) \\ \sum_{u \in G_i} \mathbf{h}_u \cdot \mathbb{I}\left(u = \arg \min_{v \in G_i} \mathbf{h}_v\right) \\ \max_{u \in G_i} \mathbf{h}_u \\ \min_{u \in G_i} \mathbf{h}_u \end{pmatrix} \quad (75)$$

By standard theory of linear systems, there exist λ_i to solve Eqn. 75 if there are at least four training data G_i whose following vectors linear independent

$$\begin{pmatrix} \sum_{u \in G_i} \mathbf{h}_u \cdot \mathbb{I}\left(u = \arg \max_{v \in G_i} \mathbf{h}_v\right) \\ \sum_{u \in G_i} \mathbf{h}_u \cdot \mathbb{I}\left(u = \arg \min_{v \in G_i} \mathbf{h}_v\right) \\ \max_{u \in G_i} \mathbf{h}_u \\ \min_{u \in G_i} \mathbf{h}_u \end{pmatrix} = \begin{pmatrix} \max_{u \in G_i} \mathbf{h}_u \cdot N_i^{\max} \\ \min_{u \in G_i} \mathbf{h}_u \cdot N_i^{\min} \\ \max_{u \in G_i} \mathbf{h}_u \\ \min_{u \in G_i} \mathbf{h}_u \end{pmatrix} \quad (76)$$

Here, $N_i^{\max}$ denotes the number of nodes that achieve the maximum degree in the graph G_i , and $N_i^{\min}$ denotes the number of nodes that achieve the min degree in the graph G_i . By the assumption of our training data that there are at least four $G_i \sim \mathcal{G}$ with linearly independent Eqn. 76. Hence, our simple GNN learns the underlying function as desired.

This completes the proof.

7.4.8 Proof of Lemma 7.3.1

Let W denote the span of the feature maps of training data $\mathbf{x}_i$, i.e.

$$W = \text{span}(\phi(\mathbf{x}_1), \phi(\mathbf{x}_2), \dots, \phi(\mathbf{x}_n)).$$

Then we can decompose the coordinates of f_{NTK} in the RKHS space, β_{NTK} , into a vector β_0 for the component of f_{NTK} in the span of training data features W , and a vector β_1 for the component in the orthogonal complement $W^\perp$, i.e.,

$$\beta_{\text{NTK}} = \beta_0 + \beta_1.$$

First, note that since f_{NTK} must be able to fit the training data (NTK is a universal kernel as we will discuss next), i.e.,

$$\phi(\mathbf{x}_i)^\top \beta_{\text{NTK}} = y_i.$$

Thus, we have $\phi(\mathbf{x}_i)^\top \beta_0 = y_i$. Then, β_0 is uniquely determined by the kernel regression solution with respect to the neural tangent kernel

$$f_{\text{NTK}}(\mathbf{x}) = (\langle \phi(\mathbf{x}), \phi(\mathbf{x}_1) \rangle, \dots, \langle \phi(\mathbf{x}), \phi(\mathbf{x}_n) \rangle) \cdot \text{NTK}_{\text{train}}^{-1} \mathbf{Y},$$

where $\text{NTK}_{\text{train}}$ is the $n \times n$ kernel for training data, $\langle \phi(\mathbf{x}), \phi(\mathbf{x}_i) \rangle$ is the kernel between test data $\mathbf{x}$ and training data $\mathbf{x}_i$, and $\mathbf{Y}$ is training labels.

The kernel regression solution f_{NTK} is uniquely determined because the neural tangent ker-

nel $\text{NTK}_{\text{train}}$ is positive definite assuming no two training data are parallel, which can be enforced with a bias term [Du et al. \(2019a\)](#). In any case, the solution is a min-norm by pseudo-inverse.

Moreover, a unique kernel regression solution f_{NTK} that spans the training data features corresponds to a unique representation in the RKHS space β_0 .

Since β_0 and β_1 are orthogonal, we also have the following

$$\|\beta_{\text{NTK}}\|_2^2 = \|\beta_0 + \beta_1\|_2^2 = \|\beta_0\|_2^2 + \|\beta_1\|_2^2.$$

This implies the norm of β_{NTK} is at least as large as the norm of any β such that $\phi(\mathbf{x}_i)^\top \beta_{\text{NTK}} = y_i$. Moreover, observe that the solution to kernel regression Eqn. 11 is in the feature span of training data, given the kernel matrix for training data is full rank.

$$f_{\text{NTK}}(\mathbf{x}) = (\langle \phi(\mathbf{x}), \phi(\mathbf{x}_1) \rangle, \dots, \langle \phi(\mathbf{x}), \phi(\mathbf{x}_n) \rangle) \cdot \text{NTK}_{\text{train}}^{-1} \mathbf{Y}.$$

Since β_1 is for the component of f_{NTK} in the orthogonal complement of training data feature span, we must have $\beta_1 = \mathbf{0}$. It follows that β_{NTK} is equivalent to

$$\begin{aligned} & \min_{\beta} \|\beta\|_2 \\ & \text{s.t. } \phi(\mathbf{x}_i)^\top \beta = y_i, \quad \text{for } i = 1, \dots, n. \end{aligned}$$

as desired.

7.4.9 Proof of Lemma 7.3.2

We first compute the neural tangent kernel $\text{NTK}(\mathbf{x}, \mathbf{x}')$ for a two-layer multi-layer perceptron (MLP) with ReLU activation function, and then show that it can be induced by the feature space $\phi(\mathbf{x})$ specified in the lemma so that $\text{NTK}(\mathbf{x}, \mathbf{x}') = \langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle$.

Recall that Jacot et al. [Jacot et al. \(2018\)](#) have derived the general framework for computing the neural tangent kernel of a neural network with general architecture and activation function. This framework is also described in [Arora et al. \(2019b\)](#); [Du et al. \(2019c\)](#), which, in addition, compute the exact kernel formula for convolutional networks and Graph Neural Networks, respectively. Following the framework in [Jacot et al. \(2018\)](#) and substituting the general activation function σ with ReLU gives the kernel formula for a two-layer MLP with ReLU activation. This has also been described in several previous works [Du et al. \(2019a\)](#); [Bietti and Mairal \(2019\)](#); [Chizat et al. \(2019\)](#).

Below we describe the general framework in [Jacot et al. \(2018\)](#) and [Arora et al. \(2019b\)](#). Let σ denote the activation function. The neural tangent kernel for an h -layer multi-layer perceptron can be recursively defined via a dynamic programming process. Here, $\Sigma^{(i)} : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$ for $i = 0 \dots h$ is the covariance for the i -th layer.

$$\begin{aligned}\Sigma^{(0)}(\mathbf{x}, \mathbf{x}') &= \mathbf{x}^\top \mathbf{x}', \\ \Lambda^{(i)}(\mathbf{x}, \mathbf{x}') &= \begin{pmatrix} \Sigma^{(i-1)}(\mathbf{x}, \mathbf{x}) & \Sigma^{(i-1)}(\mathbf{x}, \mathbf{x}') \\ \Sigma^{(i-1)}(\mathbf{x}', \mathbf{x}) & \Sigma^{(i-1)}(\mathbf{x}', \mathbf{x}') \end{pmatrix}, \\ \Sigma^{(i)}(\mathbf{x}, \mathbf{x}') &= c \cdot \mathbb{E}_{u, v \sim \mathcal{N}(\mathbf{0}, \Lambda^{(i)})} [\sigma(u) \sigma(v)].\end{aligned}$$

The derivative covariance is defined similarly:

$$\dot{\Sigma}^{(i)}(\mathbf{x}, \mathbf{x}') = c \cdot \mathbb{E}_{u, v \sim \mathcal{N}(\mathbf{0}, \Lambda^{(i)})} [\dot{\sigma}(u) \dot{\sigma}(v)].$$

Then the neural tangent kernel for an h -layer network is defined as

$$\text{NTK}^{(h-1)}(\mathbf{x}, \mathbf{x}') = \sum_{i=1}^h \left(\Sigma^{(i-1)}(\mathbf{x}, \mathbf{x}') \cdot \prod_{k=i}^h \dot{\Sigma}^{(k)}(\mathbf{x}, \mathbf{x}') \right),$$

where we let $\dot{\Sigma}^{(h)}(\mathbf{x}, \mathbf{x}') = 1$ for the convenience of notations.

We compute the explicit NTK formula for a two-layer MLP with ReLU activation function by following this framework and substituting the general activation function with ReLU, i.e. $\sigma(a) = \max(0, a) = a \cdot \mathbb{I}(a \geq 0)$ and $\dot{\sigma}(a) = \mathbb{I}(a \geq 0)$.

$$\begin{aligned} \text{NTK}^{(1)}(\mathbf{x}, \mathbf{x}') &= \sum_{i=1}^2 \left(\Sigma^{(i-1)}(\mathbf{x}, \mathbf{x}') \cdot \prod_{k=i}^h \dot{\Sigma}^{(k)}(\mathbf{x}, \mathbf{x}') \right) \\ &= \Sigma^{(0)}(\mathbf{x}, \mathbf{x}') \cdot \dot{\Sigma}^{(1)}(\mathbf{x}, \mathbf{x}') + \Sigma^{(1)}(\mathbf{x}, \mathbf{x}') \end{aligned}$$

So we can get the NTK via $\Sigma^{(1)}(\mathbf{x}, \mathbf{x}')$ and $\dot{\Sigma}^{(1)}(\mathbf{x}, \mathbf{x}')$, $\Sigma^{(0)}(\mathbf{x}, \mathbf{x}')$. Precisely,

$$\begin{aligned} \Sigma^{(0)}(\mathbf{x}, \mathbf{x}') &= \mathbf{x}^\top \mathbf{x}', \\ \Lambda^{(1)}(\mathbf{x}, \mathbf{x}') &= \begin{pmatrix} \mathbf{x}^\top \mathbf{x} & \mathbf{x}^\top \mathbf{x}' \\ \mathbf{x}'^\top \mathbf{x} & \mathbf{x}'^\top \mathbf{x}' \end{pmatrix} = \begin{pmatrix} \mathbf{x} \\ \mathbf{x}' \end{pmatrix} \cdot \begin{pmatrix} \mathbf{x} & \mathbf{x}' \end{pmatrix}, \\ \Sigma^{(1)}(\mathbf{x}, \mathbf{x}') &= c \cdot \mathbb{E}_{u, v \sim \mathcal{N}(\mathbf{0}, \Lambda^{(1)})} [u \cdot \mathbb{I}(u \geq 0) \cdot v \cdot \mathbb{I}(v \geq 0)]. \end{aligned}$$

To sample from $\mathcal{N}(\mathbf{0}, \Lambda^{(1)})$, we let L be a decomposition of $\Lambda^{(1)}$, such that $\Lambda^{(1)} = LL^\top$. Here, we can see that $L = (\mathbf{x}, \mathbf{x}')^\top$. Thus, sampling from $\mathcal{N}(\mathbf{0}, \Lambda^{(1)})$ is equivalent to first sampling $\mathbf{w} \sim \mathcal{N}(\mathbf{0}, I)$, and output

$$L\mathbf{w} = \mathbf{w}^\top (\mathbf{x}, \mathbf{x}').$$

Then we have the equivalent sampling $(u, v) = (\mathbf{w}^\top \mathbf{x}, \mathbf{w}^\top \mathbf{x}')$. It follows that

$$\Sigma^{(1)}(\mathbf{x}, \mathbf{x}') = c \cdot \mathbb{E}_{\mathbf{w} \sim \mathcal{N}(\mathbf{0}, I)} [\mathbf{w}^\top \mathbf{x} \cdot \mathbb{I}(\mathbf{w}^\top \mathbf{x} \geq 0) \cdot \mathbf{w}^\top \mathbf{x}' \cdot \mathbb{I}(\mathbf{w}^\top \mathbf{x}' \geq 0)]$$

It follows from the same reasoning that

$$\dot{\Sigma}^{(1)}(\mathbf{x}, \mathbf{x}') = c \cdot \mathbb{E}_{\mathbf{w} \sim \mathcal{N}(\mathbf{0}, I)} [\mathbb{I}(\mathbf{w}^\top \mathbf{x} \geq 0) \cdot \mathbb{I}(\mathbf{w}^\top \mathbf{x}' \geq 0)].$$

The neural tangent kernel for a two-layer MLP with ReLU activation is then

$$\begin{aligned} \text{NTK}^{(1)}(\mathbf{x}, \mathbf{x}') &= \Sigma^{(0)}(\mathbf{x}, \mathbf{x}') \cdot \dot{\Sigma}^{(1)}(\mathbf{x}, \mathbf{x}') + \Sigma^{(1)}(\mathbf{x}, \mathbf{x}') \\ &= c \cdot \mathbb{E}_{\mathbf{w} \sim \mathcal{N}(\mathbf{0}, I)} [\mathbf{x}^\top \mathbf{x}' \cdot \mathbb{I}(\mathbf{w}^\top \mathbf{x} \geq 0) \cdot \mathbb{I}(\mathbf{w}^\top \mathbf{x}' \geq 0)] \\ &\quad + c \cdot \mathbb{E}_{\mathbf{w} \sim \mathcal{N}(\mathbf{0}, I)} [\mathbf{w}^\top \mathbf{x} \cdot \mathbb{I}(\mathbf{w}^\top \mathbf{x} \geq 0) \cdot \mathbf{w}^\top \mathbf{x}' \cdot \mathbb{I}(\mathbf{w}^\top \mathbf{x}' \geq 0)]. \end{aligned}$$

Next, we use the kernel formula to compute a feature map for a two-layer MLP with ReLU activation function. Recall that by definition a valid feature map must satisfy the following condition

$$\text{NTK}^{(1)}(\mathbf{x}, \mathbf{x}') = \langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle$$

It is easy to see that the way we represent our NTK formula makes it easy to find such a decomposition. The following infinite-dimensional feature map would satisfy the requirement because the inner product of $\phi(\mathbf{x})$ and $\phi(\mathbf{x}')$ for any $\mathbf{x}, \mathbf{x}'$ would be equivalent to the expected value in NTK, after we integrate with respect to the density function of $\mathbf{w}$.

$$\phi(\mathbf{x}) = c' \left(\mathbf{x} \cdot \mathbb{I} \left(\mathbf{w}^{(k)\top} \mathbf{x} \geq 0 \right), \mathbf{w}^{(k)\top} \mathbf{x} \cdot \mathbb{I} \left(\mathbf{w}^{(k)\top} \mathbf{x} \geq 0 \right), \dots \right),$$

where $\mathbf{w}^{(k)} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, with k going to infinity. c' is a constant, and $\mathbb{I}$ is the indicator function.

Note that here the density of features of $\phi(\mathbf{x})$ is determined by the density of $\mathbf{w}$, i.e. Gaussian.

Appendix 8: Appendix: How does the Architecture Impact a Neural Network’s Robustness?

8.1 Additional Experimental Results

In this section, we include additional experimental results for the predictive power in (a) representations from randomly initialized models (Appendix 8.1.1), (b) representations learned under different types off additive label noise (Appendix 8.1.2) and (c) representations learned with a robust loss function (Appendix 8.1.3). We further demonstrates that the predictive power in well-aligned networks could even outperform sophisticated methods that also utilize clean labels (Appendix 8.1.4).

8.1.1 Predictive Power of Randomly Initialized Models

We first evaluate the predictive power of randomly initialized models (a.k.a., untrained models), and we compare their results with GNNs trained on clean data (a.k.a., 0% noise ratio). Table 8.1: Predictive power in representations from random and trained max-sum GNNs on the maximum degree task (Section 4.3.2). Notice that lower test MAPE denotes better test performance.

Model	Test MAPE	
	Random	Trained
Max-sum GNN	12.74 ± 0.57	0.37 ± 0.08

Table 8.2: Predictive power in representations from various types of random and trained GNNs on the maximum node feature task (Section 4.3.3). Notice that lower test MAPE denotes better test performance.

Model	Test MAPE		Test MAPE (log scale)	
	Random	Trained	Random	Trained
DeepSet	5.14e-05	1.06e-05	-4.29	-4.97
Max-max GNN	0.794	0.0099	-0.10	-2.00
Max-sum GNN	54.28	3.08	1.73	0.488

8.1.2 Additive Label Noise on Graph Algorithmic Datasets

We conduct additional experiments on additive label noise drawn from distributions with larger mean and larger variance. We consider four such distributions: Gaussian distributions $\mathcal{N}(10, 30)$ and $\mathcal{N}(20, 15)$, a long-tailed Gamma distribution with mean equal to 10: $\Gamma(2, \frac{1}{15}) - 20$, and another long-tailed t-distribution with mean equal to 10: $\mathcal{T}(\nu = 1) + 10$. Figure 8.1 demonstrates that for a GNN well aligned to the target function, its representations are still very predictive even under non-zero mean distributions with larger mean and large variance.

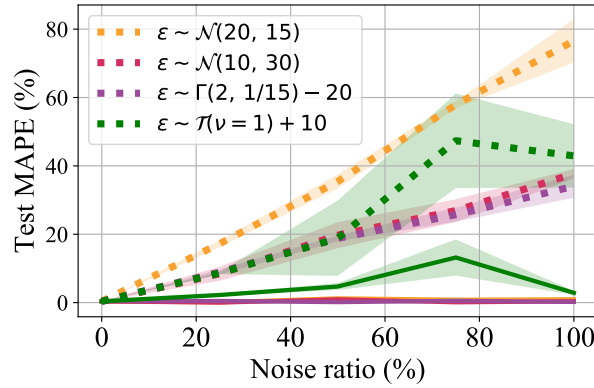
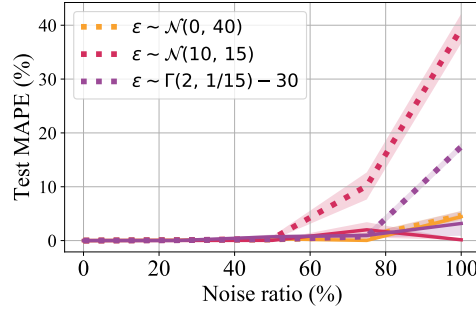


Figure 8.1: **Additional experiments on simple noise drawn from non-zero mean distributions.** On the maximum degree task, max-sum GNNs have large test errors (dotted lines) under additive label noise drawn from non-zero-mean distributions. Yet, the predictive power in representations (solid lines) greatly reduces the test errors with 10% clean labels.

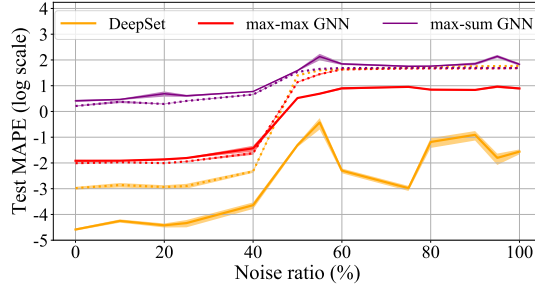
8.1.3 Training with a Robust Loss Function

We also train the models with a robust loss function—Mean Absolute Error (MAE), and we observe similar trends in the representations’ predictive power as training the models using MSE (Figure 8.2).

$$\text{loss} = \sum_{i=1}^n |y_{\text{true}} - y_{\text{pred}}|. \quad (1)$$



(a) Test errors of max-sum GNNs on the maximum degree task with additive label noise



(b) Test errors of three different GNNs on the maximum node feature task with instance-dependent label noise

Figure 8.2: **Predictive power in representations trained with MAE.** For GNNs trained with MAE, the predictive power in representations exhibits similar trends as models trained with MSE. The robust loss function, MAE, is more helpful in learning more predictive representations under smaller noise ratios.

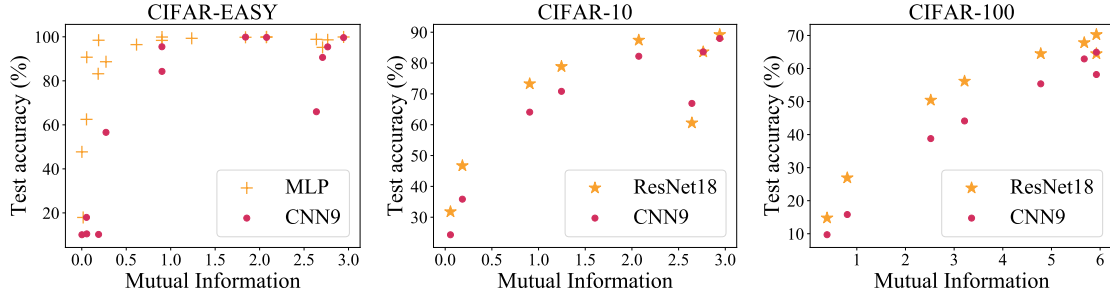


Figure 8.3: **The predictive power in the representations grows as the mutual information between the noisy labels and original clean labels increases** for models well-aligned with the target function. The x-axis of each point in the plots denotes the mutual information between a given noisy dataset and the original clean labels. The corresponding y-axis denotes the representations’ predictive power for a model trained on this noisy dataset with standard procedures (i.e., vanilla training).

8.1.4 Comparing with Sophisticated Methods Using Clean Labels

Table 8.3: Test accuracy (%) on **CIFAR-10 with uniform label noise**.

Method	# clean per class	Noise ratio					
		0%	20%	40%	50%	80%	90%
Cross-Entropy	-	95.0 ± 0.1	84.4 ± 0.4	67.9 ± 1.1	58.5 ± 1.5	27.3 ± 0.4	17.2 ± 0.5
IEG Zhang et al. (2020b)	10	94.4	92.9 ± 0.2	92.5 ± 0.5	-	85.6 ± 1.1	-
L2R Ren et al. (2018)	100	96.1	90.0 ± 0.4	86.9 ± 0.2	-	73.0 ± 0.8	-
MentorNet Jiang et al. (2018)	500	96.0	92.0	89.0	-	49.0	-
DivideMix Li et al. (2020b)	-	95.0 ± 0.1	95.7	94.4 ± 0.1	94.4	92.9	75.4
DivideMix w/ Clean Labels (DwC)	500	95.0 ± 0.1	95.9 ± 0.1	94.3 ± 0.1	94.8 ± 0.1	92.9 ± 0.2	81.5 ± 0.4
DivideMix’s Predictive Power	10	95.0 ± 0.1	96.0 ± 0.1	94.5 ± 0.1	94.7 ± 0.0	93.2 ± 0.1	74.6 ± 0.4
DivideMix’s Predictive Power	100	95.0 ± 0.1	96.1 ± 0.1	94.5 ± 0.1	94.8 ± 0.1	93.4 ± 0.1	76.7 ± 0.4
DivideMix’s Predictive Power	500	95.0 ± 0.1	96.1 ± 0.1	94.6 ± 0.1	94.9 ± 0.1	93.6 ± 0.1	80.4 ± 0.4
DwC’s Predictive Power	500	95.0 ± 0.1	95.9 ± 0.1	94.7 ± 0.1	95.0 ± 0.1	93.5 ± 0.1	87.4 ± 0.2

In previous experiments (section 4.4.2), we have shown that the predictive power in well-aligned models could further improve the test performance of SOTA methods on noisy label training. As we use a small set of clean labels to measure the predictive power, we also wonder how the improvements obtained by the predictive power compare with the sophisticated methods that also use clean labels.

Table 8.4: Test accuracy (%) on **CIFAR-10 with flipped label noise**.

Method	# clean per class	Noise ratio		
		20%	40%	80%
Cross-Entropy	-	86.1 $\pm$ 0.5	76.9 $\pm$ 1.0	54.7 $\pm$ 0.7
IEG Zhang et al. (2020b)	10	92.7 $\pm$ 0.2	90.2 $\pm$ 0.5	78.9 $\pm$ 3.5
SELF Nguyen et al. (2019)	100	92.8	89.1	-
GLC Hendrycks et al. (2018)	100	89.7 $\pm$ 0.3	88.9 $\pm$ 0.2	-
Meta-Weight-Net Shu et al. (2019)	100	90.3 $\pm$ 0.6	87.5 $\pm$ 0.2	-
DivideMix Li et al. (2020b)	-	94.0 $\pm$ 0.3	92.1	56.2 $\pm$ 0.1
DivideMix w/ Clean Labels (DwC)	500	94.2 $\pm$ 0.2	91.7 $\pm$ 0.3	56.9 $\pm$ 0.4
DivideMix’s Predictive Power	10	93.68 $\pm$ 0.34	92.14 $\pm$ 0.50	88.71 $\pm$ 0.46
DivideMix’s Predictive Power	100	94.63 $\pm$ 0.09	93.59 $\pm$ 0.12	92.68 $\pm$ 0.11
DivideMix’s Predictive Power	500	95.00 $\pm$ 0.10	94.25 $\pm$ 0.11	93.87 $\pm$ 0.09
DwC’s Predictive Power	500	94.89 $\pm$ 0.11	93.53 $\pm$ 0.12	92.84 $\pm$ 0.12

Table 8.5: Test accuracy (%) on **CIFAR-100 with uniform label noise**.

Method	# clean per class	Noise ratio					
		0%	20%	40%	50%	80%	90%
Cross-Entropy	-	77.1 $\pm$ 0.1	63.2 $\pm$ 0.2	49.8 $\pm$ 0.3	40.2 $\pm$ 0.2	11.5 $\pm$ 0.1	3.9 $\pm$ 0.1
IEG Zhang et al. (2020b)	10	72.1	69.3 $\pm$ 0.5	67.0 $\pm$ 0.8	-	60.7 $\pm$ 1.0	-
L2R Ren et al. (2018)	10	81.2	67.1 $\pm$ 0.1	61.3 $\pm$ 2.0	-	35.1 $\pm$ 1.2	-
MentorNet Jiang et al. (2018)	50	79.0	73.0	68.0	-	35.0	-
DivideMix Li et al. (2020b)	-	77.1 $\pm$ 0.1	76.9	74.8 $\pm$ 0.2	74.2	59.6	31.0
DivideMix w/ Clean Labels (DwC)	50	77.1 $\pm$ 0.1	76.8 $\pm$ 0.2	75.0 $\pm$ 0.2	74.0 $\pm$ 0.2	60.4 $\pm$ 0.2	39.8 $\pm$ 0.1
DivideMix’s Predictive Power	10	77.1 $\pm$ 0.1	76.3 $\pm$ 0.2	74.0 $\pm$ 0.1	73.6 $\pm$ 0.2	58.5 $\pm$ 0.2	32.6 $\pm$ 0.4
DivideMix’s Predictive Power	50	77.1 $\pm$ 0.1	77.2 $\pm$ 0.2	75.1 $\pm$ 0.1	74.7 $\pm$ 0.2	61.1 $\pm$ 0.1	37.6 $\pm$ 0.3
DwC’s Predictive Power	50	77.1 $\pm$ 0.1	76.4 $\pm$ 0.2	74.6 $\pm$ 0.1	73.7 $\pm$ 0.2	61.5 $\pm$ 0.1	45.1 $\pm$ 0.2

Sophisticated Methods Using Clean Labels In our experiments, we consider the following methods which use clean labels: L2R Ren et al. (2018), MentorNet Jiang et al. (2018), SELF Nguyen et al. (2019), GLC Hendrycks et al. (2018), Meta-Weight-Net Shu et al. (2019), and IEG Zhang et al. (2020b). Besides, as the SOTA method, DivideMix, keeps dividing the training data into labeled and unlabeled sets during training, we also compare with directly using clean labels in DivideMix: we mark the small set of clean data as labeled data during the semi-supervised learning step in DivideMix. We denote this method as *DivideMix w/ Clean Labels (DwC)* and further

Table 8.6: Test accuracy (%) on **CIFAR-100 with flipped label noise**.

Method	# clean per class	Noise ratio		
		20%	40%	80%
Cross-Entropy	-	63.6 ± 0.5	45.2 ± 0.3	7.4 ± 0.2
GLC Hendrycks et al. (2018)	10	63.1 ± 0.5	62.2 ± 0.6	-
Meta-Weight-Net Shu et al. (2019)	10	64.2 ± 0.3	58.6 ± 0.5	-
DivideMix Li et al. (2020b)	-	77.0 ± 0.2	55.2 ± 0.7	0.2 ± 0.0
DivideMix w/ Clean Labels (DwC)	50	76.9 ± 0.2	55.4 ± 0.8	0.2 ± 0.0
DivideMix’s Predictive Power	10	74.31 ± 0.16	72.09 ± 0.24	73.75 ± 0.31
DivideMix’s Predictive Power	50	76.74 ± 0.18	74.91 ± 0.22	76.13 ± 0.21
DwC’s Predictive Power	50	76.35 ± 0.18	74.46 ± 0.23	75.55 ± 0.22

Table 8.7: Comparison with state-of-the-art methods in test accuracy (%) on Clothing1M.

Method	# clean	Test Accuracy
Cross-Entropy	-	69.21
DivideMix Li et al. (2020b)	-	74.76
IEG Zhang et al. (2020b)	50k	77.21
CleanNet Lee et al. (2018)	50k	79.9
F-correction Patrini et al. (2017)	50k	80.38
Self-learning Han et al. (2019)	50k	81.16
DivideMix+Ours	50k	80.47

measure the predictive power in representations learned by DwC.

Datasets We conduct experiments on CIFAR-10/100 with synthetic noisy labels and on two large-scale datasets with real-world noisy labels: Clothing1M and Webvision.

Clothing1M [Xiao et al. \(2015\)](#) has real-world noisy labels with an estimated 38.5% noise ratio. The dataset has a small human-verified training data, which we use as clean data. Following recent method [Li et al. \(2020b\)](#), we use 1000 mini-batches in each epoch to train models on Clothing1M.

WebVision [Li et al. \(2017\)](#) also has real-world noisy labels with an estimated 20% noise

Table 8.8: Comparison with state-of-the-art methods trained on (mini) WebVision dataset. Numbers denote top-1 (top-5) accuracy (%) on the WebVision and the ImageNet validation sets.

Method	WebVision		ILSVRC12	
	top1	top5	top1	top5
MentorNet Jiang et al. (2018)	63.00	81.40	57.80	79.92
IEG Zhang et al. (2020b)	-	-	80.0	94.9
DivideMix Li et al. (2020b)	77.32	91.64	75.20	90.84
DivideMix+Ours	77.70 $\pm$ 0.23	90.68	75.99 $\pm$ 0.09	91.30

ratio. It shares the same 1000 classes as ImageNet [Deng et al. \(2009\)](#). For a fair comparison, we follow [Jiang et al. \(2018\)](#) to create a mini WebVision dataset with the top 50 classes from the Google image subset of WebVision. We train all models on mini WebVision dataset and evaluate on both the WebVision and ImageNet validation sets. We select 100 images per class from ImageNet training data as clean data.

Experimental Settings We use the same architectures and hyperparameters as DivideMix: an 18-layer PreAct Resnet [He et al. \(2016\)](#) for CIFAR-10/100, a ResNet-50 pre-trained on ImageNet for Clothing1M, and Inception-ResNet-V2 [Szegedy et al. \(2016\)](#) for WebVision. We use the test accuracy reported in the original papers whenever possible, and the accuracy for L2R [Ren et al. \(2018\)](#) are from [Zhang et al. \(2020b\)](#). For IEG, we use the reported test accuracy obtained by ResNet-29 rather than WRN28-10, because ResNet-29 has a comparable number of parameters as the PreAct ResNet-18 we use.

As CIFAR-10/100 do not have a validation set, we follow previous works to report the averaged test accuracy over the last 10 epochs: we measure the predictive power in representations for models from these epochs and report the averaged test accuracy. For Clothing1M and Webvision, we use the associated validation set to select the best model and measure the predictive

power in its representations.

Results Tables 8.3-8.6 show the results on CIFAR-10 and CIFAR-100 with uniform and flipped label noise, where **boldfaced numbers** denote test accuracies better than all methods we compared with. We see that across different noise ratios on CIFAR-10/100 with flipped label noise, the predictive power in representations remains roughly the same as the test performance of the model trained on clean data for a network well-aligned with the target function, which matches with Lemma 8.2.2. For CIFAR-10 with uniform label noise, the predictive power in representations achieves better test performance using only 10 clean labels per class on most noise ratios; for CIFAR-100 with uniform label noise, the predictive power in representations could achieve better test performance using only 50 labels per class.

Moreover, we observe that adding clean data to the labeled set in DivideMix (DwC) may barely improve the model’s test performance when the noise ratio is small and under flipped label noise. At 90% uniform label noise, DwC can greatly improve the model’s test performance, and the predictive power in representations can achieve a even higher test accuracy with the same set of clean data used to train DwC.

On Clothing1M, we compare the predictive power in representations learned by DivideMix with existing methods that use the small set of human-verified data: CleanNet [Lee et al. \(2018\)](#), F-correction [Patrini et al. \(2017\)](#) and Self-learning [Han et al. \(2019\)](#). As these methods also use the clean subset to fine-tune the whole model, we follow similar procedures to fine-tune the model (trained by DivideMix) for 10 epochs and then select the best model based on the validation accuracy to measure the predictive power in its representations. The predictive power in representations could further improve the test accuracy of DivideMix by around 6% and outper-

form IEG, CleanNet, and F-correction (Table 8.7). The improved test accuracy is also competitive to Han et al. (2019), which uses a much more complicated learning framework.

On Webvision, the predictive power also improves the model’s test performance (Table 8.8). The improvement is less significant than on Clothing1M as the estimated noise ratio on Webvision (20%) is smaller than Clothing1M (38.5%).

8.2 Theoretical Results

We first provide a formal version of Theorem 4.2.3 based on Xu et al. (2020). Theorem 4.2.3 connects a network’s architectural alignment with the target function to its learned representations’ predictive power *when trained on clean data*.

Theorem 8.2.1. *(Better alignment implies better predictive power on clean training data; Xu et al. (2020)).* Fix ϵ and δ . Given a target function $f : \mathcal{X} \rightarrow \mathcal{Y}$ that can be decomposed into functions $f_1, \dots, f_n$ and given a network $\mathcal{N}$, where $\mathcal{N}_1, \dots, \mathcal{N}_n$ are $\mathcal{N}$ ’s modules in sequential order, suppose the training dataset $\mathcal{S} := \{\mathbf{x}_j, y_j\}_{j=1}^M$ contains M i.i.d. samples drawn from a distribution with clean labels $y_j := f(\mathbf{x}_j)$. Then under the following assumptions, $\text{Alignment}(\mathcal{N}, f, \epsilon, \delta) \leq M$ if and only if there exists a learning algorithm A such that the network’s last module $\mathcal{N}_n$ ’s representations learned by A on the training data $\mathcal{S}$ have predictive power $P_n(f, \mathcal{N}, \mathcal{S}) \leq \epsilon$ with probability $1 - \delta$.

Assumptions:

(a) *We train each module $\mathcal{N}_i$ ’s sequentially: for each $\mathcal{N}_i$, the input samples are $\{h^{(i-1)}(\mathbf{x}_j), f_i(h^{(i-1)}(\mathbf{x}_j))\}_{j=1}^M$ with $h^{(0)}(\mathbf{x}) = \mathbf{x}$. Notice that each input $h^{(i-1)}(\mathbf{x}_j)$ is the output from the previous modules, but its label is generated by the function f_i on $h^{(i-1)}(\mathbf{x}_j)$.*

- (b) For the clean training set $\mathcal{S}$, let $\mathcal{S}' := \{\hat{\mathbf{x}}_j, y_j\}_{j=1}^M$ denote the perturbed training data ($\hat{\mathbf{x}}_j$ and $\mathbf{x}_j$ share the same label y_j). Let $f_{\mathcal{N},A}$ and $f'_{\mathcal{N},A}$ denote the functions obtained by the learning algorithm A operating on $\mathcal{S}$ and $\mathcal{S}'$ respectively. Then for any $\mathbf{x} \in \mathcal{X}$, $\|f_{\mathcal{N},A}(\mathbf{x}) - f'_{\mathcal{N},A}(\mathbf{x})\| \leq L_0 \cdot \max_{\mathbf{x}_j \in \mathcal{S}} \|\mathbf{x}_j - \hat{\mathbf{x}}_j\|$, for some constant L_0 .
- (c) For each module $\mathcal{N}_i$, let $\hat{f}_i$ denotes its corresponding function learned by the algorithm A . Then for any $\mathbf{x}, \hat{\mathbf{x}} \in \mathcal{X}$, $\|\hat{f}_i(\mathbf{x}) - \hat{f}_i(\hat{\mathbf{x}})\| \leq L_1 \|\mathbf{x} - \hat{\mathbf{x}}\|$, for some constant L_1 .

We have empirically shown that Theorem 8.2.1 also hold when we train the models on noisy data. Meanwhile, we prove Theorem 8.2.1 for a simplified noisy setting where the target function and noise function share a common feature space, but have different prediction rules. For example, the target function and noise function share the same feature space under flipped label noise (in classification setting). Yet, their mappings from the learned features to the associated labels are different.

Theorem 8.2.2. (Better alignment implies better predictive power on noisy training data). Fix ϵ and δ . Let $\{\mathbf{x}_j\}_{j=1}^M$ be i.i.d. samples drawn from a distribution. Given a target function $f : \mathcal{X} \rightarrow \mathcal{Y}$ and a noise function $g : \mathcal{X} \rightarrow \mathcal{Y}$, let $y := f(\mathbf{x})$ denote the true label for an input $\mathbf{x}$, and $\hat{y} := g(\mathbf{x})$ denote the noisy label of $\mathbf{x}$. Let $\hat{\mathcal{S}} := \{(\mathbf{x}_j, y_j)\}_{j=1}^N \cup \{(\mathbf{x}_j, \hat{y}_j)\}_{j=N+1}^M$ denote a noisy training set with $M - N$ noisy samples for some $N \in \{1, 2, \dots, M\}$. Given a network $\mathcal{N}$ with modules $\mathcal{N}_i$, suppose $\mathcal{N}$ is well-aligned with the target function f (i.e., the alignment between $\mathcal{N}$ and f is less than $M - \text{Alignment}(\mathcal{N}, f, \epsilon, \delta) \leq M$). Then under the same assumptions in Theorem 8.2.1 and the additional assumptions below, there exists a learning algorithm A and a module $\mathcal{N}_i$ such that when training the network $\mathcal{N}$ on the noisy data $\hat{\mathcal{S}}$ with algorithm A , the representations from its i -th module have predictive power $P_i(f, \mathcal{N}, \mathcal{C}) \leq \epsilon$

with probability $1 - \delta$, where $\mathcal{C}$ is a small set of clean data with a size greater than the number of dimensions in the output of module $\mathcal{N}_i$.

Additional assumptions (a simplified noisy setting):

- (a) *There exists a function h on the input domain $\mathcal{X}$ such that the target function $f : \mathcal{X} \rightarrow \mathcal{Y}$ and the noise function $g : \mathcal{X} \rightarrow \mathcal{Y}$ can be decomposed as: $f(\mathbf{x}) = f_r(h(\mathbf{x}))$ with f_r being a linear function and $g(\mathbf{x}) = g_r(h(\mathbf{x}))$ for some function g_r .*
- (b) *f_r is a linear map from a high-dimensional space to a low-dimensional space.*
- (c) *The loss function used in measuring the predictive power is mean squared error (denoted as $\|\cdot\|$).*

Remark. Theorem 8.2.2 suggests that the representations' predictive power for models well aligned with the target function should remain roughly similar across different noise ratios under flipped label noise. Empirically, we observe similar phenomena in Figures 4.7-4.8, and in Tables 8.4 and 8.6. Some discrepancy between the experimental and theoretical results could exist under vanilla training as Theorem 8.2.2 assumes sequential training, which is different from standard training procedures.

Proof of Theorem 8.2.2. According to the definition of alignment in Definition 4.2.2, since $\text{Alignment}(\mathcal{N}, f, \epsilon, \delta) \leq M$ and $f(\mathbf{x}) = f_r(h(\mathbf{x}))$, we can find a sub-structure (denoted as $\mathcal{N}_{sub}$) in the network $\mathcal{N}$ with sequential modules $\{\mathcal{N}_1, \dots, \mathcal{N}_i\}$ such that $\mathcal{N}_{sub}$ can efficiently learn the function h (i.e., the sample complexity for $\mathcal{N}_{sub}$ to learn h is no larger than M). According to Theorem 8.2.1, applying sequential learning to train $\mathcal{N}_{sub}$ with labels $h(\mathbf{x})$, the representations of $\mathcal{N}_{sub}$ will have predictive power $P_i(h, \mathcal{N}_{sub}, \mathcal{C}) \leq \epsilon$ with probability $1 - \delta$.

Since for each input $\mathbf{x}$ in the noisy training data $\hat{\mathcal{S}}$, its label can be written as $f_r(h(\mathbf{x}))$

(if it is clean) or $g_r(h(\mathbf{x}))$ (if it is noisy), when the network $\mathcal{N}$ is trained on $\hat{\mathcal{S}}$ using sequential learning, its sub-structure $\mathcal{N}_{sub}$ can still learn h efficiently (i.e., $\mathcal{M}_A(h, \mathcal{N}_{sub}, \epsilon, \delta) \leq M$ for some learning algorithm A). Thus, the representations learned from the noisy training data $\hat{\mathcal{S}}$ can still be very predictive (i.e., $P_i(h, \mathcal{N}_{sub}, \mathcal{C}) \leq \epsilon$ with probability $1 - \delta$).

Since f_r is a linear map from a high-dimensional space to a low-dimensional space, and the clean data $\mathcal{C}$ has enough samples to learn f_r ($|\mathcal{C}|$ is larger than the input dimension of f_r), the linear model L learned by linear regression can also generalize f_r (since linear regression has a closed form solution in this case as the problem is over-complete). Therefore, as $P_i(h, \mathcal{N}_{sub}, \mathcal{C}) \leq \epsilon$, $P_i(f, \mathcal{N}_{sub}, \mathcal{C}) \leq \epsilon$ also holds. Notice that $P_i(f, \mathcal{N}_{sub}, \mathcal{C}) = P_i(f, \mathcal{N}, \mathcal{C})$ as $\mathcal{N}_i$ is also the i -th module in $\mathcal{N}$. Hence, we have shown that there exist some module $\mathcal{N}_i$ such that $P_i(f, \mathcal{N}, \mathcal{C}) \leq \epsilon$ with probability $1 - \delta$.

Appendix 9: Appendix: Hindsight Learning for MDPs with Exogenous Inputs

9.1 Table of Notation

9.2 Detailed Related Work

There is extensive literature on reinforcement learning and its connection to tasks in operations management; below, we highlight the work which is closest to ours, but for more extensive references, see [Powell \(2022\)](#); [Sutton and Barto \(2018\)](#); [Agarwal et al. \(2019\)](#) for RL, and [Bubeck and Cesa-Bianchi \(2012\)](#); [Slivkins \(2019\)](#) for background on multi-armed bandits.

Information Relaxation for MDP Control: Information relaxation as an approach for calculating performance bounds on the optimal Q^* function has been developed recently using rich connections to convex duality ([Vera and Banerjee, 2021](#); [Mercier and Van Hentenryck, 2007](#); [Brown and Smith, 2022](#); [Balseiro and Brown, 2019](#); [Brown and Haugh, 2017](#); [Kallus and Zhou, 2022](#)). As discussed in the main paper, for general problems, using hindsight planning oracles as in [5.3.2](#) creates estimates for the Q^* value which are overly optimistic of their true value. These differences can be rectified by introducing a control variate, coined *information penalties*, to penalize the planner’s access to future information that a truly non-anticipatory policy would not have. The goal is to construct penalties which ensure that the estimates of Q^* are truly consistent for the underlying value. This work has been developed explicitly in the context of infinite

Table 9.1: List of common notation for MDP and HL

Symbol	Definition
Problem Setting Specification	
$\mathcal{S}, \mathcal{A}, T, s_1, R, P$	MDP primitives: state and action space, time horizon, starting state, reward function and transition probabilities
$\mathcal{X}$	Endogenous space for the system
Ξ	Exogenous input space
$\mathcal{P}_\Xi$	Distribution over exogenous inputs
$f(s, a, \xi), r(s, a, \xi)$	Underlying deterministic transition and reward as function of exogenous input
s_t	MDP state space primitive, $(x_t, \xi_{<t})$ for shorthand
s_t, a_t, ξ_t	State, action, and exogenous input for time step t
ξ	An exogenous input trace $(\xi_1, \dots, \xi_T)$
$\xi_{\geq t}$	Component of an exogenous input trace $(\xi_t, \dots, \xi_T)$
Π	Set of all admissible policies
$Q_t^\pi(s, a), V_t^\pi(s)$	Q -Function and value function for policy π at time step t
$\pi^*, Q_t^*(s, a), V_t^*(s)$	Optimal policy and the Q and value function for the optimal policy
Pr_t^π	State-visitation distribution for policy π at time step t
$\mathcal{D}$	Dataset containing N traces of exogenous inputs $\{\xi^1, \dots, \xi^N\}$
$\text{HINDSIGHT}(t, s, \xi_{\geq t}, f)$	Hindsight optimal cumulative reward r starting in state s at time t with exogenous inputs dictated by $\xi_{\geq t}$ and dynamics f (see 5)
$Q_t^\pi(s, a, \xi_{\geq t}), V_t^\pi(s, a, \xi_{\geq t})$	Q and V functions for policy π starting from s where exogenous inputs are given by $\xi_{\geq t}$ (see 5.3.1)
$\bar{\mathbb{E}}[\cdot]$	Empirical expectation taken where ξ is sampled uniformly from $\mathcal{D}$
$\bar{\pi}^*$	Policy obtained by ERM, i.e. solving $\arg \max_{\pi \in \Pi} \bar{\mathbb{E}}[V_1^\pi(s_1, \xi)]$.
$\bar{\mathcal{P}}_\Xi, \bar{Q}_t, \bar{V}_t, \bar{\pi}$	Estimated exogenous distribution using $\mathcal{D}$, Q_t^* , V_t^* estimates using the estimated distribution, and the resulting policy
Hindsight Planner	
$Q_t^+(s, a, \xi_{\geq t})$	$r(s, a, \xi_t) + \text{HINDSIGHT}(t+1, \xi_{>t}, f(s, a, \xi_t))$
$V_t^+(s, \xi_{\geq t})$	$\text{HINDSIGHT}(t, \xi_{\geq t}, s)$
$Q_t^+(s, a), V_t^+(s, a)$	Expectations of $Q_t^+(s, a, \xi_{\geq t})$ and $V_t^+(s, \xi_{\geq t})$ over $\xi_{\geq t}$
π^+	Greedy policy with respect to Q^+
$\Delta_t^+(s)$	Hindsight bias for state s at time step t (see 11)
Δ	Absolute bound on $\Delta_t^+(s)$
$\bar{\mathcal{P}}_\Xi$	Empirical distribution over ξ from $\mathcal{D}$
$\bar{\pi}^+$	Greedy policy with respect to $\bar{Q}^+$ where true expectation over $\mathcal{P}_\Xi$ replaced with $\bar{\mathcal{P}}_\Xi$
$\bar{\Delta}_t^+(s)$	Value of $\Delta_t^+(s)$ where expectation over $\mathcal{P}_\Xi$ replaced with $\bar{\mathcal{P}}_\Xi$
$\bar{\text{Pr}}_t^\pi$	State visitation distribution of π at time step t with exogenous dynamics $\bar{\mathcal{P}}_\Xi$

horizon MDPs (Brown and Haugh, 2017) where constructions are given for penalty functions as a function of the future randomness of the ξ process. Moreover, concrete algorithmic implementations using hindsight planners and information penalties has been developed in the tabular setting with no finite sample guarantees (El Shar and Jiang, 2020). Constructing these penalties in practice using suitable functions for arbitrary ξ is unknown. Our work differs by foregoing consistency of the estimates to instead focus on showing that in problem domains of interest, the policy which is greedy with respect to the hindsight planner is indeed consistent.

Behaviour Cloning: One approach for using hindsight information is behavior cloning. This will compute the hindsight-optimal actions for every $\xi \sim \mathcal{D}$, and learn to imitate these actions using a feasible non-anticipatory policy (Fang et al., 2021). This is an instance of the *probability matching* principle which is widely used in Thompson sampling (Russo et al., 2018; Hart et al., 2000). Unfortunately, this *value-agnostic* approach is uncontrollably biased. Consider the example in 5.4. Since the winds in $\mathcal{D}$ will be west 51% of the time, the hindsight-optimal distribution (marginalizing over wind) is $\Pr(\text{route1}) = 0.51; \Pr(\text{route2}) = 0.49$. A non-anticipatory learner policy trained via behavior cloning will converge either to this distribution (if learning a stochastic policy) or its mode (using a deterministic policy). Both these policies are very sub-optimal compared to the optimal policy.

Policy Based Methods with Control Variates: Recent work has developed black box tools to modify policy gradient algorithms with control variates that depend on the exogenous trace. Recall that a typical policy-based algorithm uses either on-policy data (or off-policy with re-weighted importance sampling strategies), and estimates the gradient in the return via

$$\nabla V^{\pi_\theta} = \mathbb{E}_{S \sim \Pr_t^{\pi_\theta}, A \sim \pi_\theta} [\nabla \log \pi_\theta(A \mid S) \hat{Q}^{\pi_\theta}(S, A)].$$

From here, most methods subtract an appropriate baseline (commonly taken to be an estimate of the value function) as a form of Rao-Blackwellization to reduce the variance of the estimator while incurring no additional bias. In particular, for any function $b : \mathcal{S} \rightarrow \mathbb{R}$ we can instead take

$$\nabla V^{\pi_\theta} = \mathbb{E}_{S \sim \text{Pr}_t^{\pi_\theta}, A \sim \pi_\theta} [\nabla \log \pi_\theta(A | S) (Q_{\pi_\theta}(S, A) - b(S))]$$

while remaining unbiased. However, due to the exogenous input structure on the MDP any function $b : \mathcal{X} \times \Xi^T \rightarrow \mathbb{R}$ also results in an unbiased gradient. Through this, the existing literature has taken different approaches for constructing these *input driven baselines*. In [Mao et al. \(2019b\)](#) they consider directly using a baseline of the form $b(x, \xi)$. As an architecture to learn a network representation of this baseline the authors propose either using a multi-value network or meta learning. In [Mesnard et al. \(2021\)](#) they consider using future conditional value estimates for the policy gradient baseline. In particular, they use Ψ_t as a new statistic to calculate new information from the rest of the trajectory and learn value functions which are conditioned on the additional hindsight information contained in Ψ_t . They provide a family of estimators, but do not specify which form of Ψ_t to use in generating an algorithm.

Recurrent Neural Network Policy Design: A related line of work modifies black box policy gradient methods by using a recurrent neural network (RNN) explicitly in policy design. In [Venuto et al. \(2022\)](#) they augment the state space to include ξ while simultaneously limiting information flow in the neural network to ensure that the algorithm is not overly relying on this privileged information. This approach, named *policy gradients incorporating the future*, is easy to implement as it just augments the network using an LSTM and adds a new loss term to account for the information bottleneck.

Learning to Search: The Exo-MDP model is closely related to the learning-to-search model. Expert iteration ([Anthony et al., 2017](#)) separates planning and generalization when learning to search, and provides an alternative approach to implement the HINDSIGHT oracle. Retrospective imitation ([Song et al., 2018b](#)) faces a similar challenge as us: a given ξ defines a fixed search space and we seek search policies that generalize across $\mathcal{P}_{\Xi}(\xi)$. However, retrospective imitation reduces to realizable imitation problems because the learner witnesses ξ beforehand whereas in Exo-MDPs, $\xi_{\geq t}$ is privileged information and imitating HINDSIGHT is typically unrealizable. Asymmetric Imitation Learning ([Warrington et al., 2021](#)) studies problems when imitating an expert with privileged information but essentially use RNN policies to ameliorate unrealizability.

RL for OR: In our work we primarily consider simulations on dynamic Virtual Machine (VM) scheduling. On the theoretical side, variants of greedy algorithms have been usually proposed to solve the dynamic VM scheduling problems with competitive ratio analysis. In [Stolyar \(2013\)](#) they assume the VM creation requests can be modeled as a Poisson process with lifetimes as an exponential distribution and show that the greedy algorithm achieves the asymptotically optimal policy. In [Li et al. \(2015\)](#) they develop a hybrid FIRSTFIT algorithm with an improvement on the competitive ratio. On the more practical side using deep reinforcement learning techniques, in [Mao et al. \(2016\)](#) they develop a DeepRM system which can pack tasks with multiple resource demands via a policy gradient method. They also built a job scheduler named DECIMA by modifying actor critic algorithms with input driven baselines ([Mao et al., 2019c](#)). In [Zhang et al. \(2020c\)](#) they solved the heterogeneous scheduling problem with deep Q learning. Lastly, in [Sheng et al. \(2022\)](#) they developed SchedRL, a modification of Deep Q Learning with reward shaping to develop a VM scheduling policy. All of these algorithms modify existing RL algorithms and show empirical gains on variations of the VM scheduling problem. Our work differs

from two perspectives: 1) we consider using hindsight planning explicitly during training time, 2) our algorithms can be applied to any Exo-MDP problems.

We also note that existing deep reinforcement learning has also been applied in other systems applications (without exploiting their exo-MDP structure) including ride-sharing systems (Feng et al., 2021), stochastic queueing networks (Dai and Gluzman, 2021), power grid systems (Chen et al., 2021), jitter buffers (Fang et al., 2019), and inventory control (Harsha et al., 2021).

RL for Combinatorial Optimization: A crucial assumption underpinning our algorithmic framework is the implementation of hindsight planners as in 5.3.2. Our framework is well motivated for problems where hindsight planning is efficient, building on the existing optimization literature on solving planning problems (Conforti et al., 2014; Bertsimas and Tsitsiklis, 1997). However, in general these problems as a function of a fixed exogenous input trace can be written as combinatorial optimization problems. While we consider using hindsight planners for RL problems, a dual lens is using machine learning techniques for combinatorial optimization, as has been explored in recent years (Vinyals et al., 2015; Bello et al., 2017; Chitnis and Lozano-Pérez, 2020). In particular, in Vinyals et al. (2015) they designed a new network architecture and trained using supervised learning for traveling salesman problems. Similarly in Hu et al. (2017b) they solve variants of online bin packing problems using policy gradient algorithms. In Tang et al. (2020) they design novel heuristic branch and bound algorithms using machine learning for integer programming optimization.

Exo-MDPs: Exo-MDPs, as highlighted in 5.3 were described in Powell (2022). They characterize sequential decision making problems as an evolution of *information, decision, information* sequence represented mathematically as the sequence $(s_1, a_1, \xi_1, s_2, a_2, \xi_2, \dots, s_T)$. Here, the

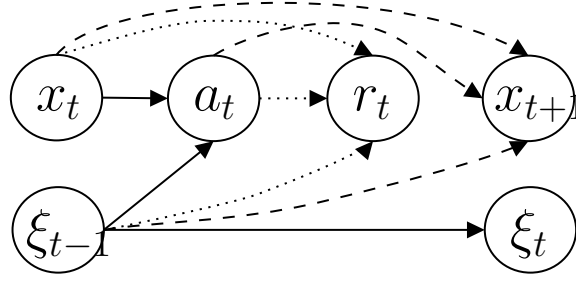


Figure 9.1: Causal diagram for an Exo-MDP where $k = 1$. Here the dotted line indicates the influence of (x_t, a_t, ξ_t) on the immediate reward r_t via $r(x_t, a_t, \xi_t)$ and the dashed line on the transition evolution as $x_{t+1} = f(s_t, a_t, \xi_t)$. The key facet to notice is the lack of influence on the ξ process from the current endogenous state x_t and action a_t .

state variable s_t is written explicitly to capture the information available to the decision maker to make a decision a_t , followed by the information we learn after making a decision, i.e. the exogenous information ξ_t . Similar models have been outlined in (Mao et al., 2019b; Dietterich et al., 2018; Efroni et al., 2022).

9.3 MDPs with Exogenous Inputs

In this section we further discuss the definition of Exo-MDPs and its relations to contextual bandits and MDPs and highlight some examples in the operations management literature.

9.3.1 Generality of MDPs with Exogenous Inputs

As highlighted in 5.3 we consider the finite horizon reinforcement learning setting where an agent is interacting with a Markov Decision Process (MDP) (Puterman, 2014). The underlying MDP is given by a five-tuple $(\mathcal{S}, \mathcal{A}, T, p, R, s_1)$ where T is the horizon, $(\mathcal{S}, \mathcal{A})$ denotes the set of states and actions, R is the reward distribution, p the distribution governing the transitions of the system, and s_1 is the given starting state.

Definition 9.3.1. *In an MDP with Exogenous Inputs (Exo-MDP) we let $\xi = (\xi_1, \dots, \xi_T)$ be a trace of exogenous inputs with each ξ_t supported on the set Ξ . We assume that ξ is sampled according to an unknown distribution $\mathcal{P}_\Xi$. The agent has access to an endogenous or system state $x \in \mathcal{X}$. With this, the dynamics and rewards of the Markov decision process evolve where at time t , the agent selects their action $a_t \in \mathcal{A}$ based solely on $s_t = (x_t, \xi_{<t})$. After, the endogenous state evolves according to $x_{t+1} = f(s_t, a_t, \xi_t)$, and the reward earned is $r(s_t, a_t, \xi_t)$, and ξ_t is observed. We assume that f and r are known by the principal in advance.*

Note that this imposes that the state space for the underlying MDP can be written as $\mathcal{S} = \mathcal{X} \times \Xi^T$ where the first component corresponds to the endogenous state and the second to the exogenous input trace observed so far. We use the shorthand s_t to refer to $(x_t, \xi_{<t})$.

As written, the distribution $\mathcal{P}_\Xi$ can be arbitrarily correlated across time. We can relax this setting to assume that ξ evolves according to a k -Markov chain. More formally, that at each step t , $\xi_t \mid (\xi_{t-k}, \xi_{t-k+1}, \dots, \xi_{t-1})$ is conditionally independent of $(\xi_1, \dots, \xi_{t-k-1})$. This allows the state space to be represented as $\mathcal{S} = \mathcal{X} \times \Xi^k$. Lastly, the dataset $\mathcal{D}$ contains a series of N traces sampled independently according to $\mathcal{P}_\Xi$ as $\mathcal{D} = \{\xi^1, \dots, \xi^N\}$ where each $\xi^i = \{\xi_1^i, \dots, \xi_T^i\}$.

For more intuition, consider the model under various values of k :

- **Case $k = T$:** Here we assume that ξ is an arbitrarily correlated process and $\mathcal{S} = \mathcal{X} \times \Xi^T$ so that $s_t = (x_t, \xi_{<t})$. An example of this is VM allocation, where exogenous VM requests are highly correlated across time ([Hadary et al., 2020](#)).
- **Case $k = 1$:** Here we assume that ξ process evolves according to a 1-Markov chain. The state space factorizes as $\mathcal{X} \times \Xi$ where $\mathcal{X}$ is the endogenous space and Ξ is the exogenous space. The current state is $s_t = (x_t, \xi_{t-1})$, and the state updates to $(f(s_t, a_t, \xi_t), \xi_t)$ where

ξ_t is drawn from the conditional distribution given ξ_{t-1} . A representation of the causal diagram under this setting is in 9.1. An example of this is bin-packing, where it is typically assumed that jobs arrive according to a Markov chain.

- **Case $k = 0$:** Here we have that $\mathcal{S} = \mathcal{X}$. After taking an action a_t based solely on x_t we transition to $x_{t+1} = f(x_t, a_t, \xi_t)$ with ξ_t sampled independently from an unknown distribution $\mathcal{P}_{\Xi}^T$. The previous variable ξ_t can be either observed or unobserved. An example of this is inventory control or newsvendor models, where the demand is typically assumed to be i.i.d. across periods.

Relation between Contextual Bandits, MDPs, and Exo-MDPs We first notice that Exo-MDPs are a bridge Between Contextual Bandit and MDPs. When $\mathcal{X}$ is empty or a singleton, an Exo-MDP describes several variants of the *contextual bandit* introduced in [Langford and Zhang \(2007\)](#). They can be solved efficiently independent of the horizon ([Foster et al., 2021](#)), unlike MDPs. If $|\Xi| \leq 1$, an Exo-MDP is simply an MDP whose complexity scales with $|\mathcal{X}|$. When both $|\mathcal{X}| > 1$ and $|\Xi| > 1$ the complexity of learning an Exo-MDP is not known in general, to the best of our knowledge. When the exogenous inputs are iid, an Exo-MDP is equivalent to an MDP with state space $\mathcal{X}$ much smaller than $\mathcal{S}$.

Difference in Dataset Assumptions We next focus briefly on the case when $k = 0$, showing that the key difference between Exo-MDPs and the typical MDP models is the assumptions on the historical dataset provided. The typical assumptions in an MDP involve that $s_{t+1} \sim P(\cdot \mid s_t, a_t)$ where the underlying distribution P is unknown. This can be written equivalently as $s_{t+1} = f(s_t, a_t, \xi_t)$ where ξ_t is sampled uniformly in $[0, 1]$ and the underlying function f is unknown. As

such, typically in an MDP we consider:

- Unknown structure of the dynamics and rewards (i.e. unknown f and r)
- Known distribution on the underlying exogenous inputs where each ξ_t is uniform over $[0, 1]$
- Access to a logged dataset of (s_t, a_t, r_t, s_{t+1}) pairs

In Exo-MDPs we instead assume:

- Known structure of the dynamics and rewards (i.e. known f and r)
- Unknown distribution on the exogenous inputs $\mathcal{P}_\Xi$
- Access to a dataset of exogenous traces $\xi_1, \dots, \xi_T$

These types of assumptions (where the *form* of the randomness is known but the true underlying distribution is unknown) is common in the graphon literature (Borgs et al., 2008; Lovász and Szegedy, 2006). In the following lemma we show that these two models are equivalent, in that any MDP can be written as an MDP with exogenous inputs and $k = 0$ using the uniform random number trick. However, the *assumptions are not equivalent* since in Exo-MDPs we assume access to a dataset of historical exogenous traces rather than trajectories under a fixed behavior policy.

Lemma 9.3.2. *Any MDP of the form $(\mathcal{S}, \mathcal{A}, T, p, R, s_1)$ where the distribution on p and R are unknown has an equivalent Exo-MDP form with $k = 0$, and vice-versa.*

Proof. Without loss of generality we will assume that both Ξ and $\mathcal{S}$ are either discrete or one dimensional (where higher dimensions follow via the same chain of reasoning).

Exo-MDP $\rightarrow$ MDP: Suppose that $s_{t+1} = f(s_t, a_t, \xi_t)$ where ξ_t is sampled from $\mathcal{P}_\Xi$ and f is known.

We can write this of the form where f is unknown and $\mathcal{P}_\Xi$ is known by setting $\tilde{\xi}_t \sim U[0, 1]$ and $\tilde{f}(s_t, a_t, \tilde{\xi}_t) = f(s_t, a_t, \mathcal{P}_\Xi^{-1}(\tilde{\xi}_t))$. Here the form of $\tilde{f}$ is unknown as we cannot evaluate $\mathcal{P}_\Xi^{-1}$, but the distribution on the underlying randomness $\tilde{\xi}$ is known.

MDP $\rightarrow$ Exo-MDP: Suppose that $s_{t+1} \sim P(\cdot \mid s_t, a_t)$ where the distribution is unknown. We can write this as $s_{t+1} = f(s_t, a_t, \xi_t)$ with a known f and unknown distribution $\mathcal{P}_\Xi$ as follows.

First set $\Xi = \Delta(\mathcal{S})^{\mathcal{S} \times \mathcal{A}} \times [0, 1]$. Given any $\xi \in \Xi$ we define the transition kernel $f(s_t, a_t, \xi)$ as follows:

- Set $\tilde{p} \in \Delta(\mathcal{S})$ to be the component of ξ indexed via s_t, a_t
- Letting z be the last component of ξ , set $s_{t+1} = \tilde{p}^{-1}(z)$.

The distribution over Ξ is then defined as an indicator variable over the first $\mathcal{S} \times \mathcal{A}$ components indicating the true unknown distribution p , and the last component over $[0, 1]$ being Uniform $[0, 1]$. □

9.3.2 Examples of Exo-MDPs

We now give several examples of Exo-MDPs alongside with their exogenous decomposition of the transition distribution. We also highlight the underlying Markovian assumption on the exogenous inputs ξ .

9.3.2.1 Inventory Control with Lead Times and Lost Sales ($k = 0$)

This models a single product stochastic inventory control problem with lost sales and lead times (Agrawal and Jia, 2022; Goldberg et al., 2021; Xin and Goldberg, 2022; Feinberg, 2016). In the beginning of every time step t , the inventory manager observes the current inventory level

Inv_t and L previous unfulfilled orders in the pipeline, denoted $o_L, \dots, o_1$ for a product. L denotes the lead time or delay in the number of time steps between placing an order and receiving it. The next inventory is obtained as follows. First, o_1 arrives and the on-hand inventory rises to $I_t = \text{Inv}_t + o_1$. Then, an exogenous demand ξ is drawn independently from the unknown demand distribution $\mathcal{P}_\Xi$. The cost to the inventory manager is

$$h(I_t - \xi)^+ + p(\xi - I_t)^+$$

where h is the holding cost for remaining inventory and p is the lost sales penalty. The on-hand inventory then finishes at level $(I_t - \xi)^+$.

This can be formulated as an Exo-MDP by letting $\mathcal{X} = [n]^{L+1}$ denote the current inventory level and previous orders, $\Xi = [n]$ as the exogenous demand, and $\mathcal{A} = [n]$ for the amount to order where n is some maximum order amount. The reward function is highlighted above, and the state transition updates as $x' = f(x_t, a_t, \xi)$ where $\text{Inv}_{t+1} = (\text{Inv}_t + o_1 - \xi)^+$, $o_k = o_{k-1}$ for all $1 < k < L$ and $o_L = a$. This model can also be expanded to include multiple suppliers with different lead times.

9.3.2.2 Online Stochastic Bin Packing ($k = 1$)

Here we consider a typical online stochastic bin packing model ([Gupta and Radovanovic, 2012](#); [Ghaderi et al., 2014](#); [Perboli et al., 2012](#)) where a principal has access to an infinite supply of bins with maximum bin size B . Items u_t arrive over a sequence of rounds $t = 1, \dots, T$ where each $u_t \in [B]$ denotes the item size. At every time step, the principal decides on a bin to allocate the item to, either allocating it to a previously opened bin or creating a new bin. The goal is to

allocate all of the items using the smallest number of bins.

This can be modeled in the framework as follows. Here we let $\mathcal{X} = \mathbb{R}^B$, $\Xi = [B]$ and $\mathcal{A} = [B]$. Each vector $x \in \mathcal{X}$ has components $x_1, \dots, x_B$ as the current number of bins opened with current utilization of one up to B , with Ξ corresponding to the current item arrival's size. Hence the state space is $\mathcal{S} = \mathcal{X} \times \Xi$ where $s_t = (x_t, \xi_{t-1})$ corresponds to the current bin capacity and current item arrival. Actions $a \in \mathcal{A}$ correspond to either 0, for opening up a new bin, or $1, \dots, B$ to be adding the current item to an existing bin with current utilization one up to B . The reward is:

$$r(x_t, \xi_{t-1}, a, \xi_t) = \begin{cases} -1 & a = 0 \\ 0 & a > 0, s_a > 0, \text{ and } a + \xi_{t-1} \leq B \\ -100 & \text{otherwise} \end{cases}$$

where -1 corresponds to the cost for opening a new bin, and the condition on zero reward verifies whether or not there is currently an open bin at level a and the action is feasible (i.e. allocating the current item to the bin at size a is smaller than the maximum bin capacity).

The transition distribution is updated similarly. Let ξ_t be drawn from the conditional distribution given ξ_{t-1} . If $a = 0$ then $x' = x$ except $x'_{\xi_{t-1}}$ is incremented by one (for opening up a new bin at the level of the size of the current item). If $a > 0$ and the action is feasible (i.e. $s_a > 0$ and $a + \xi_{t-1} \leq B$) then $x' = x$ with $x'_{a+\xi_{t-1}}$ incremented by one and x'_a decreased by one.

We note again that this model can be extended to include different reward functions, multiple dimensions of item arrivals, and departures, similar to the Virtual Machine allocation scenario.

9.3.2.3 Online Secretary ($k = 1$)

Multi-secretary is the generalization of the classic secretary problem (Buchbinder et al., 2009), where T candidates arrive sequentially but only B can be selected. Over time periods, a candidate arrives with ability $r_t \in (0, 1]$ drawn i.i.d. from a finite set of K levels of expertise. At each round, if the decision-maker has remaining budget (i.e., has chosen less than B candidates thus far), they can *accept* a candidate and collect the reward r_t , or *reject* the candidate. The goal is to maximize the expected cumulative reward.

This can be modeled as an Exo-MDP as follows. Here we let $\mathcal{X} = [B]$, $\Xi = [K]$, and $\mathcal{A} = \{0, 1\}$. The endogenous space $\mathcal{X}$ corresponds to the number of remaining candidates that can be accepted. The exogenous space Ξ corresponds to the ability level of the next time period's candidate. Lastly, actions $a \in \mathcal{A}$ correspond to either accepting or rejecting the current candidate. Hence the state space is $\mathcal{S} = \mathcal{X} \times \Xi$ where $s_t = (x_t, \xi_{t-1})$ corresponds to the number of accepted candidates thus far and the skill of the current candidate. The reward is:

$$r(x_t, \xi_{t-1}, a, \xi_t) = \begin{cases} \xi_{t-1} & a = 1 \text{ and } x_t > 0 \\ 0 & \text{otherwise} \end{cases}$$

The transition distribution is updated similarly by accounting for whether a candidate was accepted. Indeed we have:

$$f(x_t, \xi_{t-1}, a, \xi_t) = \begin{cases} x_{t-1} & a = 1 \text{ and } x_t > 0 \\ 0 & \text{otherwise} \end{cases}$$

9.3.2.4 Airline Revenue Management ($k = 0$)

Airline Revenue Management (Littlewood, 1972) is a special case of the multi-dimensional Online Bin Packing (OBP) problem, but we reiterate its model here for completeness. There are a set of $K \in \mathbb{N}$ resources, and each resource i has a maximum capacity B_i . Customers are segmented into $M \in \mathbb{N}$ types. Customers of type $j \in [M]$ request $A_j \in \mathbb{R}_+^K$ resources and yield a revenue of f_j . Over time, the algorithm will decide whether or not to accept customers of type j . Afterwards, a customer type j_t is drawn from an independent distribution. If the algorithm decided to accept customers of type j_t , the relevant resources are consumed and revenue earned. The goal of the decision-maker is to maximize the expected revenue.

This is modeled as an Exo-MDP where $\mathcal{X} = [0, B_1] \times \dots \times [0, B_K]$, $\Xi = [M]$, and $\mathcal{A} = \{0, 1\}^M$. The system space $\mathcal{X}$ corresponds to the remaining capacity of the K different resources, exogenous space Ξ to the sampled customer type, and $\mathcal{A}$ to the accept / reject decisions for each of the customer types. The reward is then defined via:

$$r(x_t, a, \xi_t) = \begin{cases} f_{\xi_t} & a_{\xi_t} = 1 \text{ and } x_t - A_{\xi_t} \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

The transition distribution is updated by accounting for consumed resources if a request is accepted:

$$f(x_t, a, \xi_t) = \begin{cases} x_t - A_{\xi_t} & a_{\xi_t} = 1 \text{ and } x_t - A_{\xi_t} \geq 0 \\ x_t & \text{otherwise} \end{cases}$$

9.3.2.5 Virtual Machine Allocation ($k = T$)

The Cloud has modified the way that users are able access computing resources ([Hadary et al., 2020](#); [Cortez et al., 2017](#); [Jiang et al., 2020b](#); [Sheng et al., 2021](#); [Hubbs et al., 2020](#)). Cloud service providers allow customers easy access to resources while simultaneously applying efficient management techniques in order to optimize their return. One of the most critical components is the Virtual Machine (VM) allocator, which assigns VM request to the physical hardware (henceforth referred to as PMs). The important issue is how to allocate physical resources to service each VM efficiently by eliminating fragmentation, performance impact and delays, and allocation failures.

These VM allocation models can be thought of as a multi-dimensional variant of bin-packing with an additional component of arrival and departures. The typical VM scheduling scenarios models users requesting resources over time, where each request contains the required CPU and memory uses, and its lifetime. The allocator then decides which available physical machine to allocate the virtual machine to. To limit notational overload, we provide a high level view of the Virtual Machine allocation scenario here, and defer concrete discussion and notation when discussing the planning oracle required for solving.

In this set-up the current system state of the model is measured by the physical resources available for each PM, including the physical cores and memory available. Over time,

- Coming VM requests ask for a certain amount of resources (CPU and memory requirements) and their lifetime. Resource requirements are varied based on the different VM requests.

- Based on the action selected by the algorithm, the VM will be allocated to and be created in a specified PM as long as that PM's remaining resources are enough.
- After a period of execution, the VM completes its tasks. The simulator will then release the resources allocated to this VM and deallocate this VM from the PM.

At a high level, these problems can be modeled as an MDP with exogenous inputs where the exogenous space Ξ contains the space of possible VM requests (along with their lifetime, memory, and CPU requirements). The endogenous space $\mathcal{X}$ measures the current capacity of each physical machine on the server, and action space $\mathcal{A}$ for allocation decisions for the current VM request to a given PM. More details on the concrete experimental set-up will be in 9.7. We also note that the VM arrival process in practice is highly correlated so this fits under the model where $k = T$ (Hadary et al., 2020).

9.4 Hindsight Planners

Here we outline the feasibility of 5.3.2 in many operations tasks. Planning problems induced by online knapsack problems as a function of a deterministic input sequence ξ can be solved via their induced linear relaxation in pseudo-polynomial time (as the constraint polytope is a polymatroid). Other problems, such as inventory control with lead times have planning problems where a simple greedy control policy is optimal. More generally, 5.3.2 requires us to solve large-scale combinatorial optimization problems. However, we note that all of these computations are done *offline* and so the computational burden is not required at run-time. Moreover, it is easy to incorporate existing heuristics from the optimization literature for efficient solutions to these problems, including linear programming or fluid relaxations (Conforti et al., 2014). This

appeals to our Hindsight Learning algorithms only relying on the objective value of the planner, instead of the actual sequence of actions.

9.4.1 Inventory Control with Lead Times and Lost Sales

In inventory control, given knowledge of the exact sequence of demands $\xi = (d_1, \dots, d_T)$, the optimal open loop control policy is trivial to write down. Indeed, setting:

$$a_t = \begin{cases} d_{t+L} & t \leq T - L \\ 0 & \text{otherwise} \end{cases}$$

is clearly the optimal policy. This is as, for any $t \leq T - L$ we ensure the current on-hand inventory is exactly equal to that period's demands. For the last L periods we order nothing in order to minimize the accumulated purchase costs for inventory which will be ordered and cannot be sold.

9.4.2 Online Stochastic Bin Packing

We give the integer programming representation of the optimal open loop control for Bin Packing as follows. Consider a state x with components $x_1, \dots, x_B$ as the current number of bins opened with a utilization of 1 up to B and a sequence of items with sizes $\xi = (u_1, \dots, u_T)$. Given $(x_1, \dots, x_B)$ we pre-process this list to a vector of length $\sum_i x_i$ where each component corresponds to the current utilization of any bin. For example, if $B = 3$ and $x = (1, 0, 2)$ then we make a list containing $\alpha = (1, 3, 3)$ for two bins with a utilization of three and one bin with a total utilization of one. Since the total number of bins required will be $\sum_i x_i + T$ we use the variables

y_b for $b \in [\sum_i x_i + T]$ to denote an indicator of whether or not bin y_b is currently utilized. We also use variables $z_{v,b}$ to denote whether item $v \in [T]$ is assigned to bin b . Similarly, denote α_b as the current utilization of a bin b . The optimization program can then be written as follows:

$$\begin{aligned}
& \max_{z,y} - \sum_b y_b \\
& \text{s.t. } \sum_b z_{v,b} = 1 \text{ for all } v \in [T] \\
& \sum_v z_{v,b} + \alpha_b \leq B y_b \text{ for all } b \\
& y_b = 1 \text{ for any } b \text{ with } \alpha_b > 0
\end{aligned}$$

The objective corresponds to minimizing the number of utilized bins. The first constraint ensures that each item is assigned to a bin. The second constraint enforces capacity constraints for each bin, and the last constraint ensures that bins are marked as used if they have current utilization on them (i.e. $\alpha_b > 0$).

9.4.3 Online Secretary

In online secretary, given knowledge of the exact sequence of future candidate qualities the open loop control policy is trivial to write down. Indeed, for any $\xi_{\geq t}$ we denote σ as the ranking function over it such that $\xi_\sigma(1) > \xi_\sigma(2) > \dots > \xi_\sigma(T-t)$, with ties broken arbitrarily. Then given a remaining number of candidates to accept x , $\text{HINDSIGHT}(t, \xi_{\geq t}, x)$ will simply be $\sum_{i=1}^x \xi_{\sigma(i)}$. This corresponds to taking the best x -candidates from the future trace $\xi_{\geq t}$.

9.4.4 Airline Revenue Management

The planning oracle for the airline revenue management problem can be formulated as a so-called “knapsack” problem. Indeed, suppose that x_t is the remaining capacity for the K resources. We use variables z_y for the number of customers of type $y \in [M]$ to accept. Then, we solve the following optimization problem:

$$\begin{aligned} \max_z \quad & \sum_y f_y z_y \\ \text{s.t.} \quad & 0 \leq z_y \leq N_y(\xi) \text{ for all } y \in [M] \\ & Az \leq x_t \end{aligned}$$

where $N_y(\xi) = \sum_t \mathbb{1}_{[\xi_t=y]}$ is the number of type y customers in the exogenous dataset ξ .

9.4.5 Virtual Machine Allocation

The planning oracle for the VM allocation problem can be formulated as a large-scale mixed integer linear program. In this section we discuss approaches which utilize this fact in developing an oracle for the hindsight planning problem.

Use $\Xi = V$ to denote the set of VM requests and P as the set of physical machines. We use the following constants which depend on the current inventory of virtual and physical machines contained in the current state s_t , including:

- $\alpha_{t,p}$ for the remaining CPU cores for physical machine p at event t
- $\beta_{t,p}$ for the remaining memory for physical machine p at event t

- CPU-CAP_p and MEM-CAP_p the CPU and memory capacity of physical machine $p \in P$
- LIFETIME_v , CORE_v , MEM_v as the lifetime, cores, and memory utilization of VM $v \in V$
- $\eta_{v,t}$ an indicator that the VM v is active at time t (i.e. $\eta_{v,t}$ is equal to one for any time starting from the time the VM v arrives until the end of its lifetime)

With this we introduce variables $x_{v,p}$ for each virtual machine v and physical machine p to indicate the assignments. We also use variables $y_{t,p}$ which encode whether physical machine p has a VM assigned to it at time step t .

We start by considering the various constraints in the problem:

- **Assignment Constraint (2):** For every v we need $\sum_p x_{v,p} = 1$ indicating that each virtual machine is assigned to a physical machine.
- **CPU Capacity Constraint (3):** For every p and t we need $\alpha_{p,t} + \sum_v \text{CORE}_v \eta_{v,t} x_{v,p} \leq \text{CPU-CAP}_p$ to ensure CPU usage capacity constraints are satisfied.
- **Memory Capacity Constraint (4):** For every p and t we need $\beta_{p,t} + \sum_v \text{MEM}_v \eta_{v,t} x_{v,p} \leq \text{MEM-CAP}_p$ to ensure memory capacity constraints are satisfied.

We also need additional constraints which encode the $y_{t,p}$ variable as follows:

- **PM Historical Utilization (5):** $y_{t,p} \geq 1$ for all p and t if $\alpha_{t,p} > 0$
- **PM VM Utilization (6):** $x_{v,p} \eta_{v,t} \leq y_{t,p}$ for all v and p
- **PM OR Constraint (7):** $\sum_v x_{v,p} \eta_{v,t} + \mathbb{1}_{[\alpha_{t,p} > 0]} \geq y_{t,p}$ for all t and p .

These constraints essentially encode that $y_{t,p}$ is an indicator for whether y has a VM assigned to it from ξ (in the second bullet), or has historical allocations on it (for when $\alpha_{t,p} > 0$).

We note that there always exists a feasible solution since we are in an over provisioned regime where we have capacity to service every VM request.

Lastly, the objective function (1) for the packing density can be formulated via:

$$- \sum_t \frac{\sum_p y_{t,p} \text{CPU-CAP}_p}{\sum_p \alpha_{t,p} + \sum_v \text{CORE}_v}$$

The numerator corresponds to the total CPU capacity of all physical machines which are in use. The denominator corresponds to the total utilization (both from the VMs currently in service and the VMs arriving over the time horizon). This then encodes the inverse of the core packing density, as described earlier.

The full integer program is now summarized below:

$$\max_{x,y} - \sum_{t \in [T]} \frac{\sum_{p \in P} y_{t,p} \text{CPU-CAP}_p}{\sum_{p \in P} \alpha_{t,p} + \sum_{v \in V} \text{CORE}_v} \quad (1)$$

$$\text{s.t. } \sum_{p \in P} x_{v,p} = 1 \quad \forall v \in V \quad (2)$$

$$\alpha_{p,t} + \sum_{v \in V} \text{CORE}_v \eta_{v,t} x_{v,p} \leq \text{CPU-CAP}_p \quad \forall t \in [T], p \in P \quad (3)$$

$$\beta_{p,t} + \sum_{v \in V} \text{MEM}_v \eta_{v,t} x_{v,p} \leq \text{MEM-CAP}_p \quad \forall t \in [T], p \in P \quad (4)$$

$$y_{t,p} \geq \mathbb{1}_{[\alpha_{t,p} > 0]} \quad \forall t \in [T], p \in P \quad (5)$$

$$x_{v,p} \eta_{v,t} \leq y_{t,p} \quad \forall v \in V, t \in [T], p \in P \quad (6)$$

$$\sum_{v \in V} x_{v,p} \eta_{v,t} + \mathbb{1}_{[\alpha_{t,p} > 0]} \geq y_{t,p} \quad \forall t \in [T], p \in P \quad (7)$$

9.5 Existing Approaches to MDPs in Exo-MDPs

Here we briefly discuss existing approaches to MDPs applied in the context of Exo-MDPs to highlight the advantages and disadvantages of our Hindsight Learning approach.

9.5.1 Predict Then Optimize

Given the historical trace dataset $\mathcal{D} = \{\xi^1, \dots, \xi^N\}$, a popular plug-in approach learns a generative model $\overline{\mathcal{P}}_{\Xi}(\xi_t \mid \xi_{<t})$ to approximate the true distribution $\mathcal{P}_{\Xi}(\xi_t \mid \xi_{<t})$, since the exogenous process is the only unknown. Given this model $\overline{\mathcal{P}}_{\Xi}$, estimates for the Q_t^* value for the optimal policy can be obtained by solving the Bellman equation with the learned predictor $\overline{\mathcal{P}}_{\Xi}$ in place of the true distribution $\mathcal{P}_{\Xi}$. More concretely, we denote $\overline{Q}_t$ as the model-based estimate of Q_t^* , which follows

$$\begin{aligned}\overline{Q}_t(s, a) &:= \mathbb{E}_{\xi \mid \xi_{<t}}[r(x, a, \xi) + \overline{V}_{t+1}(f(x, a, \xi) \mid \overline{\mathcal{P}}_{\Xi})] \\ \overline{V}_t(s) &:= \max_{a \in \mathcal{A}} \overline{Q}_t(s, a) \\ \overline{\pi}_t(s) &:= \arg \max_{a \in \mathcal{A}} \overline{Q}_t(s, a).\end{aligned}$$

While intuitive, this ML forecast approach requires high-fidelity modeling of the exogenous process to guarantee good downstream decision-making, due to the quadratic horizon multiplicative factor in regret we show below. This quadratic factor in the horizon is due to the compounding errors of distribution shift, similar to those shown in the imitation learning literature (Ross et al., 2011).

Theorem 9.5.1. *Suppose that $\sup_{t \in [T], \xi_{<t} \in \Xi^{[t-1]}} \|\overline{\mathcal{P}}_{\Xi}(\cdot \mid \xi_{<t}) - \mathcal{P}_{\Xi}(\cdot \mid \xi_{<t})\|_{TV} \leq \epsilon$ where $\|\cdot\|_{TV}$ is*

the total variation distance. Then we have that $\text{REGRET}(\bar{\pi}) \leq 2T^2\epsilon$. In addition, if $\xi \sim \mathcal{P}_\Xi$ has each ξ_t independent from $\xi_{<t}$, $\bar{\mathcal{P}}_\Xi$ is the empirical distribution, then $\forall \delta \in (0, 1)$, with probability at least $1 - \delta$, $\text{REGRET}(\bar{\pi}) \leq 2T^2 \sqrt{\frac{\log(2|\Xi|/\delta)}{N}}$.

The T^2 dependence here is tight (see Domingues et al. (2021)), in contrast to the $O(T)$ dependence in 5.6.3. Moreover, the ML forecast approach can be impractical when the exogenous process is complex since ϵ in the worst case can scale as $|\Xi|^T$ if the ξ_t are strongly correlated across t . An example of this is VM allocation where researchers observed that the VM lifetime varies substantially across time, the demand has spikes and a diurnal pattern, and that subsequent requests are highly correlated (Hadary et al., 2020).

This discrepancy highlights an advantage of our Hindsight Learning approach. Consider a VM allocation example with two physical machines each large enough to satisfy the entire demand. Under chaotic and unpredictable arrivals, a planner using erroneous forecasts might spread the requests over the two machines. In contrast, the hindsight learning policy will correctly learn that one machine is sufficient and achieve low regret, even if the total variation distance on the underlying distribution over exogenous inputs is large.

9.5.2 Reinforcement Learning

Recall that our objective is to solve $\arg \max_{\pi \in \Pi} V_1^\pi(s_1)$. This can be written as $\arg \max_{\pi \in \Pi} \mathbb{E}_\xi[V_1^\pi(s_1, \xi)]$ by 5.3.1. Therefore, an alternative way to find approximately optimal policies for an Exo-MDP is to maximize the empirical return directly, similar to the empirical risk minimization strategy of supervised learning: $\bar{\pi}^* = \arg \max_{\pi \in \Pi} \bar{\mathbb{E}}[V_1^\pi(s_1, \xi)]$ where $\bar{\mathbb{E}}[V_1^\pi(s_1, \xi)] = \frac{1}{N} \sum_n V_1^\pi(s_1, \xi^n)$. First observe that the number of samples required to learn a near optimal policy scales linearly

with T in this approach. If an additive control variate $\phi(\xi)$ (as in Mao et al. (2019b)) is used, the T term is replaced with $T - \mathbb{E}_\xi[\phi(\xi)]$.

Theorem 9.5.2. $\forall \delta \in (0, 1)$, with probability at least $1 - \delta$, $\text{REGRET}(\bar{\pi}^*) \leq T \sqrt{\frac{2 \log(2|\Pi|/\delta)}{N}}$.

9.5.2 highlights that model-free RL methods are a *theoretically viable* approach for Exo-MDPs (especially compared to Predict-Then-Optimize in 9.5.1). Indeed, 9.5.2 shows that RL methods have asymptotic consistency guarantees *if* they converge to the optimal policy in the empirical MDP. This convergence is an idealized computation assumption that hides optimization issues when studying statistical guarantees, and is incomparable to 5.3.2 for the hindsight planner (for which we showed several examples in 9.4).

Moreover, 5.1 suggests that HL and RL are trading bias and variance differently. When variance is the dominating factor, we expect an algorithm’s performance to improve with additional data. When bias is the dominating factor, however, we expect no marginal benefit from additional data. In Table 1 we see that as T (and accordingly, N , the number of data points) increases, the Tabular RL algorithm performance improves. However, hindsight learning has a stable non-zero regret even as we increase T .

9.6 Proofs of Main Results

Lemma 9.6.1 (5.3.1 of 5.3). *For every $t \in [T]$, $(s, a) \in \mathcal{S} \times \mathcal{A}$, and $\pi \in \Pi$, we have the following:*

$$Q_t^\pi(s, a) = \mathbb{E}_{\xi_{\geq t}}[Q_t^\pi(s, a, \xi_{\geq t})]$$

$$V_t^\pi(s) = \mathbb{E}_{\xi_{\geq t}}[V_t^\pi(s, \xi_{\geq t})].$$

In particular $V_1^\pi(s_1) = V_1^\pi = \mathbb{E}_\xi[V_1^\pi(s_1, \xi)]$.

Proof. First note that if $Q_t^\pi(s, a)$ is as defined then we have that:

$$\begin{aligned}
V_t^\pi(s) &= \sum_a \pi(a \mid s) Q_t^\pi(s, a) \text{ (by Bellman equations)} \\
&= \sum_a \pi(a \mid s) \mathbb{E}_{\xi_{\geq t}}[Q_t^\pi(s, a, \xi_{\geq t})] \\
&= \mathbb{E}_{\xi_{\geq t}} \left[\sum_a \pi(a \mid s) Q_t^\pi(s, a, \xi_{\geq t}) \right] \text{ (by } \pi \text{ being non-anticipatory)} \\
&= \mathbb{E}_{\xi_{\geq t}}[V_t^\pi(s, \xi_{\geq t})].
\end{aligned}$$

Now we focus on showing the result for $Q_t^\pi(s, a)$ by backwards induction on t . The base case when $t = T$ is trivial as $Q_T^\pi(s, a) = \mathbb{E}_\xi[r(s, a, \xi)] = \mathbb{E}_{\xi_{\geq T}}[Q_T^\pi(s, a, \xi_{\geq T})]$.

Step Case: $(t + 1 \rightarrow t)$ For the step-case a simple derivation shows that

$$\begin{aligned}
Q_t^\pi(s, a) &= \mathbb{E}_\xi[r(s, a, \xi) + V_{t+1}^\pi(f(s, a, \xi))] \\
&= \mathbb{E}_\xi[r(s, a, \xi) + \mathbb{E}_{\xi_{\geq t+1}}[V_{t+1}^\pi(f(s, a, \xi), \xi_{\geq t+1})]] \\
&= \mathbb{E}_{\xi_{\geq t}}[r(s, a, \xi_t) + V_{t+1}^\pi(f(s, a, \xi_t), \xi_{\geq t+1})] \\
&= \mathbb{E}_{\xi_{\geq t}}[Q_t^\pi(s, a, \xi_{\geq t})].
\end{aligned}$$

□

Theorem 9.6.2 (5.6.2 of 5.6). $\text{REGRET}(\pi^\dagger) \leq \sum_{t=1}^T \mathbb{E}_{S_t \sim \text{Pr}_t^{\pi^\dagger}}[\Delta_t^\dagger(S_t)]$ where $\text{Pr}_t^{\pi^\dagger}$ denotes the state distribution of $\pi^\dagger$ at step t induced by the exogenous randomness. In particular, if $\Delta_t^\dagger(s) \leq \Delta$ for some constant Δ then we have: $\text{REGRET}(\pi^\dagger) \leq \Delta \sum_{t=1}^T \mathbb{E}_{S_t \sim \text{Pr}_t^{\pi^\dagger}}[\text{Pr}(\pi_t^\dagger(S_t) \neq \pi^*(S_t))]$.

Proof. First we note that via the performance difference lemma we have that for any two non-anticipatory policies π and $\tilde{\pi}$ that

$$\begin{aligned} V_1^\pi(s_1) - V_1^{\tilde{\pi}}(s_1) &= \sum_t \mathbb{E}_{s_t \sim \text{Pr}_t^\pi} \left[\sum_a \pi(a | s_t) (Q_h^{\tilde{\pi}}(s, a) - V_h^{\tilde{\pi}}(s)) \right] \text{ and so} \\ V_1^{\tilde{\pi}}(s_1) - V_1^\pi(s_1) &= \sum_t \mathbb{E}_{s_t \sim \text{Pr}_t^{\tilde{\pi}}} \left[\sum_a \pi(a | s_t) (V_h^{\tilde{\pi}}(s) - Q_h^{\tilde{\pi}}(s, a)) \right] \end{aligned}$$

Moreover, for any state s we also have $Q_t^*(s, \pi^*(s)) - Q_t^*(s, \pi^\dagger(s)) \leq \Delta_t^\dagger(s)$ since:

$$\begin{aligned} &Q_t^*(s, \pi^*(s)) - Q_t^*(s, \pi^\dagger(s)) - \Delta_t^\dagger(s) \\ &= Q_t^*(s, \pi^*(s)) - Q_t^*(s, \pi^\dagger(s)) - Q_t^\dagger(s, \pi^\dagger(s)) + Q_t^*(s, \pi^\dagger(s)) - Q_t^*(s, \pi^*(s)) + Q_t^\dagger(s, \pi^*(s)) \\ &= Q_t^\dagger(s, \pi^*(s)) - Q_t^\dagger(s, \pi^\dagger(s)) \leq 0 \end{aligned}$$

as $\pi^\dagger$ is greedy with respect to $Q^\dagger$.

Finally, recall the definition of the regret of $\text{REGRET}(\pi^\dagger)$ via $V_1^*(s_1) - V_1^{\pi^\dagger}(s_1)$. However, using the previous performance difference lemma with $\tilde{\pi} = \pi^*$ and $\pi = \pi^\dagger$ we have that

$$\begin{aligned} V_1^*(s_1) - V_1^{\pi^\dagger}(s_1) &= \sum_t \mathbb{E}_{S_t \sim \text{Pr}_t^{\pi^\dagger}} [Q_t^*(S_t, \pi^*(S_t)) - Q_t^*(S_t, \pi^\dagger(S_t))] \\ &\leq \sum_t \mathbb{E}_{S_t \sim \text{Pr}_t^{\pi^\dagger}} [\Delta_t^\dagger(S_t)]. \end{aligned}$$

The second statement follows immediately from the absolute bound on $\Delta_t^\dagger(s)$. $\square$

Theorem 9.6.3 (5.6.3 of 5.6). *Let $\bar{\pi}^\dagger$ denote the hindsight planning surrogate policy for the empirical MDP w.r.t. $\mathcal{D}$. Assume $\bar{\pi}^\dagger \in \Pi$ and I achieves no-regret in the optimization problem.*

Let π be the best policy generated by 1. Then, for any $\delta \in (0, 1)$, with probability $1 - \delta$, it holds

$$\text{REGRET}(\pi) \leq T \sqrt{\frac{2 \log(2|\Pi|/\delta)}{N}} + \sum_{t=1}^T \mathbb{E}_{s_t \sim \bar{Pr}_t^{\bar{\pi}^\dagger}} [\bar{\Delta}_t^\dagger(s_t)] + o(1)$$

where $\bar{\Delta}_t^\dagger$ is the SAA approximation of Eqn. 11 and $\bar{Pr}_t^{\bar{\pi}^\dagger}$ is the state probability of $\bar{\pi}^\dagger$ in the empirical MDP.

Proof. The proof follows the standard proof technique of online IL (cf. (Yan et al., 2021)).

$$\begin{aligned} \text{REGRET}(\pi) &= \mathbb{E}[V_1^*(s_1, \xi)] - \mathbb{E}[V_1^\pi(s_1, \xi)] \\ &\leq (|\mathbb{E}[V_1^\pi(s_1, \xi)] - \mathbb{E}[V_1^{\pi^*}(s_1, \xi)]| + |\mathbb{E}[V_1^{\pi^*}(s_1, \xi)] - \mathbb{E}[V_1^{\bar{\pi}^\dagger}(s_1, \xi)]|) \\ &\quad + (\mathbb{E}[V_1^{\bar{\pi}^\dagger}(s_1, \xi)] - \mathbb{E}[V_1^{\bar{\pi}^\dagger}(s_1, \xi)] + \mathbb{E}[V_1^{\bar{\pi}^\dagger}(s_1, \xi)] - \mathbb{E}[V_1^\pi(s_1, \xi)]) \\ &\leq 2T \sqrt{\frac{2 \log(2|\Pi|/\delta)}{N}} + \mathbb{E}[V_1^{\pi^*}(s_1, \xi)] - \mathbb{E}[V_1^{\bar{\pi}^\dagger}(s_1, \xi)] + o(1) \\ &\leq 2T \sqrt{\frac{2 \log(2|\Pi|/\delta)}{N}} + \sum_{t=1}^T \mathbb{E}_{s_t \sim \bar{Pr}_t^{\bar{\pi}^\dagger}} [\bar{\Delta}_t^\dagger(s_t)] + o(1) \end{aligned}$$

We use the results of 9.6.8 to bound the first terms in the second line; we invoke the no-regret optimization assumption and the realizability assumption $\bar{\pi}^\dagger \in \Pi$ for the last term of the second line. Finally, we apply 5.6.2 in the empirical MDP w.r.t. $\mathcal{D}$ and recognize that the middle term is the empirical regret to derive the last step. $\square$

Lemma 9.6.4 (5.6.4 of 5.6). *There exists a set of Exo-MDPs such that $\text{REGRET}(\pi^\dagger) \geq \Omega(T)$.*

Proof. We first construct a three-step MDP such that $\text{REGRET}(\pi^\dagger) \geq \Omega(1)$. The main result then follows by replicating the MDP across T periods to construct a $3T$ step MDP with $\text{REGRET}(\pi^\dagger) \geq \Omega(T)$.

We consider a modification of the prototypical Pandora's Box problem (Weitzman, 1979). The endogenous state space $\mathcal{X} = \{0, 1\}$ where state 0 corresponds to “not yet accepted an item” and 1 corresponds to “accepted an item”. The action space $\mathcal{A} = \{0, 1\}$ where $a = 0$ corresponds to “reject next item” and $a = 1$ corresponds to “accept next item”. We consider a modification of the typical Pandora box model where at time step t , the next item arrivals ξ_t value is unobserved before deciding whether or not to accept.

The trace distribution has $\xi_1 \sim U[0, 1]$ and $\xi_2, \xi_3 \sim U[0, .9]$. Important to note is that $\mathbb{E}[\xi_1] = .5$, $\mathbb{E}[\xi_2] = 0.45$, $\mathbb{E}[\xi_3] = 0.45$, and a straightforward calculation shows that $\mathbb{E}[\max(\xi_2, \xi_3)] = 0.6$.

The rewards and dynamics are:

$$\begin{aligned} r(0, 0, \xi) &= 0 & f(0, 0, \xi) &= 0 \\ r(0, 1, \xi) &= \xi & f(0, 1, \xi) &= 1 \\ r(1, a, \xi) &= 0 & f(1, a, \xi) &= 1 \text{ for } a \in \{0, 1\}. \end{aligned}$$

Lastly, the starting state $s_1 = 0$. This properly encodes the exogenous dynamics and rewards. At step t in state $x = 0$ (i.e. not yet accepted an item) taking action 0 (do not accept) yields no return and transitions to the next state. However, accepting the next item returns reward ξ and transitions to state $x = 1$.

A straightforward calculation following the Bellman equations shows the following for Q_t^* and V_t^* :

$$Q_3^*(0, 0) = 0 \quad Q_2^*(0, 0) = \mathbb{E}[\xi_3] \quad Q_1^*(0, 0) = \max(\mathbb{E}[\xi_2], \mathbb{E}[\xi_3]).$$

$$\begin{array}{lll}
Q_3^*(0, 1) = \mathbb{E}[\xi_3] & Q_2^*(0, 1) = \mathbb{E}[\xi_2] & Q_1^*(0, 1) = \mathbb{E}[\xi_1] \\
Q_3^*(1, \cdot) = 0 & Q_2^*(1, \cdot) = 0 & Q_1^*(1, \cdot) = 0 \\
V_3^*(0) = \mathbb{E}[\xi_3] & V_2^*(0) = \max(\mathbb{E}[\xi_3], \mathbb{E}[\xi_2]) & V_1^*(0) = \max(\mathbb{E}[\xi_1], \mathbb{E}[\xi_2], \mathbb{E}[\xi_3]) \\
V_3^*(1) = 0 & V_2^*(1) = 0 & V_1^*(1) = 0.
\end{array}$$

Using the choice of the distributions for ξ_1, ξ_2, ξ_3 we have that $\pi_1^*(0) = 1$, as in, we will accept the first item since on average it has larger expected return. This results in $V_1^{\pi^*}(s_1) = \mathbb{E}[\xi_1] = 0.5$.

We can similarly compute Q_t^+ and V_t^+ as follows:

$$\begin{array}{lll}
Q_3^+(0, 0) = 0 & Q_2^+(0, 0) = \mathbb{E}[\xi_3] & Q_1^+(0, 0) = \mathbb{E}[\max(\xi_2, \xi_3)]. \\
Q_3^+(0, 1) = \mathbb{E}[\xi_3] & Q_2^+(0, 1) = \mathbb{E}[\xi_2] & Q_1^+(0, 1) = \mathbb{E}[\xi_1] \\
Q_3^+(1, \cdot) = 0 & Q_2^+(1, \cdot) = 0 & Q_1^+(1, \cdot) = 0.
\end{array}$$

In this scenario, we see first hand the bias introduced when considered Q^+ . In particular, $Q_1^+(0, 0) = \mathbb{E}[\max(\xi_2, \xi_3)] \geq Q_1^*(0, 0) = \max(\mathbb{E}[\xi_2, \xi_3])$. Using the choice of distributions for ξ_1, ξ_2, ξ_3 we see that the hindsight planning policy π^+ is fooled and has $\pi_1^+(0) = 0$, so the policy rejects the first item thinking it will get the maximum value of the next two items. As a result we see that $V_1^{\pi^+}(0) = 0.45$.

Hence, we have that $\text{REGRET}\pi^+ = 0.5 - 0.45 = 0.05 = \Omega(1)$ as needed. $\square$

Lemma 9.6.5 (Statement in 5.6). *Suppose that for every t and state s that $\max_{\pi \in \Pi} \mathbb{E}_{\xi_{\geq t}}[V_t^\pi(s, \xi_{\geq t})] = \mathbb{E}_{\xi_{> t}}[\text{HINDSIGHT}(t, s, \xi_{\geq t})]$. Then we have that $Q_t^*(s, a) = Q_t^+(s, a)$ for every t, s , and a .*

Proof. First notice that $\max_{\pi \in \Pi} \mathbb{E}_{\xi_{\geq t}}[V_t^\pi(s, \xi_{\geq t})] = \mathbb{E}_{\xi_{\geq t}}[V_t^*(s, \xi_{\geq t})]$ by definition. Thus using the Bellman equations and definition of $Q_t^+(s, a)$ we trivially have that:

$$\begin{aligned}
Q_t^*(s, a) &= \mathbb{E}_{\xi_t}[r(s, a, \xi_t) + V_{t+1}^*(f(s, a, \xi_t))] \\
&= \mathbb{E}_{\xi_t}[r(s, a, \xi_t) + \mathbb{E}_{\xi_{>t}}[V_{t+1}^*(f(s, a, \xi_t), \xi_{>t})]] \\
&= \mathbb{E}_{\xi_t}[r(s, a, \xi_t) + \mathbb{E}_{\xi_{>t}}[\text{HINDSIGHT}(t+1, f(s, a, \xi_t), \xi_{>t})]] \\
&= Q_t^+(s, a).
\end{aligned}$$

□

Lemma 9.6.6 (Statement in 5.6). *Define $\widetilde{GAG} = \mathbb{E}\left[\max_{\pi} \sum_{t=1}^T \Delta_t^+(S_t) \mid S_t \sim \text{Pr}_t^\pi\right]$. Then there exists an Exo-MDP such that $\widetilde{GAG} = \Omega(T)$ and yet the upper bound in 5.6.2 is zero.*

The “MakeDecision” function in [Mercier and Van Hentenryck \(2007\)](#) implements the empirical Bayes Selector policy using the offline dataset (see 6). There are several key differences between our regret analysis and theirs. First, [Mercier and Van Hentenryck \(2007\)](#) define regret with respect to the hindsight optimal policy $V^+(s_1)$, whereas our regret is with respect to the best non-anticipatory policy $V^*(s_1)$. Their main result (Theorem 1) shows that the hindsight optimal regret is bounded by the “Global Anticipatory Gap” $GAG = \mathbb{E}\left[\max_{\pi} \sum_{t=1}^T \Omega_t(S_t, A_t) \mid (S_t, A_t) \sim \text{Pr}_t^\pi\right]$. However, there are situations where the GAG can be large, and one could incorrectly conclude that hindsight optimization should not be applied when trying to learn the true optimal non-anticipative policy π^* . In the Sailing example of 5.4, the GAG will be positive since knowing the direction of the wind one can ex-post identify the optimal route compared to any non-anticipatory algorithm. However, $Q^*(\text{route}) = Q^+(\text{route})$ and so there is no hindsight bias. Hence, our results

which adjust for the difference in benchmark to π^* is more appropriate.

Moreover, their regret bound measures a worst-case overestimation bias on the states visited by any decision policy. In contrast, 5.6.2 requires that the hindsight bias be small only on states visited by π^+ . Even if we set aside the difference between our benchmarks for regret (adjusting the definition by $V^*(s_1) - V^+(s_1)$ resulting in $\widetilde{GAG}$ defined in the statement of 9.6.6), our analysis is much tighter which we illustrate with an example below.

Proof. We construct an Exo-MDP $\mathcal{M}$ as follows. Consider a starting state s_0 with two actions A and B that deterministically transition to two different “sub-MDPs” (which we denote as $\mathcal{M}_A$ and $\mathcal{M}_B$ respectively).

The first action, A , transitions to sub-MDP $\mathcal{M}_A$ which contains an absorbing state s_A (i.e. transitions are deterministic to the same state s_A regardless of the action) with large rewards. Note that since this sub-MDP is deterministic it has no hindsight bias (so $\Delta_t^+(s_A) = 0$).

The second action B , transitions to an MDP $\mathcal{M}_B$ which witnesses 5.6.4, and hence has $\Omega(T)$ hindsight bias. We adjust the rewards along $\mathcal{M}_B$ such that the value of the optimal policy in this sub-MDP is much smaller than the deterministic reward accrued in $\mathcal{M}_A$.

In this example, π^+ will always select action A in the initial state and collect higher rewards (since the optimal policy knowing the exogenous inputs in MDP $\mathcal{M}_B$ will still collect smaller rewards than the deterministic value accrued in $\mathcal{M}_A$). Hence, our regret bound in 5.6.2 will be zero (since $\Delta^+(s_A) = 0$ and π^+ will never visit s_B). However, $\widetilde{GAG}$ will conservatively account for the sub-optimal B decision which transitions to a state with a large anticipatory gap ($\Omega(T)$), thereby concluding a large regret bound. $\square$

Theorem 9.6.7 (9.5.1 of 9.5). *Suppose that $\sup_{t \in [T], \xi_{<t} \in \Xi^{[t-1]}} \|\overline{\mathcal{P}}_\Xi(\cdot | \xi_{<t}) - \mathcal{P}_\Xi(\cdot | \xi_{<t})\|_{TV} \leq \epsilon$*

where $\|\cdot\|_{TV}$ is the total variation distance. Then we have that $\text{REGRET}(\bar{\pi}) \leq 2T^2\epsilon$. In addition, if $\xi \sim \mathcal{P}_\Xi$ has each ξ_t independent from $\xi_{<t}$, $\bar{\mathcal{P}}_\Xi$ is the empirical distribution, then $\forall \delta \in (0, 1)$, with probability at least $1 - \delta$, $\text{REGRET}(\bar{\pi}) \leq 2T^2 \sqrt{\frac{\log(2|\Xi|/\delta)}{N}}$.

Proof. First note that $\bar{Q}_t$ and $\bar{V}_t$ refer to the Q and V values for the optimal policy in a modified MDP $\bar{M}$ where the true exogenous input distribution $\mathcal{P}_\Xi(\cdot \mid \xi_{<t})$ is replaced by its estimate $\bar{\mathcal{P}}_\Xi(\cdot \mid \xi_{<t})$. As such, denote by $\bar{V}_t^\pi$ as the value function for the policy π in the MDP $\bar{M}$. Note here that $\bar{V}_t^{\bar{\pi}} = \bar{V}_t$ by construction. With this we have that:

$$\begin{aligned} \text{REGRET}(\bar{\pi}) &= V_1^*(s_1) - V_1^{\bar{\pi}}(s_1) \\ &= V_1^*(s_1) - \bar{V}_1^{\pi^*}(s_1) + \bar{V}_1^{\pi^*}(s_1) - \bar{V}_1(s_1) + \bar{V}_1(s_1) - V_1^{\bar{\pi}}(s_1) \\ &\leq 2 \sup_{\pi} |V_1^\pi(s_1) - \bar{V}_1^\pi(s_1)|. \end{aligned}$$

However, using the finite horizon simulation lemma (see Lemma 1 in [Abbeel and Ng \(2005\)](#)) we have that this is bounded from above by $2T^2\|P(\cdot \mid s, a) - \bar{P})(\cdot \mid s, a)\|_{TV}$ where $\|P(\cdot \mid s, a) - \bar{P})(\cdot \mid s, a)\|_{TV}$ is the total variation distance in the induced state-transition distributions between M and $\bar{M}$. However, by definition we have that:

$$\begin{aligned} \|P(\cdot \mid s, a) - \bar{P}(\cdot \mid s, a)\|_{TV} &= \frac{1}{2} \int_{\mathcal{S}} |P(s' \mid s, a) - \bar{P}(s' \mid s, a)| ds \\ &= \frac{1}{2} \int_{\mathcal{S}} \left| \int_{\Xi} \mathbb{1}_{[s'=f(s,a,\xi)]} d\mathcal{P}_\Xi(\xi \mid \xi_{\geq t}) - \mathbb{1}_{[s'=f(s,a,\xi)]} d\bar{\mathcal{P}}_\Xi(\xi \mid \xi_{\geq t}) \right| ds \\ &= \frac{1}{2} \int_{\mathcal{S}} \int_{\Xi} \mathbb{1}_{[s'=f(s,a,\xi)]} |d\mathcal{P}_\Xi(\xi \mid \xi_{\geq t}) - d\bar{\mathcal{P}}_\Xi(\xi \mid \xi_{\geq t})| \\ &\leq \|\mathcal{P}_\Xi(\cdot \mid \xi_{\geq t}) - \bar{\mathcal{P}}_\Xi(\cdot \mid \xi_{\geq t})\|_{TV} \leq \epsilon. \end{aligned}$$

Thus we get that $\text{REGRET}(\bar{\pi}) \leq 2T^2\epsilon$ as required.

Now suppose that $\xi \sim \mathcal{P}_\Xi$ has each ξ_t independent from $\xi_{<t}$ and let $\bar{\mathcal{P}}_\Xi$ be the empirical distribution, i.e. $\bar{\mathcal{P}}_\Xi(\xi) = \frac{1}{N} \sum_{i \in [N]} \mathbb{1}_{[\xi_i = \xi]}$. A straightforward application of Hoeffding's inequality shows that the event:

$$\mathcal{E} = \left\{ \forall t, \xi_{<t} : |\bar{\mathcal{P}}_\Xi(\xi) - \mathcal{P}_\Xi(\xi)| \leq \sqrt{\frac{\log(2|\Xi|/\delta)}{2N}} \right\}$$

occurs with probability at least $1 - \delta$. Under $\mathcal{E}$ we then have that:

$$\sup_{t \in [T], \xi_{<t} \in \Xi^{[t-1]}} \|\bar{\mathcal{P}}_\Xi(\cdot | \xi_{<t}) - \mathcal{P}_\Xi(\cdot | \xi_{<t})\|_{TV} \leq \sup_{t \in [T], \xi_{<t} \in \Xi^{[t-1]}} \|\bar{\mathcal{P}}_\Xi(\cdot | \xi_{<t}) - \mathcal{P}_\Xi(\cdot | \xi_{<t})\|_1 \leq \sqrt{\frac{\log(2|\Xi|/\delta)}{2N}}.$$

Taking this in the previous result shows the claim. $\square$

Theorem 9.6.8 (9.5.2 of 9.5). *Given any $\delta \in (0, 1)$ then with probability at least $1 - \delta$ we have that if $\bar{\pi}^* = \arg \max_{\pi} \mathbb{E}[V^\pi(\xi)]$ that*

$$\text{REGRET}(\bar{\pi}^*) \leq \sqrt{\frac{2T^2 \log(2|\Pi|/\delta)}{N}}.$$

Proof. A quick calculation using Hoeffding's inequality and a union bound shows that the event

$$\mathcal{E} = \left\{ \forall \pi \in \Pi : |V_1^\pi(s_1) - \mathbb{E}[V_1^\pi(s_1)]| \leq \sqrt{\frac{T^2 \log(2|\Pi|/\delta)}{2N^2}} \right\}$$

occurs with probability at least $1 - \delta$. Under $\mathcal{E}$ we then have that:

$$\text{REGRET}(\bar{\pi}^*) = V_1^{\bar{\pi}^*}(s_1) - V_1^{\bar{\pi}^*}(s_1)$$

$$\begin{aligned}
&= V_1^{\pi^*}(s_1) - \mathbb{E}[V_1^{\pi^*}(s_1, \boldsymbol{\xi})] + \mathbb{E}[V_1^{\pi^*}(s_1, \boldsymbol{\xi})] - \mathbb{E}[V_1^{\bar{\pi}^*}(s_1, \boldsymbol{\xi})] + \mathbb{E}[V_1^{\bar{\pi}^*}(s_1, \boldsymbol{\xi})] - V_1^{\bar{\pi}^*}(s_1) \\
&\leq 2\sqrt{\frac{V_{max}^2 \log(2|\Pi|/\delta)}{2N^2}}.
\end{aligned}$$

□

Theorem 9.6.9 (9.7.1 of 9.7.1). *In stochastic online bin packing with i.i.d. arrivals we have that*

$\sup_{t,s} \Delta_t^+(s) \leq O(1)$, *independent of the time horizon and any problem primitives. As a result,*

$\text{REGRET}(\pi^\dagger) \leq O(1)$.

We show the result by starting with the lemma, highlighting that the value functions for the planning policy and the optimal non-anticipatory policy are “Lipschitz” with respect to the capacity of the current bins. Recall that the state space representation $s \in \mathcal{S}$ corresponds to $s = (x, \xi_{t-1})$ where $x \in \mathbb{R}^B$ is the current number of bins at that size, and the last component to the current arrival. We write this explicitly as containing $s \in \mathbb{R}^B$ for the bin utilization and $\xi_{t-1} \in \mathbb{R}$ for the current arrival.

Lemma 9.6.10. *For any $t \in [T]$, current bin capacity $x \in \mathbb{R}^{|B|}$, current arrival ξ_{t-1} , $\boldsymbol{\xi}_{\geq t} \in \Xi^{T-t}$, and $\Delta \in \mathbb{R}^B \geq 0$ we have that:*

- $V_t^\dagger(x, \xi_{t-1}, \boldsymbol{\xi}_{\geq t}) \geq V_t^\dagger(x - \Delta, \xi_{t-1}, \boldsymbol{\xi}_{\geq t}) \geq V_t^\dagger(x, \xi_{t-1}, \boldsymbol{\xi}_{\geq t}) - \|\Delta\|_1$
- $V_t^*(x, \xi_{t-1}) \geq V_t^*(x - \Delta, \xi_{t-1}) \geq V_t^*(x, \xi_{t-1}) - \|\Delta\|_1$

As a result for any x and x' in $\mathbb{R}^B$ and current arrival ξ_{t-1} we have that:

- $V_t^\dagger(x, \xi_{t-1}, \boldsymbol{\xi}_{\geq t}) - V_t^\dagger(x', \xi_{t-1}, \boldsymbol{\xi}_{\geq t}) \leq \|(x - x')^+\|_1$
- $V_t^*(x, \xi_{t-1}) - V_t^*(x', \xi_{t-1}) \leq \|(x - x')^+\|_1$

Proof. First consider the top statement in terms of the optimal planning policy starting from a fixed state $s = (x, \xi_{t-1})$ and sequence of future exogenous variables $\xi_{\geq t}$.

We have that $V_t^\dagger(x, \xi_{t-1}, \xi_{\geq t}) \geq V_t^\dagger(x - \Delta, \xi_{t-1}, \xi_{\geq t})$ since the sequence of actions generated by the planning oracle starting from state $(x - \Delta, \xi_{t-1})$ is feasible for the same problem starting from (x, ξ_{t-1}) . Hence, as $V_t^\dagger(x, \xi_{t-1}, \xi_{\geq t})$ denotes the optimal such policy, the inequality follows.

For the other direction consider the sequence of actions starting from (x, ξ_{t-1}) . Using at most $\|\Delta\|_1$ bins the policy is feasible for the same problem starting at $(x - \Delta, \xi_{t-1})$. Indeed, suppose the sequence of actions starting from the problem at (x, ξ_{t-1}) attempts to use a bin which is not available in the problem starting from $(x - \Delta, \xi_{t-1})$. Then by opening a new bin instead and shifting all future references of the old bin to the newly created bin, the sequence of actions is feasible. As there are at most $\|\Delta\|_1$ bins different in the (x, ξ_{t-1}) problem versus the $(x - \Delta, \xi_{t-1})$ problem, the bound follows.

Now consider the second statement in terms of the optimal non-anticipatory policy starting from a fixed state (x, ξ_{t-1}) . First note that $V_t^\star(x, \xi_{t-1}) = \mathbb{E}_{\xi_{\geq t}}[V_t^\star(x, \xi_{t-1}, \xi_{\geq t})]$ and similarly for $V_t^\star(x - \Delta, \xi_{t-1})$. We have that $V_t^\star(x, \xi_{t-1}) \geq V_t^\star(x - \Delta, \xi_{t-1})$ as the optimal policy starting from $(x - \Delta, \xi_{t-1})$ is feasible on all sample paths generated by $\xi_{\geq t}$ to the same problem starting at (x, ξ_{t-1}) . Hence by optimality of $\pi^\star$ the inequality must follow.

For the other direction, on any sample path consider the sequence of actions generated by the optimal policy starting from (x, ξ_{t-1}) . By a similar argument, again using at most $\|\Delta\|_1$ extra bins the policy is feasible for the problem starting at $(x - \Delta, \xi_{t-1})$. Hence by optimality, the inequality follows.

The second result follows via straightforward algebraic manipulations. Indeed, the previous

statement can be thought of as showing that for $x \in \mathbb{R}^B$ and $\Delta \in \mathbb{R}_+^B$ that $f(x) \geq f(x - \Delta) \geq f(x) - \|\Delta\|_1$. However,

$$f(x) - f(x') = f(x) - f(x' + (x - x')^+) + f(x' + (x - x')^+) - f(x') \leq f(x' + (x - x')^+) - f(x') \leq \|(x - x')^+\|_1$$

where the first inequality uses that $x' + (x - x')^+ \geq x$ and the second the previous result. $\square$

We are now ready to show the bound that $\Delta_t^+(s) \leq O(1)$.

Proof. For a fixed time t and state s consider $\Delta_t^+(s) = Q_t^+(s, \pi^+(s)) - Q_t^*(s, \pi^+(s)) + Q_t^*(s, \pi^*(s)) - Q_t^+(s, \pi^*(s))$

However consider $Q_t^+(s, \pi^+(s)) - Q_t^+(s, \pi^*(s))$ (with the other term dealt with similarly).

On any sample path, the difference in these terms is bounded by the immediate reward plus the difference of the value at the next states. By problem definition, the difference in immediate rewards is bounded by one. However, consider the difference in value functions at the next state. Their state representation has a value of $\|(x - x')^+\|_1$ of at most 2 (for the two bins that were potentially modified). Hence, this difference is bounded by 3 in total. A similar argument for Q^* completes the proof. $\square$

9.7 Simulation Details

In this section we provide full details on the simulations conducted, including a formal description of virtual machine allocation scenarios along with its fidelity to the real-world cloud scenarios, training implementations, hyperparameter tuning results, and a description of the heuristic algorithms compared.

9.7.1 Online Bin-Packing

In a **Stochastic Online Bin Packing (OBP)** problem the agent has an infinite supply of bins of size B . Each round, items $u_t \in \{0, \dots, B\}$ arrive sampled iid from an unknown distribution. The agent either *packs* the item into an opened feasible bin or *opens* a new bin, with the goal to minimize the expected number of opened bins after T rounds.

9.3 describes how OBP are Exo-MDPs with u_t as the exogenous inputs. Lemma 3.1 from Freund and Banerjee (2019) shows that in stochastic OBP, $\Pr(\pi_t^+(S_t) \neq \pi_t^*(S_t)) \leq O(\frac{1}{t^2})$. Intuitively, as $t \rightarrow T$ there is little contribution from $\text{HINDSIGHT}(t, \xi_{>t})$ to the Q^+ of Equation 6 and so π^+ is more likely to coincide with π^* . Using a novel absolute bound on Δ and 5.6.2, we have:

Lemma 9.7.1. *In stochastic online bin packing with i.i.d. arrivals, $\sup_{t,s} \Delta_t^+(s) \leq O(1)$, independent of any problem primitives. Hence, $\text{REGRET}(\pi^+) \leq O(1)$.*

To numerically validate this claim, we use `ORSuite` (Archer et al., 2022) as an OBP simulator with $B = 5$ and vary T from 5 to 100. For these small problem sizes, we can compute π^* by solving Bellman equations exactly. The exogenous process $\mathcal{P}_\Xi$ is uniform: $u_t \sim \text{Unif}[B]$ and we generate $|\mathcal{D}| = 1000$ traces, and benchmark the resulting learned policy against π^* . The hindsight planner is represented with the integer program in 9.4.2 (solved efficiently by linear relaxation). Any learned policy maps a $B + 2$ -dim state (a vector denoting number of bins open with utilization from 1 to B , the current item and the remaining rounds) to a decision $A \in \{0, \dots, B - u_t\}$ to select a feasible bin (0 opens a new bin).

For each OBP problem with $T \in \{5, 10, 100\}$ we report in Table 9.2 the maximum hindsight bias $\text{MaxBias} = \mathbb{E}[\max_{s,t} \Delta_t^+(s)]$, where the expectation averages over 1000 sampled problem instances. We also report the percentage of problem instances where at least a single state

Table 9.2: Hindsight bias in OBP decreases as T increases. Thus, $\pi^\dagger$ becomes a better surrogate for $\pi^\star$.

T	MaxBias	% $\{\exists s : \pi_t^\star(s) \neq \pi_t^\dagger(s)\}$
5	1.240	6.8%
10	0.646	3.4%
100	0.066	0.3%

witnesses $\pi_t^\star(s) \neq \pi_t^\dagger(s)$. We see that as T increases, $\pi^\dagger$ becomes a good surrogate for $\pi^\star$ which bodes well for HL.

9.7.2 Multi-Secretary

Multi-secretary is the generalization of the classic secretary problem (Buchbinder et al., 2009), where T candidates arrive sequentially but only B can be selected. An arriving candidate at time t has ability $r_t \in (0, 1]$ drawn i.i.d. from a finite set of K levels of expertise. At each round, if the decision-maker has remaining budget (i.e., has chosen less than B candidates thus far), they can *accept* a candidate and collect the reward r_t , or *reject* the candidate. The goal is to maximize the expected cumulative reward. 9.3.2.3 shows how the multi-secretary problem can be formulated as an Exo-MDP.

We use $T = \{5, 10, 100\}$, $B = \frac{3}{5}T$, $K = 4$ for our experiments. With four expertise levels the corresponding abilities for the expertise levels was chosen to be $\{1/4, 1/2, 3/4, 1\}$. The arrival process for the ability types is non-stationary and sinusoidal with a type-dependent shift and frequency. Denoting p_j^t as the arrival probability of a type j customer at timestep t , the distribution is as follows. First, p_j^1 is chosen to be uniformly at random from $[0, 2\pi]$. Next, the frequency for each j is chosen to be uniformly from $[0, \pi/4]$. The final arrival probabilities p_j^t are then chosen to be sinusoidal with that shift and frequency value. These values are then

normalized appropriately to be a valid distribution.

In 5.2 we report the performance by evaluating each policy using dynamic programming with the true arrivals distribution. The Greedy heuristic accepts the first B candidates regardless of their quality. ML methods uses a single trace sampled from the non-stationary candidate arrival process, and use a policy that maps a 3-dim state (the rounds and budget remaining, and the current candidate ability) to an *accept* probability. For the hindsight planner, we use Equation 2 from Banerjee et al. (2020) which implements a linear program with $2N$ variables.

9.7.3 Airline Revenue Management

Airline Revenue Management (Littlewood, 1972) is a special case of the multi-dimensional Online Bin Packing (OBP) problem (recall that OBP exhibits vanishing hindsight bias via 9.7.1). The agent has capacity B_k for K different resources. At each round, the decision-maker observes a request $A_t \in \mathbb{R}_+^K$ (the consumed capacity in each resource dimension), alongside a revenue f_t . The algorithm can either *accept* the request (obtaining revenue f_t and updating remaining capacity according to A_t), or *reject* it (note that partial acceptance is not allowed). The goal of the decision-maker is to maximize the expected revenue.

We use ORSuite (Archer et al., 2022) as an ARM simulator with fixed capacity, iid. request types and job distribution. We use $T = \{5, 10, 100\}$, $K = 3$, and 2 request types. The starting capacity for the three resources set to be $[8, 4, 4]$. The iid arrival distribution is $(1/3, 1/3, 1/3)$ (where the last category corresponds to no arrival). Job one arrivals have resource requests $[2, 3, 2]$ with revenue 1, and job two arrivals have resource requests $[3, 0, 1]$ with revenue 2. This setting satisfies a dual-degeneracy condition of the hindsight planner from Vera and Banerjee (2021)

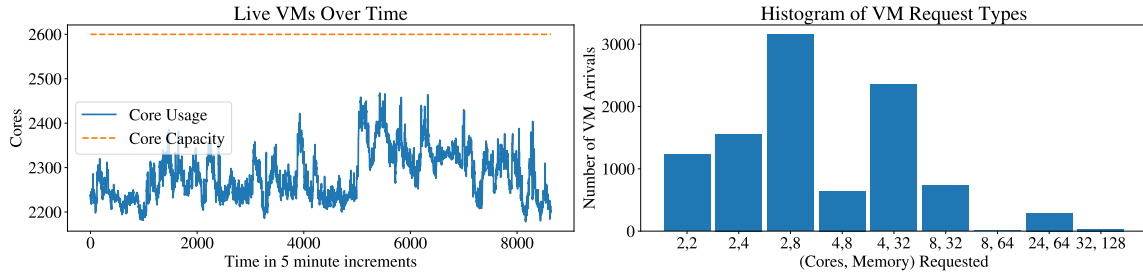


Figure 9.2: Sample of a thirty-day roll-out from the Azure Public Dataset. In ?? we show a histogram of the various VM types and their corresponding cores and memory resources requested. In ?? we plot the used cores over time on the observed trace, along with the capacity of the 80-node cluster simulated using MARO.

which shows large regret for existing heuristics on these problems.

The optimal policy is computed through dynamic programming. Both RL (Tabular Q-learning) and HL (Hindsight MAC) were trained on the same dataset, which contained 100 traces. In 5.2 we report the performance of the policies through Monte Carlo simulations averaged over 500 iterations.

9.7.4 Virtual Machine Allocation

Cloud computing has revolutionized the way that computing resources are consumed. These providers give end-users easy access to state-of-the-art resources. One of the most crucial components to cloud computing providers is the Virtual Machine (VM) allocator, which assigns specific VM requests to physical hardware. Improper placement of VM requests to physical machines (henceforth referred to as PMs) can cause performance impact, service delays, and create allocation failures. The VMs serve as the primary units of resource allocation in these models. We focus on designing allocation policies at the *cluster* level, which are a homogeneous set of physical machines with the same memory and CPU cores capacity.

The cluster-specific allocator is tasked with the following:

- Coming VM requests ask for a certain amount of resources (CPU and memory requirements) along with their lifetime. Resource requirements are varied based on the different VM requests.
- Based on the action selected by the allocator policy, the VM will be allocated to and be created in a specified PM as long as that PM's remaining resources are enough.
- After a period of execution, the VM completes its tasks. The simulator will then release the resources allocated to this VM and de-allocate this VM from the PM.

9.7.5 Stylized Environment

We use MARO, an open source package for multi-agent reinforcement learning in operations research tasks as a simulator for the VM allocator ([Jiang et al., 2020b](#)). In this scenario the VM requests are uniformly sampled from the 2019 snapshot of the Azure Public Dataset ([Cortez et al., 2017](#)). The cluster is a fictitious one consisting of 80 PMs that we found were similarly over-provisioned as in real-world clusters. See ?? to highlight the demand workload against the cluster capacity for our experiment setup.

By default, MARO provides reward metrics that can be used when specifying the objective of the algorithm. The metrics provided include income, energy consumption, profit, number of successful and failed allocations, latency, and total number of overloaded physical machines. However, typical cloud computing systems run in an *over-provisioned* regime where the capacity of the physical machines is larger than the demand in order to ensure quality of service to its customers. As a result, any reasonable algorithm has no failed allocations. Hence, any reasonable algorithm also has identical values for income, energy consumption, profit, etc.

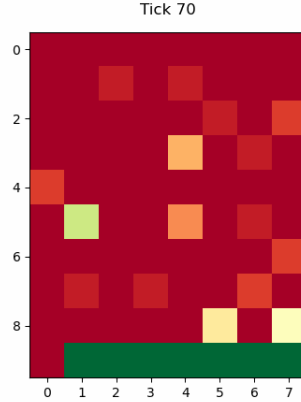


Figure 9.3: Packing density for the Best Fit policy at one time-step. Each square corresponds to a specific PM and the colour corresponds to what portion of that PMs capacity is currently utilized. Red corresponds to fully used, green is completely empty. The packing density ignores the empty (or green) PMs on the bottom and counts the cumulative utilization ratio for the remaining PMs.

For the reward function we instead consider $r(s, a, \xi) = -100 * \text{Failed-Allocation} - 1 / \text{Packing Density}$. The first component, one hundred times the number of failed allocations, helps to penalize the algorithms in training to ensure valid assignments for all of the VM requests. The second component corresponds to the `Packing-Density`, computed via:

$$\text{Packing-Density} = \frac{\sum_{v \in VM} \text{Cores Usage}_v}{\sum_{p \in PM} \mathbb{1}_{[p \text{ is utilized}]} \text{Capacity}_p}$$

The numerator here is the total cumulative cores used for all of the VMs currently assigned on the system. The denominator is the total capacity of all physical machines which are currently utilized (i.e. have a VM assigned and currently running). The reason for picking this reward (and the inverse of it) is that:

- It allows for an easily expressible linear programming formulation (see [9.4.5](#)).
- For any two policies which allocate all virtual machines, packing density serves as a criteria to differentiate the policies. An algorithm which has large packing density equivalently uses the

physical machines efficiently so that unused PMs can be re-purposed, reassigned, or potentially turned off.

- It serves as a proxy to ensure that the virtual machines are packed in as minimal number of physical machines possible. This allows the allocator to be robust to hardware failures, where entire physical machines are potentially rendered unusable.

See [9.3](#) for an illustration.

9.7.5.1 Simulator Fidelity

Our training procedure requires a faithful simulator of Virtual Machine allocation focused at a cluster level to validate our experimental results. We found that the MARO simulator captures all first-order effects of cloud computing environments specifically at the cluster level. However, there are several effects not included in the simulations:

- When a virtual machine arrives to the system, they request a *maximum* amount of CPU and memory capacity that they can use. However, over time, any given VM request might only use a fraction of their requested resources. Current cloud computing systems use an *over-subscription* model where the requested memory and CPU cores for the VMs assigned to a PM can surpass its capacity. However, when the total realized demand surpasses the PMs capacity, all of the VMs assigned to that system are failed and migrated to a different PM. In contrast, MARO assumes that each VM uses exactly its requested cores and memory over time, hence eliminating the need to model over-subscription on the cluster level.
- Typical systems involve live migration where virtual machines can be moved between physical machines without disrupting the VM request. This is used in order to eliminate stranding which

Algorithm 2 Training Procedure in MARO

```
1: Input: number of roll-outs, number of actors, number of training iterations.
2: for each roll-out do
3:   For each actor, sample a random duration uniformly at random, and execute the Best Fit
     heuristic on a historical dataset of that length starting from an empty cluster
4:   For each actor, sample a one-hour trace of VM requests  $\xi^i$ 
5:   for each actor do
6:     Collect dataset of  $(s_t, a_t, \xi_t, r_t, s_{t+1})$  pairs under the current policy  $\pi_\theta$ 
7:   end for
8:   Add collected dataset to current experience buffer
9:   for each training iteration do
10:    Sub-sample batch from current collected dataset
11:    Update policy by gradient descent along the sampled batch
12:   end for
13: end for
```

occurs when a physical machine has only a few long-running virtual machines allocated to it.

However, such an operation is costly and requires a large amount of system overhead.

- Our neural networks explicitly use the VM’s lifetime as a feature in the state. However, in true cloud computing systems the lifetime is unknown. Only when a user decides to cancel a VM does the system have access to that information. As such, it is typically observed in the trace dataset but cannot be used in policy network design. We forgo this when modelling the policy as unlike the dynamics of VM request types, lifetimes for a VM are typically easy to model and these forecasts can be used as a replacement in the network representation ([Hadary et al., 2020](#)).

We believe that MARO serves as a high fidelity simulator of the VM allocation problem at the cluster level while providing open source implementation for additional research experimentation. However, we complement these results in [5.7.3.2](#) with a real-world model of cloud computing platforms.

9.7.5.2 Training Implementation Details

[2](#) presents the pseudocode for our training procedure using MARO. We repeat the following process for five hundred roll-outs. First, we created realistic starting state distributions by executing the *BestFit* heuristic for a random duration (greater than one day). Line 4 samples one-hour traces of VM requests from the 2019 historical Microsoft Azure dataset. Then, in line five and six, with fifteen actors in parallel we evaluate the current policy π_θ on the sampled VM request trace, adding the dataset of experience to the experience buffer. Lines 9-11 samples batches of size 256 where we update the policy and algorithmic parameters θ by gradient descent on the loss function evaluated on the sampled batch. This process repeats for five thousand gradient steps.

We implemented the training framework using PyTorch ([Paszke et al., 2019](#)) along with MARO ([Jiang et al., 2020b](#)), and all experiments were conducted using the Microsoft Azure ML training platform. For hyperparameters and neural network architectures for the Sim2Real RL and Hindsight Learning algorithms, see [9.7.5.4](#). All experiments were run on the same compute hardware and took similar runtimes to finish. Runtime was dominated by the MARO simulator executing the roll-outs under the current policy versus the HINDSIGHT calls or the ML model updates. As such, each algorithm was essentially given the same computational budget.

To evaluate the trained policies we subdivided the Azure Public Dataset into a temporally contiguous and non-overlapping training (first 15 days) and test (last 15 days) portions. For evaluation we sampled fifty different one-day traces of VM requests from the held-out portion. For each of the different traces, we executed the policy in parallel to a greedy **Best Fit** algorithm. Each deep learning algorithm was evaluated over five different random seed initializations and we tuned hyperparameters using grid search. All metrics are reported with respect to cumulative

differences against the baseline **Best Fit** policy, alongside statistical significance tests. In 5.2 we evaluate the number of machines required to pack the jobs. Negative numbers correspond to fewer required PMs on average. Asterisks correspond to statistical significance computed with Welch’s t -test with $p = 0.05$. In 9.4 we provide the performance metrics on the underlying rewards as well.

9.7.5.3 Heuristic Algorithms

- **Random:** Picks a physical machine uniformly at random from the list of physical machines which have capacity to service the current VM request.
- **Round Robin:** Allocates to physical machines in a round-robin strategy by selecting a physical machine from the list of physical machines which have capacity to service the current VM request that was least recently used.
- **Best Fit:** This algorithm picks a physical machine based on a variety of metric types.
 - **Remaining Cores:** Picks a valid physical machine with minimum remaining cores
 - **Remaining Memory:** Picks a valid physical machine with minimum remaining memory
 - **Energy Consumption:** Picks the valid physical machine with maximum energy consumption
 - **Remaining Cores and Energy Consumption:** Picks a valid physical machine with minimum remaining cores, breaking ties via energy consumption

Similar heuristics to this are currently used in most large-scale cloud computing systems (Hadary et al., 2020).

- **Bin Packing:** Selects a valid physical machine which minimizes the resulting variance on the number of virtual machines on each physical machine.

9.7.5.4 Sim2Real RL Algorithms

We also compared our Hindsight Learning approaches to existing Sim2Real RL algorithms in the literature with custom implementation built on top of the MARO package. These include:

- **Deep Q Network (DQN):** Double Q -Learning algorithm from [Van Hasselt et al. \(2016\)](#).
- **Actor Critic (AC):** Actor Critic algorithm implementation from [Konda and Tsitsiklis \(2000\)](#).
- **Mean Actor Critic (MAC):** A modification of the actor critic algorithm where the actor loss is calculated along all actions instead of just the selected actions ([Allen et al., 2017](#)).
- **Policy Gradient (VPG):** A modification of the vanilla policy gradient with a exogenous input dependent baseline from [Mao et al. \(2019b\)](#). Instead of training a baseline explicitly, we use $Q_t^{\text{Best Fit}}(s, a, \xi_{\geq t})$.

9.7.5.5 State Features, Network Architecture, and Hyperparameters

The state space of the VM allocation scenario is combinatorial as we need to include the CPU and memory utilization of each physical machine across time to account for the lifetimes of the VMs currently assigned to the PM. To rectify this, for each of the deep RL algorithms we use action-dependent features when representing the state space. In particular, once a VM request arrives, we consider the set of physical machines that are available to service this particular virtual machine. Each (PM, VM) pair has associated state features, including:

- The VM’s CPU cores requirement, memory requirement, and lifetime
- The PM’s CPU cores capacity, memory capacity, and type
- The historical CPU cores allocated, utilization, energy consumption, and memory allocated over last three VM requests

The last component is serving as a proxy for the historical utilization of the PM across all time to account for all VMs currently assigned to the PM. The final action dependent features corresponds to the concatenation of these state features for each valid PM to service the current request.

We note that to use action-dependent features some of the algorithms required slight tweaking to their implementations. In particular, when considering algorithms using a policy network representation (i.e. policy gradient, actor critic, or mean actor critic) when executing the policy we take $\pi_\theta = \text{Softmax}(\pi(s, a_1), \dots, \pi(s, a_N))$ where $a_1, \dots, a_N$ is the set of physical machines that can service the current request and (s, a_i) is the corresponding state-features for the physical machine a_i .

For the actor and critic network representations in all algorithms we use a four layer neural network with (32, 16, 8) hidden dimensions, an output dimension of one (due to the action-dependent features), and LeakyReLU activation functions. For each of the algorithms we use the *RMSprop* optimization algorithm. We implemented the training framework using PyTorch (Paszke et al., 2019) and MARO. All experiments were run on the same compute hardware and took similar runtimes to finish. Runtime was dominated by the MARO simulator executing the roll-outs, and the ML model updates and HINDSIGHT oracle calls were quicker.

Lastly we provide a list of the hyperparameters used and which algorithm they apply to

when tuning algorithm performance.

Table 9.3: List of hyperparameters tuned over for the Sim2Real RL and Hindsight Learning algorithms.

Hyperparameter	Algorithm	Values
Discount Factor	DQN, AC, MAC, PG	0.9, 0.95, 0.99, 0.999
Learning Rate	All Algorithms	0.05, 0.005, 0.0005, 0.00005, 0.000005
Entropy Regularization	PG, AC, MAC	0, 0.1, 1, 10
Actor Loss Coefficient	AC, MAC	0.1, 1, 10, 100
Target Update Smoothing Parameter	DQN	0.0001, 0.001, 0.01, 0.1

For concrete parameters evaluated and experiment results, see the attached code-base.

9.7.5.6 Training Performance

In 9.4 we include a plot of the loss curves for the various algorithms.

9.7.5.7 Testing Performance

In 9.4 we plot the performance of the algorithms on the true reward function. Here we include the true reward considered:

$$r(s, a, \xi) = -100 * \text{Failed-Allocation} - 1/\text{Packing Density}(s)$$

as well as simply $\text{Packing Density}(s)$. All measures are reported with a 95% confidence interval, and computed as differences against the **Best Fit** allocation policy.

Table 9.4: Performance of heuristics, Sim2Real RL, and Hindsight Learning algorithms on VM allocation. Here we include the true reward metric the algorithms were trained on (negative inverse of the packing density) and the packing density improvements on average.

Algorithm	Performance $r = -1/\text{Packing Density}$	Packing Density
Performance Upper Bound	0.66 ± 0.29	$.09\% \pm 0.03\%$
Best Fit	0.0	0.0
Bin Packing	-64.44 ± 2.49	$-5.34\% \pm 0.2\%$
Round Robin	-56.36 ± 2.65	$-4.67\% \pm 0.22\%$
Random	-48.94 ± 2.33	$-4.08\% \pm 0.19\%$
DQN	-1.00 ± 0.41	$-0.05\% \pm 0.04\%$
MAC	-0.38 ± 0.033	$-0.03\% \pm 0.00\%$
AC	-2.94 ± 0.61	$-0.21\% \pm 0.06\%$
Policy Gradient	-1.03 ± 0.39	$-0.06\% \pm 0.04\%$
HINDSIGHT MAC	0.18 ± 0.093	$0.05\% \pm 0.00\%$
HINDSIGHT Q-DISTILLATION	0.08 ± 0.32	$0.04\% \pm 0.029\%$

9.7.6 Real-World VM Allocation

In this simulation we consider the more realistic setting of VM arrivals in continuous time and where the allocation agent has no information about VM lifetimes. We avoid giving a concrete description of the reward function trained, cluster sizes, etc. to preserve the confidentiality of the cloud operator. However, we briefly describe the training implementation details, heuristic algorithms, as well as the hindsight heuristic used.

9.7.6.1 Heuristic Algorithms

[5.3](#) summarizes the results of performance for three different algorithms: a). BestFit: its performance is shown relative to a **BestFit** strategy used in production. This strategy follows a proprietary implementation that prioritizes between CPU and memory depending on their scarcity; b). Hindsight Learning and Sim2Real RL: we adapt HINDSIGHT MAC (HL) and com-

pare it with MAC [Allen et al. \(2017\)](#) (RL), where both used the same network architecture, which embeds VM-specific and PM-specific features using a 6-layer GNN. The resulting architecture is rich enough to represent the **BestFit** heuristic, but can also express more flexible policies.

9.7.6.2 Training Implementation Details

We trained each algorithm over 3 random seeds and evaluated 5 rollouts to capture the variation in the cluster state at the start of the evaluation trace. Unlike the other experiments, we cannot account for the randomness in exogenous samples because we only have one evaluation trace for each cluster. Error metrics are computed with a paired t -test of value $p = 0.05$.

9.7.6.3 Hindsight Heuristic

Due to the large scale of the real-world scenarios, even the linear relaxation of the integer program was not tractable. Consequently, we resort here to using a carefully designed *hindsight heuristic* (3) to derive $\text{HINDSIGHT}(t, \xi, s)$. The heuristic is based on prioritizing VMs according to both their size and duration. See 3 for pseudocode.

In 9.5 we separately tested the accuracy of our heuristic for the hindsight planner (both by comparing to the optimal in small instances, as well as by comparing to a lower bound given in [Buchbinder et al. \(2021\)](#)), and concluded that the heuristic obtains a value that is within few percentages of the optimum. We found that the dual gap of 3 was typically within 4% of the optimum.

Algorithm 3 Hindsight Heuristic.

- 1: **Input:** A cluster state s , sequence of VM requests $\xi_{t:T}$.
 - 2: Sort requests in descending order of their lifetimes
 - 3: **for** Each request ξ **do**
 - 4: Allocate to the feasible PM where ξ is the only live VM on it for the least amount of time
 - 5: **end for**
-

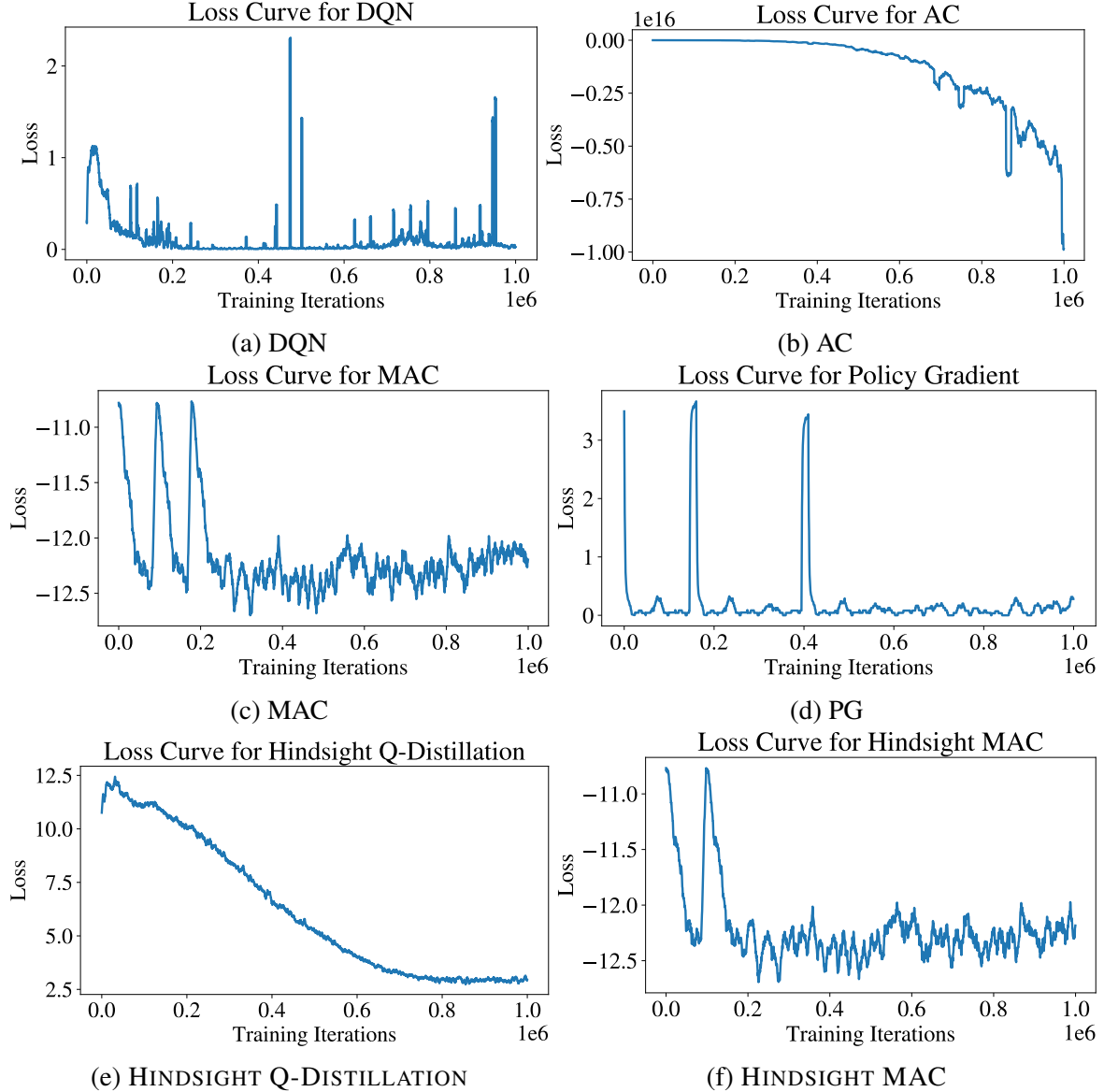


Figure 9.4: Moving average of the loss curves for the Sim2Real RL and Hindsight Learning algorithms over the one million gradient steps computed in each of the experiments. Note that some of the volatility occurs when an algorithm observes a data point with a failed allocation as there is a large penalty.

Table 9.5: How close to optimal is the Upper Bound (Oracle)? We measure the average UsedPMs of Oracle’s solution and compare it with the lower bound which assumes that VMs can be fractionally split across PMs.

Cluster	Upper Bound (Oracle)	Lower Bound	Gap (%)
A	467.42	499.625	6.89%
B	538.35	578.214	7.41%
C	383.19	391.329	2.12%
D	27.47	32.9769	20.07%
E	448.86	475.968	6.04%
F	577.97	625.083	8.15%
G	2252.65	2287.21	1.53%
H	2295.19	2332.31	1.62%
I	341.90	361.654	5.78%
J	1212.91	1239.9	2.23%
K	565.34	570.532	0.92%
L	8.23	8.85826	7.64%
M	8.77	9.3152	6.25%
N	305.10	310.096	1.64%
O	43.27	45.3596	4.82%
P	2528.72	2588.38	2.36%
Q	1457.37	1481.13	1.63%
R	123.04	124.866	1.49%
S	2452.25	2491.68	1.61%
T	68.68	70.0956	2.07%
U	533.61	539.872	1.17%
V	1243.70	1260.14	1.32%
W	1678.88	1702.09	1.38%
X	158.71	171.058	7.78%
ALL			4.33%

Bibliography

- Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Li Fei-Fei, C Lawrence Zitnick, and Ross Girshick. Clevr: A diagnostic dataset for compositional language and elementary visual reasoning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2901–2910, 2017a.
- Jason Weston, Antoine Bordes, Sumit Chopra, Alexander M Rush, Bart van Merriënboer, Armand Joulin, and Tomas Mikolov. Towards ai-complete question answering: A set of prerequisite toy tasks. *arXiv preprint arXiv:1502.05698*, 2015.
- Ronghang Hu, Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Kate Saenko. Learning to reason: End-to-end module networks for visual question answering. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 804–813, 2017a.
- François Fleuret, Ting Li, Charles Dubout, Emma K Wampler, Steven Yantis, and Donald Geman. Comparing machines and humans on a visual categorization test. *Proceedings of the National Academy of Sciences*, 108(43):17621–17625, 2011.
- Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C Lawrence Zitnick, and Devi Parikh. Vqa: Visual question answering. In *Proceedings of the IEEE international conference on computer vision*, pages 2425–2433, 2015.
- Jiaxian Guo, Junnan Li, Dongxu Li, Anthony Meng Huat Tiong, Boyang Li, Dacheng Tao, and Steven CH Hoi. From images to textual prompts: Zero-shot vqa with frozen large language models. *arXiv preprint arXiv:2212.10846*, 2022.
- Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, et al. Interaction networks for learning about objects, relations and physics. In *Advances in Neural Information Processing Systems*, pages 4502–4510, 2016.
- Nicholas Watters, Daniel Zoran, Theophane Weber, Peter Battaglia, Razvan Pascanu, and Andrea Tacchetti. Visual interaction networks: Learning a physics simulator from video. In *Advances in neural information processing systems*, pages 4539–4547, 2017.
- Katerina Fragkiadaki, Pulkit Agrawal, Sergey Levine, and Jitendra Malik. Learning visual predictive models of physics for playing billiards. In *International Conference on Learning Representations*, 2016.
- Michael B Chang, Tomer Ullman, Antonio Torralba, and Joshua B Tenenbaum. A compositional object-based approach to learning physical dynamics. In *International Conference on Learning Representations*, 2017a.

- Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter Battaglia. Learning to simulate complex physics with graph networks. In *International conference on machine learning*, pages 8459–8468. PMLR, 2020.
- David Saxton, Edward Grefenstette, Felix Hill, and Pushmeet Kohli. Analysing mathematical reasoning abilities of neural models. In *International Conference on Learning Representations*, 2019.
- Michael Chang, Abhishek Gupta, Sergey Levine, and Thomas L. Griffiths. Automatically composing representation transformations as a means for generalization. In *International Conference on Learning Representations*, 2019.
- Pan Lu, Liang Qiu, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, Tanmay Rajpurohit, Peter Clark, and Ashwin Kalyan. Dynamic prompt learning via policy gradient for semi-structured mathematical reasoning. *arXiv preprint arXiv:2209.14610*, 2022a.
- Shima Imani, Liang Du, and Harsh Shrivastava. Mathprompter: Mathematical reasoning using large language models. *arXiv preprint arXiv:2303.05398*, 2023.
- Shaohan Huang, Li Dong, Wenhui Wang, Yaru Hao, Saksham Singhal, Shuming Ma, Tengchao Lv, Lei Cui, Owais Khan Mohammed, Qiang Liu, et al. Language is not all you need: Aligning perception with language models. *arXiv preprint arXiv:2302.14045*, 2023.
- Adam Santoro, Felix Hill, David Barrett, Ari Morcos, and Timothy Lillicrap. Measuring abstract reasoning in neural networks. In *International Conference on Machine Learning*, pages 4477–4486, 2018.
- Chi Zhang, Feng Gao, Baoxiong Jia, Yixin Zhu, and Song-Chun Zhu. Raven: A dataset for relational and analogical visual reasoning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5317–5327, 2019a.
- Richard Bellman. Dynamic programming. *Science*, 153(3731):34–37, 1966.
- Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.
- Rasmus Palm, Ulrich Paquet, and Ole Winther. Recurrent relational networks. In *Advances in Neural Information Processing Systems*, pages 3368–3378, 2018.
- Damian Mrowca, Chengxu Zhuang, Elias Wang, Nick Haber, Li F Fei-Fei, Josh Tenenbaum, and Daniel L Yamins. Flexible neural representation for physics prediction. In *Advances in Neural Information Processing Systems*, pages 8799–8810, 2018.
- Alvaro Sanchez-Gonzalez, Nicolas Heess, Jost Tobias Springenberg, Josh Merel, Martin Riedmiller, Raia Hadsell, and Peter Battaglia. Graph networks as learnable physics engines for inference and control. In *International Conference on Machine Learning*, pages 4467–4476, 2018.

- Michael Janner, Sergey Levine, William T. Freeman, Joshua B. Tenenbaum, Chelsea Finn, and Jiajun Wu. Reasoning about physical interactions with object-centric models. In *International Conference on Learning Representations*, 2019.
- Leslie G Valiant. A theory of the learnable. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 436–445. ACM, 1984.
- Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009a.
- Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019.
- Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. In *International Conference on Machine Learning*, pages 5453–5462, 2018.
- Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International Conference on Machine Learning*, pages 1273–1272, 2017.
- Adam Santoro, David Raposo, David G Barrett, Mateusz Malinowski, Razvan Pascanu, Peter Battaglia, and Timothy Lillicrap. A simple neural network module for relational reasoning. In *Advances in neural information processing systems*, pages 4967–4976, 2017.
- Kexin Yi, Jiajun Wu, Chuang Gan, Antonio Torralba, Pushmeet Kohli, and Josh Tenenbaum. Neural-symbolic vqa: Disentangling reasoning from vision and language understanding. In *Advances in Neural Information Processing Systems*, pages 1031–1042, 2018.
- Jiayuan Mao, Chuang Gan, Pushmeet Kohli, Joshua B. Tenenbaum, and Jiajun Wu. The neuro-symbolic concept learner: Interpreting scenes, words, and sentences from natural supervision. In *International Conference on Learning Representations*, 2019a.
- Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Judy Hoffman, Li Fei-Fei, C Lawrence Zitnick, and Ross Girshick. Inferring and executing programs for visual reasoning. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2989–2998, 2017b.
- Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Ruslan R Salakhutdinov, and Alexander J Smola. Deep sets. In *Advances in Neural Information Processing Systems*, pages 3391–3401, 2017.
- Vladimir Vapnik. *The nature of statistical learning theory*. Springer science & business media, 2013.
- Peter L Bartlett and Shahr Mendelson. Rademacher and gaussian complexities: Risk bounds and structural results. *Journal of Machine Learning Research*, 3(Nov):463–482, 2002.

- Peter L Bartlett, Dylan J Foster, and Matus J Telgarsky. Spectrally-normalized margin bounds for neural networks. In *Advances in Neural Information Processing Systems*, pages 6240–6249, 2017.
- Behnam Neyshabur, Ryota Tomioka, and Nathan Srebro. Norm-based capacity control in neural networks. In *Conference on Learning Theory*, pages 1376–1401, 2015.
- Noah Golowich, Alexander Rakhlin, and Ohad Shamir. Size-independent sample complexity of neural networks. In *Conference On Learning Theory*, pages 297–299, 2018.
- Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.
- Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.
- Jorge Pérez, Javier Marinković, and Pablo Barceló. On the turing completeness of modern neural network architectures. In *International Conference on Learning Representations*, 2019.
- Edward Wagstaff, Fabian B Fuchs, Martin Engelcke, Ingmar Posner, and Michael Osborne. On the limitations of representing functions on sets. In *International Conference on Machine Learning*, 2019.
- Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. Computational capabilities of graph neural networks. *IEEE Transactions on Neural Networks*, 20(1):81–102, 2009b.
- Andrei N Kolmogorov. On tables of random numbers. *Theoretical Computer Science*, 207(2): 387–395, 1998.
- Zeyuan Allen-Zhu, Yuanzhi Li, and Yingyu Liang. Learning and generalization in overparameterized neural networks, going beyond two layers. In *Advances in Neural Information Processing Systems*, pages 6155–6166, 2019a.
- Sanjeev Arora, Simon Du, Wei Hu, Zhiyuan Li, and Ruosong Wang. Fine-grained analysis of optimization and generalization for overparameterized two-layer neural networks. In *International Conference on Machine Learning*, pages 322–332, 2019a.
- Sanjeev Arora, Simon S Du, Wei Hu, Zhiyuan Li, Russ R Salakhutdinov, and Ruosong Wang. On exact computation with an infinitely wide neural net. In *Advances in Neural Information Processing Systems*, pages 8139–8148, 2019b.
- 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*, 2020.
- Simon S. Du, Xiyu Zhai, Barnabas Poczos, and Aarti Singh. Gradient descent provably optimizes over-parameterized neural networks. In *International Conference on Learning Representations*, 2019a.

- Simon Du, Jason Lee, Haochuan Li, Liwei Wang, and Xiyu Zhai. Gradient descent finds global minima of deep neural networks. In *International Conference on Machine Learning*, pages 1675–1685, 2019b.
- Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in neural information processing systems*, pages 8571–8580, 2018.
- Yuanzhi Li and Yingyu Liang. Learning overparameterized neural networks via stochastic gradient descent on structured data. In *Advances in Neural Information Processing Systems*, pages 8157–8166, 2018.
- Simon S Du, Kangcheng Hou, Russ R Salakhutdinov, Barnabas Poczos, Ruosong Wang, and Keyulu Xu. Graph neural tangent kernel: Fusing graph neural networks with graph kernels. In *Advances in Neural Information Processing Systems*, pages 5724–5734, 2019c.
- Richard Bellman. On a routing problem. *Quarterly of applied mathematics*, 16(1):87–90, 1958.
- Jos Thijssen. *Computational physics*. Cambridge university press, 2007.
- Paul L DeVries and Patrick Hamill. A first course in computational physics, 1995.
- Richard M Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.
- Vikas Garg, Stefanie Jegelka, and Tommi Jaakkola. Generalization and representational limits of graph neural networks. In *International Conference on Machine Learning*, pages 3419–3430. PMLR, 2020.
- Quentin Cappart, Didier Chételat, Elias B Khalil, Andrea Lodi, Christopher Morris, and Petar Velickovic. Combinatorial optimization and reasoning with graph neural networks. *J. Mach. Learn. Res.*, 24:130–1, 2023.
- Andreea-Ioana Deac, Petar Veličković, Ognjen Milinkovic, Pierre-Luc Bacon, Jian Tang, and Mladen Nikolic. Neural algorithmic reasoners are implicit planners. *Advances in Neural Information Processing Systems*, 34, 2021.
- Andrew J Dudzik and Petar Veličković. Graph neural networks are dynamic programmers. *Advances in Neural Information Processing Systems*, 35:20635–20647, 2022.
- Brenden M Lake, Tomer D Ullman, Joshua B Tenenbaum, and Samuel J Gershman. Building machines that learn and think like people. *Behavioral and brain sciences*, 40, 2017.
- Etienne Barnard and LFA Wessels. Extrapolation and interpolation in neural network classifiers. *IEEE Control Systems Magazine*, 12(5):50–53, 1992.
- Pamela J Haley and DONALD Soloway. Extrapolation limitations of multilayer feedforward neural networks. In *International Joint Conference on Neural Networks*, volume 4, pages 25–30. IEEE, 1992.

- Petar Velickovic, Rex Ying, Matilde Padovano, Raia Hadsell, and Charles Blundell. Neural execution of graph algorithms. In *Int. Conf. on Learning Representations*, 2020.
- Guillaume Lample and François Charton. Deep learning for symbolic mathematics. In *International Conference on Learning Representations*, 2020.
- Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. In *International Conference on Learning Representations*, 2017a.
- George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.
- Ken-Ichi Funahashi. On the approximate realization of continuous mappings by neural networks. *Neural networks*, 2(3):183–192, 1989.
- Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.
- Věra Kůrková. Kolmogorov’s theorem and multilayer neural networks. *Neural networks*, 5(3):501–506, 1992.
- Moritz Hardt, Ben Recht, and Yoram Singer. Train faster, generalize better: Stability of stochastic gradient descent. In *International Conference on Machine Learning*, 2016.
- Daniel Soudry, Elad Hoffer, Mor Shpigel Nacson, Suriya Gunasekar, and Nathan Srebro. The implicit bias of gradient descent on separable data. *The Journal of Machine Learning Research*, 19(1), 2018.
- Raman Arora, Amitabh Basu, Poorya Mianjy, and Anirbit Mukherjee. Understanding deep neural networks with rectified linear units. In *International Conference on Learning Representations*, 2018.
- Boris Hanin and David Rolnick. Complexity of linear regions in deep networks. In *International Conference on Machine Learning*, 2019.
- Matthias Hein, Maksym Andriushchenko, and Julian Bitterwolf. Why relu networks yield high-confidence predictions far away from the training data and how to mitigate the problem. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 41–50, 2019.
- Alberto Bietti and Julien Mairal. On the inductive bias of neural tangent kernels. In *Advances in Neural Information Processing Systems*, 2019.
- Lénaïc Chizat and Francis Bach. On the global convergence of gradient descent for over-parameterized models using optimal transport. *Advances in Neural Information Processing Systems*, 31:3036–3046, 2018.
- Mei Song, Andrea Montanari, and P Nguyen. A mean field view of the landscape of two-layers neural networks. *Proceedings of the National Academy of Sciences*, 115:E7665–E7671, 2018a.

- Hartmut Maennel, Olivier Bousquet, and Sylvain Gelly. Gradient Descent Quantizes ReLU Network Features. *arXiv e-prints*, art. arXiv:1803.08367, March 2018.
- Pedro Savarese, Itay Evron, Daniel Soudry, and Nathan Srebro. How do infinite width bounded norm networks look in function space? In *Conference on Learning Theory*, 2019.
- Francis Williams, Matthew Trager, Daniele Panozzo, Claudio Silva, Denis Zorin, and Joan Bruna. Gradient dynamics of shallow univariate relu networks. In *Advances in Neural Information Processing Systems*, pages 8376–8385, 2019.
- Jimmy Ba, Murat Erdogdu, Taiji Suzuki, Denny Wu, and Tianzong Zhang. Generalization of two-layer neural networks: An asymptotic viewpoint. In *International Conference on Learning Representations*, 2020.
- Behrooz Ghorbani, Song Mei, Theodor Misiakiewicz, and Andrea Montanari. Linearized two-layers neural networks in high dimension. *arXiv preprint arXiv:1904.12191*, 2019.
- Zeyuan Allen-Zhu, Yuanzhi Li, and Zhao Song. A convergence theory for deep learning via over-parameterization. In *International Conference on Machine Learning*, 2019b.
- Yuan Cao and Quanquan Gu. Generalization bounds of stochastic gradient descent for wide and deep neural networks. In *Advances in Neural Information Processing Systems*, pages 10835–10845, 2019.
- Atsushi Nitanda and Taiji Suzuki. Optimal rates for averaged stochastic gradient descent under neural tangent kernel regime. In *International Conference on Learning Representations*, 2021.
- Martin Arjovsky, Léon Bottou, Ishaan Gulrajani, and David Lopez-Paz. Invariant risk minimization. *arXiv preprint arXiv:1907.02893*, 2019.
- Jonas Peters, Peter Bühlmann, and Nicolai Meinshausen. Causal inference by using invariant prediction: identification and confidence intervals. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 78(5):947–1012, 2016.
- Mateo Rojas-Carulla, Bernhard Schölkopf, Richard Turner, and Jonas Peters. Invariant models for causal transfer learning. *The Journal of Machine Learning Research*, 19(1):1309–1342, 2018.
- Alan Lapedes and Robert Farber. Nonlinear signal processing using neural networks: Prediction and system modelling. Technical report, 1987.
- David B McCaughan. On the properties of periodic perceptrons. In *International Conference on Neural Networks*, 1997.
- JM Sopena and R Alquezar. Improvement of learning in recurrent networks by substituting the sigmoid activation function. In *International Conference on Artificial Neural Networks*, pages 417–420. Springer, 1994.
- Simon S. Du and Jason D. Lee. On the power of over-parametrization in neural networks with quadratic activation. In *International Conference on Machine Learning*, 2018.

- Roi Livni, Shai Shalev-Shwartz, and Ohad Shamir. On the computational efficiency of training neural networks. In *Advances in neural information processing systems*, 2014.
- Gabriele Corso, Luca Cavalleri, Dominique Beaini, Pietro Liò, and Petar Veličković. Principal neighbourhood aggregation for graph nets. *Advances in Neural Information Processing Systems*, 2020.
- Andrew Trask, Felix Hill, Scott E Reed, Jack Rae, Chris Dyer, and Phil Blunsom. Neural arithmetic logic units. In *Advances in Neural Information Processing Systems*, 2018.
- Andreas Madsen and Alexander Rosenberg Johansen. Neural arithmetic units. In *International Conference on Learning Representations*, 2020.
- Taylor Webb, Zachary Dulberg, Steven Frankland, Alexander Petrov, Randall O’Reilly, and Jonathan Cohen. Learning representations that support extrapolation. In *International Conference on Machine Learning*, 2020.
- Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International Conference on Machine Learning*, 2020.
- Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *North American Chapter of the Association for Computational Linguistics*, 2019.
- Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. Strategies for pre-training graph neural networks. In *International Conference on Learning Representations*, 2020a.
- Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, 2013a.
- Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In *North American Chapter of the Association for Computational Linguistics*, 2018.
- Tomas Mikolov, Quoc V Le, and Ilya Sutskever. Exploiting similarities among languages for machine translation. *arXiv preprint arXiv:1309.4168*, 2013b.
- Shijie Wu and Mark Dredze. Beto, bentz, becas: The surprising cross-lingual effectiveness of bert. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 833–844, 2019.
- Michelle Yuan, Mozhi Zhang, Benjamin Van Durme, Leah Findlater, and Jordan Boyd-Graber. Interactive refinement of cross-lingual word embeddings. In *Proceedings of Empirical Methods in Natural Language Processing*, 2020.

- Mozhi Zhang, Keyulu Xu, Ken-ichi Kawarabayashi, Stefanie Jegelka, and Jordan Boyd-Graber. Are girls neko or shōjo? cross-lingual alignment of non-isomorphic embeddings with iterative normalization. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3180–3189, 2019b.
- Rolf W Banz. The relationship between return and market value of common stocks. *Journal of financial economics*, 9(1):3–18, 1981.
- Eugene F Fama and Kenneth R French. Common risk factors in the returns on stocks and bonds. *Journal of financial economics*, 33(1):3–56, 1993.
- Stephen A Ross. The arbitrage theory of capital asset pricing. *Journal of Economic Theory*, 13(3):341–360, 1976.
- Shai Ben-David, John Blitzer, Koby Crammer, Alex Kulesza, Fernando Pereira, and Jennifer Wortman Vaughan. A theory of learning from different domains. *Machine learning*, 79(1-2):151–175, 2010.
- John Blitzer, Koby Crammer, Alex Kulesza, Fernando Pereira, and Jennifer Wortman. Learning bounds for domain adaptation. In *Advances in neural information processing systems*, pages 129–136, 2008.
- Yishay Mansour, Mehryar Mohri, and Afshin Rostamizadeh. Domain adaptation: Learning bounds and algorithms. In *Conference on Learning Theory*, 2009.
- Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. Domain-adversarial training of neural networks. *The Journal of Machine Learning Research*, 17(1):2096–2030, 2016.
- Han Zhao, Shanghang Zhang, Guanhong Wu, José MF Moura, Joao P Costeira, and Geoffrey J Gordon. Adversarial multiple source domain adaptation. In *Advances in neural information processing systems*, pages 8559–8570, 2018.
- Han Zhao, Remi Tachet Des Combes, Kun Zhang, and Geoffrey Gordon. On learning invariant representations for domain adaptation. In *International Conference on Machine Learning*, pages 7523–7532, 2019.
- Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9729–9738, 2020.
- Dan Hendrycks, Xiaoyuan Liu, Eric Wallace, Adam Dziedziec, Rishabh Krishnan, and Dawn Song. Pretrained transformers improve out-of-distribution robustness. In *Association for Computational Linguistics*, 2020.
- Kaiyang Zhou, Yongxin Yang, Yu Qiao, and Tao Xiang. Domain generalization with mixstyle. In *International Conference on Learning Representations*, 2021.

- Kartik Ahuja, Jun Wang, Amit Dhurandhar, Karthikeyan Shanmugam, and Kush R. Varshney. Empirical or invariant risk minimization? a sample complexity perspective. In *International Conference on Learning Representations*, 2021.
- Elan Rosenfeld, Pradeep Kumar Ravikumar, and Andrej Risteski. The risks of invariant risk minimization. In *International Conference on Learning Representations*, 2021.
- Joel Goh and Melvyn Sim. Distributionally robust optimization and its tractable approximations. *Operations research*, 58:902–917, 2010.
- Shiori Sagawa, Pang Wei Koh, Tatsunori B. Hashimoto, and Percy Liang. Distributionally robust neural networks. In *International Conference on Learning Representations*, 2020.
- Aman Sinha, Hongseok Namkoong, and John Duchi. Certifying some distributional robustness with principled adversarial training. In *International Conference on Learning Representations*, 2018.
- Matthew Staib and Stefanie Jegelka. Distributionally robust optimization and generalization in kernel methods. In *Advances in Neural Information Processing Systems*, pages 9134–9144, 2019.
- Borja Ibarz, Vitaly Kurin, George Papamakarios, Kyriacos Nikiforou, Mehdi Bennani, Róbert Csordás, Andrew Joseph Dudzik, Matko Bošnjak, Alex Vitvitskyi, Yulia Rubanova, et al. A generalist neural algorithmic learner. In *Learning on Graphs Conference*, pages 2–1. PMLR, 2022.
- Gyoung S Na and Chanyoung Park. Nonlinearity encoding for extrapolation of neural networks. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 1284–1294, 2022.
- Xiner Li, Shurui Gui, Youzhi Luo, and Shuiwang Ji. Graph structure and feature extrapolation for out-of-distribution generalization. *arXiv preprint arXiv:2306.08076*, 2023.
- Jihoon Ko, Taehyung Kwon, Kijung Shin, and Juho Lee. Learning to pool in graph neural networks for extrapolation. *arXiv preprint arXiv:2106.06210*, 2021.
- Rion Snow, Brendan O’connor, Dan Jurafsky, and Andrew Y Ng. Cheap and fast—but is it good? evaluating non-expert annotations for natural language tasks. In *Proceedings of Empirical Methods in Natural Language Processing*, 2008.
- Peter Welinder, Steve Branson, Pietro Perona, and Serge J Belongie. The multidimensional wisdom of crowds. In *Advances in neural information processing systems*, pages 2424–2432, 2010.
- Yan Yan, Rómer Rosales, Glenn Fung, Ramanathan Subramanian, and Jennifer Dy. Learning from multiple annotators with varying expertise. *Machine learning*, 95(3):291–327, 2014.
- Xiyu Yu, Tongliang Liu, Mingming Gong, and Dacheng Tao. Learning with biased complementary labels. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 68–83, 2018.

- Avrim Blum, Adam Kalai, and Hal Wasserman. Noise-tolerant learning, the parity problem, and the statistical query model. *Journal of the ACM (JACM)*, 50(4):506–519, 2003.
- Lu Jiang, Di Huang, Mason Liu, and Weilong Yang. Beyond synthetic noise: Deep learning on controlled noisy labels. In *ICML*, 2020a. URL <https://arxiv.org/abs/1911.09781>.
- Wei Liu, Yu-Gang Jiang, Jiebo Luo, and Shih-Fu Chang. Noise resistant graph ranking for improved web image search. In *CVPR 2011*, pages 849–856. IEEE, 2011.
- Nagarajan Natarajan, Inderjit S Dhillon, Pradeep K Ravikumar, and Ambuj Tewari. Learning with noisy labels. In *Advances in neural information processing systems*, pages 1196–1204, 2013.
- Tongliang Liu and Dacheng Tao. Classification with noisy labels by importance reweighting. *IEEE Transactions on pattern analysis and machine intelligence*, 38(3):447–461, 2015.
- Yu Yao, Tongliang Liu, Bo Han, Mingming Gong, Jiankang Deng, Gang Niu, and Masashi Sugiyama. Dual t: Reducing estimation error for transition matrix in label-noise learning. *arXiv preprint arXiv:2006.07805*, 2020.
- Aritra Ghosh, Himanshu Kumar, and PS Sastry. Robust loss functions under label noise for deep neural networks. *arXiv preprint arXiv:1712.09482*, 2017.
- Yueming Lyu and Ivor W Tsang. Curriculum loss: Robust learning and generalization against label corruption. *arXiv preprint arXiv:1905.10045*, 2019.
- Yisen Wang, Xingjun Ma, Zaiyi Chen, Yuan Luo, Jinfeng Yi, and James Bailey. Symmetric cross entropy for robust learning with noisy labels. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 322–330, 2019.
- Zhilu Zhang and Mert Sabuncu. Generalized cross entropy loss for training deep neural networks with noisy labels. In *Advances in neural information processing systems*, pages 8778–8788, 2018.
- Eric Arazo, Diego Ortego, Paul Albert, Noel E O’Connor, and Kevin McGuinness. Unsupervised label noise modeling and loss correction. *arXiv preprint arXiv:1904.11238*, 2019.
- Haw-Shiuan Chang, Erik Learned-Miller, and Andrew McCallum. Active bias: Training more accurate neural networks by emphasizing high variance samples. In *Advances in Neural Information Processing Systems*, pages 1002–1012, 2017b.
- Bo Han, Gang Niu, Xingrui Yu, Quanming Yao, Miao Xu, Ivor W Tsang, and Masashi Sugiyama. Sigua: Forgetting may make learning with noisy labels more robust. In *Proceedings of the 37-th International conference on machine learning (ICML-20)*, 2020.
- Dan Hendrycks, Mantas Mazeika, Duncan Wilson, and Kevin Gimpel. Using trusted data to train deep networks on labels corrupted by severe noise. In *Advances in neural information processing systems*, pages 10456–10465, 2018.

- Xingjun Ma, Yisen Wang, Michael E Houle, Shuo Zhou, Sarah M Erfani, Shu-Tao Xia, Sudanthi Wijewickrema, and James Bailey. Dimensionality-driven learning with noisy labels. *arXiv preprint arXiv:1806.02612*, 2018.
- Giorgio Patrini, Alessandro Rozza, Aditya Krishna Menon, Richard Nock, and Lizhen Qu. Making deep neural networks robust to label noise: A loss correction approach. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1944–1952, 2017.
- Scott Reed, Honglak Lee, Dragomir Anguelov, Christian Szegedy, Dumitru Erhan, and Andrew Rabinovich. Training deep neural networks on noisy labels with bootstrapping. *arXiv preprint arXiv:1412.6596*, 2014.
- Hwanjun Song, Minseok Kim, and Jae-Gil Lee. Selfie: Refurbishing unclean samples for robust deep learning. In *International Conference on Machine Learning*, pages 5907–5915, 2019.
- Ruxin Wang, Tongliang Liu, and Dacheng Tao. Multiclass learning with partially corrupted labels. *IEEE transactions on neural networks and learning systems*, 29(6):2568–2580, 2017.
- Pengfei Chen, Benben Liao, Guangyong Chen, and Shengyu Zhang. Understanding and utilizing deep neural networks trained with noisy labels. *arXiv preprint arXiv:1905.05040*, 2019.
- Bo Han, Quanming Yao, Xingrui Yu, Gang Niu, Miao Xu, Weihua Hu, Ivor Tsang, and Masashi Sugiyama. Co-teaching: Robust training of deep neural networks with extremely noisy labels. In *Advances in neural information processing systems*, pages 8527–8537, 2018a.
- Lu Jiang, Zhengyuan Zhou, Thomas Leung, Li-Jia Li, and Li Fei-Fei. Mentornet: Learning data-driven curriculum for very deep neural networks on corrupted labels. In *International Conference on Machine Learning*, pages 2304–2313, 2018.
- Eran Malach and Shai Shalev-Shwartz. Decoupling” when to update” from” how to update”. In *Advances in Neural Information Processing Systems*, pages 960–970, 2017.
- Duc Tam Nguyen, Chaithanya Kumar Mummadi, Thi Phuong Nhung Ngo, Thi Hoai Phuong Nguyen, Laura Beggel, and Thomas Brox. Self: Learning to filter noisy labels with self-ensembling. *arXiv preprint arXiv:1910.01842*, 2019.
- Yanyao Shen and Sujay Sanghavi. Learning with bad training data via iterative trimmed loss minimization. In *International Conference on Machine Learning*, pages 5739–5748. PMLR, 2019.
- Yisen Wang, Weiyang Liu, Xingjun Ma, James Bailey, Hongyuan Zha, Le Song, and Shu-Tao Xia. Iterative learning with open-set noisy labels. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8688–8696, 2018.
- Xingrui Yu, Bo Han, Jiangchao Yao, Gang Niu, Ivor W Tsang, and Masashi Sugiyama. How does disagreement help generalization against label corruption? *arXiv preprint arXiv:1901.04215*, 2019.

- Alan Joseph Bekker and Jacob Goldberger. Training deep neural-networks based on unreliable labels. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2682–2686. IEEE, 2016.
- Xinlei Chen and Abhinav Gupta. Webly supervised learning of convolutional networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1431–1439, 2015.
- J. Goldberger and E. Ben-Reuven. Training deep neural-networks using a noise adaptation layer. In *ICLR*, 2017.
- Bo Han, Jiangchao Yao, Gang Niu, Mingyuan Zhou, Ivor Tsang, Ya Zhang, and Masashi Sugiyama. Masking: A new perspective of noisy supervision. *Advances in Neural Information Processing Systems*, 31:5836–5846, 2018b.
- Ishan Jindal, Matthew Nokleby, and Xuewen Chen. Learning deep networks from noisy labels with dropout regularization. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pages 967–972. IEEE, 2016.
- Jingling Li, Yanchao Sun, Jiahao Su, Taiji Suzuki, and Furong Huang. Understanding generalization in deep learning via tensor methods. *arXiv preprint arXiv:2001.05070*, 2020a.
- Sainbayar Sukhbaatar, Joan Bruna, Manohar Paluri, Lubomir Bourdev, and Rob Fergus. Training convolutional networks with noisy labels. *arXiv preprint arXiv:1406.2080*, 2014.
- 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, 2018.
- Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*, 2015.
- Dan Hendrycks, Kimin Lee, and Mantas Mazeika. Using pre-training can improve model robustness and uncertainty. *arXiv preprint arXiv:1901.09960*, 2019.
- Simon Jenni and Paolo Favaro. Deep bilevel learning. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 618–633, 2018.
- Gabriel Pereyra, George Tucker, Jan Chorowski, Łukasz Kaiser, and Geoffrey Hinton. Regularizing neural networks by penalizing confident output distributions. *arXiv preprint arXiv:1701.06548*, 2017.
- Ryutaro Tanno, Ardavan Saeedi, Swami Sankaranarayanan, Daniel C Alexander, and Nathan Silberman. Learning from noisy labels by regularized estimation of annotator confusion. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 11244–11253, 2019.
- Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017b.

- Luís PF Garcia, André CPLF de Carvalho, and Ana C Lorena. Noise detection in the meta-learning level. *Neurocomputing*, 176:14–25, 2016.
- Yunzhu Li, Jiajun Wu, Russ Tedrake, Joshua B. Tenenbaum, and Antonio Torralba. Learning particle dynamics for manipulating rigid bodies, deformable objects, and fluids. In *International Conference on Learning Representations*, 2019.
- Yifan Ding, Liqiang Wang, Deliang Fan, and Boqing Gong. A semi-supervised two-stage approach to learning from noisy labels. In *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1215–1224. IEEE, 2018.
- Junnan Li, Richard Socher, and Steven CH Hoi. Dividemix: Learning with noisy labels as semi-supervised learning. *arXiv preprint arXiv:2002.07394*, 2020b.
- Sheng Liu, Jonathan Niles-Weed, Narges Razavian, and Carlos Fernandez-Granda. Early-learning regularization prevents memorization of noisy labels. *arXiv preprint arXiv:2007.00151*, 2020a.
- Yan Yan, Zhongwen Xu, Ivor W Tsang, Guodong Long, and Yi Yang. Robust semi-supervised learning through label aggregation. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.
- Devansh Arpit, Stanisław Jastrzebski, Nicolas Ballas, David Krueger, Emmanuel Bengio, Maxinder S Kanwal, Tegan Maharaj, Asja Fischer, Aaron Courville, Yoshua Bengio, et al. A closer look at memorization in deep networks. *arXiv preprint arXiv:1706.05394*, 2017.
- Hartmut Maennel, Ibrahim Alabdulmohsin, Ilya Tolstikhin, Robert JN Baldock, Olivier Bousquet, Sylvain Gelly, and Daniel Keysers. What do neural networks learn when trained with random labels? *arXiv preprint arXiv:2006.10455*, 2020.
- Guillaume Alain and Yoshua Bengio. Understanding intermediate layers using linear classifier probes. *arXiv preprint arXiv:1610.01644*, 2016.
- 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*, pages 2691–2699, 2015.
- Vitaly Feldman and Chiyuan Zhang. What neural networks memorize and why: Discovering the long tail via influence estimation. *arXiv preprint arXiv:2008.03703*, 2020.
- Kimin Lee, Sukmin Yun, Kibok Lee, Honglak Lee, Bo Li, and Jinwoo Shin. Robust inference via generative classifiers for handling noisy labels. In *International Conference on Machine Learning*, pages 3763–3772. PMLR, 2019.
- Dara Bahri, Heinrich Jiang, and Maya Gupta. Deep k-nn for noisy labels. In *International Conference on Machine Learning*, pages 540–550. PMLR, 2020.
- Pengxiang Wu, Songzhu Zheng, Mayank Goswami, Dimitris Metaxas, and Chao Chen. A topological filter for learning with label noise. *arXiv preprint arXiv:2012.04835*, 2020.

- Katherine L Hermann and Andrew K Lampinen. What shapes feature representations? exploring datasets, architectures, and training. *arXiv preprint arXiv:2006.12433*, 2020.
- Harshay Shah, Kaustav Tamuly, Aditi Raghunathan, Prateek Jain, and Praneeth Netrapalli. The pitfalls of simplicity bias in neural networks. *arXiv preprint arXiv:2006.07710*, 2020.
- Amartya Sanyal, Puneet K Dokania, Varun Kanade, and Philip HS Torr. How benign is benign overfitting? *arXiv preprint arXiv:2007.04028*, 2020.
- Katherine L Hermann, Ting Chen, and Simon Kornblith. The origins and prevalence of texture bias in convolutional neural networks. *arXiv preprint arXiv:1911.09071*, 2019.
- Grégoire Montavon, Wojciech Samek, and Klaus-Robert Müller. Methods for interpreting and understanding deep neural networks. *Digital Signal Processing*, 73:1–15, 2018.
- Wei Hu, Zhiyuan Li, and Dingli Yu. Simple and effective regularization methods for training on noisily labeled data with generalization guarantee. In *International Conference on Learning Representations*, 2020b. URL <https://openreview.net/forum?id=Hke3gyHYwH>.
- Hao Cheng, Zhaowei Zhu, Xingyu Li, Yifei Gong, Xing Sun, and Yang Liu. Learning with instance-dependent label noise: A sample sieve approach. *arXiv preprint arXiv:2010.02347*, 2020.
- Xinshao Wang, Yang Hua, Elyor Kodirov, and Neil M Robertson. Prosellfc: Progressive self label correction for target revising in label noise. *arXiv preprint arXiv:2005.03788*, 2020.
- Ryoma Sato, Makoto Yamada, and Hisashi Kashima. Approximation ratios of graph neural networks for combinatorial problems. In *Advances in Neural Information Processing Systems*, pages 4081–4090, 2019.
- Denny Wu and Ji Xu. On the optimal weighted ℓ_2 regularization in overparameterized linear regression. *arXiv preprint arXiv:2006.05800*, 2020.
- A. Krizhevsky. Learning multiple layers of features from tiny images. 2009.
- Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European Conference on Computer Vision*, pages 630–645, 2016.
- Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.
- Saeed Reza Kheradpisheh, Masoud Ghodrati, Mohammad Ganjtabesh, and Timothée Masquelier. Deep networks can resemble human feed-forward vision in invariant object recognition. *Scientific reports*, 6(1):1–24, 2016.
- Qing Miao, Xiaohu Wu, Chao Xu, Wangmeng Zuo, and Zhaopeng Meng. On better detecting and leveraging noisy samples for learning with severe label noise. *Pattern Recognition*, 136: 109210, 2023.

- Long Chen, Fei Wang, Ruijing Yang, Fei Xie, Wenjing Wang, Cai Xu, Wei Zhao, and Ziyu Guan. Representation learning from noisy user-tagged data for sentiment classification. *International Journal of Machine Learning and Cybernetics*, 13(12):3727–3742, 2022.
- Mahsa Forouzesh, Hanie Sedghi, and Patrick Thiran. Leveraging unlabeled data to track memorization. *arXiv preprint arXiv:2212.04461*, 2022.
- Yingbin Bai, Erkun Yang, Bo Han, Yanhua Yang, Jiatong Li, Yinian Mao, Gang Niu, and Tongliang Liu. Understanding and improving early stopping for learning with noisy labels. *Advances in Neural Information Processing Systems*, 34:24392–24403, 2021.
- Zhaowei Zhu, Zihao Dong, and Yang Liu. Detecting corrupted labels without training a model to predict. In *International conference on machine learning*, pages 27412–27427. PMLR, 2022.
- Ori Hadary, Luke Marshall, Ishai Menache, Abhisek Pan, Esaias E Greeff, David Dion, Star Dorminey, Shailesh Joshi, Yang Chen, Mark Russinovich, and Thomas Moscibroda. Pro-tean: VM allocation service at scale. In *OSDI*, 2020.
- Warren Powell. *Reinforcement Learning and Stochastic Optimization: A unified framework for sequential decisions*. John Wiley & Sons, 2022.
- Hongzi Mao, Shaileshh Bojja Venkatakrishnan, Malte Schwarzkopf, and Mohammad Alizadeh. Variance reduction for reinforcement learning in input-driven environments. In *ICLR*, 2019b.
- Thodoris Lykouris and Sergei Vassilvitskii. Competitive caching with machine learned advice. *Journal of the ACM (JACM)*, 68(4):1–25, 2021.
- Ravi Kumar, Manish Purohit, and Zoya Svitkina. Improving online algorithms via ml predictions. In *NeurIPS*, 2018.
- Sreenivas Gollapudi and Debmalya Panigrahi. Online algorithms for rent-or-buy with expert advice. In *ICML*, 2019.
- Adam N Elmachtoub and Paul Grigas. Smart “predict, then optimize”. *Management Science*, 68(1), 2022.
- Dong Chen, Kaian Chen, Zhaojian Li, Tianshu Chu, Rui Yao, Feng Qiu, and Kaixiang Lin. PowerNet: Multi-agent deep reinforcement learning for scalable powergrid control. *IEEE Transactions on Power Systems*, 37(2), 2021.
- Joyce Fang, Martin Ellis, Bin Li, Siyao Liu, Yasaman Hosseinkashi, Michael Revow, Albert Sadovnikov, Ziyuan Liu, Peng Cheng, Sachin Ashok, David Zhao, Ross Cutler, Yan Lu, and Johannes Gehrke. Reinforcement learning for bandwidth estimation and congestion control in real-time communications. *arXiv preprint arXiv:1912.02222*, 2019.
- Hongzi Mao, Mohammad Alizadeh, Ishai Menache, and Srikanth Kandula. Resource management with deep reinforcement learning. In *HotNets*, 2016.
- Dhruv Madeka, Kari Torkkola, Carson Eisenach, Dean Foster, and Anna Luo. Deep inventory management. *arXiv preprint arXiv:2210.03137*, 2022.

- Dylan J Foster, Sham M Kakade, Jian Qian, and Alexander Rakhlin. The statistical complexity of interactive decision making. *arXiv preprint arXiv:2112.13487*, 2021.
- Thomas Mesnard, Theophane Weber, Fabio Viola, Shantanu Thakoor, Alaa Saade, Anna Harutyunyan, Will Dabney, Thomas S Stepleton, Nicolas Heess, Arthur Guez, Éric Moulines, Marcus Hutter, Lars Buesing, and Rémi Munos. Counterfactual credit assignment in model-free reinforcement learning. In *ICML*, 2021.
- Edwin KP Chong, Robert L Givan, and Hyeon Soo Chang. A framework for simulation-based network control via hindsight optimization. In *CDC*, volume 2, 2000.
- Parikshit Gopalan, Adam Klivans, and Raghu Meka. Polynomial-time approximation schemes for knapsack and related counting problems using branching programs. *arXiv preprint arXiv:1008.3187*, 2010.
- Michele Conforti, Gérard Cornuéjols, and Giacomo Zambelli. *Integer programming*, volume 271. Springer, 2014.
- Paul Dutting, Michal Feldman, Thomas Kesselheim, and Brendan Lucier. Prophet inequalities made easy: Stochastic optimization by pricing nonstochastic inputs. *SIAM Journal on Computing*, 49(3), 2020.
- Alberto Vera and Siddhartha Banerjee. The bayesian prophet: A low-regret framework for online decision making. *Management Science*, 67(3), 2021.
- Finnian Lattimore, Tor Lattimore, and Mark D Reid. Causal bandits: learning good interventions via causal inference. In *NeurIPS*, 2016.
- Yangyi Lu, Amirhossein Meisami, and Ambuj Tewari. Efficient reinforcement learning with prior causal knowledge. In *Conference on Causal Learning and Reasoning*, 2022b.
- Junzhe Zhang, Daniel Kumor, and Elias Bareinboim. Causal imitation learning with unobserved confounders. In *NeurIPS*, 2020a.
- Thomas Dietterich, George Trimonias, and Zhitang Chen. Discovering and Removing Exogenous State Variables and Rewards for Reinforcement Learning. In *ICML*, 2018.
- Yonathan Efroni, Dylan J Foster, Dipendra Misra, Akshay Krishnamurthy, and John Langford. Sample-efficient reinforcement learning in the presence of exogenous information. In *Conference on Learning Theory*, pages 5062–5127. PMLR, 2022.
- Jiekun Feng, Mark Gluzman, and Jim G Dai. Scalable deep reinforcement learning for ride-hailing. In *American Control Conference*, 2021.
- Luc Mercier and Pascal Van Hentenryck. Performance analysis of online anticipatory algorithms for large multistage stochastic integer programs. In *IJCAI*, pages 1979–1984, 2007.
- R Tyrrell Rockafellar and Roger J-B Wets. Scenarios and policy aggregation in optimization under uncertainty. *Mathematics of operations research*, 16(1):119–147, 1991.

- David B Brown and James E Smith. Information relaxations and duality in stochastic dynamic programs:: A review and tutorial. *Foundations and Trends in Optimization*, 5(3), 2022.
- Martin L Puterman. *Markov decision processes: Discrete stochastic dynamic programming*. John Wiley & Sons, 2014.
- Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. In *International conference on machine learning*, pages 2052–2062. PMLR, 2019.
- Yao Liu, Adith Swaminathan, Alekh Agarwal, and Emma Brunskill. Provably good batch off-policy reinforcement learning without great exploration. *Advances in neural information processing systems*, 33:1264–1274, 2020b.
- Paria Rashidinejad, Banghua Zhu, Cong Ma, Jiantao Jiao, and Stuart Russell. Bridging offline reinforcement learning and imitation learning: A tale of pessimism. *Advances in Neural Information Processing Systems*, 34:11702–11716, 2021.
- Ching-An Cheng, Tengyang Xie, Nan Jiang, and Alekh Agarwal. Adversarially trained actor critic for offline reinforcement learning. In *International Conference on Machine Learning*, pages 3852–3878. PMLR, 2022.
- Thomas Anthony, Zheng Tian, and David Barber. Thinking fast and slow with deep learning and tree search. *Advances in neural information processing systems*, 30, 2017.
- Yuchen Fang, Kan Ren, Weiqing Liu, Dong Zhou, Weinan Zhang, Jiang Bian, Yong Yu, and Tie-Yan Liu. Universal trading for order execution with oracle policy distillation. In *AAAI*, 2021.
- Kirk Bansak and Elisabeth Paulson. Outcome-driven dynamic refugee assignment with allocation balancing. In *EC*, 2022.
- Narges Ahani, Paul Gözl, Ariel D Procaccia, Alexander Teytelboym, and Andrew C Trapp. Dynamic placement in refugee resettlement. In *Proceedings of the 22nd ACM Conference on Economics and Computation*, pages 5–5, 2021.
- Siddhartha Banerjee and Daniel Freund. Uniform loss algorithms for online stochastic decision-making with applications to bin packing. In *SIGMETRICS*, 2020.
- Stéphane Ross and J Andrew Bagnell. Reinforcement and imitation learning via interactive no-regret learning. *arXiv preprint arXiv:1406.5979*, 2014.
- Vijay R Konda and John N Tsitsiklis. Actor-critic algorithms. In *NeurIPS*, 2000.
- Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *AAAI*, 2016.
- John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

- Andrew Y Ng and Michael Jordan. Pegasus: a policy search method for large mdps and pomdps. In *UAI*, 2000.
- Cameron Allen, Kavosh Asadi, Melrose Roderick, Abdel-rahman Mohamed, George Konidaris, and Michael Littman. Mean actor critic. *arXiv preprint arXiv:1709.00503*, 2017.
- Wen Sun, Arun Venkatraman, Geoffrey J Gordon, Byron Boots, and J Andrew Bagnell. Deeply aggravated: Differentiable imitation learning for sequential prediction. In *ICML*, 2017.
- Alberto Vera, Siddhartha Banerjee, and Itai Gurvich. Online allocation and pricing: Constant regret via bellman inequalities. *Operations Research*, 69(3), 2021.
- Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *AISTATS*, 2011.
- Xinyan Yan, Byron Boots, and Ching-An Cheng. Explaining fast improvement in online imitation learning. In *UAI*, 2021.
- David B Brown and James E Smith. Information relaxations, duality, and convex stochastic dynamic programs. *Operations Research*, 62(6), 2014.
- Ibrahim El Shar and Daniel Jiang. Lookahead-bounded q-learning. In *ICML*, 2020.
- Siddhartha Banerjee, Itai Gurvich, and Alberto Vera. Constant regret in online allocation: On the sufficiency of a single historical trace, 2020.
- Niv Buchbinder, Kamal Jain, and Mohit Singh. Secretary problems and incentives via linear programming. *ACM SIGecom Exchanges*, 8(2), 2009.
- Kenneth Littlewood. Forecasting and control of passenger bookings. In *Airline Group International Federation of Operational Research Societies*, 1972.
- Christopher Archer, Siddhartha Banerjee, Mayleen Cortez, Carrie Rucker, Sean R Sinclair, Max Solberg, Qiaomin Xie, and Christina Lee Yu. Orsuite: Benchmarking suite for sequential operations models. *ACM SIGMETRICS Performance Evaluation Review*, 49(2), 2022.
- Eli Cortez, Anand Bonde, Alexandre Muzio, Mark Russinovich, Marcus Fontoura, and Ricardo Bianchini. Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms. In *SOSP*, 2017.
- Niv Buchbinder, Yaron Fairstein, Konstantina Mellou, Ishai Menache, and Joseph Naor. Online virtual machine allocation with lifetime and load predictions. *ACM SIGMETRICS Performance Evaluation Review*, 49(1), 2021.
- Arthur Jiang, Jia Zhang, Pingchao Yu, Lyuchun Huang, Yang Qiu, Jinyu Wang, Wenlei Shi, Kaiqi Li, Zhanyu Wang, Chengruidong Zhang, Tianyi Sun, Miaoran Chen, Kuanwei Yu, Xinran Wei, Michael Li, Ning Shang, Qiwei Meng, Shan Li, Jiang Bian, Biao Cheng, and Tie-Yan Liu. Maro: A multi-agent resource optimization platform, 2020b. URL <https://github.com/microsoft/maro>.

- Rina Panigrahy, Kunal Talwar, Lincoln Uyeda, and Udi Wieder. Heuristics for vector bin packing. Technical report, Microsoft Research, 2011. Available from <https://www.microsoft.com/en-us/research/wp-content/uploads/2011/01/VBPackingESA11.pdf>.
- Ching-An Cheng, Andrey Kolobov, and Adith Swaminathan. Heuristic-guided reinforcement learning. In *NeurIPS*, 2021.
- Vincent Liu, James Wright, and Martha White. Exploiting action impact regularity and partially known models for offline reinforcement learning. *arXiv preprint arXiv:2111.08066*, 2021.
- Jonathan Lee, Alekh Agarwal, Christoph Dann, and Tong Zhang. Learning in pomdps is sample-efficient with hindsight observability. In *International Conference on Machine Learning*, pages 18733–18773. PMLR, 2023.
- Matias Alvo, Daniel Russo, and Yash Kanoria. Neural inventory control in networks via hindsight differentiable policy optimization. *arXiv preprint arXiv:2306.11246*, 2023.
- 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.
- Lenaic Chizat, Edouard Oyallon, and Francis Bach. On lazy training in differentiable programming. In *Advances in Neural Information Processing Systems*, pages 2933–2943, 2019.
- Zizhao Zhang, Han Zhang, Serkan O Arik, Honglak Lee, and Tomas Pfister. Distilling effective supervision from severe label noise. In *CVPR 2020*, pages 9294–9303. IEEE, 2020b.
- Mengye Ren, Wenyuan Zeng, Bin Yang, and Raquel Urtasun. Learning to reweight examples for robust deep learning. *arXiv preprint arXiv:1803.09050*, 2018.
- Jun Shu, Qi Xie, Lixuan Yi, Qian Zhao, Sanping Zhou, Zongben Xu, and Deyu Meng. Meta-weight-net: Learning an explicit mapping for sample weighting. In *Advances in Neural Information Processing Systems*, pages 1919–1930, 2019.
- Kuang-Huei Lee, Xiaodong He, Lei Zhang, and Linjun Yang. Cleannet: Transfer learning for scalable image classifier training with label noise. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5447–5456, 2018.
- Jiangfan Han, Ping Luo, and Xiaogang Wang. Deep self-learning from noisy labels. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 5138–5147, 2019.
- Wen Li, Limin Wang, Wei Li, Eirikur Agustsson, and Luc Van Gool. Webvision database: Visual learning and understanding from web data. *arXiv preprint arXiv:1708.02862*, 2017.
- 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.

- Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alex Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. *arXiv preprint arXiv:1602.07261*, 2016.
- Keyulu Xu, Jingling Li, Mozhi Zhang, Simon S. Du, Ken ichi Kawarabayashi, and Stefanie Jegelka. What can neural networks reason about? In *International Conference on Learning Representations*, 2020.
- Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.
- Alekh Agarwal, Nan Jiang, Sham M Kakade, and Wen Sun. Reinforcement Learning: Theory and Algorithms. Technical report, University of Washington, 2019. Available from <https://rltheorybook.github.io>.
- Sébastien Bubeck and Nicolo Cesa-Bianchi. Regret analysis of stochastic and nonstochastic multi-armed bandit problems. *Foundations and Trends in Machine Learning*, 5(1), 2012.
- Aleksandrs Slivkins. Introduction to multi-armed bandits. *Foundations and Trends in Machine Learning*, 12(1-2), 2019.
- Santiago R Balseiro and David B Brown. Approximations to stochastic dynamic programs via information relaxation duality. *Operations Research*, 67(2), 2019.
- David B Brown and Martin B Haugh. Information relaxation bounds for infinite horizon markov decision processes. *Operations Research*, 65(5), 2017.
- Nathan Kallus and Angela Zhou. Stateful offline contextual policy evaluation and learning. In *AISTATS*, 2022.
- Daniel J Russo, Benjamin Van Roy, Abbas Kazerouni, Ian Osband, and Zheng Wen. A tutorial on thompson sampling. *Foundations and Trends in Machine Learning*, 11(1), 2018.
- Peter E Hart, David G Stork, and Richard O Duda. *Pattern classification*. John Wiley & Sons, 2000.
- David Venuto, Elaine Lau, Doina Precup, and Ofir Nachum. Policy gradients incorporating the future. In *ICLR*, 2022.
- Jialin Song, Ravi Lanka, Albert Zhao, Aadyot Bhatnagar, Yisong Yue, and Masahiro Ono. Learning to search via retrospective imitation. *arXiv preprint arXiv:1804.00846*, 2018b.
- Andrew Warrington, Jonathan W Lavington, Adam Scibior, Mark Schmidt, and Frank Wood. Robust asymmetric learning in POMDPs. In *ICML*, 2021.
- Alexander L Stolyar. An infinite server system with general packing constraints. *Operations Research*, 61(5), 2013.
- Yusen Li, Xueyan Tang, and Wentong Cai. Dynamic bin packing for on-demand cloud resource allocation. *IEEE Transactions on Parallel and Distributed Systems*, 27(1), 2015.

- Hongzi Mao, Malte Schwarzkopf, Shaileshh Bojja Venkatakrishnan, Zili Meng, and Mohammad Alizadeh. Learning scheduling algorithms for data processing clusters. In *ACM Special Interest Group on Data Communication*, 2019c.
- Haitao Zhang, Xin Geng, and Huadong Ma. Learning-driven interference-aware workload parallelization for streaming applications in heterogeneous cluster. *IEEE Transactions on Parallel and Distributed Systems*, 32(1), 2020c.
- Junjie Sheng, Yiqiu Hu, Wenli Zhou, Lei Zhu, Bo Jin, Jun Wang, and Xiangfeng Wang. Learning to schedule multi-NUMA virtual machines via reinforcement learning. *Pattern Recognition*, 121, 2022.
- Jim G Dai and Mark Gluzman. Queueing network controls via deep reinforcement learning. *Stochastic Systems*, 2021.
- Pavithra Harsha, Ashish Jagmohan, Jayant Kalagnanam, Brian Quanz, and Divya Singhvi. Math programming based reinforcement learning for multi-echelon inventory management. In *NeurIPS Deep RL Workshop*, 2021.
- Dimitris Bertsimas and John N Tsitsiklis. *Introduction to linear optimization*, volume 6. Athena, 1997.
- Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *NeurIPS*, 2015.
- Irwan Bello, Hieu Pham, Quoc V Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. In *ICLR*, 2017.
- Rohan Chitnis and Tomás Lozano-Pérez. Learning compact models for planning with exogenous processes. In *Conference on Robot Learning*, pages 813–822. PMLR, 2020.
- Haoyuan Hu, Xiaodong Zhang, Xiaowei Yan, Longfei Wang, and Yinghui Xu. Solving a new 3d bin packing problem with deep reinforcement learning method. *arXiv preprint arXiv:1708.05930*, 2017b.
- Yunhao Tang, Shipra Agrawal, and Yuri Faenza. Reinforcement learning for integer programming: Learning to cut. In *ICML*, 2020.
- John Langford and Tong Zhang. The epoch-greedy algorithm for contextual multi-armed bandits. In *NeurIPS*, 2007.
- Christian Borgs, Jennifer T Chayes, László Lovász, Vera T Sós, and Katalin Vesztegombi. Convergent sequences of dense graphs I: Subgraph frequencies, metric properties and testing. *Advances in Mathematics*, 219(6), 2008.
- László Lovász and Balázs Szegedy. Limits of dense graph sequences. *Journal of Combinatorial Theory, Series B*, 96(6), 2006.
- Shipra Agrawal and Randy Jia. Learning in structured mdps with convex cost functions: Improved regret bounds for inventory management. *Operations Research*, 70(3):1646–1664, 2022.

- David A. Goldberg, Martin I. Reiman, and Qiong Wang. A Survey of Recent Progress in the Asymptotic Analysis of Inventory Systems. *Production and Operations Management*, 30(6), 2021.
- Linwei Xin and David Alan Goldberg. Distributionally robust inventory control when demand is a martingale. *Mathematics of Operations Research*, 47(3), 2022.
- Eugene A Feinberg. Optimality conditions for inventory control. In *Optimization Challenges in Complex, Networked and Risky Systems*. INFORMS, 2016.
- Varun Gupta and Ana Radovanovic. Online stochastic bin packing. *arXiv preprint arXiv:1211.2687*, 2012.
- Javad Ghaderi, Yuan Zhong, and Rayadurgam Srikant. Asymptotic optimality of bestfit for stochastic bin packing. *ACM SIGMETRICS Performance Evaluation Review*, 42(2), 2014.
- Guido Perboli, Roberto Tadei, and Mauro M Baldi. The stochastic generalized bin packing problem. *Discrete Applied Mathematics*, 160(7-8), 2012.
- Junjie Sheng, Shengliang Cai, Haochuan Cui, Wenhao Li, Yun Hua, Bo Jin, Wenli Zhou, Yiqiu Hu, Lei Zhu, Qian Peng, Hongyuan Zha, and Xiangfeng Wang. VMAgent: Scheduling simulator for reinforcement learning. *arXiv preprint arXiv:2112.04785*, 2021.
- Christian D Hubbs, Hector D Perez, Owais Sarwar, Nikolaos V Sahinidis, Ignacio E Grossmann, and John M Wassick. Or-gym: A reinforcement learning library for operations research problems. *arXiv preprint arXiv:2008.06319*, 2020.
- Omar Darwiche Domingues, Pierre Ménard, Emilie Kaufmann, and Michal Valko. Episodic reinforcement learning in finite mdps: Minimax lower bounds revisited. In *ALT*, 2021.
- Martin L Weitzman. Optimal search for the best alternative. *Econometrica: Journal of the Econometric Society*, 1979.
- Pieter Abbeel and Andrew Y Ng. Exploration and apprenticeship learning in reinforcement learning. In *ICML*, pages 1–8, 2005.
- Daniel Freund and Siddhartha Banerjee. Good prophets know when the end is near. SSRN Scholarly Paper ID 3479189, Social Science Research Network, 2019.
- 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*. 2019.