ABSTRACT

Title of Dissertation: TOWARDS RELIABLE AND EFFICIENT REPRESENTATION LEARNING

Chen Zhu
Doctor of Philosophy, 2022

Dissertation Directed by: Professor Tom Goldstein
Department of Computer Science

Large-scale representation learning has achieved enormous success during the past decade, surpassing human-level accuracy on a range of benchmarks including image recognition and language understanding. The success is supported by advances in both the algorithms and computing capabilities, which enables training large models on enormous amounts of data. While the performance continues to improve on existing benchmarks with larger model and training dataset sizes, the reliability and efficiency of large models are often questioned for deployment in practice. Uncensored datasets can have been poisoned to manipulate model behavior, while practical deployment requires models to be trained or updated quickly on the latest data, and to have low latency for inference.

This dissertation studies how to improve the reliability and efficiency of representation learning. On reliability, we study the threats of data poisoning and evasion attacks and how to defend against these threats. We propose a more vicious targeted clean-label poisoning attack that is highly effective even when the target architecture is unknown. To

defend against such threats, we develop a k-NN based method in the feature space to filter out the poison examples from the training set, which effectively reduces the success rate of poisoning attacks at an insignificant cost of accuracy. For evasion attack, we demonstrate a new threat model against transfer learning, where the attack can be successful without knowledge of the specific classification head. In a broader sense, we also propose methods to enhance the empirical and certified robustness against evasion attacks.

For efficiency, our study focuses on three dimensions: data efficiency, convergence speed and computational complexity. For data efficiency, we propose enhanced adversarial training algorithms as a general data augmentation technique to improve the generalization of models given the same amount of labeled data, where we show its efficacy for Transformer models on a range of language understanding tasks. For convergence speed, we propose an automated initialization scheme to accelerate the convergence of convolutional networks for image recognition and Transformers for machine translation. For computational complexity, to scale Transformers to long sequences, we propose a linear-complexity attention mechanism, which improves the efficiency while preserving the performance of full attention on a range of language and vision tasks.

TOWARDS RELIABLE AND EFFICIENT REPRESENTATION LEARNING

by

Chen Zhu

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
2022

Advisory Committee:
Professor Tom Goldstein, Chair/Advisor
Professor Behtash Babadi, Dean's Representative
Professor David Jacobs
Professor Furong Huang
Professor Rachel Rudinger
Professor John P. Dickerson

Acknowledgments

During my journey towards a Ph.D. degree, I was fortunate to meet various people that made this journey more fulfilling and special. I owe my gratitude to every one of them, and I apologize in advance to anyone I forget to mention in this acknowledgement.

First and foremost, I want to thank my advisor, Tom Goldstein, who provided me everything I could ask for from an advisor. He is an intelligent and open-minded person, who has made everything much easier for me. He has very broad interests and knowledge in many aspects of machine learning, from theory to various applications. He seldom teaches me what to do directly, but I have learned a lot by simply watching what he has done. His pursuit for intuitive and innovative ideas with long-term impacts, and his openness to all the interesting machine learning problems, will continue to guide my future research. I really appreciate him for being supportive for almost all my research ideas, as well as my internships outside the campus. It was a great pleasure to work with such an intelligent and nice person.

I also need to thank all the professors and colleagues I met in the graduate schools at UMD and ShanghaiTech, from whom I learned lots of useful skills. Ronny Huang was the main collaborator of my first paper published at UMD. He sparkled my interest and research ideas in poisoning attacks and defenses. He also gave me lots of support and suggestions for life, and we became good friends since after. I would like to thank all

my wonderful colleagues at UMD, including Ali Shafahi, Kezhi Kong, Micah Goldblum, Jonas Geiping, Amin Ghiasi, Jiuhai Chen, Hamid Kazemi, Phillip Pope, Ahmed Abdelkader, Huimin Zeng, Hengduo Li, and Liam Fowl, who motivated me to work on a diverse set of research topics, and even gave me inspirations for new projects. The classes taught by Furong Huang and Soheil Feizi improved my understandings of machine learning foundations, and provoked my thoughts on adversarial robustness and poisoning attacks. I enjoyed working with Xuchen You, Nirat Saini, Samyadeep Basu, Renkun Ni and Pingyeh Chiang on the interesting course projects for these classes. In addition, Furong's group meeting was a place for me to dive into theories of reinforcement learning, tensor methods and certified robustness. I want to thank Yanchao Sun, Jingling Li, Jiahao Su and Chen Chen for the impressive presentations. I was lucky to become a TA for David Jacobs's deep learning class in my first semester, from which I had my first glimpse of teaching in the US. Thanks to Behtash Babadi, Rachel Rudinger and John P. Dickerson for being my committee members and raised interesting questions for further studying. The experience at ShanghaiTech was unforgettable and taught me to be fearless. I learned 3D vision and compressive sensing from my advisor Yi Ma, as well as the fundamentals of AI and NLP from my co-advisor Kewei Tu, which led to the publication of my very first papers in the research career. A lot of other studying experiences at ShanghaiTech also strengthened my interest in research, including learning computational imaging and computer graphics from Jingyi Yu, writing operating systems in Rust from Hao Chen, and taking all the challenging classes with Shi Jin.

I was fortunate to have the opportunities to collaborate with researchers from the industry through internships. Thanks to Zheng Xu, who helped me a lot at UMD and

introduced me to Federated Learning at Google. I feel fortunate to collaborate with Greg Yang to study the large depth limit of neural networks. I appreciate the support from Ping Wei, Chaowei Xiao, Mohammad Shoeybi, Anima Anandkumar and Bryan Catanzaro to study long-range Transformers at Nvidia. I had a great time studying the memorization of large language models with Ankit Singh Rawat, Manzil Zaheer, Daliang Li, Felix Yu, and Srinadh Bhojanapalli. I had an unforgettable experience interning with Yu Cheng, Zhe Gan, Siqi Sun and Jingjing Liu at Microsoft, which enhanced my interest in NLP and led to impactful publications. I gained lots of insighs about Bayesian deep learning and generative models from David Wipf, Bin Dai and Tingran Wang at MSRA. Special thanks to Jianguo Li, who gave me the first internship opportunity at Intel Labs and became my first exploration in deep learning. Over the years of experiences in the industry, I have also met and learned a lot from Jiacheng Xu, Yandong Li, Yang Zhang, Wenhu Chen, Liqun Chen, Hao Li, Yizhe Zhang, Weizhu Chen, Sheng Zha, Yashar Mehdad, Chengxi Ye, Jingzhao Zhang, Wei Hu, and Dianqi Li.

My life would not have been so easy without the care and support of my family and friends. Thanks to my parents and relatives for supporting me over the years, even when they were in their own difficulties. Thanks to Shuang Ma and our dog Elfie, who made me a happier and more thoughtful person. My life would not have been so fulfilling without them. I am thankful to the care and joy brought by my friends in life: Xiao Su, Nannan Su, Xiangxing Lu, Maryna Holovanova, Kaiyu Yue, Jiaxuan Zong, Shiyi Lan, Xiaozhen Lin, Zeyad Emam, Kaiyan Shi, Andy Chou, Junru Wu, Wei Li, Jia Zheng, Yanbin Dong, Minye Wu, Yu Zhang, and Lai Jiang.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

SGD Stochastic Gradient Descent

Chapter 1: Introduction

1.1 Background

1.1.1 Progress of Large-scale Representation Learning

In the past decade, large-scale representation learning has made significant progresses, surpassing human-level accuracy on a range of benchmarks, including image recognition [16], speech recognition [17], and language understanding [18]. The progress is empowered by the advances in both the algorithms and compute powers, which enable training large neural networks at large scales. For algorithms, innovations in the neural architectures (skip connections [19], normalizations [20, 21] and attentions [22, 23]), together with better initialization and optimization schemes [16, 24, 25], have significantly improved the stability of deeper neural networks. Novel neural architectures also seem to improve the modeling capabilities for certain domains, e.g., Transformers [23] achieve higher parameter efficiency than LSTMs [26] for sequence modeling. VAEs [27], GANs [28] and diffusion models [29] introduce new algorithms and training objectives for neural networks to generate high-fidelity voice and images. New algorithms for unsupervised learning [30–32] have also enabled learning feature representations from large amounts of unlabeled data across multiple modalities [33]. Meanwhile, specialized hard-

wares and techniques for training neural networks [34, 35] have enabled training significantly larger models on larger scales of data. Most recent researches explore wide Transformer-based language models with up to trillions of parameters trained on TBs of text data, where these large language models [11, 36–38] can generate shockingly good responses, demonstrate zero-shot better capabilities on question answering and language understanding than many state-of-the-art supervised baselines, and can even be used to generate code from descriptions. The performance of these language models seems to scale as a power-law of the model size, dataset size, and the amount of compute used for training [39], which encourages more explorations at even larger scales.

With these advances, neural networks have already been deployed in many applications in practice, including autonomous driving, voice assistant and Google search. However, their reliability and efficiency is often questioned. In this dissertation, we summarize our recent works that tackle these challenges.

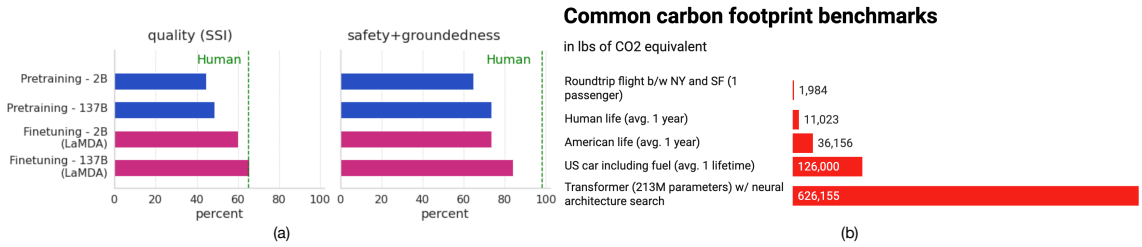

Figure 1.1: (a) The recent LaMDA model [11] can generate high-quality dialogues that matches humans, but still lags behind in terms of safety and groundness metrics. (b) Doing neural architectural search on a medium-sized Transformer produces 5x carbon footprint as a US car's lifetime [12, 13].

1.1.2 Reliability of Representation Learning

In most practical applications, the models are usually only trained to maximize performance metrics on the collected data, without considering the potential failure cases or adversaries. Without censorship, the collected data might have been manipulated or poisoned to cause misbehaviors, e.g., Tay, the Microsoft chat bot, went offline shortly after learning to generate inflammatory and offensive tweets from malicious user responses [40]. Equipped with carefully annotated data and the ability to consult external knowledge sources, the most recent, large LaMDA model [11] matches humans in terms of quality metrics, but still falls behind in the safety and groundedness metrics (Figure 1.1). Moreover, vision models are susceptible to adversarial examples [41], where imperceptible perturbations to clean images can change the prediction of the model completely. Adversarial training improves the robustness against such adversarial examples [42], at the cost of accuracy on clean examples [43].

1.1.3 Efficiency of Learning at Scale

Efficiency is never an outdated dimension for improvements. The data collection process can be costly and time-consuming. For example, natural language understanding can involve hundreds of different languages, hundreds of different tasks and numerous domains. It is impractical to expect enough labeled data is available for every scenario, but large organizations or companies may need to provide service to as many scenarios as possible. Therefore, given the same amount of data, we want to improve the data efficiency and make the metrics as high as possible. Various data augmentations have been

the pillars for high-performance vision models [44], but for language, few augmentation techniques are available that can guarantee consistent improvements on all datasets [45, 46].

Another important factor of efficiency is the training efficiency. Given the same level of computing power, this is mainly affected by the number of training iterations required for convergence and the computational complexity of the model's forward and backward passes. In addition, efforts for hyper-parameter tuning should also be counted into the training cost. Inefficient training procedures can cause significant environmental impacts. [12] showed that the carbon footprint of doing neural architecture search for a 213M-parameter Transformer model is five times as that of a US car's lifetime. With better theoretical understandings of the infinite width limit of neural networks in the feature learning regime [47], Yang et al. [48] have enabled transferring the best hyper-parameters from smaller networks to wider networks through better layer-dependent initializaitons and learning rates, which significantly reduces the cost of hyper-parameter tuning, enabling transferring the optimal hyper-parameter of a 40M model to a 6.7B GPT-3 model that outperforms the original 6.7B GPT-3 model. Although the analysis unveils parameterizations that enable transferring, the smaller networks still needs to be tuned to find best hyper-parameters. Some novel architectures like the original post-LN Transformer require tuning the learning rate schedules to converge well [23, 49], thus it will be more ideal to design a mechanism that alleviates tuning efforts for any architecture.

To be efficient, the model itself should have fewer computations in its forward and backward passes. The complexity of the attention blocks in Transformers scale quadratically with input sequence length, which limits its applications on long sequences of text

data and high resolution images. For graph neural networks, the number of neighborhoods needed for message passing scales exponentially with depth, making it difficult to preserve all the neighbors for message passing.

1.2 Organization

In this dissertation, we first introduce our efforts on understanding and improving the reliability of large-scale representation learning in Chapter 2. To draw attention to this issue, we first show a more vicious poisoning attack that transfers across architectures. We also propose an effective defense mechanism against such attacks in the supervised setting. Then, we introduce our efforts on improving the efficiency of representation learning, through improving the data efficiency (Chapter 3), convergence speed (Chapter 4) and computational complexity (Chapter 5).

1.3 Contributions

Our technical contributions are summarized as following.

- To call for attention to the threat of data poisoning attacks, we demonstrate an approach to produce transferable clean-label targeted poisoning attacks. Our method creates poison images that form a convex hull around the target image in the feature space of substitute models, which has better success rate than the previous feature collision attack [50] in the black-box setting, and it becomes even more powerful when enforced in multiple intermediate layers of the network, with high success rates in both transfer learning and end-to-end training contexts. We also show reg-

ularizations such as Dropout can further improve transferability.

- We propose a novel Deep k-NN defense for clean-label poisoning attacks in the supervised setting. We evaluate it against state-of-the-art clean-label data poisoning attacks, using a slate of architectures and show that our proposed strategy detects 99% of the poison instances without significant downgrade of overall performance.

- To improve the data efficiency, we propose a versatile adversarial training algorithm that applies successfully to multiple different domains where it can be difficult to find "legitimate" data augmentation. By accumulating the parameter gradients in every gradient ascent step on the input data, our algorithm significantly improves the generalization of models for Natural Language Understanding [51], Vision-and-Language tasks [52] and graph neural networks [53].

- To accelerate the convergence and improve the training stability, we propose an automated initialization scheme [54] that: (1) enables training 1202-layer ResNets without degeneration, (2) enables training the Post-LN Tranformer without learning rate warmup for both Adam and SGD. We also give a first-order analysis for hyper-parameter selection that alleviates the tuning efforts.

- To reduce the complexity of Transformer models, we propose a linear-complexity attention mechanism which integrates both the short-range window attention and the long-range low-rank approximated attention. We propose a DualLN technique to combine attentions from the two sources, which accelerate the convergence and improves the results. We also take the first step towards generalizing the low-rank

attentions to autoregressive models. Our model achieves state-of-the-art results on autoregressive language modeling, language understanding and ImageNet classification.

Chapter 2: Threat and Defense against Targeted Data Poisoning Attacks

In this chapter, we first show a more vicious poisoning attack against image classification models that transfers across architectures in Section 2.2, corresponding to our work [55]. Then, we show a highly effective defense against such attack in the supervised setting where training data is labeled in Section 2.3, corresponding to our work [56]. In principle, the attack could be applied to unsupervised settings, but the defense relies on labeled data.

2.1 Definition of Targeted Poisoning Attacks, the Threat Model

In this chapter, we consider the same threat model for both the attack and defense. Like the poisoning attack of [50], the attacker in our setting injects a small number of perturbed samples (whose labels are true to their class) into the training set of the victim. The attacker's goal is to cause the victim network, once trained, to classify a test image (not in the training set) as a specified class. We consider the case of image classification, where the attacker achieves its goal by adding adversarial perturbations $\boldsymbol{\delta}$ to the images. $\boldsymbol{\delta}$ is crafted so that the perturbed image $\boldsymbol{x} + \boldsymbol{\delta}$ shares the same class as the clean image $\boldsymbol{x}$ for a human labeler, while being able to change the decision boundary of the DNNs in a certain way.

Unlike [50] or [57], which requires full or query access to the victim model, here we assume the victim model is not accessible to the attacker, which is a practical assumption in many systems such as autonomous vehicles and surveillance systems. Instead, we need the attacker to have knowledge about the victim's training distribution, such that a similar training set can be collected for training substitute models.

We consider two learning approaches that the victim may adopt. The first learning approach is *transfer learning*, in which a pre-trained but frozen feature extractor ϕ, e.g., the convolutional layers of a ResNet [19] trained on a reference dataset like CIFAR or ImageNet, is applied to images, and an application-specific linear classifier with parameters $\boldsymbol{W}, \boldsymbol{b}$ is fine-tuned on the features $\phi(\mathcal{X})$ of another dataset $\mathcal{X}$. Transfer learning of this type is common in industrial applications when large sets of labeled data are unavailable for training a good feature extractor. Poisoning attacks on transfer learning were first studied in the white-box settings where the feature extractor is known in [50, 58], and similarly in [59, 60], all of which target linear classifiers over deep features.

The second learning approach is *end-to-end training*, where the feature extractor and the linear classifier are trained jointly. Obviously, such a setting has stricter requirements on the poisons than the transfer learning setting, since the injected poisons will affect the parameters of the feature extractor. [50] uses a watermarking strategy that superposes up to 30% of the target image onto about 40 poison images, only to achieve about 60% success rate in a 10-way classification setting.

2.2 A Transferable Attack

2.2.1 Overview

Deep neural networks require large datasets for training and hyper-parameter tuning. As a result, many practitioners turn to the web as a source for data, where one can automatically scrape large datasets with little human oversight. Unfortunately, recent results have demonstrated that these data acquisition processes can lead to security vulnerabilities. In particular, retrieving data from untrusted sources makes models vulnerable to *data poisoning attacks* wherein an attacker injects maliciously crafted examples into the training set in order to hijack the model and control its behavior.

We call for attention to this security concern. We explore effective and transferable *clean-label poisoning attacks* on image classification problems. In general, data poisoning attacks aim to control the model's behavior during inference by modifying its training data [50, 58, 61, 62]. In contrast to evasion attacks [41, 63, 64] and recently proposed backdoor attacks [65–67], we study the case where the targeted samples are *not* modified during inference.

Clean-label poisoning attacks differ from other poisoning attacks [60, 68] in a critical way: they do not require the user to have any control over the labeling process. Therefore, the poison images need to maintain their malicious properties even when labeled correctly by an expert. Such attacks open the door for a unique threat model in which the attacker poisons datasets simply by placing malicious images on the web, and waiting for them to be harvested by web scraping bots, social media platform operators, or

other unsuspecting victims. Poisons are then properly categorized by human labelers and used during training. Furthermore, targeted clean label attacks do not indiscriminately degrade test accuracy but rather target misclassification of specific examples, rendering the presence of the attack undetectable by looking at overall model performance.

Before this work, clean-label poisoning attacks have been demonstrated only in the white-box setting where the attacker has complete knowledge of the victim model, and uses this knowledge in the course of crafting poison examples [50, 61]. Black-box attacks of this type have not been explored; thus, we aim to craft clean-label poisons which transfer to unknown (black-box) deep image classifiers.

It has been demonstrated in evasion attacks that with only query access to the victim model, a substitute model can be trained to craft adversarial perturbations that fool the victim to classify the perturbed image into a specified class [57]. Compared to these attacks, transferable poisoning attacks remain challenging for two reasons. First, the victim's decision boundary trained on the poisoned dataset is more unpredictable than the unknown but fixed decision boundary of an evasion attack victim. Second, the attacker cannot depend on having direct access to the victim model (i.e. through queries) and must thus make the poisons model-agnostic. The latter also makes the attack more dangerous since the poisons can be administered in a distributed fashion (e.g. put on the web to be scraped), compromising more than just one particular victim model.

Here, we demonstrate an approach to produce transferable clean-label targeted poisoning attacks. We assume the attacker has no access to the victim's outputs or parameters, but is able to collect a similar training set as that of the victim. The attacker trains substitute models on this training set, and optimizes a novel objective that forces the poi-

sons to form a polytope in feature space that entraps the target inside its convex hull. A classifier that overfits to this poisoned data will classify the target into the same class as that of the poisons. This new objective has better success rate than feature collision [50] in the black-box setting, and it becomes even more powerful when enforced in multiple intermediate layers of the network, showing high success rates in both transfer learning and end-to-end training contexts. We also show that using Dropout when crafting the poisons improves transferability.

2.2.2 Revisit Feature Collision Attack

2.2.2.1 Definition

Feature collision attacks, as originally proposed in [50], are a reliable way of producing targeted clean-label poisons on white-box models. The attacker selects a base example $\boldsymbol{x}_b$ from the targeted class for crafting the poisons $\boldsymbol{x}_p$, and tries to make $\boldsymbol{x}_p$ become the same as the target $\boldsymbol{x}_t$ in the feature space by adding small adversarial perturbations to $\boldsymbol{x}_b$. Specifically, the attacker solves the following optimization problem (2.1) to craft the poisons:

$$\boldsymbol{x}_p = \arg\min_{\boldsymbol{x}} \|\boldsymbol{x} - \boldsymbol{x}_b\|^2 + \mu\|\phi(\boldsymbol{x}) - \phi(\boldsymbol{x}_t)\|^2, \tag{2.1}$$

where ϕ is a pre-trained neural feature extractor. The first term enforces the poison to lie near the base in input space, and therefore maintains the same label as $\boldsymbol{x}_b$ to a human labeler. The second term forces the feature representation of the poison to collide with

the feature representation of the target. The hyperparameter $\mu > 0$ trades off the balance between these terms.

If the poison example $\boldsymbol{x}_p$ is correctly labeled as a member of the targeted class and placed in the victim's training dataset χ, then after training on χ, the victim classifier learns to classify the poison's feature representation into the targeted class. Then, if $\boldsymbol{x}_p$'s feature distance to $\boldsymbol{x}_t$ is smaller than the margin of $\boldsymbol{x}_p$, $\boldsymbol{x}_t$ will be classified into the same class as $\boldsymbol{x}_p$ and the attack is successful. Sometimes more than one poison image is used to increase the success rate.

2.2.2.2 Improving Transferability with Ensembling

Unfortunately for the attacker, different feature extractors ϕ will lead to different feature spaces, and a small feature space distance $d_\phi(\boldsymbol{x}_p, \boldsymbol{x}_t) = \|\phi(\boldsymbol{x}_p) - \phi(\boldsymbol{x}_t)\|$ for one feature extractor does not guarantee a small distance for another.

Fortunately, the results in [69] have demonstrated that for each input sample, there exists an adversarial subspace for different models trained on the same dataset, such that a moderate perturbation can cause the models to misclassify the sample, which indicates it is possible to find a small perturbation to make the poisons $\boldsymbol{x}_p$ close to the target $\boldsymbol{x}_t$ in the feature space for different models, as long as the models are trained on the same dataset or similar data distributions.

With such observation, the most obvious approach to forge a black-box attack is to optimize a set of poisons $\{\boldsymbol{x}_p^{(j)}\}_{j=1}^k$ to produce feature collisions for an *ensemble* of models $\{\phi^{(i)}\}_{i=1}^m$, where m is the number of models in the ensemble. Such a technique

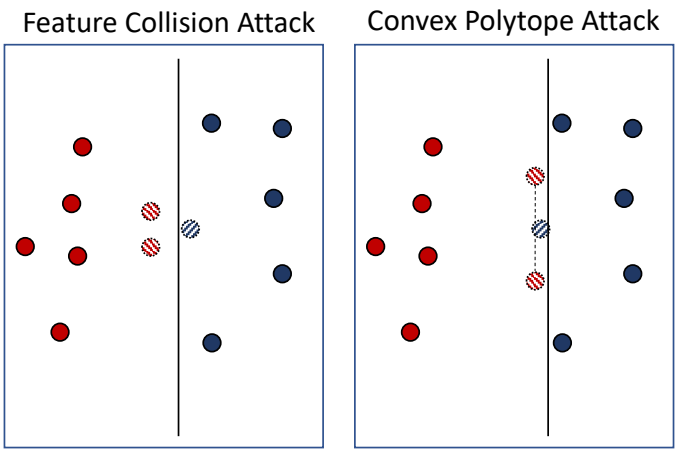

Figure 2.1: An illustrative example of a linear SVM trained on two-dimensional data with training sets poisoned by Feature Collision Attack and Convex Polytope Attack respectively.

was also used in black-box evasion attacks [70]. Because different extractors produce feature vectors with different dimensions and magnitudes, we use the following normalized feature distance to prevent any one network from dominating the objective due to such biases:

$$L_{FC} = \sum_{i=1}^{m} \sum_{j=1}^{k} \frac{\|\phi^{(i)}(\boldsymbol{x}_p^{(j)}) - \phi^{(i)}(\boldsymbol{x}_t)\|^2}{\|\phi^{(i)}(\boldsymbol{x}_t)\|^2}. \tag{2.2}$$

2.2.3 Challenges of Targeted Poisoning Attacks

Targeted poisoning attacks are more difficult to pull off than targeted evasion attacks. For image classification, targeted evasion attacks only need to make the victim model misclassify the perturbed image into a certain class. The model does not adjust to the perturbation, so the attacker only needs to find the shortest perturbation path to the decision boundary by solving a constrained optimization problem. For example, in the norm-bounded white-box setting, the attacker can directly optimize $\boldsymbol{\delta}$ to minimize the cross entropy loss L_{CE} on the target label y_t by solving $\boldsymbol{\delta}_t = \arg\min_{||\boldsymbol{\delta}||_\infty \leq \epsilon} L_{CE}(\boldsymbol{x}_t +$

$\boldsymbol{\delta}, y_{\mathrm{t}}$) with Projected Gradient Descent [42]. In the norm-bounded black-box setting, with query access, the attacker can also train a substitute model to simulate the behavior of the victim via distillation, and perform the same optimization w.r.t. the substitute model to find a transferable $\boldsymbol{\delta}$ [57].

Targeted poisoning attacks, however, face a more challenging problem. The attacker needs to get the victim to classify the target sample $\boldsymbol{x}_t$ into the alternative target class $\tilde{y}_t$ after being trained on the modified data distribution. One simple approach is to select the poisons from class $\tilde{y}_t$, and make the poisons as close to $\boldsymbol{x}_t$ as possible in the feature space. A rational victim will usually overfit the training set, since it is observed in practice that generalization keeps improving even after training loss saturates [71]. As a result, when the poisons are close to the target in the feature space, a rational victim is likely to classify $\boldsymbol{x}_t$ into $\tilde{y}_t$ since the space near the poisons are classified as $\tilde{y}_t$. However, as shown in Figure 2.1, smaller distance to the target does not always lead to a successful attack. In Figure 2.1, the two striped red dots are the poisons injected to the training set, while the striped blue dot is the target, which is not in the training set. All other points are in the training set. For feature collision attack, even when the poisons are the closest points to the target, the optimal linear SVM will classify the target correctly and the attack fails in the left figure. The Convex Polytope attack, to be introduced below, will enforce a small distance of the line segment formed by the two poisons to the target. When the line segment's distance to the target is minimized, the target's negative margin in the retrained model is also minimized if it overfits. In fact, being close to the target might be too restrictive for successful attacks. Indeed, there exists conditions where the poisons can be farther away from the target, but the attack is more successful.

2.2.4 Better transferability with Convex Polytope Attack

One problem with the feature collision attack is the emergence of obvious patterns of the target in the crafted perturbations. Unlike prevalent objectives for evasion attacks which maximize single-entry losses like cross entropy, feature collision [50] enforces each entry of the poison's feature vector $\phi(\boldsymbol{x}_p)$ to be close to $\phi(\boldsymbol{x}_t)$, which usually results in hundreds to thousands of constraints on each poison image $\boldsymbol{x}_p$. What is worse, in the black-box setting as Eq. 2.2, the poisoning objective forces a collision over an ensemble of m networks, which further increases the number of constraints on the poison images. With such a large number of constraints, the optimizer often resorts to pushing the poison images in a direction where obvious patterns of the target will occur, therefore making $\boldsymbol{x}_p$ look like the target class. As a result, human workers will notice the difference and take actions. Figure 2.2 shows a qualitative example in the process of crafting poisons from images of *hook* to attack a *fish* image with Eq. 2.2. Elements of the target image that are evident in the poison images include the fish's tail in the top image and almost a whole fish in the bottom image in column 3.

Another problem with Feature Collision Attack is its lack of transferability. Feature Collision Attack tends to fail in the black-box setting because it is difficult to make the poisons close to $\boldsymbol{x}_t$ for every model in the feature space. The feature space of different feature extractors should be very different, since neural networks are using one linear classifier to separate the deep features $\phi(\boldsymbol{x})$, which has a unique solution if $\phi(\boldsymbol{x})$ is given, but different networks usually have different accuracies. Therefore, even if the poisons $\boldsymbol{x}_p$ collide with $\boldsymbol{x}_t$ in the feature space of the substitute models, they probably do not collide

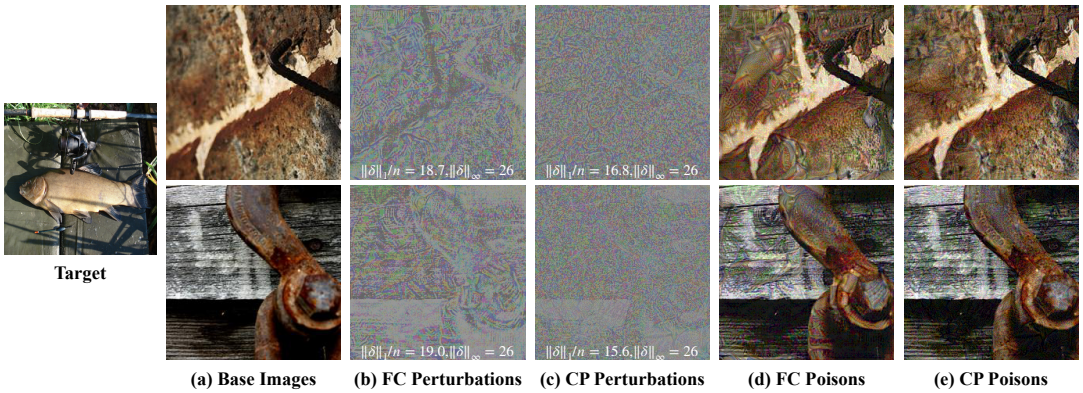

Figure 2.2: A qualitative example of the difference in poison images generated by Feature Collision (FC) Attack and Convex Polytope (CP) Attack. The target class's patterns are more obvious in the FC Attack poisons.

with $\boldsymbol{x}_t$ in the unknown target model, due to the generalization error. As demonstrated by Figure 2.1, the attack is likely to fail even when $\boldsymbol{x}_p$ has smaller distance to $\boldsymbol{x}_t$ than its intra-class samples. It is also impractical to ensemble too many substitute models to reduce such error. We provide experimental results with the ensemble Feature Collision Attack defined by Eq. 2.2 to show it can be ineffective.

We therefore seek a looser constraint on the poisons, so that the patterns of the target are not obvious in the poison images and the requirements on generalization are reduced. Noticing [50] usually use multiple poisons to attack one target, we start by deriving the necessary and sufficient conditions on the set of poison features $\{\phi(\boldsymbol{x}_p^{(j)})\}_{j=1}^k$ such that the target $\boldsymbol{x}_t$ will be classified into the poison's class.

Proposition 1. *The following statements are equivalent:*

1. *Every linear classifier that classifies $\{\phi(\boldsymbol{x}_p^{(j)})\}_{j=1}^k$ into label ℓ_p will classify $\phi(\boldsymbol{x}_t)$ into label ℓ_p.*

2. *$\phi(\boldsymbol{x}_t)$ is a convex combination of $\{\phi(\boldsymbol{x}_p^{(j)})\}_{j=1}^k$, i.e., $\phi(\boldsymbol{x}_t) = \sum_{j=1}^k c_j \phi(\boldsymbol{x}_p^{(j)})$, where*

$c_1, \ldots, c_k \geq 0, \sum_{j=1}^{k} c_j = 1.$

Proof. $2 \implies 1$:

For multi-class problems, the condition for $\phi(\boldsymbol{x})$ to be classified as ℓ_p is

$$\boldsymbol{w}_{\ell_p}^\top \phi(\boldsymbol{x}) + b_{\ell_p} > \boldsymbol{w}_i^\top \phi(\boldsymbol{x}) + b_i, \quad \text{for all } i \neq \ell_p.$$

Each of these constraints is linear, and is satisfied by a convex half-space. The region that satisfies all of these constraints in an intersection of convex half-spaces, and so is convex. Under condition (2), $\phi(\boldsymbol{x}_t)$ is a convex combination of points in this convex region, and so $\phi(\boldsymbol{x}_t)$ is itself in this convex region.

$1 \implies 2$:

Suppose that (1) holds. Let

$$\mathcal{S} = \{\sum_i c_i \phi(\boldsymbol{x}_p^j) | \sum_i c_i = 1, 0 \leq c_i \leq 1\}$$

be the convex hull of the points $\{\phi(\boldsymbol{x}_p^j)\}_{j=1}^k$. Let $\mathbf{u}_t = \arg\min_{\mathbf{u} \in \mathcal{S}} \|\mathbf{u} - \phi(\boldsymbol{x}_t)\|$ be the closest point to $\phi(\boldsymbol{x}_t)$ in $\mathcal{S}$. If $\|\mathbf{u}_t - \phi(\boldsymbol{x}_t)\| = 0$, then (2) holds and the proof is complete. If $\|\mathbf{u}_t - \phi(\boldsymbol{x}_t)\| > 0$, then define the classifier function

$$f(\mathbf{z}) = (\mathbf{u}_t - \phi(\boldsymbol{x}_t))^\top (\mathbf{z} - \mathbf{u}_t).$$

Clearly $f(\phi(\boldsymbol{x}_t)) < 0$. By condition (1), there is some j with $f(\phi(\boldsymbol{x}_p^j)) < 0$ as well.

Consider the function

$$g(\eta) = \frac{1}{2}\|\mathbf{u}_t + \eta(\phi(\boldsymbol{x}_p^j) - \mathbf{u}_t) - \phi(\boldsymbol{x}_t)\|^2.$$

Because $\mathbf{u}_t$ is the closest point to $\phi(\boldsymbol{x}_t)$ in $\mathcal{S}$, and g is smooth, the derivative of g with respect to η, evaluated at $\eta = 0$, is 0. We can write this derivative condition as

$$g'(0) = (\mathbf{u}_t - \phi(\boldsymbol{x}_t))^\top(\phi(\boldsymbol{x}_p^j) - \mathbf{u}_t) = f(\phi(\boldsymbol{x}_p^j)) \geq 0.$$

However this statement is a contradiction, since $f(\phi(\boldsymbol{x}_p^j)) < 0$. □

In words, a set of poisons from the same class is guaranteed to alter the class label of a target into theirs if that target's feature vector lies in the *convex polytope* of the poison feature vectors. We emphasize that this is a far more relaxed condition than enforcing a feature collision—it enables the poisons to lie much farther away from the target while altering the labels on a much larger region of space. As long as $\boldsymbol{x}_t$ lives inside this region in the unknown target model, and $\{\boldsymbol{x}_p^{(j)}\}$ are classified as expected, $\boldsymbol{x}_t$ will be classified as the same label as $\{\boldsymbol{x}_p^{(j)}\}$.

With this observation, we optimize the set of poisons towards forming a convex polytope in the feature space such that the target's feature vector will lie within or at least

close to the convex polytope. Specifically, we solve the following optimization problem:

$$\begin{aligned} &\underset{\{\boldsymbol{c}^{(i)}\},\{\boldsymbol{x}_p^{(j)}\}}{\text{minimize}} \frac{1}{2}\sum_{i=1}^{m} \frac{\|\phi^{(i)}(\boldsymbol{x}_t) - \sum_{j=1}^{k} c_j^{(i)}\phi^{(i)}(\boldsymbol{x}_p^{(j)})\|^2}{\|\phi^{(i)}(\boldsymbol{x}_t)\|^2} \\ &\text{subject to} \sum_{j=1}^{k} c_j^{(i)} = 1, c_j^{(i)} \geq 0, \forall i, j, \\ &\|\boldsymbol{x}_p^{(j)} - \boldsymbol{x}_b^{(j)}\|_\infty \leq \epsilon, \forall j, \end{aligned} \tag{2.3}$$

where $\boldsymbol{x}_b^{(j)}$ is the clean image of the j-th poison, and ϵ is the maximum allowable perturbation such that the perturbations are not immediately perceptible.

Eq. 2.3 simultaneously finds a set of poisons $\{\boldsymbol{x}_p^{(j)}\}$, and a set of convex combination coefficients $\{c_j^{(i)}\}$ such that the target lies in or close to the convex polytope of the poisons in the feature space of the m models. Notice the coefficients $\{c_j^{(i)}\}$ are untied, i.e., they are allowed to vary across different models, which does not require $\phi^{(i)}(\boldsymbol{x}_t)$ to be close to any specific point in the polytope, including the vertices $\{\boldsymbol{x}_p^{(j)}\}$. Given the same amount of perturbation, such an objective is also more relaxed than Feature Collision Attack (Eq. 2.2) since Eq. 2.2 is a special case of Eq. 2.3 when we fix $c_j^{(i)} = 1/k$. As a result, the poisons demonstrate almost no patterns of the target, and the imperceptibility of the attack is enhanced compared with feature collision attack, as shown in Figure 2.2. In Figure 2.2, Both attacks aim to make the model mis-classify the target *fish* image on the left into a *hook*. We show two of the five *hook* images that were used for the attack, along with their perturbations and the poison images here. Both attacks were successful, but unlike FC, which demonstrated strong regularity in the perturbations and obvious fish patterns in the poison images, CP tends to have no obvious pattern in its poisons.

The most important benefit brought by the convex polytope objective is the im-

proved transferability. For Convex Polytope Attack, $\boldsymbol{x}_t$ does not need to align with a specific point in the feature space of the unknown target model. It only needs to lie within the convex polytope formed by the poisons. In the case where this condition is not satisfied, Convex Polytope Attack still has advantages over Feature Collision Attack. Suppose for a given target model, a residual[1] smaller than ρ will guarantee a successful attack[2]. For Feature Collision Attack, the target's feature needs to lie within a ρ-ball centered at $\phi^{(t)}(\boldsymbol{x}_p^{(j^*)})$, where $j^* = \arg\min_j \|\phi^{(t)}(\boldsymbol{x}_p^{(j)}) - \phi^{(t)}(\boldsymbol{x}_t)\|$. For Convex Polytope Attack, the target's feature could lie within the ρ-expansion of the convex polytope formed by $\{\phi^{(t)}(\boldsymbol{x}_p^{(j)})\}_{j=1}^k$, which has a larger volume than the aforementioned ρ-ball, and thus tolerates larger generalization error.

2.2.5 An Efficient Algorithm for Convex Polytope Attack

We optimize the non-convex and constrained problem equation 2.3 using an alternating method that side-steps the difficulties posed by the complexity of $\{\phi^{(i)}\}$ and the convex polytope constraints on $\{\boldsymbol{c}^{(i)}\}$. Given $\{\boldsymbol{x}_p^{(j)}\}_{j=1}^k$, we use forward-backward splitting [72] to find the the optimal sets of coefficients $\{\boldsymbol{c}^{(i)}\}$. This step takes much less computation than back-propagation through the neural network, since the dimension of $\boldsymbol{c}^{(i)}$ is usually small (in our case a typical value is 5). Then, given the optimal $\{\boldsymbol{c}^{(i)}\}$ with respect to $\{\boldsymbol{x}_p^{(j)}\}$, we take one gradient step to optimize $\{\boldsymbol{x}_p^{(j)}\}$, since back-propagation through the m networks is relatively expensive. Finally, we project the poison images to be within ϵ units of the clean base image so that the perturbation is not obvious, which is

[1] For FC, it is $\min_j \|\phi^{(t)}(\boldsymbol{x}_p^{(j)}) - \phi^{(t)}(\boldsymbol{x}_t)\|$; for CP, it is $\|\sum_j c_j^{(t)} \phi^{(t)}(\boldsymbol{x}_p^{(j)}) - \phi^{(t)}(\boldsymbol{x}_t)\|$

[2] When the residual is small enough, $\phi^{(t)}(\boldsymbol{x}_t)$ will not cross the decision boundary if poisons are classified as expected.

implemented as a clip operation. We repeat this process to find the optimal set of poisons and coefficients, as shown in Algorithm 1.

In our experiments, we find that after the first iteration, initializing $\{\boldsymbol{c}^{(i)}\}$ to the value from the last iteration accelerates its convergence. We also find the loss in the target network to bear high variance without momentum optimizers. Therefore, we choose Adam [73] as the optimizer for the perturbations as it converges more reliably than SGD in our case. Although the constraint on the perturbation is ℓ_∞ norm, in contrast to [74] and the common practices for crafting adversarial perturbations such as FGSM [75], we do not take the sign of the gradient, which further reduces the variance caused by the flipping of signs when the update step is already small.

Algorithm 1 Convex Polytope Attack

Data: Clean base images $\{\boldsymbol{x}_b^{(j)}\}_{j=1}^k$, substitute networks $\{\phi^{(i)}\}_{i=1}^m$, and maximum perturbation ϵ.
Result: A set of perturbed poison images $\{\boldsymbol{x}_p^{(j)}\}_{j=1}^k$.
Initialize $\boldsymbol{c}^{(i)} \leftarrow \frac{1}{k}\mathbf{1}$, $\boldsymbol{x}_p^{(j)} \leftarrow \boldsymbol{x}_b^{(j)}$
while *not converged* **do**
  for *i=1,...,m* **do**
    $A \leftarrow [\phi^{(i)}(\boldsymbol{x}_p^{(1)}), \ldots, \phi^{(i)}(\boldsymbol{x}_p^{(k)})]$
    $\alpha \leftarrow 1/\|A^\top A\|_2$
    while *not converged* **do**
      $\boldsymbol{c}^{(i)} \leftarrow \boldsymbol{c}^{(i)} - \alpha A^\top (A\boldsymbol{c}^{(i)} - \phi^{(i)}(\boldsymbol{x}_t))$
      project $\boldsymbol{c}^{(i)}$ onto probability simplex
    end
  end
  Gradient step on $\boldsymbol{x}_p^{(j)}$ with Adam
  Clip $\boldsymbol{x}_p^{(j)}$ so that the infinity norm constraint is satisfied.
end

2.2.6 Multi-Layer Convex Polytope Attack

When the victim trains its feature extractor ϕ in addition to the classifier (last layer), enforcing Convex Polytope Attack only on the feature space of $\phi^{(i)}$ is not enough for a successful attack as we will show in experiments. In this setting, the change in feature space caused by a model trained on the poisoned data will also make the polytope more difficult to transfer.

Unlike linear classifiers, deep neural networks have much better generalization on image datasets. Since the poisons all come from one class, the whole network can probably generalize well enough to discriminate the distributions of $\boldsymbol{x}_t$ and $\boldsymbol{x}_p$, such that after trained with the poisoned dataset, it will still classify $\boldsymbol{x}_t$ correctly. As shown in Figure 2.7, when CP attack is applied to the last layer's features, the lower capacity models like SENet18 and ResNet18 are more susceptible to the poisons than other larger capacity models. However, we know there probably exist poisons with small perturbations that are transferable to networks trained on the poison distribution, as empirical evidence from [69] have shown there exist a common adversarial subspace for different models trained on the same dataset, and naturally trained networks usually have large enough Lipschitz to cause mis-classification [41], which hopefully will also be capable of shifting the polytope into such a subspace to lie close to $\boldsymbol{x}_t$.

One strategy to increase transferability to models trained end-to-end on the poisoned dataset is to jointly apply Convex Polytope Attack to multiple layers of the network. The deep network is broken into shallow networks $\phi_1, ..., \phi_n$ by depth, and the objective

now becomes

$$\underset{\{\boldsymbol{c}_l^{(i)}\},\{\boldsymbol{x}_p^{(j)}\}}{\text{minimize}} \sum_{l=1}^{n}\sum_{i=1}^{m} \frac{\|\phi_{1:l}^{(i)}(\boldsymbol{x}_t) - \sum_{j=1}^{k} c_{l,j}^{(i)} \phi_{1:l}^{(i)}(\boldsymbol{x}_p^{(j)})\|^2}{\|\phi_{1:l}^{(i)}(\boldsymbol{x}_t)\|^2}, \tag{2.4}$$

where $\phi_{1:l}^{(i)}$ is the concatenation from $\phi_1^{(i)}$ to $\phi_l^{(i)}$. Networks similar to ResNet are broken into blocks separated by pooling layers, and we let $\phi_l^{(i)}$ be the l-th layer of such blocks. The optimal linear classifier trained with the features up to $\phi_{1:l}$ ($l < n$) will have worse generalization than the optimal linear classifier trained with features of ϕ, and therefore the feature of $\boldsymbol{x}_t$ should have higher chance to deviate from the features of the same class after training, which is a necessary condition for a successful attack. Meanwhile, with such an objective, the perturbation is optimized towards fooling models with different depths, which further increases the variety of the substitute models and adds to the transferability.

2.2.7 Improved Transferability via Network Randomization

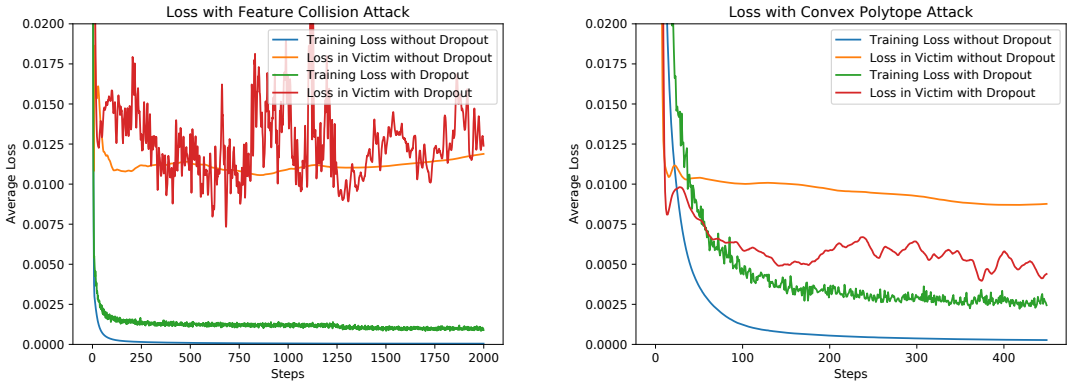

Figure 2.3: Loss curves of Feature Collision and Convex Polytope Attack on the substitute models and the victim models, tested using the target with index 2. Dropout improved the minimum achievable test loss for the FC attack, and improved the test loss of the CP attack significantly.

Even when trained on the same dataset, different models have different accuracy and therefore different distributions of the samples in the feature space. Ideally, if we craft the poisons with arbitrarily many networks from the function class of the target network then we should be able effectively minimize Eq. 2.3 in the target network. It is, however, impractical to ensemble a large number of networks due to memory constraints.

To avoid ensembling too many models, we randomize the networks with Dropout [76], turning it on when crafting the poisons. In each iteration, each substitute network $\phi^{(i)}$ is randomly sampled from its function class by shutting off each neuron with probability p, and multiplying all the "on" neurons with $1/(1-p)$ to keep the expectation unchanged. In this way, we can get an exponential (in depth) number of different networks for free in terms of memory. Such randomized networks increase transferability in our experiments. One qualititative example is given in Figure 2.3.

2.2.8 Experiments

In the following, we will use CP and FC as abbreviations for Convex Polytope Attacks and Feature Collision attacks respectively. The code for the experiments is available at `https://github.com/zhuchen03/ConvexPolytopePosioning`.

Datasets. In this section, all images come from the CIFAR10 dataset. If not explicitly specified, we take the first 4800 images from each of the 10 classes (a total of 48000 images) in the training set to pre-train the victim models and the substitute models ($\phi^{(i)}$). We leave the test set intact so that the accuracy of these models under different settings can be evaluated on the standard test set and compared directly with the

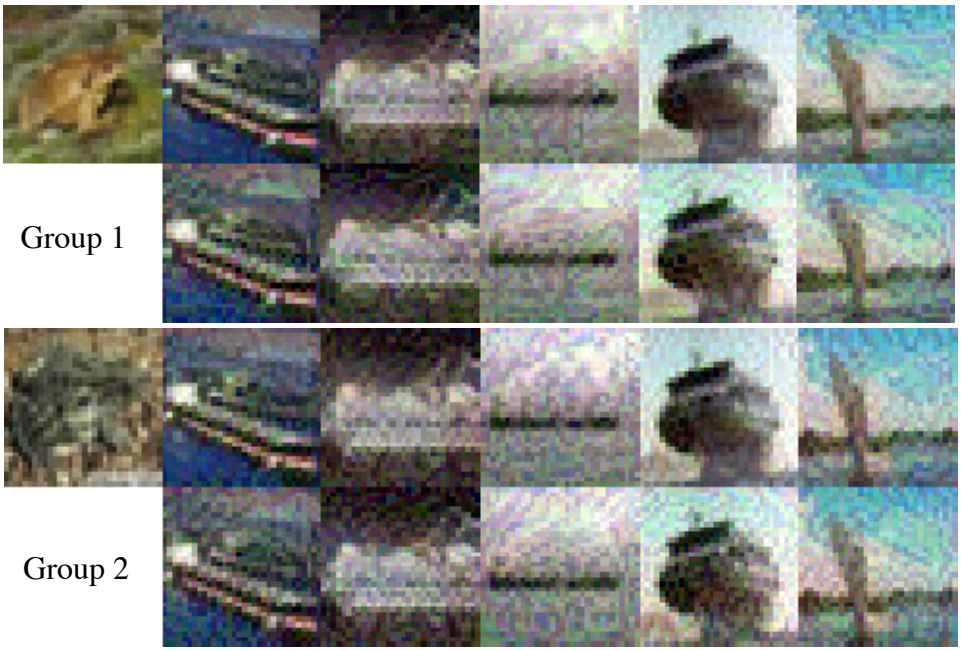

Figure 2.4: Qualitative results of the poisons crafted by FC and CP. Each group shows the target along with five poisons crafted to attack it, where the first row is the poisons crafted with FC, and the second row is the poisons crafted with CP. In the first group, the CP poisons fooled a DenseNet121 but FC poisons failed, in the second group both succeeded. The second target's image is more noisy and is probably an outlier of the frog class, so it is easier to attack. The poisons crafted by CP contain fewer patterns of $\boldsymbol{x}_t$ than with FC, and are harder to detect.

benchmarks. A successful attack should not only have unnoticeable image perturbations, but also unchanged test accuracy after fine-tuning on the clean and poisoned datasets. As shown in the supplementary, our attack preserves the accuracy of the victim model compared with the accuracy of those tuned on the corresponding clean dataset.

The remaining 2000 images of the training set serve as a pool for selecting the target, crafting the poisons, and fine-tuning the victim networks. We take the first 50 images from each class (a total of 500 images) in this pool as the clean fine-tuning dataset. This resembles the scenario where pretrained models on large datasets like Imagenet [77] are fine-tuned on a similar but usually disjoint dataset. We randomly selected "ship" as the target class, and "frog" as the targeted image's class, i.e., the attacker wants to cause a particular frog image to be misclassified as a ship. The poison images $\boldsymbol{x}_p^{(j)}$ across *all* experiments are crafted from the first 5 images of the ship class in the 500-image fine-tuning dataset. We evaluate the poison's efficacy on the next 50 images of the frog class. Each of these images is evaluated independently as the target $\boldsymbol{x}_t$ to collect statistics. Again, the target images are not included in the training and fine-tuning set.

Networks. Two sets of substitute model architectures are used in this paper. Set `S1` includes SENet18 [78], ResNet50 [19], ResNeXt29-2x64d [79], DPN92 [80], MobileNetV2 [81] and GoogLeNet [82]. Set `S2` includes all the architectures of `S1` except for MobileNetV2 and GoogLeNet. `S1` and `S2` are used in different experiments as specified below. ResNet18 and DenseNet121 [83] were used as the black-box model architectures. The poisons are crafted on models from the set of substitute model architectures. We evaluate the poisoning attack against victim models from the 6 different substitute model architectures as well as from the 2 black-box model architectures. Each victim

model, however, was trained with different random seeds than the substitute models. If the victim's architecture appears in the substitute models, we call it a gray-box setting; otherwise, it is a black-box setting.

We add a Dropout layer at the output of each Inception block for GoogLeNet, and in the middle of the convolution layers of each Residual-like blocks for the other networks. We train these models from scratch on the aforementioned 48000-image training set with Dropout probabilities of 0, 0.2, 0.25 and 0.3, using the same architecture and hyperparameters (except for Dropout) of a public repository[3]. The victim models that we evaluate were not trained with Dropout.

Attacks. We use the same 5 poison ship images to attack each frog image. For the substitute models, we use 3 models trained with Dropout probabilities of 0.2, 0.25, 0.3 from each architecture, which results in 18 and 12 substitute models for `S1` and `S2` respectively. When crafting the poisons, we use the same Dropout probability as the models were trained with. For all our experiments, we set $\epsilon = 0.1$. We use Adam [73] with a relatively large learning rate of 0.04 for crafting the poisons, since the networks have been trained to have small gradients on images similar to the training set. We perform no more than 4000 iterations on the poison perturbations in each experiment. Unless specified, we only enforce Eq. 2.3 on the features of the last layer.

For the victim, we choose its hyperparameters during fine-tuning such that it overfits the 500-image training set, which satisfies the aforementioned rational victim assumption. In the transfer learning setting, where only the final linear classifier is fine-tuned, we use Adam with a large learning rate of 0.1 to overfit. In the end-to-end setting, we use Adam

[3] https://github.com/kuangliu/pytorch-cifar

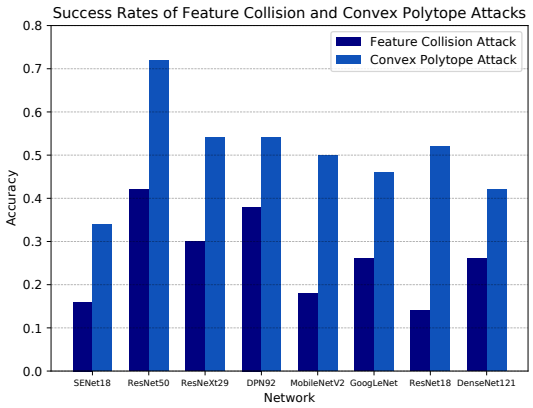

Figure 2.5: Success rates of FC and CP attacks on various models. Notice the first six entries are the gray-box setting where the models with same architecture but different weights are in the substitute networks, while the last two entries are the black-box setting.

with a small learning rate of 10^{-4} to overfit.

2.2.8.1 Comparison with Feature Collision

We first compare the transferability of poisons generated by FC and CP in the transfer learning training context. The results are shown in Figure 2.5. We use set `S1` of substitute architectures. FC never achieves a success rate higher than 0.5, while CP achieves success rates higher or close to 0.5 in most cases. A qualitative example of the poisons crafted by the two approaches is shown in Figure 2.4.

2.2.8.2 Importance of Training Set

Despite being much more successful than FC, questions remain about how reliable CP will be when we have no knowledge of the victim's training set. In the last section, we trained the substitute models on the same training set as the victim. In Figure 2.6 we provide results for when the substitute models' training sets are similar to (but mostly

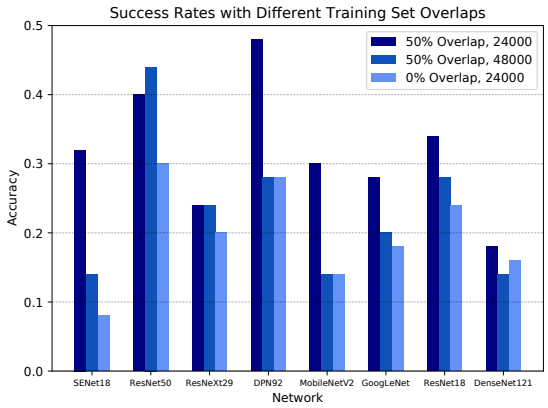

Figure 2.6: Success rates of Convex Polytope Attack, with poisons crafted by substitute models trained on the first 2400 images of each class of CIFAR10. The models corresponding to the three settings are trained with samples indexed from 1201 to 3600, 1 to 4800 and 2401 to 4800, corresponding to the settings of 50%, 50% and 0% training set overlaps respectively.

different from) that of the victim. Such a setting is sometimes more realistic than the setting where no knowledge of the victim's training set is required, but query access to the victim model is needed [57], since query access is not available for scenarios like surveillance. We use the less ideal `S2`, which has 12 substitute models from 4 different architectures. Results are evaluated in the transfer learning setting. Even with no data overlap, CP can still transfer to models with very different structure than the substitute models in the black-box setting. In the 0% overlap setting, the poisons transfer better to models with higher capacity like DPN92 and DenseNet121 than to low-capacity ones like SENet18 and MobileNetV2, probably because high capacity models overfit more to their training set. Overall, we see that CP may remain powerful without access to the training data in the transfer learning setting, as long as the victim's model has good generalization.

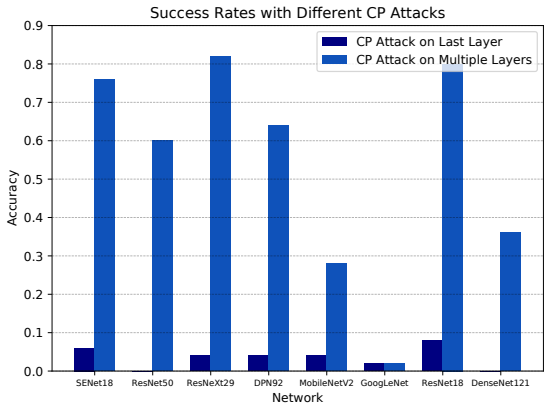

Figure 2.7: Success rates of Convex Polytope Attack in the end-to-end training setting. We use `S2` for the substitute models.

2.2.8.3 End-to-End Training

A more challenging setting is when the victim adopts end-to-end training. Unlike the transfer learning setting where models with better generalization turn out to be more vulnerable, here good generalization may help such models classify the target correctly despite the poisons. As shown in Figure 2.7, CP attacks on the last layer's feature is not enough for transferability, leading to almost zero successful attacks. It is interesting to see that the success rate on ResNet50 is 0, which is an architecture in the substitute models, while the success rate on ResNet18 is the highest, which is not an architecture in the substitute models but should have worse generalization.

Therefore, we enforce CP in multiple layers of the substitute models, which breaks the models into lower capacitie ones and leads to much better results. In the gray-box setting, all of the attacks achieved more than 0.6 success rates. However, it remains very difficult to transfer to GoogLeNet, which has a more different architecture than the substitute models. It is therefore more difficult to find a direction to make the convex

polytope survive end-to-end training.

2.3 A defense in the supervised setting

2.3.1 Overview

In this section, we initiate the study of defending against *clean-label* poisoning attacks on neural networks by considering feature collision [50] and convex polytope attacks [55] on the CIFAR-10 dataset. Although poison examples are not easily detected by human annotators, we exploit the property that adversarial examples have different feature distributions than their clean counterparts in higher layers of the network, and that those features often lie near the distribution of the target class. This intuition lends itself to a defense based on k nearest neighbors in feature space, in which the poison examples are detected and removed *prior* to training. Further, the parameter k yields a natural lever for trading off between the number of undetected poisons and number of discarded clean images when filtering the training set.

Our contributions can be outlined as follows.

- We propose a novel Deep k-NN defense for clean-label poisoning attacks. We evaluate it against state-of-the-art clean-label data poisoning attacks, using a slate of architectures and show that our proposed strategy detects 99% of the poison instances without significant downgrade of overall performance.

- We reimplement a set of general data poisoning defenses [84], including L_2-Norm Outliers, One-Class SVMs, Random Point Eviction, and Adversarial Training as

baselines and show that our proposed Deep k-NN defense is more robust at detection of poisons in the trained victim models.

- From the insights of two ablation studies, we assemble guidelines for implementing Deep k-NN in practice. First we provide instructions for picking an appropriate value for k. Second, we provide a protocol for using the Deep k-NN defense when class imbalance exists in the training set.

2.3.2 Intuition behind Deep k-NN Defense

As seen in Figure 2.8, poisons are surrounded by feature representations of the target class rather than of the base class. For instance, when $k = 3$ and $n_{poison} = 2$, each poison will almost always have a plurality of its neighbors as a non-poison in the target class. Since the plurality label of a poison's neighbors does not match the label of the poison itself, the poison can be removed from the dataset or simply not used for training. More generally, if $k > 2n_{poison}$, then we would expect the poisons to be outvoted by

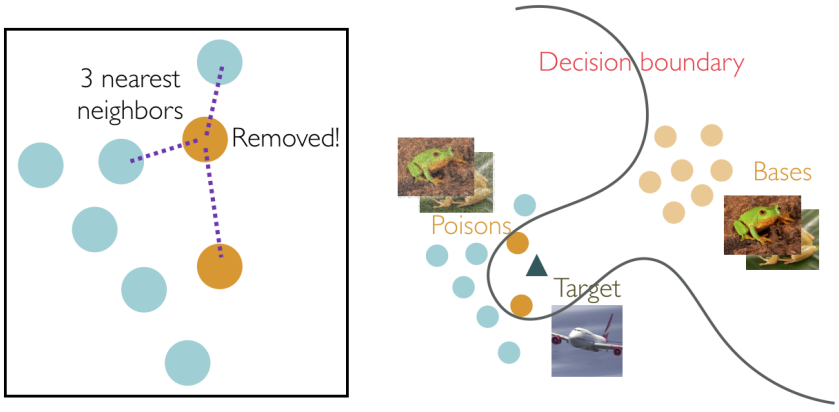

Figure 2.8: Proposed Deep k-NN defense ($k = 3$) correctly removing a poisoned example by comparing the class labels of poison with its k neighbors. Since a majority of the k points surrounding the poison do not share the same class label as the poison, it is removed.

members of the target class and be filtered from the training set. Note that by setting $k > 2n_{poison}$, the poisons' label cannot be the majority, but may still be the plurality, or mode, of the Deep k-NN set if the nearest neighbors of the current point are in multiple classes. Empirically, however, we do not observe this to be the case. Extracted features tend to be well-clustered by class; thus there are usually only 2 unique classes in the Deep k-NN neighborhood, base class and target class, with the target class being larger. Therefore, in order to successfully defend against adversarial manipulation, a victim needs only to set a sufficiently large value of k without needing to know exactly how many poisons there are *a-priori*. We further elucidate on the effect of k in Section 2.3.5.

2.3.3 Deep k-NN defense against Clean-Label Poisoning

In this section, we formally introduce the Deep k-NN defense as well as a set of other baseline defenses against clean-label targeted poisoning attacks. We compare the effectiveness of each defense against both feature collision attacks and convex polytope attacks in Section 2.3.4.

We use $\boldsymbol{x}_t$ to denote the input space representation of the target image that an adversary tries to misclassify. The target has true label l_t but the attacker seeks to misclassify it as having label l_b. We use $\boldsymbol{x}_b$ to denote a base image having label l_b that is used to build a poison after optimization. We use $\boldsymbol{x}_w$ to denote a base image watermarked with a target image, that is $\gamma \cdot \boldsymbol{x}_t + (1 - \gamma) \cdot \boldsymbol{x}_b$. To a human observer this image will retain the label l_b when γ is sufficiently low. We use $\phi(\boldsymbol{x})$ to denote the activations of the penultimate layer of a neural network. We refer to this as the *feature layer* or *feature space* and $\phi(\boldsymbol{x})$

as *features* of $\boldsymbol{x}$.

Deep k-NN Defense: For each data point in the training set, the Deep k-NN defense takes the plurality vote amongst the labels of that point's k nearest neighbors in feature space. If the point's own label is not the mode amongst labels of its k nearest neighbors, the point is flagged as anomalous, and is not used when training the model. We use Euclidean distance to measure the distance between data points in feature space. See Algorithm 2.

Algorithm 2 Deep k-NN Defense

Result: Filtered training set $X^{train'}$
Let $S_k(x^{(i)})$ denote a set of k points such that for all points $x^{(j)}$ inside the set and points $x^{(l)}$ outside the set, $|\phi(x^{(l)}) - \phi(x^{(i)})|_2 \geq |\phi(x^{(j)}) - \phi(x^{(i)})|_2$
$X^{train'} \leftarrow \{\}$
for *Data points* $\boldsymbol{x}^{(i)} \in X^{train}$ **do**
  Let l denote the label of $x^{(i)}$ and let $l(S_k(x^{(i)}))$ denote the labels of the points in $S_k(x^{(i)})$
  if $l \in mode(l(S_k(x^{(i)})))$ **then**
    $X^{train'} \leftarrow X^{train'} \cup \{x^{(i)}\}$;
  else
    Omit $x^{(i)}$ from $X^{train'}$;
  end
end

L2-Norm Outlier Defense: The L2 norm outlier defense removes an $\epsilon > 0$ fraction of points that are farthest in feature space from the centroids of their classes. For each class of label $l \in \mathcal{L}$, with size $s_l = |x^{(j)}$ s.t. $l(j) = l|$, we compute the centroid c_l as

$$c_l = \frac{1}{s_l} \sum_{x^{(j)} s.t. l(j)=l} \phi(x^{(j)})$$

and remove $\lfloor \epsilon s_l \rfloor$ points maximizing $|\phi(x^{(j)}) - c_l|_2$. The L2 norm defense relies on the position of the centroid to filter outliers. However, the position of the centroid itself is prone to data poisoning if the per-class data size is small. This defense is adapted from

traditional poison defenses not specific to neural networks [84].

One-Class SVM Defense: The one-class SVM defense examines the deep features of each class in isolation by applying the one-class SVM algorithm [85] to identify outliers in feature space for each label in the training set. It utilizes a radial basis kernel and is calibrated with a value $\nu = 0.01$.

Random Point Eviction Defense: The random point eviction defense is a simple experimental control. It filters out a random subset of all training data. We remove 1% of our training data for the feature collision attack and 10% of our training data on the convex polytope attack. If the poisoning attack is sensitive to poisons being removed, the random defense may be successful, at the cost of losing a proportionate amount of the unpoisoned training data.

Adversarial Training Defense: Thus far, we have only considered defenses which filter out examples prior to training. We consider here another defense strategy that does not involve filtering, but rather involves an alternative victim training procedure. Adversarial training, used often to harden networks against evasion attacks [42, 64], has been shown to produce neural network feature extractors which are less sensitive to weak features such as norm-bounded adversarial patterns [86]. We explore here whether a victim's use of an adversarially trained feature extractor would yield features that are robust to clean-label data poisoning. Instead of the conventional loss over the training set, adversarial training aims to optimize

$$\min_{\theta} \mathcal{L}_{\theta}(X + \delta^*), \text{where } \delta^* = \underset{\delta < \epsilon}{\operatorname{argmax}} \mathcal{L}_{\theta}(X + \delta),$$

where θ, X, and δ are the weights, training input, and adversarial perturbations, respectively, and $\mathcal{L}_\theta$ is some training loss (i.e., cross-entropy). In our experiments, we perform adversarial training following the standard procedure in [42], using an ℓ_∞ PGD adversary of 20 steps and $\epsilon = 8$.

2.3.4 Evaluation

In this section, we evaluate the effectiveness of our Deep k-NN defense and baseline defenses against the feature collision [50] and convex polytope [55] attacks on the CIFAR-10 dataset [87]. All model architectures, data splits, and hyperparameters are taken directly from the evaluation setups used in [50, 55]. We define the defense success rate as the number of times the poisoning attack fails to cause the target example to be misclassified, divided by the number of attempts. We only consider sets of poisons that lead to successful attacks in the undefended case so by definition the undefended defense success rate is 0%.

2.3.4.1 Defense against Feature Collision Attacks

Attack Procedure We randomly select 50 images in the base class. For each base image with input representation $\boldsymbol{x}_b$, we compute the watermark base $\boldsymbol{x}_w \leftarrow \gamma \cdot \boldsymbol{x}_t + (1 - \gamma) \cdot \boldsymbol{x}_b$, then optimize p with initial value $\boldsymbol{w}$ using a forward-backward splitting procedure to solve

$$\boldsymbol{x}_p = \arg\min_{\boldsymbol{x}} |\phi(\boldsymbol{x}) - \phi(\boldsymbol{x}_t)|_2^2 + \beta |\boldsymbol{x} - \boldsymbol{x}_w|_2^2$$

The hyperparameter β is fixed at 0.1. The resulting poisons $\boldsymbol{x}_p$ are both close to the target image $\boldsymbol{x}_t$ in feature space, and close to the watermarked input $\boldsymbol{x}_w$ in image space. To ensure statistical significance, we craft 16 of these collections of 50 poisons and evaluate each collection independently.

Defense Procedure As in the original setup [50], we first train a modified AlexNet to convergence using only clean data. Next we apply our defenses on the set of clean data plus poisons to obtain a filtered dataset. That filtered dataset is then used to fine tune the pretrained model over 10 epoch with a batch size of 128. We evaluate the performance of all defenses described in Section 2.3.3 against collections of 50 poisons that successfully cause a targeted misclassification.

Results As seen in Table 2.1, the Deep k-NN defense with $k = 5000$, successfully identifies all but one poison across multiple attacks, while filtering just 0.6% of the clean images from the training set. As a result, after victim training, models defended by Deep k-NN have defense success rates of 100%. In contrast, the $L2$-norm defense only identifies roughly half the feature collision poisons using $\epsilon = 0.01$. Both the One-Class SVM and the Random Point Eviction defenses are unable to detect a majority of the feature collision poisons.

Table 2.1: Comparing the effectiveness of baseline defenses aggregated for all model architectures in Feature Collision Attack

Defense Strategy	Poisons Removed	Clean Images Removed (%)	Defense Success Rate (%)	CIFAR-10 Test Accuracy (%)
Deep k-NN ($k = 5000$)	**799/800**	**0.6**	**100.0**	**74.6**
$L2$-Norm Outliers	395/800	1.0	50.0	**74.6**
One-class SVM	168/800	1.0	37.5	74.5
Random Point Eviction	84/800	10.0	12.5	74.5

2.3.4.2 Defense against Convex Polytope Attacks

Attack Procedure. Following the procedure in [55], the CIFAR-10 dataset is split into 48000 images for pretraining, and 500 images for fine-tuning. The poison base images are taken from the remaining split of 1500 images.

Since the attacker does not know the victim model parameters, they first pretrain their own model to convergence using the same subset of 48000 CIFAR-10 images used for pretraining. Next, an adversary uses this surrogate model to craft 5 poisons using the convex polytope method. To ensure statistical significance, 102 collections of 5 poisons are crafted.

When crafting convex polytope poisons, multiple surrogate models with different architectures are ensembled, so that the generated poisons generalize to victim architectures that the poisons were not crafted on. Our results are based on eight architectures: two of which are not used in crafting the poisons (black box setting), and six which use random initialization (grey box setting). The grey-box architectures are DPN92 [80], GoogLeNet [82], MobileNetV2 [81], ResNet50 [88], ResNeXT29-2x64d [79], and SENet18 [78], while the black-box architectures are DenseNet121 [83] and ResNet18 [88].

Defense Procedure. The victim model is first pretrained to convergence using a random initialization unknown to the attacker on the 48000 pretraining images from CIFAR-10[4]. Our defenses are applied to the 500 fine-tuning images plus poisons to obtain a filtered fine-tuning set[5]. Finally, this filtered dataset is used to fine-tune the victim

[4]We use conventional training loss for all except the adversarial training defense.
[5]There is no filtering in adversarial training.

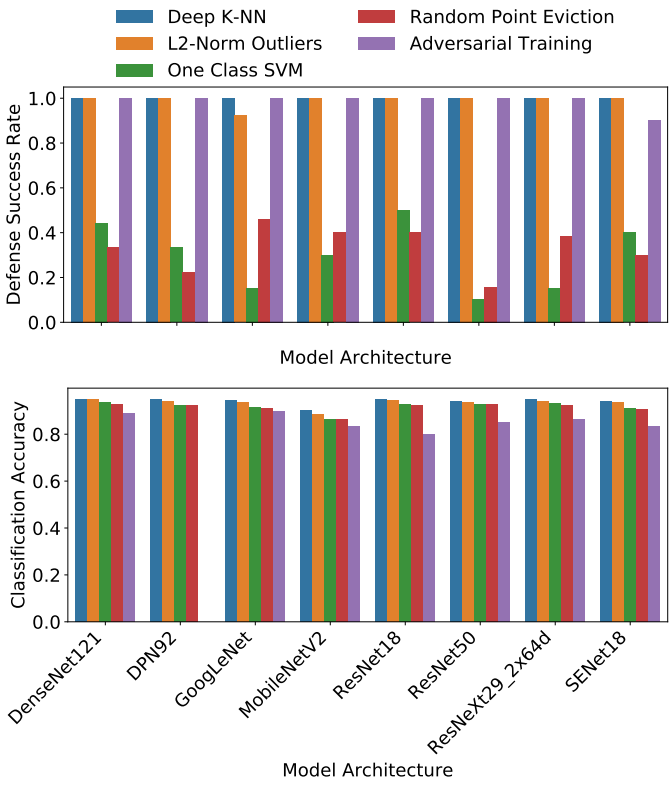

Figure 2.9: The Deep k-NN Defense is model-agnostic, achieving high defense success rate and test classification accuracy.

model.

Again, the performance of all defenses is reported only on collections of poisons that lead to a successful attack in the undefended case. Since the attacker did not have access to the victim architecture or model parameters during crafting of the poisons, the defenses are evaluated independently for each individual victim architecture.

Results. The aggregate results of each defense strategy on all 8 architectures are shown in Table 2.2. Both the Deep k-NN and $L2$-Norm defense filter out nearly all poisons, while incorrectly removing 4.3% and 9.1% of the clean training examples, respectively. Compared to feature collision poisons, convex polytope poisons trigger more false positive detections (i.e. clean images removed) across all defense methods, leading to fewer remaining clean examples and reduced test accuracy.

Surprisingly, the $L2$-Norm defense is much better able to detect convex polytope poisons compared to feature collision poisons; it detects almost as many as Deep k-NN . However, it has a lower specificity because it removes more clean images, resulting in half-percent lower test accuracy. These results are broken down for each victim architecture in Figure 2.9. The Deep k-NN attack is successful on all architectures with perfect defense success rate. $L2$-norm Outliers and Adversarial Training perform almost as well. Other strategies largely fail to be a viable defense.

We evaluate the effectiveness of adversarial training on the Convex Polytope-crafted poisons. In Table 2.2 and Figure 2.9, adversarially trained feature extractors—trained naively to provide resistance against only evasion attacks—do in fact help mitigate poisoning attacks as well. To our knowledge, this is the first time adversarial training has been shown to provide resistance against data poisoning (i.e. training time) attacks and is

a direction for future work.

The defense however significantly hurts test set accuracy (as is common for adversarially trained networks), which drops to 85% on average, compared with 94% on the same architectures without adversarial training. In scenarios when adversarial training for evasion attack robustness is not required, such as in situations when adversaries cannot control test time inputs, the Deep k-NN defense provides the poisoning resistance without the burden of decreased generalization performance.

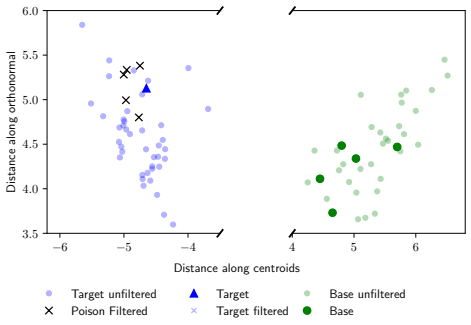

Figure 2.10: Feature space visualization of the Deep k-NN Defense against a Convex Polytope Attack on DPN92.

Table 2.2: Comparing the effectiveness of baseline defenses aggregated for all model architectures in Convex Polytope Attack

Defense Strategy	Poisons Removed	Clean Images Removed (%)	Defense Success Rate (%)	CIFAR-10 Test Accuracy (%)
Deep k-NN ($k = 50$)	**510/510**	**4.3**	**100.0**	**93.9**
$L2$-Norm Outliers	509/510	9.1	99.0	93.4
One-class SVM	114/510	7.1	29.9	91.7
Random Point Eviction	47/510	10.0	33.2	91.3
Adversarial Training	-	-	98.6	85.2

2.3.4.3 Feature Space Visualization

The favorable results of Deep k-NN defense also afford us an opportunity to understand anomaly detection in deep networks more generally via observing the effects

in feature representations. A feature space visualization of the penultimate layer of the network is shown in Figure 2.10, with both filtered poisons and non-poisons displayed.

Specifically, Figure 2.10 shows a projected visualization in the feature space of the fine tuning set in the target (blue) and base (green) classes.Following the projection scheme used in [50], where the x-axis is the direction along the line connecting the centroids of the target and base class features and the y-axis is the component of the parameter vector (i.e. decision boundary) orthogonal to the between-centroids vector, the deep features of the DPN92 network are projected into a two-dimensional plane. The "x" markers denote poisons that are filtered out by the defense and would have otherwise almost formed a convex polytope around the target (blue triangle). The Deep k-NN acts with high specificity: all the poisons are filtered, while only 2 outlying clean points in the target class (not shown) are also filtered. No points in the base class are filtered.

2.3.4.4 Limitations of the Deep k-NN Defense

The Deep k-NN defense exploits feature space clustering seen in feature collision and convex polytope attacks. It may not be as effective if this initial condition is not met. We view this as a strong and simple baseline defense for poisoning attacks that shows the need for more sophisticated and adaptive attacks.

2.3.5 Ablation Studies and Best Practices

We now turn to ablation studies to gain insight into best practices for using the Deep k-NN defense under realistic situations. All results are reported on the convex polytope

attack for CIFAR-10 as described in [55] on all 8 architectures discussed previously. We specifically focus on the convex polytope attack method since it is shown to act as a stronger poison on black-box threat models, and study the transfer learning case to mimic the common practice of using pre-trained feature-extractors trained on large datasets.

We again closely mimick the setup in [55] using the first 4800 images in each class to train a model from scratch and then using the next 50 images of each class (making a fine-tuning set size of 500) to fine-tune the model. The Adam optimizer with a learning rate of 0.1 is used. In both studies, we assign frogs as the target class and ships as the base class. The first 5 ship images from the fine-tuning set are replaced with the 5 poisoned ships. Each set of 5 poisoned ships has an associated target frog image that is neither in the training nor fine-tune set. We use the standard CIFAR-10 test split to measure test accuracy.

2.3.5.1 Choosing a Value of *k*

In our first study, we vary the value of *k* used in the Deep *k*-NN defense. Since dataset sizes vary, as well as the number of classes, we normalize *k* against the number of data points *per class*. Specifically, we measure all metrics against a normalized-*k* ratio, such that normalized-$k = k/N$ where *k* is the number of nearest neighbors considered by the Deep *k*-NN and *N* is the maximum number of examples for any class in the fine-tune set.

As seen in Figure 2.11 (top left and middle left), the defense success rate and MCC begin to reach maximum levels at normalized-$k = 0.2$, corresponding to an (unnormal-

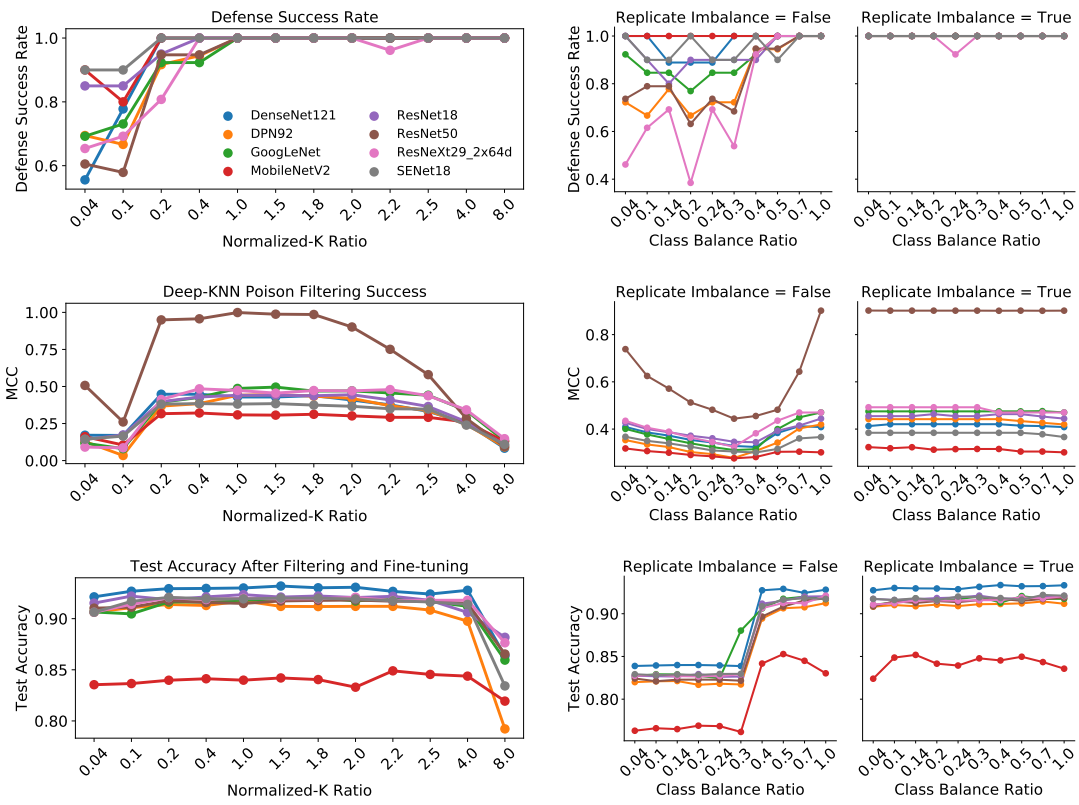

Figure 2.11: Ablation studies on the effect of k (left) and class imbalance (right). (Top Left) Defense success rate increases to 100% for all models as normalized-k ratio increases beyond 1.0 for all architectures. (Middle Left) Matthew's correlation coefficient is highest for all models when normalized-k ratio is between 0.4 and 2.0. (Bottom Left) Accuracy on the CIFAR-10 test split drops as normalized-k value increases beyond 4 times the number of examples per class. (Top Right) Defense and performance metrics under class imbalance. Defense success rate is stabilized when the target class training examples are first replicated to match the size of other classes. (Middle Right) Matthews correlation coefficient is also less dependent on the size of the target class when data replication is on. (Bottom Right) Test accuracy is highest when replicating the target examples to match the size of other classes.

ized) k of twice the number of poisons, $k = 10 = 2n_{poison}$. This confirms our intuition in Section 2.3.2: when $k > 2n_{poison}$, poisons will be marked anomalous since the poison class cannot be the majority label in the neighborhood and is unlikely to be the plurality because the neighborhood usually only contains two unique classes. Of course, the victim must set a value of k without knowledge of the number of poisons employed by

the attacker. Fortunately we observe that defense success rate remains at 100% as the normalized-*k* ratio increases beyond $k = 0.2$. Specifically, we see that after a normalized-*k* value greater than 1.0 ($k = 50$) (i.e. the situation where Deep *k*-NN considers more neighbors than the per-class number of examples) the convex polytope attack is ineffective on all models. However, there are limitations. Despite successfully detecting all the poisons, an extremely large *k* could lead to adverse effects on model test performance if too much clean data is removed (i.e. too many false positives).

To take both positives and negatives into account, we again invoke the MCC metric in Figure 2.11 (middle left) to measure the trade-off between detecting poisoned images and removing clean images. The maximum correlation coefficient for all models occurs for normalized-*k* values in the range of 1 and 2. This makes intuitive sense. On one hand, for *k* smaller than the class size, Deep *k*-NN could fail to look within a large enough neighborhood around a data point to properly judge its conformity. For example, a poison point may lie within a small, yet very tight cluster of other poison points of the same class and be improperly marked as benign even though the poison cluster itself may lie within a much larger cluster of clean target points. On the other hand, for *k* larger than 2 times the class size, the neighborhood may be too large and contain too many data points from a competing class. For example, the current target point may lie in a cluster of other target points, but since the neighborhood is so large that it contains all the target points as well as all the points in the nearby poison class cluster, the current target point will be improperly marked as anomalous.

This upper threshold of normalized-$k = 2$ is confirmed by looking at test accuracy performance in Figure 2.11 (bottom left). We note that performance is highest in the

normalized-k region from 0.2 to 2. It slightly decreases after a normalized-*k* ratio of 2 and sharply decreases after 4. This shows that a model's ability to generalize suffers when too many legitimate data points are removed under sufficiently large values of *k*. Based on these experiments, we recommend using a normalized-*k* value between 1 and 2 for optimal success in defending against poisoning attacks while minimizing false positives.

2.3.5.2 Dealing with Class Imbalance

In our second study, we consider the effectiveness of our defense on datasets with an imbalanced number of examples per class. Given an imbalanced dataset, the target class could be either the majority class or a minority class. The easiest case for the defender is when the target is the majority class. In this case, so long as *k* is set sufficiently large, there will be more than enough target training examples to cause the poisons in their midst (in feature space) to be marked as anomalous after running Deep *k*-NN . In this section, we will consider the worst case, wherein the target class is the smallest minority class in the dataset. Without applying any protocol to balance out the classes, there may not be enough target class neighbors when running Deep *k*-NN to know that the poisons clustered in their midst are anomalous.

A typical way to deal with imbalanced classes is to upweight the loss from examples in the minority classes or, equivalently, sample examples from minority classes at a higher rate that is inversely proportional to the fraction of the dataset that their class occupies. We consider a simple and equivalent modification of the latter protocol: given an imbalanced-class dataset, the examples in each class are *replicated* by a factor of N/n, where n is the

number of examples in that class and N is the maximum number of examples in any class. After this operation, the dataset will be larger, but once again balanced. We study the effect of this data replication protocol on imbalanced classes. Specifically, we set the number of examples in the target class (frog) to $n < N$ while leaving the number of examples in all other classes as N. We then replicate the frog examples by a factor of N/n such that its size match the size of the other classes. Finally, we plot the defense success rate against the class imbalance ratio n/N in Figure 2.11 (top right). The value of normalized-*k* is fixed at 2 ($k = 100$) for this experiment.

Figure 2.11 (top right, left panel) shows the defense success rate when no protocol is applied prior to running Deep *k*-NN : the success rate suffers for class balance ratios below 0.7. When our data replication protocol is applied before the Deep *k*-NN defense, the defense success rate is near perfect regardless of the class balance ratio. These results show that our minority class replication protocol, combined with the Deep *k*-NN defense, is very effective at removing poisons in an imbalanced class dataset. Our replication-based balancing protocol normalizes the number of examples considered by the Deep *k*-NN defense in feature space.

Next, we observe the MCC as a function of class imbalance in the absence of any protocol in Figure 2.11 (middle right, left panel). When the ratio is small, then the only thing that can hurt MCC is the misdetection of the targets as being anomalous. On the other hand, when the ratio is large, there is no class imbalance. MCC performs worst when there is a modest underrepresentation of the target class. That is where both the targets and the poisons can cause false negatives and false positives. When the replication protocol is applied in Figure 2.11 (middle right), the MCC experiences an improvement,

although the relative improvement is small. Interestingly, we observe that data replication stabilizes the MCC against class imbalance; the MCC is essentially a flat curve in Figure 2.11 (middle right).

All models experience better test accuracy on the CIFAR-10 test set when replicating target examples as shown in Figure 2.11 (bottom right). Despite only having n *unique* points in feature space, replicating them boosts model performance to be similar to the control experiment with a class balance ratio of 1.0. At lower class balance values, replicating data in unbalanced classes improves test accuracy by 8%. Based on these experiments, we recommend the protocol of replicating images of underrepresented classes to match the maximum number of examples in any particular class prior to running Deep *k*-NN . Defense success rate and model generalizability are both improved and stabilized by this protocol.

Chapter 3: Improving Data Efficiency with Adversarial Training

In this chapter, we revisit our works on using adversarial training as an augmentation to improve the generalization of models for Natural Language Understanding [89]. The techniques of this work has been successfully applied to Vision-and-Language tasks [52] and graph tasks [53].

Adversarial training was originally proposed as a means to enhance the security of machine learning systems [90], especially for safety-critical systems like self-driving cars [91]. During adversarial training, mini-batches of training samples are contaminated with adversarial perturbations (alterations that are small and yet cause misclassification), and then used to update network parameters until the resulting model learns to resist such attacks. Here, we turn our focus away from the security benefits of adversarial training, and instead study its effects on generalization. While adversarial training boosts the robustness, it is widely accepted by computer vision researchers that it is at odds with generalization [43], with classification accuracy on non-corrupted images dropping as much as 10% on CIFAR-10, and 15% on Imagenet [42, 92]. Surprisingly, people observe the opposite result for language models [51, 93, 94], vision-ang-language tasks [52] and graph tasks [53], showing that adversarial training can improve both generalization and robustness in these domains.

3.1 Related Works

3.1.1 Adversarial Training

To improve the robustness of neural networks against adversarial examples, many defense strategies and models have been proposed, in which PGD-based adversarial training [42] is widely considered to be the most effective, since it largely avoids the the obfuscated gradient problem [95]. It formulates a class of adversarial training algorithms [75] into solving a minimax problem on the cross-entropy loss, which can be achieved reliably through multiple projected gradient ascent steps followed by a SGD (Stochastic Gradient Descent) step.

Despite being verified by Athalye et al. [95] to avoid obfuscated gradients, Qin et al. [96] shows that PGD-based adversarial training still leads to highly convolved and non-linear loss surfaces when K is small, which could be readily broken under stronger adversaries. Thus, to be effective, the cost of PGD-based adversarial training is much higher than conventional training. To mitigate this cost, Shafahi et al. [97] proposed a "free" adversarial training algorithm that simultaneously updates both model parameters and adversarial perturbations on a single backward pass. Using a similar formulation, Zhang et al. [98] effectively reduce the total number of full forward and backward propagations for obtaining adversarial examples by restricting most of its adversarial updates in the first layer.

3.1.2 Adversarial Examples in Natural Languages

Adversarial examples have been explored primarily in the image domain, and received many attention in text domain recently. Previous works on text adversaries have focused on heuristics for creating adversarial examples in the black-box setting, or on specific tasks. Jia and Liang [99] propose to add distracting sentences to the input document in order to induce mis-classification. Zhao et al. [100] generate text adversaries by projecting the input data to a latent space using GANs, and searching for adversaries close to the original instance. Belinkov and Bisk [101] manipulate every word in a sentence with synthetic or natural noise in machine translation systems. Iyyer et al. [102] propose a neural paraphrase model based on back-translated data to produce paraphrases that have different sentence structures. Different from previous work, ours is not to produce actual adversarial examples, but only take the benefit of adversarial training for natural language understanding.

We are not the first to observe that robust language models may perform better on clean test data. Miyato et al. [93] extend adversarial and virtual adversarial training [103] to the text domain to improve the performance on semi-supervised classification tasks. Ebrahimi et al. [104] propose a character/word replacement for crafting attacks, and show employing adversarial examples in training renders the models more robust. Ribeiro et al. [105] show that adversarial attacks can be used as a valuable tool for debugging NLP models. Cheng et al. [94] also find that crafting adversarial examples can help neural machine translation significantly. Notably, these studies have focused on simple models or text generation tasks. Our work explores how to efficiently use the gradients

obtained in adversarial training to boost the performance of state-of-the-art transformer-based models.

3.2 Adversarial Training for Pre-trained Language Models

3.2.1 Where to add the perturbation?

Pre-trained large-scale language models, such as BERT [1], RoBERTa [4], ALBERT [5] and T5 [106], have proven to be highly effective for downstream tasks. We aim to further improve the generalization of these pre-trained language models on the downstream language understanding tasks by enhancing their robustness in the embedding space during finetuning on these tasks. We achieve this goal by creating "virtual" adversarial examples in the embedding space, and then perform parameter updates on these adversarial embeddings. Creating actual adversarial examples for language is difficult; even with state-of-the-art language models as guidance (e.g., [94]), it remains unclear how to construct label-preserving adversarial examples via word/character replacement without human evaluations, because the meaning of each word/character depends on the context [105]. Since we are only interested in the *effects* of adversarial training, rather than producing actual adversarial examples, we add norm-bounded adversarial perturbations to the embeddings of the input sentences using a gradient-based method. Note that our embedding-based adversary is strictly stronger than a more conventional text-based adversary, as our adversary can make manipulations on word embeddings that are not possible in the text domain.

For models that incorporate various input representations, including word or sub-

word embeddings, segment embeddings and position embeddings, our adversaries only modify the concatenated word or sub-word embeddings, leaving other components of the sentence representation unchanged. [1] Denote the sequence of one-hot representations of the input subwords as $\boldsymbol{Z} = [\boldsymbol{z}_1, \boldsymbol{z}_2, ..., \boldsymbol{z}_n]$, the embedding matrix as $\boldsymbol{V}$, and the language model (encoder) as a function $\boldsymbol{y} = f_{\boldsymbol{\theta}}(\boldsymbol{X})$, where $\boldsymbol{X} = \boldsymbol{V}\boldsymbol{Z}$ is the subword embeddings, $\boldsymbol{y}$ is the output of the model (e.g., class probabilities for classification models), and $\boldsymbol{\theta}$ denotes all the learnable parameters including the embedding matrix $\boldsymbol{V}$. We add adversarial perturbations $\boldsymbol{\delta}$ to the embeddings such that the prediction becomes $\boldsymbol{y}' = f_{\boldsymbol{\theta}}(\boldsymbol{X} + \boldsymbol{\delta})$. To preserve the semantics, we constrain the norm of $\boldsymbol{\delta}$ to be small, and assume the model's prediction should not change after the perturbation. This formulation is analogous to [93], with the difference that we do not require $\boldsymbol{X}$ to be normalized.

3.2.2 PGD-based Adversarial Training

Standard adversarial training seeks to find optimal parameters $\boldsymbol{\theta}^*$ to minimize the maximum risk for any $\boldsymbol{\delta}$ within a norm ball as:

$$\min_{\boldsymbol{\theta}} \mathbb{E}_{(\boldsymbol{Z},y)\sim\mathcal{D}} \left[\max_{\|\boldsymbol{\delta}\|\leq\epsilon} L(f_{\boldsymbol{\theta}}(\boldsymbol{X} + \boldsymbol{\delta}), y) \right], \tag{3.1}$$

where $\mathcal{D}$ is the data distribution, y is the label, and L is some loss function. We use the Frobenius norm to constrain $\boldsymbol{\delta}$. For neural networks, the outer "min" is non-convex, and the inner "max" is non-concave. Nonetheless, [42] demonstrated that this saddle-point problem can be solved reliably with SGD for the outer minimization and PGD (a stan-

[1] "Subword embeddings" refers to the embeddings of sub-word encodings such as the popular Byte Pair Encoding (BPE) [107].

dard method for large-scale constrained optimization, see [108] and [72]), for the inner maximization. In particular, for the constraint $\|\boldsymbol{\delta}\|_F \leq \epsilon$, with an additional assumption that the loss function is locally linear, PGD takes the following step (with step size α) in each iteration:

$$\boldsymbol{\delta}_{t+1} = \Pi_{\|\boldsymbol{\delta}\|_F \leq \epsilon} \left(\boldsymbol{\delta}_t + \alpha g(\boldsymbol{\delta}_t) / \|g(\boldsymbol{\delta}_t)\|_F \right), \tag{3.2}$$

where $g(\boldsymbol{\delta}_t) = \nabla_{\boldsymbol{\delta}} L(f_{\boldsymbol{\theta}}(\boldsymbol{X} + \boldsymbol{\delta}_t), y)$ is the gradient of the loss with respect to $\boldsymbol{\delta}$, and $\Pi_{\|\boldsymbol{\delta}\|_F \leq \epsilon}$ performs a projection onto the ϵ-ball. To achieve high-level robustness, multi-step adversarial examples are needed during training, which is computationally expensive. The K-step PGD (K-PGD) requires K forward-backward passes through the network, while the standard SGD update requires only one. As a result, the adversary generation step in adversarial training increases run-time by an order of magnitude—a catastrophic amount when training large state-of-the-art language models.

3.3 FreeLB: Better Efficiency through Gradient Accumulation

In this work [51], we propose a novel adversarial training algorithm, FreeLB, that promotes higher invariance in the embedding space, by adding adversarial perturbations to word embeddings and minimizing the resultant adversarial risk inside different regions around input samples. To validate the effectiveness of the proposed approach, we apply it to Transformer-based models for natural language understanding and commonsense reasoning tasks. Experiments on the GLUE benchmark show that when applied only to the finetuning stage, it is able to improve the overall test scores of BERT-base model from 78.3 to 79.4, and RoBERTa-large model from 88.5 to 88.8. In addition, the proposed

approach achieves state-of-the-art single-model test accuracies of 85.44% and 67.75% on ARC-Easy and ARC-Challenge. Extensive experiments further demonstrate that FreeLB can be generalized and boost the performance of RoBERTa-large and ALBERT on other tasks as well.

Algorithm 3 "Free" Large-Batch Adversarial Training (FreeLB-K)

Input: Training samples $X = \{(\boldsymbol{Z}, y)\}$, perturbation bound ϵ, learning rate τ, ascent steps K, ascent step size α

Initialize $\boldsymbol{\theta}$

for *epoch* $= 1 \ldots N_{ep}$ **do**

  for *minibatch* $B \subset X$ **do**

    $\boldsymbol{\delta}_0 \leftarrow \frac{1}{\sqrt{N_{\boldsymbol{\delta}}}} U(-\epsilon, \epsilon)$;

    $\boldsymbol{g}_0 \leftarrow 0$;

    for $t = 1 \ldots K$ **do**

      Accumulate gradient of parameters θ:

        $\boldsymbol{g}_t \leftarrow \boldsymbol{g}_{t-1} + \frac{1}{K} \mathbb{E}_{(\boldsymbol{Z}, y) \in B} [\nabla_{\boldsymbol{\theta}} L(f_{\boldsymbol{\theta}}(\boldsymbol{X} + \boldsymbol{\delta}_{t-1}), y)]$;

      Update the perturbation δ via gradient ascend:

        $\boldsymbol{g}_{adv} \leftarrow \nabla_{\boldsymbol{\delta}} L(f_{\boldsymbol{\theta}}(\boldsymbol{X} + \boldsymbol{\delta}_{t-1}), y)$;

        $\boldsymbol{\delta}_t \leftarrow \Pi_{\|\boldsymbol{\delta}\|_F \leq \epsilon}(\boldsymbol{\delta}_{t-1} + \alpha \cdot \boldsymbol{g}_{adv} / \|\boldsymbol{g}_{adv}\|_F)$;

    end

    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \tau \boldsymbol{g}_K$;

  end

end

In the inner ascent steps of PGD, the gradients of the parameters can be obtained with almost no overhead when computing the gradients of the inputs. From this observation, FreeAT [97] and YOPO [98] have been proposed to accelerate adversarial training. They achieve comparable robustness and generalization as standard PGD-trained models using only the same or a slightly larger number of forward-backward passes as natural training (i.e., SGD on clean samples). FreeAT takes one descent step on the parameters together with *each* of the K ascent steps on the perturbation. As a result, FreeAT may suffer from the "stale gradient" problem [109], where in every step t, $\boldsymbol{\delta}_t$ does not necessarily maximize the model with parameter $\boldsymbol{\theta}_t$ since its update is based on

$\nabla_{\boldsymbol{\delta}} L(f_{\boldsymbol{\theta}_{t-1}}(\boldsymbol{X}+\boldsymbol{\delta}_{t-1}), y)$, and vice versa, $\boldsymbol{\theta}_t$ does not necessarily minimize the adversarial risk with adversary $\boldsymbol{\delta}_t$ since its update is based on $\nabla_{\boldsymbol{\theta}} L(f_{\boldsymbol{\theta}_{t-1}}(\boldsymbol{X}+\boldsymbol{\delta}_{t-1}), y)$. Such a problem may be more significant when the step size is large.

Different from FreeAT, YOPO accumulates the gradient of the parameters from each of the ascent steps, and updates the parameters only once after the K inner ascent steps. YOPO also advocates that after each back-propagation, one should take the gradient of the first hidden layer as a constant and perform several additional updates on the adversary using the product of this constant and the Jacobian of the first layer of the network to obtain strong adversaries. However, when the first hidden layer is a linear layer as in their implementation, such an operation is equivalent to taking a larger step size on the adversary. The analysis backing the extra update steps also assumes a twice continuously differentiable loss, which does not hold for ReLU-based neural networks they experimented with, and thus the reasons for the success of such an algorithm remains obscure. We give empirical comparisons between YOPO and our approach in Sec. 3.5.3.

To obtain better solutions for the inner max and avoid fundamental limitations on the function class, we propose FreeLB, which performs multiple PGD iterations to craft adversarial examples, and simultaneously accumulates the "free" parameter gradients $\nabla_{\boldsymbol{\theta}} L$ in each iteration. After that, it updates the model parameter $\boldsymbol{\theta}$ all at once with the accumulated gradients. The overall procedure is shown in Algorithm 3, in which $\boldsymbol{X}+\boldsymbol{\delta}_t$ is an approximation to the local maximum within the intersection of two balls $\mathcal{I}_t = \mathcal{B}_{\boldsymbol{X}+\boldsymbol{\delta}_0}(\alpha t) \cap \mathcal{B}_{\boldsymbol{X}}(\epsilon)$. By taking a descent step along the averaged gradients at

$\boldsymbol{X}+\boldsymbol{\delta}_0, ..., \boldsymbol{X}+\boldsymbol{\delta}_{K-1}$, we approximately optimize the following objective:

$$\min_{\boldsymbol{\theta}} \mathbb{E}_{(\boldsymbol{Z},y)\sim\mathcal{D}} \left[\frac{1}{K} \sum_{t=0}^{K-1} \max_{\boldsymbol{\delta}_t \in \mathcal{I}_t} L(f_{\boldsymbol{\theta}}(\boldsymbol{X}+\boldsymbol{\delta}_t), y) \right], \tag{3.3}$$

which is equivalent to replacing the original batch $\boldsymbol{X}$ with a K-times larger virtual batch, consisting of samples whose embeddings are $\boldsymbol{X}+\boldsymbol{\delta}_0, ..., \boldsymbol{X}+\boldsymbol{\delta}_{K-1}$. Compared with PGD-based adversarial training (Eq. 3.1), which minimizes the maximum risk at a single estimated point in the vicinity of each training sample, FreeLB minimizes the maximum risk at each ascent step at almost no overhead.

Intuitively, FreeLB could be a learning method with lower generalization error than PGD. [110] have proved that the generalization error of a learning method invariant to a set of T transformations may be up to $\sqrt{T}$ smaller than a non-invariant learning method. According to their theory, FreeLB could have a more significant improvement over natural training, since FreeLB enforces the invariance to K adversaries from a set of up to K different norm constraints,[2] while PGD only enforces invariance to a single norm constraint ϵ.

Empirically, FreeLB does lead to higher robustness and invariance than PGD in the embedding space, in the sense that the maximum increase of loss in the vicinity of $\boldsymbol{X}$ for models trained with FreeLB is smaller than that with PGD. See Sec. 3.5.3 for details. In theory, such improved robustness can lead to better generalization [111], which is consistent with our experiments. [96] also demonstrated that PGD-based method leads to highly convolved and non-linear loss surfaces in the vicinity of input samples when K is

[2]The cardinality of the set is approximately $\min\{K, \lceil \frac{\epsilon - \mathbb{E}[\|\delta_0\|]}{\alpha} \rceil + 1\}$.

small, indicating a lack of robustness.

3.4 Improving Dropout for Adversarial Training

Usually, adversarial training is not used together with dropout [112]. However, for some language models like RoBERTa [4], dropout is used during the finetuning stage. In practice, when dropout is turned on, each ascent step of Algorithm 3 is optimizing $\boldsymbol{\delta}$ for a different network. Specifically, denote the dropout mask as $\boldsymbol{m}$ with each entry $m_i \sim \text{Bernoulli}(p)$. Similar to our analysis for FreeAT, the ascent step from $\boldsymbol{\delta}_{t-1}$ to $\boldsymbol{\delta}_t$ is based on $\nabla_{\boldsymbol{\delta}} L(f_{\boldsymbol{\theta}(\boldsymbol{m}_{t-1})}(\boldsymbol{X} + \boldsymbol{\delta}_{t-1}), y)$, so $\boldsymbol{\delta}_t$ is sub-optimal for $L(f_{\boldsymbol{\theta}(\boldsymbol{m}_t)}(\boldsymbol{X} + \boldsymbol{\delta}), y)$. Here $\boldsymbol{\theta}(\boldsymbol{m})$ is the effective parameters under dropout mask $\boldsymbol{m}$.

The more plausible solution is to use the same $\boldsymbol{m}$ in each step. When applying dropout to any network, the objective for $\boldsymbol{\theta}$ is to minimize the expectation of loss under different networks determined by the dropout masks, which is achieved by minimizing the Monte Carlo estimation of the expected loss. In our case, the objective becomes:

$$\min_{\boldsymbol{\theta}} \mathbb{E}_{(\boldsymbol{Z},\boldsymbol{y})\sim\mathcal{D},\boldsymbol{m}\sim\mathcal{M}} \left[\frac{1}{K} \sum_{t=0}^{K-1} \max_{\boldsymbol{\delta}_t \in \mathcal{I}_t} L(f_{\boldsymbol{\theta}(\boldsymbol{m})}(\boldsymbol{X} + \boldsymbol{\delta}_t), y) \right], \tag{3.4}$$

where the 1-sample Monte Carlo estimation should be $\frac{1}{K}\sum_{t=0}^{K-1} \max_{\boldsymbol{\delta}_t \in \mathcal{I}_t} L(f_{\boldsymbol{\theta}(\boldsymbol{m}_0)}(\boldsymbol{X} + \boldsymbol{\delta}_t), y)$ and can be minimized by using FreeLB with dropout mask $\boldsymbol{m}_0$ in each ascent step. This is similar to applying Variational Dropout to RNNs as used in [113].

3.5 Experiments for FreeLB

In this section, we provide comprehensive analysis on FreeLB through extensive experiments on three Natural Language Understanding benchmarks: GLUE [114], ARC [115] and CommonsenseQA [116]. We also compare the robustness and generalization of FreeLB with other adversarial training algorithms to demonstrate its strength. Additional experimental details are provided in the Appendix.

3.5.1 Datasets

GLUE Benchmark. The GLUE benchmark is a collection of 9 natural language understanding tasks, namely Corpus of Linguistic Acceptability (CoLA; [117]), Stanford Sentiment Treebank (SST; [118]), Microsoft Research Paraphrase Corpus (MRPC; [119]), Semantic Textual Similarity Benchmark (STS; [120]), Quora Question Pairs (QQP; [121]), Multi-Genre NLI (MNLI; [122]), Question NLI (QNLI; [123]), Recognizing Textual Entailment (RTE; [124]; [125]; [126]; [127]) and Winograd NLI (WNLI; [128]). 8 of the tasks are formulated as classification problems and only STS-B is formulated as regression, but FreeLB applies to all of them. For BERT-base, we use the HuggingFace implementation[3], and follow the single-task finetuning procedure as Devlin et al. [1]. For RoBERTa, we use the fairseq implementation[4]. Same as Liu et al. [4], we also use single-task finetuning for all dev set results, and start with MNLI-finetuned models on RTE, MRPC and STS-B for the test submissions.

[3]https://github.com/huggingface/pytorch-transformers
[4]https://github.com/pytorch/fairseq

ARC Benchmark. The ARC dataset [115] is a collection of multi-choice science questions from grade-school level exams. It is further divided into ARC-Challenge set with 2,590 question answer (QA) pairs and ARC-Easy set with 5,197 QA pairs. Questions in ARC-Challenge are more difficult and cannot be handled by simply using a retrieval and co-occurence based algorithm [115]. A typical question is:

Which property of a mineral can be determined just by looking at it?

(A) luster [correct] *(B) mass (C) weight (D) hardness.*

CommonsenseQA Benchmark. The CommonsenseQA dataset [116] consists of 12,102 natural language questions that require human commonsense reasoning ability to answer. A typical question is :

Where can I stand on a river to see water falling without getting wet?

(A) waterfall, ***(B) bridge [correct]****, (C) valley, (D) stream, (E) bottom.*

Each question has five candidate answers from ConceptNet [129]. To make the question more difficult to solve, most answers have the same relation in ConceptNet to the key concept in the question. As shown in the above example, most answers can be connected to "river" by "AtLocation" relation in ConceptNet. For a fair comparison with the reported results in papers and leaderboard[5], we use the official random split 1.11.

Task-specific training details. For tasks with ranking loss like ARC, CommonsenseQA, WNLI and QNLI, add the perturbation to the concatenation of the embeddings of all question/answer pairs.

Additional tricks are required to achieve high performance on WNLI and QNLI for the GLUE benchmark. We use the same tricks as [4]. For WNLI, we use the same WSC

[5]https://www.tau-nlp.org/csqa-leaderboard

Table 3.1: Results (median and variance) on the dev sets of GLUE based on the RoBERTa-large model, from 5 runs with the same hyperparameter but different random seeds. ReImp is our reimplementation of RoBERTa-large. The training process can be very unstable even with the vanilla version. Here, both PGD on STS-B and FreeAT on RTE demonstrates such instability, with one unconverged instance out of five.

Method	MNLI (Acc)	QNLI (Acc)	QQP (Acc)	RTE (Acc)	SST-2 (Acc)	MRPC (Acc)	CoLA (Mcc)	STS-B (Pearson)
Reported	90.2	94.7	92.2	86.6	96.4	90.9	68.0	92.4
ReImp	-	-	-	85.61 (1.7)	96.56 (.3)	90.69 (.5)	67.57 (1.3)	92.20 (.2)
PGD	90.53 (.2)	94.87 (.2)	92.49 (.07)	87.41 (.9)	96.44 (.1)	90.93 (.2)	69.67 (1.2)	92.43 (7.)
FreeAT	90.02 (.2)	94.66 (.2)	92.48 (.08)	86.69 (15.)	96.10 (.2)	90.69 (.4)	68.80 (1.3)	92.40 (.3)
FreeLB	**90.61** (.1)	**94.98** (.2)	**92.60** (.03)	**88.13** (1.2)	**96.79** (.2)	**91.42** (.7)	**71.12** (.9)	**92.67** (.08)

Table 3.2: Results on GLUE from the evaluation server, as of Sep 25, 2019. Metrics are the same as the leaderboard. Number under each task's name is the size of the training set. FreeLB-BERT is the single-model results of BERT-base finetuned with FreeLB, and FreeLB-RoB is the ensemble of 7 RoBERTa-Large models for each task. References: [1]: [1]; [2]: [2]; [3]: [3]; [4]: [4].

Model	Score	CoLA 8.5k	SST-2 67k	MRPC 3.7k	STS-B 7k	QQP 364k	MNLI-m/mm 393k	QNLI 108k	RTE 2.5k	WNLI 634	AX
BERT-base[1]	78.3	52.1	93.5	88.9/84.8	87.1/85.8	71.2/89.2	84.6/83.4	90.5	66.4	65.1	34.2
FreeLB-BERT	79.4	54.5	93.6	88.1/83.5	87.7/86.7	72.7/89.6	85.7/84.6	91.8	70.1	65.1	36.9
MT-DNN[2]	87.6	**68.4**	96.5	92.7/90.3	91.1/90.7	73.7/89.9	87.9/87.4	96.0	86.3	89.0	42.8
XLNet-Large[3]	88.4	67.8	**96.8**	93.0/90.7	91.6/91.1	74.2/90.3	90.2/89.8	98.6	86.3	**90.4**	47.5
RoBERTa[4]	88.5	67.8	96.7	92.3/89.8	92.2/91.9	74.3/90.2	90.8/90.2	**98.9**	88.2	89.0	48.7
FreeLB-RoB	**88.8**	68.0	**96.8**	**93.1/90.8**	**92.4/92.2**	**74.8/90.3**	**91.1/90.7**	98.8	**88.7**	89.0	**50.1**
Human	87.1	66.4	97.8	86.3/80.8	92.7/92.6	59.5/80.4	92.0/92.8	91.2	93.6	95.9	-

data provided by [4] for training. For testing, [4] also provided the test set with span annotations, but the order is different form the GLUE dataset. We re-order their test set by matching. For the QNLI, we follow [4] and formulate the problem as pairwise ranking problem, which is the same for CommonsenseQA. We find the matching pairs for both training set and testing set by matching the queries in the dev set. We predict "entailment" if the candidate has the higher score, and "not_entailment" otherwise.

3.5.2 Results

GLUE. We summarize results on the dev sets of GLUE in Table 3.1, comparing the proposed FreeLB against other adversatial training algorithms (PGD [42] and FreeAT [97]). We use the same step size α and number of steps m for PGD, FreeAT and FreeLB. FreeLB is consistently better than the two baselines. Comparisons and detailed discussions about YOPO [98] are provided in Sec. 3.5.3. We have also submitted our results to the evaluation server, results provided in Table 3.2. FreeLB lifts the performance of the BERT-base model from 78.3 to 79.4, and RoBERTa-large model from 88.5 to 88.8 on overall scores.

ARC. For ARC, a corpus of 14 million related science documents (from ARC Corpus, Wikipedia and other sources) is provided. For each QA pair, we first use a retrieval model to select top 10 related documents. Then, given these retrieved documents[6], we use RoBERTa-large model to encode $\langle s \rangle$ *Retrieved Documents* $\langle /s \rangle$ *Question* + *Answer* $\langle /s \rangle$, where $\langle$s$\rangle$ and $\langle$/s$\rangle$ are special tokens for RoBERTa model[7]. We then apply a fully-connected layer to the representation of the [CLS] token to compute the final logit, and use standard cross-entropy loss for model training.

Results are summarized in Table 3.3. Following Sun et al. [130], we first finetune the RoBERTa model on the RACE dataset [131]. The finetuned RoBERTa model achieves 85.70% and 85.24% accuracy on the development and test set of RACE, respectively. Based on this, we further finetune the model on both ARC-Easy and ARC-Challenge

[6]We thank AristoRoBERTa team for providing retrieved documents and additional Regents Living Environments dataset.

[7]Equivalent to [CLS] and [SEP] token in BERT.

Table 3.3: Results on ARC and CommonsenseQA (CQA). ARC-Merge is the combination of ARC-Easy and ARC-Challenge, "MTL" stands for multi-task learning and "Ens" stands for ensemble. Results of XLNet + RoBERTa (MTL+Ens) and AristoRoBERTaV7 (MTL) are from the ARC leaderboards. Test (E) denotes the test set results with ensembles. For CQA, we report the highest dev and test accuracies among *all* models. The models with 78.81/72.19 dev/test accuracy (as in the table) have 71.84/78.64 test/dev accuracies respectively.

	ARC-Easy		ARC-Challenge		ARC-Merge		CQA		
	Dev	Test	Dev	Test	Dev	Test	Dev	Test	Test (E)
RoBERTa (Reported)	-	-	-	-	-	-	78.43	72.1	72.5
RoBERTa (ReImp)	84.39	84.13	64.54	64.44	77.83	77.62	77.56	-	-
FreeLB-RoBERTa	**84.91**	84.81	65.89	65.36	78.37	78.39	**78.81**	**72.2**	**73.1**
AristoRoBERTaV7 (MTL)	-	85.02	-	66.47	-	78.89	-	-	-
XLNet + RoBERTa (MTL+Ens)	-	-	-	67.06	-	-	-	-	-
FreeLB-RoBERTa (MTL)	**84.91**	**85.44**	**70.23**	**67.75**	**79.86**	**79.60**	-	-	-

datasets with the same hyper-parameter searching strategy (for 5 epochs), which achieves 84.13%/64.44% test accuracy on ARC-Easy/ARC-Challenge. And by adding FreeLB finetuning, we can reach 84.81%/65.36%, a significant boost on ARC benchmark, demonstrating the effectiveness of FreeLB.

To further improve the results, we apply a multi-task learning (MTL) strategy using additional datasets. We first finetune the model on RACE [131], and then finetune on a joint dataset of ARC-Easy, ARC-Challenge, OpenbookQA [132] and Regents Living Environment[8]. Based on this, we further finetune our model on ARC-Easy and ARC-Challenge with FreeLB. After finetuning, our single model achieves 67.75% test accuracy on ARC-Challenge and 85.44% on ARC-Easy, both outperforming the best submission on the official leaderboard[9].

CommonsenseQA. Similar to the training strategy in Liu et al. [4], we construct five inputs for each question by concatenating the question and each answer separately,

[8]https://www.nysedregents.org/livingenvironment

[9]https://leaderboard.allenai.org/arc/submissions/public and https://leaderboard.allenai.org/arc_easy/submissions/public

then encode each input with the representation of the [CLS] token. A final score is calculated by applying the representation of [CLS] to a fully-connected layer. Following the fairseq repository[10], the input is formatted as: "⟨*s*⟩ *Q: Where can I stand on a river to see water falling without getting wet?* ⟨/*s*⟩ *A: waterfall* ⟨/*s*⟩", where '*Q:*' and '*A:*' are the prefix for question and answer, respectively.

Results are summarized in Table 3.3. We obtained a dev-set accuracy of 77.56% with the RoBERTa-large model. When using FreeLB finetuning, we achieved 78.81%, a 1.25% absolute gain. Compared with the results reported from fairseq repository, which obtains 78.43% accuracy on the dev-set, FreeLB still achieves better performance. Our submission to the CommonsenseQA leaderboard achieves 72.2% single-model test set accuracy, and the result of a 20-model ensemble is 73.1%, which achieves No.1 among all the submissions without making use of ConceptNet.

3.5.3 Ablation Study and Analysis

In this sub-section, we first show the importance of reusing dropout mask, then conduct a thorough ablation study on FreeLB over the GLUE benchmark to analyze the robustness and generalization strength of different approaches. We observe that it is unnecessary to perform shallow-layer updates on the adversary as YOPO for our case, and FreeLB results in improved robustness and generalization compared with PGD.

Importance of Reusing Mask. Table 3.4 (columns 2 to 4) compares the results of FreeLB with and without reusing the same dropout mask in each ascent step, as proposed in Sec. 3.4. With reusing, FreeLB can achieve a larger improvement over the naturally

[10]https://github.com/pytorch/fairseq/tree/master/examples/roberta/commonsense_qa

Table 3.4: The median and standard deviation of the scores on the dev sets of RTE, CoLA and MRPC from the GLUE benchmark, computed from 5 runs with the same hyper-parameters except for the random seeds. We use FreeLB-m to denote FreeLB with m ascent steps, and FreeLB-3* to denote the version without reusing the dropout mask.

Methods	Vanilla	FreeLB-3*	FreeLB-3	YOPO-3-2	YOPO-3-3
RTE	85.61 (1.67)	87.14 (1.29)	**88.13 (1.21)**	87.05 (1.36)	87.05 (0.20)
CoLA	67.57 (1.30)	69.31 (1.16)	**71.12 (0.90)**	70.40 (0.91)	69.91 (1.16)
MRPC	90.69 (0.54)	90.93 (0.66)	**91.42 (0.72)**	90.44 (0.62)	90.69 (0.37)

trained models. Thus, we enable mask reusing for all experiments involving RoBERTa.

Comparing the Robustness Table 3.5 provides the comparisons of the maximum increment of loss in the vicinity of each sample, defined as:

$$\Delta L_{\max}(\boldsymbol{X}, \epsilon) = \max_{\|\boldsymbol{\delta}\| \leq \epsilon} L(f_{\boldsymbol{\theta}}(\boldsymbol{X} + \boldsymbol{\delta}), y) - L(f_{\boldsymbol{\theta}}(\boldsymbol{X}), y), \tag{3.5}$$

which reflects the robustness and invariance of the model in the embedding space. In practice, we use PGD steps as in Eq. 3.2 to find the value of $\Delta L_{\max}(\boldsymbol{X}, \epsilon)$. We found that when using a step size of $5 \cdot 10^{-3}$ and $\epsilon = 0.01\|X\|_F$, the PGD iterations converge to almost the same value, starting from 100 different random initializations of $\boldsymbol{\delta}$ for the RoBERTa models, trained with or without FreeLB. This indicates that PGD reliably finds $\Delta L_{\max}$ for these models. Therefore, we compute $\Delta L_{\max}(\boldsymbol{X}, \epsilon)$ for each $\boldsymbol{X}$ via a 2000-step PGD.

Samples with small margins exist even for models with perfect accuracy, which could give a false sense of vulnerability of the model. To rule out the outlier effect and make $\Delta L_{\max}(\boldsymbol{X}, \epsilon)$ comparable across different samples, we only consider samples that all the evaluated models can correctly classify, and search for an ϵ for each sample such

Table 3.5: Median of the maximum increase in loss in the vicinity of the dev set samples for RoBERTa-Large model finetuned with different methods. Vanilla models are naturally trained RoBERTa's. M-Inc: Max Inc, M-Inc (R): Max Inc (R). Nat Loss (N-Loss) is the loss value on clean samples. Notice we require *all* clean samples here to be correctly classified by all models, which results in 227, 850 and 355 samples for RTE, CoLA and MRPC, respectively. We also give the variance in the Appendix.

Methods	RTE			CoLA			MRPC		
	M-Inc (10^{-4})	M-Inc (R) (10^{-4})	N-Loss (10^{-4})	M-Inc (10^{-4})	M-Inc (R) (10^{-4})	N-Loss (10^{-4})	M-Inc (10^{-3})	M-Inc (R) (10^{-3})	N-Loss (10^{-3})
Vanilla	5.1	5.3	4.5	6.1	5.7	5.2	10.2	10.2	1.9
PGD	4.7	4.9	6.2	128.2	130.1	436.1	5.7	5.7	5.4
FreeLB	3.0	2.6	4.1	1.4	1.3	7.2	3.6	3.6	2.7

that the reference model can correctly classify all samples within the ϵ ball.[11] However, such choice of per-sample ϵ favors the reference model by design. To make fair comparisons, Table 3.5 provides the median of $\Delta L_{\max}(\boldsymbol{X}, \epsilon)$ with per-sample ϵ from models trained by FreeLB (Max Inc) and PGD (Mac Inc (R)), respectively.

Across all three datasets and different reference models, FreeLB has the smallest median increment even when starting from a larger natural loss than vanilla models. This demonstrates that FreeLB is more robust and invariant in most cases. Such results are also consistent with the models' dev set performance (the performances for Vanilla/PGD/FreeLB models on RTE, CoLA and MRPC are 86.69/87.41/89.21, 69.91/70.84/71.40, 91.67/91.17/91.17, respectively).

Comparing with YOPO. The original implementation of YOPO [98] chooses the first convolutional layer of the ResNets as f_0 for updating the adversary in the "s-loop". As a result, each step of the "s-loop" should be using exactly the same value to update the adversary,and YOPO-m-n degenerates into FreeLB with a n-times large step size.

[11] For each sample, we start from a value slightly larger than the norm constraint during training for ϵ, and then decrease ϵ linearly until the model trained with the reference model can correctly classify after a 2000-step PGD attack. The reference model is either trained with FreeLB or PGD.

Table 3.6: The median and standard deviation of the scores on the dev sets of STS-B, SST-2, QNLI, QQP and MNLI from the GLUE benchmark, each computed from 5 runs with the same hyper-parameters except for the random seeds (except for the results with YOPO on QQP, which are from 4 runs). Also note here we use a step size of α for the adversary of YOPO-m-n, so YOPO effectively uses a step size of $n\alpha$. We use FreeLB-m to denote FreeLB with m ascent steps, and YOPO-3-n to denote YOPO with n shallow-layer ascents.

Methods	Vanilla	FreeLB-3	YOPO-3-2	YOPO-3-3
STS-B	92.20 (.2)	**92.67 (.08)**	92.60 (.17)	92.60 (0.20)
SST-2	96.56 (.3)	**96.79 (.2)**	96.44 (.2)	96.33 (.1)
QNLI	-	**94.98 (.2)**	94.96 (.1)	-
QQP	-	**92.60 (.03)**	92.55 (.05)*	92.50 (.02)*
MNLI	-	**90.61 (.1)**	90.59 (.2)	90.45 (.2)

Table 3.7: Results (median) on the dev sets of GLUE from 5 runs with the same hyperparameter but different random seeds. RoBERTa-FreeLB and ALBERT-FreeLB are RoBERTa-large and ALBERT-xxlarge-v2 models fine-tuned with FreeLB on GLUE. All other results are copied from [5].

Method	MNLI (Acc)	QNLI (Acc)	QQP (Acc)	RTE (Acc)	SST-2 (Acc)	MRPC (Acc)	CoLA (Mcc)	STS-B (Pearson)
BERT-large	86.6	92.3	91.3	70.4	93.2	88.0	60.6	90.0
XLNet-large	89.8	93.9	91.8	83.8	95.6	89.2	63.6	91.8
RoBERTa-large	90.2	94.7	92.2	86.6	96.4	90.9	68.0	92.4
RoBERTa-FreeLB	90.6	95.0	**92.6**	88.1	96.8	91.4	71.1	92.7
ALBERT-xxlarge-v2	90.8	95.3	92.2	89.2	96.9	90.9	71.4	93.0
ALBERT-FreeLB	**90.9**	**95.6**	92.5	**89.9**	**97.0**	**92.4**	**73.1**	**93.2**

To avoid that, we choose the layers up to the output of the first Transformer block as f_0 when implementing YOPO. To make the total amount of update on the adversary equal, we take the hyper-parameters for FreeLB-m and only change the step size α into α/n for YOPO-m-n. Table 3.4 shows that FreeLB performs consistently better than YOPO on all three datasets. Accidentally, we also give the results comparing with YOPO-m-n *without* changing the step size α for YOPO in Table 3.6. The gap between two approaches seem to shrink, which may be caused by using a larger total step size for the YOPO adversaries. We leave exhaustive hyperparameter search for both models as our future work.

Improving ALBERT. To further explore its ability to improve more sophisticated language models, we apply FreeLB to the fine-tuning stage of ALBERT-xxlarge-v2 [5] model on the dev set of GLUE. The implementation is based on HuggingFace's transformers library. The results are shown in Table 3.7. Our model are able to surpass ALBERT on all datasets.

Chapter 4: Improving Stability and Efficiency with Automated Initialization

Innovations in neural architectures have fostered significant breakthroughs in language modeling and computer vision. Unfortunately, novel architectures often result in challenging hyper-parameter choices and training instability if the network parameters are not properly initialized. A number of architecture-specific initialization schemes have been proposed, but these schemes are not always portable to new architectures. In this chapter, we present GradInit [54], an automated and architecture agnostic method for initializing neural networks. GradInit is based on a simple heuristic; the norm of each network layer is adjusted so that a single step of SGD or Adam with prescribed hyperparameters results in the smallest possible loss value. This adjustment is done by introducing a scalar multiplier variable in front of each parameter block, and then optimizing these variables using a simple numerical scheme. GradInit accelerates the convergence and test performance of many convolutional architectures, both with or without skip connections, and even without normalization layers. It also improves the stability of the original Transformer architecture for machine translation, enabling training it without learning rate warmup using either Adam or SGD under a wide range of learning rates and momentum coefficients.

4.1 Introduction

The initialization of network parameters has a strong impact on the training stability and performance of deep neural networks. Initializations that prevent gradient explosion/vanishing in back propagation played a key role in early successes with feed-forward networks [16, 133]. Even with cleverly designed initialization rules, complex models with many layers or multiple branches can still suffer from instability. For example, the original Transformer model [23] does not converge without learning rate warmup using the default initialization [15, 134, 135]; RoBERTa [4] and GPT-3 [36] have to tune the β_2 parameter of Adam for stability when the batch size is large. Recent innovations have shown that architecture-specific initializations, which are carefully derived to maintain stability, can promote convergence without needing normalization layers [14, 135–138]. Unfortunately, the reliance on analytically derived initializations makes it difficult to realize the benefits of these methods when performing architecture search, training networks with branched or heterogeneous components, or proposing altogether new architectures.

4.1.1 Related Works

Controlling the norms of network parameters at initialization has proven to be an effective approach for speeding up and stabilizing training. Glorot and Bengio [133] studied how the variance of features evolves with depth in feed-forward linear neural networks by assuming both activations and weight tensors are independent and identical random variables. They developed a technique in which the variance of each filter scales with its fan-in (the number of input neurons). This style of analysis was later generalized to

the case of ReLU networks [16]. These two analyses are most effective for feed-forward networks without skip connections or normalization layers. Based on the orthogonal initialization scheme [139], Mishkin and Matas [140] proposed an iterative procedure to rescale the orthogonally initialized weights of each layer in feedforward networks so that the activations of that layer have unit variance. However, this method fails to prevent the blowup of activations with depth for ResNets [6]. Recently, Gurbuzbalaban and Hu [141] proposed initialization schemes such that the network can provably preserve any given moment of order $s \in (0, 2]$ for the output of each layer. The motivation is that the stochastic gradient updates can result in heavy-tailedness in the distribution of the network weights with a potentially infinite variance, but finite s-order moment [142]. Again, these initialization schemes can only be applied for feed-forward neural networks.

For more complex architectures, normalization layers [20, 143] and skip connections [19] stabilized training dynamics and improved the state-of-the-art. Similarly, learning rate warmup is a common trick for training large Transformers [23]. These methods make training tractable for some models, but do not eliminate the high initial gradient variance that destabilizes training when the network is deep [14, 136, 137] or when the normalization layers are not carefully positioned [134].

Several authors have proposed better initializations for networks with skip connections. This is often achieved by replacing the normalization layers with simpler scaling or bias operations, and scaling the weight matrices in each layer so that the variance of activations does not increase with depth [14, 136–138]. Similar analysis has been applied to self attention in Transformers [135]. Without removing the normalization layers, it is possbile to stabilize the initial parameter updates by introducing carefully initialized

learnable scale factors to the skip connections [15] or the residual branches [144]. However, such techniques are often restricted to one specific architecture such as ResNets.

Recently, Dauphin and Schoenholz [6] proposed a task-agnostic and automatic initialization method, MetaInit, for any neural network achitecture. MetaInit optimized the norms of weight tensors to minimize the "gradient quotient", which measures the effect of curvature near the initial parameters, on minibatches of random Gaussian samples. However, as training data is usually accessible for most tasks of interest, it is simpler and potentially more efficient to use the training data for initialization. MetaInit also involves the gradient of a Hessian-vector product that requires computing a "gradient of the gradient" multiple times in tandem, which is very computationally intensive. Our proposed method distinguishes itself from MetaInit in the following ways: (i) Our method is more computationally efficient. MetaInit involves computing third-order derivatives, results in long computing times and high memory usage. The memory overhead of MetaInit is more of an issue for networks with normalization layers. For the relatively small-scale CIFAR-10 problem with batch size 64, MetaInit requires three GPUs (RTX 2080Ti), while the proposed GradInit needs just one. (ii) Our method takes the stochasticity of minibatches into consideration. MetaInit uses the local curvature evaluated on a single minibatch, which fails to capture the variance of the loss/gradient between two different stochastic minibatches. (iii) Our method considers the training dynamics of different optimization algorithms including the learning rate and the direction of the gradient step, and effectively handles different optimizers including SGD and Adam.

4.1.2 Our contributions

In this work, we propose a simple method for learning the initialization of a network with any architecture. Typically, initialization schemes draw parameters independently from a zero-mean distribution, with the variance of each distribution set to pre-determined values depending on the dimensions of the layers [16, 133]. Rather than deriving a closed-form expression for the these distribution parameters, our method re-scales each random weight tensor (e.g. convolution kernels) directly by a learned scalar coefficient. This small set of coefficients is optimized to make the first step of a stochastic optimizer (e.g. SGD or Adam) as effective as possible at minimizing the training loss, while preventing the initial gradient norm from exploding. In addition, this process is designed to take into account the direction, step size, and stochasticity of the optimizer. Finally, after the variance has been learned for each parameter tensor, the random network parameters are re-scaled and optimization proceeds as normal. We empirically find that our methods can make the initialization fall into a smooth loss region, reduce the inter-sample gradient variance, and accelerates training.

Our proposed method, GradInit, is architecture agnostic, and works with both Adam and SGD optimizers. In the vision domain, we show it accelerates the convergence and test performance of a variety of deep architectures, from the vanilla feed-forward VGG net to ResNet, with or without Batch Normalization. It is efficient and scalable, finding good initializations using less than 1% of the total training time in our experiments, and it improves the initialization of ResNet-50 on ImageNet to obtain better final test accuracy. In the language domain, GradInit enables training the original Transformer model [23]

using either Adam or SGD without learning rate warmup for machine translation, which is commonly acknowledged to be difficult [134, 145]. As an extreme example of the capabilities of GradInit, we use it to initialize and train a 1202-layer ResNet that achieves significantly higher test accuracy than ResNet-110, which other initialization methods have failed to achieve.

Finally, by visualizing the initial norms and gradient variances of the weights before and after GradInit is applied, we show that GradInit is a useful tool for identifying potential causes for instability at initialization, such as those imposed by normalization layers, and we summarize interesting scale patterns learned by GradInit that can be helpful for designing better initialization rules.

4.2 GradInit: an Automated Initialization

We aim to develop an initialization scheme applicable to arbitrary network architectures. Since previous works [6, 14, 16, 133, 136, 138] have shown that the initial weight norms effectively control the initial gradient norm on average, our method rescales the randomly initialized weight matrices using learnable scale factors.[1]

Using a small number of gradient descent steps on these scale factors, the proposed GradInit method chooses the initialization scalars so that the loss after the first gradient step taken by a stochastic optimizer (SGD or Adam) is as low as possible. The process of learning initialization coefficients accounts for the chosen learning rate, optimizer, and other parameters. To prevent gradient explosion, our method enforces a constraint that

[1]For convenience, we refer to weight vectors/matrices/tensors as "weight matrices", which includes the scale vectors of the normalization layers, the bias vectors, the weight matrices of the fully connected layers, and the convolution kernels.

the gradient norm is no larger than a constant γ.

Note that for scale-invariant weights, e.g., convolution kernels before BN layers, rescaling still changes their learning dynamics by changing their effective learning rate [146, 147]. Empirically, GradInit goes beyond simply preventing exploding or vanishing gradients; it also reduces the gradient variance, making the initialization fall into a smooth loss region with small gradient variance so that training is fast, see discussion about Figure 4.1 and comparisons in Figure 4.2.

4.2.1 Efficient Learning-based Initialization via Constrained Optimization

We begin by filling all the weight matrices $\{\boldsymbol{W}_1, \ldots, \boldsymbol{W}_M\}$ of the network with values drawn from independent zero-mean Gaussian distributions, except for the scales and biases of the normalization layers (if any), which are initialized to 1 and 0 respectively. During the initialization process, we keep $\{\boldsymbol{W}_1, \ldots, \boldsymbol{W}_M\}$ constant, but we multiply each $\boldsymbol{W}_i$ with a learnable non-negative scale factor α_i (initialized to 1). After initialization, we rescale the weights by the learned scale factors, and start training without the learnable scale factors just as normal. We use $\boldsymbol{m} = \{\alpha_1, \ldots, \alpha_M\}$ to denote the set of scale factors, and $\boldsymbol{\theta}_{\boldsymbol{m}} = \{\alpha_1 \boldsymbol{W}_1, \ldots, \alpha_M \boldsymbol{W}_M\}$ is the set of rescaled weight matrices.

Let $L(S; \boldsymbol{\theta}) = \frac{1}{|S|} \sum_{x \in S} \ell(x; \boldsymbol{\theta})$ be the average loss of the model parameterized by $\boldsymbol{\theta}$ on a minibatch of samples S, where $|S|$ is the number of samples in the minibatch. We use $\boldsymbol{g}_{S,\boldsymbol{\theta}} = \nabla_{\boldsymbol{\theta}} L(S; \boldsymbol{\theta})$ as a shorthand for the gradient of $\boldsymbol{\theta}$. During standard training, this gradient is preprocessed/preconditioned by the optimization algorithm $\mathcal{A}$, and then used

to update the network parameters. GradInit solves the following constrained optimization problem:

$$\begin{aligned} \underset{\boldsymbol{m}}{\text{minimize}} \quad & L(\tilde{S}; \boldsymbol{\theta_m} - \eta \mathcal{A}\left[\boldsymbol{g}_{S,\boldsymbol{\theta_m}}\right]), \\ \text{subject to} \quad & \|\boldsymbol{g}_{S,\boldsymbol{\theta_m}}\|_{p_{\mathcal{A}}} \leq \gamma, \end{aligned} \tag{4.1}$$

where S and $\tilde{S}$ are two different minibatches, η is a prescribed learning rate for the optimization algorithm $\mathcal{A}$, $p_{\mathcal{A}}$ is the ℓ_p-norm associated with $\mathcal{A}$, and γ is the upper bound for the norm. For the first gradient step, Adam uses $\mathcal{A}[\boldsymbol{g}_{S,\boldsymbol{\theta_m}}] = \text{sign}(\boldsymbol{g}_{S,\boldsymbol{\theta_m}})$ [148], while SGD uses $\mathcal{A}[\boldsymbol{g}(S;\boldsymbol{\theta_m})] = \gamma \boldsymbol{g}(S;\boldsymbol{\theta_m})/\|\boldsymbol{g}(S;\boldsymbol{\theta_m})\|_2$. We show how to choose γ and $p_{\mathcal{A}}$ without tuning in Section 4.2.4. We discuss the formulation of this problem and how to solve it below.

4.2.2 Solving the Constrained Problem

The problem equation 4.1 is solved using a stochastic gradient descent method in which we sample new mini-batches on each iteration. Since the proposed method uses gradient updates to compute the initialization, we dub it *GradInit*. We propose a simple solver to optimize objective equation 4.1 in Algorithm 4. A key feature of our method is that is makes a simple approximation: after $\boldsymbol{g}_{S,\boldsymbol{\theta_m}}$ is computed on the forward pass of an iteration, we treat $\mathcal{A}[\boldsymbol{g}_{S,\boldsymbol{\theta_m}}]$ as a constant and do not back-propagate through $\mathcal{A}[\boldsymbol{g}_{S,\boldsymbol{\theta_m}}]$ on the backward pass. We make this choice to keep computing costs low, and because it is not possible to back-propagate through the non-differentiable sign function for Adam.

To enforce the constraint in equation 4.1, we test whether the constraint is sat-

Algorithm 4 *GradInit* for learning the initialization of neural networks.

Input: Target optimization algorithm $\mathcal{A}$ and learning rate η for model training, initial model parameters $\boldsymbol{\theta}_0$, learning rate τ of the GradInit scales $\boldsymbol{m}$, total iterations T, upper bound of the gradient γ, lower bound for the initialization scalars $\underline{\alpha} = 0.01$.

Output: Learned scales $\boldsymbol{m}_{T+1}$.

$\boldsymbol{m}_1 \leftarrow \mathbf{1}$

for $t = 1$ ***to*** T **do**

  Sample S_t from training set;

  $L_t \leftarrow \frac{1}{|S_t|} \sum_{x_k \in S_t} \ell(x_k; \boldsymbol{\theta}_{\boldsymbol{m}_t})$, $\boldsymbol{g}_t \leftarrow \nabla_{\boldsymbol{\theta}} L_t$;

  if $\|\boldsymbol{g}_t\|_{p_{\mathcal{A}}} > \gamma$ **then**

    $\boldsymbol{m}_{t+1} \leftarrow \boldsymbol{m}_t - \tau \nabla_{\boldsymbol{m}_t} \|\boldsymbol{g}_t\|_{p_{\mathcal{A}}}$;

  else

    Sample $\tilde{S}_t$ from training set;

    $\tilde{L}_{t+1} \leftarrow \frac{1}{|\tilde{S}_t|} \sum_{x_k \in \tilde{S}_t} \ell(x_k; \boldsymbol{\theta}_{\boldsymbol{m}_t} - \eta \mathcal{A}[\boldsymbol{g}_t])$;

    $\boldsymbol{m}_{t+1} \leftarrow \boldsymbol{m}_t - \tau \nabla_{\boldsymbol{m}_t} \tilde{L}_{t+1}$;

  end

  Clamp $\boldsymbol{m}_{t+1}$ using $\underline{\alpha}$;

end

isfied after computing $\boldsymbol{g}(S; \boldsymbol{\theta}_{\boldsymbol{m}})$. If not, we take a gradient descent step to minimize $\|\boldsymbol{g}(S; \boldsymbol{\theta}_{\boldsymbol{m}})\|_{p_{\mathcal{A}}}$, which involves computing second order derivatives. If the constraint is satisfied, then we instead compute a gradient descent step for the loss. In addition, we set a lower bound $\underline{\alpha} = 0.01$ for all α_i. We find that this prevents scalars from landing on small values during minimization and keeps the GradInit optimizer stable. In our experiments, we find the only layer that ever hit this lower bound is the final FC layer on some networks (see the figures in Section 4.3.1). We find this procedure converges reliably within 2000 iterations for ImageNet, and fewer than 400 iterations for CIFAR-10, taking less than 1% of the total training time on both problems. We also find it works well to set the step size τ to values within the range between 10^{-3} and 10^{-1}. During initialization, the gradient norm constraint is satisfied for the majority of iterations. The choice of $\gamma, p_{\mathcal{A}}$ will be discussed in Section 4.2.4.

Table 4.1: Accuracies on CIFAR-10 using different overlapping ratios of $\tilde{S}$ and S for GradInit.

Model	$\frac{\lvert\tilde{S}\cap S\rvert}{\lvert S\rvert}$	Acc_1	Acc_{best}
VGG-19	0	21.9 ± 4.4	94.5 ± 0.1
w/o BN	0.5	$\mathbf{29.3 \pm 0.6}$	$\mathbf{94.7 \pm 0.02}$
(20.03 M)	1	28.7 ± 1.0	94.5 ± 0.1

Table 4.2: Using GradInit without the gradient norm constraint with different overlapping ratios r to initialize and train a VGG-19 (w/ BN). For both $r = 0.5$ and $r = 1$, we tried τ from the range of 1×10^{-4} to 2×10^{-2}. The first two rows show the results with the best final test accuracy Acc_{best} among different τ's, while the last row shows using a larger τ for $r = 1$.

Model (#Params)	$r = \lvert\tilde{S} \cap S\rvert/\lvert S\rvert$	τ	$\lVert\boldsymbol{g}\rVert_2$	Acc_0	Acc_{best}
VGG-19	0.5	4×10^{-3}	8.63 ± 0.20	$\mathbf{38.37 \pm 1.45}$	$\mathbf{94.78 \pm 0.08}$
w/ BN	1	1×10^{-4}	11.56 ± 0.05	13.81 ± 2.47	94.45 ± 0.07
(20.04 M)	1	4×10^{-3}	190.62 ± 7.65	10.30 ± 0.15	93.70 ± 0.17

4.2.3 Stochasticity of mini-batching

The objective in equation 4.1 uses two different mini-batches; S is used to compute the gradient, and $\tilde{S}$ is used to compute the loss. Ideally, S and $\tilde{S}$ should be independently sampled from the training set to capture the randomness of the stochastic optimizer. However, when the network has large initial gradient variance, the gradients on S and $\tilde{S}$ usually differ a lot, and for $\tilde{S}$, the gradient update step $\boldsymbol{\theta_m} - \eta\mathcal{A}\left[\boldsymbol{g}_{S,\boldsymbol{\theta_m}}\right]$ becomes more similar to adding random perturbations to the parameters. We find our objective less effective at accelerating convergence in this case, as shown by the first-epoch accuracy (Acc_1) in Table 4.1. On the other hand, the randomness is not captured if $S = \tilde{S}$. For this case, we consider removing the gradient norm constraint of GradInit (by setting γ to ∞) while using overlapping ratios $r = 1$ and $r = 0.5$ respectively for a VGG-19 (w/ BN) model, to highlight the different degrees of reliance of the two approaches on the gradient con-

straint. As shown in Table 4.2, when $r = 1$, we have to use the smallest τ, which results in minimum change to the scale factors, to obtain results that are not significantly worse than the baseline (Kaiming initialization listed in Table 4.4). It is easy for these large over-parameterized models to overfit a single minibatch with the scale factors. When $r = 1$, GradInit learns a greedy strategy, which increases the gradient as much as possible to enable a steeper descent that sometimes can reduce the loss on the same minibatch by more than 50% in just one iteration. The greedy strategy tends to blow up of the gradient norm at initialization, which hinders convergence and results in a higher dependence on the gradient norm constraint γ. However, when we use $\tau = 0.5$, GradInit is able to improve the baseline without any gradient norm constraint.

Without excessive tuning, we find that we get more reliable behavior for different architectures when $\tilde{S}$ is a mixture of 50% samples from S and 50% re-sampled training data, and use this setting by default unless otherwise stated.

4.2.4 Setting and Enforcing the Constraint

The constraint in equation 4.1 is included to prevent the network from minimizing the loss in a trivial way by blowing up the initial gradient. In other words, we want the optimizer to achieve small loss by choosing an effective search direction rather than by taking an extremely large step in a sub-optimal direction.

4.2.5 Setting the norm constraint through first-order analysis

We show that $p_{\mathcal{A}}$ and γ can be set easily with a rule of thumb and without a parameter search. From the first-order approximation, we expect the first gradient step to result in a change in the loss on S as following:

$$L(S;\theta_{\boldsymbol{m}}-\eta\mathcal{A}[\boldsymbol{g}_{S,\theta_{\boldsymbol{m}}}])-L(S;\theta_{\boldsymbol{m}}) \approx -\eta\mathcal{A}[\boldsymbol{g}_{S,\theta_{\boldsymbol{m}}}]^T\boldsymbol{g}_{S,\theta_{\boldsymbol{m}}} = \begin{cases} -\eta\|\boldsymbol{g}_{S,\theta_{\boldsymbol{m}}}\|_2^2, & \text{if } \mathcal{A} \text{ is SGD,} \\ -\eta\|\boldsymbol{g}_{S,\theta_{\boldsymbol{m}}}\|_1, & \text{if } \mathcal{A} \text{ is Adam.} \end{cases} \tag{4.2}$$

To effectively bound the approximated change in Eq. 4.2, we choose $\ell_{p_{\mathcal{A}}}$ to be the ℓ_2 and ℓ_1 norm for SGD and Adam respectively, so when the constraint is satisifed, the maximum change in the loss, according to our local approximation, is $\eta\gamma^2$ for SGD and $\eta\gamma$ for Adam. We recommend setting γ such that $\eta\gamma^2 = 0.1$ for SGD and $\eta\gamma = 0.1$ for Adam. According to the linear approximations, this limits the gradient magnitude so that the first step of SGD can decrease the loss by at most 0.1. This simple rule was used across all vision and language experiments.

4.2.6 Why a constraint and not a penalty?

Instead of formulating GradInit as a constrained optimization, one can alternatively formulate it as minimizing the objective with a gradient penalty: $\underset{\boldsymbol{m}}{\text{minimize}} \quad L(\tilde{S};\boldsymbol{\theta}_{\boldsymbol{m}} - \eta\mathcal{A}\left[\boldsymbol{g}_{S,\boldsymbol{\theta}_{\boldsymbol{m}}}\right]) + \lambda\|\boldsymbol{g}_{S;\boldsymbol{\theta}_{\boldsymbol{m}}}\|_{p_{\mathcal{A}}}$, where $\lambda > 0$ is the penalty strength. The penalized objective has two drawbacks compared to the constrained one in Eq. 4.1. First, every gradient descent step on the penalized objective involves second-order gradients due to the gradient

Table 4.3: Time cost and accuracy (average of 4 runs) for running one epoch of regularization/constrained form of GradInit.

Model	VGG-19 w/o BN	VGG-19 w/ BN	ResNet-110 w/o BN	ResNet-110 w/ BN
Time (s)	82 vs. 56	100 vs. 62	169 vs. 103	269 vs. 195
$\lambda = 10^{-4}$	**32.3**, 94.6	10.6, 93.1	33.7, 93.9	32.4, 95.2
$\lambda = 10^{-2}$	30.4, 94.5	10.4, 93.0	**36.7**, 94.1	32.6, 95.3
$\lambda = 1$	18.2, 74.7	38.5, 95.1	30.7, 94.2	36.5, 95.3
$\gamma = 1$	29.3, **94.7**	**47.8**, **95.1**	36.2, **94.6**	**38.2**, **95.4**

regularization, while the constrained form does not need second-order gradients when the constraint is satisfied. Second, it is difficult to choose a good λ that works well for all architectures. By contrast, we set γ by analyzing the first-order approximation mentioned above, and find the same γ works well for different architectures. The results supporting these two points are given in Table 4.3.

4.3 Experiments

We evaluate GradInit on benchmark datasets for image classification and machine translation tasks. For image classification, five different architectures are evaluated for CIFAR10 [149], and ResNet-50 is evaluated for ImageNet [150]. For machine translation, we use GradInit to find good initializations for a Post-LN Transformer without any change to its original architecture on IWSLT-14 De-En [151]. We observe that the method can remove the necessity of any form of learning rate warmup for both Adam and SGD.

We conduct our experiments in PyTorch. We use the fairseq library for machine translation [152]. All the experiments on CIFAR-10 and IWSLT-14 DE-EN can run with one single NVIDIA RTX 2080 Ti GPU with 11GB of RAM.

GradInit first initializes the weights using Kaiming initialization [16] for all the Conv and FC layers for image classification. For machine translation, we use the default Xavier initialization [133]. We optimize the scale factors $\{\alpha_i\}$ with Adam [73] using the default momentum parameters.

4.3.1 Image Datasets with Various Architectures

The introduction of Batch Normalization (BN) [20] and skip connections makes it relatively easy to train common CNNs for image classification to achieve high accuracy. Despite this, we show that when the network is very deep, the network is unstable even when both BN and skip connections are used, and GradInit can significantly improve the stability. The results on CIFAR-10 are given in Table 4.4 and results on ImageNet are given in Table 4.8.

4.3.1.1 Settings

Settings. On CIFAR-10, we focus on the feedforward VGG net and the prevalent and powerful ResNet, with and without BN layers. For networks without BN, we use learnable biases in all layers. For ResNet, we additionally evaluate a deep 1202-layer version. We give results for other architectures (Wide ResNet, DenseNet) in Table 4.5. We compare with four different methods/settings: 1) Kaiming Initialization [16]; 2) First train the network for one epoch with a constant learning rate equal to the starting learning rate, labelled as "+1 epoch (Const. LR)" in Table 4.4; 3) First train the network for one epoch with a linear warmup learning rate, labbeled as "+1 epoch (Warmup)" in Table 4.4;

4) MetaInit [6].

On ImageNet, we use the ResNet-50 model [19]. We compare with Kaiming Initialization, FixUp initialization [136] and MetaInit. For the ResNet-50 without BN, we follow the architecture of FixUp for fair comparisons, but we still use the original Kaiming initialization as the starting point of GradInit.

Architectures. The base architectures include a popular variant of VGG-19 [153] with BN for CIFAR-10, which includes all the sixteen convolutional layers but only one fully connected layer; a ResNet-110 [19] with base width 16 and two Conv layers in each residual block, as well as its 1202-layer verison with the same depth configurations as FixUp; a 28-layer Wide ResNet [154] with Widen Factor 10 (WRN-28-10) ; and a DenseNet-100 [83]. To isolate the effect of BN, we also consider removing the BN layers from these three networks and adding learnable bias parameters in their place. To compare with a strong initialization scheme that is tailor-made for an architecture family, we consider a 110-layer FixUpResNet [136]. FixUpResNet removes the BN from ResNet, replacing it with bias parameters and a learnable scale parameter after the second convolutional layer of each block. FixUp initializes the weights of the second convolutional layer in each residual block, and of the final fully connected layer, to zero. It also scales the first convolutional layer in each residual block by $1/\sqrt{M}$. This causes the gradient to be zero in the first step for all layers except for the final FC layer. When testing GradInit on this architecture, we adopt the non-zero Kaiming initialization to all convolutional and FC layers. The results are given in Table 4.5.

Hyperparameters. We set $\mathcal{A}$ to SGD and $\eta = 0.1$ (the same as the base learning rate) for GradInit in all image classification experiments. On CIFAR-10, we train net-

works with a batch size of 128. We find MetaInit often takes 2 to 3 times as much memory as GradInit. We run GradInit or MetaInit for one epoch on the data, which takes less than 1% of the total training time. For GradInit, according to our analysis in Section 4.2.4, we fix the gradient norm constraint $\gamma = 1$ in all these experiments. Therefore, as in MetaInit, the only hyperparameter that needs to be tuned is the learning rate τ of the scale factors. We do a grid search on τ in the range $[10^{-3}, 10^{-1}]$, and report the results with the best average final test accuracy on 4 runs. After GradInit initialization, we use a learning rate of 0.1 and the cosine annealing learning rate schedule without restart [155] to train the model for 200 epochs, where the learning rate decays after each iteration and decays to 0 in the last iteration. Due to their high initial gradient variance (see Figure 4.6), we have applied gradient clipping (maximum norm is 1) to all non-BN networks so that they converge without GradInit under the same schedule. We use batch size 128 to train all models, except for DenseNet-100, where the recommended batch size is 64.[2] We use random cropping, random flipping and cutout [156] for data augmentation. We do not use dropout in any of our experiments. We set weight decay to 10^{-4} in all cases. As in Algorithm 4, each scale factor is initialized to 1 and we set lower bounds $\underline{\alpha} = 0.01$. For each architecture, we try τ from $\{10^{-3}, 2\times10^{-3}, 5\times10^{-3}, 10^{-2}, 2\times10^{-2}, 5\times10^{-2}, 10^{-1}\}$, and report the results of 4 runs with the best τ. We find the best τ for VGG-19 (w/o BN), VGG-19 (w/ BN), ResNet-110 (w/o BN), ResNet-110 (w/ BN), FixUpResNet, DenseNet-100 (w/o BN), DenseNet-100 (w/ BN) are $10^{-2}, 10^{-1}, 5\times10^{-2}, 5\times10^{-3}, 2\times10^{-2}, 5\times10^{-3}, 10^{-2}$ respectively.

On ImageNet, we train the ResNet-50 model for 90 epochs with a total batch size

[2] https://github.com/gpleiss/efficient_densenet_pytorch

of 256 on 4 GPUs. Due to the difference in the library for training and the number of GPUs used, which affects the BN statistics, our baseline top-1 accuracy of ResNet-50 (w/ BN) on ImageNet is 0.79% lower than [157]. We use SGD with a starting learning rate of 0.1 and decay the learning rate by 10 after the 30th and 60th epoch. We use random cropping and flipping as data augmentation. For experiments without BN, we additionally apply MixUp [158] with $\alpha = 0.7$ for all models, for fair comparisons FixUp. We train the models for 90 epochs and decay the learning rate by a factor of 10 every 30 epochs. To fit into the memory, we use a batch size of 128 for GradInit. We simply run GradInit for 2000 iterations, which is less than half an epoch. Considering ImageNet and CIFAR-10 has 1000 and 10 classes respectively, the cross entropy loss of a random guess on ImageNet is 3 times as large as the loss on CIFAR-10, so a proper initial gradient norm for ImageNet may be 3 times as large as that for CIFAR-10. Therefore, we set $\gamma = 3$ for ImageNet. Since $\tau = 10^{-2}$ worked the best for ResNet-110 (w/ BN) on CIFAR-10, we tried $\tau \in \{1 \times 10^{-3}, 3 \times 10^{-3}, 5 \times 10^{-3}, 10^{-2}\}$ on ImageNet, and chose $\tau = 3 \times 10^{-3}$, which maximizes the test accuracy of first epoch.

4.3.1.2 Results and Analysis

GradInit further stabilizes feedforward nets with BN. BN does stabilize VGG-19 and allows training without gradient clipping, but with an average first-epoch test accuracy of only 12.57 and an average final test accuracy lower than the version without BN (see Table 4.4), it does not seem to eliminate the instability of Kaiming initialization. As shown in Figure 4.4, its initial gradient variance is still relatively high compared with

Table 4.4: First epoch (Acc_1) and best test accuracy over all epochs (Acc_{best}) for models on CIFAR-10. We report the mean and standard error of the test accuracies in 4 experiments with different random seeds. Best results in each group are in bold.

Model (# Params)		VGG-19 w/o BN (20.03M)	VGG-19 w/ BN (20.04M)	ResNet-110 w/o BN (1.72M)	ResNet-110 w/ BN (1.73M)	ResNet-1202 w/ BN (19.42M)
Kaiming	Acc_1	29.1 ± 1.5	12.6 ± 0.6	16.1 ± 2.1	23.2 ± 0.9	12.9 ± 2.8
	Acc_{best}	94.5 ± 0.1	94.4 ± 0.1	94.2 ± 0.1	95.0 ± 0.2	94.4 ± 0.6
+1 epoch (Const. LR)	Acc_1	37.2 ± 1.1	19.6 ± 4.0	21.0 ± 3.8	32.5 ± 3.8	12.6 ± 2.8
	Acc_{best}	94.4 ± 0.1	94.5 ± 0.1	93.9 ± 0.4	94.7 ± 0.3	94.0 ± 0.4
+1 epoch (Warmup)	Acc_1	37.4 ± 1.2	53.5 ± 2.9	19.8 ± 0.5	48.7 ± 1.1	28.1 ± 1.3
	Acc_{best}	94.4 ± 0.1	94.7 ± 0.1	94.1 ± 0.1	95.1 ± 0.1	95.4 ± 0.2
MetaInit	Acc_1	30.5 ± 0.9	35.1 ± 0.6	14.6 ± 2.2	29.0 ± 1.5	11.7 ± 1.6
	Acc_{best}	94.6 ± 0.1	94.6 ± 0.1	94.2 ± 0.1	94.8 ± 0.1	95.0 ± 0.5
GradInit	Acc_1	29.3 ± 0.6	47.8 ± 1.8	36.2 ± 0.8	38.2 ± 0.9	29.0 ± 1.1
	Acc_{best}	**94.7** ± 0.1	**95.1** ± 0.1	**94.6** ± 0.1	**95.4** ± 0.1	**96.2** ± 0.1

Table 4.5: First epoch (Acc_1) and best test accuracy over all epochs (Acc_{best}) for models on CIFAR-10. We report the mean and standard error of the test accuracies in 4 experiments with different random seeds. Best results in each group are in bold. For WRN, we have additionally used MixUp during training to enhance the results, but we do not consider mixup for GradInit to test its transferability to different training regularizations. Its result with MetaInit comes from the MetaInit paper.

Model (# Params)		WRN-28-10 w/ BN (36.49M)	FixUpResNet N/A (1.72M)	DenseNet-100 w/o BN (0.75M)	DenseNet-100 w/ BN (0.77M)
Kaiming	Acc_1	43.1 ± 2.7	38.2 ± 0.8	35.5 ± 0.6	51.2 ± 1.5
	Acc_{best}	97.2 ± 0.1	**95.4** ± 0.1	94.0 ± 0.1	95.5 ± 0.1
MetaInit	Acc_1	-	21.5 ± 0.6	35.1 ± 0.2	46.7 ± 4.0
	Acc_{best}	97.1	95.0 ± 0.1	94.4 ± 0.1	95.5 ± 0.1
GradInit	Acc_1	46.3 ± 0.4	35.0 ± 0.7	37.2 ± 1.1	58.2 ± 0.9
	Acc_{best}	**97.3** ± 0.1	**95.4** ± 0.1	**94.9** ± 0.1	95.5 ± 0.1

GradInit. BN could magnify the gradient variance when the variance of its input features (in the forward pass) is smaller than 1 (see Section 4.4.1). GradInit reduces the gradient variance by 4 orders of magnitude compared to Kaiming initialization , resulting in significantly higher test accuracy after the first epoch (47.79% vs. 12.57%), which also has an impact on the final test accuracy (95.13% vs. 94.41%). The reduction in gradient variance is achieved mainly by scaling down the weights of the final FC layer and the last 2 BN layers, so that the variance of the activations is reduced in the forward pass. This learned behavior is consistent with the strategy of FixUp, where the final FC layer is initialized to 0. Another source of gradient variance reduction is achieved by *increasing* the weight norms of the remaining Conv and BN layers, so that the variance of the inputs to the BN layers is increased and the gradient magnifying effect of BN is alleviated in the backward pass. This reduced the ratio $\sigma(\boldsymbol{g}_1)/\sigma(\boldsymbol{g}_{16})$ from 204.9 to 164.8 for the Conv layers in Figure 4.4. By contrast, FixUp only reduces the weight norms, which may not always be the best solution for networks with normalization layers.

Deep residual networks still need better initializations. We also gain significant improvements from GradInit for ResNet-110 and ResNet-1202. In ResNets, the skip connections cause the variance of activations to accumulate as the ResNet goes deeper, even for the version with BN [14]. This issue is more significant when the ResNet scales to 1202 layers, from which we can see that with Kaiming initialization, the first-epoch accuracy of ResNet-1202 is quite low, and the final test accuracy is even worse than the shallower ResNet-110, matching the observations of He et al. [19]. Warmup is even more effective than MetaInit at accelerating the convergence and improving the final test accuracy of ResNet-1202, but GradInit still outperforms its final test accuracy by 0.8%,

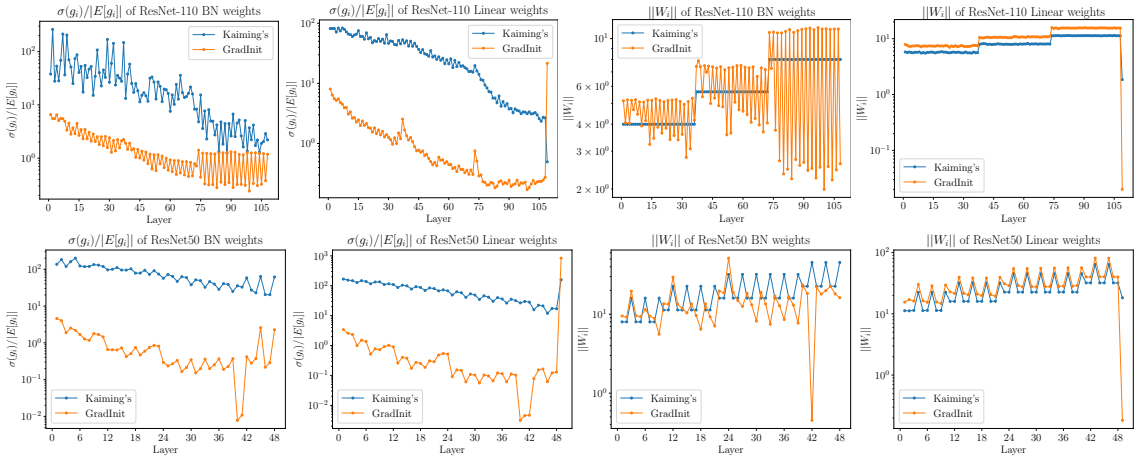

Figure 4.1: Top row: results of ResNet-110 on CIFAR-10. Bottom row: results of ResNet-50 on ImageNet. Left two columns: compare the relative cross-batch gradient variance on the training set for the BN and Conv/FC layers before and after GradInit. Right two columns: weight norms before and after GradInit. Ratio between points in the same layer reflects the scale factor. Note each of the residual blocks has 2 and 3 Conv and BN layers for the ResNet-110 and ResNet-50, respectively. The initial relative gradient variance are reduced for all layers except the final linear layer in both settings. The strategies are similar on two different datasets. Within each residual block, the last BN layer has the smallest scaling factors, and the scales of all Conv layers are surprisingly increased. Best viewed in color.

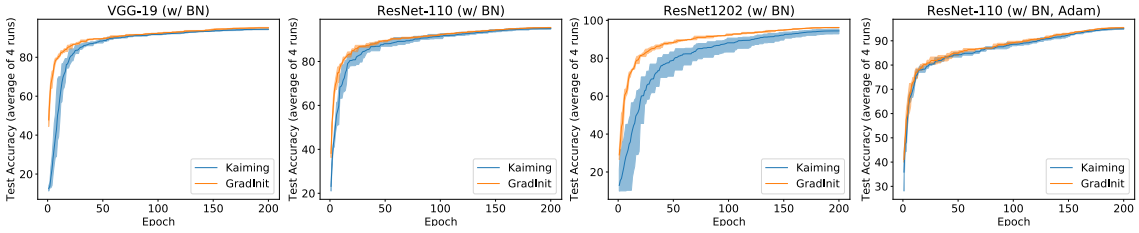

Figure 4.2: Comparing the convergence of Kaiming Initialization and GradInit on CIFAR-10, for models trained with SGD (left three) and Adam (right).

and the resulting ResNet-1202 finally achieved higher accuracy than ResNet-110.

The learned layer-wise rescaling patterns of GradInit are even more interesting for ResNets with BN. For ResNets with BN, recall that we have two Conv layers and two BN layers in each residual block. As shown in Figure 4.1, GradInit learns to *increase* the weight norms of all the linear layers except for the final FC layer, instead of decreasing as for the case without BN (see Figure 4.6). A more unique pattern is the collaborative behavior of the BN weights, where the second BN in each residual block is usually scaled down while the first BN is always scaled up. In deeper layers, the joint effect of these two BN weights is to downscale the activations and reduce their variance in the forward pass, with a more significant reducing effect as the layers get deeper. Intuitively, the marginal utility of adding a new layer decreases with depth. Therefore, for deeper layers, GradInit learns to further downscale the residual branch, and prevents the variance from increasing too much in the forward pass. Inside each residual block, increasing the scale factors of the first BN helps to reduce the magnification effect of the second BN on the gradient; forcing the input activations to the second convolution to have variance larger than 1 ensures its variance after the following convolution layer does not go below 1, avoiding the magnification effect that the second BN has on the gradient variance. See Section 4.4.1 for more discussions about the magnifying effect.

Table 4.6: Comparing the results of GradInit with fixed BN scale parameters (Fix BN) and only rescale the BN parameters (Only BN).

	Kaiming		GradInit		GradInit (Fix BN)		GradInit (Only BN)	
Model	Acc_0	Acc_{best}	Acc_0	Acc_{best}	Acc_0	Acc_{best}	Acc_0	Acc_{best}
VGG-19 (w/ BN)	12.6 ± 0.6	94.4 ± 0.1	47.8 ± 1.8	95.1 ± 0.1	13.1 ± 0.9	94.6 ± 0.1	14.4 ± 2.1	94.4 ± 0.1
ResNet-110 (w/ BN)	23.2 ± 0.9	95.0 ± 0.2	38.2 ± 0.9	95.4 ± 0.1	24.7 ± 3.1	94.7 ± 0.3	25.4 ± 3.1	94.6 ± 0.3

Generalizing to Adam. Models in previous experiments are trained with SGD. We

Table 4.7: Comparing the results with multiplying each weight matrix with a learnable scaler (Learning Scalars) on CIFAR10. The VGG-19 model is not able to converge unless we reduce the initial learning rate to 0.01, which obtained worse final accuracy. The ResNet-110 model's Acc_0 was 10% for 2 of the 4 runs.

Model	Learning Scalars		GradInit	
	Acc_0	Acc_{best}	Acc_0	Acc_{best}
VGG-19 (w/ BN, lr=0.1)	10.0 ± 0.0	10.0 ± 0.0	47.8 ± 1.8	95.1 ± 0.1
VGG-19 (w/ BN, lr=0.01)	50.6 ± 0.8	93.4 ± 0.1	-	-
ResNet-110 (w/ BN)	21.5 ± 6.9	94.7 ± 0.1	38.2 ± 0.9	95.4 ± 0.1

also consider the case when $\mathcal{A}$ is Adam and use AdamW [159] to train the ResNet-110 (w/ BN) model on CIFAR-10. Following [160], we use a cosine annealing learning rate schedule with initial learning rate 3×10^{-3} and weight decay 0.2. For GradInit, we set $\gamma = 25$. The Acc_1 and Acc_{best} of Kaiming initialization and GradInit are (36.6 ± 4.7, 94.9 ± 0.1) and (40.2 ± 0.2, 95.3 ± 0.1), respectively. We also show the per-epoch test accuracy in Figure 4.2.

The importance of rescaling BN layers. The scale parameters of BN layers usually controls the variance of activations and gradients in the forward and backward passes, while the linear layers right before the BN layers are scale-invariant. Although changing the magnitudes of the scale-invariant layers affect their learning dynamics [146, 147], we find it important for GradInit to rescale both BN and other linear layers, as shown in Table 4.6.

The importance of GradInit's objective. GradInit is designed to rescale the layers to solve the constrained optimization problem in Eq. 4.1. Simply letting the model to learn to rescale the layers cannot improve the results, and sometimes further causes instability, as shown in Table 4.7. We hypothesize that the bad results with VGG are due to a mismatch between the scales/norms of the gradients of the scalars and the weights.

To make this alternative work, we may need to set different learning rates for the scalars and the weights, which adds to the difficulty of hyperparameter tuning. Note we do not learn the scalars when training networks initialized by GradInit.

Table 4.8: Acc_1/Acc_{best} of ResNet-50 models on ImageNet. Result of MetaInit comes from Dauphin and Schoenholz [6] and we reimplemented the rest.

	Kaiming	FixUp	MetaInit	GradInit
w/ BN	14.6/75.9	-	-	19.2/76.2
w/o BN	-	18.0/75.7	-/75.4	19.2/75.8

GradInit scales to ImageNet. As shown in Table 4.8, GradInit also accelerates convergence and improves test accuracy of ResNet-50 on ImageNet, with or without BN layers, despite having to use a smaller batch size for GradInit than training due to our GPU memory limit. The acceleration achieved by GradInit is even more significant than FixUp, even on the network with the architecture designed for the initialization.

Improved Implementation of MetaInit. MetaInit was originally designed to be task-agnostic, and learns to initialize the network with random samples as inputs. Here, for fair comparisons, we also feed training data to MetaInit, as this should intuitively improve MetaInit for the specific task, and use Adam with the same gradient clipping to optimize the weight norms for MetaInit. Originally, MetaInit [6] uses signSGD with momentum [161], but we found using Adam with the hyperparameters above can give better results for MetaInit. Table 4.9 shows the comparison before and after the changes.

Table 4.9: Acc_1, Acc_{best} for different versions of MetaInit (4 runs). "rand.": using random data. "real": using real data.

config	vgg19 w/o BN	vgg19 w/ BN	res.110 w/o BN	res.110 w/ BN
rand. + signSGD	29.08, 94.36	15.62, 94.53	15.91, 93.91	24.47, 94.93
real + signSGD	30.89, 94.41	16.58, 94.46	16.21, 94.29	26.28, 94.95
real + Adam	30.48, 94.62	35.09, 94.64	14.55, 94.19	29.00, 94.76

4.3.2 Training the Original Transformer Model without Warmup

For a Transformer model to converge, either an explicit or implicit learning rate warmup stage is needed, especially for the original Transformer architecture. It is observed that this Post-LN architecture tends to outperform the Pre-LN model [15] while having higher gradient variance at initialization [134]. Is it believed that this high variance makes a warmup stage inevitable. Previous works that removes the warmup stage often involves architectural changes, e.g., removing Layer Normalizations, since it can surprisingly cause instability [134]. Here, we show that with a proper initialization, we can do away with the warmup stage for the original Post-LN Transformer without any modification to the architecture. Table 4.10 summarizes the architectural changes and best results of methods for improving the initialization of Post-LN Transformers. We compare the stability of the GradInit and Admin initialization methods without warmup in Figure 4.3.

Table 4.10: A comparison of GradInit with with the results from the papers (top 4 rows), and our reimplementation of Admin for training the Post-LN Transformer model on the IWSLT-14 De-EN dataset. "Standard" refers to training with standard initialization and warmup.

Method	Remove LN	$\boldsymbol{w}_{skip}$	Warmup	Optimizer	BLEU
Standard [15]			✓	RAdam	35.6
FixUp [136]	✓		✓	Adam	34.5
T-FixUp [135]	✓			Adam	35.5
Admin [15]		✓		RAdam	35.7
Admin		✓		Adam	36.1
Admin		✓		SGD	33.7
GradInit		✓		Adam	36.0
GradInit				Adam	36.1
GradInit				SGD	35.6

Dataset, Architecture, & Hyperparameters.

IWSLT'14 DE-EN [151] is a German to English translation dataset that has 160k

training examples. Our Transformer model is inherited from [23], which is a Post-LN Transformer placing its Layer Normalization after the summation of the skip connection and the residual branch. It has a 512-dimensional word embedding layer and 1024 dimensions in its hidden FFN layer. We also apply GradInit to the variant from Admin [15], where a learnable vector $\boldsymbol{w}_{skip}$ is element-wise multiplied with each dimension of the skip connection, but we initialize it to 1 for GradInit. Please refer to [15] for how Admin initializes these weights. Following [15], we use a linearly decaying learning rate schedule that decays from the maximum learning rate $\eta_{\max}$ to 0 as the model trains for 100K iterations. For training with SGD, we set the prescribed learning rate $\eta_{\max} = 0.15$, and use $\eta = 0.15, \gamma = 1$ for GradInit. We do a grid search on $\eta_{\max}$ for Admin and report its best result in Table 4.10. For training with Adam, we set $\eta = 5 \times 10^{-4}, \gamma = 10^3$ for the objective of GradInit, so that $\eta\gamma$ is $O(10^{-1})$ as discussed in Section 4.2.4. We train the initialized model $\eta_{\max}$ and β_2 as listed in Figure 4.3. We evaluate the BLEU score every epoch, and report the best BLEU scores throughout training for each run. For GradInit, we set the maximum number of iterations T to 780. By comparison, the warmup stage usually takes 4000 iterations, and we find that if we use 780 steps for warmup, the model does not converge with $\eta_{\max} \geq 3 \times 10^{-4}$. For $\eta_{\max} = 2 \times 10^{-4}$ with 780-step warmup, the BLEU score is 35.4, worse than GradInit's 36.0, showing the advantage of GradInit against warmup.

Stability after removing warmup for Adam. In Figure 4.3, the training process becomes more unstable as β_2 grows larger. From the analysis of RAdam [162], this is because the variance of the gradient has a stronger impact on the adaptive learning rate when β_2 is closer to 1. Therefore, the largest $\beta_2 < 1$ that maintains the performance of

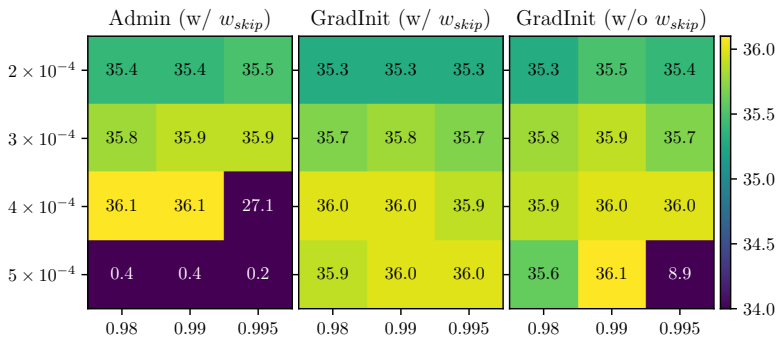

Figure 4.3: BLEU scores for the Post-LN Transformer without learning rate warmup using Adam on IWSLT-14 DE-EN under different learning rates $\eta_{\max}$ (y axis) and β_2 (x axis). Each result is averaged over 4 experiments.

the trained model reflects the stability of the initialization. We can see GradInit results in more stable models than Admin in general, though their best performance numbers are almost the same. In addition, we find $\boldsymbol{w}_{skip}$ can help stabilize training in extreme hyper parameter settings, e.g., at $\eta_{\max} = 5 \times 10^{-4}$ and $\beta_2 = 0.995$ in Figure 4.3, GradInit with $\boldsymbol{w}_{skip}$ obtains a good average BLEU score of 36.0, while without $\boldsymbol{w}_{skip}$ only succeeded in obtaining a BLEU score > 35 for one out of four experiments, resulting in an average BLEU score of 8.9. We also find the network is unable to be trained without learning rate warmup if we just fix $\boldsymbol{w}_{skip}$ to its initial value given by Admin and leave the initialization of other parameters unchanged. Nevertheless, with GradInit, we do not need to modify the architecture of Post-LN Transformer to obtain the same good result as Admin. For a closer look at the stabilization mechanism, we show the weight norms and gradient variance at initialization of the original Post-LN architecture using GradInit and Xavier initialization in Figure 4.9 of Section 4.4.2. For Xavier initialization, the gradient variance is relatively higher for all encoder layers, so GradInit downscales the encoder layer weights more in general. For the LN weights, GradInit only downscales the final LN of both the encoder

and decoder, which reduces the variance of the encoder and decoder during the forward pass. Another strategy GradInit learns is to downscale the weights of the output projection and the FFN layers, so that the residual branch is relatively down-weighted compared with the skip connection, similar to Admin.

Removing warmup without architectural change. Another widely observed phenomenon is that adaptive methods such as Adam seem to be much better than SGD for training Transformer-based language models [145]. Table 4.10 shows that, with GradInit, we can find a good initialization for the Post-LN Transformer on IWSLT-14 DE-EN that trains using SGD *without learning rate warmup nor gradient clipping*, and achieves performance close to Adam trained using the same type of learning rate schedule. By comparison, Admin also makes the Transformer trainable with SGD, but the BLEU score is lower than the one initialized with GradInit. By comparing Figures 4.9 and 4.10, we find GradInit for Adam and SGD adopts different rescaling patterns, with the Adam version depending more on downscaling the residual branches through the FFN and output projection layers than the SGD version, and the SGD version downscaling more in the final FFN block of the decoder. This highlights the importance of considering the optimization algorithm $\mathcal{A}$ in GradInit, and also indicates the presence of different ways to reduce the initial gradient variance.

4.4 Rethinking the Learned Initializations

4.4.1 Magnification Effect of BN

Intuitively, if we stop the gradient passing through the mean and bias of the BN layer during backpropagation, the BN layer will magnify the gradient variance when the variance of its input features is smaller than 1 in the forward pass. Here we show its magnification effect analytically for the practical case where the gradient is not stopped for the mean and bias of the BN layer. From the input to the output, the layers are usually ordered as Linear, BN, Activation. Without loss of generality, we assume the linear layer before BN is $\boldsymbol{X} = \boldsymbol{Z}\boldsymbol{W} + \boldsymbol{b}$, where the output features $\boldsymbol{X} = [\boldsymbol{x}_1, ..., \boldsymbol{x}_n]^T \in \mathbb{R}^{n \times d}$, the input activations $\boldsymbol{Z} \in \mathbb{R}^{n \times k}$, n is the number of samples and d, k are the dimension of each feature vector. Batch Normalization normalizes each activation vector $\boldsymbol{x}_i$ as following

$$\boldsymbol{y}_i = \boldsymbol{\gamma} \frac{\boldsymbol{x}_i - \boldsymbol{\mu}}{\sqrt{\boldsymbol{\sigma}^2 + \epsilon}} + \boldsymbol{\beta}, \tag{4.3}$$

where all operators are element-wise, $\boldsymbol{\gamma}, \boldsymbol{\beta} \in \mathbb{R}^d$ are learnable parameters usually initialized to 1 and 0 respectively, $\epsilon > 0$ is a small constant for numerical stability, and

$$\boldsymbol{\sigma}^2 = \frac{1}{n} \sum_{i=1}^{n} (\boldsymbol{x}_i - \boldsymbol{\mu})^2, \boldsymbol{\mu} = \frac{1}{n} \sum_{i=1}^{n} \boldsymbol{x}_i. \tag{4.4}$$

For most initialization schemes, $\boldsymbol{b}$ is initialized to 0. ϵ is often small and ignorable. Under these two assumptions, each $\boldsymbol{y}_i$ is invariant to the rescaling of $\boldsymbol{W}$. Rescaling $\boldsymbol{W}$ changes the scale of $\boldsymbol{x}_i$, $\boldsymbol{\sigma}$ and $\boldsymbol{\mu}$ homogeneously. Therefore, among all the parameters of the

network, if we only change $\boldsymbol{W}$ by rescaling it into $\alpha\boldsymbol{W}$ ($\alpha > 0$), then $\boldsymbol{y}_i$ does not change, and consequently, $\mathrm{Var}[\boldsymbol{y}_i]$, $\frac{\partial L}{\partial \boldsymbol{y}_i}$ and $\mathrm{Var}[\frac{\partial L}{\partial \boldsymbol{y}_i}]$ do not change, but $\boldsymbol{\sigma}^2$ becomes $\alpha^2\boldsymbol{\sigma}^2$. To see the magnification effect on the gradient variance during backward propagation, we first find the relation between $\frac{\partial L}{\partial \boldsymbol{y}_i}$ and $\frac{\partial L}{\partial(\alpha \boldsymbol{x}_i)}$ under different scales α. In fact,

$$\frac{\partial L}{\partial(\alpha \boldsymbol{x}_i)} = \frac{\boldsymbol{\gamma}}{n\sqrt{\alpha^2\boldsymbol{\sigma}^2 + \epsilon}} \left[n\frac{\partial L}{\partial \boldsymbol{y}_i} - \sum_{j=1}^{n} \frac{\partial L}{\partial \boldsymbol{y}_j} - \frac{\boldsymbol{y}_i - \boldsymbol{\beta}}{\boldsymbol{\gamma}} \sum_{j=1}^{n} \frac{\partial L}{\partial \boldsymbol{y}_j} \cdot \frac{\boldsymbol{y}_j - \boldsymbol{\beta}}{\boldsymbol{\gamma}} \right], \tag{4.5}$$

where, again, all operations are element-wise. Therefore, when α is larger, the variance of the input feature $\alpha^2\boldsymbol{\sigma}^2$ is larger, and the gradient variance becomes smaller after propagated through this BN layer. Since $\boldsymbol{Z}$ remains the same, $\mathrm{Var}\left[\frac{\partial L}{\partial \boldsymbol{W}}\right]$ becomes smaller. This explains why GradInit learns to enlarge the weights of Conv layers in the VGG-19 (w/ BN) experiments. Further, to simplify the analysis and show its magnification effect on gradient variance when $\alpha^2\boldsymbol{\sigma}^2 < 1$, let $\boldsymbol{\gamma} = 1, \boldsymbol{\beta} = 0$, and we assume each dimension of $\frac{\partial L}{\partial \boldsymbol{y}_i}$ is i.i.d., and $\boldsymbol{y}_i$ is independent from $\frac{\partial L}{\partial \boldsymbol{y}_i}$, which is not necessarily a stronger assumption than [16, 133], then

$$\begin{aligned}
\mathrm{Var}\left[\frac{\partial L}{\partial(\alpha \boldsymbol{x}_i)}\right] &= \frac{1}{n^2(\alpha^2\boldsymbol{\sigma}^2 + \epsilon)} \mathrm{Var}\left[n\frac{\partial L}{\partial \boldsymbol{y}_i} - \sum_{j=1}^{n} \frac{\partial L}{\partial \boldsymbol{y}_j} - \boldsymbol{y}_i \sum_{j=1}^{n} \frac{\partial L}{\partial \boldsymbol{y}_j} \cdot \boldsymbol{y}_j \right] \\
&= \frac{1}{n^2(\alpha^2\boldsymbol{\sigma}^2 + \epsilon)} \mathrm{Var}\left[(n - 1 - \boldsymbol{y}_i^2)\frac{\partial L}{\partial \boldsymbol{y}_i} - \sum_{j=1, j\neq i}^{n} (1 + \boldsymbol{y}_i\boldsymbol{y}_j)\frac{\partial L}{\partial \boldsymbol{y}_j} \right] \\
&\geq \frac{1}{n^2(\alpha^2\boldsymbol{\sigma}^2 + \epsilon)} \left\{ (n-1)^2 \mathrm{Var}\left[\frac{\partial L}{\partial \boldsymbol{y}_i}\right] + \sum_{j=1, j\neq i}^{n} \mathrm{Var}\left[\frac{\partial L}{\partial \boldsymbol{y}_j}\right] \right\} \\
&= \frac{n(n-1)}{n^2(\alpha^2\boldsymbol{\sigma}^2 + \epsilon)} \mathrm{Var}\left[\frac{\partial L}{\partial \boldsymbol{y}_i}\right],
\end{aligned} \tag{4.6}$$

where the inequality comes from the assumption that $\boldsymbol{y}_i$ is independent from $\frac{\partial L}{\partial \boldsymbol{y}_i}$ and the fact that $\mathrm{Var}[(X+a)Y] \geq \mathrm{Var}[X] + a^2\mathrm{Var}[Y]$ (a is a constant) when X, Y are independent, and the last equality comes from the i.i.d. assumption. Therefore, if ϵ is ignorable and $\alpha^2\boldsymbol{\sigma}^2 < \frac{n(n-1)}{n^2}$, we will have

$$\mathrm{Var}\left[\frac{\partial L}{\partial(\alpha \boldsymbol{x}_i)}\right] > \mathrm{Var}\left[\frac{\partial L}{\partial \boldsymbol{y}_i}\right], \tag{4.7}$$

i.e., the BN layer magnifies the gradient variance when $\alpha^2\boldsymbol{\sigma}^2$ is small.

4.4.2 Visualizing the Learned Initializations

In this section, we give weight norms and gradient variances before and after GradInit is applied on various datasets and networks. We consider DenseNet-100 (w/o BN) and DenseNet-100 (w/ BN) on CIFAR-10 in Figure 4.7 and Figure 4.8, as well as ResNet-50 on ImageNet in Figure 4.2. We also compare the weight norms and gradient variances of the Post-LN Transformer model initialized using GradInit with $\mathcal{A}$ set to Adam and SGD respectively in Figure 4.9 and Figure 4.10.

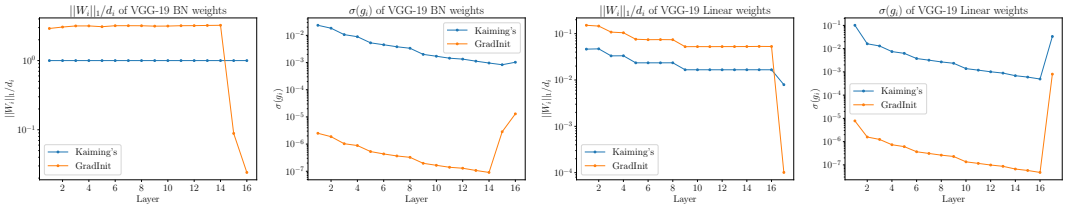

Figure 4.4: Averaged per-dimension weight magnitudes ($\|W_i\|/d_i$) and standard deviation of their gradient ($\sigma(\boldsymbol{g}_i)$) for each layer i of the VGG-19 (w/ BN) on CIFAR-10. The ratio between the weight magnitudes of GradInit and Kaiming Initialization is the learned scale factor of GradInit in each layer. The standard deviation is computed over the mini-batches, with a batch size of 128, with the BN in its training mode. This VGG-19 on CIFAR-10 has only one FC layer, but it has the same number of convolutional layers (16) as its ImageNet version. All the weights are indexed from shallow to deep, so the first 16 entries of the Linear Weights are of Conv layers, while the 17th is the FC layer. Due to the magnification effect of BN, $\sigma(\boldsymbol{g}_1)/\sigma(\boldsymbol{g}_{16})$ for the Conv layers is higher than it is in VGG-19 without BN, shown in Figure 4.6. GradInit learns to reduce the magnification effect of BN layers by scaling up all the Conv layers and most of the BN layers, given it has greatly down scaled the last two BN layers and the final FC layer to reduce the variance in the forward pass.

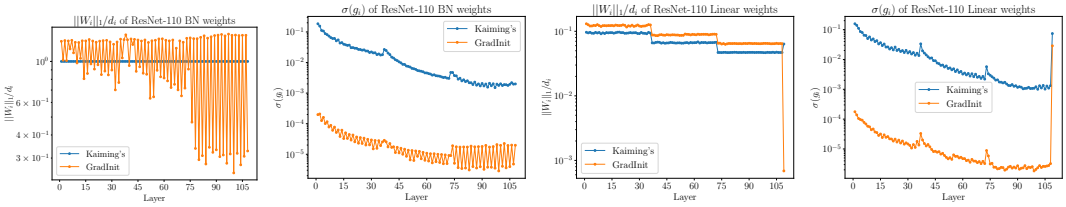

Figure 4.5: Averaged per-dimension weight magnitude ($\|W_i\|/d_i$) and standard deviation of their gradient ($(\sigma(\boldsymbol{g}_i))$) of the Batch Normalization (BN) layers and the linear layers of the ResNet-110 on CIFAR-10. All the layers are indexed from shallow to deep. The linear layers include all Conv layers (2 for each of the residual blocks) and the final FC layer. The ratio between the weight magnitudes of GradInit and Kaiming Initialization is the learned scale factor of GradInit in each layer. The gradient variance is computed with a batch size of 128. GradInit finds a combination of weight norms where the gradient variance is reduced for all layers. Specifically, it learns to further scale down the second BN layer of each residual block in deeper layers, which is a useful strategy, as deeper layers should have less marginal utility for the feature representations, and scaling down those layers helps to alleviate the growth in variance in the forward pass [14]. GradInit also learns to scale up weights of the first BN layer and all the Conv layers in each residual block, which alleviates the magnification effect of the BN layers on the gradient variance during backpropagation, happening if their input features in the forward pass have small variances. The jump on the curves occur when the dimension of the convolutional filters changes.

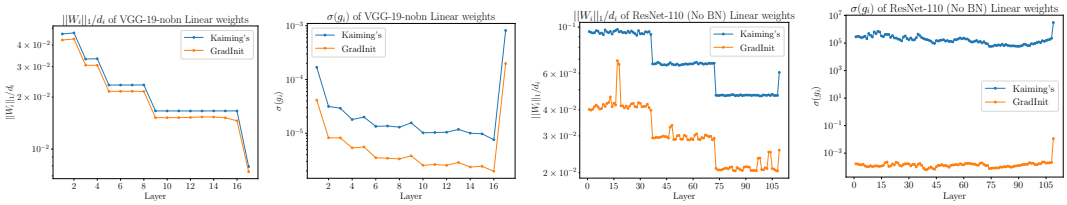

Figure 4.6: Averaged per-dimension weight magnitude ($\|W_i\|/d_i$) and standard deviation of their gradient (($\sigma(\boldsymbol{g}_i)$)) of the VGG-19 (left two) and ResNet-110 (right two) without BN on CIFAR-10, evaluated with a batch size of 128. For VGG-19 (w/o BN), $\sigma(\boldsymbol{g}_i)$ increases at Conv layers with different input and output dimensions during backpropagation. For ResNet-110 without GradInit, the gradient variance is very high due to the cumulative effect of skip connections during the forward pass. In this scenario, to reduce the gradient variance, there is no reason to increase the weights, so GradInit downscales the weights for all layers in both architectures, unlike the case with BN.

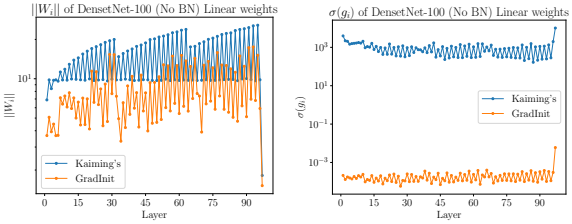

Figure 4.7: Averaged per-dimension weight magnitudes ($\|W_i\|/d_i$) and standard deviation of their gradient ($\sigma(\boldsymbol{g}_i)$) for each linear layer i in DenseNet-100 (w/o BN). All the layers are indexed from shallow to deep. The linear layers include all convolutional layers and the final fully connected layer. Inside each dense block, each layer concatenates all the preceding features, so their input dimension increases, the weight dimension increases and the weight norm increases. Compared with Figure 4.6, DenseNet-100 does not significantly increase the gradient variance during backpropagation. The standard deviation of the gradient is reduced by around 10^6 with GradInit, which explains why it is possible to train DenseNet-100 (w/o BN) without gradient clipping after using GradInit. The major source of gradient reduction of GradInit comes from reducing the weights in each layer.

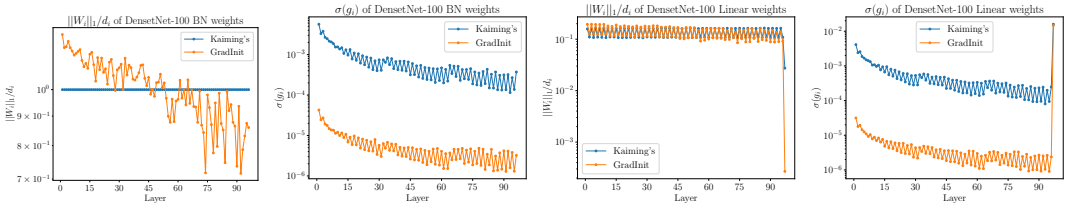

Figure 4.8: Averaged per-dimension weight magnitudes ($\|W_i\|/d_i$) and standard deviation of their gradient ($\sigma(\boldsymbol{g}_i)$) for each (BN or linear) layer i in the DenseNet-100 (w/ BN). All the layers are indexed from shallow to deep. The linear layers include all convolutional layers and the final fully connected layer. The major source of variance reduction comes from down-scaling the final FC layer.

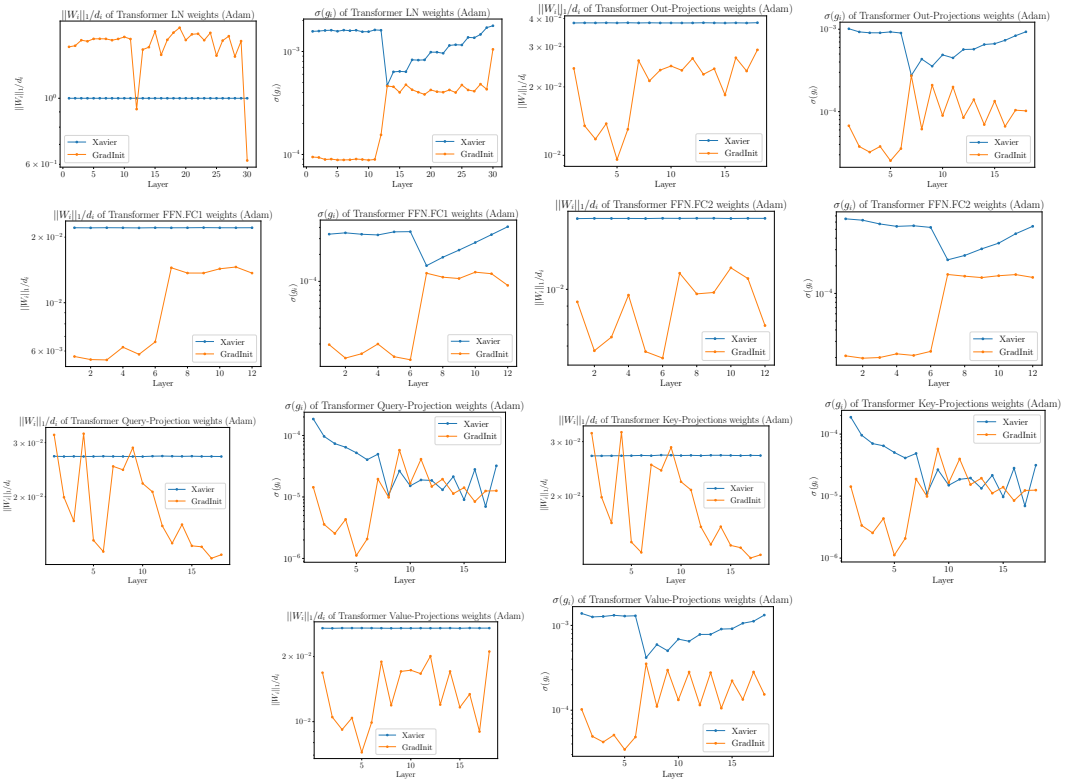

Figure 4.9: Weight norm and averaged per-dimension standard deviation of each weight of the normalization layers and linear layers in the Post-LN Transformer. Here, GradInit sets $\mathcal{A}$ to Adam. The Transformer has 6 Transformer blocks in its encoder and decoder. In each plot, we first list the values for weights in the encoder, and then those in the decoder. Inside each encoder, we first list the weights from the self attention layers and then the those from the FFN layers. Inside each decoder, we first list the weights in the order of self attention, encoder attention and FFN. In general, GradInit reduces the variance for all the weights, except for some of the Query-Projection and Key-Projection weights in the decoder, which are inside the softmax operations in the self attention blocks. The major source of gradient variance reduction comes from downscaling the final LN weights of the decoder, as well as the linear layers of each residual branch (Out-Projection and Value-Projection weights, FFN.FC1 and FFN.FC2 weights) *in each block*. The general strategy is to reduce the norms of Out-Projection, Value-Projection and the FFN layers, which reduces the magnitude of the feature in the residual branch and better preserves the signal in the main branch during forward pass, which improves the stability of training. See detailed analysis by Liu et al. [15].

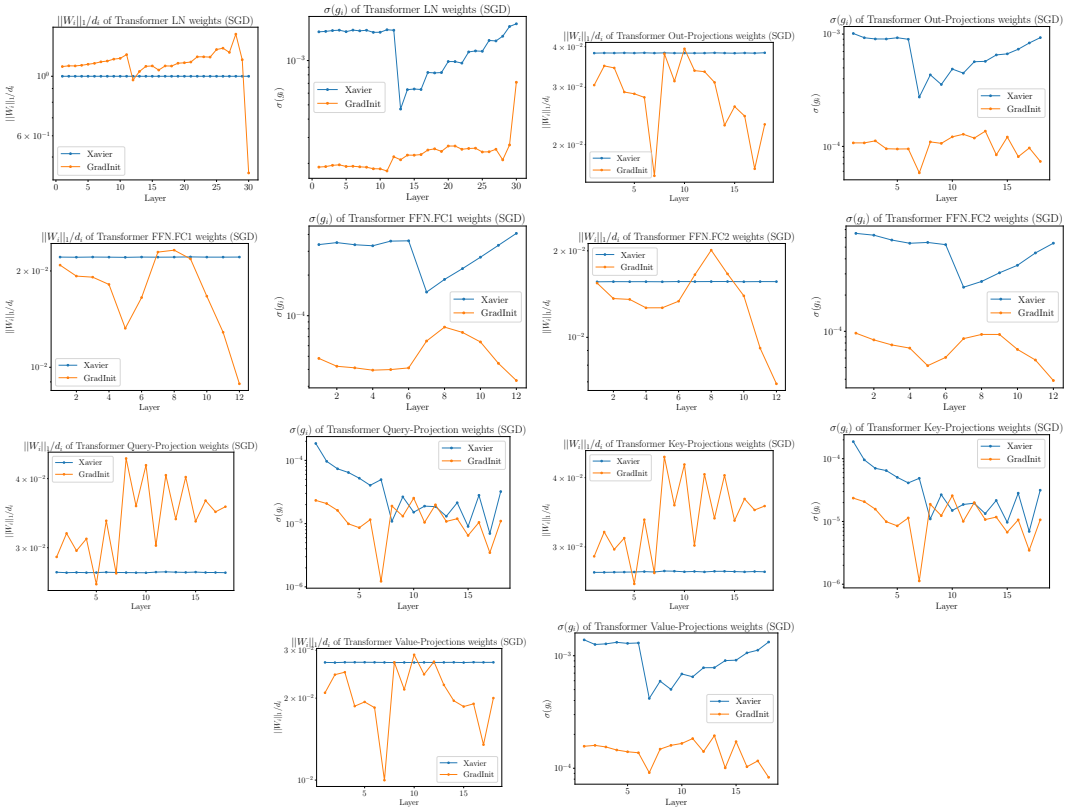

Figure 4.10: Weight norm and averaged per-dimension standard deviation of each weight of the normalization layers and linear layers in the Post-LN Transformer. Here, GradInit sets $\mathcal{A}$ to SGD. The Transformer model and the way each weight is permuted are the same as in Figure 4.9. Again, in general, GradInit reduces the variance for most of the weights, except for some of the Query-Projection and Key-Projection weights in the decoder, which are inside the softmax operations in the self attention blocks. Different from the patterns in the Adam version, which downscale all the weights in every layer except for the Query-Projection and Key-Projection weights, the SGD version of GradInit mostly reduces the weights in the final Transformer block of the decoder. Similar as the case for Adam, the general strategy is to reduce the norms of Out-Projection, Value-Projection and the FFN layers, which reduces the magnitude of the feature in the residual branch and better preserves the signal in the main branch during forward pass, which improves the stability of training. See detailed analysis by Liu et al. [15].

Chapter 5: Reducing Complexity of Transformer Models

Transformers have achieved success in both language and vision domains. However, it is prohibitively expensive to scale them to long sequences such as long documents or high-resolution images, because self-attention mechanism has quadratic time and memory complexities with respect to the input sequence length. In this chapter, we introduce our work on Long-Short Transformer (Transformer-LS), an efficient self-attention mechanism for modeling long sequences with linear complexity for both language and vision tasks [163]. It aggregates a novel long-range attention with dynamic projection to model distant correlations and a short-term attention to capture fine-grained local correlations. We propose a dual normalization strategy to account for the scale mismatch between the two attention mechanisms. Transformer-LS can be applied to both autoregressive and bidirectional models without additional complexity. Our method outperforms the state-of-the-art models on multiple tasks in language and vision domains, including the Long Range Arena benchmark, autoregressive language modeling, and ImageNet classification.

5.1 Introduction

Transformer-based models [164] have achieved great success in the domains of natural language processing (NLP) [165, 166] and computer vision [167–169]. These models benefit from the self-attention module, which can capture both adjacent and long-range correlations between tokens while efficiently scaling on modern hardware. However, the time and memory consumed by self-attention scale quadratically with the input length, making it very expensive to process long sequences. Many language and vision tasks benefit from modeling long sequences. In NLP, document-level tasks require processing long articles [e.g., 170, 171], and the performance of language models often increases with sequence length [e.g., 172, 173]. In computer vision, many tasks involve high-resolution images, which are converted to long sequences of image patches before being processed with Transformer models [167, 169, 174]. As a result, it is crucial to design an efficient attention mechanism for long sequence modeling that generalizes well across different domains.

Various methods have been proposed to reduce the quadratic cost of full attention. However, an efficient attention mechanism that generalizes well in both language and vision domains is less explored. One family of methods is to sparsify the attention matrix with predefined patterns such as sliding windows [e.g., 175–178] and random sparse patterns [179]. These methods leverage strong inductive biases to improve both computational and model performance, but they limit the capacity of a self-attention layer because each specific token can only attend to a subset of tokens. Another family of methods leverages low-rank projections to form a low resolution representation of the input se-

quence, but the successful application of these methods has been limited to certain NLP tasks [e.g., 7, 180, 181]. Unlike sparse attention, this family of methods allows each token to attend to the entire input sequence. However, due to the loss of high-fidelity token-wise information, their performance sometimes is not as good as full attention or sparse attention on tasks that require fine-grained local information, including standard benchmarks in language [8] and vision [182].

Despite the rapid progress in efficient Transformers, some proposed architectures can only be applied to bidirectional models [e.g., 7, 178, 179]. Transformer-based autoregressive models have achieved great successes in language modeling [36], image synthesis [183], and text-to-image synthesis [184], which also involve long texts or high-resolution images. It is desirable to design an efficient transformer that can be applied to both autoregressive and bidirectional models.

5.1.1 Related Works

5.1.1.1 Efficient Transformers

In recent years, many methods have been introduced for dealing with the quadratic cost of full attention. In general, they can be categorized as follows: *i)* Sparse attention mechanism with predefined patterns (e.g., sliding window), including Sparse Transformer [175], Image Transformer [176], Axial Transformer [185] for modeling images, and Longformer [177], blockwise self-attention [186], ETC [178], Big Bird [179] for modeling language. *ii)* Low-rank projection attention, including Linformer [180], Nyströmformer [7], Synthesizer [181]. For example, Linformer uses linear layers to project the original high

resolution keys (K) and values (V) with length n to low resolution with size r ($r \ll n$) and allows all query tokens (Q) to attend these compressed representations. *iii)* Memory-based mechanisms like Compressive Transformer [173] and Set Transformer [187], which use extra memories for caching global long-range information for use in computing attention between distant tokens. *iv)* Kernel-based approximation of the attention matrix, including Performer [188], Linear Transformer [189], and Random Feature Attention [190]. *vi)* Similarity and clustering based methods, including Reformer [191], Routing Transformer [192], and Sinkhorn Transformer [193].

Our method seamlessly integrates both low-rank projection and local window attentions, to leverage their strengths for modeling long-range and short-term correlations. In particular, our long-range attention uses a dynamic low-rank projection to encode the input sequence, and outperforms the previous low-rank projection method used by the Linformer [180]. In the similar vein, a few other methods also try to combine the strengths of different methods. For example, Longformer [177] and ETC [178] augment local window attention with task motivated global tokens. Such global tokens may not be applicable for some tasks (e.g., autoregressive modelling). BigBird [179] further combines local window and global token attention with random sparse attention. It is not applicable in autoregressive tasks because the global token and random sparse pattern are introduced. To compress the model footprint on edge devices, Lite Transformer [194] combines convolution and self-attention, but it still has quadratic complexity for long sequences.

5.1.1.2 Vision Transformers

Vision Transformer (ViT) [167] splits images as small patches and treats the patches as the input word tokens. It uses a standard transformer for image classification and has shown to outperform convolutional neural networks (e.g., ResNet [88]) with sufficient training data. DeiT [195] has applied the teacher-student strategy to alleviate the data efficiency problem of ViT and has shown strong comparable performance using only the standard ImageNet dataset [196]. Instead of applying transformer at a single low resolution of patches (e.g., 16×16 patches), very recent works, including Pyramid Vision Transformer (PVT) [168], Swin-Transformer [197], T2T-ViT [198], Vision Longformer (ViL) [174] and Convolutional Vision Transformer (CvT) [169], stack a pyramid of ViTs to form a multi-scale architecture and model long sequences of image patches at much higher resolution (e.g., $56 \times 56 = 3136$ patches for images with 224×224 pixels). Most of these methods have quadratic complexity of self-attention with respect to the input image size.

To reduce the complexity, Swin-Transformer [197] achieves linear complexity by limiting the computation of self-attention only within each local window. HaloNet [199] applies local attention on blocked images and only has quadratic complexity with respect to the size of the block. Perceiver [200] uses cross-attention between data and latent arrays to replace the self-attention on data to remove the quadratic complexity bottleneck. Vision Longformer (ViL) [174], another concurrent work, achieves linear complexity by adapting Longformer [177] to Vision. ViL augments local window attention with task-specific global tokens, but the global tokens are not applicable for decoding task (e.g.,

image synthesis [183, 184]). In contrast, our method reduces the quadratic cost to linear cost by combining local window attention with global dynamic projection attention, which can be applied to both encoding and decoding tasks.

5.1.2 Our contributions

In this work, we unify a local window attention and a novel long-range attention into a single efficient attention mechanism. We show that these two kinds of attention have complementary effects that together yield the state-of-the-art results on a range of tasks in language and vision, for both autoregressive and bidirectional models. Specifically, we make the following contributions:

- We propose Long-Short Transformer (Transformer-LS), an efficient Transformer that integrates a dynamic projection based attention to model long-range correlations, and a local window attention to capture fine-grained correlations. Transformer-LS can be applied to both autoregressive and bidirectional models with linear time and memory complexity.

- We compute a dynamic low-rank projection, which depends on the content of the input sequence. In contrast to previous low-rank projection methods, our dynamic projection method is more flexible and robust to semantic-preserving positional variations (e.g., insertion, paraphrasing). We demonstrate that it outperforms previous low-rank methods [7, 180] on Long Range Arena benchmark [8].

- We identify a scale mismatch problem between the embeddings from the long-range and short-term attentions, and design a simple but effective dual normalization strategy,

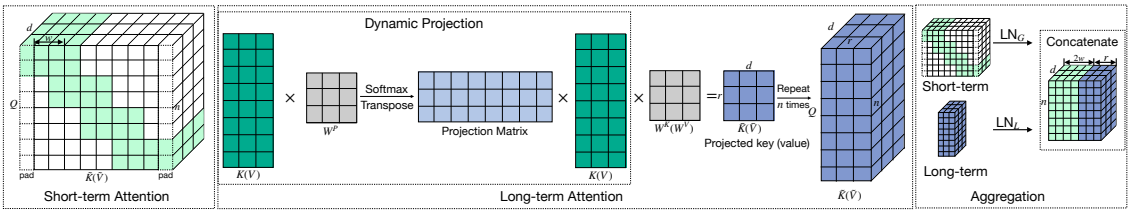

Figure 5.1: Long-short term attention of a single attention head. Here, the sequence length $n = 8$, hidden dimension $d = 3$, local window segment size $w = 2$, and rank of dynamic projection $r = 3$. Within the figure, $K(V)$ denotes key K or value V. In the left figure, we virtually replicate K or $V \in \mathbb{R}^{n \times d}$ into n rows, and highlight the keys and values within the attention span (denoted as $\tilde{K}(\tilde{V})$) of all n queries Q for the short-term attention. In the middle figure, all queries attend to the same projected keys $\bar{K}$ and values $\bar{V}$ within the long-term attention. In the right figure, $\tilde{K}(\tilde{V})$ and $\bar{K}(\bar{V})$ are first normalized with two sets of LayerNorms, and the queries attend to normalized $\tilde{K}(\tilde{V})$ and $\bar{K}(\bar{V})$ within their attention span simultaneously.

termed *DualLN*, to account for the mismatch and enhance the effectiveness of the aggregation.

- We demonstrate that Long-Short Transformer, despite its low memory and runtime complexity, outperforms the state-of-the-art models on a set of tasks from Long Range Arena, and autoregressive language modeling on enwik8 and text8. In addition, the proposed efficient attention mechanism can be easily applied to the most recent vision transformer architectures [169, 174] and provides state-of-the-art results, while being more scalable to high-resolution images. We also investigate the robustness properties of the Transformer-LS on diverse ImageNet datasets.

5.2 Long-short Transformer

Transformer-LS approximates the full attention by aggregating long-range and short-term attentions, while maintaining its ability to capture correlations between all input to-

kens. In this section, we first introduce the preliminaries of multi-head attention in Transformer. Then, we present the short-term attention via sliding window, and long-range attention via dynamic projection, respectively. After that, we propose the aggregating method and dual normalization (DualLN) strategy. See Figure 5.1 for an illustration of our long-short term attention.

5.2.1 Preliminaries and Notations

Multi-head attention is a core component of the Transformer [164], which computes contextual representations for each token by attending to the whole input sequence at different representation subspaces. It is defined as

$$\text{MultiHead}(Q, K, V) = \text{Concat}(H_1, H_2, ..., H_h)W^O, \tag{5.1}$$

where $Q, K, V \in \mathbb{R}^{n\times d}$ are the query, key and value embeddings, $W^O \in \mathbb{R}^{d\times d}$ is the projection matrix for output, the i-th head $H_i \in \mathbb{R}^{n\times d_k}$ is the scaled dot-product attention, and $d_k = d/h$ is the embedding dimension of each head,

$$H_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) = \text{softmax}\left[\frac{QW_i^Q\left(KW_i^K\right)^\intercal}{\sqrt{d_k}}\right]VW_i^V = A_iVW_i^V, \tag{5.2}$$

where $W_i^Q, W_i^K, W_i^V \in \mathbb{R}^{d\times d_k}$ are learned projection matrices, and $A_i \in \mathbb{R}^{n\times n}$ denotes the full attention matrix for each attention head. The complexity of computing and storing A_i is $O(n^2)$, which can be prohibitive when n is large. For simplicity, our discussion below is based on the case of 1D input sequences. It is straightforward to extend to the

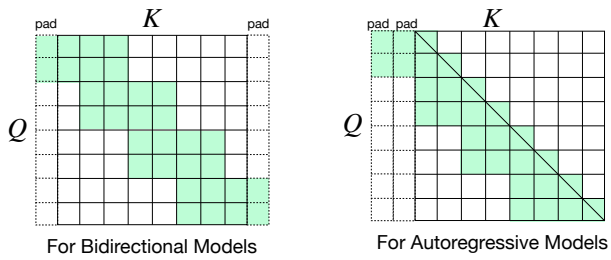

Figure 5.2: An illustration of our sliding window attention in 1D autoregressive and bidirectional models. Here, we use a group size $w = 2$. Each token inside each group are restricted to attend to at most $2w$ tokens. In the bidirectional model, they attend to w tokens from the home segment, and $w/2$ tokens to the left and right of the home segment respectively. In the autoregressive model, they attend to w tokens to the left of the home segment, as well as all tokens within the home segment that is not a future token.

2D image data given a predetermined order.

5.2.2 Short-term Attention via Segment-wise Sliding Window

We use the simple yet effective sliding window attention to capture fine-grained local correlations, where each query attends to nearby tokens within a fixed-size neighborhood. Similar techniques have also been adopted in [174, 177, 179]. Specifically, we divide the input sequence into disjoint segments with length w for efficiency reason. All tokens within a segment attend to all tokens within its home segment, as well as $w/2$ consecutive tokens on the left and right side of its home segment (zero-padding when necessary), resulting in an attention span over a total of $2w$ key-value pairs. See Figure 5.2 for an illustration. For each query Q_t at the position t within the i-th head, we denote the $2w$ key-value pairs within its window as $\tilde{K}_t, \tilde{V}_t \in \mathbb{R}^{2w \times d}$. For implementation with PyTorch, this segment-wise sliding window attention is faster than the per-token sliding window attention where each token attends to itself and w tokens to its left and right, and its memory consumption scales linearly with sequence length; see [177] and

our Figure 5.5 for more details.

The sliding window attention can be augmented to capture long-range correlations in part, by introducing different dilations to different heads of sliding window attention [177]. However, the dilation configurations for different heads need further tuning and an efficient implementation of multi-head attention with different dilations is non-trivial. A more efficient alternative is to augment sliding window attention with random sparse attention [179], but this does not guarantee that the long-range correlations are captured in each layer as in full attention. In the following section, we propose our long-range attention to address this issue.

5.2.3 Long-range Attention via Dynamic Projections

Previous works have shown that the self-attention matrix can be well approximated by the product of low-rank matrices [180]. By replacing the full attention with the product of low-rank matrices [7, 181, 188, 201, 202], each query is able to attend to all tokens. Linformer [180] is one of the most representative models in this category. It learns a fixed projection matrix to reduce the length of the keys and values, but the fixed projection is inflexible to semantic-preserving positional variations.

Starting from these observations, we parameterize the dynamic low-rank projection at i-th head as $P_i = f(K) \in \mathbb{R}^{n \times r}$, where $r \ll n$ is the low rank size and P_i depends on all the keys $K \in \mathbb{R}^{n \times d}$ of input sequence. It projects the $(n \times d_k)$-dimensional key embeddings KW_i^K and value embeddings VW_i^V into shorter, $(r \times d_k)$-dimensional key $\bar{K}_i$ and value $\bar{V}_i$ embeddings. Unlike Linformer [180], the low-rank projection matrix

is dynamic, which depends on the input sequence and is intended to be more flexible and robust to, e.g., insertion, deletion, paraphrasing, and other operations that change sequence length. See Table 5.3 for examples. Note that, the query embeddings $QW_i^Q \in \mathbb{R}^{n\times d_k}$ are kept at the same length, and we let each query attend to $\bar{K}_i$ and $\bar{V}_i$. In this way, the full $(n \times n)$ attention matrix can be decomposed into the product of two matrices with r columns or rows. Specifically, we define the dynamic projection matrix $P_i \in \mathbb{R}^{n\times r}$ and the key-value embeddings $\bar{K}_i, \bar{V}_i \in \mathbb{R}^{r\times d_k}$ of low-rank attention as

$$P_i = \text{softmax}(KW_i^P),\ \bar{K}_i = P_i^\intercal KW_i^K, \bar{V}_i = P_i^\intercal VW_i^V, \tag{5.3}$$

where $W_i^P \in \mathbb{R}^{d\times r}$ are learnable parameters,[1] and the softmax normalizes the projection weights on the first dimension over all n tokens, which stabilizes training in our experiments. Note that $K = V$ in all the experiments we have considered, so P_i remains the same if it depends on V. The computational complexity of Eq. 5.3 is $O(rn)$.

To see how the full attention is replaced by the product of low-rank matrices, we compute each head $H_i \in \mathbb{R}^{n\times d_k}$ of long-range attention as,

$$\bar{H}_i = \underbrace{\text{softmax}\left[\frac{QW_i^Q\bar{K}_i^\intercal}{\sqrt{d_k}}\right]}_{\bar{A}_i}\bar{V}_i = \bar{A}_i\big(P_i^\intercal VW_i^V\big), \tag{5.4}$$

so the full attention is now replaced with the implicit product of two low-rank matrices $\bar{A}_i \in \mathbb{R}^{n\times r}$ and $P_i^\intercal \in \mathbb{R}^{r\times n}$, and the computational complexity is reduced to $O(rn)$. Note the effective attention weights of a query on all tokens still sum to 1. Our global attention

[1] For the CvT-based vision transformer model, we replace W_i^P with a depth-wise separable convolution, just as its query, key and value projections.

allows each query to attend to all token embeddings within the same self-attention layer. In contrast, the sparse attention mechanisms [177, 179] need stack multiple layers to build such correlations.

Application to Autoregressive Models: In autoregressive models, each token can only attend to the previous tokens, so the long-range attention should have a different range for different tokens. A straightforward way to implement our global attention is to update $\bar{K}_i, \bar{V}_i$ for each query recurrently, but this requires re-computing the projection in Eq. equation 5.3 for every token due to the nonlinearity of softmax, which results in $O(rn^2)$ computational complexity. To preserve the linear complexity, for autoregressive models, we first divide the input sequence into equal-length segments with length l, and apply our dynamic projection to extract $\bar{K}_i, \bar{V}_i$ from each segment. Each token can only attend to $\bar{K}_i, \bar{V}_i$ of segments that do not contain its future tokens. Formally, let Q_t be the query at position t, $K_{(l-1)s:ls}, V_{(l-1)s:ls}$ be the key-value pairs from the s-th segment, and $s_t = \lfloor t/l \rfloor$. For autoregressive models, we compute the long-range attention of Q_t by attending to $K_{i,t}, V_{i,t}$, defined as

$$\bar{K}_{i,t} = [P_{i,1}^{\intercal} K_{1:l}; ...; P_{i,s_t}^{\intercal} K_{(l-1)s_t:ls_t}] W_i^K, \bar{V}_{i,t} = [P_{i,1}^{\intercal} V_{1:l}; ...; P_{i,s_t}^{\intercal} V_{(l-1)s_t:ls_t}] W_i^V. \quad (5.5)$$

In this way, the dynamic low-rank projection is applied to each segment only once in parallel, preserving the linear complexity and the high training speed. By comparison, Random Feature Attention [190] is slow at training due to the requirement for recurrence.

5.2.4 Aggregating Long-range and Short-term Attentions

To aggregate the local and long-range attentions, instead of adopting different attention mechanisms for different heads [175, 177, 194], we let each query at i-th head attend to the union of keys and values from the local window and global low-rank projections, thus it can learn to select important information from either of them. We find this aggregation strategy works better than separating the heads in our initial trials with the autoregressive language models. Specifically, for the i-th head, we denote the global low-rank projected keys and values as $\bar{K}_i, \bar{V}_i \in \mathbb{R}^{r \times d_k}$, and the local keys and values as $\tilde{K}_t, \tilde{V}_t \in \mathbb{R}^{2w \times d}$ within the local window of position t for the query Q_t. Then the i-th attention $H_{i,t}$ at position t is

$$H_{i,t} = \text{softmax}\left[\frac{Q_t W_i^Q \left[\tilde{K}_t W_i^K; \bar{K}_i\right]^\intercal}{\sqrt{d_k}}\right] [\tilde{V}_t W_i^V; \bar{V}_i]. \tag{5.6}$$

where $[\cdot\ ;\cdot]$ denotes concatenating the matrices along the first dimension. Furthermore, we find a scale mismatch between the initial norms of $\tilde{K}_t W_i^K$ and $\bar{K}_i$, which biases the attention to the local window at initialization for both language and vision tasks. We introduce a normalization strategy (DualLN) to align the norms and improve the effectiveness of the aggregation in the following.

DualLN: For Transformers with Layer Normalization (LN) (see [134] for an illustration), the K_i, V_i embeddings are the outputs of LN layers, so they have zero mean and unit variance at initialization. The ℓ_2 norm of vectors with zero-mean entries is proportional to their variance in expectation. We note a weighted average will reduce the

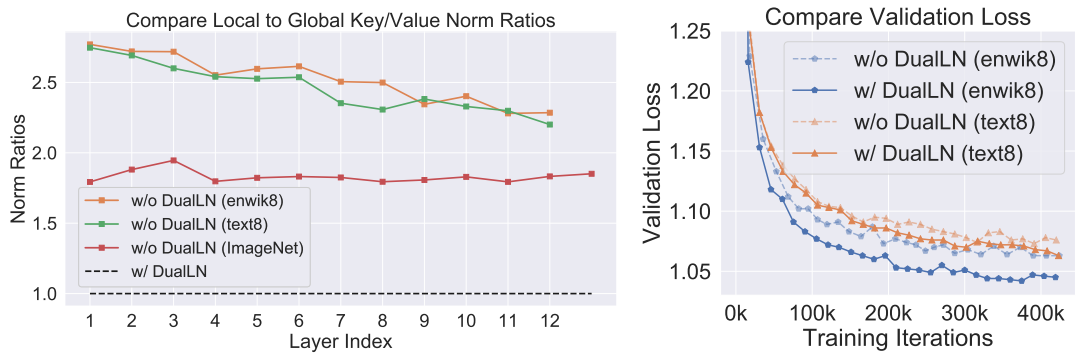

Figure 5.3: Left: Ratios of the average ℓ_2 norms of the local window to global low-rank key/value embeddings at initialization. Without DualLN, the sparse and low-rank embeddings have a magnitude mismatch. With DualLN, the ratios will be 1.0 at every layer, which will facilitate optimization. Right: The validation loss of Transformer-LS with and without DualLN on enwik8 and text8.

variance and therefore the norm of such zero-mean vectors.

Intuitively, at initialization, following similar assumptions as [16, 203], the entries of K, V should have zero mean. Since each entry of $\bar{K}, \bar{V}$ is a weighted mean of K, V, they have smaller variance unless one of the weights is 1. Given that $\bar{K}, \bar{V}$ are also zero-mean, the norm of their embedding vectors (their rows), which is proportional to the variance, is smaller. For the key-value embeddings from short-term attention, $\tilde{K}, \tilde{V}$ are just a subset of K, V, so their embedding vectors should have the same norm as rows of K, V in expectation. As a result, the embedding vectors from low-rank attention in the weighted average $\bar{K}_i, \bar{V}_i$ of Eq. equation 5.3 will have smaller norms than the regular key and value embeddings from sliding window attention (see Figure 5.3 Left for an illustration). This scale mismatch causes two side effects. First, the inner product $Q_t W_i^Q \bar{K}_i^\intercal$ from local-rank component tends to have smaller magnitude than the local window one, thus the attention scores on long-range attention is systematically smaller. Second, the key-value pairs $\bar{K}_i, \bar{V}_i$ for the low-rank attention will naturally have less impact on the direction of H_i

even when low-rank and local window are assigned with same attention scores, since $\bar{V}_i$ has smaller norms. Both effects lead to small gradients on the low-rank components and hinders the model from learning to effectively use the long-range correlations.

To avoid such issues, we add two sets of Layer Normalizations after the key and value projections for the local window and global low-rank attentions, so that their scales are aligned at initialization, but the network can still learn to re-weight the norms after training. Specifically, the aggregated attention is now computed as

$$H_{i,t} = \text{softmax}\left[\frac{Q_t W_i^Q \left[\text{LN}_L(\tilde{K}_t W_i^K); \text{LN}_G(\bar{K}_i)\right]^\intercal}{\sqrt{d_k}}\right] [\text{LN}_L(\tilde{V}_t W_i^V); \text{LN}_G(\bar{V}_i)], \quad (5.7)$$

where $\text{LN}_L(\cdot), \text{LN}_G(\cdot)$ denote the Layer Normalizations for the local and global attentions respectively. In practice, to maintain the consistency between the local attention and dynamic projection, we use $\text{LN}_L(K), \text{LN}_L(V)$ instead of K, V to compute $\bar{K}_i, \bar{V}_i$ in Eq. 5.3. As illustrated in Figure 5.3 Right, the Transformer-LS models trained with DualLN has consistently lower validation loss than the models without DualLN.

5.2.5 Long-short Term Attention for Autoregressive Models

We give an illustration for the segment-wise dynamic projection for autoregressive models as discussed in Section 5.2.3. With the segment-wise formulation, we can first compute the low-rank projection for each segment in parallel, and each query will only attend to the tokens from segments that do not contain the future token or the query token itself. The whole process is efficient and maintain the $O(n)$ complexity, unlike RFA [190] which causes a slow-down in training due to the requirement for cumulative sum. How-

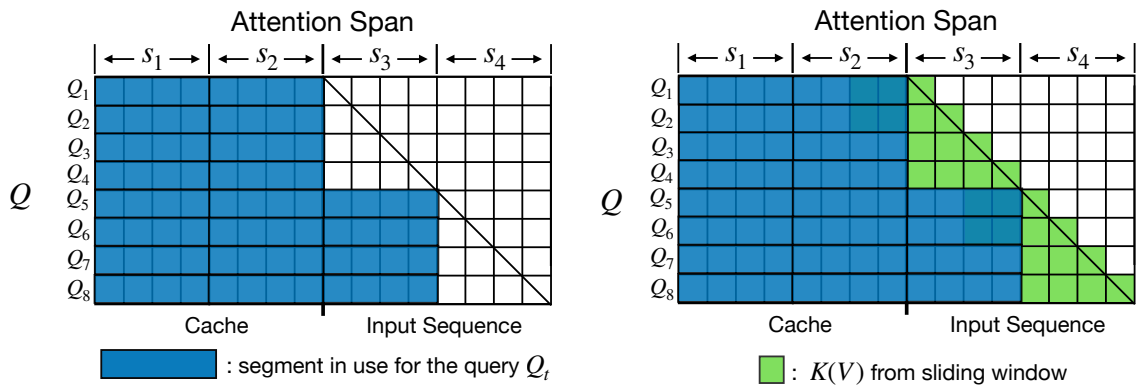

Figure 5.4: An illustration of effective attention span (colored regions) in Transformer-LS when the segment size for the low-rank attention is $\ell = 4$, and the segment size for the sliding window attention is $w = 2$. Left: the attention span of only the low-rank attention (segment-wise dynamic projection). Right: the attention span of the aggregated attention.

ever, in this way, some of the most recent tokens are ignored, as shown in Figure 5.4 (left). The window attention (with segment size $w \geq l/2$) becomes an indispensable component in this way, since it fills the gap for the missing recent tokens, as shown in Figure 5.4.

5.3 Experiments

In this section, we demonstrate the effectiveness and efficiency of our method in both language and vision domains. We use PyTorch for implementation and count the FLOPs using fvcore [204].

5.3.1 Bidirectional Modeling on Long Range Arena and IMDb

Settings on LRA. To evaluate Long-Short Transformer as a bidirectional encoder for long text, we train our models on the three NLP tasks, **ListOps**, **Text**, and **Retrieval**, from the recently proposed Long Range Arena (LRA) benchmark [8], following the setting of Peng et al. [190] and Tay et al. [205]. Details of the three tasks:

Table 5.1: Accuracy (%) and FLOPs (G) on Long Range Arena (LRA), with the model configs annotated (see Table 5.2 for details). All results are averages of 4 runs with different random seeds.

Task (mean ± std.) of sequence length	ListOps (888 ± 339)		Text (1296 ± 893)		Retrieval (3987 ± 560)		Average
Model	Acc.	FLOPs	Acc.	FLOPs	Acc.	FLOPs	Acc.
Full Attention [164]	37.1 ± 0.4	1.21	65.4 ± 0.3	4.57	**82.3** ± 0.4	9.14	61.59
Reformer [191] (2)	36.4 ± 0.4	0.27	64.9 ± 0.4	0.58	78.6 ± 0.7	1.15	59.99
Linformer [180] (k=256)	37.4 ± 0.4	0.41	56.1 ± 1.5	0.81	79.4 ± 0.9	1.62	57.62
Performer [188] ($r = 256$)	32.8 ± 9.4	0.41	65.2 ± 0.2	0.82	81.7 ± 0.2	1.63	59.90
Nyströmformer [7] ($l = 128$)	37.3 ± 0.2	0.61	65.8 ± 0.2	1.02	81.3 ± 0.3	2.03	61.46
Transformer-LS ($w, r = 8, 32$)	**37.5** ± 0.3	0.20	**66.0** ± 0.2	0.40	81.8 ± 0.3	0.80	**61.77**
Dynamic Projection (best)	37.8 ± 0.2	0.15	66.3 ± 0.7	0.69	81.9 ± 0.5	2.17	61.98
Transformer-LS (best)	**38.4** ± 0.4	0.16	**68.4** ± 0.8	0.29	**82.0** ± 0.5	2.17	**62.90**

Table 5.2: Configurations of our method corresponding to the best results (Transformer-LS (best)) in Table 5.1.

	ListOps (2k)		Text (4k)		Retrieval (4k)	
	w	r	w	r	w	r
Dynamic Projection	0	4	0	128	0	256
Transformer-LS	16	2	1	1	1	254

Table 5.3: Comparing the robustness of the models under test-time insertions and deletions. DP refers to long-range attention via Dynamic Projection, and Win. refers to sliding window attention.

Task	Text			Retrieval		
Test Perturb	None	Insertion	Deletion	None	Insertion	Deletion
Linformer	56.12	55.94	54.91	79.37	53.66	51.75
DP	66.28	63.16	58.95	81.86	**70.01**	**64.98**
Linformer + Win.	59.63	56.69	56.29	79.68	52.83	52.13
DP + Win. (ours)	**68.40**	**66.34**	**62.62**	**81.95**	69.93	64.19

Table 5.4: Comparing the results of pretrained language models fine-tuned on IMDb.

Model	RoBERTa-base	RoBERTa-large	Longformer-base	LS-base	LS-large
Accuracy	95.3	96.5	95.7	96.0	**96.8**

- **ListOps**. ListOps [206] is designed to measure the parsing ability of models through hierarchically structured data. We follow the setting in [8] in which each instance contains 500-2000 tokens.

- **Text**. This is a binary sentiment classification task of predicting whether a movie

review from IMDb is positive or negative [207]. Making correct predictions requires a model to reason with compositional unsegmented char-level long sequences with a maximum length of 4k.

- **Retrieval**. This task is based on the ACL Anthology Network dataset [208]. The model needs to classify whether there is a common citation between a pair of papers, which evaluates the model's ability to encode long sequences for similarity-based matching. The max sequence length for each byte-level document is 4k and the model processes two documents in parallel each time.

On all LRA tasks, the models have 2 layers, with embedding dimension $d = 64$, head number $h = 2$, FFN hidden dimension 128, smaller than those from [8]. Same as [8], we add a CLS token as a global token and use its embedding in the last layer for classification. We re-implement the methods evaluated by Xiong et al. [7], and report the best results of our re-implementation and those reported by Xiong et al. [7]. For our method, the results we run a grid search on the window size w and the projected dimension r, and keep $2w+r \leq 256$ to make the complexity similar to the other methods. The maximum sequence length for **ListOps** and **Text** are 2048 and 4096. For **Retrieval**, we set the max sequence for each of the two documents to 4096.

LRA Results. For fair comparisons, we use the PyTorch implementation and the same data preprocessing/split, training hyperparameters and model size from [7], except for **Retrieval** where we accidentally used more warmup steps and improved the results for all models. The results on these three tasks are given in Table 5.1. We give the results of our model on the image-based tasks, implemented in PyTorch, in Table 5.5.

Table 5.5: Comparing our model (Transformer-LS) with other methods on the image-based tasks of LRA. For the results of other models, we take their highest scores from [7] and [8].

Model	Transformer-LS	Linformer	Reformer	Performer	Sparse. Trans.	Nystromformer	Full Att.
Image	45.05	38.56	43.29	42.77	44.24	41.58	42.44
Pathfinder	76.48	76.34	69.36	77.05	71.71	70.94	74.16

To compare the results with the implementations from the original LRA paper [8], we re-implement our method in JAX and give the comparisons with other methods in Table 5.6. The accuracies of other methods come from the LRA paper. We evaluate the per-batch latency of all models on A100 GPUs using their official JAX implementation from the LRA paper. Our method still achieves improvements while being efficient enough. We were unable to run Reformer with the latest JAX since JAX has deleted `jax.custom_transforms`, which is required by the Reformer implementation, from its API.[2] Note the relative speedups from the LRA paper are evaluated on TPUs.

Table 5.6: Comparing the test scores and latency of models on LRA, implemented in JAX.

Model	ListOps		Text		Retrieval	
	Acc.	Latency (s)	Acc.	Latency (s)	Acc.	Latency (s)
Local Att	15.82	0.151	52.98	0.037	53.39	0.142
Linear Trans.	16.13	0.156	65.9	0.037	53.09	0.142
Reformer	37.27	-	56.10	-	53.40	-
Sparse Trans.	17.07	0.447	63.58	0.069	59.59	0.273
Sinkhorn Trans.	33.67	0.618	61.20	0.048	53.83	0.241
Linformer	35.70	0.135	53.94	0.031	52.27	0.117
Performer	18.01	0.138	65.40	0.031	53.82	0.120
Synthesizer	36.99	0.251	61.68	0.077	54.67	0.306
Longformer	35.63	0.380	62.85	0.112	56.89	0.486
Transformer	36.37	0.444	64.27	0.071	57.46	0.273
BigBird	36.05	0.269	64.02	0.067	59.29	0.351
Transformer-LS	37.65	0.187	76.64	0.037	66.67	0.201

Results on IMDb. In addition, we follow the pretraining procedure of Long-

[2] https://github.com/google/jax/pull/2026

former [177] to pretrain our models based on RoBERTa-base and RoBERTa-large [4] on the pre-training dataset similar to RoBERTa's, and fine-tune it on the IMDb sentiment classification dataset. The results are given in Table 5.4.

Discussions of the results. From Table 5.4, our base model outperforms Longformer-base, and our large model achieves improvements over RoBERTa-large, demonstrating the benefits of learning to model long sequences. Comparisons with models on LRA are given in Table 5.1. Transformer-LS (best) with the best configurations of w, r for each task are given in Table 5.2. We also report the results of using fixed hyperparameter $w = 8, r = 32$ on all tasks. Overall, our Transformer-LS (best) is significantly better than other efficient Transformers, and the model with $w, r = 8, 32$ performs favorably while using only about 50% to 70% computation compared to other efficient Transformers on all three tasks. The advantage of aggregating local and long-range attentions is the most significant on **ListOps**, which requires the model to understand the tree structures involving both long-term and short-term relations. On **Retrieval**, where document-level encoding capability is tested, we find our global attention more effective than window attention. The test accuracy of using only dynamic projection is about 10% higher than Linformer on **Text** (i.e., 66.28 vs. 56.12), which has the highest variance in sequence length (i.e. standard deviation 893). This demonstrates the improved flexibility of dynamic projection at learning representations for data with high variance in sequence length, compared to the learned but fixed projection of Linformer. Similarly, Linformer, Nyströmformer and our model outperform full attention on **ListOps**, indicating they may have better inductive bias, and efficient Transformers can have better efficacy beyond efficiency.

Robustness of Dynamic Projection. In Table 5.3, we compare the robustness of

Linformer and the proposed Dynamic Projection (DP) against insertion and deletion on Text and Retrieval tasks of LRA. We train the models on the original, clean training sets and only perturb their test sets. For insertion, we insert 10 random punctuations at 10 random locations of each test sample. For deletion, we delete all punctuations from the test samples. Both transforms are label-preserving in most cases. By design, dynamic projection is more robust against location changes.

5.3.2 Autoregressive Language Modeling

We compare our method with other efficient transformers on the character-level language modeling where each input token is a character.

Setup. We train and evaluate our model on enwik8 and text8, each with 100M characters and are divided into 90M, 5M, 5M for train, dev, test, following [209]. Our smaller 12-layer and larger 30-layer models are Pre-LN Transformers with the same width and depth as Longformer [8], except that we add relative position encoding to the projected segments in each layer. We adopt the cache mechanism of Transformer-XL [172], setting the cache size to be the same as the input sequence length. We follow similar training schedule as Longformer, and train our model in 3 phases with increasing sequence lengths. The input sequence lengths are 2048, 4096 and 8192 respectively for the 3 phases. By comparison, Longformer trains their model in 5 phases on GPUs with 48GB memory (The maximal of ours is 32GB) where the sequence length is 23,040 in the last phase. The window size of Longformer increases with depth and its average window size is 4352 in phase 5, while our effective number of attended tokens is 1280 on average in

the last phase. Each experiment takes around 8 days to finish on 8 V100 GPUs. For testing, same as Longformer, we split the dataset into overlapping sequences of length 32K at a step size of 512, and evaluate the BPCs for predicting the next 512 tokens given the previous 32K characters.

Hyper-parameters. Throughout training, we set the window size $w = 512$, the segment length $l = 16$, and the dimension of the dynamic low-rank projection $r = 1$, which in our initial experiments achieved better efficiency-BPC trade-off than using $l = 32, r = 1$ or $l = 64, r = 4$. Our small and large models have the same architecture as Longformer [177], except for the attention mechanisms. We use similar training schedules as Longformer [177]. Specifically, for all models and both datasets, we train the models for 430k/50k/50k steps with 10k/5k/5k linear learning rate warmup steps, and use input sequence lengths 2048/4096/8192 for the 3 phases. We use constant learning rate after warmup. We compared learning rates from {1.25e-4, 2.5e-4,5e-4,1e-3} for 100k iterations and found 2.5e-4 to work the best for both models on enwik8, and 5e-4 to work the best on text8. The batch sizes for the 3 phases are 32, 32, 16 respectively. Unlike Longformer and Transformer-XL, we remove gradient clipping and found the model to have slightly faster convergence in the beginning while converging reliably. For smaller models, we use dropout rate 0.2 and weight decay 0.01. For the larger model, we use dropout 0.4 and weight decay 0.1.

Results. Table 5.7 shows comparisons on text8 and enwik8. Our method has achieved state-of-the-art results. On text8, we achieve a test BPC of 1.09 with the smaller model. On enwik8, our smaller model achieves a test BPC of 0.99, and outperforms the state-of-the-art models with comparable number of parameters. Our larger model obtains

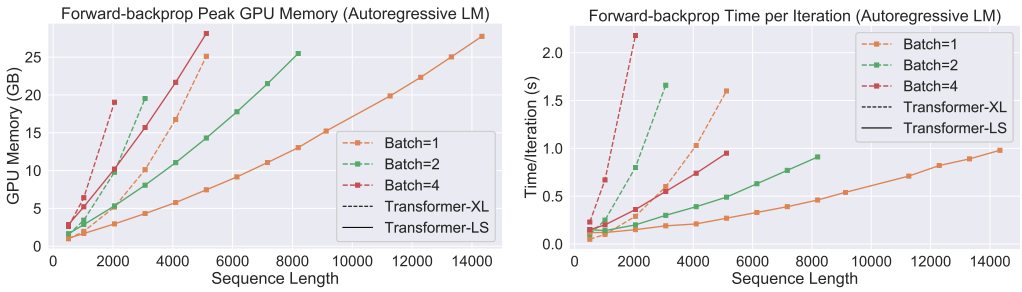

Figure 5.5: Running time and memory consumption of Transformer-XL (full attention) and our Transformer-LS on Char-LM. We increase the sequence length until we use up the 32GB of memory on a V100 GPU. Transformer-LS is the same smaller model in Table 5.7. We use dashed lines to represent the full attention Transformer and solid lines to represent our model. We use different colors to represent different batch sizes.

Table 5.7: BPC (↓) of smaller models on enwik8 and text8 (left), and larger models on enwik8 (right).

Method	#Param	text8		enwik8	
		Dev	Test	Dev	Test
T12 [210]	44M	-	1.18	-	1.11
Transformer-XL [172]	41M	-	-	-	1.06
Reformer [191]	-	-	-	-	1.05
Adaptive [211]	38M	1.05	1.11	1.04	1.02
BP-Transformer [212]	38M	-	1.11	-	1.02
Longformer [8]	41M	1.04	1.10	1.02	1.00
Transformer-LS	44M	1.03	**1.09**	1.01	**0.99**

Method	#Param	Test BPC
Transformer-XL [172]	88M	1.03
Transformer-XL [172]	277M	0.99
Routing [192]	223M	0.99
Longformer [177]	102M	0.99
Sparse [175]	95M	0.99
Adaptive [211]	209M	0.98
Compressive [173]	227M	**0.97**
Transformer-LS	110M	**0.97**

a test BPC of 0.97, on par with the Compressive Transformer with 2× parameters. Our results are consistently better than Longformer which is trained on longer sequences with 5 stages and 48 GPU memory. In Figure 5.5, we show our model is much more memory and computational efficient than full attention.

5.3.3 ImageNet Classification

We train and evaluate the models on ImageNet-1K with 1.3M images and 1K classes. We use CvT [169] and ViL [174], state-of-the art vision transformer architectures, as the backbones and replace their attention mechanisms with our long-short term

Table 5.8: Test accuracies on ImageNet, ImageNet Real [9], and ImageNet V2 [10] of models trained on ImageNet-1K. **Grey-colored rows are our results**. CvT*-LS denotes our long-short term attention based on the non-official CvT implementation. ViL models with LS suffixes are our long-short term attention based on the official ViL implementation with relative positional bias. We also provide the latency of models tested using batch size 32 on the same V100 GPU. Our improvements over ViL is mainly from a better implementation of the short-term attention.

Model	#Param (M)	Image Size	FLOPs (G)	ImageNet top-1 (%)	Real top-1 (%)	V2 top-1 (%)	Latency (s)
ResNet-50	25	224^2	4.1	76.2	82.5	63.3	-
ResNet-101	45	224^2	7.9	77.4	83.7	65.7	-
ResNet-152	60	224^2	11	78.3	84.1	67.0	-
DeiT-S [195]	22	224^2	4.6	79.8	85.7	68.5	-
DeiT-B [195]	86	224^2	17.6	81.8	86.7	70.9	-
PVT-Medium [168]	44	224^2	6.7	81.2	-	-	-
PVT-Large [168]	61	224^2	9.8	81.7	-	-	-
Swin-S [197]	50	224^2	8.7	83.2	-	-	-
Swin-B [197]	88	224^2	15.4	83.5	-	-	0.115
PVTv2-B4 [213]	62.6	224^2	10.1	83.6	-	-	-
PVTv2-B5 [213]	82.0	224^2	11.8	83.8	-	-	-
ViT-B/16 [167]	86	384^2	55.5	77.9	-	-	-
ViT-L/16 [167]	307	384^2	191.1	76.5	-	-	-
DeiT-B [195]	86	384^2	55.5	83.1	-	-	-
Swin-B [197]	88	384^2	47.1	**84.5**	-	-	0.378
CvT-13 [169]	20	224^2	6.7	81.6	86.7	70.4	0.122
CvT-21 [169]	32	224^2	10.1	82.5	87.2	71.3	0.165
CvT*-LS-13	20.3	224^2	4.9	81.9	87.0	70.5	0.083
CvT*-LS-17	23.7	224^2	9.8	82.5	87.2	71.6	-
CvT*-LS-21	32.1	224^2	7.9	82.7	87.5	71.9	0.122
CvT*-LS-21S	30.1	224^2	11.3	**82.9**	87.4	71.7	-
CvT-13 [169]	20	384^2	31.9	83.0	87.9	71.9	-
CvT-21 [169]	32	384^2	45.0	83.3	87.7	71.9	-
CvT*-LS-21	32.1	384^2	23.9	83.2	88.0	72.5	-
CvT*-LS-21	32.1	448^2	34.2	**83.6**	88.2	72.9	-
ViL-Small [177]	24.6	224^2	4.9	82.4	-	-	-
ViL-Medium [177]	39.7	224^2	8.7	83.5	-	-	0.106
ViL-Base [177]	55.7	224^2	13.4	83.7	-	-	0.164
ViL-LS-Medium	39.8	224^2	8.7	83.8	-	-	0.075
ViL-LS-Base	55.8	224^2	13.4	**84.1**	-	-	0.113
ViL-LS-Medium	39.9	384^2	28.7	**84.4**	-	-	0.271

attention, denoted as CvT*-LS and ViL-`size`-LS in Table 5.8. CvT uses overlapping convolutions to extract dense patch embeddings from the input images and feature maps, resulting in a long sequence length in the early stages (e.g., $56 \times 56 = 3136$ patches for images with 224^2 pixels). For ViL, our sliding window uses the same group size w, but each token attends to at most $2w \times 2w$ (rounding when necessary) tokens inside the window, instead of $3w \times 3w$ as ViL, which allows adding our dynamic projection without increasing the FLOPs. We set $r = 8$ for the dynamic projections for both ViL-LS-Medium and ViL-LS-Base. Note that, our efficient attention mechanism does not depend on the particular architecture, and it can be applied to other vision transformers [e.g., 167, 168, 195].

Classification Results. The results are shown in the Table 5.8, where we also list test accuracies on ImageNet Real and ImageNet V2. Except for CvT, we compare with the original ViT [167] and the enhanced DeiT [195], PVT [168] that also uses multi-scale stragey, ViL [174] that uses window attention and global tokens to improve the efficiency. Training at high-resolution usually improves the test accuracy of vision transformer. With our long-short term attention, we can easily scale the training to higher resolution, and the performance of CvT*-LS and ViL-LS also improves. Our best model with CvT (CvT*-LS-21 at 448^2) achieves 0.3% higher accuracy than the best reported result of CvT while using the same amount of parameters and 76% of its FLOPs. In CvT architecture, the spatial dimension of feature maps in earlier stages are large, representing more fine-grained details of the image. Similar to training with high-resolution images, the model should also benefit from denser feature maps. With our efficient long-short term attention, we can better utilize these fine-grained feature maps with less concerns about the computational budget. In this way, our CvT*-LS-17 achieves better result than CvT-21 at resolution 224

using fewer parameters and FLOPs, and our CvT*-LS-21S model further improves our CvT*-LS-21 model.

Our ViL-LS-Medium and ViL-LS-Base with long-short term attention improve the accuracies of ViL-Medium and ViL-Base from 83.5 and 83.7 to 83.8 and 84.1 respectively, without an increase in FLOPs. When increasing the resolution for training ViL-LS-Medium from 224^2 to 384^2, the FLOPs increased (approximately) linearly and the accuracy improved by 0.6%, showing our method still benefits greatly from increased resolution while maintaining the linear complexity in practice.

Short-term Attention Suppresses Oversmoothing. By restricting tokens from different segments to attend to different windows, our short-term sparse local attention encourages diversity of the feature representations and helps to alleviate the over-smoothing problem [214] (where all queries extract similar information in deeper layers and the attention mechanism is less important), thus can fully utilize the depth of the network. As in [214], we provide the cosine similarity of patch embeddings of our CvT*-LS-13 and re-implemented CvT-13 (81.1 accuracy) in Figure 5.6. This is one of the reasons why our efficient attention mechanism can get even better results than the full attention CvT model in the same setting.

5.3.4 Robustness Evaluation on ImageNet-derived Datasets.

As vision models have been widely used in safety-critical applications (e.g. autonomous driving), their robustness is vital. In addition to out-of-distribution robustness (ImageNet-Real and Imageet-v2), we further investigate the robustness of our vision

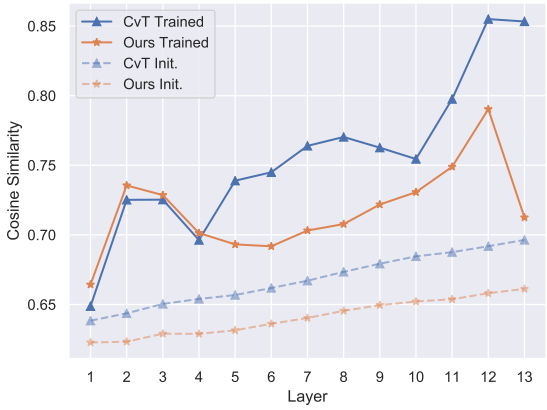

Figure 5.6: Pairwise cosine similarity between patch embeddings at different layers of CvT-13 and CvT*-LS-13, averaged on 50k images of ImageNet validation set. The larger cosine similarities at deeper layer suggest that the feature representation is less diverse.

Table 5.9: Robustness evaluation on various ImageNet datasets. Top-1/Acc.: Top-1 accuracy. mCE: Mean Corruption Error. Mixed-same/Mixed-rand: accuracies on MIXED-SAME/MIXED-RAND subsets.

Model	Params	ImageNet	IN-C [215]	IN-A [216]	IN-R [217]	ImageNet-9 [218]	
	(M)	Top-1	mCE (↓)	Acc.	Acc.	Mixed-same	Mixed-rand
ResNet-50 [88]	25.6	76.2	78.9	6.2	35.3	87.1	81.6
DeiT-S [195]	22.1	79.8	57.1	19.0	41.9	89.1	84.2
CvT-13	20	81.6	59.6	25.4	42.9	90.5	85.7
CvT-21	32	82.5	56.2	**31.1**	42.6	90.5	85.0
CvT*-LS-13	20.3	81.9	58.7	27.0	42.6	90.7	85.6
CvT*-LS-21	32.1	**82.7**	**55.2**	29.3	**45.0**	**91.5**	**85.8**

transformer against common corruption (ImageNet-C), semantic shifts (ImageNet-R), Background dependence (ImageNet-9) and natural adversarial examples (ImageNet-A).

Table 5.10: Corruption Error (CE) on ImageNet-C

Arch.	Noise			Blur				Weather				Digital			
	Gauss.	Shot	Impulse	Defocus	Glass	Motion	Zoom	Snow	Frost	Fog	Bright	Contrast	Elastic	Pixel	JPEG
ResNet-50	34.24	49.25	55.84	56.24	57.04	63.53	63.68	64.02	64.04	64.89	69.25	70.72	73.14	75.29	75.76
DeiT-S	26.93	36.81	36.89	39.38	40.14	43.32	43.80	44.36	45.71	46.90	47.27	48.57	52.15	57.53	62.91
CvT*-LS-13	25.64	36.89	37.06	38.06	43.78	43.78	44.62	45.92	47.77	47.91	49.60	49.66	54.92	57.24	68.72
CvT*-LS-17	25.26	35.06	35.48	37.38	41.37	43.95	44.47	46.05	46.17	46.38	49.08	49.37	54.29	54.54	69.54
CvT*-LS-21	24.28	34.95	35.03	35.93	39.86	40.71	41.27	41.78	44.72	45.24	45.50	47.19	51.84	53.78	67.05

For a fair comparison, we choose models with similar number of parameters. We select two representative models, including the CNN-based model (ResNet) and the transformer based model (DeiT). We give detailed results on all types of image transforms on

Table 5.11: Robustness evaluation on ImageNet-9. We report Top-1 Accuracy.

Model	Params (M)	ImageNet (%)	ImageNet-9 [218](%)		
			Original	Mixed-same	Mixed-rand
ResNet-50 [88]	25.6	76.2	94.9	87.1	81.6
DeiT-S [195]	22.1	79.8	97.1	89.1	84.2
CvT*-LS-13	20.3	81.9	97.0	90.7	85.6
CvT*-LS-21	32.1	**82.7**	**97.2**	**91.5**	**85.8**

ImageNet-C in Table 5.10. We give a brief overview of the ImageNet robustness benchmarks as follows:

- **ImageNet-C**. ImageNet-C refers to the common corruption dataset. It consists of 15 types of algorithmically common corruptions from noise, blur, weather, and digital categories. Each type contains five levels of severity. In Table 4, we report the normalized mean corruption error (mCE) defined in Hendrycks and Dietterich [215]. In Table 5.10, we report the corruption error among different types. In both tables, the lower value means higher robustness.

- **ImageNet-A**. ImageNet-A is the natural adversarial example dataset. It contains naturally collected images from online that mislead the ImageNet classifiers. It contains 7,500 adversarially filtered images. We use accuracy as our evaluation metric. The higher accuracy refers to better robustness.

- **ImageNet-R**. ImageNet-R (**R**endition) aims to evaluate the model generalization performance on out-of-distribution data. It contains renditions of 200 ImageNet classes (e.g. cartoons, graffiti, embroidery). We use accuracy as the evaluation metric.

- **ImageNet-9**. ImageNet-9 aims to evaluate the model background robustness. It designs to measure the extent of the model relying on the image background. Following

the standard setting [218], we evaluate the two categories, including MIXED-SAME and MIXED-RAND. MIXED-SAME refers to replace the background of the selected image with a random background of the same class by GrabCut [218]; MIXED-RAND refers to replace the image background with a random background of the random class.

From table 5.9, we find that our method achieves significant improvement compared to CNN-based network (ResNet). For instance, our method improves the accuracy by 23.6%, 22.1%, 9.7% compared to ResNet on ImageNet-C, ImageNet-A, and ImageNet-R, respectively. For ImageNet-9, our method also achieves favorable improvement by 4.3% on average (Mixed-same and Mixed-rand). It indicates that our method is insensitive to background changes. We guess the potential reasons for these improvements are (1) the attention mechanism and (2) the strong data augmentation strategies during the training for vision transformer [167, 195]. The first design helps the model focus more on the global context of the image as each patch could attend to the whole image areas. It reduces the local texture bias of CNN. The latter design increases the diversity of the training data to improve model's generalization ability. Compared to DeiT, we also surprisingly find that our method achieves slightly better performance. One plausible explanation is that our long-term attention has a favorable smoothing effect on the noisy representations. Such improvements also indicate that different designs of attention and network architecture can be essential to improve the robustness. As the goal of this paper is not to design a robust vision transformer, the robustness is an additional bonus of our method. We believe that our observation opens new directions for designing robust vision Transformers. We leave the in-depth study as an important future work.

The detailed results of ImageNet-C and ImageNet-9 are shown in Table 5.10 and Table 5.11 respectively.

As shown in Table 5.9, we observe that our method significantly outperforms the CNN-based method (ResNet-50).

5.4 Conclusion

In this chapter, we introduced Long-Short Transformer, an efficient transformer for long sequence modeling for both language and vision domain, including both bidirectional and autoregressive models. We design a novel global attention mechanism with linear computational and memory complexity in sequence length based on a dynamic projection. We identify the scale mismatch issue and propose the DualLN technique to eliminate the mismatch at initialization and more effectively aggregate the local and global attentions. We demonstrate that our method obtains the state-of-the-art results on the Long Range Arena, char-level language modeling and ImageNet classification. We look forward to extending our methods to more domains, including document QA, object detection and semantic segmentation on high-resolution images.

Bibliography

[1] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*, 2019.

[2] Xiaodong Liu, Pengcheng He, Weizhu Chen, and Jianfeng Gao. Multi-task deep neural networks for natural language understanding. In *ACL*, 2019.

[3] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V Le. Xlnet: Generalized autoregressive pretraining for language understanding. In *NeurIPS*, 2019.

[4] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[5] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. *ICLR*, 2020.

[6] Yann N Dauphin and Samuel Schoenholz. Metainit: Initializing learning by learning to initialize. In *NeurIPS*, pages 12645–12657, 2019.

[7] Yunyang Xiong, Zhanpeng Zeng, Rudrasis Chakraborty, Mingxing Tan, Glenn Fung, Yin Li, and Vikas Singh. Nyströmformer: A nyström-based algorithm for approximating self-attention. *AAAI*, 2021.

[8] Yi Tay, Mostafa Dehghani, Samira Abnar, Yikang Shen, Dara Bahri, Philip Pham, Jinfeng Rao, Liu Yang, Sebastian Ruder, and Donald Metzler. Long Range Arena: A benchmark for efficient transformers. In *ICLR*, 2021.

[9] Lucas Beyer, Olivier J Hénaff, Alexander Kolesnikov, Xiaohua Zhai, and Aäron van den Oord. Are we done with imagenet? *arXiv preprint arXiv:2006.07159*, 2020.

[10] Benjamin Recht, Rebecca Roelofs, Ludwig Schmidt, and Vaishaal Shankar. Do imagenet classifiers generalize to imagenet? In *International Conference on Machine Learning*, pages 5389–5400. PMLR, 2019.

[11] Romal Thoppilan, Daniel De Freitas, Jamie Hall, Noam M. Shazeer, Apoorv Kulshreshtha, Heng-Tze Cheng, Alicia Jin, Taylor Bos, Leslie Baker, Yu Du, Yaguang Li, Hongrae Lee, Huaixiu Zheng, Amin Ghafouri, Marcelo Menegali, Yanping Huang, Maxim Krikun, Dmitry Lepikhin, James Qin, Dehao Chen, Yuanzhong Xu, Zhifeng Chen, Adam Roberts, Maarten Bosma, Yanqi Zhou, Chung-Ching Chang, I. A. Krivokon, Willard James Rusch, Marc Pickett, Kathleen S. Meier-Hellstern, Meredith Ringel Morris, Tulsee Doshi, Renelito Delos Santos, Toju Duke, Johnny Hartz Søraker, Ben Zevenbergen, Vinodkumar Prabhakaran, Mark Diaz, Ben Hutchinson, Kristen Olson, Alejandra Molina, Erin Hoffman-John, Josh Lee, Lora Aroyo, Ravindran Rajakumar, Alena Butryna, Matthew Lamm, V. O. Kuzmina, Joseph Fenton, Aaron Cohen, Rachel Bernstein, Ray Kurzweil, Blaise Aguera-Arcas, Claire Cui, Marian Croak, Ed Chi, and Quoc Le. Lamda: Language models for dialog applications. *arXiv preprint arXiv:1906.02243*, 2022.

[12] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in nlp. *arXiv preprint arXiv:1906.02243*, 2019.

[13] Training a single ai model can emit as much carbon as five cars in their lifetimes. https://www.technologyreview.com/2019/06/06/239031/training-a-single-ai-model-can-emit-as-much-carbon-as-five-cars-in-th 2019-06-06.

[14] Soham De and Sam Smith. Batch normalization biases residual blocks towards the identity function in deep networks. *NeurIPS*, 2020.

[15] Liyuan Liu, Xiaodong Liu, Jianfeng Gao, Weizhu Chen, and Jiawei Han. Understanding the difficulty of training transformers. *EMNLP*, 2020.

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *ICCV*, pages 1026–1034, 2015.

[17] Wayne Xiong, Jasha Droppo, Xuedong Huang, Frank Seide, Mike Seltzer, Andreas Stolcke, Dong Yu, and Geoffrey Zweig. Achieving human parity in conversational speech recognition. *arXiv preprint arXiv:1610.05256*, 2016.

[18] Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. Deberta: Decoding-enhanced bert with disentangled attention. In *ICLR*, 2021.

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[20] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, pages 448–456. PMLR, 2015.

[21] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[22] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *ICML*, pages 2048–2057. PMLR, 2015.

[23] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *NIPS*, 30, 2017.

[24] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *ICML*, pages 1139–1147. PMLR, 2013.

[25] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

[26] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[27] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[28] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *NIPS*, 27, 2014.

[29] Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International Conference on Machine Learning*, pages 2256–2265. PMLR, 2015.

[30] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR, 2020.

[31] Xinlei Chen, Haoqi Fan, Ross Girshick, and Kaiming He. Improved baselines with momentum contrastive learning. *arXiv preprint arXiv:2003.04297*, 2020.

[32] Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Guo, Mohammad Gheshlaghi Azar, et al. Bootstrap your own latent-a new approach to self-supervised learning. *Advances in Neural Information Processing Systems*, 33:21271–21284, 2020.

[33] Alexei Baevski, Wei-Ning Hsu, Qiantong Xu, Arun Babu, Jiatao Gu, and Michael Auli. Data2vec: A general framework for self-supervised learning in speech, vision and language. *arXiv preprint arXiv:2202.03555*, 2022.

[34] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, Yonghui Wu, et al. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *Advances in neural information processing systems*, 32, 2019.

[35] Jeff Dean. Introducing pathways: A nextgeneration ai architecture. *Google Blog*, 2021.

[36] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[37] Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, et al. Efficient large-scale language model training on gpu clusters using megatron-lm. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–15, 2021.

[38] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Baindoor Rao, Parker Barnes, Yi Tay, Noam M. Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Benton C. Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier García, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Oliveira Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathleen S. Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. Palm: Scaling language modeling with pathways. *ArXiv*, abs/2204.02311, 2022.

[39] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

[40] Training a single ai model can emit as much carbon as five cars in their lifetimes. https://en.wikipedia.org/wiki/Tay_(bot). 2019-06-06.

[41] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[42] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *ICLR*, 2018.

[43] Dimitris Tsipras, Shibani Santurkar, Logan Engstrom, Alexander Turner, and Aleksander Madry. Robustness may be at odds with accuracy. *arXiv preprint arXiv:1805.12152*, 2018.

[44] Ekin D Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le. Autoaugment: Learning augmentation policies from data. *arXiv preprint arXiv:1805.09501*, 2018.

[45] Jason Wei and Kai Zou. Eda: Easy data augmentation techniques for boosting performance on text classification tasks. *arXiv preprint arXiv:1901.11196*, 2019.

[46] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Semantically equivalent adversarial rules for debugging nlp models. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2018.

[47] Greg Yang and Edward J Hu. Tensor programs iv: Feature learning in infinite-width neural networks. In *ICML*, pages 11727–11737. PMLR, 2021.

[48] Greg Yang, Edward J Hu, Igor Babuschkin, Szymon Sidor, Xiaodong Liu, David Farhi, Nick Ryder, Jakub Pachocki, Weizhu Chen, and Jianfeng Gao. Tensor programs v: Tuning large neural networks via zero-shot hyperparameter transfer. *NeurIPS*, 2021.

[49] Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tieyan Liu. On layer normalization in the transformer architecture. In *International Conference on Machine Learning*, pages 10524–10533. PMLR, 2020.

[50] Ali Shafahi, W Ronny Huang, Mahyar Najibi, Octavian Suciu, Christoph Studer, Tudor Dumitras, and Tom Goldstein. Poison frogs! targeted clean-label poisoning attacks on neural networks. *arXiv preprint arXiv:1804.00792*, 2018.

[51] Chen Zhu, Yu Cheng, Zhe Gan, Siqi Sun, Thomas Goldstein, and Jingjing Liu. Freelb: Enhanced adversarial training for language understanding. *arXiv preprint arXiv:1909.11764*, 2019.

[52] Zhe Gan, Yen-Chun Chen, Linjie Li, Chen Zhu, Yu Cheng, and Jingjing Liu. Large-scale adversarial training for vision-and-language representation learning. *NeurIPS*, 33:6616–6628, 2020.

[53] Kezhi Kong, Guohao Li, Mucong Ding, Zuxuan Wu, Chen Zhu, Bernard Ghanem, Gavin Taylor, and Tom Goldstein. Flag: Adversarial data augmentation for graph neural networks. *CVPR*, 2022.

[54] Chen Zhu, Renkun Ni, Zheng Xu, Kezhi Kong, W Ronny Huang, and Tom Goldstein. Gradinit: Learning to initialize neural networks for stable and efficient training. *Advances in Neural Information Processing Systems*, 34, 2021.

[55] Chen Zhu, W Ronny Huang, Hengduo Li, Gavin Taylor, Christoph Studer, and Tom Goldstein. Transferable clean-label poisoning attacks on deep neural nets. In *International Conference on Machine Learning*, pages 7614–7623, 2019.

[56] Neehar Peri, Neal Gupta, W Ronny Huang, Liam Fowl, Chen Zhu, Soheil Feizi, Tom Goldstein, and John P Dickerson. Deep k-nn defense against clean-label data poisoning attacks. In *European Conference on Computer Vision*, pages 55–70. Springer, 2020.

[57] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia conference on computer and communications security*, pages 506–519. ACM, 2017.

[58] Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. *arXiv preprint arXiv:1703.04730*, 2017.

[59] Shike Mei and Xiaojin Zhu. Using machine teaching to identify optimal training-set attacks on machine learners. In *AAAI*, pages 2871–2877, 2015.

[60] Battista Biggio, Blaine Nelson, and Pavel Laskov. Poisoning attacks against support vector machines. *arXiv preprint arXiv:1206.6389*, 2012.

[61] Octavian Suciu, Radu Mărginean, Yiğitcan Kaya, Hal Daumé III, and Tudor Dumitraş. When does machine learning fail? generalized transferability for evasion and poisoning attacks. *arXiv preprint arXiv:1803.06975*, 2018.

[62] Saeed Mahloujifar, Dimitrios I. Diochnos, and Mohammad Mahmoody. Learning under p-tampering attacks. *CoRR*, abs/1711.03707, 2017. URL http://arxiv.org/abs/1711.03707.

[63] Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Šrndić, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. Evasion attacks against machine learning at test time. In *Joint European conference on machine learning and knowledge discovery in databases*, pages 387–402. Springer, 2013.

[64] Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*, 2015. URL http://arxiv.org/abs/1412.6572.

[65] Yingqi Liu, Shiqing Ma, Yousra Aafer, Wen-Chuan Lee, Juan Zhai, Weihang Wang, and Xiangyu Zhang. Trojaning attack on neural networks. 2017.

[66] Xinyun Chen, Chang Liu, Bo Li, Kimberly Lu, and Dawn Song. Targeted Backdoor Attacks on Deep Learning Systems Using Data Poisoning. *arXiv preprint arXiv:1712.05526*, 2017. URL http://arxiv.org/abs/1712.05526.

[67] Alexander Turner, Dimitris Tsipras, and Aleksander Madry. Clean-label backdoor attacks, 2019. URL https://people.csail.mit.edu/madry/lab/cleanlabel.pdf.

[68] Jacob Steinhardt, Pang Wei Koh, and Percy Liang. Certified defenses for data poisoning attacks. *CoRR*, abs/1706.03691, 2017. URL http://arxiv.org/abs/1706.03691.

[69] Florian Tramèr, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. The space of transferable adversarial examples. *arXiv preprint arXiv:1704.03453*, 2017.

[70] Yanpei Liu, Xinyun Chen, Chang Liu, and Dawn Song. Delving into transferable adversarial examples and black-box attacks. *CoRR*, abs/1611.02770, 2016. URL http://arxiv.org/abs/1611.02770.

[71] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*, 2016.

[72] Tom Goldstein, Christoph Studer, and Richard Baraniuk. A field guide to forward-backward splitting with a fasta implementation. *arXiv preprint arXiv:1411.3406*, 2014.

[73] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[74] Yinpeng Dong, Fangzhou Liao, Tianyu Pang, Hang Su, Jun Zhu, Xiaolin Hu, and Jianguo Li. Boosting adversarial attacks with momentum. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9185–9193, 2018.

[75] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533*, 2016.

[76] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[77] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[78] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7132–7141, 2018.

[79] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*, pages 5987–5995. IEEE, 2017.

[80] Yunpeng Chen, Jianan Li, Huaxin Xiao, Xiaojie Jin, Shuicheng Yan, and Jiashi Feng. Dual path networks. In *Advances in Neural Information Processing Systems*, pages 4467–4475, 2017.

[81] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. *arXiv preprint arXiv:1801.04381*, 2018.

[82] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[83] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2261–2269. IEEE, 2017.

[84] Pang Wei Koh, Jacob Steinhardt, and Percy Liang. Stronger data poisoning attacks break data sanitization defenses. *arXiv preprint arXiv:1811.00741*, 2018.

[85] Bernhard Schölkopf, John C Platt, John Shawe-Taylor, Alex J Smola, and Robert C Williamson. Estimating the support of a high-dimensional distribution. *Neural computation*, 13(7):1443–1471, 2001.

[86] Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Logan Engstrom, Brandon Tran, and Aleksander Madry. Adversarial examples are not bugs, they are features. In *Advances in Neural Information Processing Systems*, pages 125–136, 2019.

[87] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[88] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[89] Chen Zhu, Yu Cheng, Zhe Gan, Siqi Sun, Tom Goldstein, and Jingjing Liu. Freelb: Enhanced adversarial training for natural language understanding. *ICLR*, 2020.

[90] Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *ICLR*, 2015.

[91] Chaowei Xiao, Ruizhi Deng, Bo Li, Fisher Yu, Mingyan Liu, and Dawn Song. Characterizing adversarial examples based on spatial consistency information for semantic segmentation. In *ECCV*, 2018.

[92] Cihang Xie, Yuxin Wu, Laurens van der Maaten, Alan L. Yuille, and Kaiming He. Feature denoising for improving adversarial robustness. In *CVPR*, 2019.

[93] Takeru Miyato, Andrew M Dai, and Ian Goodfellow. Adversarial training methods for semi-supervised text classification. In *ICLR*, 2017.

[94] Yong Cheng, Lu Jiang, and Wolfgang Macherey. Robust neural machine translation with doubly adversarial inputs. In *ACL*, 2019.

[95] Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. *arXiv preprint arXiv:1802.00420*, 2018.

[96] Chongli Qin, James Martens, Sven Gowal, Dilip Krishnan, Alhussein Fawzi, Soham De, Robert Stanforth, Pushmeet Kohli, et al. Adversarial robustness through local linearization. *arXiv preprint arXiv:1907.02610*, 2019.

[97] A. Shafahi, M. Najibi, A. Ghiasi, Z. Xu, J. Dickerson, C. Studer, L. Davis, G. Taylor, and T. Goldstein. Adversarial Training for Free! In *NeurIPS*, 2019.

[98] Dinghuai Zhang, Tianyuan Zhang, Yiping Lu, Zhanxing Zhu, and Bin Dong. You only propagate once: Accelerating adversarial training via maximal principle. *arXiv preprint arXiv:1905.00877*, 2019.

[99] Robin Jia and Percy Liang. Adversarial examples for evaluating reading comprehension systems. In *EMNLP*, 2017.

[100] Zhengli Zhao, Dheeru Dua, and Sameer Singh. Generating natural adversarial examples. In *ICLR*, 2018.

[101] Yonatan Belinkov and Yonatan Bisk. Synthetic and natural noise both break neural machine translation. In *ICLR*, 2018.

[102] Mohit Iyyer, John Wieting, Kevin Gimpel, and Luke Zettlemoyer. Adversarial example generation with syntactically controlled paraphrase networks. In *NAACL*, 2018.

[103] Takeru Miyato, Shin-ichi Maeda, Masanori Koyama, and Shin Ishii. Virtual adversarial training: A regularization method for supervised and semi-supervised learning. *TPAMI*, 2019.

[104] Javid Ebrahimi, Anyi Rao, Daniel Lowd, and Dejing Dou. HotFlip: White-box adversarial examples for text classification. In *ACL*, 2018.

[105] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Semantically equivalent adversarial rules for debugging NLP models. In *ACL*, 2018.

[106] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint 1910.10683*, 2019.

[107] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *ACL*, 2016.

[108] Patrick L Combettes and Jean-Christophe Pesquet. Proximal splitting methods in signal processing. In *Fixed-point algorithms for inverse problems in science and engineering*. 2011.

[109] Sanghamitra Dutta, Gauri Joshi, Soumyadip Ghosh, Parijat Dube, and Priya Nagpurkar. Slow and stale gradients can win the race: Error-runtime trade-offs in distributed sgd. In *AISTATS*, pages 803–812, 2018.

[110] Jure Sokolic, Raja Giryes, Guillermo Sapiro, and Miguel Rodrigues. Generalization error of invariant classifiers. In *AISTATS*, 2017.

[111] Huan Xu and Shie Mannor. Robustness and generalization. *Machine learning*, 2012.

[112] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *JMLR*, 2014.

[113] Yarin Gal and Zoubin Ghahramani. A theoretically grounded application of dropout in recurrent neural networks. In *NeurIPS*, 2016.

[114] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. In *ICLR*, 2019.

[115] Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

[116] Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. Commonsenseqa: A question answering challenge targeting commonsense knowledge. In *NAACL*, 2019.

[117] Alex Warstadt, Amanpreet Singh, and Samuel R. Bowman. Neural network acceptability judgments. *arXiv preprint 1805.12471*, 2018.

[118] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *EMNLP*, 2013.

[119] William B Dolan and Chris Brockett. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the International Workshop on Paraphrasing*, 2005.

[120] Eneko Agirre, Llu'is M'arquez, and Richard Wicentowski, editors. *Proceedings of the Fourth International Workshop on Semantic Evaluations (SemEval-2007)*. Association for Computational Linguistics, 2007.

[121] Shankar Iyer, Nikhil Dandekar, and Kornl Csernai. First quora dataset release: Question pairs, 2017. URL https://www.quora.com/q/quoradata/First-Quora-Dataset-Release-Question-Pairs.

[122] Adina Williams, Nikita Nangia, and Samuel R. Bowman. A broad-coverage challenge corpus for sentence understanding through inference. In *NAACL*, 2018.

[123] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. SQuAD: 100,000+ questions for machine comprehension of text. In *EMNLP*, 2016.

[124] Ido Dagan, Oren Glickman, and Bernardo Magnini. The PASCAL recognising textual entailment challenge. In *Machine learning challenges. evaluating predictive uncertainty, visual object classification, and recognising tectual entailment*. Springer, 2006.

[125] Roy Bar Haim, Ido Dagan, Bill Dolan, Lisa Ferro, Danilo Giampiccolo, Bernardo Magnini, and Idan Szpektor. The second PASCAL recognising textual entailment challenge. 2006.

[126] Danilo Giampiccolo, Bernardo Magnini, Ido Dagan, and Bill Dolan. The third PASCAL recognizing textual entailment challenge. In *Proceedings of the ACL-PASCAL workshop on textual entailment and paraphrasing*, 2007.

[127] Luisa Bentivogli, Ido Dagan, Hoa Trang Dang, Danilo Giampiccolo, and Bernardo Magnini. The fifth PASCAL recognizing textual entailment challenge. 2009.

[128] Hector J Levesque, Ernest Davis, and Leora Morgenstern. The Winograd schema challenge. In *AAAI Spring Symposium: Logical Formalizations of Commonsense Reasoning*, 2011.

[129] Robert Speer, Joshua Chin, and Catherine Havasi. Conceptnet 5.5: An open multilingual graph of general knowledge. In *AAAI*, 2017.

[130] Kai Sun, Dian Yu, Dong Yu, and Claire Cardie. Improving machine reading comprehension with general reading strategies. *arXiv preprint arXiv:1810.13441*, 2018.

[131] Guokun Lai, Qizhe Xie, Hanxiao Liu, Yiming Yang, and Eduard Hovy. Race: Large-scale reading comprehension dataset from examinations. *arXiv preprint arXiv:1704.04683*, 2017.

[132] Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering. *arXiv preprint arXiv:1809.02789*, 2018.

[133] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*, 2010.

[134] Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tieyan Liu. On layer normalization in the transformer architecture. In *ICML*. PMLR, 2020.

[135] Xiao Shi Huang, Felipe Perez, Jimmy Ba, and Maksims Volkovs. Improving transformer optimization through better initialization. In *ICML*, 2020.

[136] Hongyi Zhang, Yann N Dauphin, and Tengyu Ma. Fixup initialization: Residual learning without normalization. In *ICLR*, 2019.

[137] Andrew Brock, Soham De, and Samuel L Smith. Characterizing signal propagation to close the performance gap in unnormalized resnets. *ICLR*, 2021.

[138] Andrew Brock, Soham De, Samuel L. Smith, and Karen Simonyan. High-performance large-scale image recognition without normalization. *arXiv preprint arXiv:2102.06171*, 2021.

[139] Andrew M Saxe, James L McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *ICLR*, 2014.

[140] Dmytro Mishkin and Jiri Matas. All you need is a good init. *ICLR*, 2016.

[141] Mert Gurbuzbalaban and Yuanhan Hu. Fractional moment-preserving initialization schemes for training deep neural networks. In *International Conference on Artificial Intelligence and Statistics*, pages 2233–2241. PMLR, 2021.

[142] Charles H Martin and Michael W Mahoney. Traditional and heavy-tailed self regularization in neural network models. *arXiv preprint arXiv:1901.08276*, 2019.

[143] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[144] Thomas Bachlechner, Bodhisattwa Prasad Majumder, Huanru Henry Mao, Garrison W Cottrell, and Julian McAuley. Rezero is all you need: Fast convergence at large depth. *arXiv preprint arXiv:2003.04887*, 2020.

[145] Jingzhao Zhang, Sai Praneeth Karimireddy, Andreas Veit, Seungyeon Kim, Sashank J Reddi, Sanjiv Kumar, and Suvrit Sra. Why are adaptive methods good for attention models? *NeurIPS*, 2020.

[146] Sanjeev Arora, Zhiyuan Li, and Kaifeng Lyu. Theoretical analysis of auto rate-tuning by batch normalization. In *International Conference on Learning Representations*, 2019.

[147] Ruosi Wan, Zhanxing Zhu, Xiangyu Zhang, and Jian Sun. Spherical motion dynamics of deep neural networks with batch normalization and weight decay. *arXiv preprint arXiv:2006.08419*, 2020.

[148] Lukas Balles and Philipp Hennig. Dissecting adam: The sign, magnitude and variance of stochastic gradients. In *ICML*, pages 404–413, 2018.

[149] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[150] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR*, 2009.

[151] Mauro Cettolo, Jan Niehues, Sebastian Stüker, Luisa Bentivogli, and Marcello Federico. Report on the 11th iwslt evaluation campaign, iwslt 2014. In *IWSLT*, volume 57, 2014.

[152] Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. fairseq: A fast, extensible toolkit for sequence modeling. In *NAACL-HLT (Demonstrations)*, 2019.

[153] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[154] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

[155] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

[156] Terrance DeVries and Graham W Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017.

[157] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

[158] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *ICLR*, 2018.

[159] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2018.

[160] Chen Zhu, Yu Cheng, Zhe Gan, Furong Huang, Jingjing Liu, and Tom Goldstein. Maxva: Fast adaptation of step sizes by maximizing observed variance of gradients. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 628–643. Springer, 2021.

[161] Jeremy Bernstein, Yu-Xiang Wang, Kamyar Azizzadenesheli, and Animashree Anandkumar. signsgd: Compressed optimisation for non-convex problems. In *ICML*, 2018.

[162] Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. On the variance of the adaptive learning rate and beyond. *ICLR*, 2020.

[163] Chen Zhu, Wei Ping, Chaowei Xiao, Mohammad Shoeybi, Tom Goldstein, Anima Anandkumar, and Bryan Catanzaro. Long-short transformer: Efficient transformers for language and vision. *NeurIPS*, 34, 2021.

[164] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NIPS*, volume 30, 2017.

[165] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *NAACL*, 2019.

[166] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI blog*, 1 (8):9, 2019.

[167] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *ICLR*, 2021.

[168] Wenhai Wang, Enze Xie, Xiang Li, Deng-Ping Fan, Kaitao Song, Ding Liang, Tong Lu, Ping Luo, and Ling Shao. Pyramid vision transformer: A versatile backbone for dense prediction without convolutions. *arXiv preprint arXiv:2102.12122*, 2021.

[169] Haiping Wu, Bin Xiao, Noel Codella, Mengchen Liu, Xiyang Dai, Lu Yuan, and Lei Zhang. Cvt: Introducing convolutions to vision transformers. *arXiv preprint arXiv:2103.15808*, 2021.

[170] Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, et al. Natural questions: a benchmark for question answering research. *TACL*, 7:453–466, 2019.

[171] Raghavendra Pappagari, Piotr Zelasko, Jesús Villalba, Yishay Carmiel, and Najim Dehak. Hierarchical transformers for long document classification. In *2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, pages 838–844. IEEE, 2019.

[172] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. In *ACL*, 2019.

[173] Jack W Rae, Anna Potapenko, Siddhant M Jayakumar, and Timothy P Lillicrap. Compressive Transformers for long-range sequence modelling. In *ICLR*, 2020.

[174] Pengchuan Zhang, Xiyang Dai, Jianwei Yang, Bin Xiao, Lu Yuan, Lei Zhang, and Jianfeng Gao. Multi-scale vision longformer: A new vision transformer for high-resolution image encoding. *arXiv preprint arXiv:2103.15358*, 2021.

[175] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.

[176] Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran. Image transformer. In *ICML*, pages 4055–4064, 2018.

[177] Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

[178] Joshua Ainslie, Santiago Ontanon, Chris Alberti, Vaclav Cvicek, Zachary Fisher, Philip Pham, Anirudh Ravula, Sumit Sanghai, Qifan Wang, and Li Yang. Etc: Encoding long and structured inputs in transformers. In *EMNLP*, pages 268–284, 2020.

[179] Manzil Zaheer, Guru Guruganesh, Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. Big Bird: Transformers for longer sequences. In *NeurIPS*, 2020.

[180] Sinong Wang, Belinda Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2020.

[181] Yi Tay, Dara Bahri, Donald Metzler, Da-Cheng Juan, Zhe Zhao, and Che Zheng. Synthesizer: Rethinking self-attention in transformer models. In *ICML*, 2021.

[182] Pengchuan Zhang, Xiyang Dai, Jianwei Yang, Bin Xiao, Lu Yuan, Lei Zhang, and Jianfeng Gao. Multi-scale vision longformer: A new vision transformer for high-resolution image encoding. *arXiv preprint arXiv:2103.15358*, 2021.

[183] Ali Razavi, Aaron van den Oord, and Oriol Vinyals. Generating diverse high-fidelity images with vq-vae-2. *arXiv preprint arXiv:1906.00446*, 2019.

[184] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation. *arXiv preprint arXiv:2102.12092*, 2021.

[185] Jonathan Ho, Nal Kalchbrenner, Dirk Weissenborn, and Tim Salimans. Axial attention in multidimensional transformers. *arXiv preprint arXiv:1912.12180*, 2019.

[186] Jiezhong Qiu, Hao Ma, Omer Levy, Scott Wen-tau Yih, Sinong Wang, and Jie Tang. Blockwise self-attention for long document understanding. *arXiv preprint arXiv:1911.02972*, 2019.

[187] Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In *ICML*, 2019.

[188] Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, et al. Rethinking attention with performers. *ICLR*, 2021.

[189] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *ICML*, 2020.

[190] Hao Peng, Nikolaos Pappas, Dani Yogatama, Roy Schwartz, Noah A Smith, and Lingpeng Kong. Random feature attention. *ICLR*, 2021.

[191] Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. In *ICLR*, 2020.

[192] Aurko Roy, Mohammad Saffar, Ashish Vaswani, and David Grangier. Efficient content-based sparse attention with routing transformers. *TACL*, 9:53–68, 2021.

[193] Yi Tay, Dara Bahri, Liu Yang, Donald Metzler, and Da-Cheng Juan. Sparse sinkhorn attention. In *ICML*. PMLR, 2020.

[194] Zhanghao Wu, Zhijian Liu, Ji Lin, Yujun Lin, and Song Han. Lite transformer with long-short range attention. *arXiv preprint arXiv:2004.11886*, 2020.

[195] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. *arXiv preprint arXiv:2012.12877*, 2020.

[196] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009.

[197] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. *arXiv preprint arXiv:2103.14030*, 2021.

[198] Li Yuan, Yunpeng Chen, Tao Wang, Weihao Yu, Yujun Shi, Zihang Jiang, Francis EH Tay, Jiashi Feng, and Shuicheng Yan. Tokens-to-token vit: Training vision transformers from scratch on imagenet. *arXiv preprint arXiv:2101.11986*, 2021.

[199] Ashish Vaswani, Prajit Ramachandran, Aravind Srinivas, Niki Parmar, Blake Hechtman, and Jonathon Shlens. Scaling local self-attention for parameter efficient visual backbones. *arXiv preprint arXiv:2103.12731*, 2021.

[200] Andrew Jaegle, Felix Gimeno, Andrew Brock, Andrew Zisserman, Oriol Vinyals, and Joao Carreira. Perceiver: General perception with iterative attention. *arXiv preprint arXiv:2103.03206*, 2021.

[201] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *ICML*. PMLR, 2020.

[202] Hao Peng, Nikolaos Pappas, Dani Yogatama, Roy Schwartz, Noah A Smith, and Lingpeng Kong. Random feature attention. *ICLR*, 2021.

[203] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*, 2010.

[204] fvcore: Flop counter for pytorch models. https://github.com/facebookresearch/fvcore/blob/master/docs/flop_count.md, 2021.

[205] Yi Tay, Mostafa Dehghani, Vamsi Aribandi, Jai Gupta, Philip Pham, Zhen Qin, Dara Bahri, Da-Cheng Juan, and Donald Metzler. Omninet: Omnidirectional representations from transformers. *ICML*, 2021.

[206] Nikita Nangia and Samuel Bowman. Listops: A diagnostic dataset for latent tree learning. In *NAACL: Student Research Workshop*, pages 92–99, 2018.

[207] Andrew Maas, Raymond E Daly, Peter T Pham, Dan Huang, Andrew Y Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *ACL*, 2011.

[208] Dragomir R Radev, Pradeep Muthukrishnan, Vahed Qazvinian, and Amjad Abu-Jbara. The acl anthology network corpus. *Language Resources and Evaluation*, 47 (4):919–944, 2013.

[209] Matt Mahoney. Large text compression benchmark. *URL http://mattmahoney.net/dc/textdata*, 6, 2009.

[210] Rami Al-Rfou, Dokook Choe, Noah Constant, Mandy Guo, and Llion Jones. Character-level language modeling with deeper self-attention. In *AAAI*, volume 33, pages 3159–3166, 2019.

[211] Sainbayar Sukhbaatar, Edouard Grave, Piotr Bojanowski, and Armand Joulin. Adaptive attention span in transformers. *ACL*, 2019.

[212] Zihao Ye, Qipeng Guo, Quan Gan, Xipeng Qiu, and Zheng Zhang. Bp-transformer: Modelling long-range context via binary partitioning. *arXiv preprint arXiv:1911.04070*, 2019.

[213] Wenhai Wang, Enze Xie, Xiang Li, Deng-Ping Fan, Kaitao Song, Ding Liang, Tong Lu, Ping Luo, and Ling Shao. Pvtv2: Improved baselines with pyramid vision transformer. 2021.

[214] Chengyue Gong, Dilin Wang, Meng Li, Vikas Chandra, and Qiang Liu. Improve vision transformers training by suppressing over-smoothing. *arXiv preprint arXiv:2104.12753*, 2021.

[215] Dan Hendrycks and Thomas Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. In *International Conference on Learning Representations*, 2019.

[216] Dan Hendrycks, Kevin Zhao, Steven Basart, Jacob Steinhardt, and Dawn Song. Natural adversarial examples. *arXiv preprint arXiv:1907.07174*, 2019.

[217] Dan Hendrycks, Steven Basart, Norman Mu, Saurav Kadavath, Frank Wang, Evan Dorundo, Rahul Desai, Tyler Zhu, Samyak Parajuli, Mike Guo, et al. The many faces of robustness: A critical analysis of out-of-distribution generalization. *arXiv preprint arXiv:2006.16241*, 2020.

[218] Kai Xiao, Logan Engstrom, Andrew Ilyas, and Aleksander Madry. Noise or signal: The role of image backgrounds in object recognition. *ArXiv preprint arXiv:2006.09994*, 2020.