Understanding and Improving Efficiency in Training of Deep Neural Networks

Thesis by
Jiawei Zhao

In Partial Fulfillment of the Requirements for the
Degree of
Doctor of Philosophy

CALIFORNIA INSTITUTE OF TECHNOLOGY
Pasadena, California

2025
Defended Septemeber 3, 2024

Jiawei Zhao
ORCID: 0000-0002-5726-6040

ACKNOWLEDGEMENTS

I am deeply grateful to my advisor, Anima Anandkumar, for your trust and the freedom to pursue my research interests. Thank you for bringing me into this field and providing me guidance since my undergraduate years.

Thank you to my committee members, Adam Wierman, Beidi Chen, Eric Mazumdar, and Yuandong Tian, for your valuable feedback and guidance throughout my research.

I am grateful to all my collaborators—Beidi Chen, Jeremy Bernstein, Florian Schäfer, Yuandong Tian, Bill Dally, Brian Zimmer, Brucek Khailany, Jean Kossaifi, Kamyar Azizzadenesheli, Ming-Yu Liu, Rangharajan Venkatesan, Robert Joseph George, Steve Dai, Yifei Zhang, Yisong Yue, Atlas Wang, Zhenyu Zhang, and Zongyi Li. Working with you has been an enriching experience.

A special thanks to my friends, Yameng Zhang, Zongyi Li, Yujia Huang, Julius Berner, Angela Gao, Yuanyuan Shi, and many others, for making my time at Caltech memorable and enjoyable. The moments we shared will always be cherished.

I also thank Laura Flower Kim and Daniel Yoder from ISP for their support, which made my international student journey smooth.

Finally, my deepest thanks go to my parents, Hongjing Zhao and Xiaoqing Guo, for your unwavering love and support. This thesis is dedicated to you.

ABSTRACT

As deep neural networks (DNNs) continue to power advancements in fields like computer vision and natural language processing, their growing complexity has led to significant challenges in training efficiency, particularly in large language models (LLMs). These challenges include memory limitations, energy consumption, and bandwidth constraints during training.

In this thesis, I will address these challenges by understanding the training dynamics of DNNs and proposing hardware-efficient learning algorithms to improve training efficiency. I will begin by mitigating memory limitations in LLM training. Training large models like LLMs requires substantial memory not only for the parameters but also for gradients and optimizer states, which can exceed the capacity of standard hardware. To address this issue, I propose GaLore, a memory-efficient training algorithm that reduces the memory footprint of training LLMs by up to 65.5% while maintaining performance. In addition, I will introduce InRank, an incremental low-rank learning algorithm that further reduces memory usage by incrementally augmenting matrix rank.

Next, I will focus on reducing energy consumption during training. Training large models like LLMs requires significant energy, which can have a significant environmental impact. To address this issue, I propose LNS-Madam, a low-precision training algorithm that uses the logarithmic number system (LNS) to reduce the energy consumption of training without sacrificing accuracy. LNS-Madam can achieve up to 90% energy savings compared to a baseline model trained with full precision.

Finally, I will address bandwidth limitations in distributed training. Training large models like LLMs often requires distributed training across multiple devices to reduce training time. However, distributed training can be limited by the bandwidth of the network connecting the devices, which can lead to communication bottlenecks that slow down training. To address this issue, I will introduce signSGD with Majority Vote, a communication-efficient training algorithm that reduces the communication overhead of distributed training.

PUBLISHED CONTENT AND CONTRIBUTIONS

Zhao, Jiawei, Zhenyu Zhang, et al. (2025). "GaLore: Memory-Efficient LLM Training by Gradient Low-Rank Projection". In: *Proceedings of the 42nd International Conference on Machine Learning (ICML)*. URL: https://arxiv.org/abs/2403.03507.
J.Z. participated in the conception of the project, formulated and implemented the method, conducted experiments and analyzed results, and participated in the writing of the manuscript.

George, Robert Joseph et al. (2024). *Incremental Spatial and Spectral Learning of Neural Operators for Solving Large-Scale PDEs*. arXiv: 2211.15188 [cs.LG].
J.Z. participated in the conception of the project, formulated and implemented the method, conducted experiments and analyzed results, and participated in the writing of the manuscript.

Zhao, Jiawei, Yifei Zhang, et al. (2023). "InRank: Incremental Low-Rank Learning". In: *ES-FoMo Workshop at International Conference on Machine Learning (ICML)*.
J.Z. participated in the conception of the project, formulated and implemented the method, conducted experiments and analyzed results, and participated in the writing of the manuscript.

Zhao, Jiawei, Steve Dai, et al. (2022). "LNS-Madam: Low-Precision Training in Logarithmic Number System Using Multiplicative Weight Update". In: *IEEE Transactions on Computers* 71.12, pp. 3179–3190. DOI: 10.1109/TC.2022.3202747.
J.Z. participated in the conception of the project, formulated and implemented the method, conducted experiments and analyzed results, and participated in the writing of the manuscript.

Zhao, Jiawei, Florian Tobias Schaefer, and Anima Anandkumar (2022). "ZerO Initialization: Initializing Neural Networks with only Zeros and Ones". In: *Transactions on Machine Learning Research (TMLR)*. ISSN: 2835-8856. URL: https://openreview.net/forum?id=1AxQpKmiTc.
J.Z. participated in the conception of the project, formulated and implemented the method, conducted experiments and analyzed results, and participated in the writing of the manuscript.

Bernstein, Jeremy, Jiawei Zhao, Markus Meister, et al. (2020). "Learning compositional functions via multiplicative weight updates". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., pp. 13319–13330. URL: https://proceedings.neurips.cc/paper_files/paper/2020/file/9a32ef65c42085537062753ec435750f-Paper.pdf.
J.Z. participated in the conception of the project, implemented the method, con-

ducted experiments and analyzed results, and participated in the writing of the manuscript.

Bernstein, Jeremy, Jiawei Zhao, Kamyar Azizzadenesheli, et al. (2019). "signSGD with Majority Vote is Communication Efficient and Fault Tolerant". In: *International Conference on Learning Representations (ICLR)*. URL: https://openreview.net/forum?id=BJxhijAcY7.
J.Z. participated in the conception of the project, implemented the method, conducted experiments and analyzed results, and participated in the writing of the manuscript.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

Chapter 1

INTRODUCTION

1.1 The Efficiency Issue in Training of Deep Neural Networks

Deep Neural Networks have been widely used in various applications, such as computer vision and natural language processing. Especially, large language models (LLMs) have achieved remarkable performance in multiple disciplines, including conversational AI and language translation.

However, as these models become larger and more complex, the computational and energy costs associated with their training have escalated significantly. In this thesis, we focus on studying the efficiency issue in training deep neural networks from various methods and perspectives.

Memory Limitation. Training neural networks such as pre-training and fine-tuning LLMs require a large amount of memory. The memory requirements include not only billions of trainable parameters, but also their gradients and optimizer states (e.g., gradient momentum and variance in Adam) that can be larger than parameter storage themselves (Raffel et al., 2020). This brings the need of designing memory-efficient training algorithms that can reduce the memory footprint of training, without sacrificing the model performance.

Energy Consumption. The energy consumption of training deep learning models is becoming an increasingly critical concern, particularly given the significant environmental impact associated with large-scale training (Patterson et al., 2021). To address these challenges, researchers have focused on developing energy-efficient training techniques. One promising approach is low-precision arithmetic, where the bitwidth of computations is reduced to decrease the energy required for each operation. However, low-precision training can lead to a significant loss in model performance, which motivates the need for new training algorithms that can maintain model accuracy while reducing energy consumption.

Bandwidth Limitation. Training large-scale deep learning models often requires distributed training across multiple devices to reduce training time. However, distributed training can be limited by the bandwidth of the network connecting the

devices, which can lead to communication bottlenecks that slow down training. This motivates the need for new training algorithms that can reduce the communication overhead of distributed training.

1.2 Components of Deep Learning towards Efficient Training

In this section, we introduce the components of deep learning that are essential for understanding and improving the efficiency of training deep neural networks.

Initialization. The initialization of neural networks plays a crucial role in training, as it determines the starting point of the optimization process. Proper initialization can help accelerate convergence and improve the final performance of the model. In this thesis, we propose a novel initialization method called ZerO Initialization, which initializes neural networks with only zeros and ones to improve training efficiency.

Optimization in Deep Learning. Optimization algorithms are used to update the parameters of neural networks during training. Traditional optimization algorithms such as Stochastic Gradient Descent (SGD) and its variants such as Adam have been widely used in deep learning. However, these algorithms can be inefficient for training large-scale models due to their memory and computational requirements. In this thesis, we propose a few optimization algorithms that are specifically designed for efficient training of deep neural networks under different training and hardware scenarios.

Large Language Models (LLMs). Large language models (LLMs) have achieved remarkable performance in language tasks, such as conversational AI and language translation. These models are typically equipped with Multilayer perceptrons (MLPs) and self-attention mechanisms, which require large memory and computational resources for training and inference. In this thesis, we propose memory-efficient training algorithms for LLMs that can reduce the memory footprint of training without sacrificing model performance.

1.3 Thesis Structure and Contributions

Chapter 2 and Chapter 3 - Memory-Efficient Training of LLMs. Chapters 2 and 3 are dedicated to improving the memory efficiency of training large language models (LLMs). In Chapter 2, we propose GaLore, Gradient Low-Rank Projection, a novel memory-efficient training algorithm. For the first time, demonstrate that it

is possible to train LLaMA 7B from scratch on a single GPU with 24GB memory (e.g., on NVIDIA RTX 4090), without any costly memory offloading techniques.

Chapter 3 introduces InRank, an incremental low-rank learning algorithm that further reduces memory usage by incrementally augmenting matrix rank. In addition, InRank is capable of identifying intrinsic rank of networks during training.

Chapter 4 - Efficient Initialization of Neural Networks. Chapter 4 introduces ZerO Initialization, a novel initialization method that initializes neural networks with only zeros and ones. We observe that ZerO-initialized networks exhibit low-rank learning trajectories, and converge to sparse and low-rank solutions. Since ZerO does not require any random perturbations, it improves training stability and reproducibility.

Chapter 5 - Low-Precision Training in Logarithmic Number System. Chapter 5 introduces LNS-Madam, a co-designed low-precision training framework, which adopts the logarithmic number system (LNS) and apply multiplicative weight update to train neural networks with reduced precision. In this chapter, we also discuss why considering learning algorithms and number systems together is crucial for achieving effective low-precision training. LNS-Madam can achieve up to 90% energy savings compared to a baseline model trained with full precision.

Chapter 2

GALORE: MEMORY-EFFICIENT LLM TRAINING BY GRADIENT LOW-RANK PROJECTION

2.1 Introduction

Large Language Models (LLMs) have shown impressive performance across multiple disciplines, including conversational AI and language translation. However, pre-training and fine-tuning LLMs require not only a huge amount of computation but is also memory intensive. The memory requirements include not only billions of trainable parameters, but also their gradients and optimizer states (e.g., gradient momentum and variance in Adam) that can be larger than parameter storage themselves (Raffel et al., 2020; Touvron et al., 2023; Chowdhery et al., 2023). For example, pre-training a LLaMA 7B model from scratch with a single batch size requires at least 58 GB memory (14GB for trainable parameters, 42GB for Adam optimizer states and weight gradients, and 2GB for activations). This makes the training not feasible on consumer-level GPUs such as NVIDIA RTX 4090 with 24GB memory.

In addition to engineering and system efforts, such as gradient checkpointing T. Chen et al., 2016, memory offloading Rajbhandari et al., 2020, etc., to achieve faster and more efficient distributed training, researchers also seek to develop various optimization techniques to reduce the memory usage during pre-training and fine-tuning.

Parameter-efficient fine-tuning (PEFT) techniques allow for the efficient adaptation of pre-trained language models (PLMs) to different downstream applications without the need to fine-tune all of the model's parameters (Ding et al., 2022). Among them, the popular Low-Rank Adaptation (LoRA Hu et al. (2022)) *reparameterizes* weight matrix $W \in \mathbb{R}^{m\times n}$ into $W = W_0 + BA$, where W_0 is a frozen full-rank matrix and $B \in \mathbb{R}^{m\times r}$, $A \in \mathbb{R}^{r\times n}$ are additive low-rank adaptors to be learned. Since the rank $r \ll \min(m, n)$, A and B contain fewer number of trainable parameters and thus smaller optimizer states. LoRA has been used extensively to reduce memory usage for fine-tuning in which W_0 is the frozen pre-trained weight. Its variant ReLoRA is also used in pre-training, by periodically updating W_0 using previously learned low-rank adaptors (Lialin et al., 2024).

However, many recent works demonstrate the limitation of such a low-rank reparameterization. For fine-tuning, LoRA is not shown to reach a comparable performance

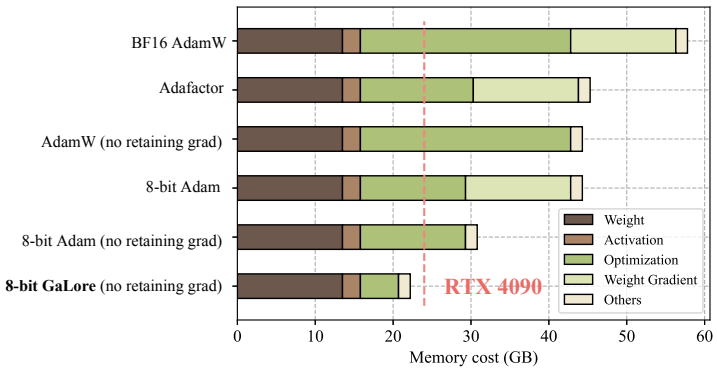

Figure 2.1: Estimated memory consumption of pre-training a LLaMA 7B model with a token batch size of 256 on a single device, without activation checkpointing and memory offloading. Details refer to Section 2.5.

as full-rank fine-tuning Xia, Qin, and Hazan, 2024. For pre-training from scratch, it is shown to require a full-rank model training as a warmup (Lialin et al., 2024), before optimizing in the low-rank subspace. There are two possible reasons: (1) the optimal weight matrices may not be low-rank, and (2) the reparameterization changes the gradient training dynamics.

To address the above challenge, we propose Gradient Low-Rank Projection (**GaLore**), a training strategy that allows *full-parameter* learning but is more *memory-efficient* than common low-rank adaptation methods, such as LoRA. Our key idea is to leverage the slow-changing low-rank structure of the *gradient* $G \in \mathbb{R}^{m \times n}$ of the weight matrix W, rather than trying to approximate the weight matrix itself as low rank.

We first show theoretically that the gradient matrix G becomes low-rank during training. Then, we propose GaLore that computes two projection matrices $P \in \mathbb{R}^{m \times r}$ and $Q \in \mathbb{R}^{n \times r}$ to project the gradient matrix G into a low-rank form $P^\top G Q$. In this case, the memory cost of optimizer states, which rely on component-wise gradient statistics, can be substantially reduced. Occasional updates of P and Q (e.g., every 200 iterations) incur minimal amortized additional computational cost. In practice, this yields up to 30% memory reduction compared to LoRA during pre-training.

We demonstrate that GaLore works well in both LLM pre-training and fine-tuning. When pre-training LLaMA 7B on C4 dataset, 8-bit GaLore, combined with 8-bit optimizers and layer-wise weight updates techniques, achieves comparable performance to its full-rank counterpart, with less than 10% memory cost of optimizer states.

Notably, for pre-training, GaLore keeps low memory throughout the entire training,

without requiring full-rank training warmup like ReLoRA. Thanks to GaLore's memory efficiency, it is possible to train LLaMA 7B from scratch on a single GPU with 24GB memory (e.g., on NVIDIA RTX 4090), without any costly memory offloading techniques (Fig. 2.1).

GaLore is also used to fine-tune pre-trained LLMs on GLUE benchmarks with comparable or better results than existing low-rank methods. When fine-tuning RoBERTa-Base on GLUE tasks with a rank of 4, GaLore achieves an average score of 85.89, outperforming LoRA, which achieves a score of 85.61.

2.2 Related Works

Low-rank adaptation. Hu et al. (2022) proposed Low-Rank Adaptation (LoRA) to fine-tune pre-trained models with low-rank adaptors. This method reduces the memory footprint by maintaining a low-rank weight adaptor for each layer. There are a few variants of LoRA proposed to enhance its performance (Renduchintala, Konuk, and Kuchaiev, 2023; Sheng et al., 2023; L. Zhang et al., 2023; Xia, Qin, and Hazan, 2024), supporting multi-task learning (Yiming Wang et al., 2023), and further reducing the memory footprint (Dettmers, Pagnoni, et al., 2024). Lialin et al. (2024) proposed ReLoRA, a variant of LoRA designed for pre-training, but requires a full-rank training warmup to achieve comparable performance as the standard baseline. Inspired by LoRA, Hao, Cao, and Mou (2024) also suggested that gradients can be compressed in a low-rank subspace, and they proposed to use random projections to compress the gradients. There have also been approaches that propose training networks with low-rank factorized weights from scratch (Kamalakara et al., 2022; H. Wang, Agarwal, et al., 2023; Zhao, Y. Zhang, et al., 2023).

Subspace learning. Recent studies have demonstrated that the learning primarily occurs within a significantly low-dimensional parameter subspace (Gur-Ari, D. A. Roberts, and Dyer, 2018; Larsen et al., 2022). These findings promote a special type of learning called *subspace learning*, where the model weights are optimized within a low-rank subspace. This notion has been widely used in different domains of machine learning, including meta-learning and continual learning (Lee and Choi, 2018; Chaudhry et al., 2020).

Projected gradient descent. GaLore is closely related to the traditional topic of projected gradient descent (PGD) (Y. Chen and Wainwright, 2015; H. Chen, Raskutti, and Yuan, 2019). A key difference is that GaLore considers the specific

gradient form that naturally appears in training multi-layer neural networks (e.g., it is a matrix with specific structures), proving many of its properties (e.g., Lemma 2.3.3, Theorem 2.3.2, and Theorem 2.3.7). In contrast, traditional PGD mostly treats the objective as a general blackbox nonlinear function, and studies the gradients in the vector space only.

Low-rank gradient. Gradient is naturally low-rank during training of neural networks, and this property have been studied in both theory and practice (Zhao, Schaefer, and Anandkumar, 2022; Cosson et al., 2023; G. Yang, Simon, and Bernstein, 2023). It has been applied to reduce communication cost (H. Wang, Sievert, et al., 2018; Vogels, Karimireddy, and Jaggi, 2020), and memory footprint during training (Gooneratne et al., 2020; Huang et al., 2023; Modoranu et al., 2023).

Memory-efficient optimization. There have been some works trying to reduce the memory cost of gradient statistics for adaptive optimization algorithms (Shazeer and Stern, 2018; Anil et al., 2019; Dettmers, Lewis, et al., 2022). Quantization is widely used to reduce the memory cost of optimizer states (Dettmers, Lewis, et al., 2022; Li, J. Chen, and Zhu, 2024). Recent works have also proposed to reduce weight gradient memory by fusing the backward operation with the optimizer update (Lv, Yan, et al., 2023; Lv, Y. Yang, et al., 2023).

2.3 GaLore: Gradient Low-Rank Projection

Background

Regular full-rank training. At time step t, $G_t = -\nabla_W \varphi_t(W_t) \in \mathbb{R}^{m\times n}$ is the backpropagated (negative) gradient matrix. Then the regular pre-training weight update can be written down as follows (η is the learning rate):

$$W_T = W_0 + \eta \sum_{t=0}^{T-1} \tilde{G}_t = W_0 + \eta \sum_{t=0}^{T-1} \rho_t(G_t), \tag{2.1}$$

where $\tilde{G}_t$ is the final processed gradient to be added to the weight matrix and ρ_t is an entry-wise stateful gradient regularizer (e.g., Adam). The state of ρ_t can be memory-intensive. For example, for Adam, we need $M, V \in \mathbb{R}^{m\times n}$ to regularize the gradient G_t into $\tilde{G}_t$:

$$M_t = \beta_1 M_{t-1} + (1-\beta_1) G_t, \tag{2.2}$$

$$V_t = \beta_2 V_{t-1} + (1-\beta_2) G_t^2, \tag{2.3}$$

$$\tilde{G}_t = M_t / \sqrt{V_t + \epsilon}, \tag{2.4}$$

Here G_t^2 and $M_t/\sqrt{V_t+\epsilon}$ means element-wise multiplication and division. η is the learning rate. Together with $W \in \mathbb{R}^{m\times n}$, this takes $3mn$ memory.

Low-rank updates. For a linear layer $W \in \mathbb{R}^{m\times n}$, LoRA and its variants utilize the low-rank structure of the update matrix by introducing a low-rank adaptor AB:

$$W_T = W_0 + B_T A_T, \tag{2.5}$$

where $B \in \mathbb{R}^{m\times r}$ and $A \in \mathbb{R}^{r\times n}$, and $r \ll \min(m,n)$. A and B are the learnable low-rank adaptors and W_0 is a fixed weight matrix (e.g., pre-trained weight).

Low-Rank Property of Weight Gradient

While low-rank updates are proposed to reduce memory usage, it remains an open question whether the weight matrix should be parameterized as low-rank. In many situations, this may not be true. For example, in linear regression $\boldsymbol{y} = W\boldsymbol{x}$, if the optimal W^* is high-rank, then imposing a low-rank assumption on W never leads to the optimal solution, regardless of what optimizers are used.

Surprisingly, while the weight matrices are not necessarily low-rank, the gradient indeed becomes low-rank during the training for certain gradient forms and associated network architectures.

Reversible networks. Obviously, for a general loss function, its gradient can be arbitrary and is not necessarily low rank. Here we study the gradient structure for a general family of nonlinear networks known as "reversible networks" Tian, Yu, et al., 2020, which includes not only simple linear networks but also deep ReLU/polynomial networks:

Definition 2.3.1 (Reversiblity Tian, Yu, et al., 2020) *A network $\mathcal{N}$ that maps input $\boldsymbol{x}$ to output $\boldsymbol{y} = \mathcal{N}(\boldsymbol{x})$ is* reversible*, if there exists $L(\boldsymbol{x}; W)$ so that $\boldsymbol{y} = L(\boldsymbol{x}; W)\boldsymbol{x}$, and the backpropagated gradient $\boldsymbol{g_x}$ satisfies $\boldsymbol{g_x} = L^\top(\boldsymbol{x}; W)\boldsymbol{g_y}$, where $\boldsymbol{g_y}$ is the backpropagated gradient at the output $\boldsymbol{y}$. Here $L(\boldsymbol{x}; W)$ depends on the input $\boldsymbol{x}$ and weight W in the network $\mathcal{N}$.*

Theorem 2.3.2 (Gradient Form of reversible models) *Consider a chained reversible neural network $\mathcal{N}(\boldsymbol{x}) := \mathcal{N}_L(\mathcal{N}_{L-1}(\dots\mathcal{N}_1(\boldsymbol{x})))$ and define $J_l :=$ Jacobian$(\mathcal{N}_L)\dots$ Jacobian$(\mathcal{N}_{l+1})$ and $\boldsymbol{f}_l := \mathcal{N}_l(\dots\mathcal{N}_1(\boldsymbol{x}))$. Then the weight matrix W_l at layer l has gradient G_l in the following form for batch size 1:*

(a) *For ℓ_2-objective $\varphi := \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{f}_L\|_2^2$:*

$$G_l = \left(J_l^\top \boldsymbol{y} - J_l^\top J_l W_l \boldsymbol{f}_{l-1}\right) \boldsymbol{f}_{l-1}^\top, \tag{2.6}$$

(b) *Left $P_{\mathbf{1}}^\perp := I - \frac{1}{K}\mathbf{1}\mathbf{1}^\top$ be the zero-mean PSD projection matrix. For K-way logsoftmax loss $\varphi(\boldsymbol{y}; \boldsymbol{f}_L) := -\log\left(\frac{\exp(\boldsymbol{y}^\top \boldsymbol{f}_L)}{\mathbf{1}^\top \exp(\boldsymbol{f}_L)}\right)$ with small logits $\|P_{\mathbf{1}}^\perp \boldsymbol{f}_L\|_\infty \ll \sqrt{K}$:*

$$G_l = \left(J_l P_{\mathbf{1}}^\perp \boldsymbol{y} - \gamma K^{-1} J_l^\top P_{\mathbf{1}}^\perp J_l W_l \boldsymbol{f}_{l-1}\right) \boldsymbol{f}_{l-1}^\top, \tag{2.7}$$

where $\gamma \approx 1$ and $\boldsymbol{y}$ is a data label with $\boldsymbol{y}^\top \mathbf{1} = 1$.

From the theoretical analysis above, we can see that for batch size N, the gradient G has certain structures: $G = \frac{1}{N}\sum_{i=1}^N (A_i - B_i W C_i)$ for input-dependent matrix A_i, Positive Semi-definite (PSD) matrices B_i and C_i. In the following, we prove that such a gradient will become low-rank during training in certain conditions:

Lemma 2.3.3 (Gradient becomes low-rank during training) *Suppose the gradient follows the parametric form:*

$$G_t = \frac{1}{N}\sum_{i=1}^N (A_i - B_i W_t C_i) \tag{2.8}$$

with constant A_i, PSD matrices B_i and C_i after $t \geq t_0$. We study vanilla SGD weight update: $W_t = W_{t-1} + \eta G_{t-1}$. Let $S := \frac{1}{N}\sum_{i=1}^N C_i \otimes B_i$ and $\lambda_1 < \lambda_2$ its two smallest distinct eigenvalues. Then the stable rank $\mathrm{sr}(G_t)$ satisfies:

$$\mathrm{sr}(G_t) \leq \mathrm{sr}(G_{t_0}^{\parallel}) + \left(\frac{1-\eta\lambda_2}{1-\eta\lambda_1}\right)^{2(t-t_0)} \frac{\|G_0 - G_{t_0}^{\parallel}\|_F^2}{\|G_{t_0}^{\parallel}\|_2^2}, \tag{2.9}$$

where $G_{t_0}^{\parallel}$ is the projection of G_{t_0} onto the minimal eigenspace $\mathcal{V}_1$ of S corresponding to λ_1.

In practice, the constant assumption can approximately hold for some time, in which the second term in Eq. 2.9 goes to zero exponentially and the stable rank of G_t goes down, yielding low-rank gradient G_t. The final stable rank is determined by $\mathrm{sr}(G_{t_0}^{\parallel})$, which is estimated to be low-rank by the following: **Remarks.** The gradient form is justified by Theorem 2.3.2. Intuitively, when N' is small, G_t is a summation of N' rank-1 update and is naturally low rank; on the other hand, when N' becomes larger and closer to n, then the training dynamics has smaller null space $\mathcal{V}_1$, which

also makes G_t low-rank. The full-rank assumption of $\{B_i\}$ is reasonable, e.g., in LLMs, the output dimensions of the networks (i.e., the vocabulary size) is often huge compared to matrix dimensions.

In general if the batch size N is large, then it becomes a bit tricky to characterize the minimal eigenspace $\mathcal{V}_1$ of S. On the other hand, if $\mathcal{V}_1$ has nice structure, then $\mathrm{sr}(G_t)$ can be bounded even further:

Corollary 2.3.4 (Low-rank G_t with special structure of $\mathcal{V}_1$) *If $\mathcal{V}_1(S)$ is 1-dim with decomposable eigenvector $\boldsymbol{v} = \boldsymbol{y} \otimes \boldsymbol{z}$, then $\mathrm{sr}(G_{t_0}^{\parallel}) = 1$ and thus G_t becomes rank-1.*

One rare failure case of Lemma 2.3.3 is when $G_{t_0}^{\parallel}$ is precisely zero, in which $\mathrm{sr}(G_{t_0}^{\parallel})$ becomes undefined. This happens to be true if $t_0 = 0$, i.e., A_i, B_i and C_i are constant throughout the entire training process. Fortunately, for practical training, this does not happen.

Transformers. For Transformers, we can also separately prove that the weight gradient of the lower layer (i.e., *project-up*) weight of feed forward network (FFN) becomes low rank over time, using the JoMA framework Tian, Yiping Wang, et al., 2024.

Gradient Low-rank Projection (GaLore)

Since the gradient G may have a low-rank structure, if we can keep the gradient statistics of a small "core" of gradient G in optimizer states, rather than G itself, then the memory consumption can be reduced substantially. This leads to our proposed GaLore strategy:

Definition 2.3.5 (Gradient Low-rank Projection (GaLore)) *Gradient low-rank projection (**GaLore**) denotes the following gradient update rules (η is the learning rate):*

$$W_T = W_0 + \eta \sum_{t=0}^{T-1} \tilde{G}_t, \quad \tilde{G}_t = P_t \rho_t(P_t^\top G_t Q_t) Q_t^\top, \tag{2.10}$$

where $P_t \in \mathbb{R}^{m\times r}$ and $Q_t \in \mathbb{R}^{n\times r}$ are projection matrices.

Different from LoRA, GaLore *explicitly utilizes the low-rank updates* instead of introducing additional low-rank adaptors and hence does not alter the training dynamics.

In the following, we show that GaLore converges under a similar (but more general) form of gradient update rule. This form corresponds to Eqn. 2.6 but with a larger batch size.

Definition 2.3.6 (L-continuity) *A function $\boldsymbol{h}(W)$ has (Lipschitz) L-continuity, if for any W_1 and W_2, $\|\boldsymbol{h}(W_1) - \boldsymbol{h}(W_2)\|_F \leq L\|W_1 - W_2\|_F$.*

Theorem 2.3.7 (Convergence of GaLore with fixed projections) *Suppose the gradient has the form of Eqn. 2.8 and A_i, B_i and C_i have L_A, L_B and L_C continuity with respect to W and $\|W_t\| \leq D$. Let $R_t := P_t^\top G_t Q_t$, $\hat{B}_{it} := P_t^\top B_i(W_t)P_t$, $\hat{C}_{it} := Q_t^\top C_i(W_t)Q_t$ and $\kappa_t := \frac{1}{N}\sum_i \lambda_{\min}(\hat{B}_{it})\lambda_{\min}(\hat{C}_{it})$. If we choose constant $P_t = P$ and $Q_t = Q$, then GaLore with $\rho_t \equiv 1$ satisfies:*

$$\|R_t\|_F \leq \left[1-\eta(\kappa_{t-1}-L_A-L_BL_CD^2)\right]\|R_{t-1}\|_F. \tag{2.11}$$

As a result, if $\min_t \kappa_t > L_A + L_BL_CD^2$, $R_t \to 0$ and thus GaLore converges with fixed P_t and Q_t.

Setting P and Q. The theorem tells that P and Q should project into the subspaces corresponding to the first few largest eigenvectors of $\hat{B}_{it}$ and $\hat{C}_{it}$ for faster convergence (large κ_t). While all eigenvalues of the positive semidefinite (PSD) matrix B and C are non-negative, some of them can be very small and hinder convergence (i.e., it takes a long time for G_t to become 0). With the projection P and Q, $P^\top B_{it}P$ and $Q^\top C_{it}Q$ only contain the largest eigen subspaces of B and C, improving the convergence of R_t and at the same time, reducing the memory usage.

While it is tricky to obtain the eigenstructure of $\hat{B}_{it}$ and $\hat{C}_{it}$ (they are parts of Jacobian), one way is to instead use the spectrum of G_t via Singular Value Decomposition (SVD):

$$G_t = USV^\top \approx \sum_{i=1}^{r} s_i u_i v_i^\top, \tag{2.12}$$

$$P_t = [u_1, u_2, ..., u_r], \quad Q_t = [v_1, v_2, ..., v_r]. \tag{2.13}$$

Difference between GaLore and LoRA. While both GaLore and LoRA have "low-rank" in their names, they follow very different training trajectories. For example, when $r = \min(m, n)$, GaLore with $\rho_t \equiv 1$ follows the exact training trajectory of the

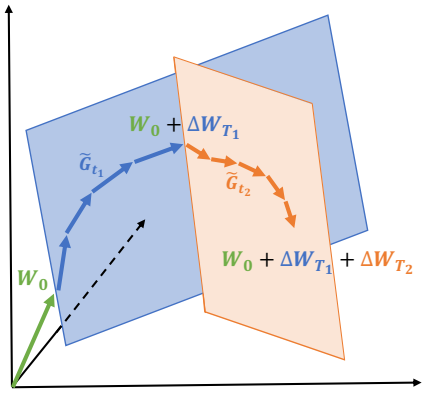

Figure 2.2: Learning through low-rank subspaces ΔW_{T_1} and ΔW_{T_2} using GaLore. For $t_1 \in [0, T_1 - 1]$, W are updated by projected gradients $\tilde{G}_{t_1}$ in a subspace determined by fixed P_{t_1} and Q_{t_1}. After T_1 steps, the subspace is changed by recomputing P_{t_2} and Q_{t_2} for $t_2 \in [T_1, T_2 - 1]$, and the process repeats until convergence.

original model, as $\tilde{G}_t = P_t P_t^\top G_t Q_t Q_t^\top = G_t$. On the other hand, when BA reaches full rank (i.e., $B \in \mathbb{R}^{m \times m}$ and $A \in \mathbb{R}^{m \times n}$), optimizing B and A simultaneously follows a very different training trajectory compared to the original model.

2.4 GaLore for Memory-Efficient Training

For a complex optimization problem such as LLM pre-training, it may be difficult to capture the entire gradient trajectory with a single low-rank subspace. One reason is that the principal subspaces of B_t and C_t (and thus G_t) may change over time. In fact, if we keep the same projection P and Q, then the learned weights will only grow along these subspaces, which is no longer full-parameter training. Fortunately, for this, GaLore can switch subspaces during training and learn full-rank weights without increasing the memory footprint.

Composition of Low-Rank Subspaces

We allow GaLore to switch across low-rank subspaces:

$$W_t = W_0 + \Delta W_{T_1} + \Delta W_{T_2} + \ldots + \Delta W_{T_n}, \tag{2.14}$$

where $t \in \left[\sum_{i=1}^{n-1} T_i, \sum_{i=1}^{n} T_i\right]$ and $\Delta W_{T_i} = \eta \sum_{t=0}^{T_i - 1} \tilde{G}_t$ is the summation of all T_i updates within the i-th subspace. When switching to i-th subspace at step $t = T_i$, we re-initialize the projector P_t and Q_t by performing SVD on the current gradient G_t by Equation 2.12. We illustrate how the trajectory of $\tilde{G}_t$ traverses through multiple low-rank subspaces in Fig. 2.2. In the experiment section, we show that allowing multiple low-rank subspaces is the key to achieving the successful pre-training of LLMs.

Following the above procedure, the switching frequency T becomes a hyperparameter. The ablation study (Fig. 2.5) shows a sweet spot exists. A very frequent subspace change increases the overhead (since new P_t and Q_t need to be computed) and breaks the condition of constant projection in Theorem 2.3.7. In practice, it may also impact the fidelity of the optimizer states, which accumulate over multiple training steps. On the other hand, a less frequent change may make the algorithm stuck in a region that is no longer important to optimize (the convergence proof in Theorem 2.3.7 only means good progress in the designated subspace, but does not mean good overall performance). While optimal T depends on the total training iterations and task complexity, we find that a value between $T = 50$ to $T = 1000$ makes not much difference. Thus, the total computational overhead induced by SVD is negligible ($< 10\%$) compared to other memory-efficient training techniques such as memory offloading (Rajbhandari et al., 2020).

Memory-Efficient Optimization

Reducing memory footprint of gradient statistics. GaLore significantly reduces the memory cost of optimizer that heavily relies on component-wise gradient statistics, such as Adam (Kingma and Ba, 2015). When $\rho_t \equiv$ Adam, by projecting G_t into its low-rank form R_t, Adam's gradient regularizer $\rho_t(R_t)$ only needs to track low-rank gradient statistics. where M_t and V_t are the first-order and second-order momentum, respectively. GaLore computes the low-rank normalized gradient N_t as follows:

$$N_t = \rho_t(R_t) = M_t/(\sqrt{V_t} + \epsilon). \tag{2.15}$$

GaLore can also apply to other optimizers (e.g., Adafactor) that have similar update rules and require a large amount of memory to store gradient statistics.

Reducing memory usage of projection matrices. To achieve the best memory-performance trade-off, we only use one project matrix P or Q, projecting the gradient G into $P^\top G$ if $m \leq n$ and GQ otherwise. We present the algorithm applying GaLore to Adam in Algorithm 1.

With this setting, GaLore requires less memory than LoRA during training. As GaLore can always merge ΔW_t to W_0 during weight updates, it does not need to store a separate low-rank factorization BA. In total, GaLore requires $(mn+mr+2nr)$ memory, while LoRA requires $(mn + 3mr + 3nr)$ memory.

As Theorem 2.3.7 does not require the projection matrix to be carefully calibrated,

Algorithm 1 Adam with GaLore

Input: A layer weight matrix $W \in \mathbb{R}^{m \times n}$ with $m \leq n$. Step size η, scale factor α, decay rates β_1, β_2, rank r, subspace change frequency T.
Initialize first-order moment $M_0 \in \mathbb{R}^{n \times r} \leftarrow 0$
Initialize second-order moment $V_0 \in \mathbb{R}^{n \times r} \leftarrow 0$
Initialize step $t \leftarrow 0$
repeat
  $G_t \in \mathbb{R}^{m \times n} \leftarrow -\nabla_W \varphi_t(W_t)$
  if $t \bmod T = 0$ **then**
    $U, S, V \leftarrow \text{SVD}(G_t)$
    $P_t \leftarrow U[:, :r]$ {Initialize left projector as $m \leq n$}
  else
    $P_t \leftarrow P_{t-1}$ {Reuse the previous projector}
  end if
  $R_t \leftarrow P_t^\top G_t$ {Project gradient into compact space}

  UPDATE(R_t) **by Adam**
    $M_t \leftarrow \beta_1 \cdot M_{t-1} + (1 - \beta_1) \cdot R_t$
    $V_t \leftarrow \beta_2 \cdot V_{t-1} + (1 - \beta_2) \cdot R_t^2$
    $M_t \leftarrow M_t / (1 - \beta_1^t)$
    $V_t \leftarrow V_t / (1 - \beta_2^t)$
    $N_t \leftarrow M_t / (\sqrt{V_t} + \epsilon)$

  $\tilde{G}_t \leftarrow \alpha \cdot P N_t$ {Project back to original space}
  $W_t \leftarrow W_{t-1} + \eta \cdot \tilde{G}_t$
  $t \leftarrow t + 1$
until convergence criteria met
return W_t

we can further reduce the memory cost of projection matrices by quantization and efficient parameterization, which we leave for future work.

Combining with Existing Techniques

GaLore is compatible with existing memory-efficient optimization techniques. In our work, we mainly consider applying GaLore with 8-bit optimizers and per-layer weight updates.

8-bit optimizers. Dettmers, Lewis, et al. (2022) proposed 8-bit Adam optimizer that maintains 32-bit optimizer performance at a fraction of the memory footprint. We apply GaLore directly to the existing implementation of 8-bit Adam.

Table 2.1: Comparison with low-rank algorithms on pre-training various sizes of LLaMA models on C4 dataset. Validation perplexity is reported, along with a memory estimate of the total of parameters and optimizer states based on BF16 format. The actual memory footprint of GaLore is reported in Fig. 2.4.

	60M	**130M**	**350M**	**1B**
Full-Rank	34.06 (0.36G)	25.08 (0.76G)	18.80 (2.06G)	15.56 (7.80G)
GaLore	**34.88** (0.24G)	**25.36** (0.52G)	**18.95** (1.22G)	**15.64** (4.38G)
Low-Rank	78.18 (0.26G)	45.51 (0.54G)	37.41 (1.08G)	142.53 (3.57G)
LoRA	34.99 (0.36G)	33.92 (0.80G)	25.58 (1.76G)	19.21 (6.17G)
ReLoRA	37.04 (0.36G)	29.37 (0.80G)	29.08 (1.76G)	18.33 (6.17G)
r/d_{model}	128 / 256	256 / 768	256 / 1024	512 / 2048
Training Tokens	1.1B	2.2B	6.4B	13.1B

Per-layer weight updates. In practice, the optimizer typically performs a single weight update for all layers after backpropagation. This is done by storing the entire weight gradients in memory. To further reduce the memory footprint during training, we adopt per-layer weight updates to GaLore, which performs the weight updates during backpropagation. This is the same technique proposed in recent works to reduce memory requirement (Lv, Yan, et al., 2023; Lv, Y. Yang, et al., 2023).

Hyperparameters of GaLore

In addition to Adam's original hyperparameters, GaLore only introduces very few additional hyperparameters: the rank r which is also present in LoRA, the subspace change frequency T (see Sec. 2.4), and the scale factor α.

Scale factor α controls the strength of the low-rank update, which is similar to the scale factor α/r appended to the low-rank adaptor in Hu et al. (2022). We note that the α does not depend on the rank r in our case. This is because, when r is small during pre-training, α/r significantly affects the convergence rate, unlike fine-tuning.

2.5 Experiments

We evaluate GaLore on both pre-training and fine-tuning of LLMs. All experiments run on NVIDIA A100 GPUs.

Pre-training on C4. To evaluate its performance, we apply GaLore to train LLaMA-based large language models on the C4 dataset. C4 dataset is a colossal,

Table 2.2: Pre-training LLaMA 7B on C4 dataset for 150K steps. Validation perplexity and memory estimate are reported.

	Mem	**40K**	**80K**	**120K**	**150K**
8-bit GaLore	18G	17.94	15.39	14.95	14.65
8-bit Adam	26G	18.09	15.47	14.83	14.61
Tokens (B)		5.2	10.5	15.7	19.7

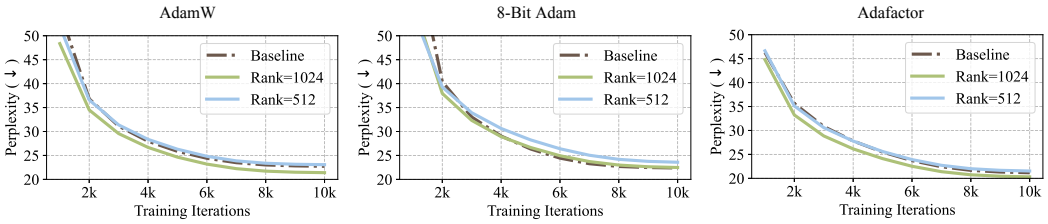

Figure 2.3: Applying GaLore to different optimizers for pre-training LLaMA 1B on C4 dataset for 10K steps. Validation perplexity over training steps is reported. We apply GaLore to each optimizer with the rank of 512 and 1024, where the 1B model dimension is 2048.

cleaned version of Common Crawl's web crawl corpus, which is mainly intended to pre-train language models and word representations (Raffel et al., 2020). To best simulate the practical pre-training scenario, we train without data repetition over a sufficiently large amount of data, across a range of model sizes up to 7 Billion parameters.

Architecture and hyperparameters. We follow the experiment setup from Lialin et al. (2024), which adopts a LLaMA-based[3] architecture with RMSNorm and SwiGLU activations (B. Zhang and Sennrich, 2019; Shazeer, 2020; Touvron et al., 2023). For each model size, we use the same set of hyperparameters across methods, except the learning rate. We run all experiments with BF16 format to reduce memory usage, and we tune the learning rate for each method under the same amount of computational budget and report the best performance. The details of our task setups and hyperparameters are provided in the appendix.

Fine-tuning on GLUE tasks. GLUE is a benchmark for evaluating the performance of NLP models on a variety of tasks, including sentiment analysis, question

[3]LLaMA materials in our paper are subject to LLaMA community license.

answering, and textual entailment (A. Wang et al., 2019). We use GLUE tasks to benchmark GaLore against LoRA for memory-efficient fine-tuning.

Comparison with Existing Low-Rank Methods

We first compare GaLore with existing low-rank methods using Adam optimizer across a range of model sizes.

Full-Rank Our baseline method that applies Adam optimizer with full-rank weights and optimizer states.

Low-Rank We also evaluate a traditional low-rank approach that represents the weights by learnable low-rank factorization: $W = BA$ (Kamalakara et al., 2022).

LoRA Hu et al. (2022) proposed LoRA to fine-tune pre-trained models with low-rank adaptors: $W = W_0 + BA$, where W_0 is fixed initial weights and BA is a learnable low-rank adaptor. In the case of pre-training, W_0 is the full-rank initialization matrix. We set LoRA alpha to 32 and LoRA dropout to 0.05 as their default settings.

ReLoRA Lialin et al. (2024) proposed ReLoRA, a variant of LoRA designed for pre-training, which periodically merges BA into W, and initializes new BA with a reset on optimizer states and learning rate. ReLoRA requires careful tuning of merging frequency, learning rate reset, and optimizer states reset. We evaluate ReLoRA without a full-rank training warmup for a fair comparison.

For GaLore, we set subspace frequency T to 200 and scale factor α to 0.25 across all model sizes in Table 2.1. For each model size, we pick the same rank r for all low-rank methods, and we apply them to all multi-head attention layers and feed-forward layers in the models. We train all models using Adam optimizer with the default hyperparameters (e.g., $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$). We also estimate the memory usage based on BF16 format, including the memory for weight parameters and optimizer states. As shown in Table 2.1, GaLore outperforms other low-rank methods and achieves comparable performance to full-rank training. We note that for 1B model size, GaLore even outperforms full-rank baseline when $r = 1024$ instead of $r = 512$. Compared to LoRA and ReLoRA, GaLore requires less memory for storing model parameters and optimizer states. A detailed training setting of each model and memory estimation for each method are in the appendix.

GaLore with Memory-Efficient Optimizers

We demonstrate that GaLore can be applied to various learning algorithms, especially memory-efficient optimizers, to further reduce the memory footprint. We apply GaLore to AdamW, 8-bit Adam, and Adafactor optimizers (Shazeer and Stern, 2018; Loshchilov and Hutter, 2019; Dettmers, Lewis, et al., 2022). We consider Adafactor with first-order statistics to avoid performance degradation.

We evaluate them on LLaMA 1B architecture with 10K training steps, and we tune the learning rate for each setting and report the best performance. As shown in Fig. 2.3, applying GaLore does not significantly affect their convergence. By using GaLore with a rank of 512, the memory footprint is reduced by up to 62.5%, on top of the memory savings from using 8-bit Adam or Adafactor optimizer. Since 8-bit Adam requires less memory than others, we denote 8-bit GaLore as GaLore with 8-bit Adam, and use it as the default method for the following experiments on 7B model pre-training and memory measurement.

Scaling up to LLaMA 7B Architecture

Scaling ability to 7B models is a key factor for demonstrating if GaLore is effective for practical LLM pre-training scenarios. We evaluate GaLore on an LLaMA 7B architecture with an embedding size of 4096 and total layers of 32. We train the model for 150K steps with 19.7B tokens, using 8-node training in parallel with a total of 64 A100 GPUs. Due to computational constraints, we compare 8-bit GaLore ($r = 1024$) with 8-bit Adam with a single trial without tuning the hyperparameters. As shown in Table 2.2, after 150K steps, 8-bit GaLore achieves a perplexity of 14.65, comparable to 8-bit Adam with a perplexity of 14.61.

Memory-Efficient Fine-Tuning

GaLore not only achieves memory-efficient pre-training but also can be used for memory-efficient fine-tuning. We fine-tune pre-trained RoBERTa models on GLUE tasks using GaLore and compare its performance with a full fine-tuning baseline and LoRA. We use hyperparameters from Hu et al. (2022) for LoRA and tune the learning rate and scale factor for GaLore. This demonstrates that GaLore can serve as a full-stack memory-efficient training strategy for both LLM pre-training and fine-tuning.

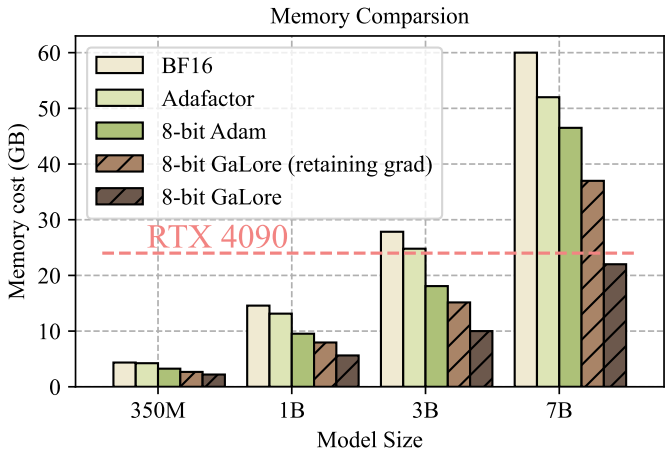

Figure 2.4: Memory usage for different methods at various model sizes, evaluated with a token batch size of 256. 8-bit GaLore (retaining grad) disables per-layer weight updates but stores weight gradients during training.

Measurement of Memory and Throughput

While Table 2.1 gives the theoretical benefit of GaLore compared to other methods in terms of memory usage, we also measure the actual memory footprint of training LLaMA models by various methods, with a token batch size of 256. The training is conducted on a single device setup without activation checkpointing, memory offloading, and optimizer states partitioning (Rajbhandari et al., 2020).

Training 7B models on consumer GPUs with 24G memory. As shown in Fig. 2.4, 8-bit GaLore requires significantly less memory than BF16 baseline and 8-bit Adam, and only requires 22.0G memory to pre-train LLaMA 7B with a small per-GPU token batch size (up to 500 tokens). This memory footprint is within 24GB VRAM capacity of a single GPU such as NVIDIA RTX 4090. In addition, when activation checkpointing is enabled, per-GPU token batch size can be increased up to 4096. While the batch size is small per GPU, it can be scaled up with data parallelism, which requires much lower bandwidth for inter-GPU communication, compared to model parallelism. Therefore, it is possible that GaLore can be used for elastic training Lin et al., 2019 7B models on consumer GPUs such as RTX 4090s.

Specifically, we present the memory breakdown in Fig. 2.1. It shows that 8-bit GaLore reduces 37.92G (63.3%) and 24.5G (52.3%) total memory compared to BF16 Adam baseline and 8-bit Adam, respectively. Compared to 8-bit Adam, 8-bit GaLore mainly reduces the memory in two parts: (1) low-rank gradient projection reduces 9.6G (65.5%) memory of storing optimizer states, and (2) using per-layer

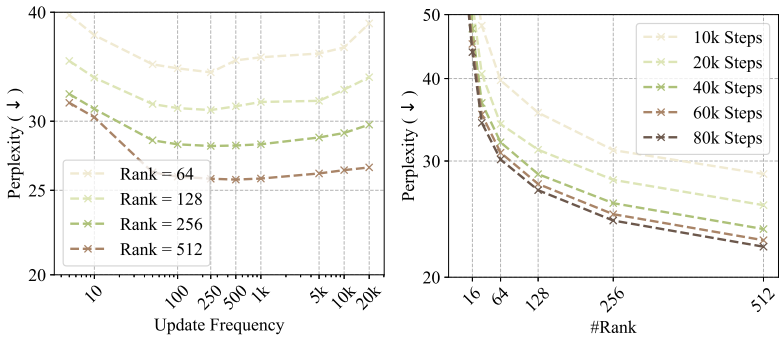

Figure 2.5: Ablation study of GaLore on 130M models. **Left:** varying subspace update frequency T. **Right:** varying subspace rank and training iterations.

weight updates reduces 13.5G memory of storing weight gradients.

Throughput overhead of GaLore. We also measure the throughput of the pre-training LLaMA 1B model with 8-bit GaLore and other methods, where the results can be found in the appendix. Particularly, the current implementation of 8-bit GaLore achieves 1019.63 tokens/second, which induces 17% overhead compared to 8-bit Adam implementation. Disabling per-layer weight updates for GaLore achieves 1109.38 tokens/second, improving the throughput by 8.8%. We note that our results do not require offloading strategies or checkpointing, which can significantly impact training throughput. We leave optimizing the efficiency of GaLore implementation for future work.

2.6 Ablation Study

How many subspaces are needed during pre-training? We observe that both too frequent and too slow changes of subspaces hurt the convergence, as shown in Fig. 2.5 (left). The reason has been discussed in Sec. 2.4. In general, for small r, the subspace switching should happen more to avoid wasting optimization steps in the wrong subspace, while for large r the gradient updates cover more subspaces, providing more cushion.

How does the rank of subspace affect the convergence? Within a certain range of rank values, decreasing the rank only slightly affects the convergence rate, causing a slowdown with a nearly linear trend. As shown in Fig. 2.5 (right), training with a rank of 128 using 80K steps achieves a lower loss than training with a rank of 512 using 20K steps. This shows that GaLore can be used to trade-off between memory and computational cost. In a memory-constrained scenario, reducing the

rank allows us to stay within the memory budget while training for more steps to preserve the performance.

2.7 Conclusion

We propose GaLore, a memory-efficient pre-training and fine-tuning strategy for large language models. GaLore significantly reduces memory usage by up to 65.5% in optimizer states while maintaining both efficiency and performance for large-scale LLM pre-training and fine-tuning.

We identify several open problems for GaLore, which include (1) applying GaLore on training of various models such as vision transformers (Dosovitskiy et al., 2021) and diffusion models (Ho, Jain, and Abbeel, 2020), (2) further enhancing memory efficiency by employing low-memory projection matrices, and (3) exploring the feasibility of elastic data distributed training on low-bandwidth consumer-grade hardware.

We hope that our work will inspire future research on memory-efficient training from the perspective of gradient low-rank projection. We believe that GaLore will be a valuable tool for the community, enabling the training of large-scale models on consumer-grade hardware with limited resources.

References

Anil, Rohan et al. (2019). "Memory efficient adaptive optimization". In: *Advances in Neural Information Processing Systems*.

Chaudhry, Arslan et al. (2020). "Continual learning in low-rank orthogonal subspaces". In: *Advances in Neural Information Processing Systems*.

Chen, Han, Garvesh Raskutti, and Ming Yuan (2019). "Non-Convex Projected Gradient Descent for Generalized Low-Rank Tensor Regression". In: *Journal of Machine Learning Research*.

Chen, Tianqi et al. (2016). "Training Deep Nets with Sublinear Memory Cost". In: *ArXiv preprint arXiv:1604.06174*.

Chen, Yudong and Martin J. Wainwright (2015). "Fast Low-Rank Estimation by Projected Gradient Descent: General Statistical and Algorithmic Guarantees". In: *ArXiv preprint arXiv:1509.03025*.

Chowdhery, Aakanksha et al. (2023). "Palm: Scaling language modeling with pathways". In: *Journal of Machine Learning Research*.

Cosson, Romain et al. (2023). "Low-Rank Gradient Descent". In: *IEEE Open Journal of Control Systems*.

Dettmers, Tim, Mike Lewis, et al. (2022). “8-bit Optimizers via Block-wise Quantization”. In: *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.

Dettmers, Tim, Artidoro Pagnoni, et al. (2024). “Qlora: Efficient finetuning of quantized llms”. In: *Advances in Neural Information Processing Systems*.

Ding, Ning et al. (2022). “Delta Tuning: A Comprehensive Study of Parameter Efficient Methods for Pre-trained Language Models”. In: *ArXiv preprint arXiv:2203.06904*.

Dosovitskiy, Alexey et al. (2021). “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale”. In: *International Conference on Learning Representations*.

Gooneratne, Mary et al. (2020). “Low-Rank Gradient Approximation for Memory-Efficient on-Device Training of Deep Neural Network”. In: *2020 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2020, Barcelona, Spain, May 4-8, 2020*. IEEE.

Gur-Ari, Guy, Daniel A. Roberts, and Ethan Dyer (2018). “Gradient Descent Happens in a Tiny Subspace”. In: *ArXiv preprint arXiv:1812.04754*.

Hao, Yongchang, Yanshuai Cao, and Lili Mou (2024). “Flora: Low-Rank Adapters Are Secretly Gradient Compressors”. In: *ArXiv preprint arXiv:2402.03293*.

Ho, Jonathan, Ajay Jain, and Pieter Abbeel (2020). “Denoising diffusion probabilistic models”. In: *Advances in neural information processing systems*.

Hu, Edward J. et al. (2022). “LoRA: Low-Rank Adaptation of Large Language Models”. In: *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.

Huang, Siyuan et al. (2023). “Low-Rank Gradient Descent for Memory-Efficient Training of Deep In-Memory Arrays”. In: *ACM Journal on Emerging Technologies in Computing Systems*.

Kamalakara, Siddhartha Rao et al. (2022). “Exploring Low Rank Training of Deep Neural Networks”. In: *ArXiv preprint arXiv:2209.13569*.

Kingma, Diederik P. and Jimmy Ba (2015). “Adam: A Method for Stochastic Optimization”. In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Larsen, Brett W. et al. (2022). “How many degrees of freedom do we need to train deep networks: a loss landscape perspective”. In: *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.

Lee, Yoonho and Seungjin Choi (2018). "Gradient-Based Meta-Learning with Learned Layerwise Metric and Subspace". In: *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*. PMLR.

Li, Bingrui, Jianfei Chen, and Jun Zhu (2024). "Memory efficient optimizers with 4-bit states". In: *Advances in Neural Information Processing Systems*.

Lialin, Vladislav et al. (2024). "ReLoRA: High-Rank Training Through Low-Rank Updates". In: *The Twelfth International Conference on Learning Representations*.

Lin, Haibin et al. (2019). "Dynamic mini-batch sgd for elastic distributed training: Learning in the limbo of resources". In: *arXiv preprint arXiv:1904.12043*.

Loshchilov, Ilya and Frank Hutter (2019). "Decoupled Weight Decay Regularization". In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Lv, Kai, Hang Yan, et al. (2023). "AdaLomo: Low-memory Optimization with Adaptive Learning Rate". In: *ArXiv preprint arXiv:2310.10195*.

Lv, Kai, Yuqing Yang, et al. (2023). "Full Parameter Fine-tuning for Large Language Models with Limited Resources". In: *ArXiv preprint arXiv:2306.09782*.

Modoranu, Ionut-Vlad et al. (2023). "Error Feedback Can Accurately Compress Preconditioners". In: *ArXiv preprint arXiv:2306.06098*.

Raffel, Colin et al. (2020). "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer". In: *J. Mach. Learn. Res.*

Rajbhandari, Samyam et al. (2020). "Zero: Memory optimizations toward training trillion parameter models". In: *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*.

Renduchintala, Adithya, Tugrul Konuk, and Oleksii Kuchaiev (2023). "Tied-Lora: Enhacing Parameter Efficiency of LoRA with Weight Tying". In: *ArXiv preprint arXiv:2311.09578*.

Shazeer, Noam (2020). "Glu variants improve transformer". In: *arXiv preprint arXiv:2002.05202*.

Shazeer, Noam and Mitchell Stern (2018). "Adafactor: Adaptive Learning Rates with Sublinear Memory Cost". In: *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*. PMLR.

Sheng, Ying et al. (2023). "S-LoRA: Serving Thousands of Concurrent LoRA Adapters". In: *ArXiv preprint arXiv:2311.03285*.

Tian, Yuandong, Yiping Wang, et al. (2024). "JoMA: Demystifying Multilayer Transformers via Joint Dynamics of MLP and Attention". In: *The Twelfth International Conference on Learning Representations*.

Tian, Yuandong, Lantao Yu, et al. (2020). "Understanding self-supervised learning with dual deep networks". In: *ArXiv preprint arXiv:2010.00578*.

Touvron, Hugo et al. (2023). "Llama 2: Open foundation and fine-tuned chat models". In: *arXiv preprint arXiv:2307.09288*.

Vogels, Thijs, Sai Praneeth Karimireddy, and Martin Jaggi (2020). "Practical low-rank communication compression in decentralized deep learning". In: *Advances in Neural Information Processing Systems*.

Wang, Alex et al. (2019). "GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding". In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Wang, Hongyi, Saurabh Agarwal, et al. (2023). "Cuttlefish: Low-rank model training without all the tuning". In: *Proceedings of Machine Learning and Systems*.

Wang, Hongyi, Scott Sievert, et al. (2018). "Atomo: Communication-efficient learning via atomic sparsification". In: *Advances in neural information processing systems* 31.

Wang, Yiming et al. (2023). "MultiLoRA: Democratizing LoRA for Better Multi-Task Learning". In: *ArXiv preprint arXiv:2311.11501*.

Xia, Wenhan, Chengwei Qin, and Elad Hazan (2024). "Chain of LoRA: Efficient Fine-tuning of Language Models via Residual Learning". In: *ArXiv preprint arXiv:2401.04151*.

Yang, Greg, James B Simon, and Jeremy Bernstein (2023). "A spectral condition for feature learning". In: *arXiv preprint arXiv:2310.17813*.

Zhang, Biao and Rico Sennrich (2019). "Root mean square layer normalization". In: *Advances in Neural Information Processing Systems* 32.

Zhang, Longteng et al. (2023). "Lora-fa: Memory-efficient low-rank adaptation for large language models fine-tuning". In: *arXiv preprint arXiv:2308.03303*.

Zhao, Jiawei, Florian Tobias Schaefer, and Anima Anandkumar (2022). "ZerO Initialization: Initializing Neural Networks with only Zeros and Ones". In: *Transactions on Machine Learning Research*.

Zhao, Jiawei, Yifei Zhang, et al. (2023). "Inrank: Incremental low-rank learning". In: *arXiv preprint arXiv:2306.11250*.

Chapter 3

INRANK: INCREMENTAL LOW-RANK LEARNING

3.1 Introduction

The generalization ability of deep neural networks continues to intrigue researchers since the classical theory is not applicable in the over-parameterized regime, where there are more learnable parameters than training samples. Instead, efforts to understand this puzzle are based on the belief that first-order learning algorithms (e.g., stochastic gradient descent) implicitly bias the neural networks toward simple solutions.

For instance, it has been shown that stochastic gradient descent implicitly minimizes the rank of solutions during training (Arora et al., 2019). Recent theoretical studies have further demonstrated one of its training characterizations - Greedy Low-Rank Learning (GLRL) (Z. Li, Luo, and Lyu, 2021; Jacot et al., 2022). GLRL characterizes the trajectory of stochastic gradient descent, which performs a rank-constrained optimization and greedily increases the rank whenever it fails to reach a global minimizer.

However, one major drawback is that the GLRL theory requires the assumption of infinitesimal initialization, which is impractical as gradient descent cannot effortlessly escape from the saddle point at zero, unless the noise is large enough. Therefore, a generalized notion of GLRL under practical initialization is needed to bridge the gap between theory and practice.

In this work, we generalize the theory of GLRL by removing the requirement of infinitesimal initialization. To do this, we focus on characterizing the trajectories of a new set of quantities, *cumulative weight updates*, instead of weight matrices. Cumulative weight updates do not include the initialization values, and only incorporate the rest of the updates of the weight matrices during training. This allows us to remove the requirement of infinitesimal initialization in GLRL.

We establish incremental rank augmentation of cumulative weight updates during training under arbitrary orthogonal initialization of the weights. This new formulation proves that low-rank learning can be extended to non-zero initialization, where the singular vector with a larger value in the associated target matrix is learned

exponentially faster. We prove this relationship by following the work of Saxe, McClelland, and Ganguli (2014) to analyze the evolution of each mode (singular vector) independently, which can be achieved by ensuring orthogonality over the weights matrices and inputs in a three-layer linear network.

Empirically, we further demonstrate that standard networks (e.g., transformers) and training algorithms (e.g., SGD, Adam) follow low-rank learning trajectories on the cumulative weight updates, under standard weight initialization. However, current algorithms can not exploit the low-rank property to improve computational efficiency as the networks are not parameterized in low-rank.

To address this, we propose *Incremental Low-Rank Learning* (InRank), which parameterizes the cumulative weight updates in low-rank while incrementally augmenting its rank during training, as illustrated in Figure 3.1. InRank adds a new batch of modes whenever a certain quantity, known as the explained ratio, exceeds a certain threshold. The explained ratio represents the amount of information in the underlying spectrum that the current rank can explain. A low explained ratio indicates that the current rank is inadequate to represent the spectrum, necessitating the addition of more modes.

InRank is capable of identifying *intrinsic rank* of networks during training. The intrinsic rank of a neural network is defined as the minimum sufficient rank that trains the network from scratch without sacrificing performance. The capability of finding the intrinsic rank addresses the challenge of pre-defining the fixed ranks in training low-rank neural networks, which requires expensive hyperparameter tuning. An inappropriate selection of rank may either limit model capacity, hinder the learning process, or result in excessive memory usage and computation, thereby negating the benefits of low-rank factorization. We further improve computational efficiency by applying InRank only in the initial phase of training. This approach mitigates the computational cost induced by expensive SVD operations in InRank, while maintaining comparable accuracy as the full-rank models.

3.2 Related Work

Implicit regularization has been well studied to explain excellent generalization in neural networks (Gunasekar et al., 2018; Rahaman et al., 2019). Implicit rank regularization stands out among the diverse aspects of implicit regularization, which demonstrates that a neural network minimizes its rank implicitly during training (Arora et al., 2019; Gissin, Shalev-Shwartz, and Daniely, 2019). Further research

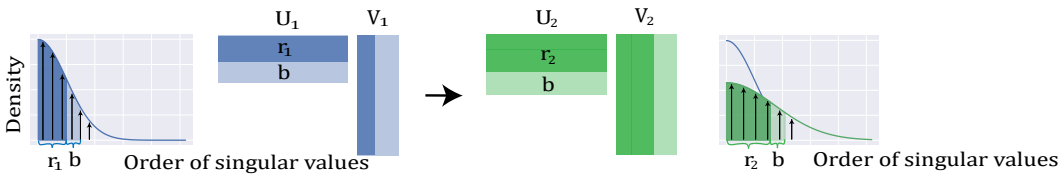

Figure 3.1: **Incremental Low-Rank Learning from iteration t_1 to t_2.** U and V represent any factorized layer. Density plots indicate the strength of each singular vector (normalized by the total strengths). Solid areas represent how much information in the spectrum is explained by the current rank r_t at iteration t. From iteration t_1 to t_2, InRank adds $r_2 - r_1$ additional ranks to ensure the ratio of the explained information is greater than a certain threshold α.

has corroborated that neural networks pursue a greedy low-rank learning strategy under infinitesimal initialization (Razin, Maman, and Cohen, 2021; Jacot et al., 2022; Z. Li, Luo, and Lyu, 2021). However, the practical advantages of such an approach remain unexplored, predominantly due to the challenges of deviating from the infinitesimal initialization assumption.

Low-rank training and other structured pruning methods aim to promote structured sparsity within neural networks (NNs) throughout the training process, enabling substantial computational acceleration (You et al., 2022; Dao et al., 2022). The low-rank training technique has proven effective for training low-rank neural networks from scratch (Ioannou et al., 2016; Yang et al., 2020; Schotthöfer et al., 2022). Nonetheless, this method often necessitates extra hyperparameters, such as the rank of the factorization, which can be challenging to determine accurately, and thus it requires careful tuning.

Idelbayev and Carreira-Perpinan (2020) propose the LC compression method that explicitly integrates the learning of low-rank factors into the training process, despite its computational intensity. More recently, Wang, Agarwal, U-chupala, et al. (2023) introduce Cuttlefish, a low-rank training method capable of automatically determining the factorization rank in the early stages of training. However, Cuttlefish requires a pre-set full-rank initialization and lacks a theoretical comprehension of its low-rank behavior, unlike our proposed InRank.

Moreover, low-rank training has been employed for fine-tuning large-scale pre-trained models (Hu et al., 2021), and for reducing communication overhead in distributed training (Vogels, Karimireddy, and Jaggi, 2019; Wang, Agarwal, and

Papailiopoulos, n.d.). C. Li et al. (2018) adopt the low-rankness in cumulative weight updates to measure the intrinsic dimension of objective landscapes. The concept of incremental learning has been examined within the context of learning partial differential equations using neural networks, such as parameter expansion in the frequency domain (Zhao, George, et al., 2022), and increasing the complexity of the underlying PDE problem (Huang and Alkhalifah, 2021).

Algorithm 2 Greedy Low-Rank Learning (GLRL)

Input: Convex cost C, product matrix $A_\theta = W^1 \cdots W^L$, tolerance ϵ, learning rate η, training steps T
Compute the first singular vector of $\nabla C(0)$: $u, s, v \leftarrow \mathrm{SVD}_1(\nabla C(0))$
Initialize parameters and network width: $\theta \leftarrow (-\epsilon v^T, \epsilon, \ldots, \epsilon u)$, $w \leftarrow 1$
repeat
  Train width-w deep linear network for T steps using SGD with learning rate η
  Compute the first singular vectors of $\nabla C(A_\theta)$: $u, s, v \leftarrow \mathrm{SVD}_1(\nabla C(A_\theta))$
  Expand network width: $w \leftarrow w + 1$
  Initialize additional parameters:

$$\theta \leftarrow \left(\begin{pmatrix} W^1 \\ -\epsilon v^T \end{pmatrix}, \begin{pmatrix} W^2 & 0 \\ 0 & \epsilon \end{pmatrix}, \ldots, \begin{pmatrix} W^L & \epsilon u \end{pmatrix} \right)$$

until $C(A_\theta) \geq C_{min} + \epsilon$

3.3 Greedy Low-Rank Learning

In this section, we first introduce greedy low-rank learning (GLRL) and its practical limitations. We wish to train the function $\mathcal{F}(x)$ to learn a particular input-output map given a set of P training samples $(x_\mu, y_\mu) \in \mathbb{R}^{N_x \times N_y}$, where $\mu = 1, ..., P$. Training is accomplished by minimizing the squared error $\mathcal{L} = \frac{1}{2} \sum_{\mu=1}^{P} \|y_\mu - \mathcal{F}(x_\mu)\|_2^2$ using gradient descent with a step size η.

We first model $\mathcal{F}(x)$ to be a deep linear network: $\mathcal{F}(x) = W^L ... W^1 x$, where $W^l \in \mathbb{R}^{N_h \times N_h}$ for $l \in 1, ..., L$. We let $A_\theta = W^L ... W^1$ denote the product matrix of the network, and θ denote the whole parameter vector. Thus, we also denote the training error as $C(A_\theta)$ where C is a convex error (e.g., the squared error).

The following theorem characterizes the implicit rank regularization behavior of gradient descent under infinitesimal initialization.

Theorem 3.3.1 (Greedy Low-Rank Learning, informal) *If we initialize $W^1, ..., W^L$ to be infinitesimal, then the product matrix A_θ follows a greedy low-rank learning*

trajectory, such that the gradient descent first searches over a rank-1 subspace of A_θ, and then greedily increases the rank by one whenever it fails to reach a local minimizer.

Theorem 3.3.1 characterizes the trajectory of gradient descent, which performs a rank-constrained optimization and greedily relaxes the rank restriction until it finds a local minimizer.

Inspired by this implicit low-rank trajectory, the greedy low-rank learning (GLRL) algorithm is proposed to capture this implicit behavior explicitly (Z. Li, Luo, and Lyu, 2021). As shown in Algorithm 2, GLRL incrementally increases the rank of the weight matrices in a deep linear network and initializes the additional rows and columns based on the top singular vector of the current matrix derivative.

Although the GLRL algorithm provides a theoretical understanding of implicit rank regularization, it has some practical drawbacks. One notable limitation is the infinitesimally small initialization, which leads to slow convergence and makes it difficult to apply the algorithm in large-scale settings. In addition, GLRL is only applicable to linear networks as it highly relies on the product matrix A_θ. This makes it inapplicable to practical neural networks with non-linear activation functions.

3.4 Cumulative Weight Updates follow Low-Rank Learning Trajectory

In order to generalize GLRL beyond infinitesimal initialization, we focus on *cumulative weight updates* that characterize GLRL for any regular initializations. We define the cumulative weight updates as follows:

Definition 3.4.1 (Cumulative Weight Updates) *The* cumulative weight updates d_t *at iteration t is defined as the difference between the current parameterization w_t and initialization w_0 in the parameter space, such that*

$$d_t = w_t - w_0 = \sum_{i=1}^{t} \Delta w_i. \tag{3.1}$$

The cumulative weight updates d_t have been widely studied in the literature, especially in the field of distributed training (Vogels, Karimireddy, and Jaggi, 2019), as it is known to exhibit low-rank properties.

This is attributed to the fact that d_t is a summation of updates to the weights Δw_i, with each update being determined by the learning algorithm and current gradient g_t.

Gradient g_t has been shown to possess low-rank properties, which has been exploited to reduce communication costs in distributed training through low-rank compression (Vogels, Karimireddy, and Jaggi, 2019; Wang, Agarwal, and Papailiopoulos, n.d.).

We theoretically prove that the cumulative weight updates d_t follow a low-rank learning trajectory, even when the initialization is not infinitesimal. We continue to focus on a linear network and analyze the difference of the product matrix $D_t = A_t - A_0$ (which can be viewed as the cumulative weight updates of the product matrix). Our goal is to demonstrate that D_t exhibits an exponential rank increase even when the initial weights are not close to zero. Our analysis builds upon the work of Saxe, McClelland, and Ganguli (2014), which studies training dynamics under orthogonal inputs.

Assumption 3.4.2 (Orthogonal Inputs) *We assume the inputs are orthogonal, i.e.,* $x_i^T x_j = 0$ *for* $i \neq j$.

Consider the input-output correlation matrix:

$$\Sigma^{yx} = \sum_{\mu=1}^{P} y_\mu x_\mu^T = U^{yy} S^{yx} V^{xx} = \sum_{\alpha=1}^{N_x} s_\alpha u_\alpha v_\alpha^T, \tag{3.2}$$

where U^{yy} and V^{xx} represent the left and right singular vectors of Σ^{yx}, and S^{yx} denotes its singular value matrix. The singular values are ordered such that $s_1 \geq s_2 \geq \cdots \geq s_{N_x}$.

We analyze a 3-layer linear network where $y = W^2 W^1 x$, $W^1 \in \mathbb{R}^{N_h \times N_x}$ and $W^2 \in \mathbb{R}^{N_y \times N_h}$ are the weight matrices of the first and second layers, respectively, and $N_h < N_x, N_y$. After training, the converged network should satisfy:

$$W^2 W^1 = \sum_{\alpha=1}^{N_h} s_\alpha u_\alpha v_\alpha^T, \tag{3.3}$$

which is the closest rank-N_h approximation to Σ^{yx}. To further analyze its trajectory, we assume that the weights are initialized as $W_0^2 = U^{yy} M^2 O^T, W_0^1 = O M^1 V^{xx^T}$, where M^2, M^1 are diagonal matrices, and O is an arbitrary orthogonal matrix. We have the following theorem for the training evolution of D_t:

Theorem 3.4.3 *For any orthogonal matrix O and scaled diagonal matrices* M^2 *and* M^1*, each singular value* $u_f(t)$ *in* D_t *at iteration t follows the trajectory:*

$$u_f(t) = \frac{s e^{2st/\tau}}{e^{2st/\tau} - 1 + s/u_0} - u_0, \tag{3.4}$$

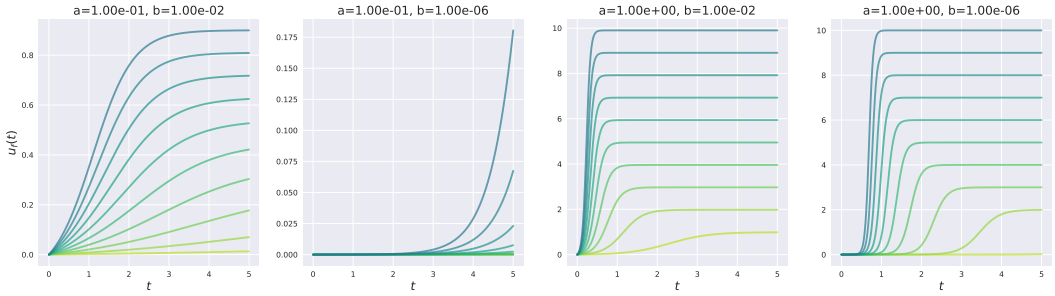

Figure 3.2: $u_f(t)$ follows low-rank learning trajectory regardless of s and u_0. We generate a set of s given $s_i = a \times i, i = 1, ..., 10$ while varying a from 0.1 to 1.0. We also generate a set of u_0 given $u_0 \sim \mathcal{N}(0, b^2)$. Darker colors indicate singular vectors with higher strengths.

where s is the target singular value in Σ^{yx}, u_0 is the initial value determined by M^2 and M^1, and τ is a constant.

$W_0^2 W_0^1$ ensures that each mode α is learned independently right from the beginning of training, enabling us to analyze the learning trajectory of each mode separately. The diagonal matrices M^2 and M^1 control the scale of the initial weights, i.e., the initial value u_0 of each mode α. Consequently, a larger u_0 that is closer to s accelerates the learning speed. We provide comprehensive proof of the theorem in the appendix for further clarity.

The sigmoid function in Theorem 3.4.3 exhibits a sharp transition from a state of no learning to full learning, with the transition point determined by the initial value u_0 and s. This indicates that if the target singular values s are distinct enough (given $s >> u_0$), each $u_f(t)$ will follow an independent sigmoid trajectory, permitting ranks to be learned sequentially and independently.

To validate this, we carry out an empirical simulation using different sets of u_0 and s. As illustrated in Figure 3.2, under various scales of initialization, the evolution of $u_f(t)$ consistently adheres to the low-rank learning trajectory. We note that analyzing weights $W^2 W^1$ directly under infinitesimal initialization in Z. Li, Luo, and Lyu (2021) can be viewed as a special case of analyzing D_t here.

Shifting our focus to practical non-linear networks, we analyze the difference of layer-wise weight matrix $D_t = W_t^l - W_0^l$ for $l = 1, ..., L$ instead of the product matrix $D_t = W_t^L ... W_t^1 - W_0^L ... W_0^1$. We also extend our evaluation to more practical cases with modern weight initialization methods. As shown in Figure 3.3, cumu-

lative weight updates D_t follow the greedy low-rank learning trajectory even under regular initializations, including Orthogonal, ZerO, and Kaiming methods (Saxe, McClelland, and Ganguli, 2014; He et al., 2015; Zhao, Schäfer, and Anandkumar, 2021).

We further verify our theory on a broad range of neural networks (e.g., transformers) and standard training algorithms (e.g., SGD, Adam), as shown in the appendix. This observation motivates us to design an efficient incremental learning algorithm that leverages the properties of cumulative weight updates.

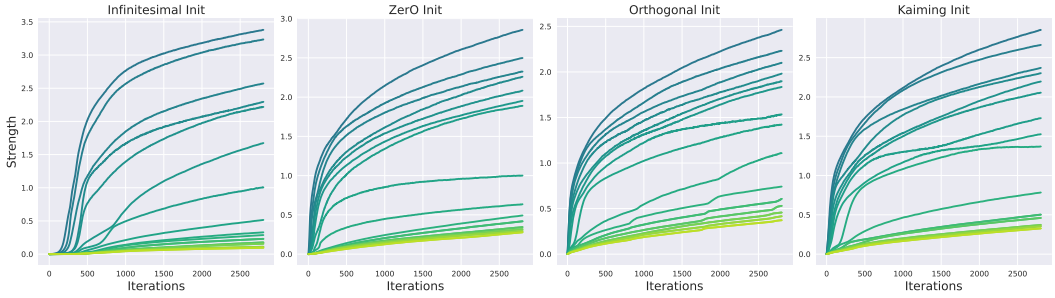

Figure 3.3: The evolutions of top 20 singular vectors of cumulative weight updates D_t over training under different initializations. They are evaluated on the training of a 3-layer perceptron on Fashion MNIST. Darker colors indicate singular vectors with higher strengths.

3.5 Incremental Learning

Motivated by the previous findings, we propose an incremental low-rank learning algorithm that leverages the implicit low-rank learning trajectory in practice. To explicitly represent the cumulative weight updates, we parametrize the weight matrix W^l at any layer l as follows:

$$W^l = W_0^l + D^l, \tag{3.5}$$

where W_0^l is the initial matrix and D^l is the summation of weight updates. Since D^l exhibits low-rank properties, we can factorize it as $D^l = U^l V^l$, resulting in:

$$W^l = W_0^l + U^l V^l, \tag{3.6}$$

where $U^l \in \mathbb{R}^{p^l \times r^l}$ and $V^l \in \mathbb{R}^{r^l \times q^l}$ are the factorized matrices, and r^l is the rank of D^l.

To emulate the implicit low-rank learning, we train factorized matrices $U^l V^l$ with an initially small rank r^l, subsequently increasing the rank (i.e., the matrix size) throughout the training process.

Algorithm 3 Incremental Low-Rank Learning (InRank)

Input: Cost function $L(W_t)$ for total weights $W = (W^1, ..., W^L)$ at iteration t, where $W_t^l = W_0^l + U_t^l V_t^l$ for each layer l (with W_0^l being non-trainable). Parameters: rank increment r, buffer b, explained variance threshold α, initialization scale ϵ, learning rate η, and number of iterations T

Initialize W_0^l using standard initialization, and set $U_0^l, V_0^l \leftarrow 0$

Compute the top $(1 + b)$ singular vectors: $u^l, s^l, v^l \leftarrow \text{SVD}_{(1+b)}\left(\frac{\partial L(W_0)}{\partial W_0^l}\right)$

Initialize factorized weights with small ϵ: $U_0^l \leftarrow -\epsilon v^l$, $V_0^l \leftarrow \epsilon u^l$, and set initial rank $r_0^l \leftarrow 1$

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

Train low-rank network and update $U_t^l V_t^l$ using SGD with learning rate η

Compute the top $(r^l + b)$ singular vectors: $u^l, s^l, v^l \leftarrow \text{SVD}_{(r^l+b)}(U_t^l V_t^l)$

Increment rank r_t^l to r_{t+1}^l until the explained variance ratio $g(U_t^l V_t^l, r_{t+1}^l, b) \geq \alpha$

Initialize additional parameters:

$$U_{t+1}^l \leftarrow \begin{bmatrix} U_t^l & U^* \end{bmatrix}, \quad V_{t+1}^l \leftarrow \begin{bmatrix} V_t^l & V^* \end{bmatrix}$$

{Where $U^* \in \mathbb{R}^{p^l \times (r_{t+1}^l - r_t^l)}$ and $V^* \in \mathbb{R}^{(r_{t+1}^l - r_t^l) \times q^l}$ are randomly initialized with small values}

end for

A crucial challenge lies in determining how to increase the rank r^l during training. An inappropriate choice of rank may either lead to insufficient model capacity, hinder the learning process, or result in excessive memory usage and computation, negating the benefits of low-rank factorization.

To address this, we propose a novel method for dynamically identifying when a rank increase is necessary, based on measuring the representability of the current rank r^l. Inspired by Zhao, George, et al. (2022), we define *explained ratio*:

$$g(M, r^l, b) = \frac{\sum_{i=1}^{r^l} s_i^l}{\sum_{i=1}^{r^l+b} s_i^l}, \tag{3.7}$$

where s_i^l is the i-th singular value of a matrix M, and b is a buffer size used to encompass a broader spectrum for determination. The explained ratio g quantifies the representability of the current rank r^l in the truncated spectrum (of size $(r^l + b)$) of M. A low explained ratio g indicates that the existing rank r^l cannot sufficiently represent the truncated spectrum, necessitating an increase in r^l to incorporate more useful modes.

We let $M = U^l V^l$ for each layer l and compute the explained ratio $g(U^l V^l, r^l, b)$ at each iteration (can be relaxed each k iterations in practice). By predefining an appropriate threshold α and ensuring that $g(U^l V^l, r^l, b)$ remains larger than α during training, the rank r^l can automatically increase when needed. This process is illustrated in Figure 3.1.

It is worth noting that b buffer ranks serve to provide a wider spectrum range, but their corresponding singular vectors may be less useful. These buffer ranks can be discarded by fine-tuning in the post-training stage. The full algorithm is detailed in Algorithm 3.

3.6 Evaluation

In this section, we conduct a comprehensive evaluation of our proposed InRank algorithm on GPT-2.

Our method particularly focuses on the fully-connected layers in the models, where we substitute the conventional weight parameterization with our relative parameterization as described in Equation 3.6. This operation involves applying InRank to the resulting low-rank factorized matrices. Notably, our approach is not exclusively limited to fully-connected layers. It bears the flexibility to be extended to various types of layers, including convolution and self-attention layers. However, to maintain the focus on our current research, we leave this promising exploration for future work.

We benchmark the effectiveness of our method mainly on Generative Pre-trained Transformer 2 (GPT-2), a model widely used in language tasks. In our experiment, we apply InRank to all the MLP layers in GPT-2 and assess the training of GPT-2 from scratch on the WikiText-103 dataset.

We fix the hyperparameters of InRank across all experiments and different models, including an initial rank of $r_0 = 1$, a buffer size of $b = 100$, and a threshold of $\alpha = 0.9$. We find both values r_0 and b are insensitive to the performance of InRank, and we will discuss the selection of the threshold α in the following section.

Automatic Rank Determination

A key finding from our evaluation is that InRank can automatically find the intrinsic rank of the model during training, facilitated by the automatic rank determination feature in cumulative weight updates. Figure 3.4 demonstrates that the rank identified by InRank aligns with the intrinsic rank discovered by costly sweeping across

a wide range of ranks. This capability could potentially eliminate the need for the laborious and time-intensive process of tuning the rank hyperparameters for training low-rank networks.

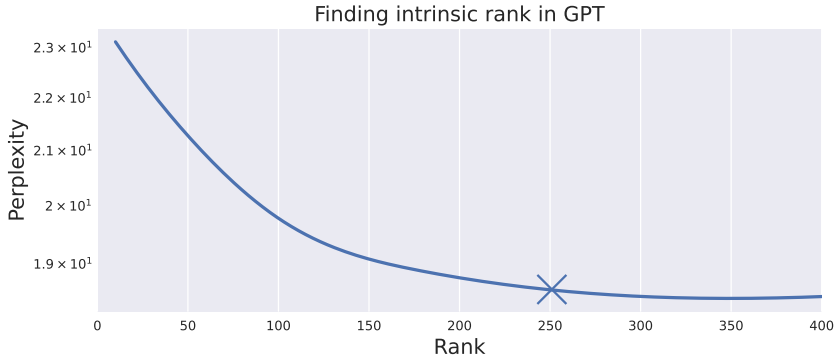

Figure 3.4: Identifying intrinsic rank in GPT-small on WikiText-103. The cross marker signifies the rank determined by InRank. The rank varies from 10 to 400.

InRank-Efficient

We aim to improve the efficiency of InRank. We find the rank increment mostly occurs during the early stages of training, remaining relatively stable thereafter. This observation suggests that the initial training phase can sufficiently infer the intrinsic rank of the model, corroborating the findings of previous work Wang, Agarwal, U-chupala, et al., 2023.

This motivates us to apply InRank only in the early stage to determine an appropriate rank for low-rank training, fixing its rank afterward. We denote this variant as InRank-Efficient. The conventional InRank is computationally expensive due to the $O(n^3)$ cost of the SVD operation for a matrix of size $n \times n$. On the other hand, InRank-Efficient reduces the computational burden by only applying InRank during the initial training stage. In the remaining evaluation on GPT-2, InRank-Efficient is only applied for the first epoch.

In the InRank-Efficient approach, once we determine the optimal rank r^* for UV using InRank, we parameterize W as a rank-r^* factorization of UV only, eliminating the need for representing a separated W_0. By removing W_0, we can reduce both memory usage and computational costs as it avoids additional matrix multiplication. Moreover, we can enhance efficiency by discarding the buffer size b once the optimal rank has been determined. We provide further details of InRank-Efficient in the appendix.

Comparison

We compare InRank and InRank-Efficient with a full-rank baseline using different sizes of GPT models on the WikiText-103 dataset. All methods are trained with the same hyperparameters, including the learning rate, weight decay, and the number of epochs. We use the Adam optimizer to train for 100 epochs with an initial learning rate of 0.001. All experiments are run using the same computational setting with 8 NVIDIA® Tesla® V100 GPUs.

As shown in Table 3.1, both InRank and InRank-Efficient achieve validation perplexity comparable to the full-rank baseline while requiring at most 33% of the total rank. The rank is calculated as the average rank across all weight matrices in the model. We observed that InRank outperforms InRank-Efficient, even though they find the same rank. This can be attributed to the fact that InRank-Efficient discards the parameterization of W_0 during the training process.

We also measure several efficiency metrics to compare the computational efficiency of different methods. Specifically, we measure the total training time, memory usage, number of parameters, and the number of floating point operations (FLOPs) required for training.

Notably, InRank-Efficient significantly reduces both computational cost and memory usage. For instance, when compared to the baseline on GPT-medium, InRank-Efficient reduces the total training time by 20% and memory usage by 37%. On the other hand, throughout the entire training process, InRank-Efficient requires a maximum of 63% memory usage, enabling the training of large language models from scratch on memory-constrained devices.

Moreover, InRank-Efficient demonstrates even greater efficiency benefits with larger models. In the case of GPT-large, InRank-Efficient reduces 75% of the total rank, resulting in a reduction of 30% in training time and 42% in memory usage when measured over a single epoch. Unfortunately, due to our limited computational resources, we were unable to report its performance over a full training run. Additional results and discussions are provided in the appendix for further reference.

Selection of Threshold α

To determine the optimal configuration for InRank, we conduct evaluations using various values of threshold α. Table 3.2 demonstrates that the performance of each threshold value is consistent across different model sizes. Taking both prediction performance and efficiency into consideration, we have selected $\alpha = 0.9$ as the

Table 3.1: Performance comparison of different methods.

Model	Method	PPL	Rank	Memory	Params
GPT-small	Baseline	**18.5**	768	248Mb	124M
	InRank	18.6	**254**	295Mb	147.7M
	InRank-Efficient	18.9	**254**	**182Mb**	**91.2M**
GPT-medium	Baseline	**19.5**	1024	709Mb	355M
	InRank	19.6	**286**	850Mb	424M
	InRank-Efficient	19.9	**286**	**447Mb**	**223M**

Table 3.2: Varying threshold α in InRank-Efficient.

Threshold α	**GPT-small**					
	PPL	**Rank**	**Runtime**	**Memory**	**Parameters**	**FLOPs**
Baseline	18.5	768	24.5h	248Mb	124M	292G
0.8	19.4	152	20.2h	163Mb	81.8M	156G
0.85	19.1	193	21.9h	171Mb	85.6M	165G
0.9	18.9	254	22.2h	182Mb	91.2M	178G

Threshold α	**GPT-medium**					
	PPL	**Rank**	**Runtime**	**Memory**	**Parameters**	**FLOPs**
Baseline	19.5	1024	60.5h	709Mb	355M	828G
0.8	20.6	168	45.9h	389Mb	194M	363G
0.85	20.2	213	48.1h	411Mb	205M	389G
0.9	19.9	286	48.6h	447Mb	223M	428G

default value for all experiments. The stable choice of α ensures that InRank can automatically identify the optimal rank for new tasks and models without the need for extensive tuning, thereby minimizing the associated costs.

3.7 Conclusion

In this work, we generalize the Greedy Low-Rank Learning (GLRL) to arbitrary orthogonal initialization, leading to the development of Incremental Low-Rank Learning (InRank). Our method is capable of discovering the intrinsic rank of networks and has demonstrated comparable performance to full-rank counterparts on training GPT-2, while utilizing a maximum of 33% of total ranks throughout training. The efficient variant of InRank also achieves a significant reduction of 20% in total training time and 37% in memory usage when training GPT-medium on WikiText-103.

We believe our work offers a novel approach to training low-rank networks through automatic rank determination. In the future, we aim to expand our method to encompass various network architectures and datasets. Additionally, we intend to optimize our algorithm implementation to further improve its computational efficiency.

References

Arora, Sanjeev et al. (Oct. 2019). "Implicit Regularization in Deep Matrix Factorization". In: *arXiv:1905.13655 [cs, stat]*. arXiv: 1905.13655. URL: http://arxiv.org/abs/1905.13655 (visited on 04/12/2022).

Dao, Tri et al. (Apr. 2022). *Monarch: Expressive Structured Matrices for Efficient and Accurate Training*. en. arXiv:2204.00595 [cs]. URL: http://arxiv.org/abs/2204.00595 (visited on 05/15/2023).

Gissin, Daniel, Shai Shalev-Shwartz, and Amit Daniely (Dec. 2019). *The Implicit Bias of Depth: How Incremental Learning Drives Generalization*. en. arXiv:1909.12051 [cs, stat]. URL: http://arxiv.org/abs/1909.12051 (visited on 04/26/2023).

Gunasekar, Suriya et al. (July 2018). "Characterizing Implicit Bias in Terms of Optimization Geometry". en. In: *Proceedings of the 35th International Conference on Machine Learning*. ISSN: 2640-3498. PMLR, pp. 1832–1841. URL: https://proceedings.mlr.press/v80/gunasekar18a.html (visited on 04/29/2023).

He, Kaiming et al. (2015). "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification". In: *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*. IEEE Computer Society.

Hu, Edward J. et al. (Oct. 2021). *LoRA: Low-Rank Adaptation of Large Language Models*. en. arXiv:2106.09685 [cs]. URL: http://arxiv.org/abs/2106.09685 (visited on 12/05/2022).

Huang, Xinquan and Tariq Alkhalifah (Sept. 2021). "PINNup: Robust neural network wavefield solutions using frequency upscaling and neuron splitting". In: *arXiv:2109.14536 [physics]*. arXiv: 2109.14536. URL: http://arxiv.org/abs/2109.14536 (visited on 05/13/2022).

Idelbayev, Yerlan and Miguel A. Carreira-Perpinan (2020). "Low-Rank Compression of Neural Nets: Learning the Rank of Each Layer". In: pp. 8049–8059. URL: https://openaccess.thecvf.com/content_CVPR_2020/html/Idelbayev_Low-Rank_Compression_of_Neural_Nets_Learning_the_Rank_of_Each_CVPR_2020_paper.html (visited on 05/13/2023).

Ioannou, Yani et al. (Feb. 2016). *Training CNNs with Low-Rank Filters for Efficient Image Classification*. arXiv:1511.06744 [cs]. DOI: 10.48550/arXiv.1511.06744. URL: http://arxiv.org/abs/1511.06744 (visited on 04/29/2023).

Jacot, Arthur et al. (Jan. 2022). "Saddle-to-Saddle Dynamics in Deep Linear Networks: Small Initialization Training, Symmetry, and Sparsity". en. In: *arXiv:2106.15933 [cs, stat]*. arXiv: 2106.15933. URL: http://arxiv.org/abs/2106.15933 (visited on 02/21/2022).

Li, Chunyuan et al. (2018). "Measuring the Intrinsic Dimension of Objective Landscapes". In: *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net.

Li, Zhiyuan, Yuping Luo, and Kaifeng Lyu (Apr. 2021). "Towards Resolving the Implicit Bias of Gradient Descent for Matrix Factorization: Greedy Low-Rank Learning". In: *arXiv:2012.09839 [cs, stat]*. arXiv: 2012.09839. URL: http://arxiv.org/abs/2012.09839 (visited on 02/25/2022).

Rahaman, Nasim et al. (May 2019). *On the Spectral Bias of Neural Networks*. Tech. rep. arXiv:1806.08734. arXiv:1806.08734 [cs, stat] type: article. arXiv. DOI: 10.48550/arXiv.1806.08734. URL: http://arxiv.org/abs/1806.08734 (visited on 07/09/2022).

Razin, Noam, Asaf Maman, and Nadav Cohen (June 2021). "Implicit Regularization in Tensor Factorization". In: *arXiv:2102.09972 [cs, stat]*. arXiv: 2102.09972. URL: http://arxiv.org/abs/2102.09972 (visited on 04/12/2022).

Saxe, Andrew M., James L. McClelland, and Surya Ganguli (Feb. 2014). "Exact solutions to the nonlinear dynamics of learning in deep linear neural networks". In: *arXiv:1312.6120 [cond-mat, q-bio, stat]*. arXiv: 1312.6120. URL: http://arxiv.org/abs/1312.6120 (visited on 04/20/2021).

Schotthöfer, Steffen et al. (May 2022). *Low-rank lottery tickets: finding efficient low-rank neural networks via matrix differential equations*. en. arXiv:2205.13571 [cs, math, stat]. (Visited on 07/23/2022).

Vogels, Thijs, Sai Praneeth Karimireddy, and Martin Jaggi (2019). "PowerSGD: Practical Low-Rank Gradient Compression for Distributed Optimization". In: *Advances in Neural Information Processing Systems*. Vol. 32. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper/2019/hash/d9fbed9da256e344c1fa46bb46c34c5f-Abstract.html (visited on 05/10/2022).

Wang, Hongyi, Saurabh Agarwal, and Dimitris Papailiopoulos (n.d.). "Pufferfish: Communication-efficient Models At No Extra Cost". en. In: (), p. 22.

Wang, Hongyi, Saurabh Agarwal, Pongsakorn U-chupala, et al. (May 2023). *Cuttlefish: Low-rank Model Training without All The Tuning*. arXiv:2305.02538 [cs]. DOI: 10.48550/arXiv.2305.02538. URL: http://arxiv.org/abs/2305.02538 (visited on 05/07/2023).

Yang, Huanrui et al. (June 2020). "Learning Low-rank Deep Neural Networks via Singular Vector Orthogonality Regularization and Singular Value Sparsification". en. In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. Seattle, WA, USA: IEEE, pp. 2899–2908. ISBN: 978-1-72819-360-1. DOI: 10.1109/CVPRW50498.2020.00347. URL: https://ieeexplore.ieee.org/document/9150852/ (visited on 04/03/2022).

You, Haoran et al. (Feb. 2022). *Drawing Early-Bird Tickets: Towards More Efficient Training of Deep Networks*. arXiv:1909.11957 [cs, stat]. DOI: 10.48550/arXiv.1909.11957. URL: http://arxiv.org/abs/1909.11957 (visited on 05/15/2023).

Zhao, Jiawei, Robert Joseph George, et al. (Nov. 2022). *Incremental Spectral Learning in Fourier Neural Operator*. en. URL: https://arxiv.org/abs/2211.15188v3 (visited on 05/09/2023).

Zhao, Jiawei, Florian Schäfer, and Anima Anandkumar (Oct. 2021). *ZerO Initialization: Initializing Residual Networks with only Zeros and Ones*. arXiv:2110.12661 [cs]. DOI: 10.48550/arXiv.2110.12661. URL: http://arxiv.org/abs/2110.12661 (visited on 08/10/2022).

Chapter 4

ZERO INITIALIZATION: INITIALIZING NEURAL NETWORKS WITH ONLY ZEROS AND ONES

4.1 Introduction

An important question in training deep neural networks is how to initialize the weights. Currently, random weight initialization is the de-facto practice across all architectures and tasks. However, choosing the *variance* of the initial weight distribution is a delicate balance when training deep neural networks. If the variance is too large, it can lead to an excessive amplification of the activations propagating through the network during training, resulting in *exploding gradients*. On the other hand, if the weights are initialized too small, the activations may not propagate at all, resulting in *vanishing gradients*. These issues become more challenging as the number of layers in the network grows.

The above challenges can be avoided if *identity initialization* is used instead. It initializes each layer in the network as an identity mapping, such that the input data can be identically propagated to the network output. In this case, there is no need to introduce any randomness or consider its variance. Identity initialization is well studied theoretically from an optimization perspective. Hardt and Ma (2017) prove the existence of a global minimum close to the identity parameterization in a deep residual network. Bartlett, Helmbold, and Long (2019) further prove the rate of convergence of gradient-based optimization under identity initialization.

However, prior theoretical works on identity initialization assume that all layers had the same dimensionality, which does not hold for practical networks. Typically, practical networks have *varying dimensionality* across layers, such as the variations of spatial and channel dimensions in ResNet architectures (He et al., 2016). Directly applying identity initialization to these networks leads to a problem of *training degeneracy*, as our theoretical study will demonstrate later.

To avoid the training degeneracy, previous works employ identity initialization with *random perturbations* to facilitate escapes from a saddle point or to break feature symmetry (Blumenfeld, Gilboa, and Soudry, 2020). Broadly, these approaches satisfy the property of *dynamical isometry*, to preserve the signal propagation and ensure well-behaved gradients at initialization (Saxe, McClelland, and Ganguli,

2014). Despite the efficacy of random perturbations, they inevitably introduce additional tuning of variances, which can result in gradient explosion in deep networks without careful tuning.

We propose ZerO initialization that removes *all randomness in the weight initialization*. As illustrated in Figure 4.1, ZerO initializes networks with Hadamard and identity transforms, which assigns all the weights to only *zeros and ones*.

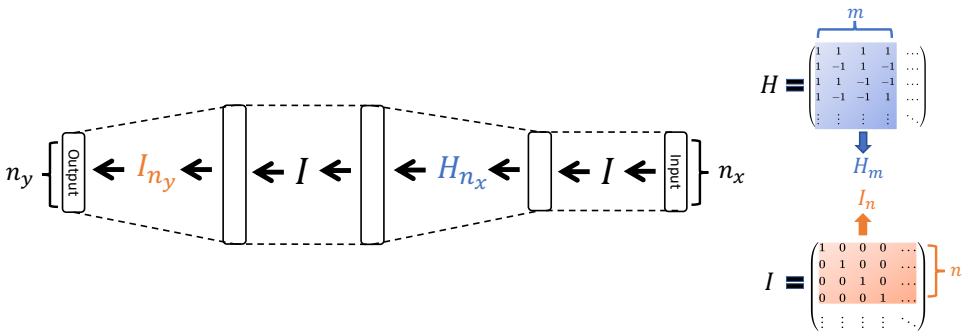

Figure 4.1: Illustrating ZerO in a 3-layer network with input dimension n_x, hidden dimension n_h, and output dimension n_y, where $n_h > n_x, n_y$. $\boldsymbol{H}$ and $\boldsymbol{I}$ are $n_h \times n_h$ Hadamard and identity matrix, respectively. The dimension-increasing layer is initialized by columns of the Hadamard matrix. The rest layers are initialized by identity matrix or rows of it.

ZerO is not affected by the problem of training degeneracy or accuracy loss. Compared to random initialization, ZerO provides state-of-the-art performance over various image classification tasks, including ImageNet. We further discover many unique properties and benefits of ZerO:

Stable training without batch normalization. ZerO ensures well-behaved signal propagation, which provides stable training without batch normalization. Testing ResNet with over 500 layers, we find ZerO converges faster than carefully designed random methods, such as Fixup and ReZero (Zhang, Dauphin, and Ma, 2019; Bachlechner et al., 2021).

Low-rank learning trajectory. We find that ZerO exhibits a low-rank learning trajectory, where the rank of each matrix gradually increases during training. We believe this is the first time that the greedy low-rank learning (GLRL) trajectory, a theoretical characterization of gradient descent, has been observed in large-scale deep learning applications. GLRL is a consequence of implicit rank regularization by gradient descent under infinitesimal initialization (Li, Luo, and Lyu, 2021; Razin, Maman, and Cohen, 2021). It can be viewed as performing a rank-constrained optimization and greedily relaxing the rank restriction by one whenever it fails to reach

a global minimizer. GLRL has been used to explain the excellent generalization in gradient-based deep learning, as it converges to a global (or local) minima with the minimum rank. However, the GLRL trajectory has never been observed in practice due to its impractical requirement of infinitesimal initialization.

Sparse and low-rank solutions. We observe that ZerO-initialized networks converge to sparse and low-rank solutions. Compared to randomly initialized networks, the sub-networks obtained in trained ZerO-initialized networks achieve 30% lower (matrix or tensor) rank or 25% higher sparsity without sacrificing accuracy.

Better training reproducibility. Since ZerO does not require any random perturbations, it is a fully deterministic initialization scheme. Unlike determinism in random initialization, which needs fixing pseudorandom number generators in hardware, the weights initialized by ZerO are fixed regardless of how the random seed varies or which hardware is used. ZerO significantly reduces the training variation and thus achieves better training reproducibility (the remaining randomness is only due to batch selection). Compared to random initialization, ZerO produces 20%-40% lower standard deviation of the final accuracy over repeated experiments with different random seeds.

Theoretical analysis of ZerO. Theoretically, we demonstrate that ZerO breaks a training degeneracy that arises when applying identity initialization to networks with varying dimensionality across layers. We prove that the training degeneracy necessarily occurs in standard identity initialization because the rank of any $n_h \times n_h$ matrix in the network is upper bounded by the input and output dimensions n_x and n_y throughout the entire training, no matter how large the size of n_h is. This limits the expressivity of each matrix, resulting in the degeneracy of training.

Our contributions are summarized as follows:

1. We design ZerO initialization, the first fully deterministic initialization that achieves state-of-the-art performance in practice.

2. ZerO is backed with theoretical understanding. As shown in Theorem 4.3.3, we prove how ZerO breaks the training degeneracy by applying Hadamard transforms.

3. ZerO has many benefits, such as training ultra deep networks (without batch-normalization), exhibiting low-rank learning trajectory, converging to sparse and low-rank solutions, and improving training reproducibility.

4.2 Related Works

To ensure stable training with random weight initialization, previous works such as Glorot and Bengio (n.d.) and He et al. (2015) study the propagation of variance in the forward and backward pass under different activations. Several studies provide a more detailed characterization of the signal propagation with dynamical isometry (Saxe, McClelland, and Ganguli, 2014; Pennington, Schoenholz, and Ganguli, 2017; Xiao et al., 2018).

Inspired by the dynamical isometry property, various initialization methods are proposed to increase the convergence speed and stabilize the signal propagation, including Saxe, McClelland, and Ganguli (2014), Bachlechner et al. (2021), Gehring et al. (2017), and Balduzzi et al. (2017). De and Smith (2020) and Hoffer et al. (2019) study the reason behind the success of batch normalization (Ioffe and Szegedy, 2015) , and Zhang, Dauphin, and Ma (2019) and De and Smith (2020) propose initialization methods to train residual networks without batch normalization.

Many of the previous methods can be categorized as identity initialization with random perturbations. However, ZerO eliminates all the randomness using Hadamard and identity transforms. In another related work, Blumenfeld, Gilboa, and Soudry (2020) discusses whether random initialization is needed from the perspective of feature diversity. They propose networks with identical features at initialization, which still require random perturbations to avoid the symmetric problem and improve performance.

Gradient descent biasing models towards low-rank solutions has been well studied in matrix factorization (Arora et al., 2019). Recent works also demonstrate the existence of greedy low-rank learning trajectory induced by gradient descent (Li, Luo, and Lyu, 2021; Razin, Maman, and Cohen, 2021; Jacot et al., 2022). However, no prior work demonstrates the greedy low-rank learning trajectory in large-scale applications of deep learning, as most only consider the problems of matrix factorization or applications of shallow neural networks.

4.3 Is Randomness Necessary in Identity Initialization?

Background

We wish to train a function $\mathcal{F}(\boldsymbol{x})$ to learn a particular input-output map given a set of P training samples $(\boldsymbol{x}^\mu, \boldsymbol{y}^\mu) \in \mathbb{R}^{n_x \times n_y}$, where $\mu = 1, ..., P$. Training is accomplished by minimizing the squared error $\mathcal{L} = \frac{1}{2} \sum_{\mu=1}^{P} \|\boldsymbol{y}^\mu - \mathcal{F}(\boldsymbol{x}^\mu)\|_2^2$ using gradient descent with a step size η.

We model $\mathcal{F}(\boldsymbol{x})$ to be a multilayer perceptron with $L > 2$ hidden layers, such that:

$$\boldsymbol{x}_l = \boldsymbol{W}_l \boldsymbol{z}_{l-1} \qquad \boldsymbol{z}_l = \varphi(\boldsymbol{x}_l),$$

with $l \in 1, ..., L$. Let $\boldsymbol{z}_0 = \boldsymbol{x}$ and $\mathcal{F}(\boldsymbol{x}) = \boldsymbol{z}_L$. φ is an element-wise nonlinearity. We assume that $\mathcal{F}$ has uniform hidden dimension n_h, with $\boldsymbol{W}_l \in \mathbb{R}^{n_h \times n_h}$ for $l \in 2, ..., L-1$, $\boldsymbol{W}_1 \in \mathbb{R}^{n_h \times n_x}$, and $\boldsymbol{W}_L \in \mathbb{R}^{n_y \times n_h}$.

The input-output Jacobian is a well-studied proxy for estimating the stability of signal propagation at initialization, which is defined as: $\boldsymbol{J}_{io} = \frac{\partial \boldsymbol{z}_L}{\partial \boldsymbol{z}_0}$. Proposed by (Saxe, McClelland, and Ganguli, 2014), dynamical isometry is a condition where all singular values of the Jacobian $\boldsymbol{J}_{io}$ concentrate near 1. If $\boldsymbol{J}_{io}$ is well-conditioned, the backpropagation error $\frac{\partial \mathcal{L}}{\partial \boldsymbol{z}_l}$ at any layer l will be well-conditioned as well. This ensures stable signal propagation and well-behaved gradients at initialization.

Consider the case of $n_x = n_y = n_h$. Identity initialization is defined as initializing each matrix to be an identity matrix: $\boldsymbol{W}_l = \boldsymbol{I}$. In this case, the dynamical isometry property for a linear $\mathcal{F}$ can be easily verified as $\boldsymbol{J}_{io} = \boldsymbol{I}$. It also holds for certain nonlinearities when applying the identity initialization on residual networks: $\boldsymbol{z}_l = \varphi(\boldsymbol{x}_l) + \boldsymbol{x}_{l-1}$ where $\boldsymbol{W}_l = 0$, such that no signal is passed through the residual branches at initialization.

From an optimization perspective, Hardt and Ma (2017) suggest that $\mathcal{F}$ has a global minimum very close to its identity initialization, such that $\max_{1 \le l \le L} \left\|\boldsymbol{W}_l'\right\| \le O(1/L)$ for large L, where $\boldsymbol{W}' = \boldsymbol{W} - \boldsymbol{I}$. Bartlett, Helmbold, and Long (2019) also proves that under the identity initialization, gradient descent learns an ϵ-approximation of $\mathcal{F}$ within iterations polynomial in $log(1/\epsilon)$.

Extending to Large Hidden Dimension

So far, we have discussed identity initialization in the special case of fixed dimensionality. Now we extend our discussion to a more practical setting where $\mathcal{F}$ is equipped with a large hidden dimension, such that $n_h >> n_x, n_y$. We also focus on the specific case where φ is a Relu nonlinearity.

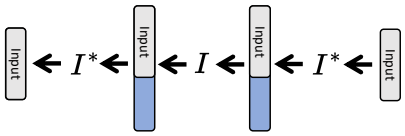

Figure 4.2: A 3-layer network $\mathcal{F}$ ($n_h > n_x = n_y$) where $\boldsymbol{W}_1, \boldsymbol{W}_3 = \boldsymbol{I}^*$ and $\boldsymbol{W}_2 = \boldsymbol{I}$ at initialization.

A straightforward approach to identity initialization in this case is to initialize $\boldsymbol{W}_1$ such that it projects the input into a n_x-dimensional subspace of the n_h-dimensional hidden space. This can be achieved by initializing $\boldsymbol{W}_1$ with a *partial identity matrix*:

Definition 4.3.1 (Partial Identity Matrix) *Let $\boldsymbol{I}^* \in \mathbb{R}^{l \times r}$, the partial identity matrix $\boldsymbol{I}^*$ is defined as follows:*

$$\boldsymbol{I}^* = \begin{cases} (\boldsymbol{I}, \boldsymbol{0}) & \text{where } \boldsymbol{I} \in \mathbb{R}^{l,l} \text{ and } \boldsymbol{0} \in \mathbb{R}^{l,r-l} \quad \text{if } l < r, \\ \boldsymbol{I} & \text{where } \boldsymbol{I} \in \mathbb{R}^{l,l} \quad \text{if } l = r, \\ (\boldsymbol{I}, \boldsymbol{0})^\top & \text{where } \boldsymbol{I} \in \mathbb{R}^{r,r} \text{ and } \boldsymbol{0} \in \mathbb{R}^{r,l-r} \quad \text{otherwise.} \end{cases}$$

For a vector $\boldsymbol{a} \in \mathbb{R}^r$, if $l < r$, then $\boldsymbol{I}^*(\boldsymbol{a})$ clips the last few dimension such that $\boldsymbol{I}^*(\boldsymbol{a}) = (a_1, a_2, ..., a_l)^\top$. If $l > r$, then $\boldsymbol{I}^*$ pads $l - r$ additional dimensions with zero, such that $(a_1, a_2, ..., a_r, 0...0)^\top$. This is also known as a zero-padding operator, such as used in channel-expanding layers in ResNet (He et al., 2016). In the network $\mathcal{F}$, $\boldsymbol{I}^*$ only needs to be applied in the dimension-varying matrices $\boldsymbol{W}_1$ and $\boldsymbol{W}_L$, while leaving the remaining $n_h \times n_h$ matrices to be identity matrix $\boldsymbol{I}$.

We visualize this process in Figure 4.2. This may seem like a natural extension of identity initialization to a large width setting, but we will show in Section 4.3 it suffers from a problem we call "training degeneracy". To avoid the problem, we use the Hadamard matrix $\boldsymbol{H}$ to initialize the dimension-increasing matrices, such that $\boldsymbol{W}_1 = \boldsymbol{H}\boldsymbol{I}^*$. A Hadamard matrix is defined as follows:

Definition 4.3.2 (Hadamard matrix) *For any Hadamard matrix $\boldsymbol{H} = \boldsymbol{H}_m \in \mathbb{R}^{2^m \times 2^m}$ where m is a positive integer, we define $\boldsymbol{H}_0 = 1$ by the identity, and the matrix with large m is defined recursively:*

$$\boldsymbol{H}_m = \begin{pmatrix} \boldsymbol{H}_{m-1} & \boldsymbol{H}_{m-1} \\ \boldsymbol{H}_{m-1} & -\boldsymbol{H}_{m-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 & \dots \\ 1 & -1 & 1 & -1 & \dots \\ 1 & 1 & -1 & -1 & \dots \\ 1 & -1 & -1 & 1 & \dots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix} \in \mathbb{R}^{2^m \times 2^m}.$$

The linear transformation described by the Hadamard matrix, called the Hadamard transform, rotates the coordinate axes to be equally weakly aligned with the standard basis. For example, in a two-dimensional plane, the Hadamard transform rotates the standard basis by an exact angle of 45 degree. This turns out to be an important property for breaking the training degeneracy.

Identity Initialization limits Network Expressivity

We now present our main result differentiating the training behavior of different initialization methods and describing the problem of training degeneracy.

Theorem 4.3.3 *Let $\mathcal{F}$ be a neural network with L matrices, where $\boldsymbol{W}_l \in \mathbb{R}^{n_h \times n_h}$ for $l \in 2, ..., L-1$, $\boldsymbol{W}_1 \in \mathbb{R}^{n_h \times n_x}$, and $\boldsymbol{W}_L \in \mathbb{R}^{n_y \times n_h}$. $\mathcal{F}$ has a uniform hidden dimension n_h, input dimension n_x, and output dimension n_y, where $n_h \geq n_x, n_y$. Define residual component $W'_l = W_l - \boldsymbol{I}$. Let $\boldsymbol{z}_l(\boldsymbol{x})$ to be the activation in the l-th layer under the input $\boldsymbol{x}$. Then we have the following results for different initializations:*

1. *Consider a random perturbation $\mu \in \mathbb{R}^{n_h \times n_h}$ where each element is sampled from a Gaussian distribution: $\mu_{ij} \sim \mathcal{N}(0, \sigma^2)$. It is well-known that the randomly perturbed matrices $\boldsymbol{W}_l = \boldsymbol{I} + \mu_l$ (for $l \neq 1, L$) are full-rank almost surely:*

$$Prob(\text{rank}(\boldsymbol{W}'_l) = n_h) = 1 \quad \text{for } l \in 2, ..., L-1. \tag{4.1}$$

2. *When initializing $\boldsymbol{W}_1, \boldsymbol{W}_L = \boldsymbol{I}^*$ and the remaining matrices as $\boldsymbol{W}_l = \boldsymbol{I}$ (for $l \neq 1, L$), for any $\boldsymbol{x} \in \mathbb{R}^{n_x}$, the linear dimension of the set of all possible activations is bounded throughout training as*

$$\dim\left(\text{span}\left(\{\boldsymbol{z}_l(\boldsymbol{x}) | \boldsymbol{x} \in \mathbb{R}^{n_x}\}\right)\right) \leq n_x \quad \text{for } l \in 2, ..., L-1. \tag{4.2}$$

   As a result, the ranks of the weight matrices remain bounded throughout training as

$$\text{rank}(\boldsymbol{W}'_l) \leq n_x \quad \text{for } l \in 2, ..., L-1. \tag{4.3}$$

3. *When initializing $\boldsymbol{W}_1 = \boldsymbol{H}\boldsymbol{I}^*$, $\boldsymbol{W}_L = \boldsymbol{I}^*$, and the remaining matrices as $\boldsymbol{W}_l = \boldsymbol{I}$ (for $l \neq 1, L$) it is possible for the activations at an intermediate layer to attain*

$$\dim\left(\text{span}\left(\{\boldsymbol{z}_l(\boldsymbol{x}) | \boldsymbol{x} \in \mathbb{R}^{n_x}\}\right)\right) > n_x, \tag{4.4}$$

   breaking the constraint on the linear dimension described in Equation 4.2.

Remark 1 *The constraints on linear dimensions and matrix ranks in Equation 4.2 and 4.3 suggest that no matter how large hidden dimension n_h is, the network $\mathcal{F}$ is only optimized within a low-dimensional subspace (depending on input dimension n_x) of the full parameter space. This restricts the maximum network expressivity of $\mathcal{F}$, and thus the training may only converge to an underfitting regime, leading to* ***training degeneracy****.*

Remark 2 *Under the assumptions of 3, the breaking of training degeneracy described in Equation 4.4 appears to be the generic case. As verified empirically in Figure 4.3, applying the Hadamard transform in 2 also breaks the rank constraint in Equation 4.3.*

Remark 3 *1 and 3 avoid the training degeneracy from different directions. Unlike 1, the proposed 3 doesn't introduce any randomness with the help of the Hadamard transform.*

Almost all existing works use random weight initialization, which largely affects the rank of each matrix as shown in 1. 1 can be proved by showing any column (or row) in a random matrix is linearly independent to the other columns (or rows), almost surely. A detailed proof can be found in the appendix.

Consider identity initialization without any randomness. In 2, $\boldsymbol{W}_1$ identically maps the input $\boldsymbol{x}$ into a subspace of the hidden layer $\boldsymbol{z}_1$, such that $\boldsymbol{z}_1 = (\boldsymbol{x}^\top, 0, ..., 0)^\top$. Thus, the linear dimension on $\boldsymbol{z}_l$ (i.e., the linear dimension on activations $\boldsymbol{z}_l^1, ..., \boldsymbol{z}_l^P$) is equivalent to the linear dimension on $\boldsymbol{x}$, which is upper bounded by n_x. This result is held for every layer $\boldsymbol{z}_l$ (where $l \neq 1, L$).

To show the rank constraint in Equation 4.3, we track a single derivative of the residual component at layer l:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}_l'} = \sum_{\mu=1}^{P} \frac{\partial \mathcal{L}}{\partial \boldsymbol{x}_l^\mu} \otimes \boldsymbol{z}_{l-1}^\mu, \tag{4.5}$$

where $\otimes$ denotes the outer product. We use the following well-known fact:

Lemma 4.3.4 *Consider a matrix $\boldsymbol{M}$ to be a sum of vector outer products:$\boldsymbol{M} = \sum_{\mu=1}^{Q} \boldsymbol{a}^\mu \otimes \boldsymbol{b}^\mu$, where $\boldsymbol{a}^\mu \in \mathbb{R}^{n_a}$ and $\boldsymbol{b}^\mu \in \mathbb{R}^{n_b}$ for $\mu \in 1, ..., Q$. Let $Q > n_a, n_b$.*

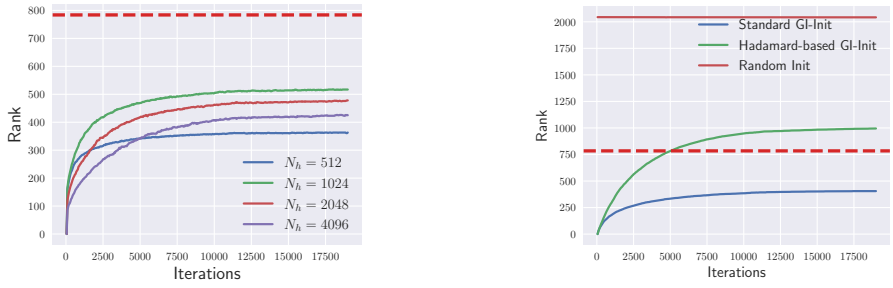

Figure 4.3: *Verify Theorem 4.3.3 in practice. We train a network $\mathcal{F}$ with $L = 3$ on MNIST, and visualize* $\text{rank}(W_2)$ *over the training. The red dash line denotes the rank constraint on W_2 predicted by the theorem, which is $n_x = 784$. **Left:** we verify 2 in the theorem by varying n_h. No matter how large n_h is,* $\text{rank}(W_2)$ *follows the rank constraint through the entire training. **Right:** we verify 3 where applying Hadamard transform breaks the rank constraint introduced in 2 , given $n_h = 2048$. We denote the initializations in 2 and 3 as standard and Hadamard-based GI-Init, respectively. As predicted in 1, random initialization achieves its maximum rank immediately after the initialization.*

If linear dimensions $\dim(\text{span}(\boldsymbol{a}^1, ..., \boldsymbol{a}^Q)) \leq U$ *and* $\dim(\text{span}(\boldsymbol{b}^1, ..., \boldsymbol{b}^Q)) \leq V$, *where $U \leq n_a$ and $V \leq n_b$, then:*

$$\text{rank}(\boldsymbol{W}) \leq \min(U, V).$$

By Lemma 4.3.4, at initialization, the upper bound n_x on the linear dimension on $\boldsymbol{z}_{l-1}$ results in a rank constraint on $\text{rank}(\boldsymbol{W}_l')$. The rank constraint holds during the entire training as $\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}_l'}$ has a zero-valued $n_y \times (n_h - n_x)$ sub-matrix at every iteration (as shown in the appendix). Since $W_l' = \sum_{t=1}^{T} -\eta \frac{\partial \mathcal{L}}{\partial \boldsymbol{W}_l'}\Big|_t$ after T weight updates (by gradient descent with a step size η), $\text{rank}(W_l')$ is bounded by n_x no matter what T is. This results in the training degeneracy as described in Remark 1. We also verify it empirically in Figure 4.3.

To avoid the training degeneracy, we need to overcome limitations on the linear dimension of the set of possible activations. This is indeed possible when using the Hadamard matrix as $\boldsymbol{W}_1 = \boldsymbol{H}\boldsymbol{I}^*$, as we will illustrate it by means of an example.

Lemma 4.3.5 *Assume $n_h = 4$ and $n_x = 3$. For any vector $\boldsymbol{x} \in \text{span}(\boldsymbol{e}_2, \boldsymbol{e}_3)$ where $\boldsymbol{e}_2$ and $\boldsymbol{e}_3$ are coordinate vectors $(0, 1, 0)^\top$ and $(0, 0, 1)^\top$, it holds that:*

$$\text{span}\left(\{\boldsymbol{z}_1(\boldsymbol{x}) | \boldsymbol{x} \in \mathbb{R}^{n_x}\}\right) = \text{span}(\text{Relu}(\boldsymbol{H}\boldsymbol{I}^*\boldsymbol{e}_2), \text{Relu}(-\boldsymbol{H}\boldsymbol{I}^*\boldsymbol{e}_2), \text{Relu}(\boldsymbol{H}\boldsymbol{I}^*\boldsymbol{e}_3), \text{Relu}(-\boldsymbol{H}\boldsymbol{I}^*\boldsymbol{e}_3)), \tag{4.6}$$

where $\text{Relu}(\boldsymbol{H}\boldsymbol{I}^*\boldsymbol{e}_2)$, $\text{Relu}(-\boldsymbol{H}\boldsymbol{I}^*\boldsymbol{e}_2)$, $\text{Relu}(\boldsymbol{H}\boldsymbol{I}^*\boldsymbol{e}_3)$, *and* $\text{Relu}(-\boldsymbol{H}\boldsymbol{I}^*\boldsymbol{e}_3)$ *are linearly independent. This indicates that:*

$$\dim\left(\text{span}\left(\{\boldsymbol{z}_1(\boldsymbol{x})|\boldsymbol{x}\in\mathbb{R}^{n_x}\}\right)\right) = 4 = n_h > n_x = 3.$$

When using Hadamard matrix as $\boldsymbol{W}_1 = \boldsymbol{H}\boldsymbol{I}^*$, the breaking of the training degeneracy described in Lemma 4.3.5 appears to be the generic case. As verified empirically in Figure 4.3, this also breaks the rank constraint in Equation 4.3.

We point out that the increase of the linear dimension of the set of possible $\boldsymbol{z}_l$ is only possible due to the nonlinearity. If $\mathcal{F}$ is a linear network, the linear dimension on every $\boldsymbol{z}_l$ is at most n_x, no matter how the weights are initialized.

Nevertheless, the nonlinearity can not increase the linear dimensionality if we initialize the network with a partial identity matrix. This is because when $\boldsymbol{W}_1 = \boldsymbol{I}^*$, $\text{span}\left(\{\boldsymbol{z}_l(\boldsymbol{x})|\boldsymbol{x}\in\mathbb{R}^{n_x}\}\right)$ is aligned with the standard basis, i.e., each vector in the span at least has $n_h - n_x$ zero coefficients when expressed in the standard basis. Thus, an element-wise nonlinearity can not increase the linear dimension of its input beyond n_x.

To break the alignment $\text{span}\left(\{\boldsymbol{z}_l(\boldsymbol{x})|\boldsymbol{x}\in\mathbb{R}^{n_x}\}\right)$ with the standard basis, we use the Hadamard transform. This is because it transforms the subspace such that the new basis is equally weakly aligned with the standard basis. We note that other linear transforms may also detach the subspace from the standard basis, but the Hadamard transform is the most natural choice.

4.4 ZerO Initialization

The initialization analyzed in 3 of Theorem 4.3.3 is based on a network condition in which all hidden spaces $\boldsymbol{z}_1, ..., \boldsymbol{z}_{L-1}$ have the same dimension n_h. Motivated by our theoretical understanding, we propose ZerO initialization, which initializes the weights of any network with arbitrary hidden dimensions. As described in Algorithm 4, ZerO only initializes dimensional-increasing layers with Hadamard matrices to avoid the training degeneracy. Other layers are simply initialized by (partial) identity matrices. We also rescale Hadamard matrices by a normalization factor $2^{-(m-1)/2}$, resulting in an orthonormal Hadamard transform.

We also apply ZerO to the well-developed ResNet architectures in He et al. (2016). As shown in Algorithm 5, we apply ZerO to convolution in a similar way by considering the variation in channel dimensions. When $\boldsymbol{K}$ is a 1x1 convolution, $\boldsymbol{K}$ also can

Algorithm 4 ZerO Initialization

Input: A neural network $\mathcal{F}$ with L matrices $\boldsymbol{W}_l \in \mathbb{R}^{P_l \times Q_l}$ for l in $1, ..., L$. $\boldsymbol{I}^*$ is the partial identity matrix (see Definition 4.3.1). $\boldsymbol{H}_m$ is the Hadamard matrix (see Definition 4.3.2).

for $l = 1$ **to** L **do**
  if $P_l = Q_l$ **then**
    $\boldsymbol{W}_l \leftarrow \boldsymbol{I}$ {Identity mapping}
  else if $P_l < Q_l$ **then**
    $\boldsymbol{W}_l \leftarrow \boldsymbol{I}^*$ {Propagate the first P_l dimensions}
  else if $P_l > Q_l$ **then**
    $m \leftarrow \lceil \log_2(P_l) \rceil$
    $c \leftarrow 2^{-(m-1)/2}$
    $\boldsymbol{W}_l \leftarrow c\, \boldsymbol{I}^* \boldsymbol{H}_m \boldsymbol{I}^*$ {Apply Hadamard matrix}
  end if
end for

Algorithm 5 ZerO Initialization on Convolution

Input: Number of input channels c_{in}, number of output channels c_{out}, odd kernel size k

Output: A $c_{out} \times c_{in} \times k \times k$ convolutional kernel $\boldsymbol{K}$

Initialize: $n \leftarrow \lfloor k/2 \rfloor$

if $c_{out} = c_{in}$ **then**
  $\boldsymbol{K}[:, :, n, n] \leftarrow \boldsymbol{I}$ {Identity mapping}
else if $c_{out} < c_{in}$ **then**
  $\boldsymbol{K}[:, :, n, n] \leftarrow \boldsymbol{I}^*$ {Propagate the first c_{out} channels}
else if $c_{out} > c_{in}$ **then**
  $m \leftarrow \lceil \log_2(c_{out}) \rceil$
  $c \leftarrow 2^{-(m-1)/2}$
  $\boldsymbol{K}[:, :, n, n] \leftarrow c\, \boldsymbol{I}^* \boldsymbol{H}_m \boldsymbol{I}^*$ {Apply Hadamard matrix}
end if

be viewed a $c_{out} \times c_{in}$ matrix, which matches the initialization in Algorithm 4. We note that Algorithm 5 can be applied to every convolution in ResNet, including the first 3x3 convolution, 1x1 convolutions in spatial-downsampling skip connections, and convolutions in basic block and bottleneck block.

To achieve dynamical isometry at initialization, we apply a common technique that initializes the last convolution in each residual block as zero. This helps suppress the signals from residual branches to stabilize signal propagations, as studied in Zhang, Dauphin, and Ma (2019) and Bachlechner et al. (2021).

We also apply ZerO to networks with or without batch normalization. For ResNet with batch normalization, we follow the standard practice to initialize the scale and

Dataset	**Model**	**Initialization**	**Test Error** (mean ± std)
CIFAR-10	ResNet-18	**ZerO Init**	5.13 ± 0.08
		Kaiming Init	5.15 ± 0.13
		Xavier Init	5.23 ± 0.16
ImageNet	ResNet-50	**ZerO Init**	23.43 ± 0.04
		Kaiming Init	23.46 ± 0.07
		Xavier Init	23.65 ± 0.11

Table 4.1: Benchmarking ZerO on CIFAR-10 and ImageNet. We repeat each run 10 times with different random seeds.

bias in batch normalization as one and zero, respectively. For training without batch normalization, we adopt a technique proposed by Zhang, Dauphin, and Ma (2019), where the batch normalization is replaced by learnable scalar multipliers and biases.

4.5 Experiments

In this section, we empirically benchmark ZerO on CIFAR-10 and ImageNet datasets, where we evaluate ResNet-18 on CIFAR-10 and ResNet-50 on ImageNet (Krizhevsky, 2009; Deng et al., 2009). Both ResNet structures follow the design from He et al. (2016), which includes batch normalization by default.

Hyperparameter settings. We find that *ZerO can fully utilize the default hyperparameters*, which include a learning rate of 0.1, a momentum of 0.9, and a weight decay of 0.0001. In addition, we observe the learning rate warmup is essential for ZerO to achieve a large maximal learning rate, as most of the weights start from the exact zero. We warm up the learning rate with 5 and 10 epochs for ImageNet and CIFAR-10, respectively.

We present our main results that compare different initialization schemes. For each dataset, all experiments use the same hyperparameter settings by default. Each experiment is repeated for ten runs with different random seeds. As shown in Table 4.1, ZerO achieves state-of-the-art accuracy on both datasets compared to other random methods.

In addition, we compare ZerO with a broad range of related works on CIFAR-10 using ResNet-18 and ResNet-50, including ReZerO (Bachlechner et al., 2021), Fixup (Zhang, Dauphin, and Ma, 2019), SkipInit (De and Smith, 2020) and ConstNet

(Blumenfeld, Gilboa, and Soudry, 2020). As shown in Table 4.2, ZerO consistently achieves top performance compared to other methods.

We note that the ConstNet proposed by Blumenfeld, Gilboa, and Soudry (2020) is also a deterministic initialization. However, unlike ZerO which preserves feature diversity, ConstNet is designed to eliminate the diversity by averaging the features through layers. The feature symmetric problem in ConstNet causes significant degradation, and additional random noise (e.g., non-deterministic GPU operation and dropout) is needed to break the symmetry.

Method	ZerO	ReZero	Fixup	SkipInit	ConstNet	ConstNet*
ResNet-18	5.13	5.20	5.17	5.26	72.39	5.41
ResNet-50	4.53	4.72	4.51	4.63	71.58	4.88

Table 4.2: Compare ZerO with other initialization methods on CIFAR-10. ConstNet* denotes ConstNet with non-deterministic GPU operations discussed in Blumenfeld, Gilboa, and Soudry (2020). Top-1 test error is reported.

Training ultra deep network without batch normalization Although there are methods attempting to train networks without batch normalization (by achieving dynamical isometry), they inevitably introduce random perturbations at initialization, affecting the training stability when the network is sufficiently deep (Zhang, Dauphin, and Ma, 2019; De and Smith, 2020). We benchmark ZerO with state-of-the-art methods on training without batch normalization. As shown in Figure 4.4 (left), compared to other methods, ZerO achieves the best training stability for networks with even around 500 layers. It also matches the baseline where batch normalization is enabled.

Improved reproducibility. In addition, as shown in Table 4.1, ZerO achieves the lowest standard deviation over the repeated runs. On ImageNet, the gap between ZerO and other methods is even more than 40%. Thus, removing the randomness in the weight initialization improves reproducibility, with possible implications for topics such as trustworthy machine learning and network interpretation.

ZerO on Transformer

We also apply ZerO to Transformer and evaluate it on WikiText-2 dataset (Vaswani et al., 2017). In each Transformer layer, we use ZerO to initialize both multi-head

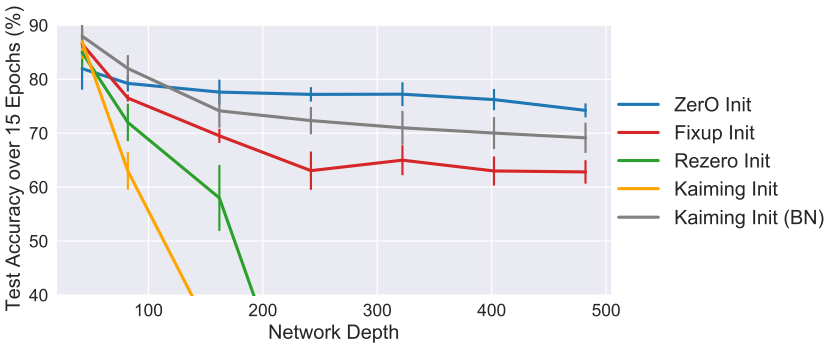

Figure 4.4: Training extreme deep ResNet on CIFAR-10 over 15 epochs.

attention and feed-forward layers. Because the embedding size is fixed in the multi-head attention, we initialize the projection matrix of queries $\boldsymbol{W}_Q$ as identity and the projection matrices of keys and values $\boldsymbol{W}_K, \boldsymbol{W}_V$ at zero. For the feed-forward layers, we initialize the connection matrices according to their hidden dimensions using Algorithm 4.

We train the Transformer models for 20 epochs with a single learning rate decay at epoch 10 [1]. We also vary the number of layers in the model from 2 to 20. As shown in Table 4.3, ZerO achieves similar performance compared to the standard initialization. In addition, it has better training stability over deeper Transformer models, which is consistent with our previous results on ResNet.

Number of layers	2	4	6	8	10	20
Standard	200.44	168.69	154.67	146.43	diverged	diverged
ZerO	192.34	169.73	151.91	149.27	145.62	141.81

Table 4.3: Evaluate Transformer on WikiText-2. We vary the number of layers in Transformer, where each layer consists of a multi-head attention and a feed-forward layer. Test perplexity is reported (lower is better).

4.6 Low-Rank Learning Trajectory

Although ZerO and random initialization achieve similar test accuracies, their training trajectories differ significantly. In contrast to random initialization, which begins optimization from a complex network (i.e., full-rank weight matrices, as shown in

[1] We use a transformer architecture (provided by the link here) that was smaller than the transformers typically used for this task, explaining the general degradation of the results.

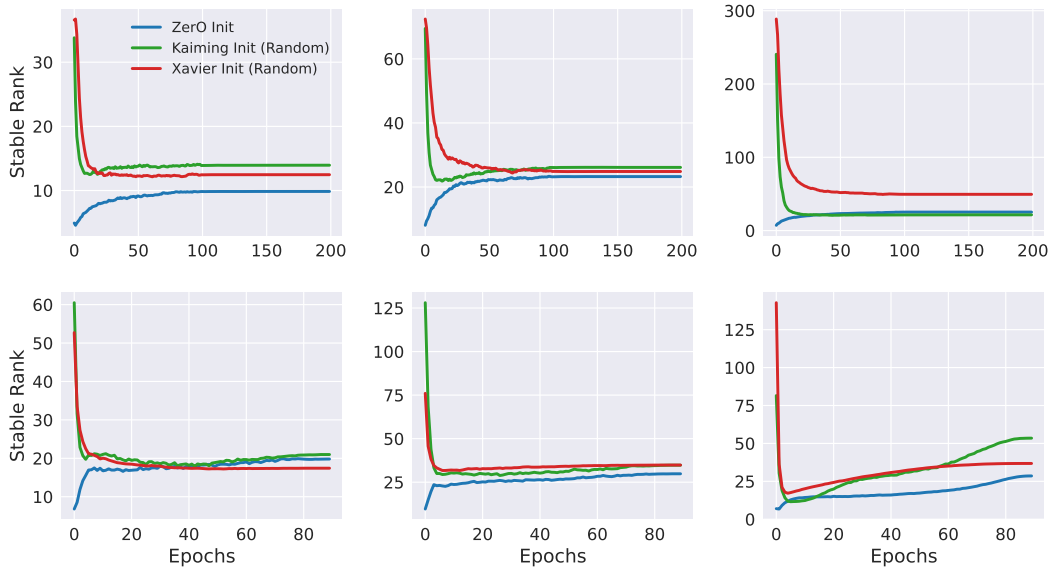

Figure 4.5: *Low-rank training trajectories in ResNet-18 on CIFAR-10 (top row) and ResNet-50 on ImageNet (bottom row). We visualize trajectories of the first convolutions in second, third, and fourth groups of residual blocks in ResNet.*

Figure 4.3), ZerO starts the training from a "simple" network and gradually increases its complexity.

To show the difference in practice, we track the ranks of convolutional kernels in ResNets during training, where the rank of each kernel can reflect its complexity. We measure the stable rank, which is defined as

$$\|\boldsymbol{W}\|_F^2 / \|\boldsymbol{W}\|_2^2 = \sum_{i=1}^{k} \sigma_i^2(\boldsymbol{W}) / \sigma_{max}^2(\boldsymbol{W}),$$

for any matrix $\boldsymbol{W}$ with k singular values σ_i. The stable rank is a soft version of the operator rank, and unlike the operator rank, it is insensitive to small singular values. We compare the stable ranks of various kernels between ZerO and random initialization during training. As shown in Figure 4.5, in contrast to random methods that begin with extremely high stable ranks, ZerO starts with low stable ranks and gradually increases them during training.

We believe ZerO's learning trajectory is the first demonstration of greedy low-rank learning (GLRL) in large-scale deep learning applications. GLRL is a theoretical characterization of gradient descent, such that: when matrices are initialized with infinitesimal values, gradient descent performs a rank-constrained optimization and

greedily relaxes the rank restriction by one whenever it fails to reach a minimizer (Li, Luo, and Lyu, 2021; Razin, Maman, and Cohen, 2021).

For example, when a matrix is initialized sufficiently small (where its rank is approximately zero), gradient descent first searches the solution over all rank-one matrices. If it fails to find a minimizer, it will relax the rank constraint by one and search again over all rank-two matrices. The search is stopped at rank-n if it finds a minimizer among all rank-n matrices.

GLRL suggests that gradient descent implicitly biases the model towards simple solutions by searching through the solution space in an incremental order of the matrix rank. This helps to explain the excellent generalization in gradient-based deep learning, as it converges to a global (or local) minima with the minimum rank.

Although the previous works have proved the existence of GLRL trajectory, it has never been observed in practice due to its impractical requirement of infinitesimal initialization. ZerO's learning trajectory we observed suggests that GLRL not only exists under infinitesimal initialization, but also under initialization around the identity. If a matrix $\boldsymbol{W}$ is initialized as $\boldsymbol{I}$, the low-rank structure is actually inside its residual component: $\boldsymbol{W}_l' = \boldsymbol{W}_l - \boldsymbol{I}$. To be noted, every convolutional kernel $\mathrm{conv}(x)$ we measured in Figure 4.5 can be viewed as the residual component of $\mathrm{conv}(x) + \boldsymbol{I}$, where the skip connection is included.

Figure 4.5 also suggests that the kernels never reach their maximal stable ranks during training under ZerO initialization. This implies that the searching over the space of full-rank weight matrices may be unnecessary, suggesting new avenues towards improved computational efficiency. We hope to explore this direction in future work.

We also observe that ZerO-based networks converge to low-complexity solutions. As shown in both Figure 4.5 and 4.6 (left), the convolutional kernels trained by ZerO usually have lower ranks than the kernels trained by random initialization. We further measure model complexity through both network pruning and low-rank approximation.

For network pruning, we use a standard magnitude-based pruning that prunes a portion of weights with the lowest magnitudes in each layer (Frankle and Carbin, 2019). For low-rank approximation, we apply Tucker-2 decomposition over channel dimensions in convolutions to select the most significant components (Kim et al., 2016).

As shown in Figure 4.6, compared to randomly initialized networks, the sub-networks obtained in trained ZerO-initialized networks achieve 25% higher sparsity or 30% lower (matrix or tensor) rank without sacrificing accuracy. This suggests ZerO encourages the networks to converge to low-complexity solutions, which improves the computational efficiency for inference.

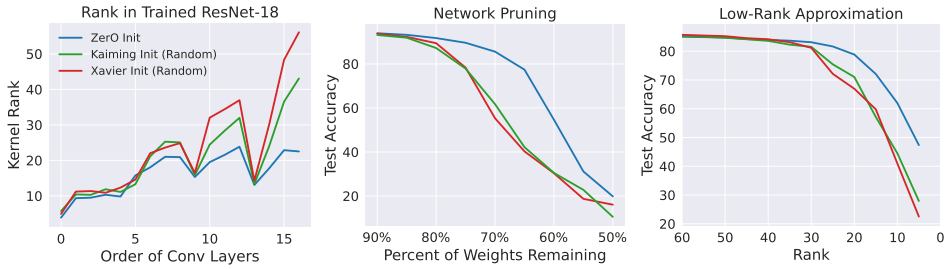

Figure 4.6: **Left:** comparing kernel rank in ResNet-18 trained by ZerO and Kaiming methods. **Middle:** a magnitude-based network pruning on ResNet-18. **Right:** a Tucker-2 decomposition for a particular convolution with 512 channels in ResNet-18.

4.7 Conclusion

In this work, we propose a simple and fully deterministic initialization called ZerO. Extensive experiments demonstrate that ZerO achieves state-of-the-art performance, suggesting that random weight initialization may not be necessary for initializing deep neural networks. ZerO has many benefits, such as training ultra deep networks (without batch-normalization), exhibiting low-rank learning trajectories that result in low-rank and sparse solutions, and improving training reproducibility.

We believe that ZerO opens up many new possibilities given its various benefits. It can be applied to networks and tasks sensitive to the variances in weight initialization. Its low-rank learning trajectories enable the development of rank-constrained training methods that improve computational efficiency. Finally, the improved training reproducibility can aid model interpretability. We hope our results will inspire other researchers to consider deterministic initialization schemes and to rethink the role of weight initialization in training deep neural networks.

References

Arora, Sanjeev et al. (Oct. 2019). "Implicit Regularization in Deep Matrix Factorization". In: *arXiv:1905.13655 [cs, stat]*. arXiv: 1905.13655. URL: `http://arxiv.org/abs/1905.13655` (visited on 04/12/2022).

Bachlechner, Thomas et al. (2021). "ReZero is all you need: fast convergence at large depth". In: *Proceedings of the Thirty-Seventh Conference on Uncertainty in Artificial Intelligence, UAI 2021, Virtual Event, 27-30 July 2021*. AUAI Press.

Balduzzi, David et al. (July 2017). "The Shattered Gradients Problem: If resnets are the answer, then what is the question?" en. In: *Proceedings of the 34th International Conference on Machine Learning*. ISSN: 2640-3498. PMLR, pp. 342–350. URL: https://proceedings.mlr.press/v70/balduzzi17b.html (visited on 10/04/2021).

Bartlett, Peter L., David P. Helmbold, and Philip M. Long (Mar. 2019). "Gradient Descent with Identity Initialization Efficiently Learns Positive-Definite Linear Transformations by Deep Residual Networks". en. In: *Neural Computation* 31.3, pp. 477–502. ISSN: 0899-7667, 1530-888X. DOI: 10.1162/neco_a_01164. URL: https://direct.mit.edu/neco/article/31/3/477-502/8456 (visited on 01/12/2022).

Blumenfeld, Yaniv, Dar Gilboa, and Daniel Soudry (2020). "Beyond Signal Propagation: Is Feature Diversity Necessary in Deep Neural Network Initialization?" In: *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*. PMLR.

De, Soham and Samuel L. Smith (2020). "Batch Normalization Biases Residual Blocks Towards the Identity Function in Deep Networks". In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Deng, Jia et al. (June 2009). "ImageNet: A large-scale hierarchical image database". In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. ISSN: 1063-6919, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848.

Frankle, Jonathan and Michael Carbin (Mar. 2019). "The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks". In: *arXiv:1803.03635 [cs]*. arXiv: 1803.03635. URL: http://arxiv.org/abs/1803.03635 (visited on 10/27/2021).

Gehring, Jonas et al. (July 2017). "Convolutional Sequence to Sequence Learning". en. In: *Proceedings of the 34th International Conference on Machine Learning*. ISSN: 2640-3498. PMLR, pp. 1243–1252. URL: https://proceedings.mlr.press/v70/gehring17a.html (visited on 10/04/2021).

Glorot, Xavier and Yoshua Bengio (n.d.). "Understanding the Difficulty of Training Deep Feedforward Neural Networks". In: ().

Hardt, Moritz and Tengyu Ma (2017). "Identity Matters in Deep Learning". In: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net. URL: https://openreview.net/forum?id=ryxB0Rtxx.

He, Kaiming et al. (2015). "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification". In: *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*. IEEE Computer Society.

– (2016). "Deep Residual Learning for Image Recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*. IEEE Computer Society.

Hoffer, Elad et al. (Feb. 2019). "Norm matters: efficient and accurate normalization schemes in deep networks". In: *arXiv:1803.01814 [cs, stat]*. arXiv: 1803.01814. URL: http://arxiv.org/abs/1803.01814 (visited on 10/04/2021).

Ioffe, Sergey and Christian Szegedy (2015). "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*. JMLR.org.

Jacot, Arthur et al. (Jan. 2022). "Saddle-to-Saddle Dynamics in Deep Linear Networks: Small Initialization Training, Symmetry, and Sparsity". en. In: *arXiv:2106.15933 [cs, stat]*. arXiv: 2106.15933. URL: http://arxiv.org/abs/2106.15933 (visited on 02/21/2022).

Kim, Yong-Deok et al. (Feb. 2016). "Compression of Deep Convolutional Neural Networks for Fast and Low Power Mobile Applications". In: *arXiv:1511.06530 [cs]*. arXiv: 1511.06530. URL: http://arxiv.org/abs/1511.06530 (visited on 03/25/2022).

Krizhevsky, Alex (2009). "Learning Multiple Layers of Features from Tiny Images". In.

Li, Zhiyuan, Yuping Luo, and Kaifeng Lyu (2021). "Towards Resolving the Implicit Bias of Gradient Descent for Matrix Factorization: Greedy Low-Rank Learning". In: *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net. URL: https://openreview.net/forum?id=AHOs7Sm5H7R.

Pennington, Jeffrey, Samuel S. Schoenholz, and Surya Ganguli (Nov. 2017). "Resurrecting the sigmoid in deep learning through dynamical isometry: theory and practice". In: *arXiv:1711.04735 [cs, stat]*. arXiv: 1711.04735. URL: http://arxiv.org/abs/1711.04735 (visited on 04/27/2022).

Razin, Noam, Asaf Maman, and Nadav Cohen (June 2021). "Implicit Regularization in Tensor Factorization". In: *arXiv:2102.09972 [cs, stat]*. arXiv: 2102.09972. URL: http://arxiv.org/abs/2102.09972 (visited on 04/12/2022).

Saxe, Andrew M., James L. McClelland, and Surya Ganguli (Feb. 2014). "Exact solutions to the nonlinear dynamics of learning in deep linear neural networks". In: *arXiv:1312.6120 [cond-mat, q-bio, stat]*. arXiv: 1312.6120. URL: http://arxiv.org/abs/1312.6120 (visited on 04/20/2021).

Vaswani, Ashish et al. (Dec. 2017). "Attention Is All You Need". In: *arXiv:1706.03762 [cs]*. arXiv: 1706.03762. URL: http://arxiv.org/abs/1706.03762 (visited on 01/01/2021).

Xiao, Lechao et al. (June 2018). "Dynamical Isometry and a Mean Field Theory of CNNs: How to Train 10,000-Layer Vanilla Convolutional Neural Networks". en. In: URL: https://arxiv.org/abs/1806.05393v2 (visited on 05/23/2019).

Zhang, Hongyi, Yann N. Dauphin, and Tengyu Ma (2019). "Fixup Initialization: Residual Learning Without Normalization". In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Chapter 5

LNS-MADAM: LOW-PRECISION TRAINING IN LOGARITHMIC NUMBER SYSTEM USING MULTIPLICATIVE WEIGHT UPDATE

5.1 Introduction

Deep neural networks (DNNs) have shown impressive performance in many applications, including image classification and language processing. However, training and deploying DNNs typically incurs significant computation and energy costs. Traditionally, values in neural networks are represented using floating-point (32-bit) numbers, which incur a large arithmetic and memory footprint, and hence significant energy consumption. However, recent studies suggest that high-precision number formats are redundant, and models can be quantized in low-precision with little loss in accuracy (Gupta et al., 2015; N. Wang et al., 2018). Low-precision numbers only require low-bitwidth computational units, leading to better computational efficiency and less required memory bandwidth and capacity.

While low-precision training methods generally reduce computational costs, energy efficiency can be further improved by choosing a logarithmic number system (LNS) for representing numbers. LNS achieves a higher computational efficiency by transforming expensive multiplication operations in the network layers to inexpensive additions in their logarithmic representations. In addition, it attains a wide dynamic range and can provide a good approximation of the non-uniform weight distribution in neural networks. Thus LNS is an excellent candidate for training DNNs in low-precision (Frankle and Carbin, 2019; Bartol et al., 2015; Miyashita, Edward H. Lee, and Murmann, 2016).

Although previous studies demonstrate that it is feasible to train networks in low-precision using LNS, these approaches have not yet shown promising results on larger datasets and state-of-the-art models (Miyashita, Edward H. Lee, and Murmann, 2016; Sun, N. Wang, et al., 2020). Standard LNS fixes the base of the logarithm, termed log-base, to be precisely two. However, a more flexible log-base is needed since the numbers in DNNs require different quantization gaps during training (Vogel et al., 2018). A flexible log-base can introduce additional hardware overhead due to expensive conversion operations between the logarithmic and inte-

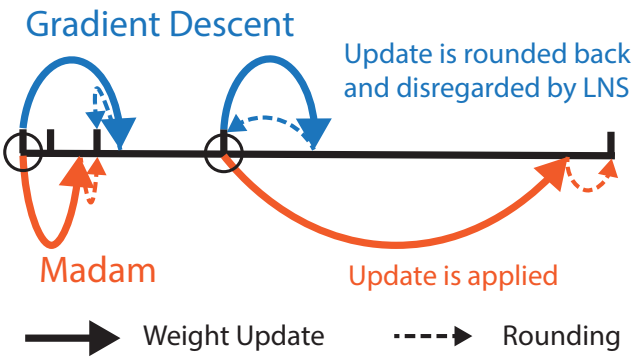

Figure 5.1: Illustration for updating weights using Gradient Descent (GD) and Madam under logarithmic representation. Each coordinate represents a number stored in LNS. Assume the weights at two circles receive the same gradient. The updates generated from GD are disregarded as the weights move larger, whereas the updates generated by Madam are adjusted with the weights.

ger (linear) formats. This motivates us to design a LNS that has a flexible choice of the log-base while maximizing the efficiency of LNS-to-integer conversions.

Conventional low-precision training methods typically require high-precision copies of weights and gradients during weight update to maintain optimization stability. In fact, most recent studies even use a full-precision (FP32) copy of weights (Miyashita, Edward H. Lee, and Murmann, 2016; Sun, N. Wang, et al., 2020). This introduces additional energy costs and expensive FP32 arithmetic, which becomes prohibitive especially in energy-constrained edge devices.

The high-precision requirement for weight updates is due to complex interactions between learning algorithms and number systems, which has usually been ignored in previous studies. For example, as illustrated in Fig. 5.1, updates generated by stochastic gradient descent (SGD) are disregarded more frequently by LNS when the weights become larger. This is because the quantization gap grows exponentially as the weights become larger in LNS, which suggests that conventional learning algorithms may not be suitable for LNS. Hence, in previous studies, high-precision weight copies are required to avoid numerical instabilities (Miyashita, Edward H. Lee, and Murmann, 2016; Vogel et al., 2018).

To directly update the weights in low-precision, we employ a learning algorithm tailored to LNS. Recently, Bernstein et al. (Bernstein, J. Zhao, Meister, et al., 2020) proposed the Madam optimizer based on multiplicative updates, which is equivalent to updating weights additively in the logarithmic space. As illustrated in Fig. 5.1, Madam generates larger magnitudes of the updates when the weights become larger,

making it more suitable for a logarithmic weight representation. However, Bernstein et al. (Bernstein, J. Zhao, Meister, et al., 2020) still employ full-precision training with Madam without considering low-precision LNS.

In this work, we propose a co-designed low-precision training framework called LNS-Madam in which we adopt LNS (with a more flexible log-base) for representing DNNs and apply a modified Madam (tailored to LNS) to train these networks. LNS-Madam reduces the precision requirements for all components of the training, including *forward and backward propagation, as well as weight updates*.

Our contributions are summarized as follows:

1. We design a multi-base LNS where the log-base can be fractional powers of two. The multi-base LNS accommodates the precision and range requirements of the training dynamics while being hardware-friendly. In addition, we propose an approximation for the addition arithmetic in LNS to further improve its energy efficiency.

2. We propose an efficient hardware implementation of LNS-Madam that addresses challenges in designing an efficient datapath for LNS computations, including accumulation and conversion between logarithmic and integer formats. We leverage this implementation to study the energy benefits of training in LNS.

3. To achieve low-precision weight updates in LNS, we replace standard SGD or Adam optimizers with our proposed optimizer based on Madam, which directly updates the weights in the LNS. Through theoretical analysis and empirical evaluations, we show that the proposed Madam optimizer achieves significantly lower quantization error in LNS.

4. In our experiments, LNS-Madam achieves full-precision accuracy with 8 bit representations on popular computer vision and natural language tasks while reducing energy consumption by over 90%. The energy efficiency results for training different models with various number formats are summarized in Fig. 5.2.

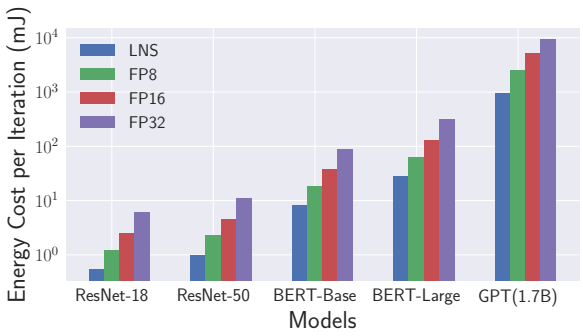

Figure 5.2: Energy efficiency for training different models with various number formats. The per-iteration energy consumption (mJ) is listed.

5.2 Related Works

Low-precision training

To achieve good accuracy at reduced precision, quantization-aware training (QAT) is commonly applied to directly train the quantized model using straight-through estimators (Bengio, Léonard, and Courville, 2013; S. Zhou et al., 2016; Rastegari et al., 2016; A. Zhou et al., 2017; Jacob et al., 2018). To accelerate the training phase, several studies suggest quantizing the gradients during backward propagation (Banner et al., 2018; Sun, N. Wang, et al., 2020; N. Wang et al., 2018). To maintain the fidelity of the gradient accumulation, some low-precision training methods assume a full-precision copy for weights during the weight update (Banner et al., 2018; Chen et al., 2020). Other studies reduce the precision for the weight update by using high-precision gradient accumulator (Sakr and Shanbhag, 2019), stochastic rounding (S. Wu et al., 2018; N. Wang et al., 2018) or additionally quantizing the residual part of weights (Sun, Choi, et al., 2019; Sa et al., 2018). Cambricon-Q accelerates the weight update from a hardware perspective by avoiding costly data transferring in weight update (Y. Zhao et al., 2021). However, they mostly apply SGD or Adam during the weight update without considering the relationship between the precision of the weights and the underlying learning algorithms.

Logarithmic number system

Previous works demonstrate the effectiveness of using logarithmic representation for DNNs (Miyashita, Edward H. Lee, and Murmann, 2016; E. H. Lee et al., 2017; Jeff Johnson, 2018; J. Johnson, 2020). Furthermore, some studies suggest using multiple levels of log-base to reduce the quantization error (Miyashita, Edward H. Lee, and Murmann, 2016; Vogel et al., 2018). However, few of them address the additional computational cost induced by this multi-base design nor scale the

system to state-of-the-art neural networks for both training and inference. From the perspective of hardware design, a few studies focus on improving the efficiency of LNS by utilizing the significant cost reduction of multiplications (Saadat, Bokhari, and Parameswaran, 2018; Saadat, Javaid, and Parameswaran, 2019; Saadat, Javaid, Ignjatovic, et al., 2020; Jeff Johnson, 2018; J. Johnson, 2020).

Multiplicative weight update

Multiplicative algorithms, such as exponentiated gradient algorithm and Hedge algorithm in AdaBoost framework (Kivinen and Warmuth, 1997; Freund and Schapire, 1997), have been well studied in the field of machine learning. In general, multiplicative updates are applied to problems where the optimization domain's geometry is described by relative entropy, such as probability distribution (Kivinen and Warmuth, 1997). Recently, (Bernstein, J. Zhao, Meister, et al., 2020) proposes an optimizer Madam that focuses on optimization domains described by any relative distance measure instead of only relative entropy. Madam shows great performance in training large-scale neural networks. However, Madam requires full-precision training without considering its connection to LNS-based low-precision training.

5.3 Hardware-friendly Multi-Base Logarithmic Number System

In this section, we introduce our multi-base logarithmic number system (LNS), including the corresponding number representation and arithmetic operations.

We start with the mathematical formulation that we will use throughout this paper. We assume that the DNN $F(\cdot, W)$ is composed of L layers with learnable weights W and input activations X across the layers. $\mathcal{L}(W)$ denotes the training loss. The forward propagation is defined as: $X_l = f_l(X_{l-1}, W_l)$, where $l \in [1, L]$ denotes layer l. $\nabla_{X_l} = \frac{\partial \mathcal{L}(W)}{\partial X_l}$ and $\nabla_{W_l} = \frac{\partial \mathcal{L}(W)}{\partial W_l}$ denote gradients with respect to input activations and weights, respectively, at layer l. For a number system, we define $\mathcal{B}$ as the bitwidth, x as a number, and x^q as the number in quantized format.

Multi-base Logarithmic Representation

Unlike prior work that uses exactly 2 as the base of the logarithmic representation, we propose a multi-base logarithmic representation that allows the base to be two with a fractional exponent, such that:

$$x = \text{sign} \times 2^{\tilde{x}/\gamma} \quad \tilde{x} = 0, 1, 2, ..., 2^{\mathcal{B}-1} - 1,$$

where $\tilde{x}$ is an integer and γ is the base factor that controls the fractional exponent of the base. γ controls the quantization gap, which is the distance between successive

representable values within the number system. Previous work has already demonstrated that logarithmic quantized neural networks achieve better performance when relaxing γ from 1 to 2 (Miyashita, Edward H. Lee, and Murmann, 2016). We find that further relaxation can help adapt to different models and datasets. Therefore we generalize the base factor setting, enabling more flexibility in controlling the quantization gap in order to more accurately approximate the training dynamics. In addition, we specially restrict γ to be powers of 2 for hardware efficiency, as described later.

Arithmetic Operations in LNS

One of the benefits of using LNS stems from the low computational cost of its arithmetic operations. We use dot product operations as an example since they are prevalent during training. Consider two vectors $\boldsymbol{a} \in \mathbb{R}^n$ and $\boldsymbol{b} \in \mathbb{R}^n$ that are represented by their integer exponents $\tilde{\boldsymbol{a}}$ and $\tilde{\boldsymbol{b}}$ in LNS. A dot product operation between them can be represented as follows:

$$\begin{aligned} \boldsymbol{a}^T\boldsymbol{b} &= \sum_{i=1}^{n} \text{sign}_i \times 2^{\tilde{\boldsymbol{a}}_i/\gamma} \times 2^{\tilde{\boldsymbol{b}}_i/\gamma} \\ &= \sum_{i=1}^{n} \text{sign}_i \times 2^{(\tilde{\boldsymbol{a}}_i+\tilde{\boldsymbol{b}}_i)/\gamma} \\ &= \sum_{i=1}^{n} \text{sign}_i \times 2^{\tilde{p}_i/\gamma}, \end{aligned} \tag{5.1}$$

where $\text{sign}_i = sign(\boldsymbol{a}_i) \oplus sign(\boldsymbol{b}_i)$. In this dot product operation, each element-wise multiplication is computed as an addition between integer exponents, which significantly reduces the computational cost by requiring adders instead of multipliers.

While the multiplications are easy to compute in LNS, the accumulation is difficult to compute efficiently as it requires first converting from logarithmic to integer format and then performing the addition operation. The conversion between these formats is generally expensive as it requires computing $2^{\tilde{p}_i/\gamma}$ using polynomial expansion. To overcome this challenge, we decompose the exponent $2^{\tilde{p}_i/\gamma}$ into a quotient component $\tilde{p}_{iq}$ and and a remainder component $\tilde{p}_{ir}$ as follows:

$$2^{\tilde{p}_i/\gamma} = 2^{\tilde{p}_{iq}+\tilde{p}_{ir}/\gamma} = 2^{\tilde{p}_{iq}} \cdot 2^{\tilde{p}_{ir}/\gamma}. \tag{5.2}$$

With this decomposition, converting from LNS to integer requires only a table lookup for $2^{\tilde{p}_{ir}/\gamma}$ followed by a shift for $2^{\tilde{p}_{iq}}$. For the table lookup, we simply maintain γ constants $2^{i/\gamma}$ $\forall i \in \{0, 1, ..., \gamma - 1\}$ and select the constant based on the remainder $\tilde{p}_{ir}$. Note that because γ is restricted to be power of 2, the remainder can

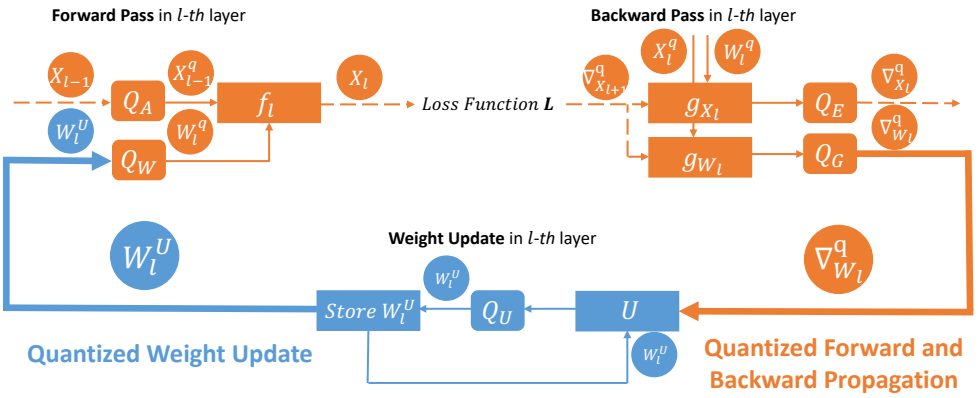

Figure 5.3: Illustration of LNS-Madam. Quantized training includes quantizing weights W and activations X in forward propagation, and weight gradients ∇_W and activation gradients ∇_X in backward propagation. g_X and g_W denote the functions to compute gradients. Quantized weight update applies a quantization function Q_U over weights after any learning algorithm U updates them. The quantized weights W^U are the actual numbers stored in the system.

be efficiently extracted from the least-significant bits (LSB) of the exponent while the quotient can be extracted from the most-significant bits (MSB). Typically the lookup table (LUT) requires $2^{\mathcal{B}}$ entries for storing all possible values.

Conversion Approximation

In addition to the exact conversion technique discussed above, we can further reduce the cost of the LNS-to-integer conversion using a hybrid approximation method. Our method is based on Mitchell approximation (Mitchell, 1962): $2^{\tilde{x}/\gamma} \approx (1+\tilde{x}/\gamma)$, where the logarithmic format can be efficiently approximated to the integer format when $\tilde{x}/\gamma$ is small. Specifically, we further split the remainder into a LSB and a MSB component. The value of the LSB is approximated using Mitchell approximation, and the value of the MSB is performed with table lookup. This helps reduce the size of the LUT. We present a detailed description of our approximation method in the appendix. In addition, since the approximation serves as an additional non-linear operation in neural networks, we find the approximated training does not damage accuracy in practice.

5.4 Quantized Forward and Backward Propagation on LNS

In this section, we introduce how to apply multi-base LNS to quantized training, as illustrated in Fig. 5.3.

To realize reduced precision for values and arithmetic during training, we define

a logarithmic quantization function LogQuant : $\mathbb{R} \rightarrow \mathbb{R}$, which quantizes a real number into a sign and an integer exponent using a limited number of bits. LogQuant is defined as follows:

$$\text{LogQuant}(x) = sign(x) \times s \times 2^{(\tilde{x}/\gamma)}, \tag{5.3}$$

where $\tilde{x} = clamp(\, round(\, \log_2(|x|/s) \times \gamma), 0, 2^{\mathcal{B}-1} - 1)$, and $s \in \mathbb{R}$ denotes a scale factor. LogQuant first brings scaled numbers $|x|/s$ into their logarithmic space, magnifies them by the base factor γ and then performs rounding and clamping functions to convert them into desired integer exponents $\tilde{x}$. The scale factor s usually is shared within a group of numbers, and its value is assigned to match the maximum number within the group.

We apply quantization-aware training (QAT) for quantizing weights and activations during forward propagation. Each quantizer is associated with a STE to allow the gradients to directly flow through the non-differentiable quantizer during backward pass (Bengio, Léonard, and Courville, 2013). Because QAT views each quantization function as an additional non-linear operation in the networks, the deterministic quantization error introduced by any quantizer in the forward pass is implicitly reduced through training. We define weight quantization function as $\mathrm{Q_W}$ and activation quantization function as $\mathrm{Q_A}$ for each layer during forward propagation, where $W_l^q = \mathrm{Q_W}\,(W_l)$ and $X_l^q = \mathrm{Q_A}\left(f_l\left(X_{l-1}^q, W_l^q\right)\right)$.

In order to accelerate training in addition to inference, gradients also need to be quantized into low-precision numbers. As shown by recent studies, the distribution of gradients resembles a Gaussian or Log-Normal distribution (Chmiel et al., 2021; Bernstein, J. Zhao, Azizzadenesheli, et al., 2019). This suggests that logarithmic representation may be more suitable than fixed-point representations when quantizing gradients to attain hardware efficiency. We quantize the activation gradients using quantization function $\mathrm{Q_E}$: $\nabla_{X_l}^q = \mathrm{Q_E}\left(\nabla_{X_l}\right)$. We also quantize the weight gradients using quantization function $\mathrm{Q_G}$: $\nabla_{W_l}^q = \mathrm{Q_G}\left(\nabla_{W_l}\right)$. In this work, we aim to reduce the precision requirement for both weight gradients and activation gradients in the backward pass.

5.5 Multiplicative Weight Update Algorithm for LNS

Although logarithmic quantized training significantly improves training efficiency, its overall efficiency continues to be hampered by the high precision requirement of weight updates. We note that quantized weight update is orthogonal to quantized training due to the difference in their objectives. Quantized training tries to

maintain the fidelity of weight gradients while accelerating forward and backward propagation. This provides accurate gradient information for the weight update. On the other hand, after receiving quantized weight gradients, quantized weight update aims to reduce gaps between updated weights and their (rounded) quantized counterparts. Fig. 5.3 distinguishes the two parts by different colors.

Previous works generally assume that the weight update is computed over a full-precision weight space. In other words, a full-precision copy of weights is maintained (Miyashita, Edward H. Lee, and Murmann, 2016; Vogel et al., 2018) and very little rounding follows weight update. However, this offsets the efficiency benefits of quantized training and requires expensive floating-point arithmetic not available especially in cheap energy-constrained edge devices. Therefore, in this work, we consider quantized weight update in LNS, where the weights are updated over a discrete logarithmic space instead of a full-precision one. We aim to minimize the rounding error given that weights are represented in LNS.

Quantized Weight Update

To better understand this problem, we first define a generalized form of a weight update as: $W_{t+1} = U\left(W_t, \nabla_{W_t}\right)$, where U represents any learning algorithm. For example, gradient descent (GD) algorithm takes $U_{GD} = W_t - \eta \nabla_{W_t}$, where η is learning rate.

Because the weights need to be represented in a quantized format in LNS, it is necessary to consider the effect of logarithmic quantization during weight update. We define *logarithmic quantized weight update* as follows:

$$W_{t+1}^{U} = \text{LogQuant}\left(U\left(W_t, \nabla_{W_t}\right)\right). \tag{5.4}$$

In this case, W_{t+1}^{U} can be directly stored in a logarithmic format without using floating-point data type. For simplicity, we assume weight gradients ∇_{W_t} are exact as quantized training is orthogonal to this problem. Switching to the approximated gradient estimates will not affect our theoretical results.

Quantization Error Analysis

Because the logarithmic quantization requires representing values in a discrete logarithmic scale, quantized weight update inevitably introduces a mismatch between the quantized weights and their full-precision counterparts. To preserve the reliability of the optimization, we aim to reduce the quantization error (i.e., the mismatch). For the following, we take a theoretical perspective to discuss how different learning

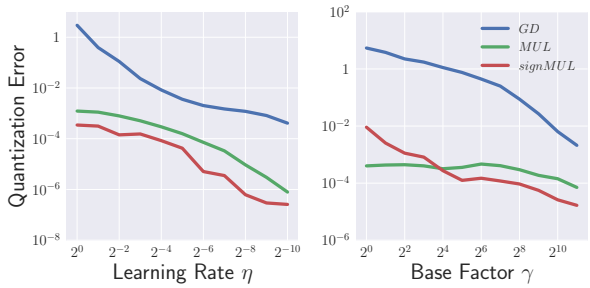

Figure 5.4: Quantization error from different learning algorithms on ImageNet. The errors are averaged over all iterations in the first epoch. The results suggest that multiplicative algorithms introduce significantly lower errors compared to the gradient descent, which are also in line with our theoretical results.

algorithms affect the quantization error under LNS.

Due to the logarithmic structure, we focus on minimizing a quantizaion error $r_t = \|\log_2 |W^U_{t+1}| - \log_2 |W_{t+1}|\|^2$, which measures the L2 norm of the difference between the weights and their quantized counterparts in logarithmic space. Because r_t quantify the relative difference between $|W^U_{t+1}|$ and $|W_{t+1}|$, minimizing r_t is largely similar to minimizing the relative quantization error $\|(W_{t+1} - W^U_{t+1})/W_{t+1}\|^2$.

We assume a simplified logarithmic quantization where the scale factor and the clamping function are ignored. This ensures our focus is on the effect of the quantization gap determined by γ instead of the dynamic range. We also replace the deterministic rounding with a stochastic counterpart SR where $\mathbb{E}\,\mathrm{SR}(x) = x$ for any real number. Although SR helps us establish the theoretical results, in practice SR requires random generators that induce additional costs, and thus are not suitable for energy-efficient training.

Given everything we need, we use gradient descent as an example to discuss why traditional learning algorithms are not suited for LNS-based quantized weight updates. The theorem is stated as follows:

Theorem 5.5.1 *The quantization error $r_{t,GD}$ introduced by logarithmic quantized gradient descent at iteration t can be bounded in expectation, as:*

$$\mathbb{E}\, r_{t,GD} \leq \frac{\sqrt{d}}{\gamma} \|\log_2 \left(|W_t| - \eta_1 \nabla_{W_t}\right)\|, \tag{5.5}$$

where d is the dimension of W and η_1 is the learning rate of U_{GD}.

Theorem 5.5.1 suggests that $r_{t,GD}$ is magnified when the magnitudes of weights become larger. This is because the updates $\eta_1 \nabla_{W_t}$ generated by GD are not pro-

portional to the magnitudes of weights. $\eta_1 \nabla_{W_t}$ can be orders of magnitude smaller than the quantization gaps as weights become larger, and thus these updates often are disregarded by quantization function LogQuant. We intuitively illustrate this problem in Fig. 5.1.

To ensure the updates are proportional to the weights, a straightforward way is to update the weights multiplicatively. Because the weights are represented in LNS, we further consider a special multiplicative learning algorithm tailored to LNS, which updates the weights directly over their logarithmic space:

$$U_{MUL} = sign(W_t) \odot 2^{\tilde{W}_t - \eta \nabla_{W_t} \odot sign(W_t)} \tag{5.6}$$

where $\tilde{W}_t = \log_2 |W_t|$ are the exponents of the magnitude of weights, and $\odot$ denotes element-wise multiplication. U_{MUL} makes sure the magnitude of each element $W_t(k)$ of the weights decreases when the sign $sign(W_t(k))$ and $\nabla_{W_t(k)}$ agree and increases otherwise. The quantization error with regards to U_{MUL} is stated as follows:

Theorem 5.5.2 *The quantization error $r_{t,MUL}$ introduced by logarithmic quantized multiplicative weight update at iteration t can be bounded in expectation, as:*

$$\mathbb{E}\, r_{t,MUL} \leq \frac{\sqrt{d}\,\eta_2}{\gamma} \|\nabla_{W_t}\|, \tag{5.7}$$

where d is the dimension of W and η_2 is the learning rate of U_{MUL}.

Theorem 5.5.2 indicates that $r_{t,MUL}$ does not depend on the magnitudes of weights, and thus the quantization error is not magnified when the weights become larger. This is in stark contrast to the quantization error from gradient descent shown in Equation 5.5. The comparison is illustrated in Fig. 5.1.

Interestingly, we find that the quantization error $r_{t,MUL}$ can be further simplified by regularizing the information of gradients for the learning algorithm U_{MUL}:

Lemma 5.5.3 *Assume the multiplicative learning algorithm U_{MUL} only receives the sign information of gradients where $U_{MUL} = \tilde{W}_t - \eta_2\, sign(\nabla_{W_t}) \odot sign(W_t)$. The upper bound on quantization error $r_{t,MUL}$ becomes:*

$$\mathbb{E}\, r_{t,MUL} \leq \frac{d\,\eta_2}{\gamma}. \tag{5.8}$$

The result in Lemma 5.5.3 suggests that $r_{t,MUL}$ can be independent of both weights and gradients when only taking sign information of gradients during weight update.

We denote this special learning algorithm as $U_{signMUL}$. $U_{signMUL}$ is a multiplicative version of signSGD, which has been studied widely (Bernstein, J. Zhao, Azizzadenesheli, et al., 2019; Bernstein, Y.-X. Wang, et al., 2018).

To verify our theoretical results, we empirically measure the quantization errors for the three aforementioned learning algorithms over a range of η and γ. As shown in Fig. 5.4, the empirical findings are in line with our theoretical results. Although all learning algorithms introduce less errors when η and γ become smaller, the multiplicative algorithms introduce significantly lower errors compared to the gradient descent.

In addition to reducing the quantization error in the quantized weight update, $U_{signMUL}$ must also have the ability to minimize the loss function $\mathcal{L}(W)$. Interestingly, we notice that $U_{signMUL}$ resembles a recently proposed learning algorithm Madam, where Bernstein et al. (Bernstein, J. Zhao, Meister, et al., 2020) proves that Madam optimizes the weights in a descent direction. Madam updates the weights multiplicatively using normalized gradients:

$$\begin{aligned} U_{Madam} &= W_t \odot e^{-\eta\, sign(W_t) \odot g_t^*} \\ g_t^* &= g_t / \sqrt{g_{2_t}}, \end{aligned} \tag{5.9}$$

where g_t represents the gradient vector ∇_{W_t}, and g_t^* denote a normalized gradient, which is the fraction between g_t and the square root of its second moment estimate $\sqrt{g_{2_t}}$. Bernstein et al. (Bernstein, J. Zhao, Meister, et al., 2020) demonstrates that Madam achieves state-of-the-art accuracy over multiple tasks with a relatively fixed learning rate η. They also theoretically prove the descent property of Madam that ensures its convergence. Although Bernstein et al. (Bernstein, J. Zhao, Meister, et al., 2020) further shows the possibility of applying Madam over a discrete logarithmic weight space, they still employ full-precision training without considering low-precision LNS.

To ensure low-precision weight updates in LNS, we apply a modified version of the Madam optimizer to enable fast convergence while preserving low quantization error. The modified Madam directly optimizes the weights over their base-2 logarithmic space using the gradient normalization technique described in Equation 5.9. Details of our optimizer are shown in Algorithm 6. Because our Madam optimizer directly updates base-2 exponents of weights in LNS, there is no need for integer-to-LNS conversion during weight update when the weights are already in LNS, further reducing the energy cost.

Algorithm 6 *Madam optimizer on LNS*

Require: Base-2 weight exponents $\tilde{W}$, where $\tilde{W} = \log_2(W)$, learning rate η, momentum β
Initialize $g_2 \leftarrow 0$
repeat
  $g \leftarrow$ StochasticGradient()
  $g_2 \leftarrow (1-\beta)g^2 + \beta g_2$
  $g^* \leftarrow g/\sqrt{g_2}$
  $\tilde{W} \leftarrow \tilde{W} - \eta\, g^* \odot sign(W)$
until converged

5.6 Hardware Implementation

We extend a previously optimized DNN accelerator (Venkatesan et al., 2019) to support LNS-based DNN computations. Fig. 5.5 shows the micro-architecture of the PE which performs dot-product operations. Each PE consists of set of vector MAC units fed by the buffers that store weights, input activations, and output gradients. Additionally, the accumulation collectors store and accumulate partial sums which are passed to the PPU for post-processing (e.g., quantization scaling, non-linear activation functions) if necessary.

Fig. 5.6 shows the LNS-based datapath inside the LNS-Madam Vector MAC Unit. Here we model exact LNS-to-integer conversion without any approximation. With a vector size of 32 and input bitwidths of 8, the datapath processes 32 7-bit exponent values at each of its exponent inputs (e_a and e_b) and 32 1-bit sign values (s_a and s_b) at each of its sign inputs to produce a 24-bit partial sum. First, the LNS datapath performs the dot-product multiplication by adding the exponents and XOR-ing the sign bits. The output of the product computation requires an additional bit to account for the carry-out of the adder. At this point, each exponent is split into a quotient component (e_q) and a remainder component (e_r) based on the LSB/MSB property mentioned in Section 5.3. Second, the datapath performs shifting by the quotient to implement the quotient component in Equation 5.2. Depending on the sign bit, the corresponding signed shifted value is selected and passed to the corresponding adder tree based on the remainder select signal. Third, the result of shifted values are reduced through the set of adder trees and registered. At last, the results of the adder trees are multiplied with corresponding remainder constants (described in Section 5.3) from a LUT and accumulated into the final partial sum, represented in integer (linear) format. This partial sum needs to be converted back into logarithmic format and written back to the global buffer for subsequent LNS-based computations.

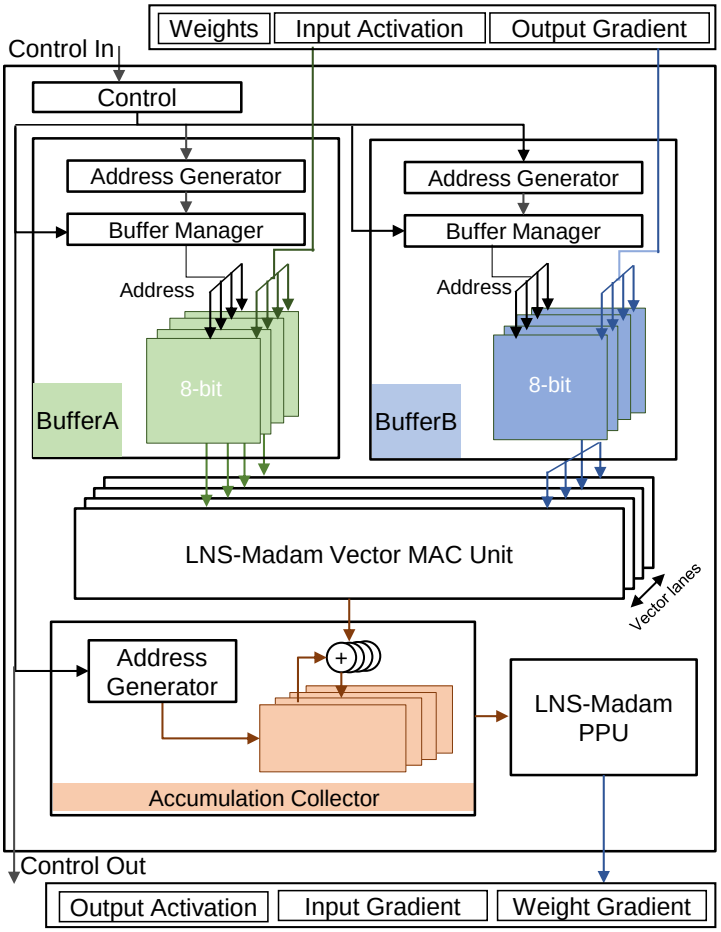

Figure 5.5: LNS-Madam processing element (PE).

Additional microarchitectural details of the PE are listed in Table 5.6. Notably, our accelerator uses a multi-level dataflow called output-stationary local-A-stationary(Venkatesan et al., 2019) to optimize reuse across different operands. Inputs from buffer A are read out once every 16 cycles and stored in a register for temporal reuse. Inputs from buffer B are read once every cycle and reused across the 32 lanes spatially. Partial sums are temporally accumulated in a 16-entry latch array collector before sending the completed sum to the post-processing unit. The two buffers in the PE store different data depending on whether output activation, input gradient, or weight gradient is being computed. For example, weights and input activations are stored in `BufferA` and `BufferB` respectively during forward propagation to compute the output activations. On the other hand, input activations and output gradients are stored in the respective buffers during backward propagation to compute the weight gradient. Table 5.2 outlines how we map various tensors in DNN computation

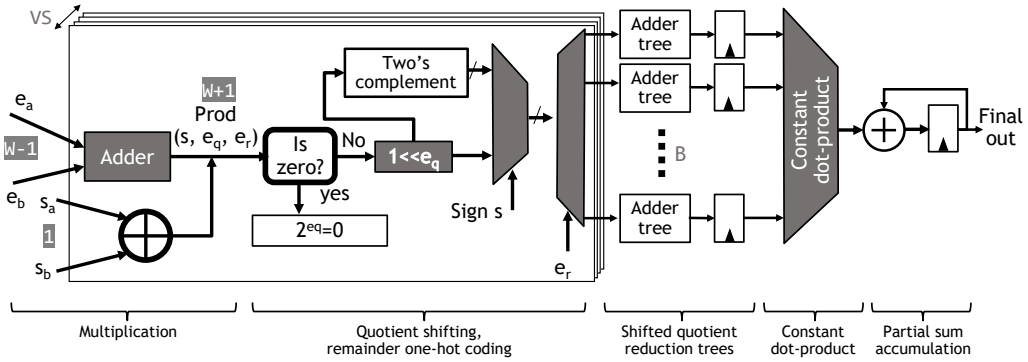

Figure 5.6: LNS-Madam Vector MAC Unit – Performs dot-products of inputs represented in LNS and produces partial sum outputs in integer format. Bitwidths of different signals are highlighted. `VS` stands for vector size; `W` stands for bitwidth of input values; `B` refers to base factor and number of remainder bins.

to buffers in our hardware during different computation passes. Note that weight updates are performed outside of the PEs through the global buffer.

Table 5.1: Microarchitectural details of LNS-Madam PE

Dataflow	Multi-level
Vector size / # Vector lanes	32
Weight/activation precision	8-bit
Gradient precision	8-bit
# Remainder Bins	8
Accumulation precision	24-bit
Accumulation collector size	1.5 KB
BufferA size	128 KB
BufferB size	8 KB

5.7 Experiments

In this section, we evaluate both the accuracy and energy efficiency of using LNS-Madam to train state-of-the-art models on large-scale datasets.

Model Accuracy

To evaluate accuracy, we simulate LNS-Madam using a PyTorch-based neural network quantization library that implements a set of common neural network layers (e.g., convolution, fully-connected) for training and inference in both full and quantized modes (H. Wu et al., 2020). The baseline library supports integer quantization in a fixed-point number system, and we further extend it to support LNS. The library also provides utilities for scaling values to the representable integer range of

Table 5.2: Mapping of tensors to buffers in PE during different computation passes[1]

Pass	BufferA	BufferB
Forward	Weight	Input Activation
Backward (Input)	Weight	Output Gradient
Backward (Weight)	Input Activation	Output Gradient

[1] Backward pass consists of backward computation for input gradient, denoted `Backward(Input)`, and backward computation for weight gradient, denoted `Backward(Weight)`.

Table 5.3: Base Factor Selection on ImageNet[1,2]

γ	Dynamic Range	Forward	Backward
1	(0,127)	NaN	NaN
2	(0,63.5)	75.81	75.79
4	(0,31.8)	75.96	76.07
8	(0,15.9)	75.88	76.23
16	(0,7.9)	76.32	63.67
32	(0,4.0)	68.15	20.71

[1] Bitwidth is 8-bit across settings. Quant Forward or Quant Backward denotes the settings where either forward propagation or backward propagation is quantized while leaving the rest of computation in full-precision.
[2] The results of test accuracy (%) are listed.

the specific number format. With this library, a typical quantized layer consists of a conventional layer implemented in floating-point preceded by a weight quantizer and an input quantizer that converts the weights and inputs of the layer to the desired quantized format. For the backward pass, after the gradients pass through the STE in each quantizer, they will be quantized by their quantizers as well.

We benchmark LNS-Madam on various tasks including ResNet models on CIFAR-10 and ImageNet, and BERT-base and BERT-large language models on SQuAD and GLUE. Specifically, we train ResNet models from scratch on CIFAR-10 and ImageNet, and fine-tune pre-trained BERT models on SQuAD and GLUE. Detailed descriptions of datasets and models can be found in the appendix.

Parameter Settings

We fix the bitwidth to be 8-bit for both forward and backward propagation, which includes the bitwidth for weights, activations, activation gradients, and weight gradients. We note that 8-bit weight gradients are lower than previous studies that use 16-bit or even 32-bit weight gradients (Sun, N. Wang, et al., 2020; Miyashita, Edward H. Lee, and Murmann, 2016).

To find an appropriate base factor γ under the 8-bit setting, we vary γ to find the appropriate dynamic ranges for forward and backward qantization. The dynamic range in LNS is $(0, (2^{\mathcal{B}-1} - 1)/\gamma)$, which is controlled by both bitwidth and base factor.

As shown in Table 5.3, we fix the bitwidth as 8-bit and vary the base factor γ to find the appropriate dynamic ranges for forward and backward quantization. According to the results, we find the base factor of 8 with the dynamic range $(0, 15.9)$ that uniformly works across $\mathrm{Q_W}$,$\mathrm{Q_A}$,$\mathrm{Q_E}$, and $\mathrm{Q_G}$.

In order to maintain optimization stability, the bitwidth of the weight updates require to be larger than the bitwidth of the weights. When the bitwidth of $\mathrm{Q_U}$ is larger than 8-bit, we increase its base factor to match the desired dynamic range $(0, 15.9)$.

We empirically search the best learning rate η for our Madam optimizer from 2^{-4} to 2^{-10}, and we find $\eta = 2^{-7}$ works best uniformly across tasks, which suggests the learning rate for Madam is robust.

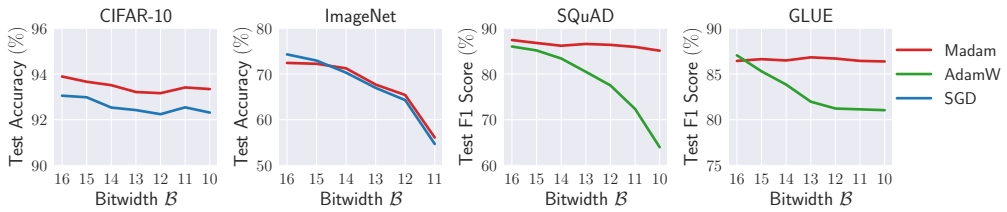

Figure 5.7: Comparing Madam with SGD and Adam optimizers under the logarithmic quantized weight update (defined in Equation 5.4). The bitwidth of the weight update Q_U is varied from 16-bit to 10-bit.

Comparisons

Given the settings above, we compare LNS-Madam with FP8 and FP32. For FP8 and FP32, the standard optimizers are applied for tasks by default. We use a tuned SGD optimizer for CIFAR-10 and ImageNet datasets, and a tuned AdamW optimizer for SQuAD and GLUE datasets. For all settings, we use per-channel scaling for

Table 5.4: Benchmarking LNS-Madam on various datasets and models[1]

Dataset	Model	LNS-Madam[2]	FP8[2]	FP32
CIFAR-10	ResNet-18	93.41	93.12	93.51
ImageNet	ResNet-50	76.14	75.83	76.38
SQuAD	BERT-base	88.13	88.07	88.36
SQuAD	BERT-large	90.75	90.54	90.80
GLUE	BERT-base	88.89	88.73	88.92
GLUE	BERT-large	89.24	88.91	89.35

[1] The results of test accuracy (%) are listed.
[2] Forward and backward propagation are in 8-bit, and the weight update is in 16-bit.

Table 5.5: Comparing LNS-Madam with recent low-precision training methods on 8-bit training[1]

	Data format	16-bit	32-bit
LNS-Madam	LNS	**76.14**	76.23
BHQ Chen et al., 2020	INT	74.89	**76.35**
Unified INT8 Zhu et al., 2020	INT	74.73	76.27
FP8 N. Wang et al., 2018	FP	71.46	71.53

[1] Evaluate ResNet-50 on ImageNet. Forward and backward propagation are in 8-bit. Test accuracy (%) evaluated under 32-bit and 16-bit weight update are presented.

Table 5.6: Comparing LNS-Madam and BHQ over a range of bitwidth[1]

	4-bit	5-bit	6-bit	7-bit	8-bit
LNS-Madam	**74.23**	**75.89**	74.41	**76.16**	76.23
BHQ Chen et al., 2020	74.04	75.70	**76.21**	76.14	**76.35**

[1] Bitwidth of activation gradients varies from 4-bit to 8-bit. The results of test accuracy (%) are listed.

ResNet and per-feature scaling for BERT, both of which are commonly used scaling techniques. The clamping function is performed by matching the largest value within each group of numbers.

To demonstrate LNS-Madam is better than popular number systems, we compare it with both FP8 and FP32, where our FP8 contains 4-bit exponent and 3-bit mantissa. As shown in Table 5.4, LNS-Madam yields better performance than FP8, and it even achieves performance comparable to the full-precision counterpart.

In addition, we compare LNS-Madam with recent methods on low-precision training. For all methods, we fix forward and backward propagation in 8-bit while varying the weight update precision from 32-bit to 16-bit. As shown in Table 5.5, LNS-Madam achieves the best accuracy under 16-bit weight update, which demonstrates its effectiveness under the quantized setting. FP8 (N. Wang et al., 2018) also achieves negligible degradation after switching to 16-bit, as it applies stochastic rounding over weight update process. Since BHQ (Chen et al., 2020) achieves the best accuracy under 32-bit, we also compare it with LNS-Madam over a range of bitwidth settings in Table 5.6.

We also compare Madam with the default optimizers SGD and AdamW under logarithmic quantized weight update, as defined in Equation 5.4. All optimizers use the same learning rates as above.

As shown in Fig. 5.7, we vary the bitwidth of the quantized weight update Q_U from 16-bit to 10-bit to test their performance over a wide range. The results suggest compared to other optimizers, *Madam always maintains higher accuracy when precision is severely limited*. Notably, for BERT model on SQuAD and GLUE benchmarks, Madam is 20% better than Adam with respect to F-1 score, when the weight update is in 10-bit. We observe large degradation for both Madam and SGD on ImageNet training, and we believe this is because the weights in some layers inevitably require higher precision settings. We leave it as future work to explore LNS-Madam under a customized precision setting.

Table 5.7: Design tools used for LNS-Madam hardware experiments

HLS Compiler	Mentor Graphics Catapult HLS
Verilog simulator	Synopsys VCS
Logic synthesis	Synopsys Design Compiler
Place-and-route	Synopsys ICC2
Power Analysis	Synopsys PT-PX

Energy Efficiency

We leverage the hardware implementation described in Section 5.6 to evaluate the energy efficiency of LNS-Madam. We code the hardware model in C++ and Verilog and synthesize it to a combined cycle-accurate RTL using a commercial high-level synthesis tool (McFarland, Parker, and Camposano, 1990). Once the RTL is generated, a standard logic synthesis flow is used to obtain the gate-level netlist that is then simulated with representative inputs. To extract energy consumption, we supply the gate-level simulation results from post-synthesis to a standard power analysis tool. We then use an analytical model to compute the total energy consumption for different workloads. We perform our analysis in a sub-16nm state-of-the-art process technology at 0.6V targeting a frequency of 1.05 GHz. Table 5.7 summarizes the design tools used in the evaluation.

Table 5.8: Energy efficiency for different models and number formats[1]

Model	LNS	FP8	FP16	FP32
ResNet-18	0.54	1.22	2.50	5.99
ResNet-50	0.99	2.25	4.59	11.03
BERT-Base	7.99	18.23	37.21	89.35
BERT-Large	27.85	63.58	129.74	311.58

[1] The per-iteration energy consumption in mJ are listed.

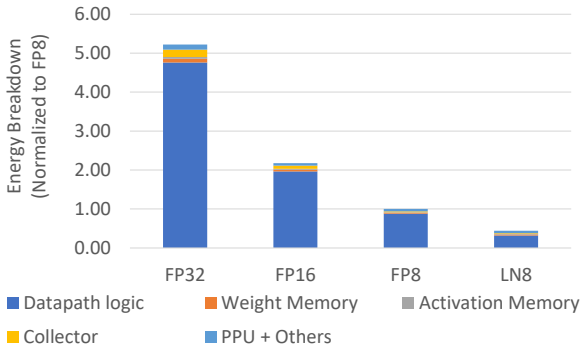

Figure 5.8: Energy breakdown of the PE shown in Fig. 5.6 for different dataformats.

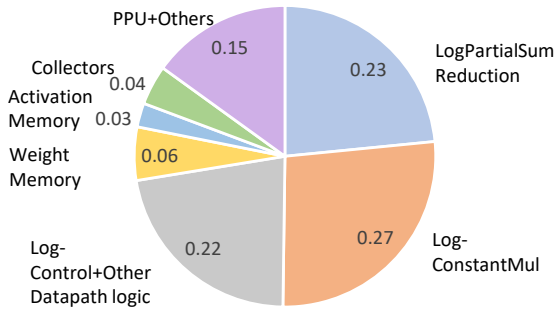

Figure 5.9: LNS PE energy breakdown showing different components of datapath

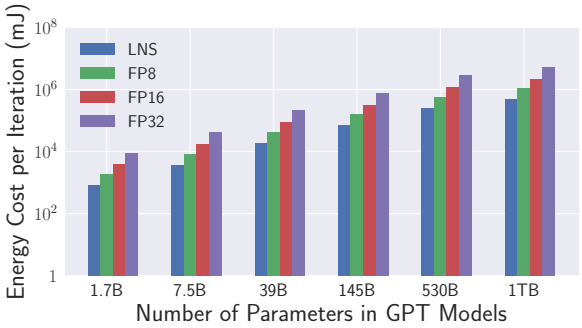

Figure 5.10: Energy efficiency over a range of GPT models from 1 billion to 1 trillion parameters. The models are scaled by a throughput efficient method proposed by Narayanan et al. Narayanan et al., 2021.

In our experiment, the LNS-Madam hardware is designed with bitwidth $\mathcal{B} = 8$ and base factor $\gamma = 8$ for both forward and backward computations. In addition to experimenting with the LNS-based datapath shown in Fig. 5.6, we also consider FP8, FP16, and FP32 datapath baselines for comparison.

Table 5.8 presents the energy efficiency per iteration of one forward pass and one backward pass of training. Because different number systems share the same training iterations, per-iteration energy results imply the energy comparison over the entire training. We also present the energy breakdown of the whole PE in Figure 5.8. As shown in the figure, FP arithmetic is extremely expensive, contributing a large fraction to the total energy consumption of PEs. The proposed LNS datapath offers significant reduction in the logic complexity, leading to 2.2X, 4.6X, and 11X energy efficiency improvements over FP8, FP16, and FP32 implementations, respectively. We also provide a detailed energy breakdown showing different components of the LNS PE in Fig. 5.9. In addition, Fig. 5.10 shows the energy efficiency over a range

of GPT models from 1 billion to 1 trillion parameters.

5.8 Conclusions

In this work, we propose a co-designed low-precision training framework LNS-Madam that jointly considers the logarithmic number system and the multiplicative weight update algorithm. Experimental results show that LNS-Madam achieves comparable accuracy to full-precision counterparts even when forward and backward propagation, and weight updates are all in low-precision. To support the training framework in practice, we design a hardware implementation of LNS-Madam to efficiently perform the necessary LNS computations for DNN training. Based on our energy analysis, LNS-Madam reduces energy consumption by over 90% compared to a floating-point baseline.

An important application of our low-precision training framework is learning neural networks over energy-constrained edge devices. This is fundamental for intelligent edge devices to easily adapt to changing and non-stationary environments by learning on-device and on-the-fly. By enabling highly energy-efficient training, our work carries the promising opportunity for using LNS-based hardware to conduct environmental-friendly deep learning research in the near future.

References

Banner, Ron et al. (2018). "Scalable methods for 8-bit training of neural networks". In: *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*. Ed. by Samy Bengio et al., pp. 5151–5159. URL: https://proceedings.neurips.cc/paper/2018/hash/e82c4b19b8151ddc25d4d93baf7b908f-Abstract.html.

Bartol Thomas M, Jr et al. (Nov. 2015). "Nanoconnectomic upper bound on the variability of synaptic plasticity". In: *eLife* 4. Ed. by Sacha B Nelson, e10778. ISSN: 2050-084X. DOI: 10.7554/eLife.10778. URL: https://doi.org/10.7554/eLife.10778.

Bengio, Yoshua, Nicholas Léonard, and Aaron Courville (2013). "Estimating or propagating gradients through stochastic neurons for conditional computation". In: *arXiv preprint arXiv:1308.3432*.

Bernstein, Jeremy, Yu-Xiang Wang, et al. (Oct. 2018). "signSGD: Compressed Optimisation for Non-Convex Problems". In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, pp. 560–569. URL: https://proceedings.mlr.press/v80/bernstein18a.html.

Bernstein, Jeremy, Jiawei Zhao, Kamyar Azizzadenesheli, et al. (2019). "signSGD with Majority Vote is Communication Efficient and Fault Tolerant". In: *International Conference on Learning Representations (ICLR)*. URL: https://openreview.net/forum?id=BJxhijAcY7.

Bernstein, Jeremy, Jiawei Zhao, Markus Meister, et al. (2020). "Learning compositional functions via multiplicative weight updates". In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. Ed. by Hugo Larochelle et al. URL: https://proceedings.neurips.cc/paper/2020/hash/9a32ef65c42085537062753ec435750f-Abstract.html.

Chen, Jianfei et al. (2020). "A Statistical Framework for Low-bitwidth Training of Deep Neural Networks". In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. Ed. by Hugo Larochelle et al. URL: https://proceedings.neurips.cc/paper/2020/hash/099fe6b0b444c23836c4a5d07346082b-Abstract.html.

Chmiel, Brian et al. (2021). "Neural gradients are near-lognormal: improved quantized and sparse training". In: *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net. URL: https://openreview.net/forum?id=EoFNy62JGd.

Frankle, Jonathan and Michael Carbin (2019). "The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks". In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net. URL: https://openreview.net/forum?id=rJl-b3RcF7.

Freund, Yoav and Robert E Schapire (1997). "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting". In: *Journal of Computer and System Sciences*.

Gupta, Suyog et al. (2015). "Deep Learning with Limited Numerical Precision". In: *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*. Ed. by Francis R. Bach and David M. Blei. Vol. 37. JMLR Workshop and Conference Proceedings. JMLR.org, pp. 1737–1746. URL: http://proceedings.mlr.press/v37/gupta15.html.

Jacob, Benoit et al. (2018). "Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference". In: *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*. IEEE Computer Society, pp. 2704–2713. DOI: 10.1109/CVPR.2018.00286. URL: http://openaccess.thecvf.com/content%5C_cvpr%5C_2018/html/Jacob%5C_Quantization%5C_and%5C_Training%5C_CVPR%5C_2018%5C_paper.html.

Johnson, J. (2020). "Efficient, arbitrarily high precision hardware logarithmic arithmetic for linear algebra". In: *2020 IEEE 27th Symposium on Computer Arithmetic (ARITH)*, pp. 25–32. DOI: 10.1109/ARITH48897.2020.00013.

Johnson, Jeff (2018). "Rethinking floating point for deep learning". In: *ArXiv preprint* abs/1811.01721. URL: https://arxiv.org/abs/1811.01721.

Kivinen, Jyrki and Manfred K. Warmuth (1997). "Exponentiated Gradient versus Gradient Descent for Linear Predictors". In: *Information and Computation*.

Lee, E. H. et al. (2017). "LogNet: Energy-efficient neural networks using logarithmic computation". In: *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 5900–5904. DOI: 10.1109/ICASSP.2017.7953288.

McFarland, M.C., A.C. Parker, and R. Camposano (1990). "The high-level synthesis of digital systems". In: *Proceedings of the IEEE* 78.2, pp. 301–318.

Mitchell, J. N. (1962). "Computer Multiplication and Division Using Binary Logarithms". In: *IRE Transactions on Electronic Computers* EC-11.4, pp. 512–517. DOI: 10.1109/TEC.1962.5219391.

Miyashita, Daisuke, Edward H. Lee, and Boris Murmann (2016). *Convolutional Neural Networks using Logarithmic Data Representation*. arXiv: 1603.01025 [cs.NE].

Narayanan, Deepak et al. (2021). *Efficient Large-Scale Language Model Training on GPU Clusters*. arXiv: 2104.04473 [cs.CL].

Rastegari, Mohammad et al. (2016). "Xnor-net: Imagenet classification using binary convolutional neural networks". In: *European conference on computer vision*. Springer, pp. 525–542.

Sa, Christopher De et al. (2018). *High-Accuracy Low-Precision Training*. arXiv: 1803.03383 [cs.LG].

Saadat, Hassaan, Haseeb Bokhari, and Sri Parameswaran (2018). "Minimally biased multipliers for approximate integer and floating-point multiplication". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37.11, pp. 2623–2635.

Saadat, Hassaan, Haris Javaid, Aleksandar Ignjatovic, et al. (2020). "REALM: reduced-error approximate log-based integer multiplier". In: *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, pp. 1366–1371.

Saadat, Hassaan, Haris Javaid, and Sri Parameswaran (2019). "Approximate integer and floating-point dividers with near-zero error bias". In: *2019 56th ACM/IEEE Design Automation Conference (DAC)*. IEEE, pp. 1–6.

Sakr, Charbel and Naresh R. Shanbhag (2019). "Per-Tensor Fixed-Point Quantization of the Back-Propagation Algorithm". In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net. URL: https://openreview.net/forum?id=rkxaNjA9Ym.

Sun, Xiao, Jungwook Choi, et al. (2019). "Hybrid 8-bit Floating Point (HFP8) Training and Inference for Deep Neural Networks". In: *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*. Ed. by Hanna M. Wallach et al., pp. 4901–4910. URL: https://proceedings.neurips.cc/paper/2019/hash/65fc9fb4897a89789352e211ca2d398f-Abstract.html.

Sun, Xiao, Naigang Wang, et al. (2020). "Ultra-Low Precision 4-bit Training of Deep Neural Networks". In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. Ed. by Hugo Larochelle et al. URL: https://proceedings.neurips.cc/paper/2020/hash/13b919438259814cd5be8cb45877d577-Abstract.html.

Venkatesan, Rangharajan et al. (2019). "MAGNet: A Modular Accelerator Generator for Neural Networks". In: *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–8.

Vogel, Sebastian et al. (2018). "Efficient Hardware Acceleration of CNNs Using Logarithmic Data Representation with Arbitrary Log-Base". In: *Proceedings of the International Conference on Computer-Aided Design*. ICCAD '18. San Diego, California: Association for Computing Machinery. ISBN: 9781450359504. DOI: 10.1145/3240765.3240803. URL: https://doi.org/10.1145/3240765.3240803.

Wang, Naigang et al. (2018). "Training Deep Neural Networks with 8-bit Floating Point Numbers". In: *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*. Ed. by Samy Bengio et al., pp. 7686–7695. URL: https://proceedings.neurips.cc/paper/2018/hash/335d3d1cd7ef05ec77714a215134914c-Abstract.html.

Wu, Hao et al. (2020). "Integer Quantization for Deep Learning Inference: Principles and Empirical Evaluation". In: *ArXiv preprint* abs/2004.09602. URL: https://arxiv.org/abs/2004.09602.

Wu, Shuang et al. (2018). "Training and Inference with Integers in Deep Neural Networks". In: *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net. URL: https://openreview.net/forum?id=HJGXzmspb.

Zhao, Yongwei et al. (June 2021). "Cambricon-Q: A Hybrid Architecture for Efficient Training". en. In: *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. Valencia, Spain: IEEE, pp. 706–719. ISBN: 978-1-66543-333-4. DOI: 10.1109/ISCA52012.2021.00061. URL: https://ieeexplore.ieee.org/document/9499944/ (visited on 05/15/2022).

Zhou, Aojun et al. (2017). "Incremental Network Quantization: Towards Lossless CNNs with Low-precision Weights". In: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net. URL: https://openreview.net/forum?id=HyQJ-mclg.

Zhou, Shuchang et al. (2016). "Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients". In: *ArXiv preprint* abs/1606.06160. URL: https://arxiv.org/abs/1606.06160.

Zhu, Feng et al. (2020). "Towards Unified INT8 Training for Convolutional Neural Network". In: pp. 1969–1979. URL: https://openaccess.thecvf.com/content_CVPR_2020/html/Zhu_Towards_Unified_INT8_Training_for_Convolutional_Neural_Network_CVPR_2020_paper.html (visited on 12/08/2020).

Appendix A

APPENDIX - CHAPTER 2

A.1 Additional Related Works

Adafactor (Shazeer and Stern, 2018) achieves sub-linear memory cost by factorizing the second-order statistics by a row-column outer product. GaLore shares similarities with Adafactor in terms of utilizing low-rank factorization to reduce memory cost, but GaLore focuses on the low-rank structure of the gradients, while Adafactor focuses on the low-rank structure of the second-order statistics.

GaLore can reduce the memory cost for both first-order and second-order statistics, and can be combined with Adafactor to achieve further memory reduction. In contrast to the previous memory-efficient optimization methods, GaLore operates independently as the optimizers directly receive the low-rank gradients without knowing their full-rank counterparts.

The fused backward operation proposed by LOMO (Lv, Yang, et al., 2023) mitigates the memory cost of storing weight gradients during training. Integrated with the standard SGD optimizer, LOMO achieves zero optimizer and gradient memory cost during training. AdaLOMO (Lv, Yan, et al., 2023) enhances this approach by combining the fused backward operation with adaptive learning rate for each parameter, similarly achieving minimal optimizer memory cost.

While LOMO and AdaLOMO represent significant advancements in memory-efficient optimization for fine-tuning or continual pre-training, they might not be directly applicable to pre-training from scratch at larger scales. For example, the vanilla Adafactor, adopted by AdaLOMO, has been demonstrated to lead to increased training instabilities at larger scales (Rae et al., 2021; Chowdhery et al., 2023; Wortsman et al., 2023; Zhai et al., 2022). We believe integrating GaLore with the fused backward operation may offer a promising avenue for achieving memory-efficient large-scale pre-training from scratch.

A.2 Proofs

Reversibility

Definition A.2.1 (Reversiblity Tian, Yu, et al., 2020) *A network $\mathcal{N}$ that maps input $\boldsymbol{x}$ to output $\boldsymbol{y} = \mathcal{N}(\boldsymbol{x})$ is* reversible*, if there exists $L(\boldsymbol{x}; W)$ so that $\boldsymbol{y} = L(\boldsymbol{x}; W)\boldsymbol{x}$,*

and the backpropagated gradient $\boldsymbol{g}_{\boldsymbol{x}}$ *satisfies* $\boldsymbol{g}_{\boldsymbol{x}} = L^\top(\boldsymbol{x}; W)\boldsymbol{g}_{\boldsymbol{y}}$*, where* $\boldsymbol{g}_{\boldsymbol{y}}$ *is the backpropagated gradient at the output* $\boldsymbol{y}$*. Here* $L(\boldsymbol{x}; W)$ *depends on the input* $\boldsymbol{x}$ *and weight* W *in the network* $\mathcal{N}$*.*

Note that many layers are reversible, including linear layer (without bias), reversible activations (e.g., ReLU, leaky ReLU, polynomials, etc). Furthermore, they can be combined to construct more complicated architectures:

Property 1 *If* $\mathcal{N}_1$ *and* $\mathcal{N}_2$ *are reversible networks, then (***Parallel***)* $\boldsymbol{y} = \alpha_1\mathcal{N}_1(\boldsymbol{x}) + \alpha_2\mathcal{N}_2(\boldsymbol{x})$ *is reversible for constants* α_1 *and* α_2*, and (***Composition***)* $\boldsymbol{y} = \mathcal{N}_2(\mathcal{N}_1(\boldsymbol{x}))$ *is reversible.*

From this property, it is clear that ResNet architecture $\boldsymbol{x} + \mathcal{N}(\boldsymbol{x})$ is reversible, if $\mathcal{N}$ contains bias-free linear layers and reversible activations, which is often the case in practice. For a detailed analysis, please check Appendix A in Tian, Yu, et al., 2020. For architectures like self-attention, one possibility is to leverage JoMA Tian, Wang, et al., 2024 to analyze, and we leave for future work.

The gradient of chained reversible networks has the following structure:

Theorem 2.3.2 (Gradient Form of reversible models) *Consider a chained reversible neural network* $\mathcal{N}(\boldsymbol{x}) := \mathcal{N}_L(\mathcal{N}_{L-1}(\ldots\mathcal{N}_1(\boldsymbol{x})))$ *and define* $J_l := \text{Jacobian}(\mathcal{N}_L)\ldots\text{Jacobian}(\mathcal{N}_{l+1})$ *and* $\boldsymbol{f}_l := \mathcal{N}_l(\ldots\mathcal{N}_1(\boldsymbol{x}))$*. Then the weight matrix* W_l *at layer* l *has gradient* G_l *in the following form for batch size 1:*

(a) *For* ℓ_2*-objective* $\varphi := \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{f}_L\|_2^2$*:*

$$G_l = \left(J_l^\top \boldsymbol{y} - J_l^\top J_l W_l \boldsymbol{f}_{l-1}\right)\boldsymbol{f}_{l-1}^\top, \tag{2.6}$$

(b) *Left* $P_{\mathbf{1}}^\perp := I - \frac{1}{K}\mathbf{1}\mathbf{1}^\top$ *be the zero-mean PSD projection matrix. For* K*-way logsoftmax loss* $\varphi(\boldsymbol{y}; \boldsymbol{f}_L) := -\log\left(\frac{\exp(\boldsymbol{y}^\top \boldsymbol{f}_L)}{\mathbf{1}^\top\exp(\boldsymbol{f}_L)}\right)$ *with small logits* $\|P_{\mathbf{1}}^\perp \boldsymbol{f}_L\|_\infty \ll \sqrt{K}$*:*

$$G_l = \left(J_l P_{\mathbf{1}}^\perp \boldsymbol{y} - \gamma K^{-1} J_l^\top P_{\mathbf{1}}^\perp J_l W_l \boldsymbol{f}_{l-1}\right)\boldsymbol{f}_{l-1}^\top, \tag{2.7}$$

where $\gamma \approx 1$ *and* $\boldsymbol{y}$ *is a data label with* $\boldsymbol{y}^\top\mathbf{1} = 1$*.*

Proof A.2.1.1 *Note that for layered reversible network, we have*

$$\mathcal{N}(\boldsymbol{x}) = \mathcal{N}_L(\mathcal{N}_{L-1}(\ldots\mathcal{N}_1(\boldsymbol{x}))) = K_L(\boldsymbol{x})K_{L-1}(\boldsymbol{x})\ldots K_1(\boldsymbol{x})\boldsymbol{x} \tag{A.1}$$

Let $\boldsymbol{f}_l := \mathcal{N}_l(\mathcal{N}_{l-1}(\ldots\mathcal{N}_1(\boldsymbol{x})))$ and $J_l := K_L(\boldsymbol{x})\ldots K_{l+1}(\boldsymbol{x})$, and for linear layer l, we can write $\mathcal{N}(\boldsymbol{x}) = J_l W_l \boldsymbol{f}_{l-1}$. Therefore, for the linear layer l with weight matrix W_l, we have:

$$\begin{aligned}
\mathrm{d}\varphi &= (\boldsymbol{y} - \mathcal{N}(\boldsymbol{x}))^\top \mathrm{d}\mathcal{N}(\boldsymbol{x}) && \text{(A.2)}\\
&= (\boldsymbol{y} - \mathcal{N}(\boldsymbol{x}))^\top K_L(\boldsymbol{x})\ldots K_{l+1}(\boldsymbol{x})\mathrm{d}W_l \boldsymbol{f}_{l-1} \;+\; \text{terms not related to } \mathrm{d}W_l && \text{(A.3)}\\
&= (\boldsymbol{y} - J_l W_l \boldsymbol{f}_{l-1})^\top J_l \mathrm{d}W_l \boldsymbol{f}_{l-1} && \text{(A.4)}\\
&= \mathrm{tr}(\mathrm{d}W_l^\top J_l^\top (\boldsymbol{y} - J_l W_l \boldsymbol{f}_{l-1})\boldsymbol{f}_{l-1}^\top) && \text{(A.5)}
\end{aligned}$$

This gives the gradient of W_l:

$$G_l = J_l^\top \boldsymbol{y}\boldsymbol{f}_{l-1}^\top - J_l^\top J_l W_l \boldsymbol{f}_{l-1}\boldsymbol{f}_{l-1}^\top \tag{A.6}$$

Softmax Case. Note that for softmax objective with small logits, we can also prove a similar structure of backpropagated gradient, and thus Theorem 2.3.2 can also apply.

Lemma A.2.2 (Gradient structure of softmax loss) *For K-way logsoftmax loss $\varphi(\boldsymbol{y}; \boldsymbol{f}) := -\log\left(\frac{\exp(\boldsymbol{y}^\top \boldsymbol{f})}{\mathbf{1}^\top \exp(\boldsymbol{f})}\right)$, let $\hat{\boldsymbol{f}} = P_{\mathbf{1}}^\perp \boldsymbol{f}$ be the zero-mean version of network output $\boldsymbol{f}$, where $P_{\mathbf{1}}^\perp := I - \frac{1}{K}\mathbf{1}\mathbf{1}^\top$, then we have:*

$$-\mathrm{d}\varphi = \boldsymbol{y}^\top \mathrm{d}\hat{\boldsymbol{f}} - \gamma \hat{\boldsymbol{f}}^\top \mathrm{d}\hat{\boldsymbol{f}}/K + O(\hat{\boldsymbol{f}}^2/K)\mathrm{d}\hat{\boldsymbol{f}} \tag{A.7}$$

where $\gamma(\boldsymbol{y}, \boldsymbol{f}) \approx 1$ and $\boldsymbol{y}$ is a data label with $\boldsymbol{y}^\top \mathbf{1} = 1$.

Proof A.2.2.1 *Let $\hat{\boldsymbol{f}} := P_{\mathbf{1}}^\perp \boldsymbol{f}$ be the zero-mean version of network output $\boldsymbol{f}$. Then we have $\mathbf{1}^\top \hat{\boldsymbol{f}} = 0$ and $\boldsymbol{f} = \hat{\boldsymbol{f}} + c\mathbf{1}$. Therefore, we have:*

$$-\varphi = \log\left(\frac{\exp(c)\exp(\boldsymbol{y}^\top \hat{\boldsymbol{f}})}{\exp(c)\mathbf{1}^\top \exp(\hat{\boldsymbol{f}})}\right) = \boldsymbol{y}^\top \hat{\boldsymbol{f}} - \log(\mathbf{1}^\top \exp(\hat{\boldsymbol{f}})) \tag{A.8}$$

Using the Taylor expansion $\exp(x) = 1 + x + \frac{x^2}{2} + o(x^2)$, we have:

$$\mathbf{1}^\top \exp(\hat{\boldsymbol{f}}) = \mathbf{1}^\top(\mathbf{1} + \hat{\boldsymbol{f}} + \frac{1}{2}\hat{\boldsymbol{f}}^2) + o(\hat{\boldsymbol{f}}^2) = K(1 + \hat{\boldsymbol{f}}^\top \hat{\boldsymbol{f}}/2K + o(\hat{\boldsymbol{f}}^2/K)) \tag{A.9}$$

So

$$-\varphi = \boldsymbol{y}^\top \hat{\boldsymbol{f}} - \log(1 + \hat{\boldsymbol{f}}^\top \hat{\boldsymbol{f}}/2K + o(\hat{\boldsymbol{f}}^2/K)) - \log K \tag{A.10}$$

Therefore

$$-\mathrm{d}\varphi = \boldsymbol{y}^\top \mathrm{d}\hat{\boldsymbol{f}} - \frac{\gamma}{K}\hat{\boldsymbol{f}}^\top \mathrm{d}\hat{\boldsymbol{f}} + O\left(\frac{\hat{\boldsymbol{f}}^2}{K}\right)\mathrm{d}\hat{\boldsymbol{f}} \tag{A.11}$$

where $\gamma := (1 + \hat{\boldsymbol{f}}^\top \hat{\boldsymbol{f}}/2K + o(\hat{\boldsymbol{f}}^2/K))^{-1} \approx 1$.

Remarks. With this lemma, it is clear that for a reversible network $\boldsymbol{f} := \mathcal{N}(\boldsymbol{x}) = J_l(\boldsymbol{x})W_l\boldsymbol{f}_{l-1}(\boldsymbol{x})$, the gradient G_l of W_l has the following form:

$$G_l = \underbrace{J_l P_{\mathbf{1}}^{\perp} \boldsymbol{y} \boldsymbol{f}_{l-1}}_{A} - \underbrace{\gamma J_l^{\top} P_{\mathbf{1}}^{\perp} J_l}_{B} W_l \underbrace{\boldsymbol{f}_{l-1}\boldsymbol{f}_{l-1}^{\top}/K}_{C} \tag{A.12}$$

Gradient becomes low-rank

Lemma A.2.3 (Gradient becomes low-rank during training) *Suppose the gradient follows the parametric form:*

$$G_t = \frac{1}{N}\sum_{i=1}^{N}(A_i - B_i W_t C_i) \tag{2.8}$$

with constant A_i, PSD matrices B_i and C_i after $t \geq t_0$. We study vanilla SGD weight update: $W_t = W_{t-1} + \eta G_{t-1}$. Let $S := \frac{1}{N}\sum_{i=1}^{N} C_i \otimes B_i$ and $\lambda_1 < \lambda_2$ its two smallest distinct eigenvalues. Then the stable rank $\mathrm{sr}(G_t)$ satisfies:

$$\mathrm{sr}(G_t) \leq \mathrm{sr}(G_{t_0}^{\parallel}) + \left(\frac{1-\eta\lambda_2}{1-\eta\lambda_1}\right)^{2(t-t_0)} \frac{\|G_0 - G_{t_0}^{\parallel}\|_F^2}{\|G_{t_0}^{\parallel}\|_2^2}, \tag{2.9}$$

where $G_{t_0}^{\parallel}$ is the projection of G_{t_0} onto the minimal eigenspace $\mathcal{V}_1$ of S corresponding to λ_1.

Proof A.2.3.1 *We have*

$$G_t = \frac{1}{N}\sum_{i=1}^{N}(A_i - B_i W_t C_i) = \frac{1}{N}\sum_{i=1}^{N} A_i - B_i(W_{t-1} + \eta G_{t-1})C_i = G_{t-1} - \frac{\eta}{N}\sum_{i=1}^{N} B_i G_{t-1} C_i \tag{A.13}$$

Let $S := \frac{1}{N}\sum_{i=1}^{N} C_i \otimes B_i$, and $g_t := \mathrm{vec}(G_t) \in \mathrm{r}^{mn}$ be a vectorized version of the gradient $G_t \in \mathrm{r}^{m\times n}$. Using $\mathrm{vec}(BWC) = (C^{\top} \otimes B)\mathrm{vec}(W)$, we have:

$$g_t = (I - \eta S) g_{t-1} \tag{A.14}$$

Now let's bound the stable rank of G_t:

$$\textit{stable-rank}(G_t) := \frac{\|G_t\|_F^2}{\|G_t\|_2^2} \tag{A.15}$$

Now $\lambda_1 < \lambda_2$ are the smallest two distinct eigenvectors of S. The smallest eigenvalue λ_1 has multiplicity κ_1. We can decompose g_0 into two components, $g_0 = g_0^{\parallel} + g_0^{\perp}$, in which $g_0^{\parallel}$ lies in the κ_1-dimensional eigenspace $\mathcal{V}_1$ that corresponds to the minimal

eigenvalue λ_1, and $g_0^\perp$ is its residue. Then $\mathcal{V}_1 \subset \mathrm{r}^{mn}$ and its orthogonal complements are invariant subspaces under S and thus:

$$\|G_t\|_F^2 = \|g_t\|_2^2 = \|(I-\eta S)^t g_0\|_2^2 = \|(I-\eta S)^t g_0^{\|}\|_2^2 + \|(I-\eta S)^t g_0^{\perp}\|_2^2 \quad \text{(A.16)}$$

$$\leq (1-\eta\lambda_2)^{2t}\|g_0^{\perp}\|_2^2 + (1-\eta\lambda_1)^{2t}\|g_0^{\|}\|_2^2 \quad \text{(A.17)}$$

On the other hand, by our assumption, $G_0^{\|}$ is rank L and thus has SVD decomposition:

$$G_0^{\|} = \sum_{l=1}^{L} c_l \boldsymbol{z}_l \boldsymbol{y}_l^\top \quad \text{(A.18)}$$

with orthonormal unit vectors $\{\boldsymbol{z}_l\}_{l=1}^L$ and $\{\boldsymbol{y}_l\}_{l=1}^L$ and singular values $\{c_l\}_{l=1}^L$1. This means that

$$g_0^{\|} = \mathrm{vec}(G_0^{\|}) = \sum_{l=1}^{L} c_l(\boldsymbol{y}_l \otimes \boldsymbol{z}_l) =: \sum_{l=1}^{L} c_l \boldsymbol{v}_l \quad \text{(A.19)}$$

with unit vector $\boldsymbol{v}_l := \boldsymbol{y}_l \otimes \boldsymbol{z}_l \in \mathcal{V}_1$. It is clear that

$$\boldsymbol{v}_l^\top \boldsymbol{v}_{l'} = (\boldsymbol{y}_l^\top \otimes \boldsymbol{z}_l^\top)(\boldsymbol{y}_{l'} \otimes \boldsymbol{z}_{l'}) = (\boldsymbol{y}_l^\top \boldsymbol{y}_{l'})(\boldsymbol{z}_l^\top \boldsymbol{z}_{l'}) = \mathbb{I}(l = l') \quad \text{(A.20)}$$

Therefore, by the definition of spectral norm (or matrix 2-norm), we know it corresponds to the largest singular value, which means:

$$\|G_t\|_2 = \max_{\|\boldsymbol{y}'\|_2=1, \|\boldsymbol{z}'\|_2=1} \boldsymbol{z}'^\top G_t \boldsymbol{y}' \quad \text{(A.21)}$$

$$\geq \max_l \boldsymbol{z}_l^\top G_t \boldsymbol{y}_l = \max_l (\boldsymbol{y}_l \otimes \boldsymbol{z}_l)^\top g_t \quad \text{(A.22)}$$

$$= \max_l \boldsymbol{v}_l^\top (1-\eta S)^t g_0 = (1-\eta\lambda_1)^t \max_l \boldsymbol{v}_l^\top g_0 \quad \text{(A.23)}$$

Note that the last equation is because any $\boldsymbol{v} \in \mathcal{V}_1$ is an eigenvector of S with eigenvalue of λ_1.

Since $\boldsymbol{v}_l^\top g_0 = \boldsymbol{v}_l^\top (g_0^\perp + g_0^{\|}) = c_l$, $\max_l c_l = \|G_0^{\|}\|_2$ and $\|g_0^{\|}\|_2^2 = \|G_0^{\|}\|_F^2$, we have:

$$\textit{stable-rank}(G_t) := \frac{\|G_t\|_F^2}{\|G_t\|_2^2} \leq \textit{stable-rank}(G_0^{\|}) + \left(\frac{1-\eta\lambda_2}{1-\eta\lambda_1}\right)^{2t} \frac{\|G_0^{\perp}\|_F^2}{\|G_0^{\|}\|_2^2} \quad \text{(A.24)}$$

Corollary A.2.4 (Low-rank G_t with special structure of $\mathcal{V}_1$) *If $\mathcal{V}_1(S)$ is 1-dim with decomposable eigenvector $\boldsymbol{v} = \boldsymbol{y} \otimes \boldsymbol{z}$, then $\mathrm{sr}(G_{t_0}^{\|}) = 1$ and thus G_t becomes rank-1.*

Proof A.2.4.1 *In this case, we have $g_0^{\|} = \boldsymbol{v}\boldsymbol{v}^\top g_0 \propto \boldsymbol{v}$. Since $\boldsymbol{v} = \boldsymbol{y} \otimes \boldsymbol{z}$, the resulting $G_0^{\|}$ is a rank-1 matrix and thus $\mathrm{sr}(G_{t_0}^{\|}) = 1$.*

Gradient Low-rank property for Transformers

Note that Transformers do not belong to the family of reversible networks. However, we can still show that the gradient of the lower layer (i.e., *project-up*) weight $W \in \mathrm{r}^{m\times n}$ of feed forward network (FFN) becomes low rank over time, using the JoMA framework Tian, Wang, et al., 2024. Here m is the embedding dimension, and n is the number of hidden nodes in FFNs.

Lemma A.2.5 (Gradient of Project-up in Transformer FFNs) *Suppose the embedding matrix $U \in \mathrm{r}^{m\times M}$ is fixed and column-orthonormal (M is vocabulary size), the activation functions are linear and the backpropagated gradient are stationary Tian, Wang, et al., 2024, then the training dynamics of transformed project-up matrix $V := U^\top W \in \mathrm{r}^{M\times n}$ satisfies the following:*

$$\dot{V} = \frac{1}{A}\,\mathrm{diag}\left(\exp\left(\frac{V\circ V}{2}\right)\mathbf{1}\right)\Delta \tag{A.25}$$

where A is the normalization factor of softmax, $\circ$ is the Hadamard (element-wise) product and Δ is defined in the proof. As a result, the gradient of V is "exponentially more low-rank" than V itself.

Proof A.2.5.1 *Let $\Delta := [\boldsymbol{\Delta}_1, \ldots, \boldsymbol{\Delta}_n] \in \mathrm{r}^{M\times n}$, where $\boldsymbol{\Delta}_j := \mathbb{E}_q[g_j \boldsymbol{x}] \in \mathrm{r}^M$. Here g_j is the backpropagated gradient of hidden node j in FFN layer, $\mathbb{E}_q[\cdot]$ is the conditional expectation given the query is token q, and $\boldsymbol{x}$ is the representation of token distribution in the previous layer of Transformer. Specifically, for intermediate layer, $\boldsymbol{x}$ represents the activation output of the previous project-up layer; for the first layer, $\boldsymbol{x}$ represents the frequency count of the input tokens. Then following the derivation of Theorem 2 Tian, Wang, et al., 2024, we have for each hidden node j and its weight $\boldsymbol{w}_j$, the transformed weight $\boldsymbol{v}_j := U^\top \boldsymbol{w}_j$ satisfies the following dynamics:*

$$\dot{\boldsymbol{v}}_j = \frac{1}{A}\boldsymbol{\Delta}_j \circ \exp(\boldsymbol{v}_j^2/2) \tag{A.26}$$

where $\boldsymbol{v}_j^2 := \boldsymbol{v}_j \circ \boldsymbol{v}_j$ is the element-wise square of a vector and $\circ$ is the Hadamard (element-wise) product. Since $V := [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n]$, Eqn. A.25 follows.

Note that the dynamics of $\boldsymbol{v}_j$ shows that the direction of $\boldsymbol{v}_j$ will change over time (because of $\exp(\boldsymbol{v}_j^2/2)$), and it is not clear how such dynamics leads to low-rank V

and even more low-rank $\dot{V}$. For this, we per-row decompose the matrix V:

$$V := \begin{bmatrix} \boldsymbol{u}_1^\top \\ \boldsymbol{u}_2^\top \\ \cdots \\ \boldsymbol{u}_M^\top \end{bmatrix} \tag{A.27}$$

where $\boldsymbol{u}_l \in \mathrm{r}^n$. We can also do the same for Δ:

$$\Delta := \begin{bmatrix} \boldsymbol{\mu}_1^\top \\ \boldsymbol{\mu}_2^\top \\ \cdots \\ \boldsymbol{\mu}_M^\top \end{bmatrix} \tag{A.28}$$

where $\boldsymbol{\mu}_l \in \mathrm{r}^n$. Then Eqn. A.25 can be decomposed along each row:

$$\dot{\boldsymbol{u}}_l = \frac{1}{A}(e^{\boldsymbol{u}_l^2} \cdot \mathbf{1})\boldsymbol{\mu}_l \tag{A.29}$$

Then it is clear that $\boldsymbol{u}_l$ is always along the direction of $\boldsymbol{\mu}_l$, which is a fixed quality since the backpropagated gradient g_j and input $\boldsymbol{x}$ are assumed to be stationary (and thus $\boldsymbol{\Delta}_j := \mathbb{E}_q[g_j \boldsymbol{x}]$ is a constant).

Therefore, let $\boldsymbol{u}_l(t) = \alpha_l(t)\boldsymbol{\mu}_l$ with initial condition of the magnitude $\alpha_l(0) = 0$, and we have:

$$\dot{\alpha}_l = \frac{1}{A} e^{\alpha_l^2 \boldsymbol{\mu}_l^2} \cdot \mathbf{1} = \frac{1}{A} \sum_{j=1}^{n} e^{\alpha_l^2 \mu_{lj}^2} \tag{A.30}$$

where $1 \le l \le M$ is the token index. In the following we will show that for different l, the growth of α_l can be very different. This leads to very different row norms of V and $\dot{V}$ over time, leading to their low-rank structures. Note that Eqn. A.30 does not have a close form solution, instead we could estimate its growth:

$$\frac{1}{A} e^{\alpha_l^2 \bar{\mu}_l^2} \le \dot{\alpha}_l \le \frac{n}{A} e^{\alpha_l^2 \bar{\mu}_l^2} \tag{A.31}$$

where $\bar{\mu}_l^2 := \max_j \mu_{lj}^2$.

Note that both sides have analytic solutions using Gaussian error functions $\mathrm{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} \mathrm{d}t \in [-1, 1]$. *Specifically, for dynamic system like $\dot{x} = Ce^{\beta^2 x^2}$, we have*

$$e^{-\beta^2 x^2} \mathrm{d}x = C \mathrm{d}t \tag{A.32}$$

which gives:

$$\frac{\sqrt{\pi}}{2\beta} \mathrm{erf}\,(\beta x(t)) = \int_0^{x(t)} e^{-\beta^2 y^2} \mathrm{d}y = Ct \tag{A.33}$$

or

$$x(t) = \frac{1}{\beta}\text{erf}^{-1}\left(\frac{2\beta C}{\sqrt{\pi}}t\right) \tag{A.34}$$

For inequality like $\dot{x} \geq Ce^{\beta^2 x^2}$ *or* $\dot{x} \leq Ce^{\beta^2 x^2}$*, similar equation can be derived. Plug that in, we have:*

$$\frac{1}{\bar{\mu}_l}\text{erf}^{-1}\left(\frac{2\bar{\mu}_l}{A\sqrt{\pi}}t\right) \leq \alpha_l(t) \leq \frac{1}{\bar{\mu}_l}\text{erf}^{-1}\left(\frac{2n\bar{\mu}_l}{A\sqrt{\pi}}t\right) \tag{A.35}$$

Let

$$h(t;a) := \frac{1}{a}\text{erf}^{-1}\left(\frac{2}{\sqrt{\pi}}\frac{a}{A}t\right) \tag{A.36}$$

then $\lim_{t\to A\sqrt{\pi}/2a} h(t;a) = +\infty$*, and* $h(t;\bar{\mu}_l) \leq \alpha_l(t) \leq nh(t;n\bar{\mu}_l)$*.*

Let $l^* = \arg\max_l \bar{\mu}_l^*$ *be the row with the largest entry of* μ*, then if* $\bar{\mu}_l^* > n\bar{\mu}_l$ *for all* $l \neq l^*$*, then when* $t \to t^* := \frac{A\sqrt{\pi}}{2\bar{\mu}_l^*}$*, the magnitude* $\alpha_{l^*}(t) \geq h(t;\bar{\mu}_{l^*}) \to +\infty$*, while* $\alpha_l(t) \leq nh(t;n\bar{\mu}_l)$ *still stay finite, since its critical time* $t' := \frac{A\sqrt{\pi}}{2n\bar{\mu}_l} > t^*$*. Since* $\alpha_l(t)$ *controls the magnitude of each row of V, This means that V eventually becomes rank-1 and so does W.*

Finally, $\dot{V}$ *is even more low rank than V, since* $\dot{\alpha}_l$ *has* α_l *in its exponents.*

Convergence of GaLore

Theorem 2.3.7 (Convergence of GaLore with fixed projections) *Suppose the gradient has the form of Eqn. 2.8 and* A_i*,* B_i *and* C_i *have* L_A*,* L_B *and* L_C *continuity with respect to W and* $\|W_t\| \leq D$*. Let* $R_t := P_t^\top G_t Q_t$*,* $\hat{B}_{it} := P_t^\top B_i(W_t)P_t$*,* $\hat{C}_{it} := Q_t^\top C_i(W_t)Q_t$ *and* $\kappa_t := \frac{1}{N}\sum_i \lambda_{\min}(\hat{B}_{it})\lambda_{\min}(\hat{C}_{it})$*. If we choose constant* $P_t = P$ *and* $Q_t = Q$*, then GaLore with* $\rho_t \equiv 1$ *satisfies:*

$$\|R_t\|_F \leq \left[1-\eta(\kappa_{t-1}-L_A-L_BL_CD^2)\right]\|R_{t-1}\|_F. \tag{2.11}$$

As a result, if $\min_t \kappa_t > L_A + L_BL_CD^2$*,* $R_t \to 0$ *and thus GaLore converges with fixed* P_t *and* Q_t*.*

Proof A.2.5.1 *Using* $\text{vec}(AXB) = (B^\top \otimes A)\text{vec}(X)$ *where* $\otimes$ *is the Kronecker product, the gradient assumption can be written as the following:*

$$g_t = a_t - S_t w_t \tag{A.37}$$

where $g_t := \text{vec}(G_t) \in \mathrm{r}^{mn}$*,* $w_t := \text{vec}(W_t) \in \mathrm{r}^{mn}$ *be the vectorized versions of* G_t *and* W_t*,* $a_t := \frac{1}{N}\sum_i \text{vec}(A_{it})$ *and* $S_t = \frac{1}{N}\sum_i C_{it} \otimes B_{it}$ *are mn-by-mn PSD matrix.*

Using the same notation, it is clear to show that:

$$(Q \otimes P)^\top g_t = (Q^\top \otimes P^\top)\mathrm{vec}(G_t) = \mathrm{vec}(P^\top G_t Q) = \mathrm{vec}(R_t) =: r_t \quad \text{(A.38)}$$

$$\tilde{g}_t := \mathrm{vec}(\tilde{G}_t) = \mathrm{vec}(PP^\top G_t Q Q^\top) = (Q \otimes P)\mathrm{vec}(R_t) = (Q \otimes P) r_t \quad \text{(A.39)}$$

Then we derive the recursive update rule for g_t:

$$\begin{aligned} g_t &= a_t - S_t w_t && \text{(A.40)} \\ &= (a_t - a_{t-1}) + (S_{t-1} - S_t) w_t + a_{t-1} - S_{t-1} w_t && \text{(A.41)} \\ &= e_t + a_{t-1} - S_{t-1}(w_{t-1} + \eta \tilde{g}_{t-1}) && \text{(A.42)} \\ &= e_t + g_{t-1} - \eta S_{t-1} \tilde{g}_{t-1} && \text{(A.43)} \end{aligned}$$

where $e_t := (a_t - a_{t-1}) + (S_{t-1} - S_t) w_t$. Left multiplying by $(Q \otimes P)^\top$, we have:

$$r_t = (Q \otimes P)^\top e_t + r_{t-1} - \eta (Q \otimes P)^\top S_{t-1} (Q \otimes P) r_{t-1} \quad \text{(A.44)}$$

Let

$$\hat{S}_t := (Q \otimes P)^\top S_t (Q \otimes P) = \frac{1}{N} \sum_i (Q \otimes P)^\top (C_{it} \otimes B_{it})(Q \otimes P) = \frac{1}{N} \sum_i (Q^\top C_{it} Q) \otimes (P^\top B_{it} P) \quad \text{(A.45)}$$

Then we have:

$$r_t = (I - \eta \hat{S}_{t-1}) r_{t-1} + (Q \otimes P)^\top e_t \quad \text{(A.46)}$$

Now we bound the norm. Note that since P and Q are projection matrices with $P^\top P = I$ and $Q^\top Q = I$, we have:

$$\|(Q \otimes P)^\top e_t\|_2 = \|\mathrm{vec}(P^\top E_t Q)\|_2 = \|P^\top E_t Q\|_F \le \|E_t\|_F \quad \text{(A.47)}$$

where $E_t := \frac{1}{N} \sum_i (A_{it} - A_{i,t-1}) + \frac{1}{N} \sum_i (B_{i,t-1} W_t C_{i,t-1} - B_{it} W_t C_{it})$. So we only need to bound $\|E_t\|_F$. Note that:

$$\begin{aligned} \|A_t - A_{t-1}\|_F &\le L_A \|W_t - W_{t-1}\|_F = \eta L_A \|\tilde{G}_{t-1}\|_F \le \eta L_A \|R_{t-1}\|_F && \text{(A.48)} \\ \|(B_t - B_{t-1}) W_t C_{t-1}\|_F &\le L_B \|W_t - W_{t-1}\|_F \|W_t\|_F \|C_{t-1}\|_F = \eta L_B L_C D^2 \|R_{t-1}\|_F && \text{(A.49)} \\ \|B_t W_t (C_{t-1} - C_t)\|_F &\le L_C \|B_t\|_F \|W_t\|_F \|W_{t-1} - W_t\|_F = \eta L_B L_C D^2 \|R_{t-1}\|_F && \text{(A.50)} \end{aligned}$$

Now we estimate the minimal eigenvalue of $\hat{S}_{t-1}$. Let $\underline{\lambda}_{it} := \lambda_{\min}(P^\top B_{it} P)$ and $\underline{\nu}_{it} := \lambda_{\min}(Q^\top C_{it} Q)$, then $\lambda_{\min}((P^\top B_{it} P) \otimes (Q^\top C_{it} Q)) = \underline{\lambda}_{it} \underline{\nu}_{it}$ and for any unit vector $\boldsymbol{v}$:

$$\boldsymbol{v}^\top \hat{S}_t \boldsymbol{v} = \frac{1}{N} \sum_i \boldsymbol{v}^\top \left[(P^\top B_{it} P) \otimes (Q^\top C_{it} Q) \right] \boldsymbol{v} \ge \frac{1}{N} \sum_i \underline{\lambda}_{it} \underline{\nu}_{it} \quad \text{(A.51)}$$

And thus $\lambda_{\min}(\hat{S}_t) \geq \frac{1}{N}\sum_i \underline{\lambda}_{it}\underline{\nu}_{it}$. *Therefore,* $\lambda_{\max}(I - \eta\hat{S}_{t-1}) \leq 1 - \frac{\eta}{N}\sum_i \underline{\lambda}_{i,t-1}\underline{\nu}_{i,t-1}$. *Therefore, let* $\kappa_t := \frac{1}{N}\sum_i \underline{\lambda}_{it}\underline{\nu}_{it}$ *and using the fact that* $\|r_t\|_2 = \|R_t\|_F$, *we have:*

$$\|R_t\|_F \leq \left[1 - \eta(\kappa_{t-1} - L_A - 2L_B L_C D^2)\right] \|R_{t-1}\|_F \tag{A.52}$$

and the conclusion follows.

A.3 Details of Pre-Training Experiment

Architecture and Hyperparameters

We introduce details of the LLaMA architecture and hyperparameters used for pre-training. Table A.1 shows the most hyperparameters of LLaMA models across model sizes. We use a max sequence length of 256 for all models, with a batch size of 131K tokens. For all experiments, we adopt learning rate warmup for the first 10% of the training steps, and use cosine annealing for the learning rate schedule, decaying to 10% of the initial learning rate.

Table A.1: Hyperparameters of LLaMA models for evaluation. Data amount are specified in tokens.

Params	Hidden	Intermediate	Heads	Layers	Steps	Data amount
60M	512	1376	8	8	10K	1.3 B
130M	768	2048	12	12	20K	2.6 B
350M	1024	2736	16	24	60K	7.8 B
1 B	2048	5461	24	32	100K	13.1 B
7 B	4096	11008	32	32	150K	19.7 B

For all methods on each size of models (from 60M to 1B), we tune their favorite learning rate from a set of $\{0.01, 0.005, 0.001, 0.0005, 0.0001\}$, and the best learning rate is chosen based on the validation perplexity. We find GaLore is insensitive to hyperparameters and tends to be stable with the same learning rate across different model sizes. For all models, GaLore use the same hyperparameters, including the learning rate of 0.01, scale factor α of 0.25, and the subspace change frequency of T of 200. We note that since α can be viewed as a fractional learning rate, most of the modules (e.g., multi-head attention and feed-forward layers) in LLaMA models have the actual learning rate of 0.0025. This is, still, a relatively large stable learning rate compared to the full-rank baseline, which usually uses a learning rate ≤ 0.001 to avoid spikes in the training loss.

Memory Estimates

As the GPU memory usage for a specific component is hard to measure directly, we estimate the memory usage of the weight parameters and optimizer states for each method on different model sizes. The estimation is based on the number of original parameters and the number of low-rank parameters, trained by BF16 format. For example, for a 60M model, LoRA ($r = 128$) requires 42.7M parameters on low-rank adaptors and $60M$ parameters on the original weights, resulting in a memory cost of 0.20G for weight parameters and 0.17G for optimizer states.

Training Progression

We show the training progression of 130M, 350M, 1B and 7B models in Figure A.1. Compared to LoRA, GaLore closely matches the training trajectory of the full-rank baseline, and it even converges slightly faster at the beginning of the training.

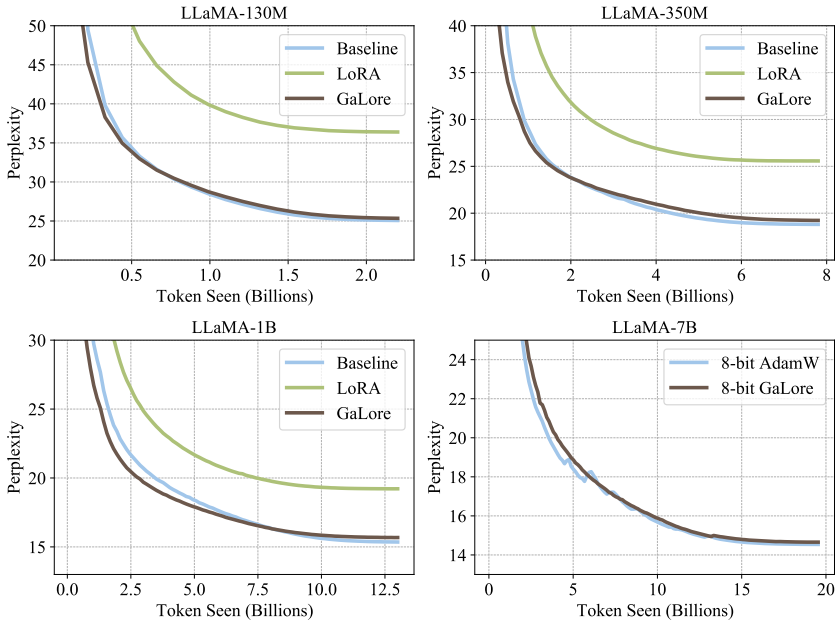

Figure A.1: Training progression for pre-training LLaMA models on C4 dataset.

A.4 Fine-Tuning Experiments

Details of Fine-Tuning on GLUE

We fine-tune the pre-trained RoBERTa-Base model on the GLUE benchmark using the model provided by the Hugging Face[1]. We trained the model for 30 epochs with a batch size of 16 for all tasks except for CoLA, which uses a batch size of

[1] `https://huggingface.co/transformers/model_doc/roberta.html`

32. We tune the learning rate and scale factor for GaLore. Table A.3 shows the hyperparameters used for fine-tuning RoBERTa-Base for GaLore.

Table A.2: Hyperparameters of fine-tuning RoBERTa base for GaLore.

	MNLI	SST-2	MRPC	CoLA	QNLI	QQP	RTE	STS-B
Batch Size	16	16	16	32	16	16	16	16
# Epochs	30	30	30	30	30	30	30	30
Learning Rate	1E-05	1E-05	3E-05	3E-05	1E-05	1E-05	1E-05	1E-05
Rank Config.				$r = 4$				
GaLore α				4				
Max Seq. Len.				512				

	MNLI	SST-2	MRPC	CoLA	QNLI	QQP	RTE	STS-B
Batch Size	16	16	16	32	16	16	16	16
# Epochs	30	30	30	30	30	30	30	30
Learning Rate	1E-05	2E-05	2E-05	1E-05	1E-05	2E-05	2E-05	3E-05
Rank Config.				$r = 8$				
GaLore α				2				
Max Seq. Len.				512				

Table A.3: Hyperparameters of fine-tuning RoBERTa base for GaLore.

	MNLI	SST-2	MRPC	RTE	STS-B
Batch Size	16	16	16	16	16
# Epochs	30	30	30	30	30
Learning Rate	1E-05	1E-05	3E-05	1E-05	1E-05

	MNLI	SST-2	MRPC	RTE	STS-B
Batch Size	16	16	16	16	16
# Epochs	30	30	30	30	30
Learning Rate	1E-05	2E-05	2E-05	2E-05	3E-05

Fine-Tuning on SQuAD dataset

We evaluate GaLore on the SQuAD dataset (Rajpurkar et al., 2016) using the pre-trained BERT-Base model. We use rank 16 for both GaLore and LoRA. GaLore outperforms LoRA in both Exact Match and F1 scores.

Fine-Tuning on OpenAssistant Conversations Dataset

We apply GaLore on fine-tuning experiments on the OpenAssistant Conversations dataset (Köpf et al., 2024), using the pre-trained models, including Gemma-2b,

Table A.4: Evaluating GaLore on SQuAD dataset. Both Exact Match and F1 scores are reported.

	Exact Match	**F1**
Baseline	80.83	88.41
GaLore	**80.52**	**88.29**
LoRA	77.99	86.11

Phi-2, and LLaMA-7B (Touvron et al., 2023; Team et al., 2024). We use rank of 128 for both GaLore and LoRA. The results are shown in Table A.5.

Table A.5: Evaluating GaLore on OpenAssistant Conversations dataset. Testing perplexity is reported.

	Gemma-2b	**Phi-2**	**LLaMA-7B**
Baseline	4.53	3.81	2.98
GaLore	**4.51**	**3.83**	2.95
LoRA	4.56	4.24	**2.94**

Fine-Tuning on Belle-1M Dataset

We also apply GaLore on fine-tuning experiments on the Belle-1M dataset (BELLEGroup, 2023), using the pre-trained models, including Gemma-2b, Phi-2, and LLaMA-7B. We use rank of 128 for both GaLore and LoRA. The results are shown in Table A.6.

Table A.6: Evaluating GaLore on Belle-1M dataset. Testing perplexity is reported.

	Gemma-2b	**Phi-2**	**LLaMA-7B**
Baseline	5.44	2.66	2.27
GaLore	**5.35**	**2.62**	**2.28**
LoRA	5.37	2.75	2.30

References

BELLEGroup (2023). *BELLE: Be Everyone's Large Language model Engine*. `https://github.com/LianjiaTech/BELLE`.

Chowdhery, Aakanksha et al. (2023). "Palm: Scaling language modeling with pathways". In: *Journal of Machine Learning Research*.

Köpf, Andreas et al. (2024). "Openassistant conversations-democratizing large language model alignment". In: *Advances in Neural Information Processing Systems*.

Lv, Kai, Hang Yan, et al. (2023). “AdaLomo: Low-memory Optimization with Adaptive Learning Rate”. In: *ArXiv preprint arXiv:2310.10195*.

Lv, Kai, Yuqing Yang, et al. (2023). “Full Parameter Fine-tuning for Large Language Models with Limited Resources”. In: *ArXiv preprint arXiv:2306.09782*.

Rae, Jack W et al. (2021). “Scaling language models: Methods, analysis & insights from training gopher”. In: *arXiv preprint arXiv:2112.11446*.

Rajpurkar, Pranav et al. (2016). “SQuAD: 100,000+ Questions for Machine Comprehension of Text”. In: *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.

Shazeer, Noam and Mitchell Stern (2018). “Adafactor: Adaptive Learning Rates with Sublinear Memory Cost”. In: *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*. PMLR.

Team, Gemma et al. (2024). “Gemma: Open models based on gemini research and technology”. In: *arXiv preprint arXiv:2403.08295*.

Tian, Yuandong, Yiping Wang, et al. (2024). “JoMA: Demystifying Multilayer Transformers via Joint Dynamics of MLP and Attention”. In: *The Twelfth International Conference on Learning Representations*.

Tian, Yuandong, Lantao Yu, et al. (2020). “Understanding self-supervised learning with dual deep networks”. In: *ArXiv preprint arXiv:2010.00578*.

Touvron, Hugo et al. (2023). “Llama 2: Open foundation and fine-tuned chat models”. In: *arXiv preprint arXiv:2307.09288*.

Wortsman, Mitchell et al. (2023). “Stable and low-precision training for large-scale vision-language models”. In: *Advances in Neural Information Processing Systems*.

Zhai, Xiaohua et al. (2022). “Scaling Vision Transformers”. In: *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE.

Appendix B

APPENDIX - CHAPTER 3

B.1 Proof

In this section, we present the proof of our main analysis, as shown in Theorem 3.4.3.

As introduced in the main text, we analyze the learning trajectory of a 3-layer linear network where $y = W^2W^1x$, $W^1 \in \mathbb{R}^{N_h \times N_x}$ and $W^2 \in \mathbb{R}^{N_y \times N_h}$ are the weight matrices of the first and second layers, respectively, and $N_h < N_x, N_y$.

We assume the inputs are orthogonal, i.e., $x_i^T x_j = 0$ for $i \neq j$. In this case, the continuous gradient flow follows the following differential equations:

$$\frac{\partial}{\partial t}W^1 = W^{2^T}\left(\Sigma^{yx} - W^2W^1\Sigma^{xx}\right), \quad \frac{\partial}{\partial t}W^2 = \left(\Sigma^{yx} - W^2W^1\Sigma^{xx}\right)W^{21^T}. \tag{B.1}$$

Since the inputs are orthogonal $\Sigma^{xx} = I$, the input-output correlation matrix Σ^{yx} contains all information we need to learn the network. We decompose Σ^{yx} using SVD as follows:

$$\Sigma^{yx} = U^{yy}S^{yx}V^{xx^T} = \sum_{\alpha=1}^{N_x} s_\alpha u_\alpha v_\alpha^T. \tag{B.2}$$

Learning the direction and strength of each mode α is crucial to interpolate the input-output correlation matrix Σ^{yx}.

To analyze the evolution of each mode independently, we let a^α be the α^{th} column of $\bar{W}^1$, and let $b^{\alpha T}$ be the α^{th} row of $\bar{W}^2$, where $W^1 = \bar{W}^1V^{xx^T}$, $W^2 = U^{yy}\bar{W}^2$. Based on Equation B.1 we can characterize the evolution of each mode using a^α and b^α:

$$\frac{\partial}{\partial t}a^\alpha = (s_\alpha - a^\alpha \cdot b^\alpha)\, b^\alpha - \sum_{\gamma \neq \alpha} b^\gamma\,(a^\alpha \cdot b^\gamma), \quad \frac{\partial}{\partial t}b^\alpha = (s_\alpha - a^\alpha \cdot b^\alpha)\, a^\alpha - \sum_{\gamma \neq \alpha} a^\gamma\,(b^\alpha \cdot a^\gamma). \tag{B.3}$$

For both $\frac{\partial}{\partial t}a^\alpha$ and $\frac{\partial}{\partial t}b^\alpha$, the first term characterizes the cooperative learning of the strength s_α using the a^α and b^α. The second term characterizes the competitive learning of the direction a^α and b^α given the distraction from other directions a^γ and b^γ.

It is difficult to solve Equation B.3 given arbitrary weight initialization due to complex competitive interaction between modes. Therefore, we assume the weight initialization follows $W_0^2 = U^{yy}M^2O^T$, $W_0^1 = OM^1V^{xx^T}$, where M^2, M^1 are diagonal

matrices, and O is an arbitrary orthogonal matrix. $W_0^2 W_0^1$ ensures that each mode α is learned independently right from the beginning of training, enabling us to analyze the learning trajectory of each mode separately. a^α and b^α will remain parallel to a certain direction r^α throughout the learning process, and we can rewrite Equation B.3 as follows:

$$\frac{\partial}{\partial t} a = b(s - ab), \quad \frac{\partial}{\partial t} b = a(s - ab), \tag{B.4}$$

where we let $a = a^\alpha \cdot r^\alpha$, $b = b^\alpha \cdot r^\alpha$, and $s = s^\alpha$. By further assuming $a = b$ and $u = ab$, we obtain:

$$\frac{\partial}{\partial t} u = 2u(s - u). \tag{B.5}$$

Integrate the above equation to obtain:

$$t = \tau \int_{u_0}^{u_f} \frac{du}{2u(s-u)} = \frac{\tau}{2s} \ln \frac{u_f\,(s - u_0)}{u_0\,(s - u_f)}, \tag{B.6}$$

where u_0 is the initial value determined by M^2 and M^1, u_f is the target value of strength, and τ is a constant. t is the time it takes for u to travel from u_0 to u_f.

As we analyze the difference of the product matrix $D_t = A_t - A_0$, it is equivalent to analyzing the residual of each mode: $u_t - u_0$. To analyze the entire evolution ($u_f \approx s$) of u over time, we yield the following equation:

$$u_f(t) = \frac{s e^{2st/\tau}}{e^{2st/\tau} - 1 + s/u_0} - u_0. \tag{B.7}$$

B.2 Clarification on Greedy Low-Rank Learning

In this section, we additional clarification on the greedy low-rank learning hypothesis, which is presented in Theorem 3.3.1.

Several works have demonstrated the greedy low-rank learning behavior under various settings and assumptions. Li, Luo, and Lyu (2021) prove it under matrix factorization setting for deep linear network by analyzing the asymptotic behavior of gradient flow under infinitesimal initialization. Jacot et al. (2022) also demonstrate the saddle-to-saddle learning behavior for deep linear networks, although they prove the rank-one case only. Razin, Maman, and Cohen (2021) further extend the discussion to the setting of tensor factorization.

A formal description of Theorem 3.3.1 is given below:

Theorem B.2.1 *Let $\tilde{W}_r$ be the r-th critical point of a rank-r subspace of W, and let $\tilde{W}_0 = 0$ be the saddle point at zero. From an infinitesimal initialization ($W_0 \approx \tilde{W}_0$),*

the gradient flow $G(W)$ first visits the critical point $\tilde{W}_1$. If $\tilde{W}_1$ is not a minimizer, $G(W)$ will expand the searching space to a rank-2 subspace and converge to the critical point $\tilde{W}_2$. If $\tilde{W}_2$ is also not a minimizer, this process continues until $G(W)$ reaches $\tilde{W}_{r^*}$ in a rank-r^* subspace that minimizes the objective function, provided that $r^* < \text{rank}(W)$.

The theorem implies the greedy low-rank learning trajectory, such that the gradient descent first searches over a rank-1 subspace of A_θ, and then greedily increases the rank by one whenever it fails to reach the minimizer.

Proving this requires the analysis of the limiting flow $G_{r\to r+1}(W)$, which is the gradient flow between two critical points $\tilde{W}_r$ and $\tilde{W}_{r+1}$. Theorem B.2.1 holds by showing that the flows $G_{0\to 1}(W)$, $G_{1\to 2}(W)$, ..., $G_{r^*-1\to r^*}(W)$ all exist during learning, which is a general proving direction adopted by recent works. The details of the proof can be found in Li, Luo, and Lyu (2021) and Jacot et al. (2022).

B.3 Low-Rank Learning in Practice

In this section, we provide additional results demonstrating that the cumulative weight updates follow the low-rank learning trajectory over a broad range of network architectures and learning algorithms.

Low-Rank learning under different architectures

LSTM

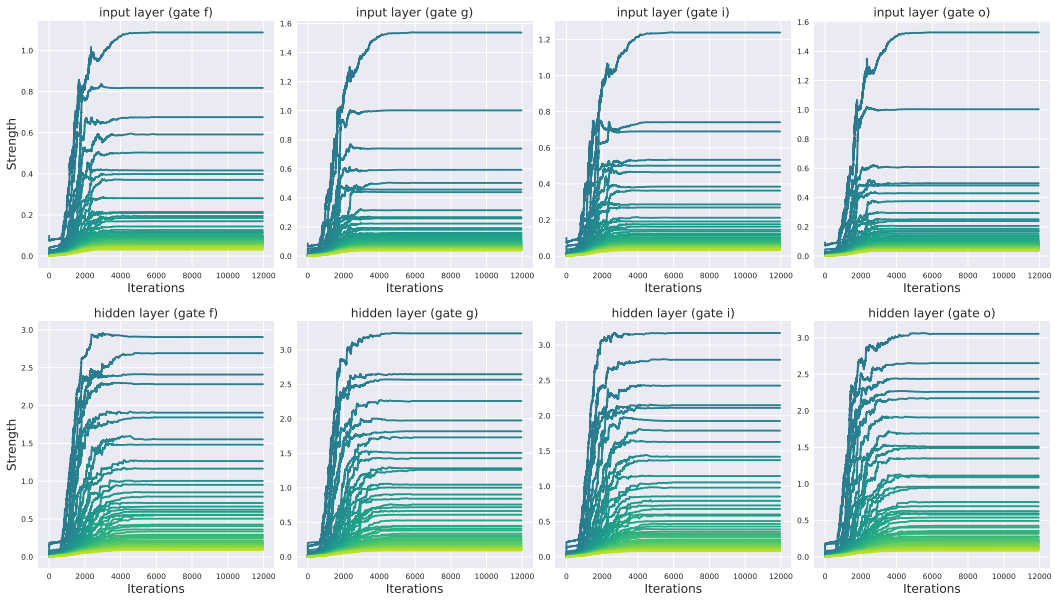

Figure B.1: The evolutions of all singular vectors of cumulative weight updates D_t over the training of LSTM. The top row shows the input-to-hidden weight matrix W_{ih}, and the bottom row shows the hidden-to-hidden weight matrix W_{hh}. Darker colors indicate singular vectors with higher strengths.

Transformer

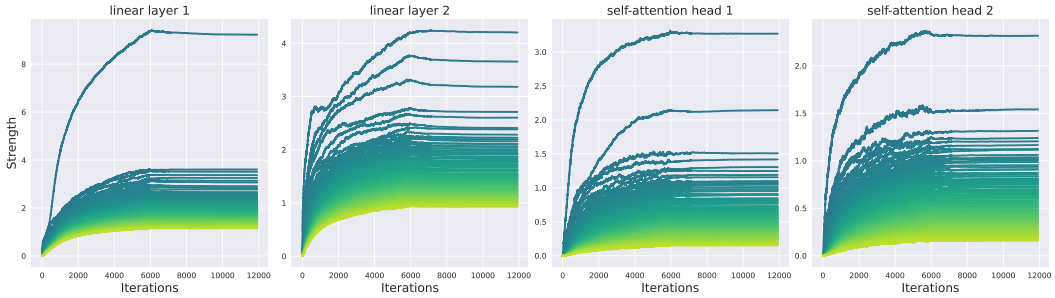

Figure B.2: The evolutions of all singular vectors of cumulative weight updates D_t over the training of Transformer. In a single layer, we visualize two weight matrices in MLP and two K matrices in self-attention. Darker colors indicate singular vectors with higher strengths.

Low-Rank learning under different learning algorithms

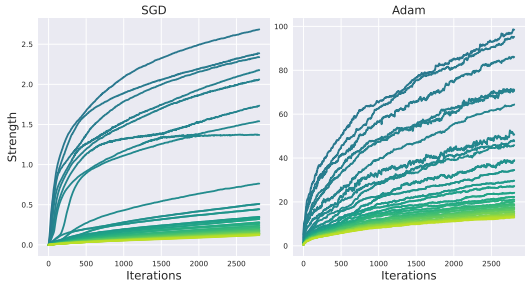

Figure B.3: The evolutions of all singular vectors of cumulative weight updates D_t over the training of MLP using SGD and Adam optimizers. Darker colors indicate singular vectors with higher strengths.

B.4 Rank Evolution during Training

We present the rank evolution in various MLP layers when applying InRank on GPT-small model. As shown in Figure B.4, we visualize the rank evolution over the first 5% of the total training iterations. The figure indicates that the increment of rank mostly happens in the early stage of training.

B.5 Repeated Experiment on different GPT models

We report the evaluation results of InRank and InRank-Efficient on different sizes of GPT models in Table B.1. All experiments are repeated 3 times. We also report testing perplexity instead of validation perplexity.

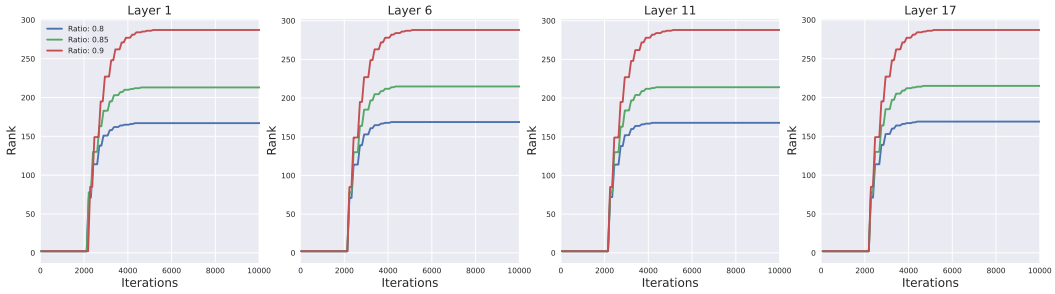

Figure B.4: The rank evolution in various MLP layers when applying InRank on GPT-small model.

Table B.1: Evaluating InRank across different sizes of GPT models. All experiments are repeated 3 times.

Model	**Method**	**Test Perplexity**
GPT-small	InRank	19.96 ± 0.10
	InRank-Efficient	20.07 ± 0.14
GPT-medium	InRank	22.14 ± 0.07
	InRank-Efficient	21.23 ± 0.05
GPT-large	InRank	22.63 ± 0.09
	InRank-Efficient	21.49 ± 0.07

References

Jacot, Arthur et al. (Jan. 2022). "Saddle-to-Saddle Dynamics in Deep Linear Networks: Small Initialization Training, Symmetry, and Sparsity". en. In: *arXiv:2106.15933 [cs, stat]*. arXiv: 2106.15933. URL: `http://arxiv.org/abs/2106.15933` (visited on 02/21/2022).

Li, Zhiyuan, Yuping Luo, and Kaifeng Lyu (Apr. 2021). "Towards Resolving the Implicit Bias of Gradient Descent for Matrix Factorization: Greedy Low-Rank Learning". In: *arXiv:2012.09839 [cs, stat]*. arXiv: 2012.09839. URL: `http://arxiv.org/abs/2012.09839` (visited on 02/25/2022).

Razin, Noam, Asaf Maman, and Nadav Cohen (June 2021). "Implicit Regularization in Tensor Factorization". In: *arXiv:2102.09972 [cs, stat]*. arXiv: 2102.09972. URL: `http://arxiv.org/abs/2102.09972` (visited on 04/12/2022).

Appendix C

APPENDIX - CHAPTER 5

C.1 Quantization Error Analysis

Here we present proofs of theorems and lemmas presented in the main paper, as well as some additional details of the empirical evaluations.

Proofs

Before presenting the proofs, we want to clarify the error definition and assumptions introduced in the main paper. Previously, we claim that minimizing $r_t = \|\log_2 |W_{t+1}^U| - \log_2 |W_{t+1}|\|^2$ is equivalent to minimizing relative quantization error $\|(W_{t+1} - W_{t+1}^U)/W_{t+1}\|^2$. To better understand it, we transform the form of relative quantization error as follows:

$$
\begin{aligned}
&\|(W_{t+1} - W_{t+1}^U)/W_{t+1}\|^2 \\
&= \|(I - W_{t+1}^U)/W_{t+1}\|^2 \\
&= \|(I - |W_{t+1}^U)|/|W_{t+1}|\|^2 \\
&(sign(W_{t+1}^U) = sign(W_{t+1})) \\
&= \|(I - 2^{\log_2 |W_{t+1}^U| - \log_2 |W_{t+1}|}\|^2 \\
&\text{(transfer to base-2 logarithmic space)}
\end{aligned}
$$

This relaxation suggests that minimizing r_t is equivalent to minimizing the relative quantization error.

We start to introduce the simplified logarithmic quantization we used for the analysis. The stochastic rounding (SR) is defined as follows:

$$
\mathrm{SR}(x) = \begin{cases} \lfloor x \rfloor + 1 & \text{for } p \leq x - \lfloor x \rfloor, \\ \lfloor x \rfloor & \text{otherwise,} \end{cases} \tag{C.1}
$$

where $p \in [0, 1]$ is generated by a uniform random number generator. SR makes sure the rounded number is an unbiased estimate of its full-precision counterpart: $\mathbb{E}\,\mathrm{SR}(x) = x$, which is an important property for the analysis.

Equipped with SR, we define the simplified logarithmic quantization function:

$$
\mathrm{LogQuant}(x) = sign(x) \times 2^{\tilde{x}/\gamma}, \tag{C.2}
$$

where $\tilde{x} = \mathrm{SR}(\log_2 |x| \times \gamma)$. We ignore the scale factor and the clamping function to ensure our focus is on the effect of the quantization gap instead of the dynamic range.

Before proving our main results, we want to introduce an important proposition that describes the error introduced by stochastic rounding.

Proposition C.1.1 *For any vector x, the quantization error introduced by stochastic rounding $r = \mathrm{SR}(x) - x$ can be bounded in expectation, as:*

$$\mathbb{E}\,\|r\|^2 \leq \sqrt{d}\,\|x\|, \tag{C.3}$$

where d is the dimension of x.

Proof C.1.1.1 *Let r_i denotes the ith element of r and let $q_i = x_i - \lfloor x_i \rfloor$. r_i can be represented as follows:*

$$\begin{aligned} r_i &= \begin{cases} \lfloor x_i \rfloor + 1 - x_i & \text{for } p \leq x_i - \lfloor x_i \rfloor, \\ \lfloor x_i \rfloor - x_i & \text{otherwise,} \end{cases} \\ &= \begin{cases} -q_i + 1 & \text{for } p \leq q_i, \\ -q_i & \text{otherwise.} \end{cases} \end{aligned}$$

r_i can be bounded by expectation, as:

$$\begin{aligned} \mathbb{E}\, r_i^2 &\leq (-q_i + 1)^2 q_i + (-q_i)^2 (1 - q_i) \\ &= q_i(1 - q_i) \\ &\leq \min\{q_i, 1 - q_i\} \\ &= \min\{x_i - \lfloor x_i \rfloor, 1 - x_i + \lfloor x_i \rfloor\} \\ &\leq |x_i|. \end{aligned}$$

Therefore, by summing over index i, we can get:

$$\begin{aligned} \mathbb{E}\,\|r\|^2 &\leq \|x\|_1 \\ &\leq \sqrt{d}\,\|x\|. \end{aligned}$$

Now we start to prove Theorem 5.5.1 given $U_{GD} = W - \eta\,\nabla_W$.

Theorem 5.5.1 *The quantization error $r_{t,GD}$ introduced by logarithmic quantized gradient descent at iteration t can be bounded in expectation, as:*

$$\mathbb{E}\, r_{t,GD} \leq \frac{\sqrt{d}}{\gamma}\,\|\log_2\left(|W_t| - \eta_1 \nabla_{W_t}\right)\|, \tag{5.5}$$

where d is the dimension of W and η_1 is the learning rate of U_{GD}.

Proof C.1.1.1 *We know that:*

$$\mathbb{E}\, r_{t,GD} = \| \log_2 |\text{LogQuant}(W_t - \eta_1 \nabla_{W_t})| - \log_2 |W_t - \eta_1 \nabla_{W_t}|\|^2.$$

By replacing LogQuant *with Equation C.2, we can get:*

$$\log_2 |\text{LogQuant}(W_t - \eta_1 \nabla_{W_t})| = \frac{1}{\gamma} \text{SR}(\gamma \log_2 |W_t - \eta_1 \nabla_{W_t}|).$$

Plug it back to $\mathbb{E}\, r_{t,GD}$*, we get:*

$$\mathbb{E}\, r_{t,GD} = \frac{1}{\gamma^2} \| \text{SR}(\gamma \log_2 |W_t - \eta_1 \nabla_{W_t}|) - \gamma \log_2 |W_t - \eta_1 \nabla_{W_t}|\|^2.$$

Given Proposition C.1.1, we can upper bound the quantization error introduced by stochastic rounding:

$$\begin{aligned} &\|\text{SR}(\gamma \log_2 |W_t - \eta_1 \nabla_{W_t}|) - \gamma \log_2 |W_t - \eta_1 \nabla_{W_t}|\|^2 \\ &\leq \sqrt{d}\, \|\gamma \log_2 |W_t - \eta_1 \nabla_{W_t}|\|. \end{aligned}$$

Therefore, we can get:

$$\begin{aligned} \mathbb{E}\, r_{t,GD} &\leq \frac{\sqrt{d}}{\gamma^2} \|\gamma \log_2 |W_t - \eta_1 \nabla_{W_t}|\| \\ &\leq \frac{\sqrt{d}}{\gamma} \|\log_2 |W_t - \eta_1 \nabla_{W_t}|\|. \end{aligned}$$

Given $U_{MUL} = sign(W) \odot 2^{\tilde{W} - \eta \nabla_W \odot sign(W)}$, Theorem 5.5.2 follows a similar proof as Theorem 5.5.1.

Theorem 5.5.2 *The quantization error* $r_{t,MUL}$ *introduced by logarithmic quantized multiplicative weight update at iteration t can be bounded in expectation, as:*

$$\mathbb{E}\, r_{t,MUL} \leq \frac{\sqrt{d}\, \eta_2}{\gamma} \|\nabla_{W_t}\|, \tag{5.7}$$

where d is the dimension of W and η_2 *is the learning rate of* U_{MUL}*.*

Proof C.1.1.1

$$\mathbb{E}\, r_{t,MUL} = \| \log_2 |\text{LogQuant}(2^{\tilde{W}_t - \eta_2 \nabla_{W_t} \odot sign(W_t)})| - \log_2 |2^{\tilde{W}_t - \eta_2 \nabla_{W_t} \odot sign(W_t)}|\|^2. \quad \text{(C.4)}$$

By replacing LogQuant *with Equation C.2, we can get:*

$$\log_2 |\text{LogQuant}(2^{\tilde{W}_t - \eta_2 \nabla_{W_t} \odot sign(W_t)})| = \frac{1}{\gamma} \text{SR}(\gamma\, (\tilde{W}_t - \eta_2 \nabla_{W_t} \odot sign(W_t))).$$

Plug it back to Equation C.4:

$$\mathbb{E}\, r_{t,MUL} = \frac{1}{\gamma^2} \| \text{SR}(\gamma\, (\tilde{W}_t - \eta_2 \nabla_{W_t} \odot sign(W_t))) - \gamma\, (\tilde{W}_t - \eta_2 \nabla_{W_t} \odot sign(W_t))\|^2.$$

Because $\tilde{W}_t$ is already an integer, $\text{SR}(\gamma \tilde{W}_t) - \gamma \tilde{W}_t = 0$, and thus we can eliminate $\tilde{W}_t$ in the equation:

$$\mathbb{E}\, r_{t,MUL} = \frac{1}{\gamma^2} \| \text{SR}(-\gamma\, \eta_2 \nabla_{W_t} \odot sign(W_t)) + \gamma\, \eta_2 \nabla_{W_t} \odot sign(W_t)\|^2.$$

Similar to the proof of Theorem 5.5.1, we can upper bound it using Proposition C.1.1, and get:

$$\begin{aligned} \mathbb{E}\, r_{t,MUL} &\leq \frac{\sqrt{d}}{\gamma^2} \|\gamma\, \eta_2 \nabla_{W_t} \odot sign(W_t)\| \\ &\leq \frac{\sqrt{d}\, \eta_2}{\gamma} \|\nabla_{W_t}\|. \end{aligned}$$

Lemma C.1.2 *Assume the multiplicative learning algorithm U_{MUL} only receives the sign information of gradients where $U_{MUL} = \tilde{W}_t - \eta_2\, sign(\nabla_{W_t}) \odot sign(W_t)$. The upper bound on quantization error $r_{t,MUL}$ becomes:*

$$\mathbb{E}\, r_{t,MUL} \leq \frac{d\, \eta_2}{\gamma}. \quad (5.8)$$

Proof C.1.2.1 *We can simply replace ∇_{W_t} with $sign(\nabla_{W_t})$ in the result of Theorem 5.5.2, and show:*

$$\frac{\sqrt{d}\, \eta_2}{\gamma} \|sign(\nabla_{W_t})\| \leq \frac{d\, \eta_2}{\gamma}.$$

Evaluations

As shown in Figure 5.3, we evaluate empirical quantization errors from different learning algorithms when training ResNet-50 on ImageNet. The quantization error is computed at each iteration by $\|\log_2 |W^U_{t+1}| - \log_2 |W_{t+1}|\|^2$. We run each experiment with a full epoch and average the quantization error over iterations. When varying learning rate η, we fix the base factor γ as 2^{10}. We also fix η as 2^{-6} when varying γ.

C.2 Multi-Base LNS

Conversion Approximation

We first recap the dot product operation we defined before.

$$\begin{aligned} \boldsymbol{a}^T \boldsymbol{b} &= \sum_{i=1}^{n} \text{sign}_i \times 2^{\tilde{\boldsymbol{a}}_i/\gamma} \times 2^{\tilde{\boldsymbol{b}}_i/\gamma}, \\ &= \sum_{i=1}^{n} \text{sign}_i \times 2^{(\tilde{\boldsymbol{a}}_i+\tilde{\boldsymbol{b}}_i)/\gamma}, \\ &= \sum_{i=1}^{n} \text{sign}_i \times 2^{\tilde{p}_i/\gamma}, \end{aligned} \tag{C.5}$$

where $\text{sign}_i = sign(\boldsymbol{a}_i) \oplus sign(\boldsymbol{b}_i)$.

To understand how we approximate the conversion, we first introduce how ordinary conversion is computed in LNS. Let $\tilde{p_{iq}}$ and $\tilde{p_{ir}}$ be positive integers representing quotient and remainder of the intermediate result $\tilde{p}_i/\gamma$ in Equation C.5, and let $v_r = 2^{\tilde{p_{ir}}/\gamma}$. Therefore,

$$\begin{aligned} 2^{\tilde{p}_i/\gamma} = 2^{\tilde{p}_i/\gamma} &= 2^{\tilde{p_{iq}}+\tilde{p_{ir}}/\gamma} = 2^{\tilde{p_{iq}}} \times 2^{\tilde{p_{ir}}/\gamma} \\ &= (v_r << \tilde{p_{iq}}), \end{aligned} \tag{C.6}$$

where $<<$ is left bit-shifting. This transformation enables fast conversion by applying efficient bit-shifting over v_r whose value is bounded by the remainder. The different constant values of $v_r = 2^{\tilde{p_{ir}}/\gamma}$ can be pre-computed and stored in a hardware look-up table (LUT), where the remainder $\tilde{p_{ir}}$ is used to select the constant for v_r. The quotient $\tilde{p_{iq}}$ then determines how far to shift the constant. Furthermore, because $\gamma = 2^b$, the least significant bits (LSB) of the exponent are the remainder and the most significant bits (MSB) are the quotient. As the size of the LUT grows, the computational overhead from conversion increases significantly. Typically, the LUT is required to contain 2^b entries for storing all possible values of v_r, which can be a large overhead for large values of b.

A straightforward solution for reducing the size of LUT is utilizing Mitchell approximation Mitchell, 1962: $v_r = 2^{\tilde{p_{ir}}/2^b} = (1 + \tilde{p_{ir}}/2^b)$. However, if v_r is far away from zero or one, the approximation error induced by Mitchell approximation will be significant. To alleviate this error, we propose a hybrid approximation that trades off efficiency and approximation error. Specifically, we split p_{ir} into p_{irM} and p_{irL} to represent the MSB and LSB of the remainder, respectively. LSB values $2^{\tilde{p_{irL}}}$ are approximated using Mitchell approximation, and MSB values $2^{\tilde{p_{irM}}}$ are pre-computed and stored using LUT, such that:

$$\begin{aligned} v_r = 2^{\tilde{p_{ir}}/2^b} &= 2^{\tilde{p_{irM}}/2^b} \times 2^{\tilde{p_{irL}}/2^b} \\ &= (1 + \tilde{p_{irL}}/2^b) \times 2^{\tilde{p_{irM}}/2^b}, \end{aligned} \tag{C.7}$$

where p_{irM} and p_{irL} represent b_m MSB and b_l LSB bits of p_{ir}. This reduces the size of LUT to 2^{b_m} entries. For efficient hardware implementation, we use 2^{b_m} registers to accumulate different partial sum values and then multiply with constants from the LUT.

References

Mitchell, J. N. (1962). "Computer Multiplication and Division Using Binary Logarithms". In: *IRE Transactions on Electronic Computers* EC-11.4, pp. 512–517. DOI: 10.1109/TEC.1962.5219391.

Appendix D

APPENDIX - CHAPTER 4

D.1 Additional Proofs in Theorem 4.3.3

Proof of 1

We want to show that for any matrix $\boldsymbol{W} \in \mathbb{R}^{M\times N}$, if each entry is sampled from a Gaussian distribution $\mathcal{N}(\mu, \sigma^2)$, then $\boldsymbol{W}$ will achieve full-rank (i.e., $\text{rank}(\boldsymbol{W}) = \min(M, N)$) with probability of one.

Assume $M \geq N$ without losing generality. We want to prove the following statement:

$$Prob(\text{rank}(\boldsymbol{W}) < N) = 0.$$

Let $\{\boldsymbol{v}_i\}$ be columns of $\boldsymbol{W}$, where $\boldsymbol{v}_i \in \mathbb{R}^M$ for $i \in \{1, ..., N\}$. We denote P_i as the probability of the event where a column $\boldsymbol{v}_i$ is linearly dependent on the rest of the columns $\{\boldsymbol{v}_j\}$, for $j \neq i$. If $\boldsymbol{W}$ does not reach the full-rank, then there exsists at least a column that is linearly dependent on the rest of the columns. Thus, by the union bound, we know that:

$$Prob(\text{rank}(\boldsymbol{W}) < N) \leq \sum_{i=1}^{N} P_i.$$

For each P_i, it is equivalent to the probability of the event where $\boldsymbol{v}_i \in \mathbb{S} = \text{span}(\boldsymbol{v}_1, ..., \boldsymbol{v}_j)$, for $j \neq i$. Since $\mathbb{S}$ at most has $N - 1$ dimension, it is a subspace of $\mathbb{R}^M$. Because each entry of $\boldsymbol{v}_i \in \mathbb{R}^M$ is sampled from a continuous Gaussian distribution, the probability that $\boldsymbol{v}_i$ fails into a low-dimensional subspace $\mathbb{S}$ is zero. Thus, $P_i = 0$ for any $i \in \{1, ..., N\}$, and $Prob(\text{rank}(\boldsymbol{W}) < N) = 0$. We can conclude that $\boldsymbol{W}$ reaches the full-rank almost surely.

Additional Proofs of 2

Here we prove that the rank constraint in 2 is persistent through the entire training. Assume we perform gradient descent during training. The initial derivatives become:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}_1'} = \begin{pmatrix} \Lambda \\ \boldsymbol{0} \end{pmatrix} \in \mathbb{R}^{n_h \times n_x}, \quad \frac{\partial \mathcal{L}}{\partial \boldsymbol{W}_l'} = \begin{pmatrix} \Lambda & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} \end{pmatrix} \in \mathbb{R}^{n_h \times n_h}, \quad \frac{\partial \mathcal{L}}{\partial \boldsymbol{W}_L'} = \begin{pmatrix} \Lambda & \boldsymbol{0} \end{pmatrix} \in \mathbb{R}^{n_y \times n_h},$$

where $\Lambda = \sum_{\mu=1}^{P} \text{Relu}'(z^\mu) \odot (z^\mu - y^\mu)x^{\mu^T} \in \mathbb{R}^{n_y \times n_x}$, and $z^\mu = \boldsymbol{I}_L \circ \text{Relu} \circ \boldsymbol{I}_1 x^\mu$ and $l \in 2, ..., L-1$. $\mathbf{0}$ is a zero vector or a zero matrix depending on the dimensionality. $\odot$ represents an element-wise multiplication.

As shown above, in the initial derivative of each matrix, the elements are zero except for a sub-matrix determined by Λ. Because the parameters with zero initial derivatives stall at zero during the entire training, the rank of each derivative in $\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}_2'}, ..., \frac{\partial \mathcal{L}}{\partial \boldsymbol{W}_{L-1}'}$ is upper bounded by n_x. Thus, the rank constraint in 2 is satisfied during the entire training.

Proof of Lemma 4.3.4

We know that:

$$\boldsymbol{M} = \sum_{\mu=1}^{Q} \boldsymbol{a}^\mu \otimes \boldsymbol{b}^\mu = \boldsymbol{A}\boldsymbol{B}^\top,$$

where we define $\boldsymbol{A} = (\boldsymbol{a}^1, ..., \boldsymbol{a}^Q)$ and $\boldsymbol{B} = (\boldsymbol{b}^1, ..., \boldsymbol{b}^Q)$. Since $\dim(\text{span}(\boldsymbol{a}^1, ..., \boldsymbol{a}^Q)) \leq U$, there are at most U independent columns in $\boldsymbol{A}$., and thus $\text{rank}(\boldsymbol{A}) \leq U$. Similarly, we can prove that $\text{rank}(\boldsymbol{B}) \leq V$. By the linear algebra, we can show that:

$$\text{rank}(\boldsymbol{M}) = \text{rank}(\boldsymbol{A}\boldsymbol{B}^\top) \leq \min(\text{rank}(\boldsymbol{A}), \text{rank}(\boldsymbol{B}^\top)) = \min(U, V)$$

.

Proof of Lemma 4.3.5

Let $\boldsymbol{x} = \alpha \cdot \boldsymbol{e}_2 + \beta \cdot \boldsymbol{e}_3$ where α and β are scalars. Then we have:

$$\text{Relu}\, \boldsymbol{H}\boldsymbol{I}^*\boldsymbol{x} = \begin{cases} \alpha\, \text{Relu}(\boldsymbol{H}\boldsymbol{I}^*\boldsymbol{e}_2) + \beta\, \text{Relu}(\boldsymbol{H}\boldsymbol{I}^*\boldsymbol{e}_3) & \text{for } \alpha > 0 \text{ and } \beta > 0, \\ \alpha\, \text{Relu}(-\boldsymbol{H}\boldsymbol{I}^*\boldsymbol{e}_2) + \beta\, \text{Relu}(\boldsymbol{H}\boldsymbol{I}^*\boldsymbol{e}_3) & \text{for } \alpha < 0 \text{ and } \beta > 0, \\ \alpha\, \text{Relu}(\boldsymbol{H}\boldsymbol{I}^*\boldsymbol{e}_2) + \beta\, \text{Relu}(-\boldsymbol{H}\boldsymbol{I}^*\boldsymbol{e}_3) & \text{for } \alpha > 0 \text{ and } \beta < 0, \\ \alpha\, \text{Relu}(-\boldsymbol{H}\boldsymbol{I}^*\boldsymbol{e}_2) + \beta\, \text{Relu}(-\boldsymbol{H}\boldsymbol{I}^*\boldsymbol{e}_3) & \text{for } \alpha < 0 \text{ and } \beta < 0. \end{cases}$$

Thus, it holds that:

$$\text{span}\,(\{\text{Relu}\, \boldsymbol{H}\boldsymbol{I}^*\boldsymbol{x} | \boldsymbol{x} \in \mathbb{R}^{n_x}\}) = \text{span}(\text{Relu}(\boldsymbol{H}\boldsymbol{I}^*\boldsymbol{e}_2), \text{Relu}(-\boldsymbol{H}\boldsymbol{I}^*\boldsymbol{e}_2), \text{Relu}(\boldsymbol{H}\boldsymbol{I}^*\boldsymbol{e}_3), \text{Relu}(-\boldsymbol{H}\boldsymbol{I}^*\boldsymbol{e}_3)).$$

After a simple computation, we also know that:

$$\boldsymbol{HI}^{*}\boldsymbol{e}_2 = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \end{pmatrix} \qquad -\boldsymbol{HI}^{*}\boldsymbol{e}_2 = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \end{pmatrix}$$

$$\boldsymbol{HI}^{*}\boldsymbol{e}_3 = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \end{pmatrix} \qquad -\boldsymbol{HI}^{*}\boldsymbol{e}_3 = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \end{pmatrix}.$$

Therefore, the four vectors are linearly independent, and thus:

$$\dim\left(\operatorname{span}\left(\{\operatorname{Relu}\boldsymbol{HI}^{*}\boldsymbol{x} | \boldsymbol{x} \in \mathbb{R}^{n_x}\}\right)\right) = 4.$$

D.2 Details of the experiments on MNIST

We train a network $\mathcal{F}$ defined in Theorem 4.3.3 with depth $L = 3$ on MNIST, where $\boldsymbol{W}^1 \in \mathbb{R}^{2048\times 784}$, $\boldsymbol{W}^2 \in \mathbb{R}^{2048\times 2048}$, and $\boldsymbol{W}^3 \in \mathbb{R}^{10\times 2048}$. We initialize $\boldsymbol{W}^1 = \boldsymbol{H}\boldsymbol{I}^*$, $\boldsymbol{W}^2 = \boldsymbol{I}$, and $\boldsymbol{W}^3 = \boldsymbol{I}^*$. The network is trained for 14 epochs using SGD with a learning rate of 0.1. We compare ZerO with Kaiming and Xavier initialization under the same setting. All candidates achieve test accuracy above 98% after the training. Figure D.1 shows the weight distributions at different training iterations.

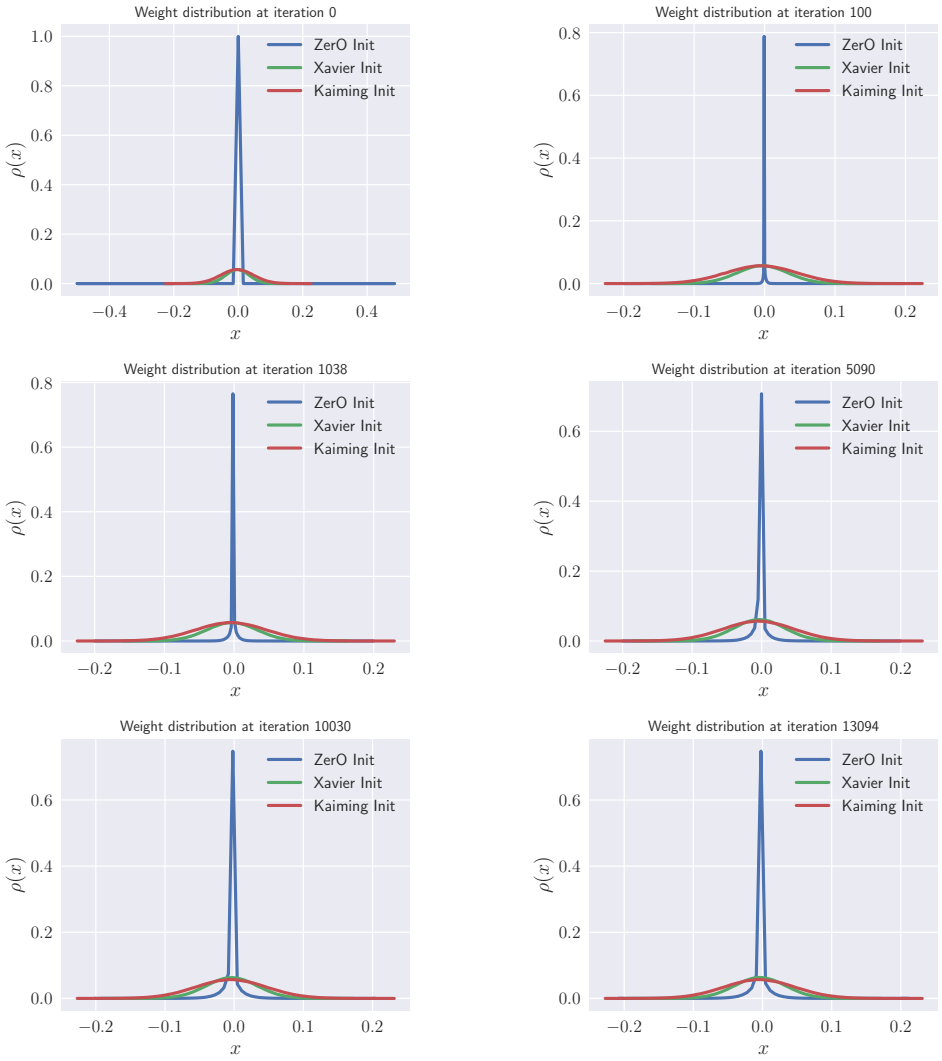

Figure D.1: Weight distributions at different training iterations.

Appendix E

CONSENT FORM