

Layerwise Training of Deep Neural Networks



*Elicia Ye
Tianyu Pang
Alex Zhao
Yefan Zhou
Yaoqing Yang
Michael Mahoney
Kannan Ramchandran*

Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2023-126

<http://www2.eecs.berkeley.edu/Pubs/TechRpts/2023/EECS-2023-126.html>

May 12, 2023

Copyright © 2023, by the author(s).
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

Acknowledgement

I would like to express my deep gratitude to my amazing advisor, Prof. Kannan Ramchandran, for his unwavering support and the exceptional opportunity to work in his research group. I was fortunate to work with wonderful mentors and collaborators Yaoqing Yang, Alex Zhao, Yefan Zhou, Tianyu Pang, and Prof. Michael Mahoney on projects in this thesis. I had the privilege to learn from my brilliant and dedicated labmates Landon Butler, Efe Aras, Syomantak Chaudhuri, Yigit Efe Erginbas, Justin Kang, Nived Rajaraman, Jichan Chung, and Ajil Jalal. Lastly, I dedicate this thesis to my loving friends and family, and to UC Berkeley.

Layerwise Training of Deep Neural Networks

by Elicia Ye

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences,
University of California at Berkeley, in partial satisfaction of the requirements for the
degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

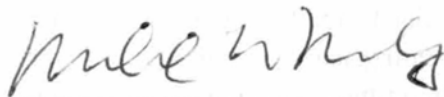
Committee:



Professor Kannan Ramchandran
Research Advisor

May 9th, 2023

(Date)



Professor Michael W. Mahoney
Second Reader

5/11/23

(Date)

Abstract

Layerwise Training of Deep Neural Networks

by

Yi Chen Ye

Master of Science in Electrical Engineering and Computer Science

University of California, Berkeley

The answer to many questions in the artificial intelligence realm has been a data-dependent solution, supplemented by deep neural networks (DNNs) as data processors. With an optimal combination of settings, dataset, and architecture, deep networks were introduced to mimic the human brain through artificial neurons, performing a variety of tasks in computer vision (CV) and natural language processing (NLP). They serve as tools in data analytics applications including self-driving, language translation services, medical diagnosis, stock market trading signals and more. It is natural to assume that a network’s representational power must scale in complexity with the tasks or dataset it processes. In practice, however, increasing the amount of data or number of layers and parameters is not always the answer. In certain resource-constrained settings, training deep networks for an extended period of time is not only intractable but also unfavorable; redundancies in the network architecture also have the potential to negatively impact test time performance.

This motivates a more comprehensive view of the inner workings of a deep neural network, taking a deep dive into each of its components. A common approach is to directly examine its weights, but the tradeoff is potentially missing out on information about the network structure. To take the middle ground, we analyze structural characteristics arising from *layerwise* spectral distributions in order to explain network performance and inform training procedures. We find that (1) allocating learning rate across layers based on measurements of their spectral distribution results in more improvement on “vanilla” architectures such as VGG19, i.e. networks without built-in interactions among layers; and (2) using the same measurements to inform channel pruning on DenseNet40 leads to our model implicitly gaining self-awareness of its “bottleneck” layers to maintain higher accuracies.

I dedicate this thesis to the greatest of first loves.

Mom and Dad, for your unconditional support and immeasurable sacrifices,
even through some of my most incorrigible years.

To my grandparents, for always choosing to see good, do good, be good.

To my uncle, without whom I could not imagine where I would be today.

I am blessed to grow up surrounded by the strongest people I know.
If given the chance, I would choose you as family in every universe.
Thank you for giving me the world as I stand on your shoulders.

Acknowledgments

Here is a short list of people I would like to thank, from the moment I entered UC Berkeley as an eighteen-year-old who thought she knew it all, to the confused yet somehow-in-one-piece twenty-three-year-old writing this thesis today.

I would like to start by thanking my exceptional and benevolent advisor **Professor Kannan Ramchandran** for his wisdom and reassurance through the years. I will never forget being amazed at the way he taught probability on the chalkboard when I first met him as a student in EECS126, Spring 2020. I continued to take EE229A with him the following semester, where I learned profound ideas in using mathematical approaches to model human communication, a long term goal of mine. As my research advisor, he encouraged me to (i) tune into my intuition while staying skeptical, and (ii) dive deeper to find “the spark.” There have been times when he identified my thoughts before I could gather them coherently myself. I would like to thank my mentor **Yaoqing Yang** for welcoming me aboard the pruning robustness project with virtually no research experience in Fall 2020; for not giving up on me even when I thought I have given him many reasons to; and for encouraging me to pursue research and academia. Their guidance, support and discussions on the theoretical and empirical aspects of this ever-evolving project over the past 3 years have taught me what it means to learn, to teach and to work in science. Yet more importantly, their patience, forgiveness and dedication have shown me what it takes to be an educator and a researcher.

They also introduced me to brilliant members of the research communities at Berkeley. I would like to thank **Yefan Zhou** and **Alex Zhao** for tirelessly mentoring me when I first started research and for setting the bar high since day one. They started from showing me how to submit slurm jobs and setup PyTorch environments. I would like to express my deepest appreciation for **Landon Butler, Justin S. Kang, Yigit Efe Erginbas, Syomantak Chaudhuri, Nived Rajaraman, Ajil Jalal, Jichan Chung, Karna Mendonca, Arin Chang**, for welcoming me into the BLISS family with open arms and for putting up with all of my shenanigans this year; and **Efe Aras**, who will be missed dearly in the lab as he ventures to New York City, for sharing *entropic* contributions with me. I will always carry the nuggets of wisdom they have generously shared with me to conduct, discuss, analyze, and write about research.

I would like to thank my intern manager **Shanshan Zhang** and peer-turned-mentor **Yan Luo** at Facebook, for sparking my interests in research, for making me realize we could do much more than software using software. They blessed me with trust and guidance through my first industry research project, shielded me from corporate tides, and made me feel at home in OSI albeit virtually.

To **Professor Jelani Nelson**, with whom I had a blast teaching CS170, and from whom I learned incredible life stories. I will always remember and be inspired by his passion to make CS education equitable without borders, along with the endless energy he brings to lectures and staff meetings, which gets everyone motivated to go to the extra mile.

To the exceptional faculty members from the Math and French departments I have had the fortune of taking courses with 2+ times: **Ryan Hass, Professor Richard Kern, Professor Ariel Shannon, Professor Susan Maslan**. Along with everyone in my home department, they complete the narrative of a liberal arts education at Berkeley. Their classrooms and

office hours have offered much joy and comfort to me, even (or rather, especially) on some of my most difficult days.

I could not have made it this far, or even taken the first steps, in this field of study without my NJ mentors, whose values and philosophies keep my eyes on the stars and feet on the ground: **Derek Wan** for his encouraging emails through CS61A, for giving the best actionable advice and sharing my small successes since I came to Cal; **Lakshya Jain** for knowing exactly what I go through and how I feel without a word spoken; **Kevin Miao** and **Adrian Chiu** for reminding me to dream big, create visions, believe, and be.

I would like to thank **Andrea Mejía Valencia** for her advice and support through my frosh-soph years in the CS program; **Heather Levien** and **Leslie Mach** for helping me navigate uncertainties as an upperclassman through quarantine. I am extremely grateful for **Michael Sun**, **Patrick Hernan**, **Shirley Salanio**, **Glenna Anton**, for helping me find my bearings in the department; and the staff members of Cory 264 and Soda 465, for making lab work possible. To the professional community of **Upsilon Pi Epsilon** for showing us the ropes of being a part of this ever-growing industry. To the wonderful community of volunteers I met through **CS Mentors**, whose dedication to mentorship and education never fail to move me. Shoutout to **Ryan Nuqui**, for there was no way I could have co-coordinated 61B without him.

Here is to the study crew **Clark D. Wang**, **Thomas Lu**, **Robert Quach**, **Sean Chen**, **Adam Chois**, **Michael Z. Wright**, **Jonathan Wang**, **David Shau** et al., for being a blast through in-person lectures, recruiting, long-winded problem sets, full-day practice test grinds, existential musings, spiritual discoveries, and COVID-19. Special tribute to James K. Moffitt, C. V. Starr East Asian, Charles Franklin Doe Memorial, Bancroft, Kresge Engineering, Mathematics Statistics libraries, Main Stacks, and Wheeler 150.

To my dearest **Rita L. Wang**, **Rachel Lam**, **Cameron L. Cheung**, **Jishing I. Yu**, **Matthew Zhang**, **Alexis Tran-Luu**, **Felicity Wang**, **Jasmine M. Wu**, **Evan J. Sakuma**, **Katherine Zhou**, **Ashley Su**: for growing with me through the years and years and for being my brightest stars. Their souls shine with such spirit, sincerity and integrity, and their friendships have made California feel like home.

Last but not least, to my family, who try their absolute best to understand and share my moments of joy, pain, growth: for enduring my never-ending cycles of self-doubt, for anchoring me to reality when I spiraled, and for creating countless delicious meals that convey all their love and care. I apologize for instances of my absence or absentmindedness in times of need and celebration in the past 5 years. In becoming me, I have wanted to become great for them.

To the inexhaustible list of names not on here who have shown me kindness and grace, my sincerest gratitude for their invaluable trust and opportunities.

It is an understatement to say that I am beyond lucky to have you be part of my life, hold up my world, and make me who I am today. You are constant reminders that I am on a journey of seeking the true treasures.

Contents

Contents	5
List of Figures	6
1 Introduction	1
1.1 Problem Statement	1
1.2 Report Overview	1
1.3 Deep Neural Networks and Regularization	1
1.4 Heavy-Tail Self-Regularization (HT-SR) in Layer Weight Matrices	5
2 Learning Rate Allocation Across Layer Weight Matrices	9
2.1 Related Work	9
2.2 Experimental Setup	11
2.3 Heavy-tailed Metrics for Layerwise Allocation	11
2.4 Accounting for Randomness in Observations of Heavy-tailed Metrics	18
2.5 Future Work	20
3 Layerwise Structured Pruning of Weight Matrices	22
3.1 Related Work	22
3.2 Comparing Metrics for Assigning Pruning Ratios	26
3.3 Layerwise Learning Rates with Uniform Pruning	28
3.4 Future Work	36
Bibliography	41
A Convolution of Exponential and Normal Distributions	49

List of Figures

1.1	Simplified Sketch of the hierarchical structure of a deep neural network.	2
1.2	Finding an optimal learning rate is crucial for reaching global optima.	4
1.3	Log-log ESD plot showing a favorable linear decay (left), corresponding to heavy-tailed behaviors in the original ESD vs. one with more randomness (right), resembling a Gaussian distribution closer to correlation ESDs at initialization.	8
2.1	Test and Train improvements on shallow ResNets realized by weighted α probabilities 2.3, at different widths and learning rates. On ResNet34, test time improvements (blue) are more prominent (especially at initial learning rate of 0.2) while suffering minimal training accuracy loss compared to ResNet18, especially at 25%-width.	12
2.2	The CKA similarity between every other layer on the baseline (left) ResNet34 model (with 50%-width, initial learning rate 0.2) vs. using learning rate allocations by α probabilities from 2.3 (right). The baseline plot shows high similarities between adjacent layers throughout the model while more differences arise between the shallower and deeper layers with the learning rate allocations.	13
2.3	Learning rate allocations on ResNet18 (left) and ResNet34 (right) using weighted probability of α values mapped to identity (orange), square root (green) , cube root (red), logarithm base-2 (purple).	15
2.4	Plots of learning rates assigned to parameter groups using each scheme (left) and the learning rates ranked in descending order (right). The mapping function of the α 's before converting them to weighted probabilities is important in determining their scaling relationship. This plot informs our results that we want variations among layerwise learning rates while keeping the majority of them as far from 0 as possible.	16
2.5	Plots of learning rates assigned to parameter groups in order (left) using Softmax weighted probability 2.19 and Negative powers of two 2.3 and the learning rates ranked in descending order (right). Most parameters have learning rates assigned 0, so the network is not able to train to full saturation under such schemes.	17
2.6	Approximate exponential distribution of α values emerge towards the end of training on the ResNet18 (left) and ResNet34 (right) models with 0.25-width fraction, initial learning rate 0.2.	20

3.1	Comparing layerwise pruning ratios achieved on WideResNet28 (blue) and DenseNet40 (orange) using Erdos-Renyi Kernel layerwise [18], Layer-adaptive Sparsity for the Magnitude-based Pruning (LAMP) [51], and uniform (baseline) schemes.	24
3.2	DenseNet40 (top) and WRN28 (bottom): Effects of AugMix (blue) and traditional augmentation (light blue) on α (left) and matrix entropy (right) through initial training, before pruning.	25
3.3	Clean test accuracy vs. number of parameters remaining on DenseNet40 : Using Different Shape (blue-green colors) and Scale (red-pink colors) Metrics vs. uniform (“even”) pruning (yellow).	29
3.4	Prune more for layers with larger metrics.	30
3.5	Prune more for layers with smaller metrics.	30
3.6	Layerwise pruning ratios with traditional augmentation on DenseNet40 . Columns represent the different 10 HT-SR metrics. Uniform pruning would show a straight line across the graph. From bottom to top row, in increasing order of model pruning ratio.	30
3.7	Prune more for layers with larger metrics.	31
3.8	Prune more for layers with smaller metrics.	31
3.9	Layerwise pruning ratios with AugMix on DenseNet40 . Columns represent the different 10 HT-SR metrics. Uniform pruning would show a straight line across the graph. From bottom to top row, in increasing order of model pruning ratio.	31
3.10	Prune more for layers with larger metrics.	32
3.11	Prune more for layers with smaller metrics.	32
3.12	Layerwise pruning ratios with traditional augmentation on MobileNetV2 . Columns represent the different 10 HT-SR metrics. Uniform pruning would show a straight line across the graph. From bottom to top row, in increasing order of model pruning ratio.	32
3.13	Prune more for layers with larger metrics.	33
3.14	Prune more for layers with smaller metrics.	33
3.15	Layerwise pruning ratios with AugMix on MobileNetV2 . Columns represent the different 10 HT-SR metrics. Uniform pruning would show a straight line across the graph. From bottom to top row, in increasing order of model pruning ratio.	33
3.16	Prune more for layers with larger metrics.	34
3.17	Prune more for layers with smaller metrics.	34
3.18	Layerwise pruning ratios with traditional augmentation on WideResNet28 . Columns represent the different 10 HT-SR metrics. Uniform pruning would show a straight line across the graph. From bottom to top row, in increasing order of model pruning ratio.	34
3.19	Prune more for layers with larger metrics.	35
3.20	Prune more for layers with smaller metrics.	35
3.21	Layerwise pruning ratios with AugMix on WideResNet28 . Columns represent the different 10 HT-SR metrics. Uniform pruning would show a straight line across the graph. From bottom to top row, in increasing order of model pruning ratio.	35

3.22	Zooming into select figures of layer α from VGG19 with 4M parameters 3.3. Lower overall average α values across all layers are recorded through the finetuning epochs compared to baseline, indicating overall better training completeness. The top four figures show that layers assigned learning rates with the largest difference from the baseline also have smaller α values through finetuning, showing better model convergence.	37
3.23	VGG19 at 11 Sizes: Change of α through finetuning for 3 layers with the largest increase and 3 layers with largest decrease from initial baseline learning rates (continued on next page).	38
3.24	(Continued from previous page) VGG19 at 11 Sizes: Change of α through finetuning for 3 layers with the largest increase and 3 layers with largest decrease from initial baseline learning rates.	39
3.25	Effects of layerwise learning rate allocation on uniformly pruned VGG19, across a wide range of sparsities.	39
3.26	Comparing the four settings on varying sizes on DenseNet40: uniform pruning and global learning rate (baseline), uniform pruning and layerwise learning rates determined by α , layerwise pruning ratios determined by α and global learning rate (α prune), and <i>both</i> layerwise pruning ratios and layerwise learning rates determined by α . Learning rates are reallocated for the finetuning stage only, not for the initial training.	40

Chapter 1

Introduction

1.1 Problem Statement

How much granularity can one have with the architecture and the training of deep neural networks? Past work in deep learning literature traditionally allocates resources and control training at the “blackbox” model level or takes the perspective from the weights level. In this thesis, we explore methods that reconcile these two granularity levels.

1.2 Report Overview

In the first half of this technical report, we discuss using heavy-tailed measurements to determine the amount of learning rate assigned to each layer weight matrix. For the second half, we use heavy-tailed measurements to inform structured pruning of the proportion of channels across the layers. Note that we will use “layer weight matrices” with “layer” and “layerwise” interchangeably in this thesis to refer to this level instead of the higher level layer abstraction.

1.3 Deep Neural Networks and Regularization

Supervised Learning

The three main types machine learning problems are supervised, unsupervised, and reinforcement learning. To formulate and solve each of these problems, we follow the machine learning pipeline of finding a model that minimizes the loss function and picking an optimizer to update our model. We focus on the supervised paradigm, where the goal is to have a model approximate a function that maps from input examples to target labels [8]. The two main types of supervised learning problems are classification and regression, where the model learns to predict a class label and the latter a numerical label, respectively.

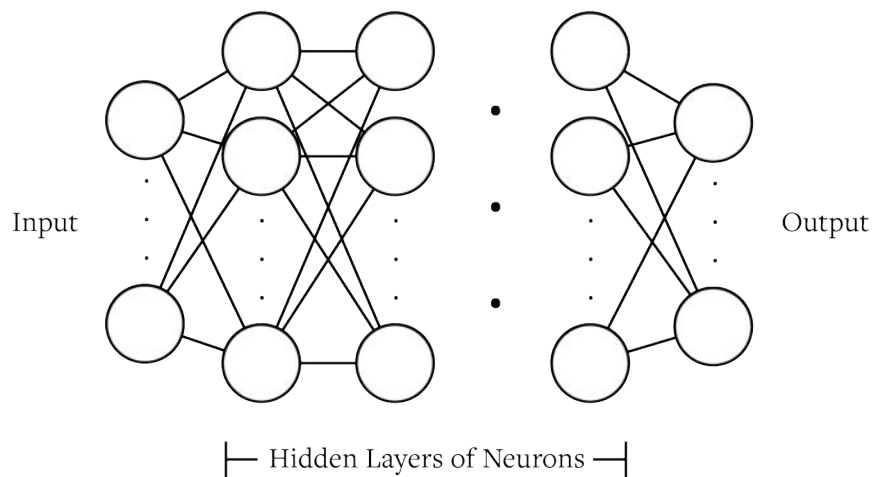


Figure 1.1: Simplified Sketch of the hierarchical structure of a deep neural network.

Deep Learning

To gain a more in-depth perspective of deep learning and deep neural networks, refer to [27]. Deep learning is a subfield in machine learning that attempts to understand and use data to fulfill tasks by attempting to simulate the behavior of the human brain. This motivates the structure of deep neural networks, which consist of a hierarchical organization of neuron layers that propagates computations from the first layer forward through the final layer.

The goal of the network is to learn parameters θ to approximate a function f to predict label y from input data x , i.e. minimizing the loss $\ell(\cdot, \cdot)$ for prediction \hat{y} :

$$\begin{aligned} \ell(\theta) &= \ell(y, \hat{y}), \\ \hat{y} &= f(x; \theta). \end{aligned} \tag{1.1}$$

We are using the chained layers to implement the approximated f , which could be considered as a composition of functions. For a particular epoch t , one would update the weight parameters with gradient descent for the next epoch $t + 1$,

$$\theta_{t+1} = \theta_t - \eta \cdot \frac{\delta \ell}{\delta \theta}, \tag{1.2}$$

where the loss function ℓ consists of penalizing incorrect network predictions and constraints on the weights.

Neurons connected between neighboring layers have weights that define how much the input impacts the output of the next neuron; these weights are updated to improve model performance. Backpropagation algorithms such as gradient descent update the weights of the network by correcting errors. The model ingests data in the input layer and gives a prediction at the output layer.

One could refer to [83] for an overview of the evolution of deep artificial neural networks since the 1940s (and the 1800s).

Generalization and Regularization

Formally, regularization is defined as “any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error” [27].

Most regularization strategies regularize estimators by reducing variance, at the expense of increased bias. The bias-variance tradeoff formulates the problem of finding the Goldilocks zone between overfitting and underfitting, which result from high variance and high bias, respectively.

Regularization could be implemented as a parameter norm penalty to limit the model capacity (e.g.: LASSO and ridge regression), including a regularization term $r(\theta)$ in the loss function:

$$\ell(\theta) = \ell(y, \hat{y}) + \lambda r(\theta). \quad (1.3)$$

The λ coefficient is commonly referred to as the weight decay term while the regularization function r imposes certain constraints on the weights during training.

l_0, l_1, l_2 **Regularization:** Given an n -length vector \mathbf{x} , we define its l_p -Norm as

$$\|\mathbf{x}\|_p = \sqrt[p]{\sum_{i=1}^n |x_i|^p}. \quad (1.4)$$

- The l_0 -Norm follows from definition to be the number of non-zero elements of the vector \mathbf{x} . l_0 -Regularization penalizes the objective function to minimize the number of non-zero parameters, an NP-hard optimization problem that is computationally difficult to solve.
- l_1 -Regularization, also called LASSO, is an approximation for l_0 -Regularization to yield more sparse solutions.
- l_2 -Regularization, or *ridge regression*, takes the form of the commonly used Euclidean norm. It penalizes large values of θ and yields less sparse solutions compared to the previous two forms.

Data augmentation adds transformed data to expand the size of the training dataset. More regularization methods include early stopping, label smoothing, dropout, noise, ensembling [9, 4, 70].

Prior work correlates complexity measures (norm-based, sharpness-based, optimization-based) to the generalization of a model, i.e. the model’s performance on unseen data from the same population as training data [42]. However, adding the complexity measure as a regularizer makes optimization more difficult and does not lead causality claims. [43] use correlations in the weight distribution as an explicit regularizer to improve generalization. Weight correlation decent (WCD) could complement common regularizers such as weight decay and dropout.

Both [93] and [65] produce phase plots relating different combinations of varying amounts of explicit regularization methods to effect generalization. For example, finding the optimal amount of added noise, which decreases the amount of effective data relative to the

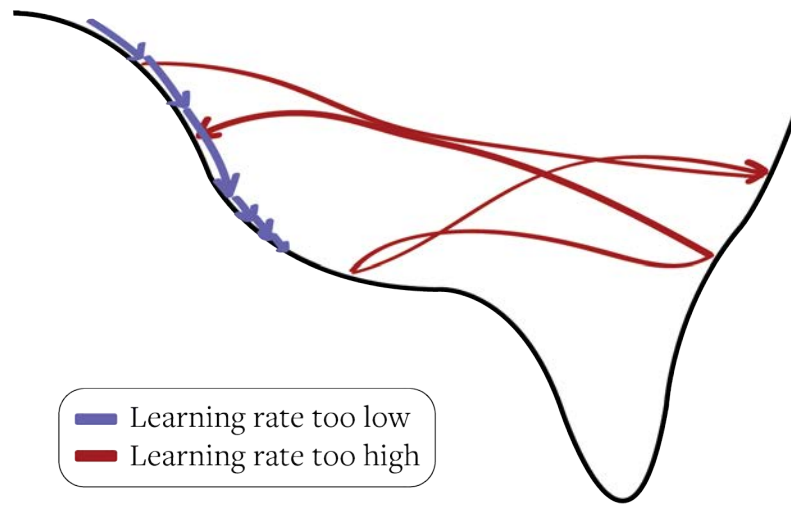


Figure 1.2: Finding an optimal learning rate is crucial for reaching global optima.

model complexity, and early stopping (or higher learning rates) could improve generalization [65]. [93] shows empirically that good generalization is correlated with locally flat, globally smooth loss landscape, which is a structure that could be realized by explicit regularization.

Learning Rate and Stochastic Gradient Descent

The $\frac{\partial \ell}{\partial \theta}$ gradient term requires expensive computation over the entire dataset.

In response, one could use stochastic single example to update the gradient over a single training point, or perform stochastic mini-batch update using a random set of samples [45].

This is effective in making the gradient computation cheaper without using potentially redundant information in large datasets with similar examples. It does naturally increase the chances of staying trapped in local minima or encountering saddle-points, where the function is neither a local maximum value nor a local minimum value, especially in high-dimensional settings.

The performance of stochastic gradient descent (SGD) then depends largely on the role of the learning rate η_t at every epoch t , which is crucial in finding optima 1.2. Larger η_t causes updates to be too large and skip over the optimal region(s) while smaller η_t would delay convergence in a reasonable number of steps.

Most state-of-the-art methods assign one single global learning rate to all parameters, allowing minimal granularity in tuning such settings across the network.

Implicit Regularization

The aforementioned methods are considered explicit regularization, where it takes on the form of a hyperparameter or direct modification of the model or data. Without imposing

restrictions on parameters or including any forms of explicit regularization, implicit regularization results unexpectedly from the optimization method to improve generalization [72].

[73] proposes the existence of implicit regularization on capacity control, unrelated to model size. Even with large number of parameters, the model implicitly finds a solution with small “complexity”, i.e. implicit norm regularization, without lowering the test time performance. The authors propose that increasing the model complexity might allow for “low complexity” solutions and generalize better. This implicit regularization could then motivate an infinite network.

This report follows prior work analyzing Heavy-Tailed properties that arise in the correlations of deep neural network weight matrices and characterize the increase of *Implicit Self-Regularization* over the course of training [65, 64, 61]. We formally introduce Heavy-Tail Self-Regularization (HT-SR) in the following section.

1.4 Heavy-Tail Self-Regularization (HT-SR) in Layer Weight Matrices

Layer Weight Matrices

This thesis work is motivated by the analysis of layer weight matrix convergence during neural network training. Instead of the commonly termed layer abstractions in Deep Neural Networks, we refer to the layer weight matrices, the lowest level of abstraction in models.

Each layer abstraction consists of different data matrices representing its different counterparts. For example, the first 3×3 convolutional layer of DenseNet40 has one layer weight matrix; both the three transportation layers and the 36 layers across the 3 dense blocks include weight and bias matrices. We analyze only the weight matrices for each layer.

Note that a single layer abstraction could encapsulate multiple layer weight matrices. For example, aside from the layer weight matrices in the first convolution layer and the final linear layer, the 8 layer abstractions in the 18-layer ResNet18 encompass shortcut layer(s), two convolutional layers, each followed by a batch norm layer [33]. We analyze the weight matrices of the $8 \times 2 = 16$ convolutional layers, along with the matrices representing the projection shortcuts that go across feature maps of two sizes. These match dimensions using 1×1 convolutions with a stride of two.

Note that if we use zero-mapping shortcuts to match the dimensions instead of the projection matrix, we do not introduce new parameters; similarly, the other residual shortcuts are identity mapping, which does not entail extra parameters [35].

We regularize deep neural networks from a layer weight matrices perspective, resulting in more granularity than at the model level.

Empirical Spectral Density (ESD)

The spectrum of a $n \times m$ matrix is the set of its eigenvalues $\Lambda = \{\lambda_1, \dots, \lambda_n\} \subset \mathbb{R}$. The empirical spectral density (ESD), or spectral distribution, of the matrix counts the occurrence

of each eigenvalue, normalized by the total count [48]:

$$\mu(\Lambda) = \frac{1}{n} \#\{\lambda_i \in \Lambda\}. \quad (1.5)$$

Prior work analyzes the behaviors of the limiting spectral distribution for certain classes of random matrices, including the circular law, which states that the ESD of the covariance matrix converges almost surely to the uniform distribution on the unit disk $\{z \in \mathbf{C} : |z| \leq 1\}$ [94, 5, 24, 86, 28]. The Marchenko-Pastur law [63] states that assuming $n/m \rightarrow y \in (0, 1]$, the spectral distribution converges to a deterministic measure, whose density is given by

$$\frac{d\mu}{dx} = \frac{1}{2\pi xy} \sqrt{(b-x)(x-a)} 1_{(a \leq x \leq b)}, \quad (1.6)$$

where $a(y) = (1 - \sqrt{y})^2$, $b(y) = (1 + \sqrt{y})^2$.

[74] shows the limiting ESD of real random matrices with dependent entries is also given by the Marchenko-Pastur law and gives a rate of convergence of the *expected* ESD.

We contextualize the study of ESDs to analyze our layer weight matrices. We provide the steps of computing the ESDs. For each layer weight matrix \mathbf{W} , we compute its correlation matrix $\mathbf{X} = \frac{1}{N} \mathbf{W}^T \mathbf{W}$, whose (i, j) -entry is the correlation between the i th and j th columns \mathbf{W} . The i th and j th columns of \mathbf{W} are the weight vectors of the i th and j th neurons, respectively, for the weight matrix of every layer. Large values in the correlation matrix denote a strong linear relationship between two neurons [77].

For each correlation matrix \mathbf{X} , we compute its empirical spectral density (ESD), i.e. a histogram showing the distribution of eigenvalues of \mathbf{X} . Prior work analyzes the ESD of covariance matrices and their convergence through training [43, 59, 89]. The following are steps to compute the ESD of layer weight matrix $\mathbf{W}_i \in \mathbb{R}^{n \times m}$, for $i \in \{1, \dots, n\}$:

1. Compute $\mathbf{X}_i = \frac{1}{N} \mathbf{W}_i^T \mathbf{W}_i$, correlation matrix of \mathbf{W}_i .
2. Collect the eigenvalues $\Lambda_i = \{\lambda_j(\mathbf{X}_i) \in \mathbb{R} : j = 1, \dots, n\}$.
3. ESD := estimate the density distribution over Λ_i .

Heavy-Tailed Spectral Distribution of Correlations

To observe the *heavy-tailedness* of the ESD that gradually arises through training, we fit an estimated power-law (PL) to the tail-end of the ESD through maximum likelihood estimation (MLE) of α [14, 3], where the fitted power-law takes on the following form,

$$p(x) \propto x^{-\alpha}, \quad x_{\min} < x < x_{\max}, \quad (1.7)$$

where x_{\min} is the lower bound of the power-law behavior.

Given the likelihood measure,

$$p(x|\alpha) = \prod_{i=1}^n \frac{\alpha - 1}{x_{\min}} \left(\frac{x_i}{x_{\min}} \right)^{-\alpha}, \quad (1.8)$$

the following are steps to derive $\bar{\alpha}$, the MLE of α :

$$\begin{aligned}
\mathcal{L} &= \ln(p(x|\alpha)) = \sum_{i=1}^n [\ln(\alpha - 1) - \ln(x_{\min}) - \alpha \ln(\frac{x_i}{x_{\min}})] \\
&= n \ln(\alpha - 1) - n \ln(x_{\min}) - \alpha \sum_{i=1}^n \ln(\frac{x_i}{x_{\min}}) \\
\frac{\delta \mathcal{L}}{\delta \alpha} &= \frac{n}{\alpha - 1} - \sum_{i=1}^n \ln(\frac{x_i}{x_{\min}}) = 0 \\
\bar{\alpha} &= \frac{n}{\sum_{i=1}^n \ln(\frac{x_i}{x_{\min}})} + 1.
\end{aligned} \tag{1.9}$$

The power-law (PL) exponent (slope) of the ESD, α , assigns a quality score for each layer of the network. x_{\max} denotes the maximum eigenvalue, $\max_j \lambda_j$, of the covariance matrix while x_{\min} localizes where the power-law structure is first observed in the ESD. Recent work shows that α predicts the trends of model quality and generalization abilities in modern neural networks on both computer vision (CV) and natural language processing (NLP) tasks, suggesting a negative correlation between α and test time performance within the optimal range of $\alpha \in [2, 6]$ [66, 67, 92, 64]. An $\alpha > 6$ value indicates an undertrained layer weight matrix, with correlations resembling those of a Gaussian distribution, while an $\alpha < 2$ value indicates an overtrained layer weight matrix in the ‘‘rank collapse’’ phase, where its ESD is dominated by one or few very large eigenvalues. This corresponds to an ESD with a flattened tail, where the correlation spectrum is dominated by unusually large elements, resulting in so-called ‘‘correlation traps.’’ The authors of [64] find no rank-collapse on the natural language processing models from AllenNLP [22].

The regularized and *well-trained* matrices exhibit correlations that behave similarly to those of random matrices with entries drawn from non-Gaussian Universality classes (e.g., power-law distribution with heavy tails), which have traditionally modeled strongly correlated physical systems [6]. Therefore, it is likely that a matrix whose correlations exhibit Heavy-Tailed properties could be characterized as a matrix that is implicitly self-regularized and *well-trained*. A matrix with strong correlations has the Empirical Spectral Density (ESD) of its correlation matrix display a Heavy-Tailed distribution, which would show up as a steady linear decay in a log-log ESD plot as in 1.3.

Deep Neural Networks are commonly initialized with the Glorot (or Xavier) initialization [25], with biases as 0 and layer weight entries as

$$(W_l)_{ij} \sim U[-\frac{1}{\sqrt{n_c}}, \frac{1}{\sqrt{n_c}}], \tag{1.10}$$

where U denotes a uniform distribution and n_c is the fan-in dimension, i.e. number of columns in W_{l-1} .

For networks that use the rectified linear activation function (ReLU), we need to account for half of the layer output being 0. We use the He initialization [34] instead such that for each weight entry,

$$(W_l)_{ij} \sim \mathcal{N}\left(0, \frac{2}{\sqrt{n_c}}\right). \tag{1.11}$$

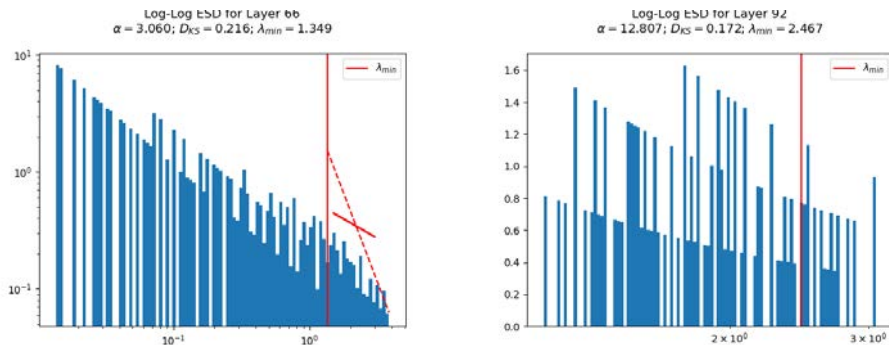


Figure 1.3: Log-log ESD plot showing a favorable linear decay (left), corresponding to heavy-tailed behaviors in the original ESD vs. one with more randomness (right), resembling a Gaussian distribution closer to correlation ESDs at initialization.

Empirically, ESDs of layer weight matrix correlations follow a Gaussian distribution in the beginning of training, which converges to a heavy-tailed power-law distribution through training. Weight matrices with correlation ESDs that exhibit heavy-tailed behaviors have been empirically shown to correlate with improved generalization on models, characterized by the generalization gap (i.e., the difference between model performance on training data and its performance on unseen data from the same distribution) [64, 67, 92].

Model generalization is correlated with levels of implicit regularization, as defined in Heavy-Tail Self-Regularization (HT-SR) Theory, and this line of work is motivated by examining measurements that leave a “signature” on the model components, i.e. its layer weight matrices.

Chapter 2

Learning Rate Allocation Across Layer Weight Matrices

The discussion of this section uses the power-law coefficient, α , to characterize heavy-tailedness in correlation ESDs, as defined in the previous chapter. One could also attempt to generalize to a wide selection of scale-based, shape-based, and hybrid metrics characterizing heavy-tailed distributions, available in the WeightWatcher package.

Smaller α measurements correlate with a more well-trained layer weight matrix while larger α is correlated with a layer weight matrix whose correlations have Gaussian-like ESDs, demonstrating more random behaviors closer to those at initialization.

2.1 Related Work

Multiple previous work has explored optimizers and learning rate assignment over the past decade of the Deep Learning literature. Optimizers often take an exponentially weighted average (v_t) of gradients g_t through training, with β as the learning parameter

$$v_t = \beta v_{t-1} + (1 - \beta)g_t. \quad (2.1)$$

In practice, stochastic gradient descent (SGD) is used with momentum as introduced in [80], so that the weight update is a linear combination of the previous update and the exponentially weighted average of gradients,

$$w_{t+1} = w_t - \eta \cdot v_t, \quad (2.2)$$

where η is the learning rate. The learning parameter β is commonly set to 0.9 in this case.

RMSProp [87] keeps a weighted average of its squared gradients to scale the gradient,

$$\begin{aligned} v_t &= \beta v_{t-1} + (1 - \beta)g_t^2 \\ w_{t+1} &= w_t - \frac{\eta}{\sqrt{v_t + \epsilon}} \cdot g_t, \end{aligned} \quad (2.3)$$

with ϵ added to avoid divide by zero errors. Adam (*adaptive moment estimation*) combines SGD with momentum and RMSProp to compute individual adaptive learning rates for different parameters using first and second moment gradients [46]. Adam tracks two exponentially weighted averages

$$\begin{aligned} v_t &= \beta_1 v_{t-1} + (1 - \beta_1) g_t \\ s_t &= \beta_2 s_{t-1} + (1 - \beta_2) g_t^2 \end{aligned} \quad (2.4)$$

Weight update is defined as

$$w_{t+1} = w_t - \eta \frac{v_t}{\sqrt{s_t + \epsilon}} \cdot g_t \quad (2.5)$$

Common hyperparameter settings for Adam include

$$\begin{aligned} \eta &= 0.001, \\ \beta_1 &= 0.9, \\ \beta_2 &= 0.999, \\ \text{and } \epsilon &= 10^{-8}. \end{aligned} \quad (2.6)$$

SGD, Momentum, and AdaGrad are popular optimization methods that require setting the learning rate to the correct magnitude to ensure good convergence to optima. [55] increases the learning rate at an exponential rate, switches to an exponential growth with lower exponent when validation loss levels off, and repeats this procedure until model convergence.

[82] introduces variance-based SGD (vSGD) to find optimal learning rates that minimize the expected loss from weight updates; the network learns to progressively decrease the learning rate towards its optimal value, without a schedule set in advance.

[91] proposes a dynamically updated adaptive learning rate through an iterative process between a learning rate controller who proposes learning rates and a trainee network, which is trained to report back learning rates that reduce validation loss. Instead of a stepwise or exponentially decreasing schedule, [85] explores cyclical learning rates that vary between bounds. They introduce triangular, Welch, and Hann learning rate policies, where the learning rates experience linear, parabolic, and sinusoidal increase then decrease, respectively.

The space of layerwise learning rate schemes have been explored but not extensively or with substantial backing. MetaLR learns to assign learning rates to each layer automatically and proposes a bi-directional fine-tuning scheme, motivated by the hypothesis that lower-level layers are domain-specific while higher-level layers are more task specific [12]. However, nested, bi-level optimization schemes are computationally expensive in practice [19].

AutoLR proposes to contradict the previous notion that lower-level layers extract general features while higher-level layers extract specific features [78]. The authors propose an ordering of the layerwise weight variations such that for the k -th layer,

$$v_t^k = \frac{1}{n_k} \|\Delta w_t^k\|, \quad (2.7)$$

which should be small for low-level layers while large for high-level layers to adopt themselves to a new task. The key to the algorithm is automatically tuning the learning rate of

each layer to control all weight variations such that

$$v_t^1 \leq v_t^2 \leq \dots \leq v_t^K. \quad (2.8)$$

2.2 Experimental Setup

Dataset

We use the CIFAR-100 dataset consisting of 100 classes (20 superclasses), each with 500 training images and 100 images for testing [49].

Architecture

We conduct our experiments on the ResNet architecture introduced in [33, 90] with varying depths. Namely, we try to determine trends in the shallower 18-layer and 34-layer ResNets and also try to observe effects on the deeper 101-layer and 152-layer ResNets.

Effects of the skip connections are shown to be stronger on the deeper ResNets than on their shallower counterparts. Residual networks include skip connections that make the output of one layer the input of layers deeper in the network to ameliorate impact of vanishing or exploding gradients. In other words, let the output of any stacked layers be $f_n(x)$ for input x on the plain network, including their activation functions (e.g.: ReLU), then a residual net would have output

$$f_n(x) + x. \quad (2.9)$$

To observe the effects of our algorithm on networks with various depths and widths, we conduct experiments on ResNets to the following fractions of their original width:

$$\{0.25, 0.50, 1.0, 1.5, 2, 2.5\}, \quad (2.10)$$

with initial learning rates

$$\{0.025, 0.05, 0.075, 0.1, 0.15, 0.2\}. \quad (2.11)$$

2.3 Heavy-tailed Metrics for Layerwise Allocation

We attempt to assign explicit regularization to matrices whose correlation ESDs exhibiting less power-law behaviors. For instance, we could give these less well-trained layer weight matrices larger learning rates and larger weight decay values.

The work in this chapter is based on the `TempBalance` algorithm by Tianyu Pang, with a high-level description as follows: One usually considers regularization from the model perspective, such as assigning a universal learning rate for all model parameters. Then, given n layer weight matrices, each would be assigned $\frac{1}{n}$ -proportion of the model-wise regularization.

We assign the parameter groups associated with each layer weight matrix a different learning rate, but one could generalize to other training hyperparameters such as weight decay.

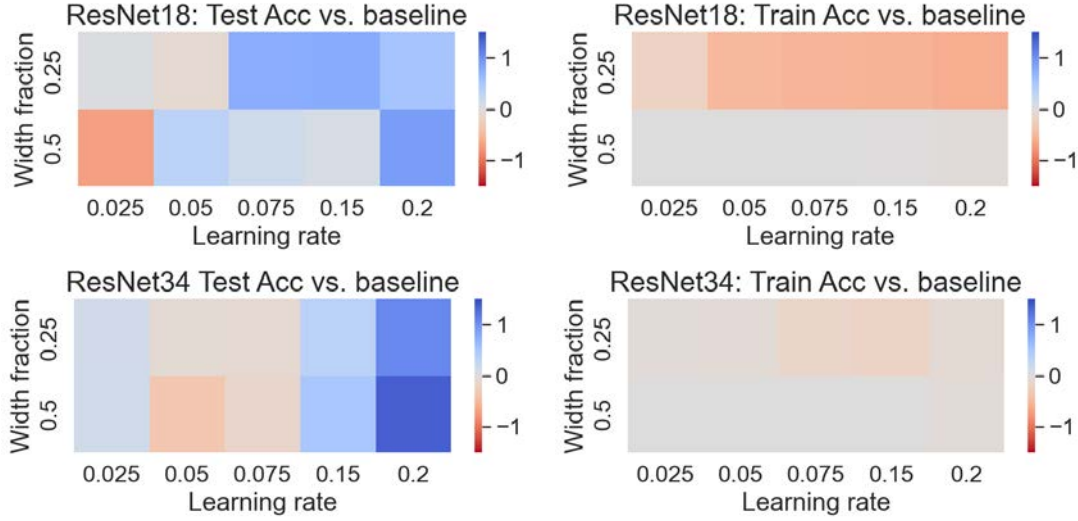


Figure 2.1: Test and Train improvements on shallow ResNets realized by weighted α probabilities 2.3, at different widths and learning rates. On ResNet34, test time improvements (blue) are more prominent (especially at initial learning rate of 0.2) while suffering minimal training accuracy loss compared to ResNet18, especially at 25%-width.

Heavy-tailed Metrics as Weighted Probabilities

A natural starting point is to take the power-law α associated with each layer weight matrix W_i , normalize over their sum, and assign the parameters associated each layer weight matrix $\frac{\alpha_i}{\sum_{i=1}^n \alpha_i}$ -fraction of the universal learning rate. In practice, this allocation is performed at every epoch t , for $T(= 200)$ total epochs. For simplicity, we use the cosine annealing learning rate schedule for baseline comparison to analyze the effects of allocating learning rates by layer,

$$lr_t = \frac{lr_0}{2} \cdot \left(1 + \cos\left(\frac{(t+1) \cdot \pi}{T}\right)\right). \quad (2.12)$$

In other words, we use $\alpha_{t,i}$ to assign $lr_{t,i}$. The weighted average learning rate for every layer i at every epoch t is,

$$lr_{t,i} = lr_t \cdot n \cdot \frac{\alpha_{t,i}}{\sum_{i=1}^n \alpha_{t,i}}. \quad (2.13)$$

CKA Similarity [93] uses the correlation-like metric, centered kernel alignment (CKA) similarity introduced in [47], to measure similarity between outputs of different trained models. The normalized index CKA is defined as

$$\text{CKA}(K, L) = \frac{\text{HSIC}(K, L)}{\sqrt{\text{HSIC}(K, K) \text{HSIC}(L, L)}}, \quad (2.14)$$

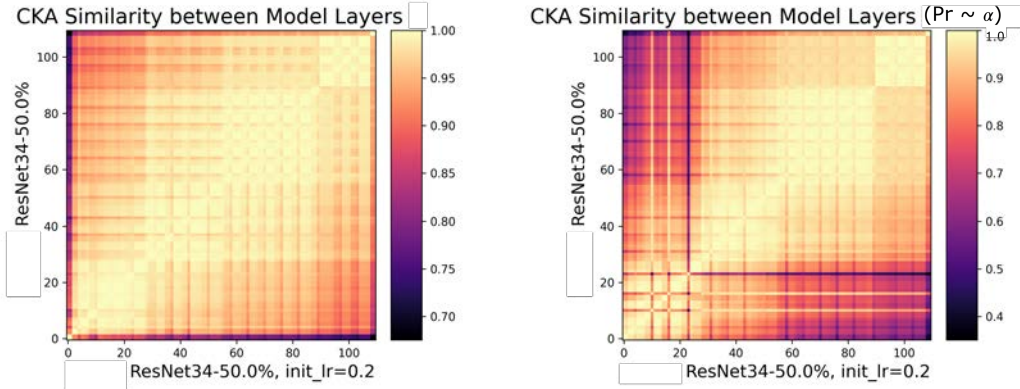


Figure 2.2: The CKA similarity between every other layer on the baseline (left) ResNet34 model (with 50%-width, initial learning rate 0.2) vs. using learning rate allocations by α probabilities from 2.3 (right). The baseline plot shows high similarities between adjacent layers throughout the model while more differences arise between the shallower and deeper layers with the learning rate allocations.

where the empirical estimator of Hilbert-Schmidt Independence Criterion (HSIC) is

$$\text{HSIC}(K, L) = \frac{1}{(n-1)^2} \text{tr}(KHLH),$$

with H as the centering matrix

$$H_n = I_n - \frac{1}{n} \mathbf{1}\mathbf{1}^T, \quad (2.15)$$

$$K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$$

$$\text{and } L_{ij} = l(\mathbf{y}_i, \mathbf{y}_j),$$

k and l are kernels.

The authors claim that CKA measures hidden-layer correspondences in neural networks with different widths and with different random initializations at different widths, fulfilling invariance to orthogonal transformation and isotropic scaling. In this work, we compare the CKA (representational) similarity plots between layers within one model. Concretely, we compare the weighted average α for learning rate allocation against baseline for ResNet34, 50%-width, $lr_0 = 0.2$ in 2.3.

Note the layer count is not the same as the number of layer representations. The axes here represent the layer modules. Each ResNet layer includes the following modules: layer wrappers, two convolutional layers, two batch norm layers, and one or three shortcuts. ResNet18 has 61 modules and ResNet34 has 109 modules.

A limitation is that the CKA similarity plots are generated on a random batch each time, so the similarities differ accordingly on other models whose plots excluded. Future work is needed to incorporate the entire dataset or include data augmentations.

Variations of α as Weighted Probabilities

We assign learning rates based on a linear weighted relationship among α 's in the previous section. We can incorporate powers into the probability weighting,

$$lr_{t,i} = lr_t \cdot n \cdot \frac{\alpha_{t,i}^2}{\sum_{i=1}^n \alpha_{t,i}^2}. \quad (2.16)$$

$$lr_{t,i} = lr_t \cdot n \cdot \frac{\alpha_{t,i}^3}{\sum_{i=1}^n \alpha_{t,i}^3}. \quad (2.17)$$

With squared average and larger initial learning rates $\{0.15, 0.2\}$, the smallest ResNet model (ResNet18, 25%-width) outperforms baseline during test time (70.09 vs. 69.124 and 69.842 vs. 69.57, respectively), but trained to less saturation. Results worsen with cubed average, exhibiting lower test accuracies and more unsaturated train accuracies.

Instead, we try the “opposite” direction of function classes, e.g.

$$\alpha_{t,i}^{\frac{1}{2}}, \alpha_{t,i}^{\frac{1}{3}}, \alpha_{t,i}^{\frac{1}{4}} \dots \quad (2.18)$$

along with $\log(\alpha_{t,i})$. In our experiments, the square root or logarithmic functions improve test time performance on most models.

Another common approach for probability assignment is using the softmax function,

$$lr_{t,i} = lr_t \cdot n \cdot \frac{e^{\alpha_{t,i}}}{\sum_{i=1}^n e^{\alpha_{t,i}}}. \quad (2.19)$$

Similar to increasing the power on $\alpha_{i,j}$'s, softmax does not work as most learning rates are assigned values of 0, as e^x approaches positive infinity quickly as x grows.

Heavy-tailed Metrics as Reference for Permutation

Instead of using weighted variations of α values directly in the assignment, we can define a probability distribution and use α 's as reference to give us a permutation of the probabilities for each layer weight matrix.

Probability Distribution: Negative Powers of Two One inspiration comes from Huffman Coding, whose optimality of expected codelength is guaranteed when the probabilities of source symbols are negative powers of two [60]. In the context of layerwise learning rates, we can consider entropy from two perspectives.

1. Across layer weight matrices: For the baseline scenario, we evenly assign the model's learning rate across all layer weight matrices like a uniform distribution, which has maximum entropy. By assigning layerwise learning rate we are decreasing the entropy.
2. Across learning rates: If the learning rate values are the random variable, the baseline random variable has no entropy as it takes on one value at every epoch. Our layerwise assignment would then increase the entropy of this random variable.

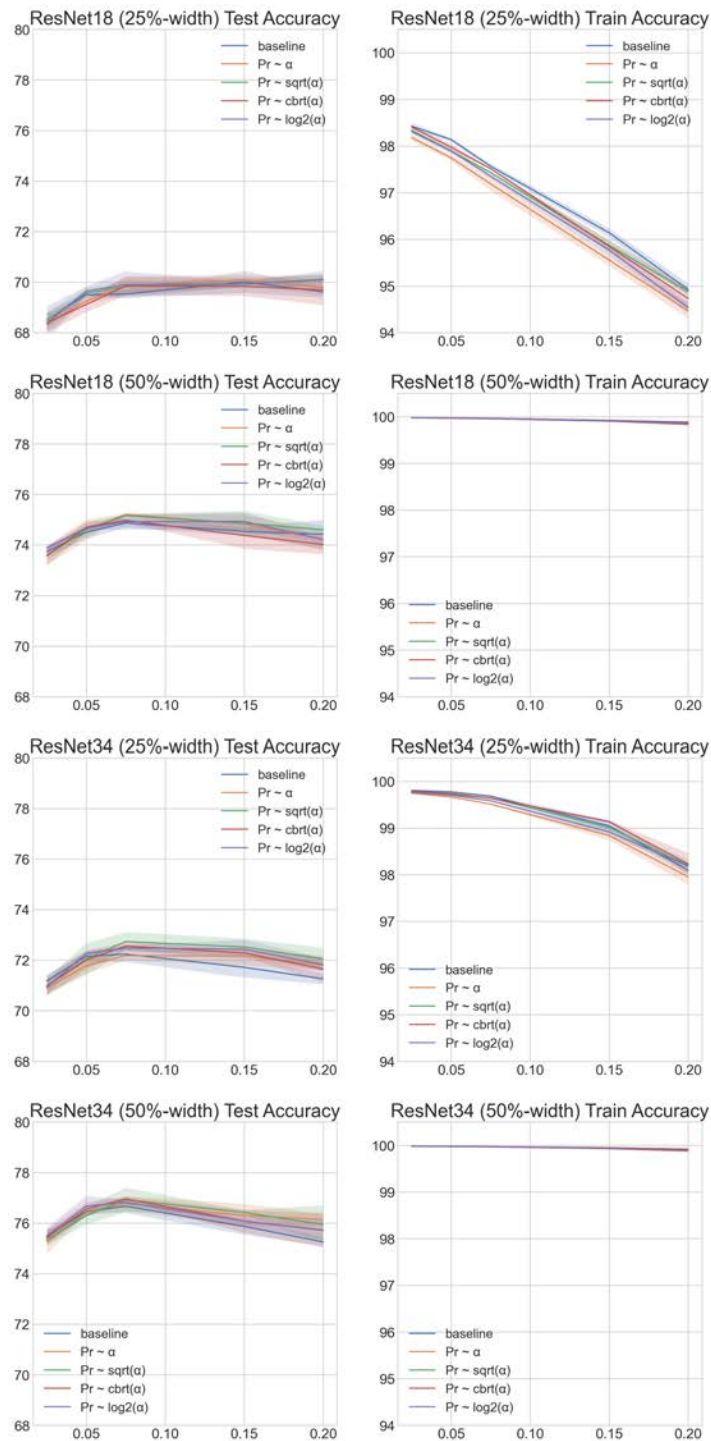


Figure 2.3: Learning rate allocations on ResNet18 (left) and ResNet34 (right) using weighted probability of α values mapped to identity (orange), **square root (green)**, cube root (red), logarithm base-2 (purple).

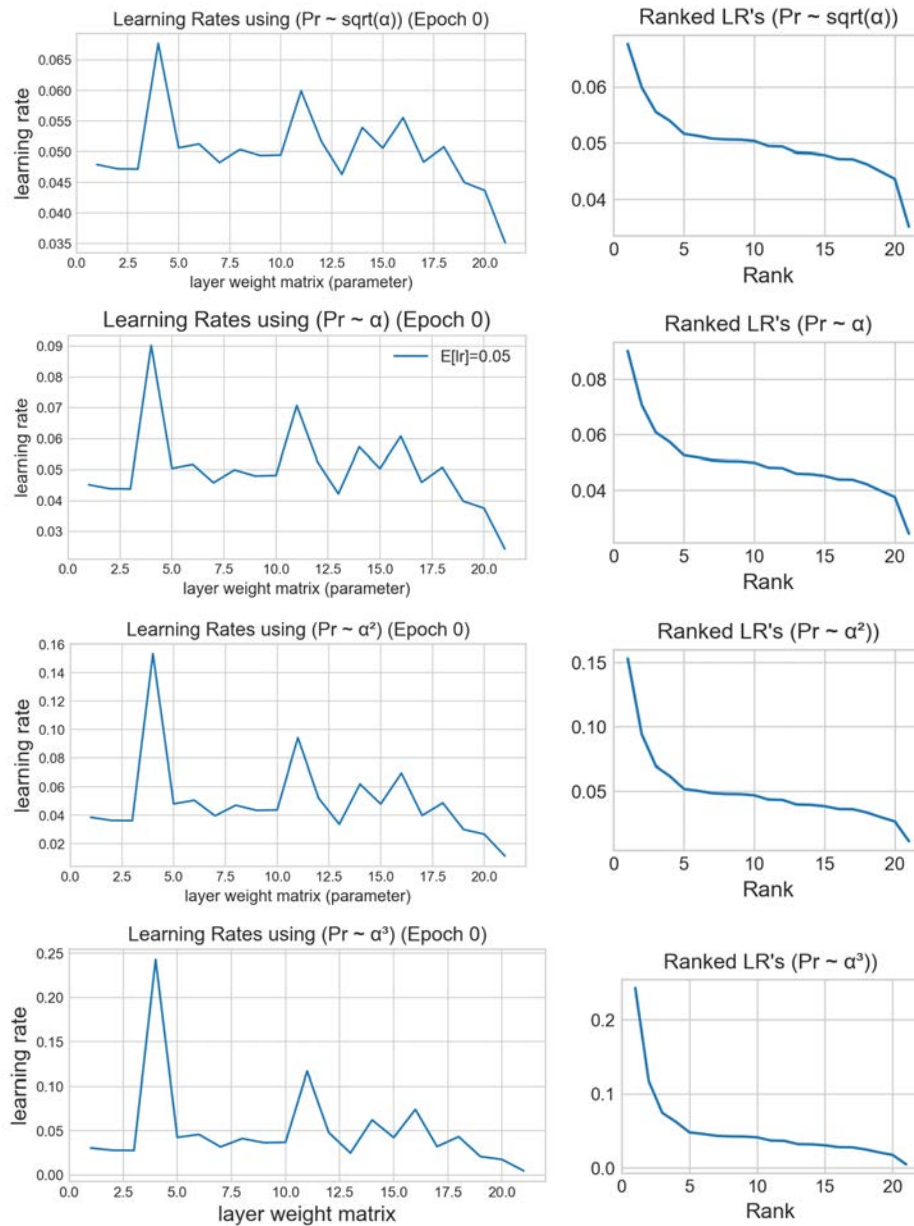


Figure 2.4: Plots of learning rates assigned to parameter groups using each scheme (left) and the learning rates ranked in descending order (right). The mapping function of the α 's before converting them to weighted probabilities is important in determining their scaling relationship. This plot informs our results that we want variations among layerwise learning rates while keeping the majority of them as far from 0 as possible.

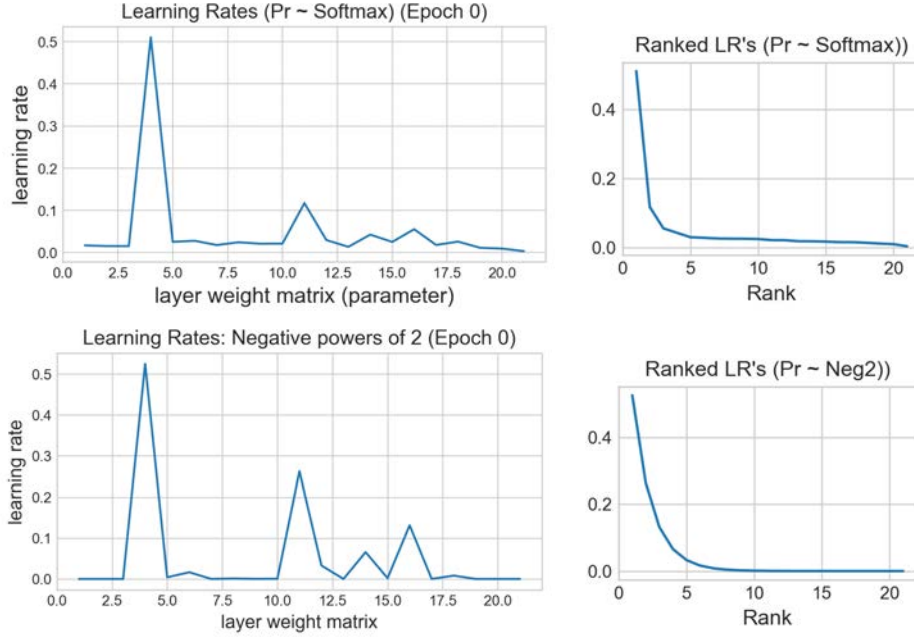


Figure 2.5: Plots of learning rates assigned to parameter groups in order (left) using Softmax weighted probability 2.19 and Negative powers of two 2.3 and the learning rates ranked in descending order (right). Most parameters have learning rates assigned 0, so the network is not able to train to full saturation under such schemes.

Our probabilities are then

$$\text{NegPow}2_n = [2^{-n}, 2^{-n-1}, 2^{-n-2}, \dots, 2^{-1}]. \quad (2.20)$$

As $n \rightarrow \infty$, $\sum_{i=1}^n 2^{-i} \rightarrow 1$. Since n is finite, we add the residue to the largest probability to maintain the ordering,

$$\text{NegPow}2_n[-1] = 2^{-1} + (1 - \text{sum}(\text{NegPow}2_n)). \quad (2.21)$$

Next, we assign $\text{NegPow}2_n[j] \cdot lr_t \cdot n$ to the layer with the j -th smallest α , $j = \sigma(i)$:

$$lr_{t,j} = lr_t \cdot n \cdot \text{NegPow}2_n[j]. \quad (2.22)$$

Similar to the softmax assignment above, because the negative powers of two approach zero very quickly as n grows, many parameter groups are not updated. This results in worse model performance during test time and the model not training to convergence, with 65-70% train accuracies. A future direction would be to increase the near-zero values, such as distributing some fraction of the learning rate over from the layer that was assigned the 2^{-1} -fraction.

2.4 Accounting for Randomness in Observations of Heavy-tailed Metrics

Power-law-fitted α 's are not always informative. From empirical data, we observe some sort of randomness, or noise, involved in our α observations. Controlling α is important as we hypothesize its negative correlation with the training progress of the layer weight matrices. This hypothesis arises from empirical observations, so we have yet to draw a necessary or strong theoretical condition regarding α 's and overall model performance.

Improved test time results sometimes simultaneously arise with less variation or lower values of α across the layer weight matrices through training while other times no improvements are seen. In most cases, we do not observe α converge to the optimal range of $[2, 6]$ after 200 epochs of training as detailed in [67]. For models with worse performance we do not always observe α values larger than 6.

As an attempt to account for randomness with respect to the α 's, we model α as a random variable that we could sample, which could add more “slack” for the otherwise “fixed” power-law fitted values it takes on. Without any given assumptions about the moments of the layerwise α 's, we assume it is normally distributed, i.e. $\alpha \sim \mathcal{N}(0, 1)$. Unknown distributions are traditionally modeled using the (standard) Gaussian, its motivation generally revolving around two types of discussions:

1. **Distributional universality:** The Central Limit Theorem states that for n -large independent and identically distributed samples X_1, \dots, X_n from any *arbitrary* distribution with mean μ and variance σ^2 , their sample average

$$\bar{X}_n = \frac{\sum_{i=1}^n X_i}{n} \quad (2.23)$$

converges to the normal distribution $\mathcal{N}(\mu, \sigma^2/n)$ [7]. Gaussian modeling in this scenario would guarantee “universality,” where the converged distribution remains invariant to its initial distributional assumptions.

2. **Claim of maximum uncertainty:** According to the Principle of Maximum Entropy, we should model unknown random variables with the probability distribution that gives the largest remaining uncertainty, i.e. maximum entropy [39]. This minimizes biases and assumptions that could otherwise be introduced when representing the current state of knowledge. [23] draws an equivalence between the Principle of Maximum Entropy and the conditioning by Bayesian inference, which updates the prior distribution $q(\theta)$ to the Bayesian posterior $q(\theta|x')$, upon observing new data x' . We would replace the prior joint distribution $q(x, \theta)$ with the distribution $p(x, \theta)$ that matches the observed data and minimizes the distributional distance:

$$\text{KL}(p(x, \theta)||q(x, \theta)) = \int \text{KL}(p(\theta|x)||q(\theta|x))p(x)dx + \text{KL}(p(x)||q(x)). \quad (2.24)$$

This reduces to minimizing $\text{KL}(p(\theta|x')||q(\theta|x'))$ by setting

$$\begin{aligned} p(\theta|x') &= q(\theta|x') \\ p(x, \theta) &= p(x)q(\theta|x). \end{aligned} \quad (2.25)$$

The Bayesian posterior $q(\theta|x')$ then captures the distribution of maximum entropy after observing x' [41, 40]. Additionally, maximum entropy distributions most naturally model systems evolving towards thermodynamic equilibrium, by the second law of thermodynamics, i.e. entropy only increases [60, 15, 17].

Sampling from n α_i -mean distributions

Given our limited empirical observations and lack of prior knowledge regarding the α distributions and their associated noise, we will sample from the Normal distribution $\mathcal{N}(0, 1)$ but with mean shifted by each of the layerwise α_i 's, $\mathcal{N}(\alpha_i, 1)$. Thus for each epoch $t \in [T]$, for each layer i , we create n distributions

$$Y_i \sim \mathcal{N}(\alpha_{t,i}, 1). \quad (2.26)$$

We then sample one value y_i from each distribution Y_i . Following our discussion in 2.3, we normalize by the weighted average of the y_i 's to obtain n learning rates every epoch t ,

$$lr_{t,i} = lr_t \cdot n \cdot y_i. \quad (2.27)$$

Sampling Probability Weights from a lr_t -mean distribution

Given our baseline schedule of each epoch having one learning rate lr_t , we can shift the mean of $\mathcal{N}(0, 1)$ by lr_t . For each epoch $t \in [T]$, we create one distribution

$$X_t \sim \mathcal{N}(lr_t, 1). \quad (2.28)$$

We then sample n values of x_i from each distribution X_i for each epoch. Following our discussion in 2.3, we sort the $\alpha_{t,i}$ to create a permutation $x_{t,j}$ of $x_{t,i}$ such that $j = \sigma(i)$. Normalize the $x_{t,j}$ to obtain probabilities $\hat{x}_{t,j}$ so that the layer with the j th largest $\alpha_{t,j}$ gets the j th largest $\hat{x}_{t,j}$,

$$lr_{t,j} = lr_t \cdot n \cdot \hat{x}_{t,j}. \quad (2.29)$$

Modeling Noise of α with Distributional Assumptions

By observing the evolution of the density of layerwise α_i 's through epochs on ResNet18 and ResNet34 models, we see that the density roughly resembles an exponential (or power-law) distribution 2.4. Reminder that α is measure of *power-law*-ness itself.

Following this observation, instead of our original assumptions about the distribution we sample from, $\mathcal{N}(\mu, \sigma^2)$, we can model α as the exponential distribution $Z \sim Exp(\nu)$ with probability density function

$$\begin{aligned} f &= \nu e^{-\nu x}, \\ \mathbf{E}[Z] &= \frac{1}{\nu}, \text{var}(Z) = \frac{1}{\nu^2}. \end{aligned} \quad (2.30)$$

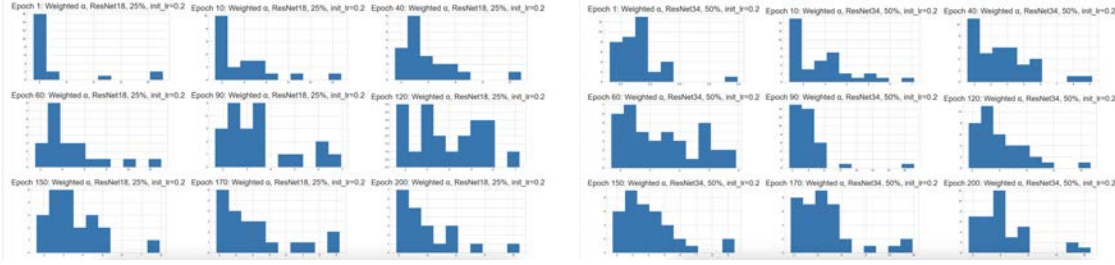


Figure 2.6: Approximate exponential distribution of α values emerge towards the end of training on the ResNet18 (left) and ResNet34 (right) models with 0.25-width fraction, initial learning rate 0.2.

Because our observations are empirically inexhaustive and we work with relatively small sample sizes of n , we can add an error term $\epsilon_\alpha \sim \mathcal{N}(0, 1)$ to model randomness.

Let $A = Z + \epsilon_\alpha$, which is an "exp-norm", or an exponentially modified Gaussian (EMG) distribution common in biological sciences [29, 26]. A is still unimodal and if we convolve the two known distributions, approximately

$$A \sim \mathcal{N}\left(\frac{1}{\nu} + 0, \frac{1}{\nu^2} + 1\right). \quad (2.31)$$

A proof of the convolution can be found in Appendix A. $A = Z + \epsilon_\alpha$ models an Additive white Gaussian noise (AWGN) channel [15] with capacity

$$C = \frac{1}{2} \log_2\left(1 + \frac{\rho}{\sigma^2}\right), \quad (2.32)$$

where $\sigma^2 = 1, \rho = \frac{1}{\nu^2}$ in our setting.

As the test accuracy increases we see a decrease in train accuracy, suggesting the role of regularization with the sampling, whose effects weaken as the network grows deeper. This is revealed through lower test accuracy and higher train accuracy on deeper residual networks (ResNet101, ResNet152) than on shallower ones (ResNet18, ResNet34).

2.5 Future Work

- We currently assume a standard Gaussian distribution for sampling new values of α and the learning rate, where variance $\sigma^2 = 1$. Instead of fixing our variance, we could generalize to optimize over the variance σ^2 and proceed to optimize over the signal-to-noise ratio $\frac{\mu^2}{\sigma^2}$. How do we optimize σ^2 to maximize the channel capacity in 2.4?
- What is a more optimal way to restrict to sampling from non-negative continuous probability distributions?

- How could we effectively incorporate a noise term ϵ_α , and what assumptions about its distribution could we make?
- Is it possible to sample to replace select $k < n$ of the learning rate or α values? How would we choose k , and which k of the n values do we replace?

Chapter 3

Layerwise Structured Pruning of Weight Matrices

3.1 Related Work

Modern day neural networks are frequently over-parameterized. Pruning is a well studied compression technique with the main motivation of minimizing network size in order to reduce training time and resources while maintaining, and in some cases improving, performance by removing redundancies. [32, 50, 69, 52] detail the network pruning problem statement and techniques dating back to the 1980s. [10, 13] provide a literature review and analysis of recently introduced neural network pruning and techniques.

The most immediate method is weight-based pruning, where unimportant connections that fall below a certain threshold of importance are removed [31]. A series of prior work proposes one could find a subnetwork consisting of a subset of weights that achieves the performance of the original network. The Lottery Ticket Hypothesis presents an algorithm to identify subnetworks, i.e. “winning tickets”, that converge to the accuracy of the unpruned model in similar number of iterations [20]. This finding was extended to pre-trained BERT networks, where pre-trained subnetworks are found at initialization instead of after some training time [11]. As an extension, the authors characterize an *invariant* used to predict the error of all members of a network family at all dataset sizes and all pruning densities; this shows that the error of iteratively magnitude-pruned networks has a structural behavior that could be captured in a simple functional form and parameters [79]. They also adopt an ensembling approach of combining “sibling” network copies to produce a subnetwork [68].

[75] connects pruning random ReLU networks to the SUBSETSUM problem, showing that any target network of width d and depth l can be approximated by pruning a random network that is $O(\log(dl))$ wider and twice as deep. [76] proposes an algorithm to find “un-trained subnetworks”; for example, they find a subnetwork in WideResNet-50 that matches the performance of a ResNet34 on ImageNet.

[58] encourages a different perspective to identify an optimal pruned architecture rather than simply a set of weights. Regularization is often used to ensure the optimal tradeoff between network accuracy and pruning ratio. [57] prunes insignificant channels to get thin

and compact models and lower the number of operations.

However, prior work has found that unstructured pruning, which produces large-sparse models, outperforms their small, lightly compressed counterparts resulting from structured pruning [96], and [56] extends this study to Transformer models.

A middle ground of the two levels of granularity between sparse and structured pruning has been proposed through a series of work with block sparsity [71, 16, 53, 62, 88]. [21] studies the effect of block sparsity on accuracy with varying patterns and granularity.

Prior Work

The work in this chapter is an extension on the Deep Network Pruning project by Alex Zhao (axyzhao@berkeley.edu) during his time as an undergraduate at Berkeley, where he implemented both `AugPrune` and `ME-Prune` algorithms. We omit the algorithmic details in this thesis but will give a high-level description for both in the following sections.

The AugPrune Method

The `AugPrune` algorithm follows the traditional three-step pipeline in the pruning landscape, train-prune-finetune. `AugPrune` performs structured pruning under the criterion of keeping its top channels ranked by their importance score.

Importance Score The importance score is calculated for each channel z given an augmentation scheme $a(x'|x)$, which produces transformed sample x' from given sample x . Its intuition is to prune features whose presence (i) have little impact on the network output and (ii) largely increase changes in the network’s response to changes in the input as modeled by $a(x'|x)$.

How to select pruning ratio k We prune k -fraction of the network channels. A natural question arises, what is a good k value to set? Popular pruning algorithms set a global k -fraction for the network, but we consider from a layerwise view as well. As a progression of `AugPrune`, we determine layerwise k_i by normalizing all matrix entropy measurements in an extended method `ME-Prune`.

The ME-Prune Method

The `ME-Prune` Method is based on the algorithm of HYDRA [84], which incorporates network pruning with robustness considerations. The method assigns importance scores to weights defined as:

$$s_i^{(0)} = \sqrt{\frac{6}{\text{fan-in}_i}} \times \frac{1}{\max(|\theta_{\text{pretrain},i}|)} \times \theta_{\text{pretrain},i}. \quad (3.1)$$

Unlike `AugPrune`, the importance scores are simply scaled values of the original weights, but we largely follow the same three-stage pipeline. For pruning, we use the matrix entropy measurements across all layers to determine the layerwise k_i -fractions.

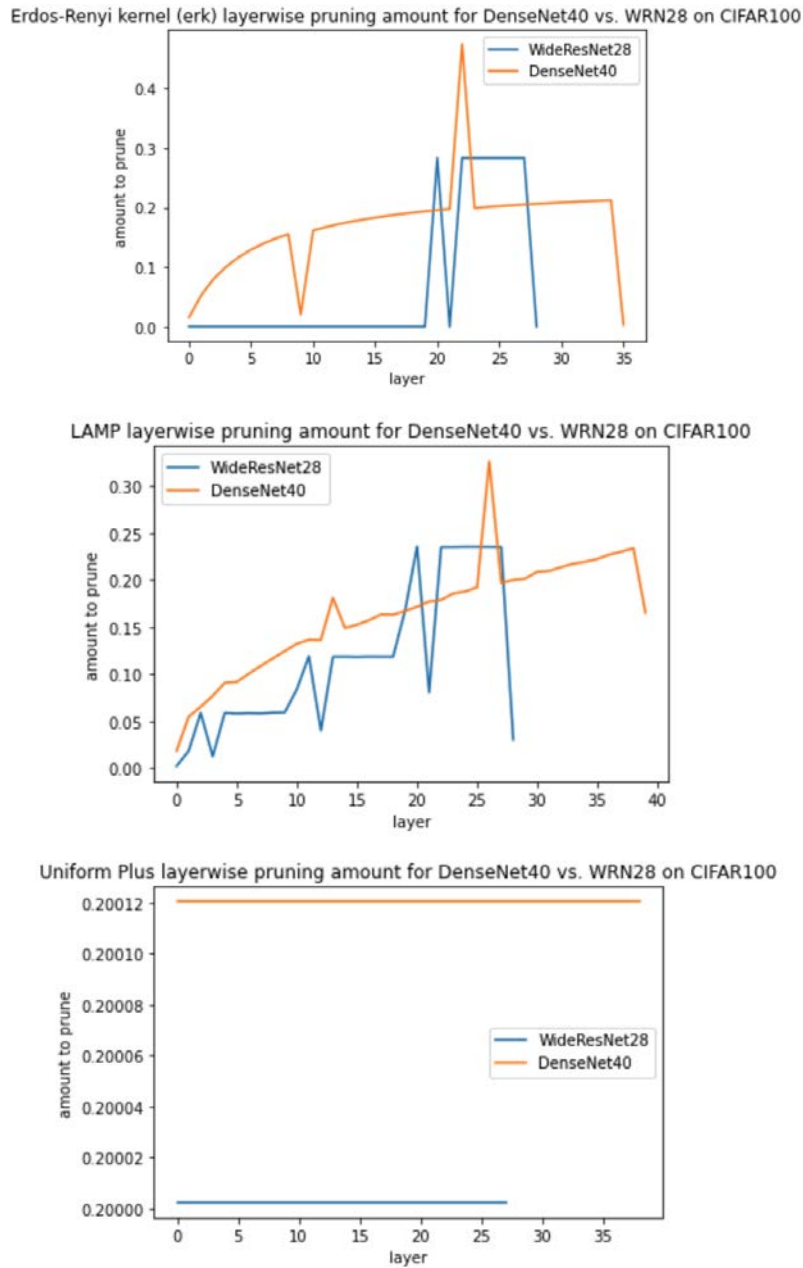


Figure 3.1: Comparing layerwise pruning ratios achieved on WideResNet28 (blue) and DenseNet40 (orange) using Erdos-Renyi Kernel layerwise [18], Layer-adaptive Sparsity for the Magnitude-based Pruning (LAMP) [51], and uniform (baseline) schemes.

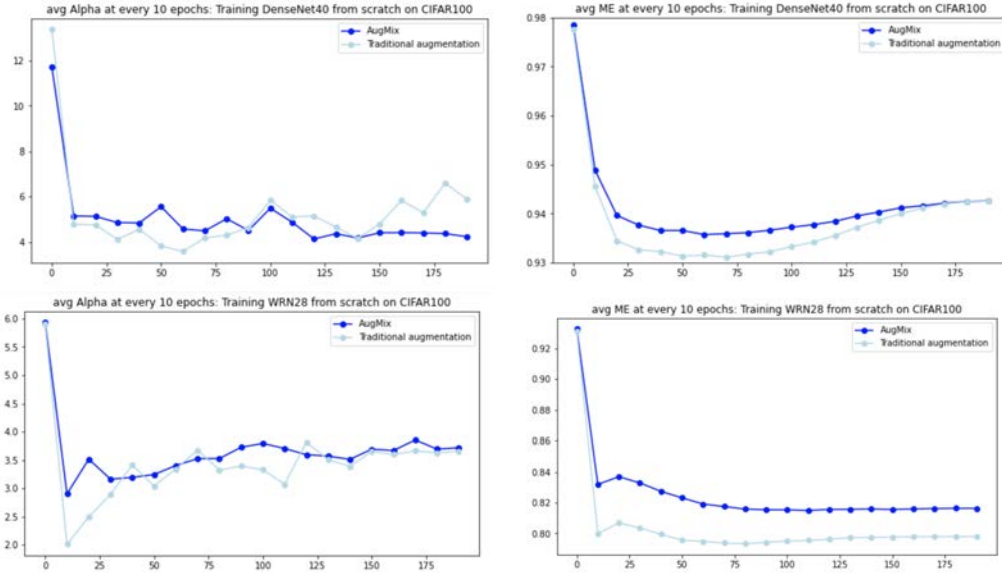


Figure 3.2: **DenseNet40** (top) and **WRN28** (bottom): Effects of AugMix (blue) and traditional augmentation (light blue) on α (left) and matrix entropy (right) through initial training, before pruning.

Given matrix \mathbf{W} , its matrix entropy follows the definition from [64]:

$$\mathcal{S}(\mathbf{W}) = \frac{-1}{\log(R(\mathbf{W}))} \sum_i p_i \log p_i, \tag{3.2}$$

where $p_i = \nu_i^2 / \sum_i \nu_i^2$, ν_i is the i^{th} singular value of \mathbf{W} , and $R(\mathbf{W})$ refers to the rank of \mathbf{W} .

The intuition is that matrix entropy measures randomness observed in the spectral distribution of the layer weight matrix correlations. More formally, the matrix entropy also captures information about the network structure and its usage is considered a form of spectral regularization, so it falls under the same category of a shape metric as α introduced in 1.7. Various recent empirical studies have shown spectral regularization improves generalization power in sparse network and data regimes [1, 30, 54], with [2] explaining when and how regularizing the spectral representation improves generalization.

Effects of Augmentation during Initial Training

We track two shape-based HT-SR metrics, powerlaw coefficient α and matrix entropy during the initial training stage on DenseNet40 and WRN28 to compare the effects of AugMix and traditional augmentation 3.1. Using AugMix results in higher matrix entropy uniformly through time, but mitigates the observed *entropy rebound* phenomenon on DenseNet40, compared to traditional augmentation. Using AugMix also levels the rebound of α on DenseNet40 during training.

3.2 Comparing Metrics for Assigning Pruning Ratios

In this section, we observe the effects of using different shape and scale measurements of layer weight matrix ESDs to inform pruning ratios. In addition to the matrix entropy metric, we compare against baseline (“even” pruning) the effects of using various HT-SR-motivated metrics that characterize heavy-tailness of layer ESDs:

- *Shape* metrics: alpha, random distance, stable rank, matrix entropy
- *Scale* metrics: (log)-norm, log Frobenius norm, (log)-spectral norm λ_{max}
- *Hybrid* metrics: weighted alpha $\hat{\alpha}$, log-alpha norm

Random Distance measures the distance, captured by the Jensen-Shannon divergence, between the eigenvalue distribution of the layer weight matrix \mathbf{W} and the eigenvalue distribution of a randomly generated weight matrix \mathbf{W}_{rand} of the same dimension: $\frac{1}{n} \sum_{i=1}^n JS(p_i, p_{\text{rand},i})$.

Jensen-Shannon divergence (JS) on two discrete distributions p_1 and p_2 is defined as the following,

$$JS(p_1, p_2) = \frac{1}{2}(\text{KL}(p_1 || \bar{p}) + \text{KL}(p_2 || \bar{p})),$$

where $\bar{p} = \frac{1}{2}(p_1 + p_2)$ (3.3)

and $\text{KL}(p || q) = \sum_{x \in X} p(x) \log \frac{p(x)}{q(x)}$ denotes the Kullback–Leibler divergence [60].

Stable Rank, the ratio of the Frobenius norm to Spectral norm, is defined as in [64]:

$$\mathcal{R}_s(\mathbf{W}) = \frac{\|\mathbf{W}\|_F^2}{\|\mathbf{W}\|_2^2} = \frac{\sum_i \nu_i^2}{\nu_{\max}^2} = \frac{\sum_i \lambda_i}{\lambda_{\max}}. \quad (3.4)$$

The $\hat{\alpha}$ metric combines α (shape) and λ_{max} (scale) metrics: $\hat{\alpha} = \sum \alpha_i \log \lambda_{max,i}$.

Controlling Number of Parameters in Channel Pruning

The majority of pruning literature is concerned with the number of parameters in models, but as each channel is associated with weights of different sizes, channel pruning compromises the consideration of pruning ratios at the lowest granularity level. Without extra handling, determining pruning fractions k_i based entirely on matrix entropy fails to give us adequate control over the overall model size in practice.

To address this problem, we define k_i as the pruning budget, serving as a stopping condition for every layer. Instead of directly pruning k_i -fraction of the channels for each layer i , we increment the pruning ratio by 0.01 per iteration. We stop pruning a layer when its pruning ratio has reached the maximum k_i -fraction. In this approach, we would rather under-prune than over-prune. This added functionality is important as we could now compare the effects of layerwise learning rate assignment across custom pruning ratios on the same model.

We have four analysis cases, divided by augmentation scheme and relationship between pruning ratios and HT-SR metrics, as shown in table 3.1.

Augmentation Scheme	Layerwise Pruning Ratios \sim Metric	
	Directly Proportional	Inversely Proportional
Traditional	1	2
AugMix	3	4

Table 3.1: Resulting figures for this section follow the sequence enumerated in this table. Settings 1 and 3 prune more for layers with larger HT-SR metrics. Settings 2 and 4 prune more for layers with smaller HT-SR metrics.

Experimental Setup

Architectures We use the DenseNet40, MobileNetV2, and WideResNet28 models.

DenseNet40 is introduced in [38], where feature maps of each layer are used as inputs into all subsequent layers. According to the authors, advantages of DenseNets include ameliorating the vanishing-gradient problem, encouraging feature reuse, reducing number of parameters, and more. ResNets follow the similar idea of having high connectivity but combine features summation while DenseNets concatenate them.

For an L -layer DenseNet, it has $\frac{L(L+1)}{2}$ instead of L connections for an L -layer network, hence its name DenseNet (Dense Convolutional Network). Specifically for DenseNet40, it consists of 1 3×3 convolutional layer, 3 dense blocks with 12 layers each, 2 transportation layers between the dense blocks, ending with the final classification layer.

MobileNetV2 requires fewer number of operations and memory while maintaining accuracy and is designed for mobile and resource constrained settings [81]. It consists of two types of blocks, a residual block of stride 1 and a downsizing block of stride 2. Each block consists of three layers: 1×1 convolution followed by a ReLU6 activation, a depth-wise convolution, then a 1×1 convolution without non-linearity, i.e. no ReLU [37]. The accuracy improves with the removal of the ReLU at the output of the so-called "bottleneck operator" module.

WideResNet28-10 (WRN28-10) is a widened ResNet architecture motivated by increasing representational power of residual blocks [95]. The network consists of a convolutional layer, 3 groups of residual blocks, and average pooling and final classification. The authors introduce deepening factor l and widening factor k , where l is the number of convolutions in a block and k scales the number of features in convolutional layers; here we have $k = 10$.

We use the same CIFAR-100 **dataset** as we did in the above experiment.

The motivation behind **data augmentations** is to help the network predict more accurately by providing more variations of the original examples to add to the training data. The original methods AugPrune and ME-Prune found AugMix to be the best among different augmentation schemes, including traditional augmentations (random flipping and random cropping) and ImgAug [36, 44].

Results

On DenseNet40 with AugMix data augmentation, post-pruning clean accuracy outperforms the baseline especially in sparse settings when setting larger pruning fraction k_i for layers with larger shape metrics and smaller pruning fraction k_i for layers with smaller scale metrics 3.2.

Similar trends arise when using traditional augmentation, but the improvement from baseline is less significant.

On MobileNetV2, most metric-based pruning does not achieve gains similar to those on DenseNet40.

On WRN28, uniform pruning is the most optimal option, suggesting the power architecture of the residual blocks with increased depth and width.

Implicit Special Handling of Bottleneck Layers

From observing the layerwise pruning ratios determined by the HT-SR metrics on DenseNet40, two layers have notably different ratios when using select metrics 3.2.

We zoom in on entropy (namely, matrix entropy), which yields lower pruning ratios in Settings 1 and 3 and higher pruning ratios in Settings 2 and 4; and norm and spectral norm, for which the inverse is true. These two layers correspond to the two *Transition Layers* on DenseNet40 between the dense blocks.

This observation provides an explanation for the results in the previous section: Methods outperform uniform pruning when they implicitly identify the *Transition Layers*, for which they prune less. *Transition Layers* have observably lower matrix rank, so it would make sense that pruning less for layers starting with low number of parameters correlated with improved model performance, as a result of better forward propagation through the layers.

This type of implicit layer differentiation is not observed on WideResNet28 and MobileNetV2, which correlate with smaller performance gains from the baseline uniform pruning in various settings. Future work is needed in the direction of developing algorithms that could determine such trends and structural differentiation for any architectural backbone.

3.3 Layerwise Learning Rates with Uniform Pruning

To analyze the effects of HT-SR-motivated layerwise allocation of learning rates separately from that of pruning ratios, we perform uniform pruning of channels across all layers. We now prune k -fraction of all channels for every layer before finetuning with layer-specific learning rates determined by the algorithm in 2.3. Note that different layers could still end up with different number of channels.

Experimental Setup

We use the same CIFAR-100 **dataset** as we did in the above experiment.

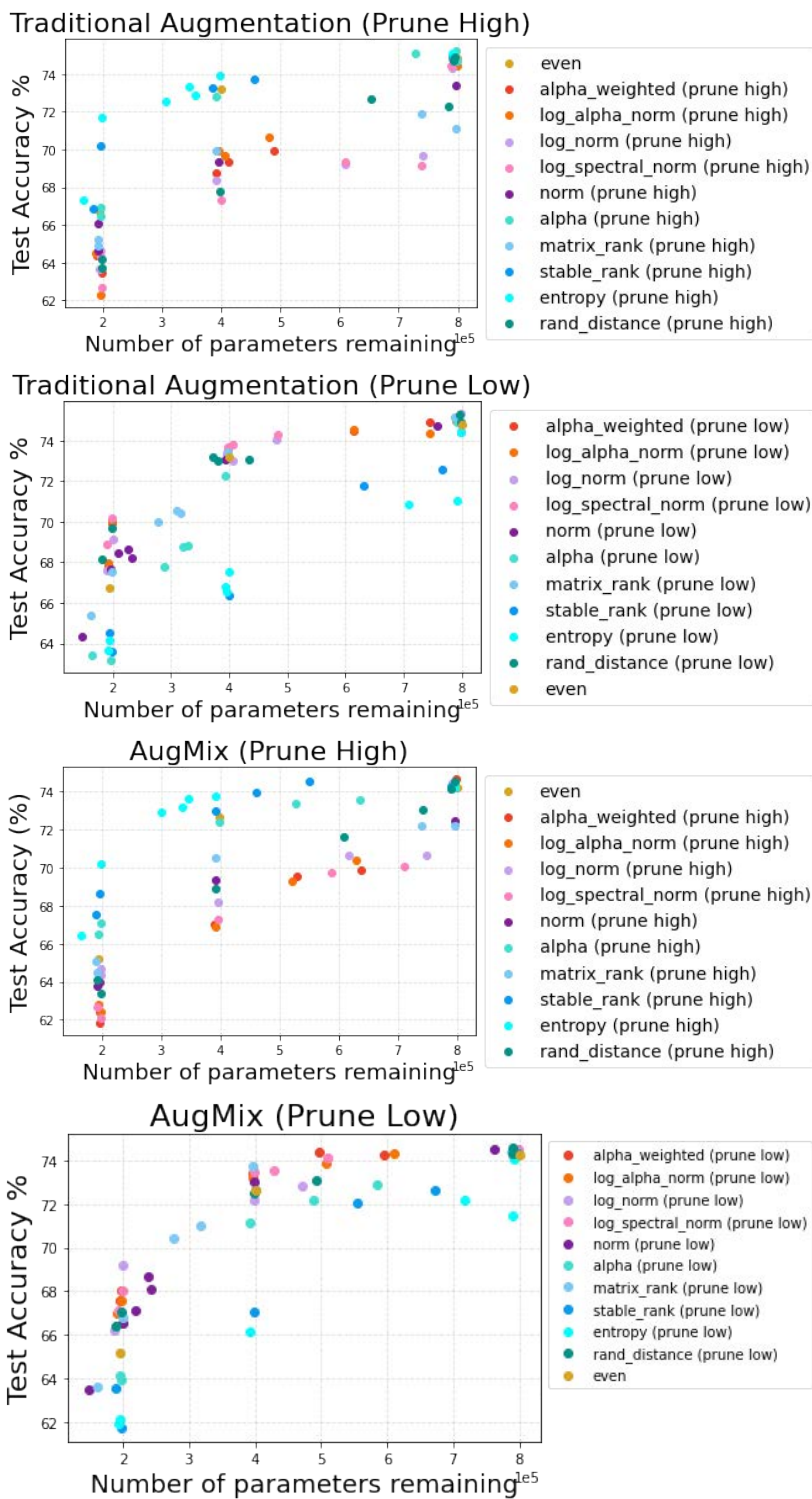


Figure 3.3: Clean test accuracy vs. number of parameters remaining on **DenseNet40**: Using Different Shape (blue-green colors) and Scale (red-pink colors) Metrics vs. uniform (“even”) pruning (yellow).

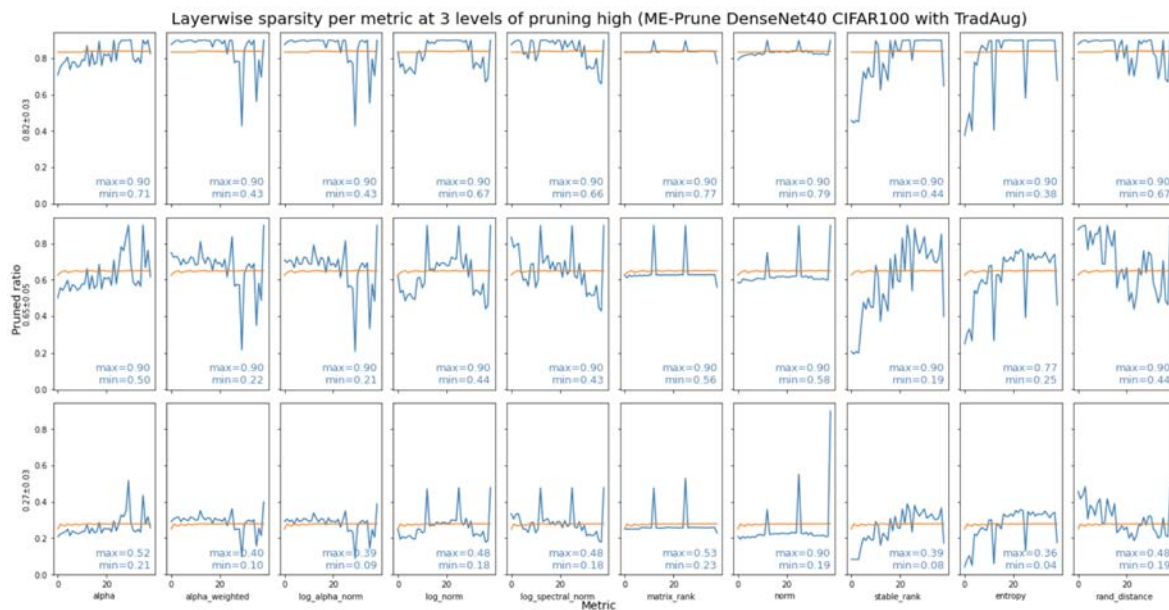


Figure 3.4: Prune more for layers with larger metrics.

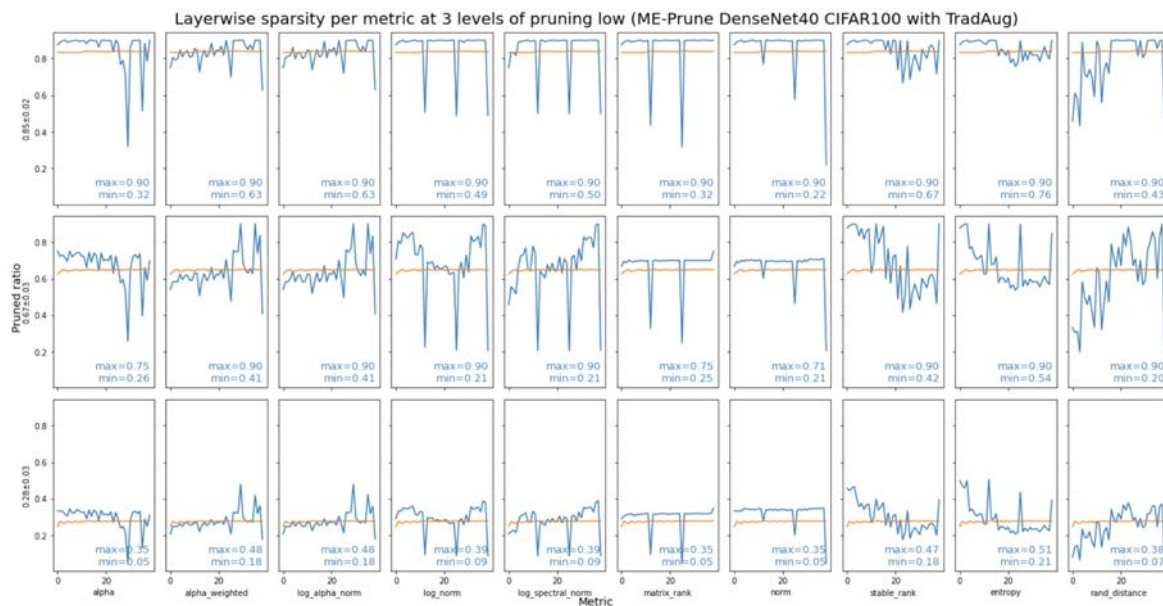


Figure 3.5: Prune more for layers with smaller metrics.

Figure 3.6: Layerwise pruning ratios with **traditional** augmentation on **DenseNet40**. Columns represent the different 10 HT-SR metrics. Uniform pruning would show a straight line across the graph. From bottom to top row, in increasing order of model pruning ratio.

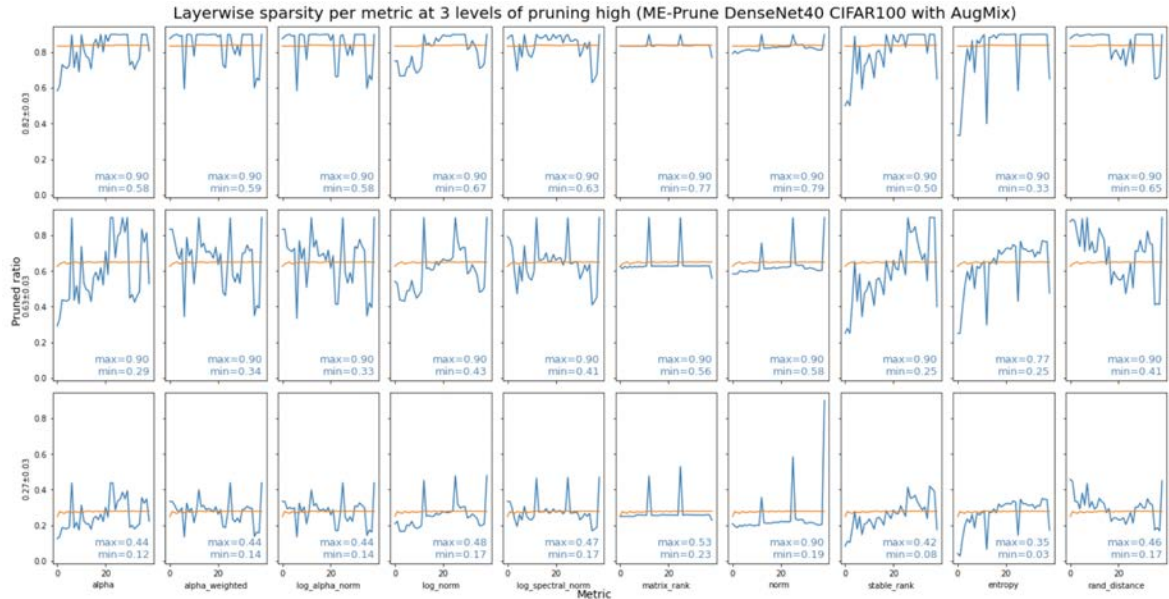


Figure 3.7: Prune more for layers with larger metrics.

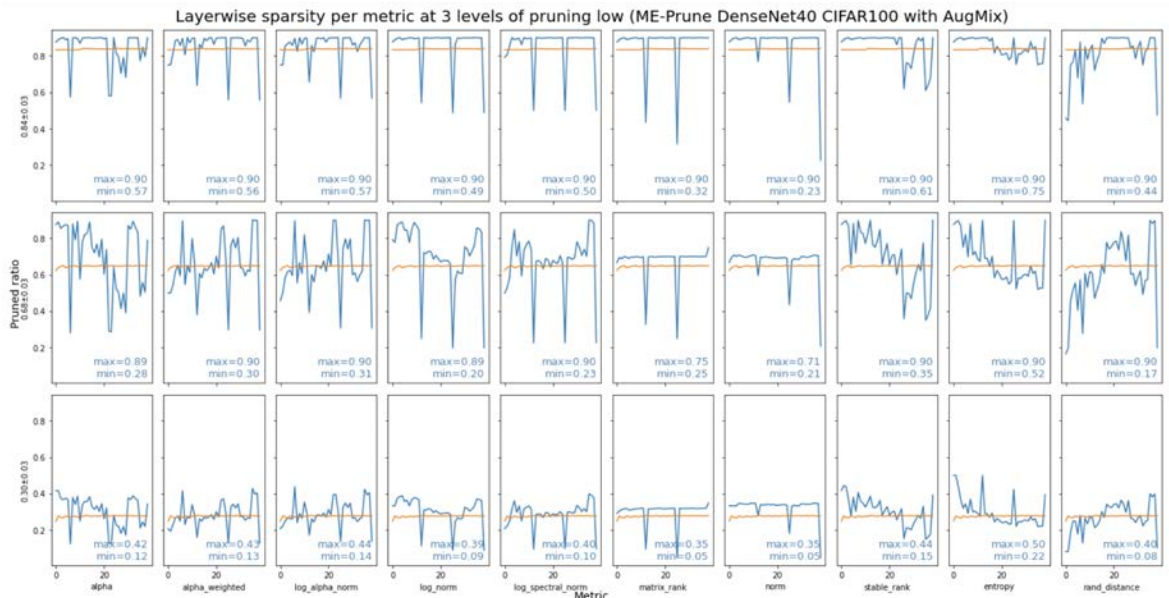


Figure 3.8: Prune more for layers with smaller metrics.

Figure 3.9: Layerwise pruning ratios with AugMix on DenseNet40. Columns represent the different 10 HT-SR metrics. Uniform pruning would show a straight line across the graph. From bottom to top row, in increasing order of model pruning ratio.

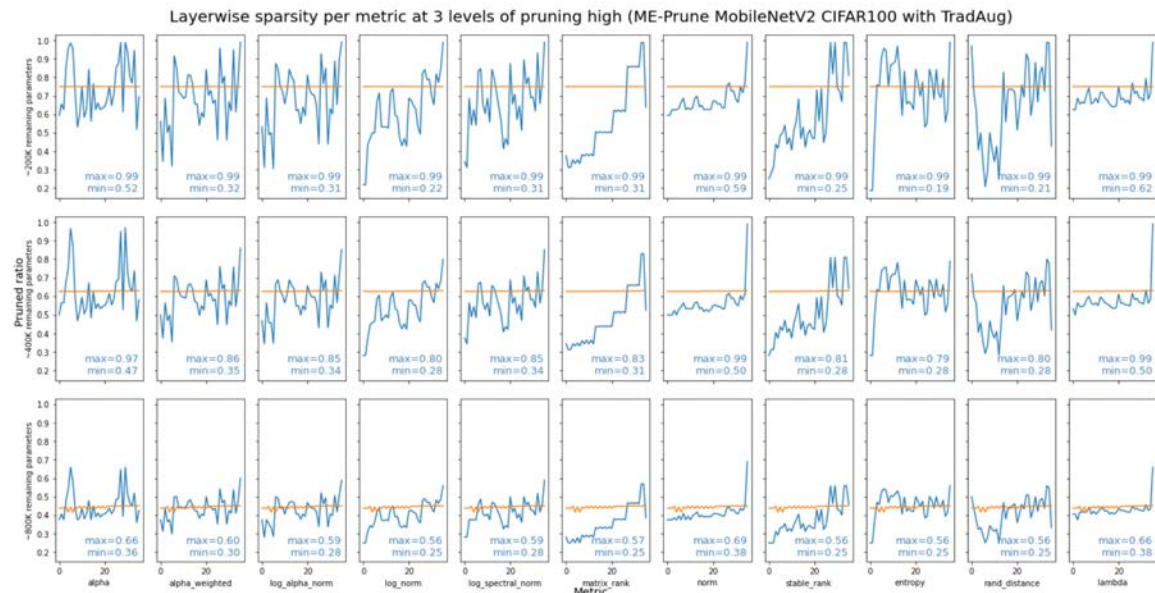


Figure 3.10: Prune more for layers with larger metrics.

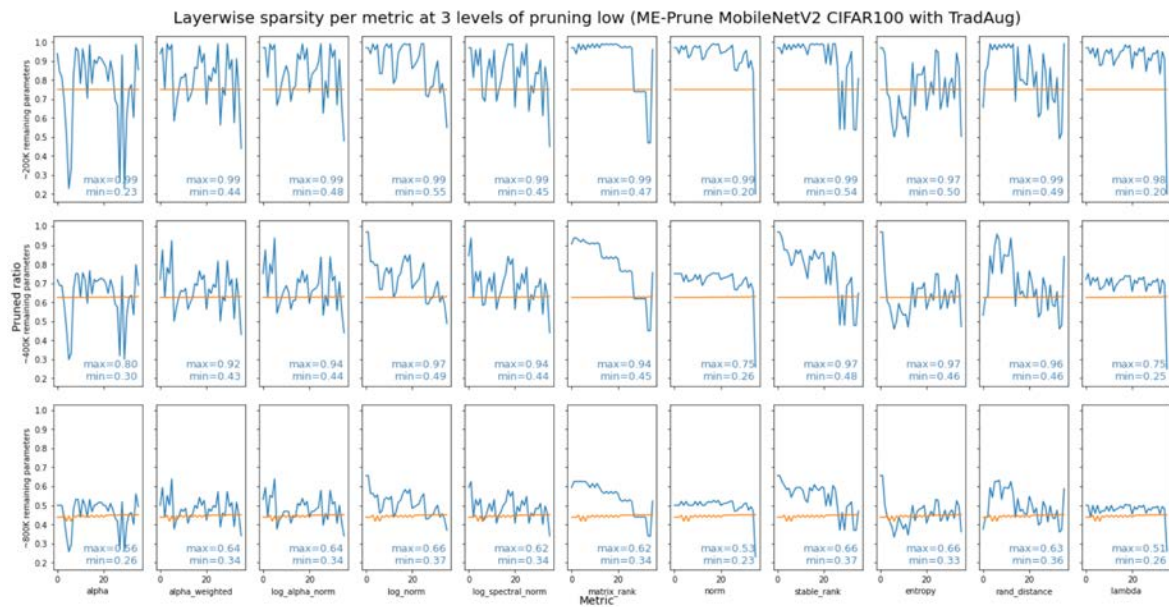


Figure 3.11: Prune more for layers with smaller metrics.

Figure 3.12: Layerwise pruning ratios with **traditional** augmentation on **MobileNetV2**. Columns represent the different 10 HT-SR metrics. Uniform pruning would show a straight line across the graph. From bottom to top row, in increasing order of model pruning ratio.

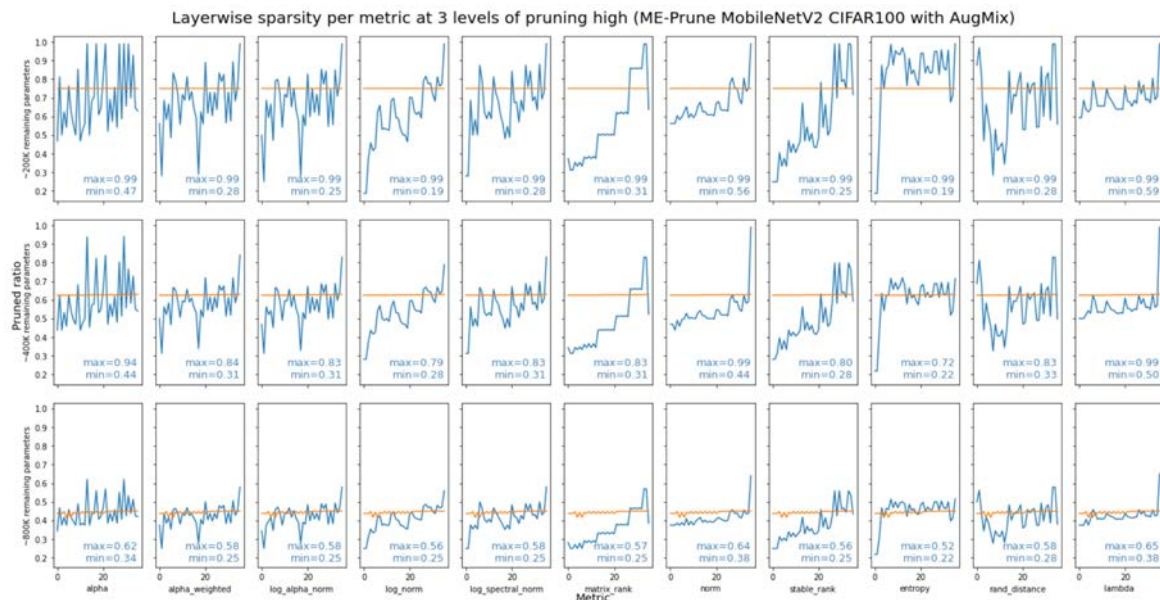


Figure 3.13: Prune more for layers with larger metrics.

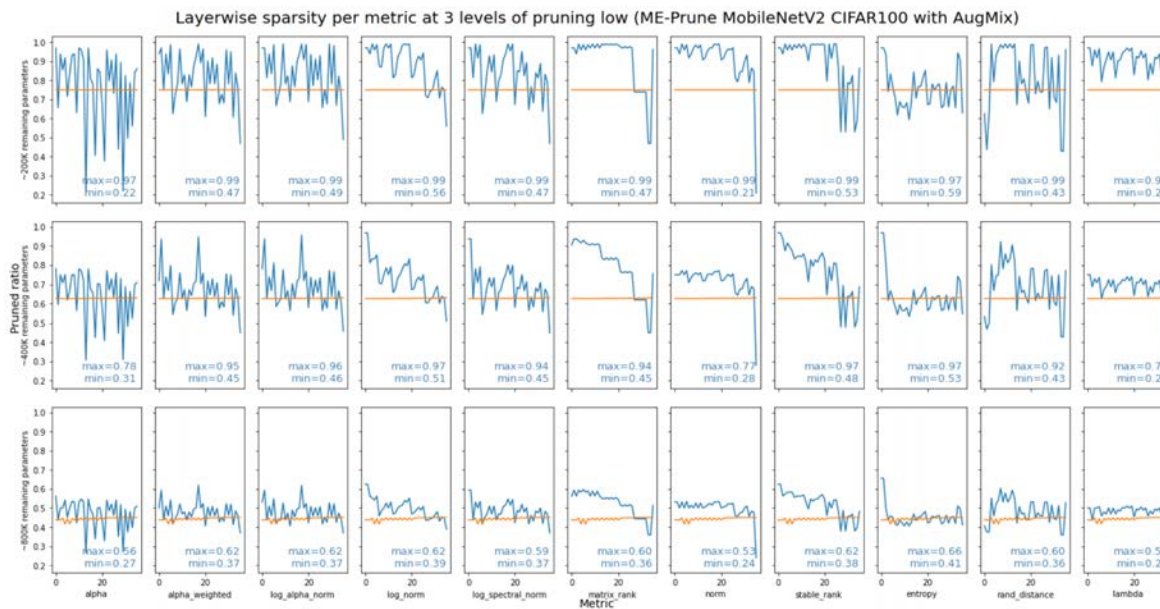


Figure 3.14: Prune more for layers with smaller metrics.

Figure 3.15: Layerwise pruning ratios with **AugMix** on **MobileNetV2**. Columns represent the different 10 HT-SR metrics. Uniform pruning would show a straight line across the graph. From bottom to top row, in increasing order of model pruning ratio.

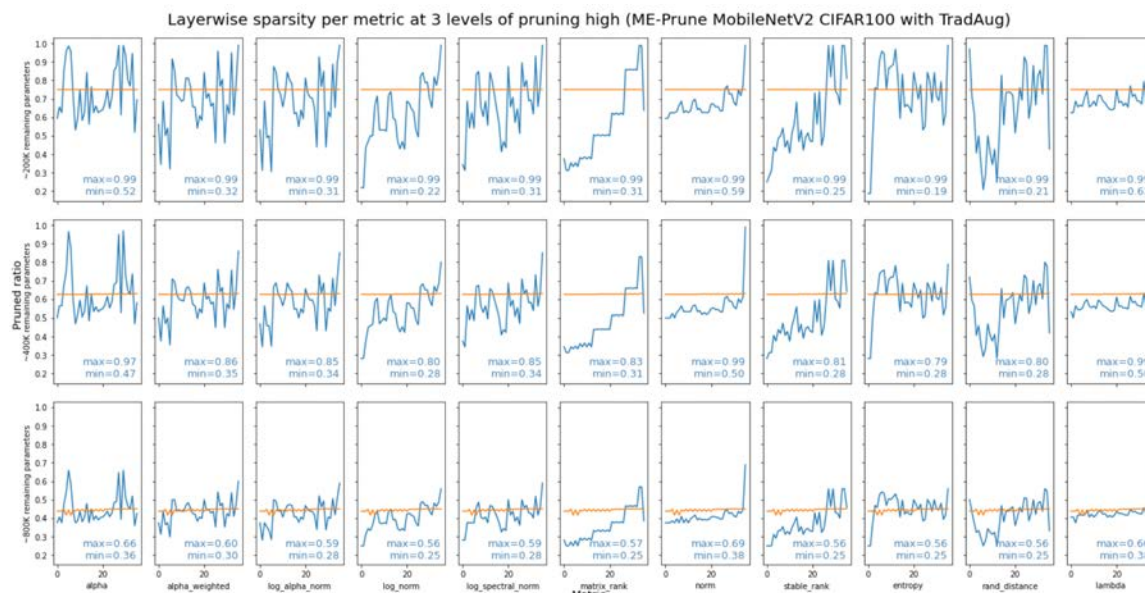


Figure 3.16: Prune more for layers with larger metrics.

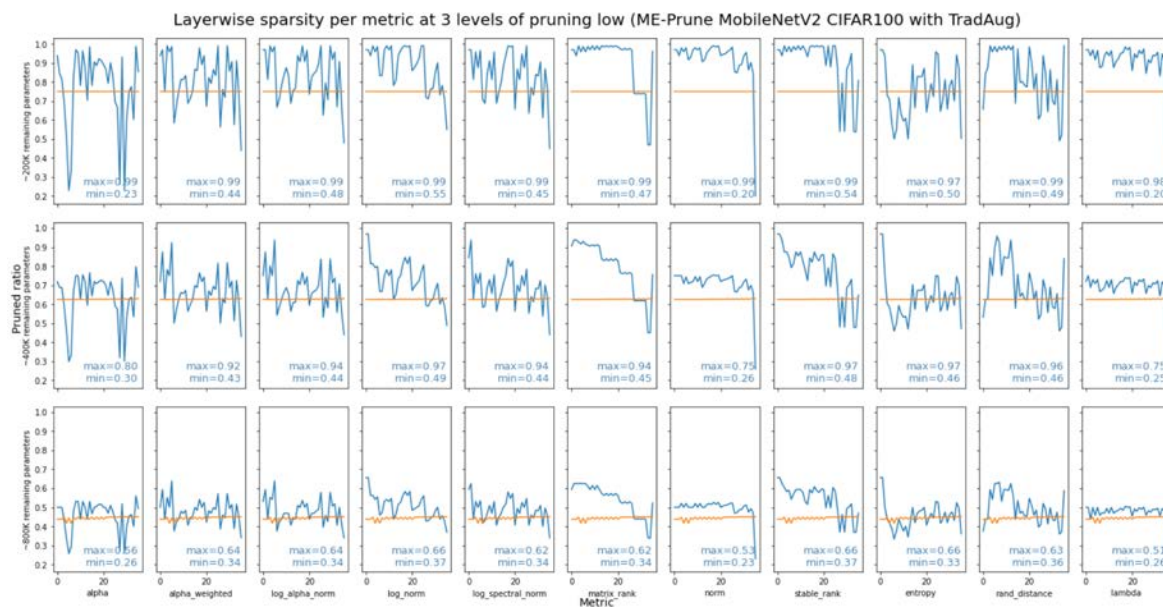


Figure 3.17: Prune more for layers with smaller metrics.

Figure 3.18: Layerwise pruning ratios with **traditional** augmentation on **WideResNet28**. Columns represent the different 10 HT-SR metrics. Uniform pruning would show a straight line across the graph. From bottom to top row, in increasing order of model pruning ratio.

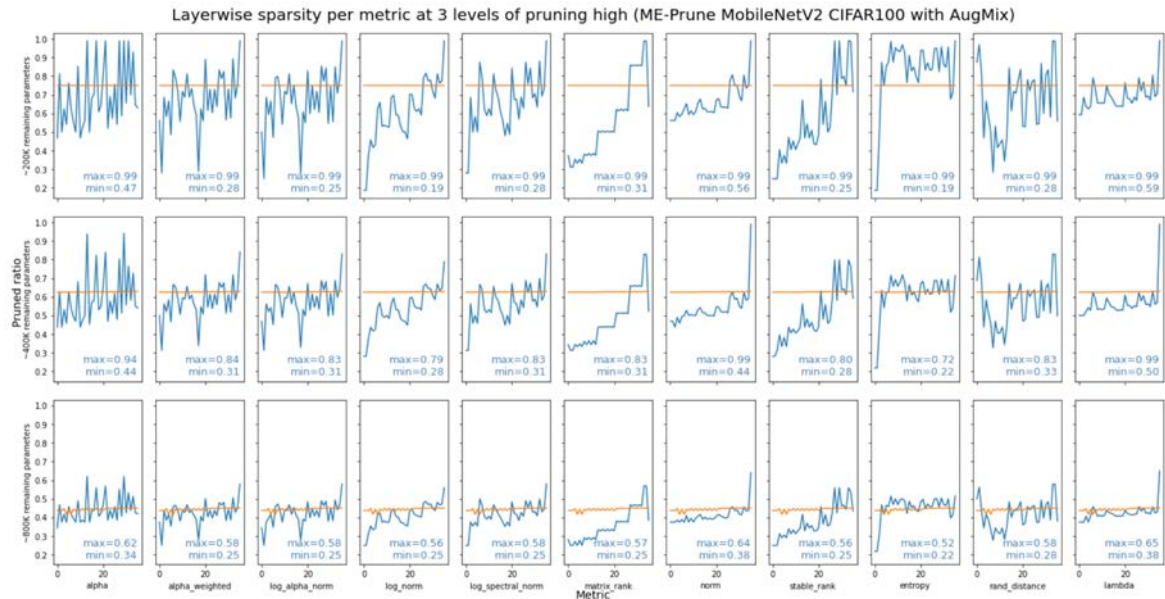


Figure 3.19: Prune more for layers with larger metrics.

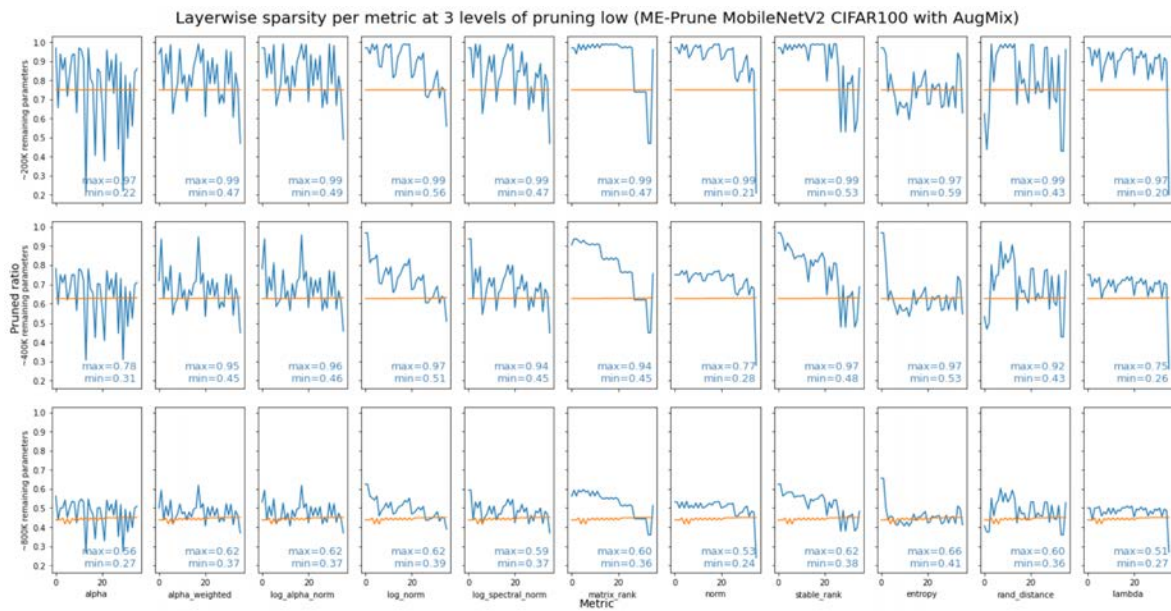


Figure 3.20: Prune more for layers with smaller metrics.

Figure 3.21: Layerwise pruning ratios with **AugMix** on **WideResNet28**. Columns represent the different 10 HT-SR metrics. Uniform pruning would show a straight line across the graph. From bottom to top row, in increasing order of model pruning ratio.

We use the DenseNet40 and VGG19 **architectures**. Each layer of DenseNet40 connects with every other layer, resulting in $O(L^2)$ connections for L layers. VGG19, on the other hand, is considered to be more vanilla and has no cross or skip connections.

Our baseline learning rate adopts the cosine annealing schedule as before, and we mainly consider an initial learning rate value of 0.05. A more comprehensive study would include a wide range of learning range as in previous experiments.

Results

On the VGG19 architecture, layerwise learning rate allocation reaches baseline performance in the sparsest and densest settings without underperforming the baseline training accuracy. For models with 10 to 80 percent of parameters remaining, the layerwise learning rate method consistently outperforms the baseline.

On DenseNet40, the observable optimal range of fraction of remaining parameters is between 55 to 65 percent 3.3. The architectural structures of VGG19 and DenseNet40 could explain the performance differences: as DenseNet40 already accounts for built-in layerwise interactions through skip links, its performance gain using layerwise learning rate allocation is less significant than on the more vanilla VGG19.

3.4 Future Work

A natural direction would be to combine HT-SR-motivated layerwise allocations for *both* pruning ratios and learning rates on the same model. We would most likely need to take into consideration coupling effects as networks could respond differently due to many factors including size, architectural features, dataset, etc.

We show preliminary results on all four enumerated settings on the pruned DenseNet40 architecture: uniform pruning and global learning rate (baseline), uniform pruning and layerwise learning rates determined by α , layerwise pruning ratios determined by α and global learning rate (α prune), and *both* layerwise pruning ratios and layerwise learning rates determined by α 3.4. We could try this layerwise finetuning approach on multiple pruning algorithms, and explore different parameter settings in the initial training stage.

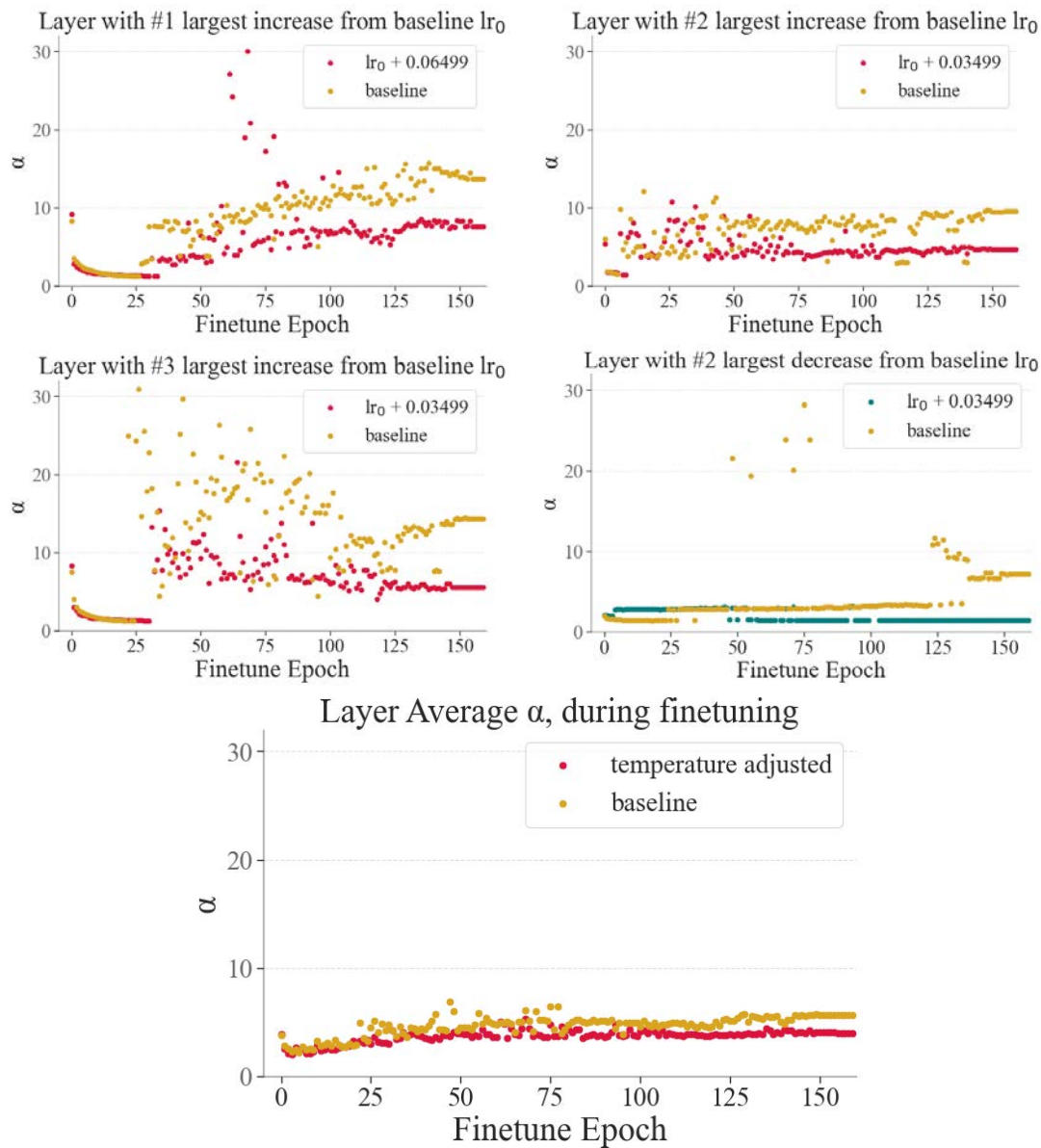
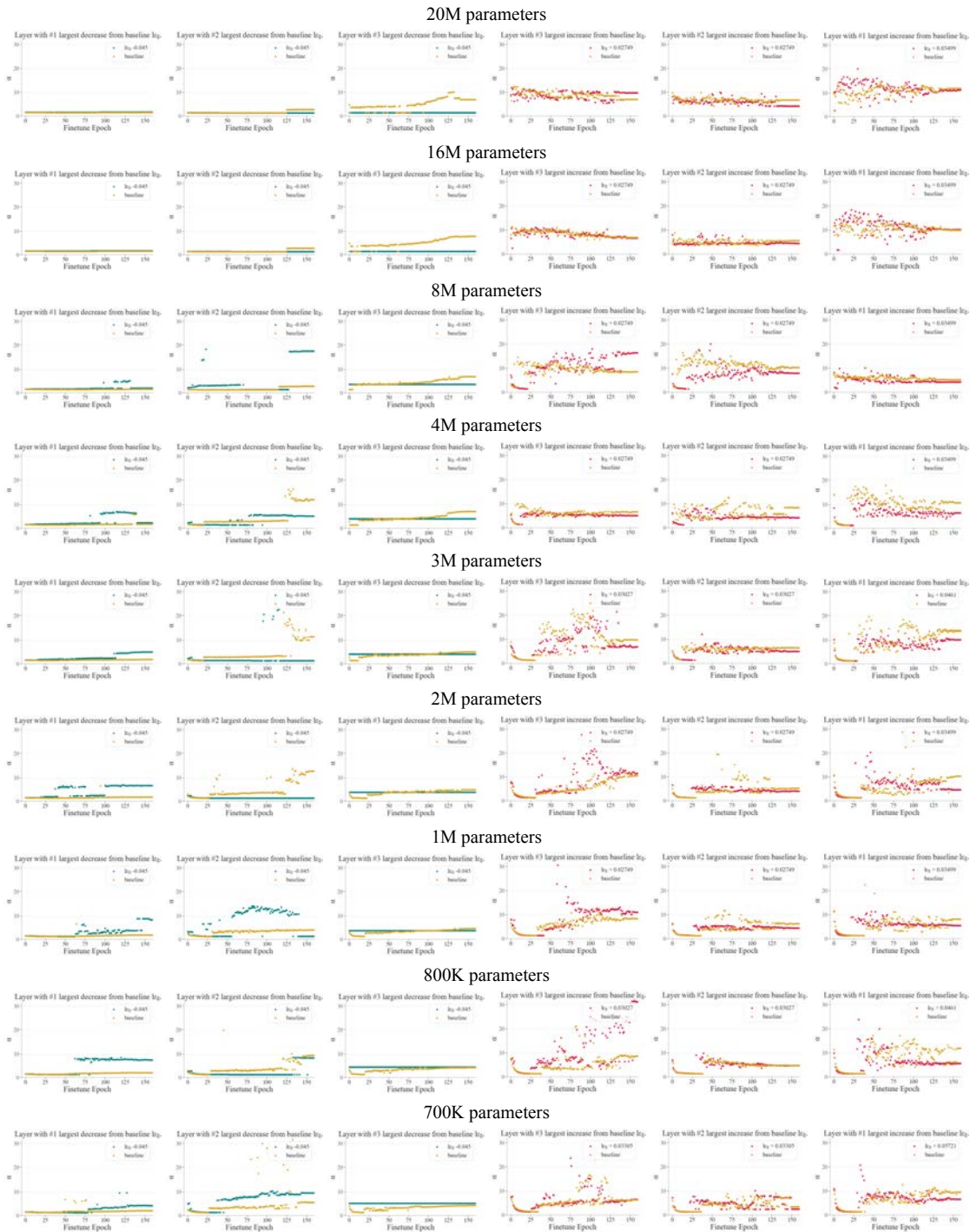


Figure 3.22: Zooming into select figures of layer α from VGG19 with 4M parameters 3.3. Lower overall average α values across all layers are recorded through the finetuning epochs compared to baseline, indicating overall better training completeness. The top four figures show that layers assigned learning rates with the largest difference from the baseline also have smaller α values through finetuning, showing better model convergence.

Figure 3.23: VGG19 at 11 Sizes: Change of α through finetuning for 3 layers with the largest increase and 3 layers with largest decrease from initial baseline learning rates (continued on next page).



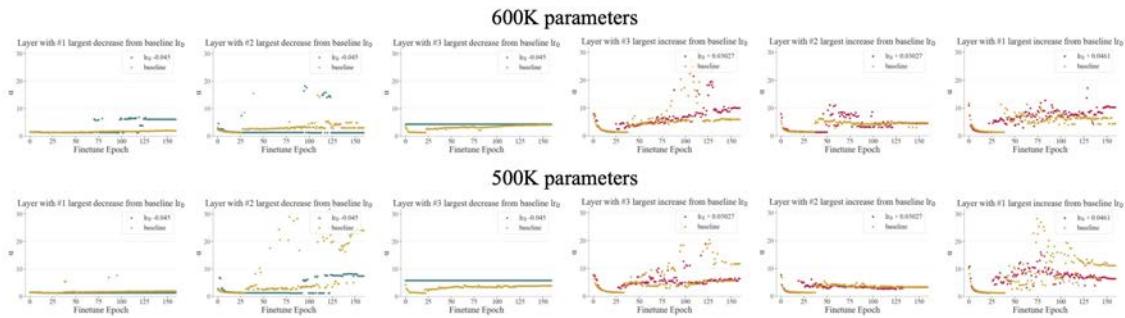


Figure 3.24: (Continued from previous page) VGG19 at 11 Sizes: Change of α through finetuning for 3 layers with the largest increase and 3 layers with largest decrease from initial baseline learning rates.

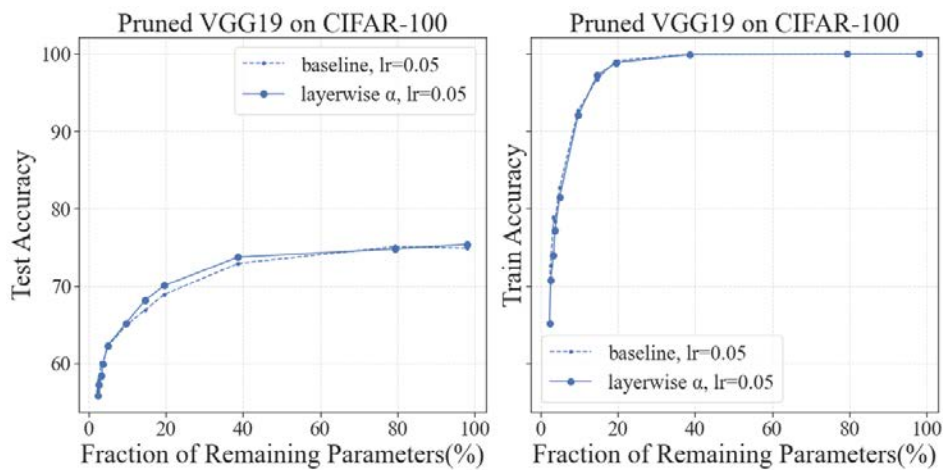


Figure 3.25: Effects of layerwise learning rate allocation on uniformly pruned VGG19, across a wide range of sparsities.

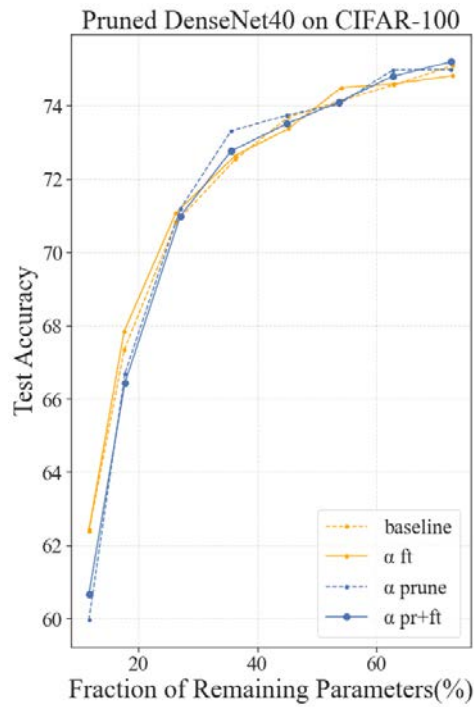


Figure 3.26: Comparing the four settings on varying sizes on DenseNet40: uniform pruning and global learning rate (baseline), uniform pruning and layerwise learning rates determined by α , layerwise pruning ratios determined by α and global learning rate (α prune), and *both* layerwise pruning ratios and layerwise learning rates determined by α . Learning rates are reallocated for the finetuning stage only, not for the initial training.

Bibliography

- [1] Amirali Aghazadeh et al. “Epistatic Net allows the sparse spectral regularization of deep neural networks for inferring fitness functions”. In: *Nature Communications* 12 (Sept. 2021). DOI: 10.1038/s41467-021-25371-3.
- [2] Amirali Aghazadeh et al. “Spectral Regularization Allows Data-frugal Learning over Combinatorial Spaces”. In: (2022). arXiv: 2210.02604 [stat.ML].
- [3] Jeff Alstott, Ed Bullmore, and Dietmar Plenz. “powerlaw: A Python Package for Analysis of Heavy-Tailed Distributions”. In: *PLoS ONE* 9.1 (Jan. 2014). Ed. by Fabio Rapallo, e85777. DOI: 10.1371/journal.pone.0085777. URL: <https://doi.org/10.1371/journal.pone.0085777>.
- [4] Yingbin Bai et al. “Understanding and Improving Early Stopping for Learning with Noisy Labels”. In: *CoRR* abs/2106.15853 (2021). arXiv: 2106.15853. URL: <https://arxiv.org/abs/2106.15853>.
- [5] Marwa Banna, Florence Merlevède, and Magda Peligrad. “On the limiting spectral distribution for a large class of symmetric random matrices with correlated entries”. In: *Stochastic Processes and their Applications* 125.7 (2015), pp. 2700–2726. ISSN: 0304-4149. DOI: <https://doi.org/10.1016/j.spa.2015.01.010>. URL: <https://www.sciencedirect.com/science/article/pii/S0304414915000290>.
- [6] Timothy Bellay et al. *Critical Phenomena in Natural Sciences - Chaos, Fractals, Self-Organization and Disorder - Concepts and Tools - 2nd Edition - 2006*. Jan. 2016.
- [7] Dimitri P. Bertsekas and John N. Tsitsiklis. *Introduction to Probability*. 2008.
- [8] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006. ISBN: 0387310738.
- [9] Christopher M. Bishop. “Regularization and complexity control in feed-forward networks”. International Conference on Artificial Neural Networks ICANN’95. 1995. URL: <https://publications.aston.ac.uk/id/eprint/524/>.
- [10] Davis W. Blalock et al. “What is the State of Neural Network Pruning?” In: *CoRR* abs/2003.03033 (2020). arXiv: 2003.03033. URL: <https://arxiv.org/abs/2003.03033>.
- [11] Tianlong Chen et al. “The Lottery Ticket Hypothesis for Pre-trained BERT Networks”. In: *CoRR* abs/2007.12223 (2020). arXiv: 2007.12223. URL: <https://arxiv.org/abs/2007.12223>.

- [12] Yixiong Chen et al. “MetaLR: Layer-wise Learning Rate based on Meta-Learning for Adaptively Fine-tuning Medical Pre-trained Models”. In: (2022). arXiv: 2206.01408 [cs.CV].
- [13] Yu Cheng et al. “A Survey of Model Compression and Acceleration for Deep Neural Networks”. In: *CoRR* abs/1710.09282 (2017). arXiv: 1710.09282. URL: <http://arxiv.org/abs/1710.09282>.
- [14] Aaron Clauset, Cosma Rohilla Shalizi, and M. E. J. Newman. “Power-Law Distributions in Empirical Data”. In: *SIAM Review* 51.4 (Nov. 2009), pp. 661–703. DOI: 10.1137/070710111. URL: <https://doi.org/10.1137%2F070710111>.
- [15] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing)*. USA: Wiley-Interscience, 2006. ISBN: 0471241954.
- [16] Erich Elsen et al. “Fast Sparse ConvNets”. In: *CoRR* abs/1911.09723 (2019). arXiv: 1911.09723. URL: <http://arxiv.org/abs/1911.09723>.
- [17] Massimiliano Esposito, Katja Lindenberg, and Christian Van den Broeck. “Entropy production as correlation between system and reservoir”. In: *New Journal of Physics* 12.1 (Jan. 2010), p. 013013. DOI: 10.1088/1367-2630/12/1/013013. URL: <https://doi.org/10.1088%2F1367-2630%2F12%2F1%2F013013>.
- [18] Utku Evci et al. “Rigging the Lottery: Making All Tickets Winners”. In: *CoRR* abs/1911.11134 (2019). arXiv: 1911.11134. URL: <http://arxiv.org/abs/1911.11134>.
- [19] Luca Franceschi et al. “Bilevel Programming for Hyperparameter Optimization and Meta-Learning”. In: (2018). arXiv: 1806.04910 [stat.ML].
- [20] Jonathan Frankle and Michael Carbin. “The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks”. In: *International Conference on Learning Representations*. 2019. URL: <https://openreview.net/forum?id=rJl-b3RcF7>.
- [21] Jonathan Frankle et al. “Pruning Neural Networks at Initialization: Why Are We Missing the Mark?” In: *International Conference on Learning Representations*. 2021. URL: <https://openreview.net/forum?id=Ig-VyQc-MLK>.
- [22] Matt Gardner et al. “AllenNLP: A Deep Semantic Natural Language Processing Platform”. In: *Proceedings of Workshop for NLP Open Source Software (NLP-OSS)*. Melbourne, Australia: Association for Computational Linguistics, July 2018, pp. 1–6. DOI: 10.18653/v1/W18-2501. URL: <https://aclanthology.org/W18-2501>.
- [23] Adom Giffin et al. “Updating Probabilities with Data and Moments”. In: *AIP Conference Proceedings*. AIP, 2007. DOI: 10.1063/1.2821302. URL: <https://doi.org/10.1063%2F1.2821302>.
- [24] V L Girko. “Asymptotics of the distribution of the spectrum of random matrices”. In: *Russian Mathematical Surveys* 44.4 (Aug. 1989), p. 3. DOI: 10.1070/RM1989v044n04ABEH002143.

- [25] Xavier Glorot and Yoshua Bengio. “Understanding the difficulty of training deep feed-forward neural networks”. In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Yee Whye Teh and Mike Titterton. Vol. 9. Proceedings of Machine Learning Research. Chia Laguna Resort, Sardinia, Italy: PMLR, 13–15 May 2010, pp. 249–256. URL: <https://proceedings.mlr.press/v9/glorot10a.html>.
- [26] A. Golubev. “Exponentially modified Gaussian (EMG) relevance to distributions related to cell proliferation and differentiation”. In: *Journal of Theoretical Biology* 262.2 (2010), pp. 257–266. ISSN: 0022-5193. DOI: <https://doi.org/10.1016/j.jtbi.2009.10.005>. URL: <https://www.sciencedirect.com/science/article/pii/S0022519309004809>.
- [27] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. <http://www.deeplearningbook.org>. MIT Press, 2016.
- [28] Friedrich Götze and Alexander Tikhomirov. “Rate of convergence in probability to the Marchenko-Pastur law”. In: *Bernoulli* 10.3 (2004), pp. 503–548. DOI: [10.3150/bj/1089206408](https://doi.org/10.3150/bj/1089206408). URL: <https://doi.org/10.3150/bj/1089206408>.
- [29] Eli Grushka. “Characterization of exponentially modified Gaussian peaks in chromatography.” In: *Analytical chemistry* 44 11 (1972), pp. 1733–8.
- [30] Wooseok Ha et al. “Adaptive wavelet distillation from neural networks through interpretations”. In: (2021). arXiv: 2107.09145 [stat.ML].
- [31] Song Han et al. “Learning Both Weights and Connections for Efficient Neural Networks”. In: *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*. NIPS’15. Montreal, Canada: MIT Press, 2015, pp. 1135–1143.
- [32] Babak Hassibi and David Stork. “Second order derivatives for network pruning: Optimal Brain Surgeon”. In: *Advances in Neural Information Processing Systems*. Ed. by S. Hanson, J. Cowan, and C. Giles. Vol. 5. Morgan-Kaufmann, 1992. URL: https://proceedings.neurips.cc/paper_files/paper/1992/file/303ed4c69846ab36c2904d3ba8573050-Paper.pdf.
- [33] Kaiming He et al. “Deep Residual Learning for Image Recognition”. In: *CoRR* abs/1512.03385 (2015). arXiv: 1512.03385. URL: <http://arxiv.org/abs/1512.03385>.
- [34] Kaiming He et al. “Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification”. In: *CoRR* abs/1502.01852 (2015). arXiv: 1502.01852. URL: <http://arxiv.org/abs/1502.01852>.
- [35] Kaiming He et al. “Identity Mappings in Deep Residual Networks”. In: *CoRR* abs/1603.05027 (2016). arXiv: 1603.05027. URL: <http://arxiv.org/abs/1603.05027>.
- [36] Dan Hendrycks et al. “AugMix: A Simple Data Processing Method to Improve Robustness and Uncertainty”. In: (2020). arXiv: 1912.02781 [stat.ML].

- [37] Andrew G. Howard et al. “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications”. In: *CoRR* abs/1704.04861 (2017). arXiv: 1704.04861. URL: <http://arxiv.org/abs/1704.04861>.
- [38] Gao Huang et al. “Densely Connected Convolutional Networks”. In: (2018). arXiv: 1608.06993 [cs.CV].
- [39] E. T. Jaynes. “Information Theory and Statistical Mechanics”. In: *Physical Review* 106.4 (May 1957), pp. 620–630. DOI: 10.1103/PhysRev.106.620.
- [40] E. T. Jaynes. *Probability theory: The logic of science*. Cambridge: Cambridge University Press, 2003.
- [41] E. T. Jaynes and Roger D. Rosenkrantz. *E.T. Jaynes : papers on probability, statistics, and statistical physics / edited by R.D. Rosenkrantz*. English. D. Reidel ; Sold, distributed in the U.S.A., and Canada by Kluwer Boston Dordrecht, Holland ; Boston : Hingham, MA, 1983, xxiv, 434 p. : ISBN: 9027714487.
- [42] Yiding Jiang et al. “Fantastic Generalization Measures and Where to Find Them”. In: *CoRR* abs/1912.02178 (2019). arXiv: 1912.02178. URL: <http://arxiv.org/abs/1912.02178>.
- [43] Gaojie Jin et al. “How does Weight Correlation Affect the Generalisation Ability of Deep Neural Networks”. In: (2020). arXiv: 2010.05983 [cs.LG].
- [44] Alexander B. Jung. *imgaug*. <https://github.com/aleju/imgaug>. [Online; accessed 30-Oct-2018]. 2018.
- [45] Nikhil Ketkar. “Stochastic Gradient Descent”. In: *Deep Learning with Python: A Hands-on Introduction*. Berkeley, CA: Apress, 2017, pp. 113–132. ISBN: 978-1-4842-2766-4. DOI: 10.1007/978-1-4842-2766-4_8. URL: https://doi.org/10.1007/978-1-4842-2766-4_8.
- [46] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: (2014). DOI: 10.48550/ARXIV.1412.6980. URL: <https://arxiv.org/abs/1412.6980>.
- [47] Simon Kornblith et al. “Similarity of Neural Network Representations Revisited”. In: (2019). arXiv: 1905.00414 [cs.LG].
- [48] Erwin Kreyszig, Herbert Kreyszig, and E. J. Norminton. *Advanced Engineering Mathematics*. Tenth. Hoboken, NJ: Wiley, 2011. ISBN: 0470458364.
- [49] A. Krizhevsky and G. Hinton. “Learning multiple layers of features from tiny images”. In: *Master’s thesis, Department of Computer Science, University of Toronto* (2009).
- [50] Yann LeCun, John Denker, and Sara Solla. “Optimal Brain Damage”. In: *Advances in Neural Information Processing Systems*. Ed. by D. Touretzky. Vol. 2. Morgan-Kaufmann, 1989. URL: https://proceedings.neurips.cc/paper_files/paper/1989/file/6c9882bbac1c7093bd25041881277658-Paper.pdf.

- [51] Jaeho Lee et al. “Layer-adaptive Sparsity for the Magnitude-based Pruning”. In: *International Conference on Learning Representations*. 2021. URL: <https://openreview.net/forum?id=H6ATjJ0TKdf>.
- [52] Asriel Levin, Todd Leen, and John Moody. “Fast Pruning Using Principal Components”. In: *Advances in Neural Information Processing Systems*. Ed. by J. Cowan, G. Tesauro, and J. Alspector. Vol. 6. Morgan-Kaufmann, 1993.
- [53] Hao Li et al. “Pruning Filters for Efficient ConvNets”. In: *CoRR* abs/1608.08710 (2016). arXiv: 1608.08710. URL: <http://arxiv.org/abs/1608.08710>.
- [54] Ming Li et al. “Fast Haar Transforms for Graph Neural Networks”. In: *CoRR* abs/1907.04786 (2019). arXiv: 1907.04786. URL: <http://arxiv.org/abs/1907.04786>.
- [55] Zhiyuan Li and Sanjeev Arora. “An Exponential Learning Rate Schedule for Deep Learning”. In: (2019). arXiv: 1910.07454 [cs.LG].
- [56] Zhuohan Li et al. “Train Large, Then Compress: Rethinking Model Size for Efficient Training and Inference of Transformers”. In: *CoRR* abs/2002.11794 (2020). arXiv: 2002.11794. URL: <https://arxiv.org/abs/2002.11794>.
- [57] Zhuang Liu et al. “Learning Efficient Convolutional Networks through Network Slimming”. In: *CoRR* abs/1708.06519 (2017). arXiv: 1708.06519. URL: <http://arxiv.org/abs/1708.06519>.
- [58] Zhuang Liu et al. “Rethinking the Value of Network Pruning”. In: (2019). arXiv: 1810.05270 [cs.LG].
- [59] Guanping Lu et al. “Analysis on the Empirical Spectral Distribution of Large Sample Covariance Matrix and Applications for Large Antenna Array Processing”. In: *IEEE Access* 7 (2019), pp. 30135–30141. DOI: 10.1109/ACCESS.2019.2902039.
- [60] David J. C. MacKay. *Information Theory, Inference, and Learning Algorithms*. Copyright Cambridge University Press, 2003.
- [61] Michael W. Mahoney and Lorenzo Orecchia. “Implementing regularization implicitly via approximate eigenvector computation”. In: *CoRR* abs/1010.0703 (2010). arXiv: 1010.0703. URL: <http://arxiv.org/abs/1010.0703>.
- [62] Huizi Mao et al. “Exploring the Regularity of Sparse Structure in Convolutional Neural Networks”. In: *CoRR* abs/1705.08922 (2017). arXiv: 1705.08922. URL: <http://arxiv.org/abs/1705.08922>.
- [63] V A Marčenko and L A Pastur. “DISTRIBUTION OF EIGENVALUES FOR SOME SETS OF RANDOM MATRICES”. In: *Mathematics of the USSR-Sbornik* 1.4 (Apr. 1967), p. 457. DOI: 10.1070/SM1967v001n04ABEH001994. URL: <https://dx.doi.org/10.1070/SM1967v001n04ABEH001994>.
- [64] Charles H Martin and Michael W Mahoney. “Implicit Self-Regularization in Deep Neural Networks: Evidence from Random Matrix Theory and Implications for Learning”. In: *Journal of Machine Learning Research* 22.165 (2021), pp. 1–73.

- [65] Charles H Martin and Michael W Mahoney. *Rethinking generalization requires revisiting old ideas: statistical mechanics approaches and complex learning behavior*. Tech. rep. Preprint: arXiv:1710.09553. 2017.
- [66] Charles H Martin and Michael W Mahoney. “Traditional and heavy tailed self regularization in neural network models”. In: *International Conference on Machine Learning*. 2019, pp. 4284–4293.
- [67] Charles H Martin, Tongsu (Serena) Peng, and Michael W Mahoney. “Predicting trends in the quality of state-of-the-art neural networks without access to training or testing data”. In: *arXiv preprint arXiv:2002.06716* (2020).
- [68] Rajiv Movva, Jonathan Frankle, and Michael Carbin. “Studying the Consistency and Composability of Lottery Ticket Pruning Masks”. In: *CoRR* abs/2104.14753 (2021). arXiv: 2104.14753. URL: <https://arxiv.org/abs/2104.14753>.
- [69] M. C. Mozer and P. Smolensky. “Skeletonization: A technique for trimming the fat from a network via relevance assessment”. In: *Advances in Neural Information Processing Systems (NIPS) 1*. Ed. by D. S. Touretzky. Morgan Kaufmann, 1989, pp. 107–115.
- [70] Rafael Müller, Simon Kornblith, and Geoffrey E. Hinton. “When Does Label Smoothing Help?” In: *CoRR* abs/1906.02629 (2019). arXiv: 1906.02629. URL: <http://arxiv.org/abs/1906.02629>.
- [71] Sharan Narang, Eric Undersander, and Gregory F. Diamos. “Block-Sparse Recurrent Neural Networks”. In: *CoRR* abs/1711.02782 (2017). arXiv: 1711.02782. URL: <http://arxiv.org/abs/1711.02782>.
- [72] Behnam Neyshabur. “Implicit Regularization in Deep Learning”. In: *CoRR* abs/1709.01953 (2017). arXiv: 1709.01953. URL: <http://arxiv.org/abs/1709.01953>.
- [73] Behnam Neyshabur, Ryota Tomioka, and Nathan Srebro. *In Search of the Real Inductive Bias: On the Role of Implicit Regularization in Deep Learning*. 2015. arXiv: 1412.6614 [cs.LG].
- [74] Sean O’Rourke. “A note on the Marchenko-Pastur law for a class of random matrices with dependent entries”. In: *Electronic Communications in Probability* 17.none (Jan. 2012). DOI: 10.1214/ecp.v17-2020. URL: <https://doi.org/10.1214%2Fecp.v17-2020>.
- [75] Ankit Pensia et al. “Optimal Lottery Tickets via SubsetSum: Logarithmic Over-Parameterization is Sufficient”. In: *CoRR* abs/2006.07990 (2020). arXiv: 2006.07990. URL: <https://arxiv.org/abs/2006.07990>.
- [76] Vivek Ramanujan et al. “What’s Hidden in a Randomly Weighted Neural Network?” In: *CoRR* abs/1911.13299 (2019). arXiv: 1911.13299. URL: <http://arxiv.org/abs/1911.13299>.
- [77] John A. Rice. *Mathematical Statistics and Data Analysis*. Third. Belmont, CA: Duxbury Press., 2006.

- [78] Youngmin Ro and Jin Young Choi. “AutoLR: Layer-wise Pruning and Auto-tuning of Learning Rates in Fine-tuning of Deep Networks”. In: (2021). arXiv: 2002.06048 [cs.CV].
- [79] Jonathan S. Rosenfeld et al. “On the Predictability of Pruning Across Scales”. In: *CoRR* abs/2006.10621 (2020). arXiv: 2006.10621. URL: <https://arxiv.org/abs/2006.10621>.
- [80] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. “Learning representations by back-propagating errors”. In: *Nature* 323 (1986), pp. 533–536.
- [81] Mark Sandler et al. “MobileNetV2: Inverted Residuals and Linear Bottlenecks”. In: (2019). arXiv: 1801.04381 [cs.CV].
- [82] Tom Schaul, Sixin Zhang, and Yann LeCun. “No More Pesky Learning Rates”. In: (2013). arXiv: 1206.1106 [stat.ML].
- [83] Jürgen Schmidhuber. “Deep learning in neural networks: An overview”. In: *Neural Networks* 61 (Jan. 2015), pp. 85–117. DOI: 10.1016/j.neunet.2014.09.003. URL: <https://doi.org/10.1016%2Fj.neunet.2014.09.003>.
- [84] Vikash Sehwal et al. “HYDRA: Pruning Adversarially Robust Neural Networks”. In: (2020). arXiv: 2002.10509 [cs.CV].
- [85] Leslie N. Smith. “Cyclical Learning Rates for Training Neural Networks”. In: *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*. 2017, pp. 464–472. DOI: 10.1109/WACV.2017.58.
- [86] Terence Tao and Van Vu. “From the Littlewood-Offord problem to the Circular Law: universality of the spectral distribution of random matrices”. In: (2009). arXiv: 0810.2994 [math.PR].
- [87] Tijmen Tieleman and Geoffrey Hinton. “Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude”. In: *COURSERA: Neural networks for machine learning* 4.2 (2012), pp. 26–31.
- [88] Wei Wen et al. “Learning Structured Sparsity in Deep Neural Networks”. In: *CoRR* abs/1608.03665 (2016). arXiv: 1608.03665. URL: <http://arxiv.org/abs/1608.03665>.
- [89] Haokai Xi, Fan Yang, and Jun Yin. “Convergence of eigenvector empirical spectral distribution of sample covariance matrices”. In: *The Annals of Statistics* 48.2 (2020), pp. 953–982. DOI: 10.1214/19-AOS1832. URL: <https://doi.org/10.1214/19-AOS1832>.
- [90] Saining Xie et al. “Aggregated Residual Transformations for Deep Neural Networks”. In: (2017). arXiv: 1611.05431 [cs.CV].
- [91] Zhen Xu et al. “Learning an Adaptive Learning Rate Schedule”. In: (2019). arXiv: 1909.09712 [cs.LG].

- [92] Yaoqing Yang et al. “Evaluating natural language processing models with generalization metrics that do not need access to any training or testing data”. In: (2022). DOI: 10.48550/ARXIV.2202.02842. URL: <https://arxiv.org/abs/2202.02842>.
- [93] Yaoqing Yang et al. “Taxonomizing local versus global structure in neural network loss landscapes”. In: (2021). arXiv: 2107.11228 [cs.LG].
- [94] Y.Q. Yin. “Limiting spectral distribution for a class of random matrices”. In: *Journal of Multivariate Analysis* 20.1 (1986), pp. 50–68. ISSN: 0047-259X. DOI: [https://doi.org/10.1016/0047-259X\(86\)90019-9](https://doi.org/10.1016/0047-259X(86)90019-9). URL: <https://www.sciencedirect.com/science/article/pii/0047259X86900199>.
- [95] Sergey Zagoruyko and Nikos Komodakis. “Wide Residual Networks”. In: *CoRR* abs/1605.07146 (2016). arXiv: 1605.07146. URL: <http://arxiv.org/abs/1605.07146>.
- [96] Michael H. Zhu and Suyog Gupta. “To Prune, or Not to Prune: Exploring the Efficacy of Pruning for Model Compression”. In: (2018). URL: <https://openreview.net/forum?id=S11N69AT->.

Appendix A

Convolution of Exponential and Normal Distributions

Thank you to Syomantak Chaudhuri for elaborating and correcting this convolution proof.

Let $X \sim \text{Exp}(\lambda)$ and $Y \sim \mathcal{N}(\mu, \sigma^2)$ and $Z = X + Y$. We have

$$f_X(x) = \lambda e^{-\lambda x} \text{ and } f_Y(x) = \frac{1}{\sigma\sqrt{2\pi}e^{-\frac{(x-\mu)^2}{2\sigma^2}}} \quad (\text{A.1})$$

$$f_{X+Y}(x) = \frac{\lambda}{\sigma\sqrt{2\pi}} \int_0^\infty \exp\left(-\lambda t - \frac{(x-\mu)^2 + t^2 - 2t(x-\mu)}{2\sigma^2}\right) dt \quad (\text{A.2})$$

$$f_{X+Y}(x) = \frac{\lambda}{\sigma\sqrt{2\pi}} \int_0^\infty \exp\left(-\frac{2\sigma^2\lambda t + (x-\mu)^2 + t^2 - 2t(x-\mu)}{2\sigma^2}\right) dt \quad (\text{A.3})$$

$$f_{X+Y}(x) = \frac{\lambda}{\sigma\sqrt{2\pi}} \int_0^\infty \exp\left(-\frac{t^2 + 2t(\sigma^2\lambda - (x-\mu)) + (x-\mu)^2}{2\sigma^2}\right) dt \quad (\text{A.4})$$

$$f_{X+Y}(x) = \frac{\lambda}{\sigma\sqrt{2\pi}} \int_0^\infty \exp\left(-\frac{t^2 - 2t(x-\mu - \sigma^2\lambda) + (x-\mu)^2}{2\sigma^2}\right) dt. \quad (\text{A.5})$$

To write the numerator of the exponent as the square of a difference,

$$[t - (x - \mu - \sigma^2\lambda)]^2 = t^2 - 2t(x - \mu - \sigma^2\lambda) + (x - \mu - \sigma^2\lambda)^2, \quad (\text{A.6})$$

where $(x - \mu - \sigma^2\lambda)^2 = x^2 + \mu^2 + \sigma^4\lambda^2 - 2x\mu + 2\mu\sigma^2\lambda - 2x\sigma^2\lambda = (x - \mu)^2 + \sigma^2\lambda(\sigma^2\lambda + 2\mu - 2x)$.

$$f_{X+Y}(x) = \frac{\lambda}{\sigma\sqrt{2\pi}} \int_0^\infty \exp\left(-\frac{[t - (x - \mu - \sigma^2\lambda)]^2 - \sigma^2\lambda(\sigma^2\lambda + 2\mu - 2x)}{2\sigma^2}\right) dt \quad (\text{A.7})$$

$$f_{X+Y}(x) = \frac{\lambda}{\sigma\sqrt{2\pi}} \exp\left(\frac{\sigma^2\lambda(\sigma^2\lambda + 2\mu - 2x)}{2\sigma^2}\right) \int_0^\infty \exp\left(-\frac{[t - (x - \mu - \sigma^2\lambda)]^2}{2\sigma^2}\right) dt \quad (\text{A.8})$$

Change of variables $z := \frac{t-(x-\mu-\sigma^2\lambda)}{\sqrt{2\sigma^2}}$, $dz = \frac{dt}{\sqrt{2\sigma^2}}$

$$f_{X+Y}(x) = \frac{\lambda}{\sigma\sqrt{2\pi}} \exp\left(\frac{\lambda}{2}(\sigma^2\lambda + 2\mu - 2x)\right) \int_{-\frac{(x-\mu-\sigma^2\lambda)}{\sqrt{2\sigma^2}}}^{\infty} \exp(-z^2) dz \cdot \sqrt{2\sigma^2} \quad (\text{A.9})$$

$$f_{X+Y}(x) = \frac{\lambda}{\sqrt{\pi}} \exp\left(\frac{\lambda}{2}(\sigma^2\lambda + 2\mu - 2x)\right) \int_{-\frac{(x-\mu-\sigma^2\lambda)}{\sqrt{2\sigma^2}}}^{\infty} \exp(-z^2) dz. \quad (\text{A.10})$$

We write the integral as

$$\int_{-\frac{(x-\mu-\sigma^2\lambda)}{\sqrt{2\sigma^2}}}^{\infty} \exp(-z^2) dz = \int_{-\frac{(x-\mu-\sigma^2\lambda)}{\sqrt{2\sigma^2}}}^0 \exp(-z^2) dz + \int_0^{\infty} \exp(-z^2) dz, \quad (\text{A.11})$$

where $\int_0^{\infty} \exp(-z^2) dz = \text{erf}(\infty) \cdot \frac{\sqrt{\pi}}{2} = \frac{\pi}{2}$ and $\text{erf}(\infty) = 1$. The error function is defined as $\text{erf}(y) = \frac{2}{\sqrt{\pi}} \int_0^y e^{-t^2} dt$ [60]. For a random variable $X \sim \mathcal{N}(0, \frac{1}{2})$, $\text{erf}(y)$ is the probability that X falls in the range $[-y, y]$.

$$\int_{-\frac{(x-\mu-\sigma^2\lambda)}{\sqrt{2\sigma^2}}}^{\infty} \exp(-z^2) dz = \frac{\sqrt{\pi}}{2} \text{erf}\left(\frac{x-\mu-\sigma^2\lambda}{\sqrt{2\sigma^2}}\right) + \frac{\sqrt{\pi}}{2}. \quad (\text{A.12})$$

We have

$$f_{X+Y}(x) = \frac{\lambda}{\sqrt{\pi}} \exp\left(\frac{\lambda}{2}(2\mu + \sigma^2\lambda - 2x)\right) \cdot \left(\frac{\sqrt{\pi}}{2} \text{erf}\left(\frac{x-\mu-\sigma^2\lambda}{\sqrt{2\sigma^2}}\right) + \frac{\sqrt{\pi}}{2}\right). \quad (\text{A.13})$$

$$f_Z(x) = \frac{\lambda}{2} \exp\left(\frac{\lambda}{2}(2\mu + \sigma^2\lambda - 2x)\right) \cdot \left(\text{erf}\left(\frac{x-\mu-\sigma^2\lambda}{\sqrt{2\sigma^2}}\right) + 1\right). \quad (\text{A.14})$$