

UNIVERSITY OF CALGARY

Sparse Training from Random Initialization:
Aligning Lottery Ticket Masks using Weight Symmetry

by

Rohan Jain

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE
DEGREE OF MASTER OF SCIENCE

GRADUATE PROGRAM IN ELECTRICAL ENGINEERING

CALGARY, ALBERTA

JUNE, 2025

© Rohan Jain 2025

Abstract

The Lottery Ticket Hypothesis (LTH) suggests there exists a sparse LTH mask and weights that achieve the same generalization performance as the dense model while using significantly fewer parameters. LTH achieves this by iteratively sparsifying and re-training within the pruned solution basin. However, finding a LTH solution is computationally expensive, and a LTH’s sparsity mask does not generalize to other random weight initializations. Recent work has suggested that neural networks trained from random initialization find solutions within the same basin modulo permutations, and proposes a method to align trained models within the same loss basin. We hypothesize that misalignment of basins is the reason why LTH masks do not generalize to new random initializations and propose permuting the LTH mask to align with the new optimization basin when performing sparse training from a different random initialization. We empirically show a significant increase in generalization when sparse training from random initialization with the permuted mask as compared to using the non-permuted LTH mask, on multiple datasets (CIFAR-10/100 & ImageNet) and models (VGG11 & ResNet20/50).

Preface

Manuscripts

1. **Rohan Jain**¹, Mohammed Adnan^{*1}, Ekansh Sharma, and Yani Ioannou. (2024). *Winning Tickets from Random Initialization: Aligning Masks for Sparse Training*. Accepted to UniReps: 2nd Edition of the Workshop on Unifying Representations in Neural Models.
2. Mohammed Adnan^{*1}, **Rohan Jain**^{*1}, Ekansh Sharma, and Yani Ioannou. (2025). *Sparse Training from Random Initialization: Aligning Lottery Ticket Masks using Weight Symmetry*. Accepted to [The Second Conference on Parsimony and Learning \(Spotlight Track\)](#).
3. Mohammed Adnan^{*1}, **Rohan Jain**^{*1}, Ekansh Sharma, and Yani Ioannou. (2025). *Sparse Training from Random Initialization: Aligning Lottery Ticket Masks using Weight Symmetry*. Accepted to the Forty-Second International Conference on Machine Learning (ICML).

¹Equally Contributed 1st Authors.

- **Chapter 2 Background:** The content in this chapter is written by Rohan Jain. Figures 2.3, 2.4, and 2.5 are from Ioannou (2022).
- **Chapter 3 Methodology:** The content in this chapter is based on our submitted ICML paper, with the section jointly written by Mohammed Adnan and Rohan Jain. Figures 3.1, 3.2, 3.3, and 3.4 were made by Yani Ioannou.
- **Chapter 4 Experiments and Results:** The content in this chapter is based on our submitted ICML paper.
 1. **Chapter 4.1:** The experiments and content in this chapter were jointly conducted and written by Mohammed Adnan and Rohan Jain.
 2. **Chapter 4.2:** The experiments in section 4.2.1 was conducted and written by Ekansh Sharma. The experiments and content in section 4.2.2 were jointly conducted and written by Mohammed Adnan and Rohan Jain.
 3. **Chapter 4.3:** The experiments and content in this section were conducted and written by Rohan Jain.
 4. **Chapter 4.4:** The experiments and content were jointly conducted and written by Mohammed Adnan and Rohan Jain.

Acknowledgments

I would first like to express my sincere gratitude to my supervisor, **Dr. Yani Ioannou**, for his exceptional guidance and support throughout this research journey. His assistance in navigating roadblocks, resolving engineering challenges, providing crucial technical insights, and maintaining the overall vision of this project has been truly amazing, and for this, I am deeply grateful. I also extend my appreciation to my primary co-author and mentor, **Mohammed Adnan** (Ph.D. Candidate). His dedicated support and mentorship was tremendously valuable to this study, as he was directly involved in the technical development, experimental procedures, and provided continuous guidance. Furthermore, I would like to thank my other co-author and mentor, **Ekansh Sharma** (Ph.D. Candidate), for his technical contributions, assistance with the experiments, and overall mentorship throughout this project. I am also grateful to Dr. Ioannou for his support in providing essential compute resources, including the Calgary Machine Learning Lab cluster and the Digital Research Alliance of Canada. Finally, I would like to thank my lab mate, **Mike Lasby** (Ph.D. Candidate), for his helpful contributions in providing literature resources and engineering tips during the initial stages of this research project.

Table of Contents

Abstract	ii
Preface	iii
Manuscripts	iii
Acknowledgments	v
Table of Contents	vi
List of Tables	x
List of Figures	xiii
1 Introduction	1
1.1 Overview	1
1.2 Our Contributions	3
1.3 Thesis Organization	4
2 Background	6
2.1 Formal Setup: Machine Learning	6
2.2 Neural Networks	7
2.2.1 Formal Setup: Feedforward Neural Networks	7
2.2.2 What is the Loss Landscape?	10
2.2.3 Geometry of the Loss Landscape	10
2.2.4 Permutation Symmetries in the Loss Landscape	15

2.2.5	Permutation Invariance and Mode Connectivity . . .	17
2.2.5.1	What is Mode Connectivity?	17
2.2.5.2	Linear Mode Connectivity (LMC)	18
2.2.5.3	LMC <i>modulo</i> Permutation	19
2.2.5.4	Aligning Neural Networks via Permutation	20
2.2.5.5	Aligning Neural Networks: Weights	23
2.2.5.6	Aligning Neural Networks: Activations	26
2.2.5.7	REPAIR: Renormalizing Permuted Activations	27
2.3	Sparsity in Neural Networks	29
2.3.1	What are Sparse Neural Networks?	29
2.3.1.1	Sparsity and the Brain	29
2.3.1.2	Optimal Brain Damage	30
2.3.1.3	Optimal Brain Surgeon	31
2.3.2	How to Prune Deep Neural Networks?	32
2.3.2.1	Unstructured Pruning	33
2.3.2.2	Iterative Magnitude Pruning	33
2.3.3	What is Sparse Training?	35
2.3.3.1	Sparse Initialization: Challenges	36
2.3.3.2	The Sparse Training Problem	38
2.3.4	Lottery Ticket Hypothesis	39
2.3.4.1	Linear Mode Connectivity and Sparsity	42
2.3.4.2	Limitations of Lottery Ticket Hypothesis	43
3	Methodology	44
3.1	Motivation	44
3.1.1	Visualization: 2D Loss Landscape	46

3.1.1.1	Dense Training and Pruning	46
3.1.1.2	Lottery Ticket Hypothesis	47
3.1.1.3	Our Method	48
3.2	Permutation Matching	48
3.2.1	Evaluating Permutation Matching	49
3.2.2	Aligning Masks via Weight Symmetry	50
3.3	Sparse Training	51
4	Experiments and Results	53
4.1	Experimental Results	54
4.1.1	ResNet20/CIFAR-10 & CIFAR-100	54
4.1.2	VGG11/CIFAR-10	56
4.1.3	ResNet50/ImageNet	57
4.1.4	Early Permutation Matching	59
4.1.4.1	Computational Cost of the Permuted Solution	60
4.1.5	Analysis of Mask Reusage	60
4.1.5.1	Experimental Setup and Results	60
4.2	Diversity Analysis of Permuted Models	62
4.2.1	Loss Landscape Analysis	62
4.2.2	Permuted Ensemble Analysis	63
4.3	Effect of Model Width Multiplier	67
4.3.1	Experimental Details	67
4.3.2	Model Width Analysis	67
4.4	Additional Results and Details	69
4.4.1	Architectural Details	69
4.4.1.1	ResNet20 & VGG11 on CIFAR-10/100	69

4.4.1.2	ResNet50 on ImageNet	70
4.4.2	Pruning Hyperparameters	70
5	Conclusion	78
5.1	Discussion	78
5.2	Limitations and Future Directions	79
	Bibliography	81
	Appendix	99
	Mathematical Proof: Permutation Symmetries	99

List of Tables

4.1	ResNet20×{1}/CIFAR-10. Results using the ResNet20×{1} trained on CIFAR-10, from a rewind point $k = 20$, using various methods of sparse training with sparsity S . The LTH and naive methods remain fixed, as they are independent of matching. For the permuted method, the permutation, π , is obtained by matching a fully trained dense model $\mathbf{w}_A^{t=T}$ and $\mathbf{w}_B^{t=i}$ at an early point in training, where $i \in \{5, 20, 50, 100\}$.	61
4.2	ResNet20×{1}/SVHN. Mask reuse results for different sparsity levels and rewind points k .	62
4.3	Ensemble Diversity Metrics for CIFAR-10/CIFAR-100. Although the mean test accuracy of LTH is higher, the ensemble of permuted models achieves better test accuracy due to better functional diversity of permuted models.	65
4.4	Hyperparameters for dense and sparse training of both ResNet20 and VGG11.	69
4.5	Hyperparameters for dense and sparse training of ResNet50.	70
4.6	Hyperparameters used for pruning ResNet20/VGG11 on CIFAR-10/100 and ResNet50 on ImageNet.	70

4.8	ResNet20×{4}/CIFAR-10. Results using the ResNet20×{4} trained on CIFAR-10, from a rewind point k , using various methods of sparse training with sparsity S . LTH trains within the original dense/pruned solution basin, while naive/permutated train from a new random initialization. Note this table is the same setting as table 4.7 except $w = 4$.	71
4.7	ResNet20×{1}/CIFAR-10. Results using the ResNet20×{1} trained on CIFAR-10, from a rewind point k , using various methods of sparse training with sparsity S . LTH trains within the original dense/pruned solution basin, while naive/permutated train from a new random initialization.	71
4.9	ResNet20×{8}/CIFAR-10. Results using the ResNet20×{8} trained on CIFAR-10, from a rewind point k , using various methods of sparse training with sparsity S . LTH trains within the original dense/pruned solution basin, while naive/permutated train from a new random initialization. Note this table is the same setting as table 4.7 except $w = 8$.	72
4.10	ResNet20×{16}/CIFAR-10. Results using the ResNet20×{8} trained on CIFAR-10, from a rewind point k , using various methods of sparse training with sparsity S . LTH trains within the original dense/pruned solution basin, while naive/permutated train from a new random initialization. Note this table is the same setting as table 4.7 except $w = 16$.	72
4.11	VGG11×{1}/CIFAR-10. Results using the VGG11 trained on CIFAR-10, from a rewind point k , using various methods of sparse training with sparsity S . LTH trains within the original dense/pruned solution basin, while naive/permutated train from a new random initialization. . . .	73

4.12	ResNet50×{1}/ImageNet. Top-1 and Top-5 Accuracies of ResNet50×{1} trained on ImageNet, from a rewind point k , using various methods of sparse training with sparsity S	73
4.13	ResNet20×{1}/CIFAR-100. Results using the ResNet20×{1} trained on CIFAR-100, from a rewind point k , using various methods of sparse training with sparsity S . LTH trains within the original dense/pruned solution basin, while naive/permutated train from a new random initialization.	74
4.14	ResNet20×{4}/CIFAR-100. Results using the ResNet20×{4} trained on CIFAR-100, from a rewind point k , using various methods of sparse training with sparsity S . LTH trains within the original dense/pruned solution basin, while naive/permutated train from a new random initialization. Note this table is the same setting as table 4.13 except $w = 4$	74
4.15	ResNet20×{8}/CIFAR-100. Results using the ResNet20×{8} trained on CIFAR-100, from a rewind point k , using various methods of sparse training with sparsity S . LTH trains within the original dense/pruned solution basin, while naive/permutated train from a new random initialization. Note this table is the same setting as table 4.13 except $w = 8$	75
4.16	ResNet20×{16}/CIFAR-100. Results using the ResNet20×{16} trained on CIFAR-100, from a rewind point k , using various methods of sparse training with sparsity S . LTH trains within the original dense/pruned solution basin, while naive/permutated train from a new random initialization. Note this table is the same setting as table 4.13 except $w = 16$	75

List of Figures

2.1	An intuitive visualization of the loss landscapes of ResNet56 without (a) and with skip connections (b), showing how architectural design choices affect the notion of sharpness and flatness of loss surfaces. Sourced from Li et al. (2018b).	11
2.2	Permutation symmetry in a single hidden layer Deep Neural Network (DNN), illustrating that the outputs, y and y' , remain equivalent for the same input despite the swapping of neurons within the hidden layer, thus demonstrating the invariance of the loss function under permutation. For simplicity, we do not include the biases.	16
2.3	Visualization of the sparse training problem paradigm (highlighted in red). Training a fixed mask, \mathbf{m}_A , obtained from a parent initialization, $\mathbf{w}_A^{t=0}$, does not perform well when equipped with a random, arbitrary initialization. Sourced from Ioannou (2022).	38

2.4	Visualization of the standard LTH paradigm. The Lottery Ticket (LT) initialization is composed of the original dense initialization, $\mathbf{w}_A^{t=0}$, we used to first train our model to convergence and then pruned via Iterative Magnitude Pruning (IMP) to obtain our mask (i.e. winning ticket), \mathbf{m}_A . Then we sparse train, $\mathbf{w}_A^{t=0} \odot \mathbf{m}_A$. Sourced from Ioannou (2022).	40
2.5	Similar to the setting from Figure 2.4, but instead our LT initialization is some “early snapshot” of dense training, equipped with the same \mathbf{m}_A . Then we sparse train, $\mathbf{w}_A^{t=k} \odot \mathbf{m}_A$. Sourced from Ioannou (2022).	42
3.1	Visualization of the standard dense training and post-trained pruning via IMP to obtain a sparse mask, \mathbf{m}_A . Sourced from Yani Ioannou.	47
3.2	Visualization of LTH. Sourced from Yani Ioannou.	48
3.3	Visualization of the sparse training problem and how we aim to solve it. Sourced from Yani Ioannou.	49

3.4	The overall framework of the training procedure, beginning with two distinct dense random weight initializations, $\mathbf{w}_A^{t=0}, \mathbf{w}_B^{t=0}$ sampled from a normal distribution, \mathcal{N} . The sparse training problem attempts to train the random initialization, $\mathbf{w}_B^{t=0}$ using the naive mask \mathbf{m}_A , found by pruning a dense trained model, $\mathbf{w}_A^{t=T}$. However, this results in poor generalization performance (Frankle et al., 2020b). We propose to instead train $\mathbf{w}_B^{t=k}$ at some rewind epoch k , equipped with a <i>permuted</i> mask $\pi(\mathbf{m}_A)$. We show that this achieves more comparable generalization to the pruned model/trained LTH solution, $\mathbf{w}_A^{t=T} \odot \mathbf{m}_A$. Sourced from Yani Ioannou.	51
4.1	ResNet20\times{w}/CIFAR-10. Test accuracy of sparse network solutions vs. increasing rewind points for different sparsity levels and widths, w . The effect of the rewind point on the test accuracy for different sparsities is shown. The dashed (- -) line shows the dense model accuracy.	55

-
- 4.2 **ResNet20 $\times\{w\}$ /CIFAR-100.** Test accuracy of sparse network solutions vs. increasing rewind points for different sparsity levels and widths, w . The effect of the rewind points on the test accuracy for different sparsities is shown. As the width increases, the gap between training from a random initialization with the permuted mask and the LTH/dense baseline (dashed line) decreases, unlike training with the non-permuted mask (naive), showing the model trained with the permuted model generalizes better than naive. The dashed (- -) line shows the dense model accuracy. 56
- 4.3 **VGG11 $\times\{1\}$ /CIFAR-10.** Test accuracy of sparse network solutions at increasing rewind points for different sparsity levels. In fig. 4.3b, the permuted solution closely matches the LTH solution. We also see a more noticeable gap between the permuted and naive solutions compared to fig. 4.3a. The dashed (- -) line shows the dense model accuracy. 58
- 4.4 **ResNet50 $\times\{1\}$ /ImageNet.** Top-5 test accuracy vs. rewinds points of sparse network solutions at various sparsity levels. We observe the permuted solution consistently performing better than the naive solution for all sparsities. The dashed (- -) line shows the dense model accuracy. 59

4.5 **LTH solution remains in the same linearly connected mode as the dense solution.** In fig. 4.5a we plot the error barrier between the dense solution and the sparse solution (y -axis) vs the IMP iteration corresponding to the sparse solution (x -axis), for 90% sparsity. We observe that after fixing variance collapse via the REPAIR method, the error barrier between the dense and the sparse solutions remains small, thus showing that the LTH solution remains in the same linearly connected mode as the dense solution. In fig. 4.5b, we demonstrate the same result as fig. 4.5a except we use Iterative Magnitude Pruning - Fine Tuning (IMP-FT) and still see a small error barrier between the dense and pruned solution. 63

4.6 We show that, modulo permutations, reusing the permuted mask leads to convergence in the same mode as the LTH solution. Hence, there is a small loss barrier between the permuted and LTH solutions, demonstrating they are within the same linearly connected mode. Sourced from Ekansh Sharma. 64

-
- 4.7 **Larger width exhibits better Linear-Mode Connectivity (LMC).** Plots showing linear interpolation between $\pi(\mathbf{w}_A^{t=T})$ and $\mathbf{w}_B^{t=T}$ where π was obtained through activation matching between two dense models for varying widths, w . As the width of the model increases, the permutation matching algorithm gets more accurate, thereby reducing the loss barrier (i.e; better LMC), which is evaluated on the test set. This shows that the permutation matching can find a better mapping, π , for wider models, explaining why the permuted mask works better in the case of wider models. 68
- 4.8 **Accuracy vs sparsity trend for ResNet20×{w}/CIFAR-10.** As the width increases, the gap between permuted and naive solutions increases, showing permuted masks help with sparse training. With increased width, we observe a more significant gap seen throughout figs. 4.8d, 4.8h, 4.8l and 4.8p and the permuted solution approaches the LTH solution. The dashed (- -) line shows the dense model accuracy. . . . 76
- 4.9 **Accuracy vs sparsity trend for ResNet20×{w}/CIFAR-100.** Similar to the phenomenon seen in fig. 4.8, with higher width the gap between permuted and naive solutions increases. As seen in figs. 4.9d, 4.9h, 4.9l and 4.9p and the permuted solution approaches the LTH solution. The dashed (- -) line shows the dense model accuracy. 77
- 4.10 **ResNet50×{1}/ImageNet.** Top-1 test accuracy vs rewinds points of sparse network solutions at various sparsity levels. We observe the permuted solution consistently performing better than the naive solution for all sparsities. The dashed (- -) line shows the dense model accuracy. 77

Chapter 1

Introduction

1.1 Overview

In recent years, foundation models have achieved state-of-the-art results for different tasks. However, the exponential increase in the size of state-of-the-art models requires a similarly exponential increase in the memory and computational costs required to train, store and use these models — decreasing the accessibility of these models for researchers and practitioners alike. To overcome this issue, different model compression methods, such as pruning, quantization and knowledge distillation, have been proposed to reduce the model size at different phases of training or inference. Post-training model pruning ([Han et al., 2016](#)) has been shown to be effective in compressing the model size, and seminal works have demonstrated that large models can be pruned after training with minimal loss in accuracy ([Gale et al., 2019](#); [Han et al., 2015](#)). While model pruning makes inference more efficient, it does not reduce the computational cost of training the model.

Motivated by the goal of training a sparse model from a random initial-

ization, [Frankle and Carbin \(2019\)](#) demonstrated that training with a highly sparse mask is possible and proposed the LTH to identify sparse subnetworks that, when trained, can match the performance of a dense model. The key caveat is that a dense model must first be trained to find the sparse mask, which can *only* be used with the same random initialization that was used to train the dense model. Despite LTH seeing significant interest in the research community, LTH masks cannot be used to train from a new random initialization. Furthermore, it has been observed empirically that the LTH is impractical for finding a diverse set of solutions ([Evcı et al., 2022](#)).

This posits our main research questions: *How can we train a LTH mask from a different random initialization while maintaining good generalization? Would doing so find a more diverse set of solutions than observed with the LTH itself?*

In this work, we try to understand why the LTH does not work for different random initializations from a weight-space symmetry perspective. Our hypothesis is that to reuse the LTH winning ticket mask with a different random initialization, the winning ticket mask obtained needs to be permuted such that it aligns with the optimization basin associated with the new random initialization. We illustrate our hypothesis in [fig. 3.3](#). To empirically validate our hypothesis,

- We obtain a sparse mask from a dense model, A , using a pruning technique ([Renda et al., 2020](#); [Han et al., 2015](#)).
- Demonstrate that a permuted sparse mask aligned with a new initialization, B , optimization basin) can be reused for sparse training.
- We showed that the permuted sparse model achieves comparable

generalization to the LTH solution.

- Found that the permuted mask improves generalization compared to a non-permuted mask.
- Observed significantly higher functional diversity compared to LTH solutions.

1.2 Our Contributions

Our contributions are as follows:

1. We hypothesize that the LTH ([Frankle and Carbin, 2019](#)) fails to find good solutions with a new random initialization due to a mismatch between the optimization basin of the winning ticket mask and the new random initialization’s solution basin. We propose a method based on permutation matching between two dense models, that permutes the winning ticket’s sparse mask to align with the optimization basin of the new random initialization. We empirically demonstrate on CIFAR-10/100 and ImageNet datasets using VGG11 and ResNet models of varying widths that permuting the LTH sparse mask to align with the new random initialization improves the performance of the trained model (permuted), compared to the model trained without permuting the sparse mask (naive).
2. We show that models trained from random initialization using the permuted LTH mask are much more functionally diverse in the solutions they learn than those found from training the LTH winning ticket

mask and initialization alone (Evcı et al., 2022), across several existing functional diversity metrics and improved ensemble performance.

3. Furthermore, our experiments provide novel insights about the LTH and the corresponding dense model: we show that for a fixed initialization, the dense solution and the corresponding LTH solution remain in the same loss basin once we take into account *variance collapse*. Notably, our conclusion differs from the conclusion drawn by Paul et al. (2023), where they did not consider the variance collapse issue when interpolating between the sparse and dense solutions.

1.3 Thesis Organization

The thesis is organized in the following sections:

- **Chapter 2: Background and Preliminaries** establishes the foundational concepts for this thesis through a structured three-part exploration. Firstly, it briefly formalizes the standard supervised learning framework within Machine Learning (ML). Secondly, we study the geometric properties of DNN loss landscapes and the concept of mode connectivity, discussing neuronal alignment algorithms and the contribution of permutation symmetries to the non-convexity of these landscapes. Thirdly, we explore sparse neural networks and the lottery ticket hypothesis.
- **Chapter 3: Methodology** details the proposed methodology of this study. This chapter outlines the motivation for addressing our research problem, followed by a visual demonstration designed to build

intuition regarding the relationship between weight symmetries and sparsity. Subsequently, it provides rigorous explanations of our experimental design and the neuronal alignment technique.

- **Chapter 4: Experiments and Results** presents the experiments conducted to validate our primary hypothesis. This chapter details the testing of our hypothesis across a diverse range of models and datasets, followed by additional analyses of the loss landscape and functional diversity.
- **Chapter 5: Conclusion and Future Work** summarizes the key findings of our research and emphasizes the importance of the problem investigated. This chapter also discusses the limitations of the current work and proposes potential future directions for further investigation and improvement.

Chapter 2

Background

2.1 Formal Setup: Machine Learning

In supervised learning, we are given a training dataset of size n , $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ where each training example (\mathbf{x}_i, y_i) is drawn i.i.d. from some unknown underlying probability distribution \mathcal{N} over the input space $X \subseteq \mathbb{R}^m$ and the output space $Y \subseteq \mathbb{R}^k$. Our goal is to learn a function $f : X \rightarrow Y$, parameterized by a weight vector $\mathbf{w} \in \mathbb{R}^d$, such that for a given input $\mathbf{x} \in X$, the prediction $f(\mathbf{x}; \mathbf{w})$ is as close as possible to the true output \mathbf{y} according to the distribution \mathcal{N} .

To formalize this notion of "closeness", we introduce a loss function $l(\mathbf{y}, \hat{\mathbf{y}})$, where \mathbf{y} is the true output and $\hat{\mathbf{y}} = f(\mathbf{x}; \mathbf{w})$ is the predicted output. The loss function $l : Y \times Y \rightarrow \mathbb{R}_{\geq 0}$ measures the discrepancy between the prediction and the ground truth. It should ideally be non-negative, and a lower value indicates a better prediction. The overall learning objective is to minimize the *objective function*, $L(\mathbf{w})$, which is the average of the loss function over

all training examples:

$$L(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^n l(y_i, f(x_i; \mathbf{w})). \quad (2.1)$$

We aim to find the optimal parameters \mathbf{w}^* that minimize this objective:

$$\mathbf{w}^* = \underset{\mathbf{w} \in \mathbb{R}^d}{\operatorname{argmin}} L(\mathbf{w}). \quad (2.2)$$

This optimization is often performed using an iterative algorithm like *gradient descent*, which updates the parameters in the direction opposite to the gradient of the objective function:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \nabla_{\mathbf{w}} L(\mathbf{w}_t), \quad (2.3)$$

where η is the learning rate, controlling the step size of each update.

2.2 Neural Networks

2.2.1 Formal Setup: Feedforward Neural Networks

In our research, the ML models we use for image classification tasks, falls under the umbrella of neural networks. Specifically, we utilize and modify architectures such as Convolutional Neural Networks (CNNs) (LeCun et al., 1989a, 2015) and Residual Neural Networks (ResNets) (He et al., 2016) due to their proven effectiveness in learning complex patterns from high-dimensional data for classification tasks. In this section, we formally define the structure of standard feedforward neural networks with the incorporation of homogeneous nonlinearity within each layer. While a neural network layer performs an affine transformation (a linear operation involving a weighted sum and a bias), it is the subsequent application of a non-linear

activation function that allows the network with the capacity to model and learn intricate, non-linear relationships present in real-world data.

A neural network with L layers can be formally defined as a sequence of transformations applied to an input vector $\mathbf{x} = h^{(0)} \in \mathbb{R}^{n_0}$. Each layer $l \in \{1, 2, \dots, L\}$ performs an affine transformation followed by a non-linear activation function. For the l -th layer, the affine transformation of the input from the previous layer, $h^{(l-1)} \in \mathbb{R}^{n_{l-1}}$, is given by:

$$z^{(l)} = W^{(l)}h^{(l-1)} + b^{(l)}, \quad (2.4)$$

where $W^{(l)} \in \mathbb{R}^{n_l \times n_{l-1}}$ is the weight matrix connecting the $(l-1)$ -th layer to the l -th layer, and $b^{(l)} \in \mathbb{R}^{n_l}$ is the bias vector for the l -th layer. Here, n_{l-1} and n_l denote the number of neurons in the $(l-1)$ -th and l -th layers, respectively. Following the affine transformation, a non-linear activation function $\sigma_l : \mathbb{R} \rightarrow \mathbb{R}$ is applied element-wise to the pre-activation $z^{(l)}$ to produce the output of the l -th layer:

$$h^{(l)} = \sigma_l(z^{(l)}) = \sigma_l(W^{(l)}h^{(l-1)} + b^{(l)}). \quad (2.5)$$

The process starts with the input layer ($l = 1$), where $h^{(0)} = x$. The output of each subsequent layer serves as the input to the next layer. The final layer, $l = L$, produces the output of the neural network, $h^{(L)} \in \mathbb{R}^{n_L}$. The activation function of the final layer, σ_L , is often chosen based on the specific task (e.g., sigmoid or softmax for classification, identity for regression). The entire operation of the neural network can be seen as a function f that maps the input x to the output $h^{(L)}$, parameterized by the set of all weights $W = \{W^{(1)}, W^{(2)}, \dots, W^{(L)}\}$ and biases $b = \{b^{(1)}, b^{(2)}, \dots, b^{(L)}\}$. Therefore,

the overall output of the neural network can be written as:

$$f(x; W, b) = h^{(L)}, \quad (2.6)$$

where $h^{(L)}$ is obtained by iteratively applying the transformations defined above from $l = 1, \dots, L$. We can collectively denote the set of all learnable parameters (weights and biases) as $\theta = \{W^{(1)}, \dots, W^{(L)}, b^{(1)}, \dots, b^{(L)}\}$. In this case, the output of the neural network can be concisely written as:

$$f(\mathbf{x}; \theta) = h^{(L)}. \quad (2.7)$$

In contexts where the distinction between weights and biases is not crucial for the high-level formulation, or when they are treated together as parameters to be learned, we can simply use weights to represent the entire set of learnable parameters θ . Thus, the overall output of the neural network can be expressed as:

$$f(\mathbf{x}; \mathbf{w}) = h^{(L)}. \quad (2.8)$$

This formulation encapsulates the fundamental structure of a feedforward neural network, where information flows in one direction from the input to the output through a series of affine transformations and non-linear activations, encompassed by the network’s learnable parameters \mathbf{w} . Having established the fundamental formulation of fully-connected feedforward neural networks—a foundational architecture that extends to more complex models like CNNs, ResNets, and RNNs—we will now explore the landscape defined by the loss function over the network’s parameter space. Understanding the geometric properties of this loss landscape, a high-dimensional and non-convex space, is crucial for understanding our training dynamics and motivates the core objective of this research: to better design parsimonious deep learning algorithms leveraging weight symmetry.

2.2.2 What is the Loss Landscape?

The loss landscape of a neural network refers to the high-dimensional surface (or more formally, a non-convex manifold) (Choromanska et al., 2015; Li et al., 2018b) formed by evaluating the objective function, $L(\theta)$, across its parameter space. Specifically, it describes the mapping from the space of neural network weights and biases, represented by the parameter vector $\theta \in \mathbb{R}^n$ (where n represents the total number of trainable parameters), to a scalar loss value, typically measuring prediction error on a given dataset, \mathcal{D} . Mathematically, for a fixed training dataset and neural network architecture, the objective function $L(\theta) : \mathbb{R}^n \rightarrow \mathbb{R}$ assigns a scalar loss to each possible parameter vector. The loss landscape is the graph of this function, embedded in \mathbb{R}^{n+1} , where the "height" at any point θ corresponds to the value of $L(\theta)$. Due to the nonlinearity of neural networks and the high dimensionality of parameter spaces, this surface is generally non-convex and non-smooth, particularly in architectures employing ReLU activation functions (Arjevani and Field, 2022; Karhadkar et al., 2024).

2.2.3 Geometry of the Loss Landscape

Training deep neural networks involves minimizing a highly non-convex objective function defined over a high-dimensional parameter space—a task that is generally NP-hard in the worst case (Shapiro and Nemirovski, 2005). Yet, in practice, simple first-order methods such as Stochastic Gradient Descent (SGD) (Robbins and Monro, 1951; Allen-Zhu et al., 2019) reliably find low-loss solutions (i.e. global minimizers with zero or near-zero training loss), even under challenging conditions such as randomized

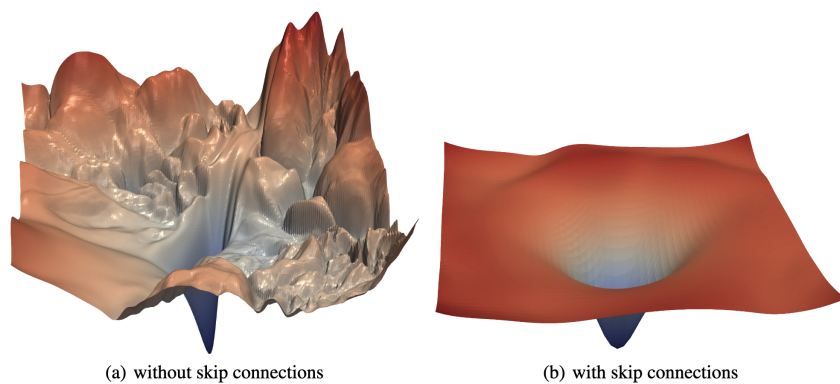


Figure 2.1: An intuitive visualization of the loss landscapes of ResNet56 without (a) and with skip connections (b), showing how architectural design choices affect the notion of sharpness and flatness of loss surfaces. Sourced from [Li et al. \(2018b\)](#).

labels or highly nonlinear architectures ([Hardt et al., 2016](#); [Jin et al., 2017](#); [Zhang et al., 2017](#)). This empirical success suggests that the geometry of the loss landscape—despite its complexity—exhibits structures that favours optimization. A central reason for this apparent paradox lies in **overparameterization**: when the number of network parameters significantly exceeds the number of data points, the loss surface undergoes a dramatic geometric transformation. In particular, when the width of the network layers is sufficiently large—typically at least polynomial in both the number of layers, L , and the number of samples, n , DNN, exhibit optimization-friendly properties such as the disappearance of spurious minima and the emergence of large, geometry-induced connectivity structures ([Arjevani and Field, 2022](#); [Baldassi et al., 2022](#)). In this overparameterized regime, the network has enough degrees of freedom to exactly interpolate the training data, often leading to a proliferation of global minimizers corresponding to different but functionally equivalent parameter configurations. Specifically, in overparameterized neural networks, the abundance of parameters relative to data

gives rise to large connected regions of global minima, symmetry-induced saddle points (Brea et al., 2019), flat plateaus, narrow ravines and low-loss basins, all of which interact with the dynamics of optimization in subtle ways (Simsek et al., 2021; Dauphin et al., 2014; Nguyen and Hein, 2017; Choromanska et al., 2015; Goodfellow and Vinyals, 2015; Brea et al., 2019; Pittorino et al., 2022). We will now formally define the geometric structures of the loss landscape, supported by relevant work.

1. **Global Minima:** A global minimum is any point in the parameter space where the loss function achieves its smallest possible value. In the overparameterized regime, there typically exists a large set of global minima, many of which are related by *permutation symmetries* or reside on continuous manifolds of solutions (Simsek et al., 2021; Cooper, 2018). Simsek et al. (2021) show that in overparameterized feedforward neural networks, permutation symmetries can cause multiple isolated global minima to merge into a single connected manifold when additional neurons are added. Cooper (2018) analytically establishes that, under generic conditions, the set of global minima forms an $(n - d)$ -dimensional submanifold of parameter space. These global minima are functionally equivalent, producing the same output for all training inputs, though potentially with different generalization properties (Lim et al., 2024; Grigsby et al., 2023).
2. **Local Minima:** A local minimum is a point where the loss is lower than at all nearby points, but not necessarily globally minimal. While local minima can be problematic in traditional optimization problems, studies have shown that in deep overparameterized networks the loss

function has no strict local minima under certain regimes (Li et al., 2018a; Karhadkar et al., 2024; Kawaguchi, 2016).

- 3. Saddle Points:** A saddle point is a critical point that is neither a local minimum nor a maximum—i.e., the gradient vanishes, but the Hessian matrix has both positive and negative eigenvalues. Brea et al. (2019) demonstrate that when two neurons in the same layer are permuted (i.e., weights are interchanged), the network passes through a permutation saddle point. These points lie on flat valleys with directions of both descent and ascent. The Hessian at these permutation points exhibits flatness, with multiple zero eigenvalues corresponding to the redundant symmetry directions.
- 4. Plateaus and Flat Regions:** Plateaus are regions of parameter space where the loss remains nearly constant across a large volume. These are closely associated with overparameterization and symmetry. Zhao et al. (2022) emphasize that due to parameter-space symmetries, gradient flow tends to evolve within these low-loss valleys rather than explore isolated optima. Similarly, Pittorino et al. (2022) show that when working in symmetry-reduced coordinates (a toroidal space), one can identify flat regions in the loss landscape.
- 5. Narrow Ravines:** Narrow ravines refer to steep, sharp valleys in the loss surface, where the curvature (i.e., the magnitude of the Hessian's eigenvalues) is large in some directions and small in others. These regions can interfere with optimization by causing instability or gradient vanishing/exploding during training. The literature in this domain between sharp vs. flat minima is highly disputed, particularly when dis-

cussing the relationship between curvature and generalization (Zhao et al., 2023). The common belief is that training neural networks with small batches using SGD results in "flat" minima in the loss landscape, which tend to generalize well. Conversely, large batches are thought to lead to "sharp" minima that generalize poorly (Hochreiter and Schmidhuber, 1997; Keskar et al., 2017; He et al., 2019). However, Dinh et al. (2017); Kawaguchi et al. (2017) argue that the sharpness or flatness of the loss surface might not be the primary factor determining generalization.

6. **Loss Basins:** A *loss basin* loosely refers to a connected region of low loss in the loss landscape, often corresponding to a local or global minimum along with its neighborhood. In light of permutation symmetries, the concept of a basin becomes more complex: what appears as separate basins in the loss landscape may, in fact, be functionally identical because they represent permuted versions of the same network. Building on a conjecture initially proposed by Entezari et al. (2022) and then subsequently adapted by Ainsworth et al. (2023), argue that neural network loss landscapes reduce to a single approximately convex basin *modulo* permutation symmetries, especially in wide networks.

Having outlined the core geometric structures that characterize the loss landscape of DNN, as visualized in Figure 2.1. We now turn to a key property that helps explain many of these phenomena from Section 2.2.3: **permutation symmetries**. In the next section, we explore how these symmetries contribute to overparameterization, create equivalence classes of solutions,

and fundamentally influence the optimization dynamics during neural network training.

2.2.4 Permutation Symmetries in the Loss Landscape

Empirical observations indicate that when independently training DNN with different random initializations and batch orders using stochastic gradient descent (SGD), the resulting training trajectories are often remarkably similar (Ainsworth et al., 2023; Zhang et al., 2017). This consistency is frequently attributed to the overparameterized nature of modern networks, which induces a highly redundant optimization landscape containing a multitude of global minimizers i.e., distinct parameter configurations that represent functionally equivalent solutions (Kawaguchi, 2016; Neyshabur et al., 2017). As first observed by (Hecht-Nielsen, 1990), and further formalized in recent work (Entezari et al., 2022; Simsek et al., 2021; Brea et al., 2019; Godfrey et al., 2022), a feedforward neural network remains functionally unchanged if any two neurons within a hidden layer are swapped and the corresponding incoming and outgoing weights are permuted accordingly. These symmetries mean that the neural network function is invariant under a large group of parameter-space transformations (Zhao et al., 2023), leading to permutation-equivalent minima in the loss landscape. Each of these minima lies at a distinct location in parameter space, but all produce the same outputs for any given input. This introduces additional non-convexity to the optimization problem—not because of isolated bad local minima, but due to the abundance of permutation-induced global minima scattered throughout the parameter space. The presence of these symmetries carves

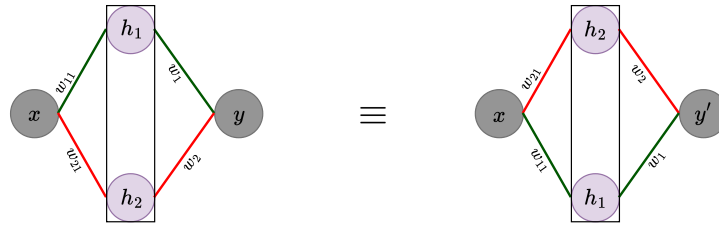


Figure 2.2: Permutation symmetry in a single hidden layer DNN, illustrating that the outputs, y and y' , remain equivalent for the same input despite the swapping of neurons within the hidden layer, thus demonstrating the invariance of the loss function under permutation. For simplicity, we do not include the biases.

out wide, flat valleys or ridges—often with complex topology—in the loss landscape (Grigsby et al., 2023), and plays a central role in explaining why independently trained models can be connected via low-loss interpolation paths after appropriate neuron alignment (Ainsworth et al., 2023; Singh and Jaggi, 2020; Entezari et al., 2022; Tatro et al., 2020).

Conclusion: Neural networks with at least one hidden layer of size greater than one exhibit permutation symmetry as demonstrated in fig. 2.2, meaning that for any given weight configuration θ achieving a particular loss, there exist other distinct weight configurations θ_π , obtained by permuting the neurons within the hidden layer. Under the distinct parameterizations, θ and θ_π , our network learns different functions, but are functionally equivalent and are loss-invariant under permutation. Hence, the existence of permutation symmetries is one of the sources characterizing the non-convexity of the neural network loss landscape, as it creates copies of global minima at different points in weight space. As stated by Brea et al. (2019), in a DNN of $\ell - 1$ hidden layers each with n neurons, any given global

minimum in the loss landscape has,

$$(n!)^{\ell-1} - 1, \tag{2.9}$$

completely equivalent minima due to permutation symmetries.

2.2.5 Permutation Invariance and Mode Connectivity

SGD has shown remarkable performance in training DNN and finding good solutions despite operating in such highly non-convex loss landscapes. Having studied how permutation symmetries contribute to the non-convexity of the networks’ loss landscapes, we will now explore how this property of *permutation invariance* underlies recent observations about mode connectivity in independently SGD trained DNN.

2.2.5.1 What is Mode Connectivity?

Although linearly interpolating between the parameters of two DNN trained independently with SGD typically traverses regions of significantly higher loss or 0–1 classification error, exceptions have been observed. Several studies demonstrate that, contrary to the expectation for non-convex systems, SGD tends to produce basins of attraction that are surprisingly “convex-like” in practice (Nagarajan and Kolter, 2019; Wortsman et al., 2021). In particular, interpolating between models along the same training trajectory often produces monotonic decreases in error, suggesting the absence of *loss barriers* (Goodfellow and Vinyals, 2015; Vlaar and Frankle, 2022; Lucas et al., 2021). Building on these geometric intuitions, Draxler et al. (2018) and Garipov et al. (2018) independently demonstrated that even DNN trained from different random initializations—which converge to different points in

weight space—can often be connected by nonlinear low-loss curves, giving rise to the concept of **mode connectivity**. These curves relax the assumption of linearity and are constructed through heuristic or optimization-based methods that “bend” through weight space to avoid high-loss regions. (Informal definition: two DNN θ_1 and θ_2 are *mode connected* if there exists a path between them along which the loss barrier height ≈ 0 .) This observation conjectures that the loss landscape is not composed of isolated minima, but as forming a connected manifold. The emergence of such connected structures in the loss landscape leads naturally to the proposal that the apparent disconnectedness of independently trained solutions via SGD may be largely due to permutation symmetries. We will now examine recent breakthroughs that conjecture that by appropriately accounting for permutation symmetries, it can be shown that multiple solutions reside within the same functional loss basin, connected by linear paths with low loss. This observation has led to the concept of *linear mode connectivity* (i.e., LMC).

2.2.5.2 Linear Mode Connectivity (LMC)

A pair of trained neural networks are said to be linearly connected if the loss along the linear path between the models remains small. The phenomenon of linear (mode) connectivity was first observed in the context of SGD by [Nagarajan and Kolter \(2019\)](#), where they showed that two DNN trained from the same initialization but with different data orders exhibit linear connectivity. The term LMC was introduced by [Frankle et al. \(2020b\)](#), where they showed that independently trained SGD DNN can be connected via a linear path.

2.2.5.3 LMC *modulo* Permutation

Entezari et al. (2022) further observed that while a model and its randomly permuted counterpart are functionally equivalent, they are rarely linearly connected in the weight space. This misalignment suggests the presence of *loss barriers*—regions along a linear path between models where the loss is significantly higher than at the endpoints. They conjectured that independently obtained SGD solutions exhibit no loss barrier when accounting for permutation symmetries, suggesting that all SGD-trained networks converge to a single basin modulo permutations.

More formally, let θ_1, θ_2 be the parameters of two networks, then the loss barrier \mathcal{B} is defined as:

$$\mathcal{B}(\theta_1, \theta_2) := \sup_{\alpha \in [0,1]} \left[\mathcal{L}((1-\alpha)\theta_1 + \alpha\theta_2) - ((1-\alpha)\mathcal{L}(\theta_1) + \alpha\mathcal{L}(\theta_2)) \right],$$

where \mathcal{L} is the loss function evaluated on the training dataset. If $\mathcal{B}(\theta_1, \theta_2) \approx 0$, it is said that θ_1 and θ_2 are linearly mode connected. Definition 2.10 adapted from (Frankle et al., 2020b), is more suitable as it does not assign a barrier value to a loss that changes linearly between θ_1 and θ_2 .

Enterzari et al. Conjecture

Let $f(\theta)$ be the function representing a feedforward network with parameters $\theta \in \mathbb{R}^k$, \mathcal{P} be the set of all valid permutations for the network, $\mathcal{P} : \mathbb{R}^k \times \mathcal{P} \rightarrow \mathbb{R}^k$ be the function that applies a given permutation to parameters and returns the permuted version, and $\mathcal{B}(\cdot, \cdot)$ be the function that returns barrier value between two solutions as defined in Equation 2.10. Then, there exists a width $h > 0$ such that for any network $f(\theta)$ of width at least h the following holds:

There exist a set of solutions $S \subseteq \mathbb{R}^k$ and a function $Q : S \rightarrow \mathcal{P}$ such that for any $\theta_1, \theta_2 \in S$, $\mathcal{B}(\mathcal{P}(Q(\theta_1), \theta_1), \theta_2) \approx 0$ and with high probability over an SGD solution θ , we have $\theta \in S$.

Explanation:

Functionally identical neural networks can have their neurons reordered due to permutation invariance as established in 2.2.4. If this permutation invariance is not accounted for, a linear interpolation directly between their weights can traverse regions of high loss in the parameter space. This may incorrectly lead to the conclusion that the two solutions are separated by a significant barrier. However, if the neurons in one model are permuted to match the structure of the other—a process known *neuronal alignment*—the two endpoints may lie in a smoothly connected region of the landscape. In this aligned space, the models can be linearly interpolated without incurring a significant increase in loss, revealing what appears to be a convex basin that contains *all possible* SGD solutions differing only by permutation. This conjecture theoretically aims to simplify the complexity of our loss landscape such that rather than having many isolated basins corresponding to different local minima, the actual landscape may consist of a single basin, modulo permutations.

2.2.5.4 Aligning Neural Networks via Permutation

Building on this conjecture, several algorithms have been developed to address permutation invariance by aligning trained networks to the same optimization basin (Ainsworth et al., 2023; Jordan et al., 2023; Singh and

Jaggi, 2020; Tatro et al., 2020). They focus on improving particular neuronal alignment techniques used to bring the hidden units of two networks into alignment, to reduce the loss barrier upon interpolation between the models in a symmetry-aware manner.

Tatro et al. (2020) introduced a neuron alignment method based on aligning neurons within hidden layers by comparing their activation distributions on a shared dataset. This alignment is achieved by minimizing the difference in the empirical second-order statistics (i.e. means and covariances) of these activations, effectively pairing neurons that exhibit similar behaviour during forward passes. They integrate this neuron alignment step into a piecewise-linear interpolation procedure to discover a low-loss path between pre-trained models. Notably, their work demonstrates, both theoretically and empirically, that this method significantly reduces the interpolation barrier without requiring explicit determination of the true optimal permutation. This approach offers computational efficiency by leveraging activation statistics as a proxy for neuron correspondence, thereby avoiding the need to solve a computationally intensive combinatorial optimization problem directly.

Building on this research, Singh and Jaggi (2020) addressed the alignment problem by framing it as a discrete optimal transport task. This method treats neurons in one network as “mass distributions” and matches them to neurons in another network using a cost matrix that quantifies their dissimilarity, typically based on activation patterns, weight differences, or a combination of both. The resulting optimal transport plan determines the best permutation (or a more flexible soft matching) of neurons to minimize the total misalignment cost. The authors explored various cost functions

designed to break symmetries and rigorously evaluated how these choices affect the quality of interpolation between models and their performance in downstream tasks such as ensembling and functional averaging. This optimal transport approach provides a versatile and theoretically sound alignment strategy that outperforms simpler matching techniques, particularly in wide networks with significant permutation redundancy, and demonstrates better generalization across different DNNs and datasets.

A few years later, [Ainsworth et al. \(2023\)](#) made a significant advancement showing that by adapting the conjecture of [Entezari et al. \(2022\)](#); that two models trained from different random initializations find solutions within the same basin modulo permutation symmetries. They introduced a set of “rebasining” greedy algorithms designed to find permutations aligning the hidden units of independently trained SGD models in weight space. When such alignment is achieved, the models could be linearly interpolated without encountering a significant loss barrier, effectively merging them into a *single* basin. They showcased this with near-zero loss barriers between wide ResNets trained on CIFAR-10 by using Layer Normalization instead of standard Batch Normalization.

A rigorous study from [Sharma et al. \(2024\)](#) introduced the notion of *simultaneous weak linear connectivity*, demonstrating that a single permutation, π , capable of aligning two networks can also simultaneously align larger, fully trained networks throughout the entire SGD training trajectory. Furthermore, this same π aligns successive iterations of independently sparsified networks obtained via weight rewinding. The authors further investigated the recurring conjecture that *all SGD solutions are linearly connected modulo permutation*. While prior work formalized the notion of

“weak linear connectivity”—where a single permutation is guaranteed to align a pair of DNNs either after training (Ainsworth et al., 2023; Guerrero-Peña et al., 2022) or at initialization (Benzing et al., 2022). They provided the first evidence towards Strong Linear Connectivity Modulo Permutation (SLMC), where the single permutation is sufficient to mutually align all network pairs in set, in particular for networks with large width. Additionally, Sharma et al. (2024) also showed that for certain neural networks, sparse masks obtained via weight rewinding can be effectively reused across different instances of the same DNN architecture, provided the neuron order is accounted for through permutations, without significantly impacting test performance. Finally, the study offered further insights into neuronal alignment techniques, comparing weight and activation matching, and generally highlighting the superior performance of matching activations between DNNs.

2.2.5.5 Aligning Neural Networks: Weights

The goal of Weight Matching (WM) is to find a permutation of neurons in one model that aligns it with another network minimizing the L_2 distance between the two models. Hence, enabling interpolated models to maintain a low loss barrier. This problem, which is NP-hard, involves a discrete optimization over the extensive space of possible neuron permutations, leading to a significant computational challenge, especially for larger networks. However, the important contribution of (Ainsworth et al., 2023) proposed a method to make WM computationally tractable. They achieve an approximate solution in polynomial time by decoupling the optimization for each layer separately and reducing the per-layer alignment to the well-known

(Linear Assignment Problem (LAP)). We now revisit the formulation of permutation symmetries, as detailed in Appendix section 5.2, and extend it to an L -layer Multi Layer Perceptron (MLP) in order to establish the structure of the WM problem.

Formally: Consider an L layer MLP, with input feature matrix $\mathbf{x} = \mathbf{Z}^{(0)}$ and output $f(\mathbf{x}) = \mathbf{Z}^{(L)}$ where activation of layer $l \in [L]$,

$$\mathbf{Z}^{(l)} = \sigma(\mathbf{Z}^{(l-1)}(\mathbf{W}^{(l)})^\top + \mathbf{b}^{(l)}). \quad (2.10)$$

Let $\mathbf{P} \in \mathcal{S}_{d_l}$ be a permutation matrix that permutes the output features of an intermediate layer l , where d_l is the number of output features of layer l . All permutation matrices are orthogonal, such that $\mathbf{P}^\top = \mathbf{P}^{-1}$.

Now consider the activation of layer $l + 1$,

$$\begin{aligned} \mathbf{Z}^{(l+1)} &= \sigma(\mathbf{Z}^{(l)}(\mathbf{W}^{(l+1)})^\top + \mathbf{b}^{(l+1)}) \\ &= \sigma(\sigma(\mathbf{Z}^{(l-1)}(\mathbf{W}^{(l)})^\top + \mathbf{b}^{(l)})(\mathbf{W}^{(l+1)})^\top + \mathbf{b}^{(l+1)}). \end{aligned} \quad (2.11)$$

For layers l to $l + 1$, we have these mappings:

$$\begin{aligned} \mathbf{W}^{(l)'} &\mapsto \mathbf{P}^\top \mathbf{W}^{(l)} \\ \mathbf{b}^{(l)'} &\mapsto \mathbf{b}^{(l)} \mathbf{P} \\ \mathbf{W}^{(l+1)'} &\mapsto \mathbf{W}^{(l+1)} \mathbf{P} \\ \mathbf{b}^{(l+1)'} &\mapsto \mathbf{b}^{(l+1)} \end{aligned} \quad (2.12)$$

We now want to show that if we apply the permutation mappings from 2.12 to the weights and biases in Equation 2.11 the activation of layer $l + 1$ remains invariant.

Let $\mathbf{Z}^{(l+1)'}$ denote the output of layer $l + 1$ after the network's parameters

have been acted upon by the permutation, \mathbf{P} .

$$\begin{aligned}
\mathbf{Z}^{(l+1)'} &= \sigma(\sigma(\mathbf{Z}^{(l-1)}(\mathbf{W}^{(l)'})^\top + \mathbf{b}^{(l)'}) (\mathbf{W}^{(l+1)'})^\top + \mathbf{b}^{(l+1)'}) \\
&= \sigma(\sigma(\mathbf{Z}^{(l-1)}(\mathbf{P}^\top \mathbf{W}^{(l)})^\top + \mathbf{b}^{(l)} \mathbf{P}) (\mathbf{W}^{(l+1)} \mathbf{P})^\top + \mathbf{b}^{(l+1)}) \\
&= \sigma(\sigma(\mathbf{Z}^{(l-1)}(\mathbf{W}^{(l)})^\top \mathbf{P} + \mathbf{b}^{(l)} \mathbf{P}) \mathbf{P}^\top (\mathbf{W}^{(l+1)})^\top + \mathbf{b}^{(l+1)}) \\
&= \sigma(\sigma((\mathbf{Z}^{(l-1)}(\mathbf{W}^{(l)})^\top + \mathbf{b}^{(l)}) \mathbf{P}) \mathbf{P}^\top (\mathbf{W}^{(l+1)})^\top + \mathbf{b}^{(l+1)}) \quad (2.13) \\
&= \sigma(\sigma(\mathbf{Z}^{(l-1)}(\mathbf{W}^{(l)})^\top + \mathbf{b}^{(l)}) (\mathbf{P} \mathbf{P}^\top) (\mathbf{W}^{(l+1)})^\top + \mathbf{b}^{(l+1)}) \\
&= \sigma(\sigma(\mathbf{Z}^{(l-1)}(\mathbf{W}^{(l)})^\top + \mathbf{b}^{(l)}) (\mathbf{W}^{(l+1)})^\top + \mathbf{b}^{(l+1)}) \\
&= \mathbf{Z}^{(l+1)}
\end{aligned}$$

Hence, for any layer $l \in [L]$ the activation output $\mathbf{Z}^{(l)}$ remains invariant once acted upon by the permutation, \mathbf{P} .

Let the tuple of permutations corresponding to each layer as $\pi = (\mathbf{P}^{(l)})_{l \in [L]}$.

Consider the weights of two, independently trained SGD models of equivalent architectural settings but different data orders and initializations, A and B , with weights Θ_A and Θ_B , respectively. The WM problem aims to find a π that minimizes the L_2 distance between Θ_A and $\pi(\Theta_B)$:

$$\arg \min_{\pi} \|\Theta_A - \pi(\Theta_B)\|^2 = \arg \min_{\pi} \sum_{l \in [L]} \|\mathbf{W}_A^{(l)} - \mathbf{P}^{(l)} \mathbf{W}_B^{(l)} (\mathbf{P}^{(l-1)})^\top\|^2, \quad (2.14)$$

where $\mathbf{P}^{(0)} = \mathbf{I}$, $\mathbf{P}^{(L)} = \mathbf{I}$. In Equation 2.14, we focused on the weights for simplicity; however, biases can also be incorporated into the formulation. For our study, we did not utilize WM as (Sharma et al., 2024) found a disconnect between the WM objective and the loss barrier early in training. They hypothesize this might be caused by the inherent noisiness and the small magnitude of weights during the initial training phase.

2.2.5.6 Aligning Neural Networks: Activations

The use of Activation Matching (AM) for model alignment was originally introduced by [Li et al. \(2015\)](#), to ensure models learn similar representations when performing the same task. Their proposed method involved maximizing the sum of correlations between the activations of paired neurons across a batch of training data. The intuition behind this approach is that if two neurons in different networks exhibit highly correlated activations for the same inputs, they are likely performing similar functions within their respective networks.

Formally: According to [Ainsworth et al. \(2023\)](#), they believe a linear relationship may exist between the activations of the two models. Given activations for each model, they aim to associate each unit A with a unit in B . They associate the units across the two models by fitting it into a regression framework, constituting it into a LAP. AM tries to find a permutation mapping, $\pi \in S_{d_l}$ (where S_{d_l} is the permutation group of order $d_l!$) such that by permuting the parameters of the second model, the correlation between the activations of the two models is maximized. For a model consisting of L layers, each layer is sequentially matched and permuted starting from the input layer. Let $Z_l^A, Z_l^B \in \mathbb{R}^{d \times n}$ be the activations of layer l of model A and B respectively obtained using the training data, where d represents the dimensionality of the activations at layer l and n is the number of training data points. Then a permutation mapping for layer l , π_l , is obtained by solving:

$$\begin{aligned} \pi_l &= \arg \min_{\pi} \|Z_l^B - \pi Z_l^A\| \\ &= \arg \max_{\pi} \langle \pi, Z_l^B (Z_l^A)^\top \rangle_F \end{aligned} \tag{2.15}$$

where $\langle \cdot, \cdot \rangle_F$ denotes the Frobenius inner product. eq. (2.15) can be formulated as a LAP (Bertsekas, 1998; Ito et al., 2024) solved via the Hungarian algorithm (Kuhn, 2010); however, the permutation found is not a global optima, but a greedy/approximate solution as permutation matching is a NP-hard problem. Once the permutation mapping is obtained for all the layers, the model A can be permuted to match the model B . To ensure that the permuted model does not change functionally when permuting the output dimension of layer l , the input dimension of the next layer is also permuted accordingly. Let $\mathbf{W}^{(l)}$ and $\mathbf{b}^{(l)}$ be the weights and bias of layer l respectively, then the permuted weight matrix $\mathbf{W}^{(l)'}$ and permuted bias $\mathbf{b}^{(l)'}$ for each layer can be mathematically represented as,

$$\mathbf{W}^{(l)'} = \pi_l \mathbf{W}^{(l)} (\pi_{l-1})^\top, \quad \mathbf{b}^{(l)'} = \pi_l \mathbf{b}^{(l)}. \quad (2.16)$$

Based on the findings of Sharma et al. (2024), which demonstrated its superior effectiveness for neuronal alignment, we use activation matching to find a permutation between two independently SGD trained DNN in our methodology. Furthermore, we enhanced the interpolation process by resetting the normalization statistics, allowing for the effective use of Batch Normalization as suggested by Jordan et al. (2023).

2.2.5.7 REPAIR: Renormalizing Permuted Activations

Jordan et al. (2023) investigated the poor performance of interpolated networks with Batch Normalization, attributing it to a phenomenon they termed ‘‘Variance Collapse (VC)’. They observed that when two independently SGD trained DNN are interpolated in weight space—even after optimizing alignment through permutation matching—the resulting interpolated networks

often suffer a sharp degradation in accuracy and generalization. Their analysis revealed that this degradation often stems not from misalignment in matching, but from the collapse of activation variance in the interpolated model, progressively with deeper layers into the model. Their technique known as "REPAIR: RENormalizing Permuted Activations for Interpolation Repair" involves rescaling and shifting the activations to align with the average statistics of the individual expert models.

Formally: Given an interpolated model, $\theta_\alpha = (1 - \alpha)\theta_A + \alpha\theta_B$ for some $0 < \alpha < 1$. Let $X_{\theta_A}^{(k)}, X_{\theta_B}^{(k)}$ denote the channel activations of the k^{th} module of the endpoints θ_A, θ_B , respectively, can be thought of as random variables over the training data. They propose a rescaling transformation applied to the hidden units of the merged model, θ_α , to restore activation variance:

$$\begin{aligned}\mathbb{E}[X_{\theta_\alpha}^{(k)}] &= (1 - \alpha) \cdot \mathbb{E}[X_{\theta_A}^{(k)}] + \alpha \cdot \mathbb{E}[X_{\theta_B}^{(k)}], \\ \text{std}[X_{\theta_\alpha}^{(k)}] &= (1 - \alpha) \cdot \text{std}[X_{\theta_A}^{(k)}] + \alpha \cdot \text{std}[X_{\theta_B}^{(k)}].\end{aligned}\tag{2.17}$$

Prior to applying the rescaling and shifting procedure detailed in 2.17, the VC phenomenon resulted in the standard deviation of the activations in the interpolated model, $\text{std}(X_{\theta_\alpha}^{(k)})$, being significantly smaller than the minimum of the standard deviations of the activations in the individual models, $\min(\text{std}(X_{\theta_A}^{(k)}), \text{std}(X_{\theta_B}^{(k)}))$. In this study, we adapt the AM algorithm, as implemented by [Jordan et al. \(2023\)](#), to be generalizable for all VGG and ResNet models. Throughout our methodology in 3, we correct the activation statistics using the REPAIR method, demonstrating improved performance when studying loss barriers and LMC between networks.

2.3 Sparsity in Neural Networks

2.3.1 What are Sparse Neural Networks?

Sparse Neural Network (SNN) are a subset of DNNs in which many of the weights or connections between neurons are set to zero, reducing the number of active parameters compared to standard dense networks. The primary motivation for introducing sparsity stems from computational efficiency: sparse models require less memory and can reduce the computational cost of training and inference, which is especially important in resource-constrained environments. With the rapid advances in Artificial Intelligence (AI), the escalating scale and complexity of DNNs is fueling state-of-the-art models to grow from millions, to billions and even trillions of parameters (Cheng et al., 2023; Hoefler et al., 2021; Mostafa and Wang, 2019; Ma and Niu, 2018; Cerebras, 2022). This vast over-parameterization leads to substantial computational costs, memory storage, and energy consumption demands during training and inference phases (Hunter et al., 2021; Ye et al., 2024; Cheng et al., 2023). However, in these overparameterized regimes many of these parameters become superfluous, which made researchers question: if a significant portion of these parameters are indeed superfluous, how can we effectively identify and remove them without compromising the generalization performance achieved by their dense counterparts?

2.3.1.1 Sparsity and the Brain

The concept of sparsity in neural networks has deep historical roots, beginning with early brain-inspired models. Neurons in the brain exhibit

both sparse interconnectivity and sparse activity patterns. Studies indicate that local connection densities in the cortex can be less than 5% (Hunter et al., 2021), and only a small percentage of neurons activate in response to stimuli. This biological sparsity is believed to be intrinsically linked to the brain's remarkable energy efficiency and its capacity for learning and memory (Hoefler et al., 2021). This biological inspiration of sparsity in the brain provided a key motivation for early network pruning techniques like Optimal Brain Damage (OBD), which aimed to selectively remove less important connections.

2.3.1.2 Optimal Brain Damage

The primary motivation behind OBD was to address the problem of overfitting in neural networks. OBD aimed to find optimal balance between network complexity and the error on the training set by selectively removing weights (LeCun et al., 1989b). The core mechanism of OBD involves calculating "saliency" for each weight in the network. Saliency, quantifies the predicted increase in the training error if that specific weight were removed (set to zero). Weights with the lowest saliency are categorized as least important and become candidates for pruning. To make this calculation feasible without actually removing and retraining for each weight, OBD utilizes a second-order Taylor series expansion of the error function around a presumed minimum point reached after training. To avoid the computational cost of calculating the full Hessian matrix, OBD approximates it as a diagonal matrix.

Formally: Under this assumption, the saliency S_k of weight w_k simplifies to

$$S_k = \frac{1}{2} h_{kk} w_k^2, \quad (2.18)$$

where h_{kk} is the diagonal element of the Hessian. The OBD procedure:

1. Choose a reasonable network architecture.
2. Train the network to convergence.
3. Compute the second derivatives h_{kk} for each parameter.
4. Calculate the saliency S_k for each weight based on 2.18.
5. Prune the weights with the lowest saliency.
6. Retrain the pruned network and repeat from Step 2.

2.3.1.3 Optimal Brain Surgeon

Recognizing the limitations proposed by [LeCun et al. \(1989b\)](#) and [Hassibi and Stork \(1992\)](#), they argued that the diagonal approximation in OBD was often inaccurate, as Hessians in practice tend to be strongly non-diagonal, implying OBD could potentially be removing important weights. Instead, they proposed Optimal Brain Surgeon (OBS) which utilizes information from the **full** Hessian matrix, using efficient recursive methods to calculate the inverse, making the approach scalable to larger networks.

Formally: They calculate the saliency L_q of a weight w_q as:

$$L_q = \frac{w_q^2}{2[H^{-1}]_{qq}}, \quad (2.19)$$

where $[H^{-1}]_{qq}$ is the q^{th} diagonal element of the inverse Hessian. The important advantage of OBS over OBD is its mechanism for adjusting the remaining weights. When a weight w_q is pruned, OBS calculates the optimal

update δw for all other weights in the network to minimize the increase in error caused by removing w_q .

$$\delta w = -\frac{w_q}{[H^{-1}]_{qq}} H^{-1} \cdot \mathbf{e}_q, \quad (2.20)$$

where \mathbf{e}_q is the unit vector corresponding to weight w_q . The OBS procedure:

1. Choose a reasonable network architecture.
2. Train the network to convergence.
3. Compute H^{-1} .
4. Calculate saliencies L_q based on 2.19.
5. Identify the weight with the minimum saliency.
6. Prune this corresponding weight and adjust the remaining weights using 2.20.
7. Repeat steps 3-5 iteratively.

2.3.2 How to Prune Deep Neural Networks?

The foundational work on OBD and OBS served as a stepping stone into the field of neural network pruning. Decades later, Han et al. (2016) demonstrated the efficacy of post-training model pruning in compressing model size, and numerous seminal works have shown that large models can be pruned after training while maintaining similar performance to their dense counterparts (Gale et al., 2019; Han et al., 2015). While model pruning makes inference more efficient, it does not reduce the computational cost of training the model. This is because the pruning process typically occurs after the model has been fully trained to convergence. The computational resources for training are determined by the initial, dense network architecture.

2.3.2.1 Unstructured Pruning

In our research, we use Unstructured Pruning (UP); it is a technique that sparsifies DNN by individually setting parameters (i.e. weights) to zero according to a saliency criterion that measures the importance of each weight. This results in a sparse weight matrix or tensor with an irregular, non-predetermined distribution of zero and non-zero elements (Cheng et al., 2023). UP offers maximum flexibility—by targeting individual weights, it has the potential to remove only the truly unimportant parameters, allowing for higher levels of sparsity to be achieved, by more than 90% (Li et al., 2016; Han et al., 2015), while preserving model accuracy compared to structured approaches (Hoefler et al., 2021). However, this flexibility in pruning regime comes at the cost of hardware compatibility. The irregular sparse matrices generated by UP are notoriously difficult to accelerate efficiently on conventional hardware designed for dense computations. Achieving practical speed ups often requires specialized sparse matrix libraries or dedicated hardware support, which are not universally available or standardized (Yao et al., 2019).

2.3.2.2 Iterative Magnitude Pruning

Magnitude-based pruning stands out as the most prevalent and conceptually straightforward method within unstructured pruning. Its primary saliency criterion is the L_2 norm, where weights with smaller absolute magnitudes are categorized as less significant to the network’s function and are thus targeted for removal (Blalock et al., 2020; Hoefler et al., 2021). Han et al. (2015) played a pivotal role in revitalizing interest in pruning modern

DNN. Their work demonstrated that state-of-the-art models of that era, such as AlexNet (Krizhevsky et al., 2012) and VGG16, could achieve substantial reductions in parameter count (ranging from 9× to 13× through pruning alone) by employing a technique known as IMP in conjunction with retraining (fine-tuning), IMP-FT. This work established the now-standard "train, prune, fine-tune" pipeline and showed the feasibility of drastic compression on complex models (Renda et al., 2020).

Definitions: We define a neural network architecture as a function family $f(\cdot; \cdot)$. A neural network model is a particular parameterization of an architecture, denoted as $f(x; W)$ for specific parameters W . Neural network pruning takes as input a model $f(x; W)$ and produces a new model $f(x; M \odot W')$. Here, W' is the set of parameters of the pruned network (which may be different from W after retraining), and $M \in \{0, 1\}^{|W|}$ is a binary mask that fixes certain parameters to 0. The symbol \odot represents the element-wise product operator.

General IMP-FT Procedure: Given a pre-trained dense neural network model, $f(x; W_0)$, with initial weights W_0 , a target sparsity level S (or a number of pruning iterations N), and a pruning rate ρ , the IMP-FT can be formally described as follows:

1. Initialize the pruning mask M_0 as a binary tensor of the same shape as W_0 , with all elements set to 1 ($M_0 = \mathbf{1}$).
2. Set the current weights W_0 .
3. For $t = 0$ to $N - 1$ (or until the desired sparsity is reached):

- a) Calculate the magnitude of each weight in the current weight tensor W_t .
 - b) Determine the set of weights \mathcal{P}_t to prune, which are the $\rho \times 100\%$ of the weights in W_t with the smallest magnitudes that are not already pruned.
 - c) Create a new pruning mask M_{t+1} by setting the elements corresponding to the weights in \mathcal{P}_t to 0 in M_t .
 - d) Apply the pruning mask to the current weights to obtain the pruned weight tensor: $W'_{t+1} = M_{t+1} \odot W_t$.
 - e) Fine-tune the pruned model $f(x; W'_{t+1})$ on the training dataset for a certain number of epochs. The updated weights after fine-tuning are denoted as W_{t+1} .
4. The final pruned model is $f(x; W'_N)$ with the corresponding sparse mask M_N .

At each iteration t , the model with weights W_t is pruned according to the magnitude of its weights, resulting in a masked weight tensor W'_{t+1} . This pruned model is then fine-tuned to obtain W_{t+1} for the next pruning iteration. The process continues until the desired sparsity level is achieved or the maximum number of iterations is reached.

2.3.3 What is Sparse Training?

The goal of SNN training algorithms is to train a DNN with a reduced number of active weights, either from the beginning or throughout the training process, in a way that offers comparable or superior performance compared

to the dense models. At its core, the sparse training problem concerns how to effectively train a DNN in a such a way that it maintains sparse connectivity throughout the training process, while achieving strong generalization and performance. This raises several challenges:

1. **Optimization under Sparse Constraints:** Sparsity introduces discontinuities in the loss landscape where masking projects the function into a lower-dimensional subspace. This can negatively affect gradient flow and training dynamics. Networks with sparse connections are more susceptible to poor local minima, vanishing gradients, or optimization instability during training making it more difficult for parameters to co-adapt during training (Wang et al., 2020; Lee et al., 2019; Evci et al., 2022).
2. **Identifying and Selecting Sparse Structures:** Sparse training requires deciding which weights or connections to retrain and which ones to remove. Post-training pruning: where sparsity is introduced by removing parameters from a densely trained DNN. Pruning at Initialization (PaI): where a binary mask is applied before training starts, often using gradients or sensitivity estimates as a saliency criterion (Frankle et al., 2020a; Tanaka et al., 2020).

2.3.3.1 Sparse Initialization: Challenges

One of the fundamental questions in SNN training is whether it is possible to train a sparse model directly from scratch, without instantiating a fully dense network, while achieving strong generalization performance and convergence guarantees. This goal motivates research directions focused

on determining sparse initialization strategies: how can we initialize SNN in a way that is effectively able to learn? SNN trained from scratch must be initialized with a fixed connectivity structure. These structures are often designed based on insights from graph theory or empirical heuristics:

- **Erdős–Rényi (ER) topology:** Connectivity is determined by randomly sampling edges with a fixed probability (Hoefler et al., 2021). The resulting sparse graph is uniformly random, ensuring a standard level of connectivity. However, ER graphs may suffer from inadequate connectivity for gradient propagation, especially in larger networks.
- **Layer-wise heuristics:** Some sparse initialization strategies employ varying connectivity distributions across network layers, often preserving a higher density of connections in the initial layers to enhance early learning signal propagation (Lee et al., 2019).
- **Gradient flow:** The vast majority of works aiming to train SNN from scratch have used standard initialization techniques meant for dense DNN (Glorot and Bengio, 2010; He et al., 2015). Evcı et al. (2022) demonstrated that SNN exhibit poor gradient flow at initialization due to the inadequacy of these current initialization techniques, which do not consider heterogeneous connectivity. They develop a sparsity-aware initialization that improves gradient flow and training.

While these designs offer flexibility in constructing sparse architectures, their performance often depends on the initialization and capacity to preserve signal propagation—which becomes an important part of our study and discussion in the following section.

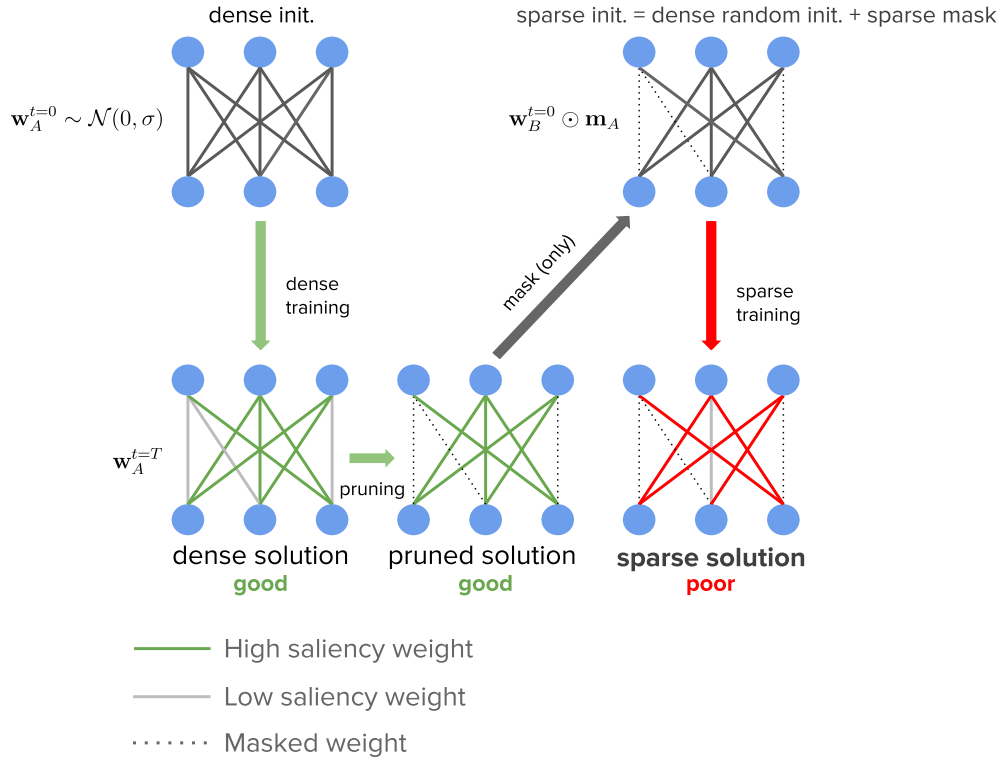


Figure 2.3: Visualization of the sparse training problem paradigm (highlighted in red). Training a fixed mask, \mathbf{m}_A , obtained from a parent initialization, $\mathbf{w}_A^{t=0}$, does not perform well when equipped with a random, arbitrary initialization. Sourced from [Ioannou \(2022\)](#).

2.3.3.2 The Sparse Training Problem

Even if a sparse topology has desirable structural properties, the fundamental issue still arises whether any arbitrary random initialization (not derived from training a dense DNN) equipped with a good sparse mask is sufficient to obtain strong generalization performance. Consider a random instantiation of some DNN, $\mathbf{w}_A^{t=0} \sim \mathcal{N}(0, \sigma)$. We train to convergence, to obtain, $\mathbf{w}_A^{t=T}$ and then post-train prune via IMP to a pruned solution, \mathbf{m}_A . Now, consider training another random instantiation $\mathbf{w}_B^{t=0} \sim \mathcal{N}(0, \sigma)$ equipped with \mathbf{m}_A at a fixed sparsity; results in poor generalization performance.

2.3.4 Lottery Ticket Hypothesis

In 2019, [Frankle and Carbin \(2019\)](#) proposed the LTH to attempt to solve the sparse training problem by re-using the same initialization as used to train the pruned models. They conjectured, “*a randomly-initialized, dense DNN contains a subnetwork that is initialized such that—when trained in isolation—it can match the test accuracy of the original network after training for at most the same number of iterations.*”

Formally: Consider a dense DNN, $f(x; \theta)$ with initial parameters $\theta = \theta_0 \sim \mathcal{N}$. We train f on the training set with SGD to reach minimal validation loss l at iteration i with test accuracy a . Consider training, $f(x; \mathbf{m} \odot \theta)$ with a binary mask $\mathbf{m} \in \{0, 1\}^{|\theta|}$ on its parameters such that its initialization is $\mathbf{m} \odot \theta_0$. Training with SGD, keeping \mathbf{m} fixed, f reaches minimum validation loss l' at iteration i' with test accuracy a' . The LTH predicts that \exists a binary mask, \mathbf{m} , such that for some iteration $i' \leq i$, a subnetwork with parameters $\mathbf{m} \odot \theta$ achieves on-par or better generalization than the original dense DNN. We illustrate this methodology in 2.4.

[Frankle and Carbin \(2019\)](#) employed IMP to identify trainable subnetworks, termed *winning tickets*, with parameters $f(x; \mathbf{m} \odot \theta_0)$. Their study yielded two key observations:

- For relatively small models and simple datasets such as MNIST and CIFAR-10, initializing a network with the weights of a winning ticket allows training to maintain the generalization performance of the pruned model, demonstrating the possibility of effective training even with highly sparse masks.
- However, when a model with the same sparse mask is randomly re-

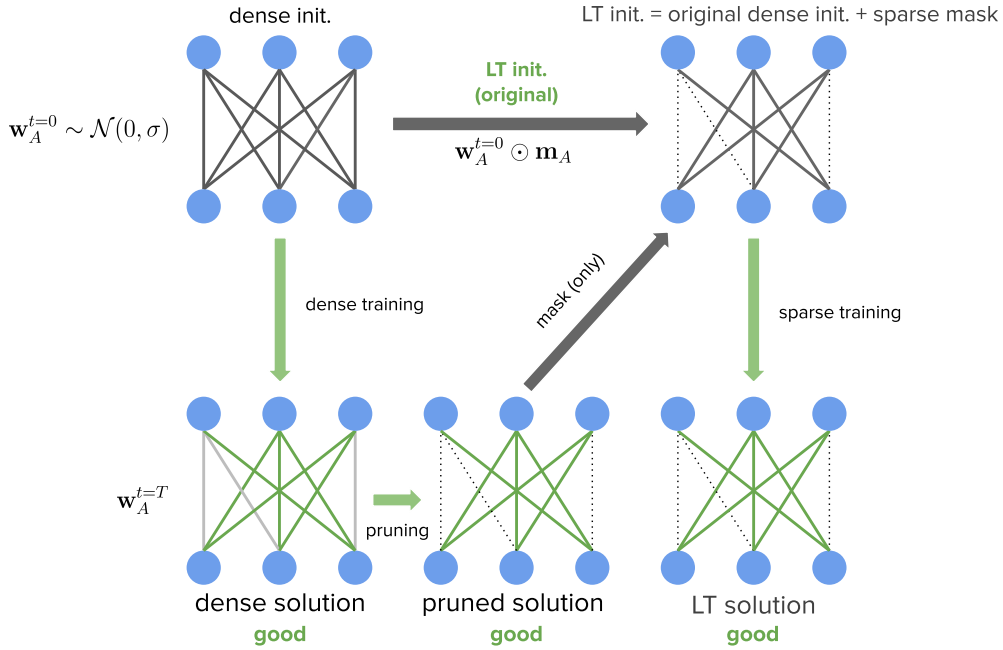


Figure 2.4: Visualization of the standard LTH paradigm. The LT initialization is composed of the original dense initialization, $\mathbf{w}_A^{t=0}$, we used to first train our model to convergence and then pruned via IMP to obtain our mask (i.e. winning ticket), \mathbf{m}_A . Then we sparse train, $\mathbf{w}_A^{t=0} \odot \mathbf{m}_A$. Sourced from [Ioannou \(2022\)](#).

initialized (i.e., $f(x; \mathbf{m} \odot \theta'_0)$ where $\theta'_0 \sim \mathcal{N}$), it fails to achieve the performance of the original dense network. This suggests that the specific initial weight values of the winning ticket, rather than just the sparsity pattern, are crucial for achieving strong generalization.

Procedure to find Winning Tickets: [Frankle and Carbin \(2019\)](#) adapted the IMP algorithm which repeatedly trains, prunes, and resets unpruned connection's value back to its initialization.

1. Randomly initialize a dense DNN $f(x; \theta_0)$, $\theta_0 \sim \mathcal{N}$.
2. Train the network for j iterations, θ_j .

3. Prune $p\%$ of the parameters of the parameters in θ_j , each round (i.e. n rounds in total) prunes $p^{\frac{1}{n}}\%$ of the weights that remained from the $n - 1^{\text{th}}$ round. Resulting in a mask, \mathbf{m} .
4. Reset the remaining parameters back to their original values in θ_0 , resulting in the winning ticket, $f(x; \mathbf{m} \odot \theta_0)$.

However, recognizing the limitations of finding winning tickets solely for very small models and datasets, [Frankle et al. \(2020b\)](#) in subsequent work demonstrated that obtaining winning tickets for moderately-sized models—such as ResNet50 on ImageNet—requires the use of *weight rewinding*. This technique modifies the standard IMP by resetting the weights of pruned subnetworks to their values at a specific earlier iteration k , rather than their initial values at iteration 0. This choice drastically improves the ability of the pruned subnetwork to converge successfully. [Frankle et al. \(2020b\)](#) explained that this rewinding process enhances subnetwork *stability* as the subnetwork becomes closer to the original optimum within the loss landscape, thereby making it more robust to the inherent noise of SGD. Moreover, their findings suggest a relationship between the rewinding and LMC such that subnetworks pruned and rewind earlier in training are more likely to reside in basins connected by low-loss paths to their dense counterparts.

Formally: Consider a dense DNN, $f(x; \theta_0)$ with initial parameters $\theta_0 \sim \mathcal{N}$. We train f on the training set with SGD to convergence, to obtain accuracy a in i iterations. Let θ_k represent the weights at iteration k . There exist an iteration $k \ll i$ and fixed pruning mask $\mathbf{m} \in \{0, 1\}^{|\theta_0|}$ such that the subnetwork $\mathbf{m} \odot \theta_k$ trains to accuracy $a^* \geq a$ in $i^* \leq i - k$ iterations.

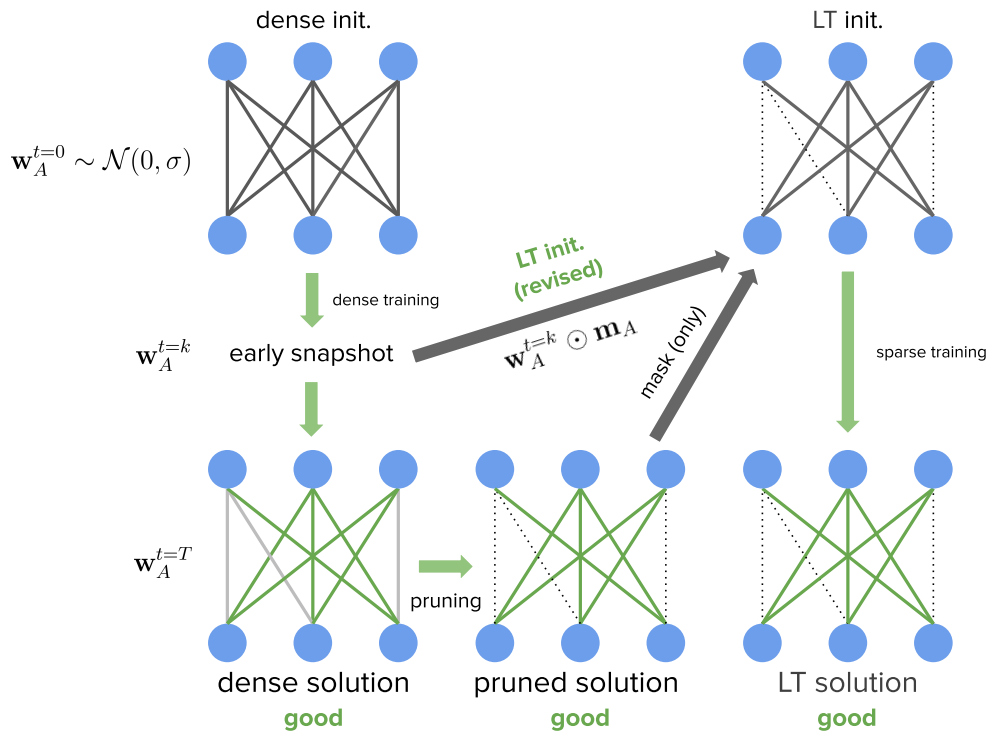


Figure 2.5: Similar to the setting from Figure 2.4, but instead our LT initialization is some “early snapshot” of dense training, equipped with the same \mathbf{m}_A . Then we sparse train, $\mathbf{w}_A^{t=k} \odot \mathbf{m}_A$. Sourced from Ioannou (2022).

2.3.4.1 Linear Mode Connectivity and Sparsity

While LMC has been widely studied in dense DNN, recent research has extended its role to SNN. Frankle et al. (2020b) demonstrated that IMP can successfully identify winning tickets, provided the network being pruned remains sufficiently stable to the noise inherent in SGD. To quantify this stability, they defined LMC by characterizing the loss barrier between independently trained child networks that originate from the same parent state. Paul et al. (2023) analyzed the IMP algorithm and showed that sparse network obtained after K^{th} IMP iteration is linearly connected to the sparse model obtained after $K + 1^{\text{th}}$ IMP iteration. In this work, we show that once

we take into account the *variance collapse* studied in [Jordan et al. \(2023\)](#), we are able to show that the sparse solution obtained after the K^{th} iteration is linearly connected to the dense solution. Furthermore, recent work has shown that the LTH effectively re-learns the original pruned solution it is derived from ([Evci et al., 2022](#)).

2.3.4.2 Limitations of Lottery Ticket Hypothesis

Although [Frankle et al. \(2020b\)](#) provided a foundational step towards training SNN from scratch by demonstrating that sparse subnetworks (“winning tickets”) exist within certain dense, randomly initialized DNN and can match the performance of the original model, the LTH relies on identifying these winning tickets after an initial training phase of the dense network to convergence. This requirement inherently contradicts the goal of avoiding the computational cost associated with training large DNN. Recent works extend LTH by finding winning tickets at initialization, but these methods have shown limited success, often performing worse than the pruned solution, especially at higher sparsity levels ([Wang et al., 2020](#); [Frankle et al., 2021](#); [Lee et al., 2019](#)). Therefore, while LTH highlights the potential of training SNN from scratch, it does not yet offer a practical, general solution for training SNN directly from scratch with an **arbitrary random initialization**.

The limitations inherent in the LTH motivate the central focus of our study: addressing the sparse training problem. Specifically, assuming we have obtained a sufficiently effective mask, \mathbf{m}_A , from a dense parent model A through a pruning technique such as IMP-FT, we aim to investigate the training of a network with this fixed sparsity pattern \mathbf{m}_A but initialized with an arbitrary weight initialization, $\mathbf{w}_B \sim \mathcal{N}$.

Chapter 3

Methodology

3.1 Motivation

DNN with sparse connectivity (i.e. a large fraction of weights or activations being zero) have gained significant attention as a way to address the growing costs and scale of modern DNN. Sparsity offers a range of benefits – from efficiency gains in computation and memory to potential improvements in generalization – making it a powerful paradigm especially as networks become ever larger. Structured pruning and sparsity learning methods can drastically reduce the number of computations (FLOPs) and memory usage during inference, leading to faster execution and lower energy consumption (Shi et al., 2021). These efficiency gains make sparsity especially effective for deploying DNN in resource-constrained settings. As a result, sparsity has had a strong impact in domains such as mobile vision, where real-time performance and memory constraints are critical; large-scale language models, where sparsity enables scaling to billions of parameters without proportional computational cost; and edge computing, where power and

bandwidth are limited (Haberer and Landsiedel, 2022; Munir et al., 2023; Zheng et al., 2024). Han et al. (2015) demonstrated that AlexNet and VGG-16 can be compressed by over 9× and 13× respectively with negligible accuracy loss, and modern hardware accelerators (e.g., NVIDIA Ampere’s 2 : 4 sparsity support) have begun exploiting such sparsity for real runtime gains (Lasby et al., 2024; Lei et al., 2023). Despite this success in the inference phase, sparse training from scratch remains challenging. In effort to solve this problem, the LTH showed that sparse subnetworks exist within dense networks and can be trained in isolation to match dense generalization performance under particular settings. However, a key limitation remains where LTH masks fail to generalize across random initializations not associated to the original mask. Furthermore, Dynamic Sparse Training (DST) methods (Evcı et al., 2021; Mocanu et al., 2018) evolve the sparse connectivity throughout training and can match dense generalization performance. But they take significantly longer to converge and additionally maintaining and updating the sparse topology during training introduces more computational overhead (Evcı et al., 2021). In this work, we try to understand *why LTH masks fail to transfer to a new random initialization*. Our hypothesis is that the loss basin corresponding to the LTH mask is not aligned with the new random initialization, as shown in fig. 3.3. Since the sparse mask is not aligned with the basin of the new random initialization, sparse training does not work well; therefore, aligning the LTH mask with the new random initialization may improve sparse training and enable the transfer of LTH masks to new random initializations.

3.1.1 Visualization: 2D Loss Landscape

Consider, a DNN with a single layer and only two parameters, $\mathbf{w} = (w_0, w_1)$, operating on a single input scale x_0 has the weight symmetry in the 2D loss landscape with symmetric basins of attraction as illustrated in figures 2.3, 2.4, and 2.5.

3.1.1.1 Dense Training and Pruning

As shown in fig. 3.1, the original dense model, denoted by its weights \mathbf{w}_A , is initially trained from a random dense initialization $\mathbf{w}_A^{t=0}$. This random initialization naturally results in a high training loss at the beginning of training. The model is then trained using SGD until it converges to a dense solution $\mathbf{w}_A^{t=T}$ shown by a low training loss. Subsequently, this converged dense solution is pruned using IMP-FT, yielding the binary mask $\mathbf{m}_A = (1, 0)$. As further explained in fig. 4.5, the application of this mask effectively projects the solution into an axis-aligned subspace within the same solution basin.

Pruning Setup: We apply standard IMP-FT (Frankle and Carbin, 2019; Han et al., 2015; Renda et al., 2020) to obtain our final mask, \mathbf{m}_A , producing a sparse subnetwork $\mathbf{w}_A^{t=T} \odot \mathbf{m}_A$. For pruning, we utilize PyTorch’s `torch.nn.utils.prune` library (Paganini and Forde, 2020).

1. In an unstructured, global manner, we identify and mask (set to zero) the smallest 20% of unpruned weights based on their magnitude.
2. This process is repeated for N rounds to achieve the target sparsity S , with each subsequent round pruning 20% of the remaining weights.
3. During each round, the model is trained for `train_epochs_per_prune`

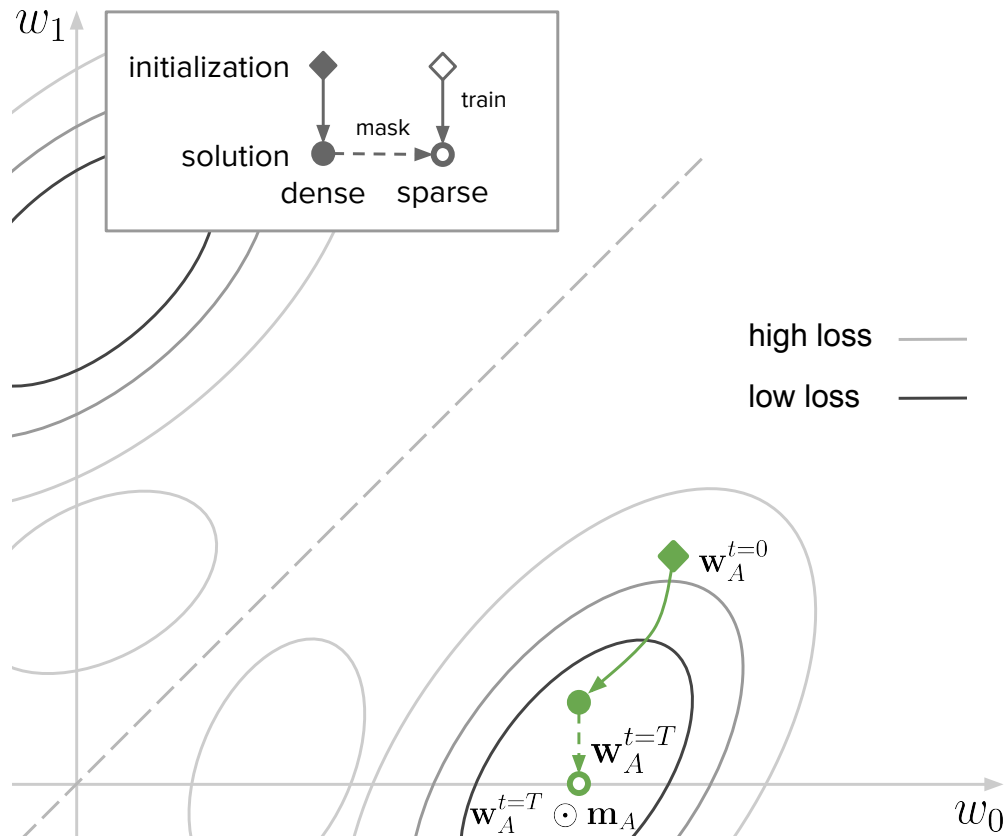


Figure 3.1: Visualization of the standard dense training and post-trained pruning via IMP to obtain a sparse mask, \mathbf{m}_A . Sourced from Yani Ioannou.

epochs. (Details about specific model hyperparameters is discussed in section 4.4.2.)

3.1.1.2 Lottery Ticket Hypothesis

In fig. 3.2, we re-use the initialization $\mathbf{w}_A^{t=0}$, to train model A with the pruned mask from fig. 3.1, \mathbf{m}_A , as in LTH. Consistent with the findings of Evcı et al. (2022) and further supported by our experimental results presented in 4.6, LTH solution appears to be effectively relearning the characteristics of the pruned solution and remains within the same loss basin as our parent dense solution.

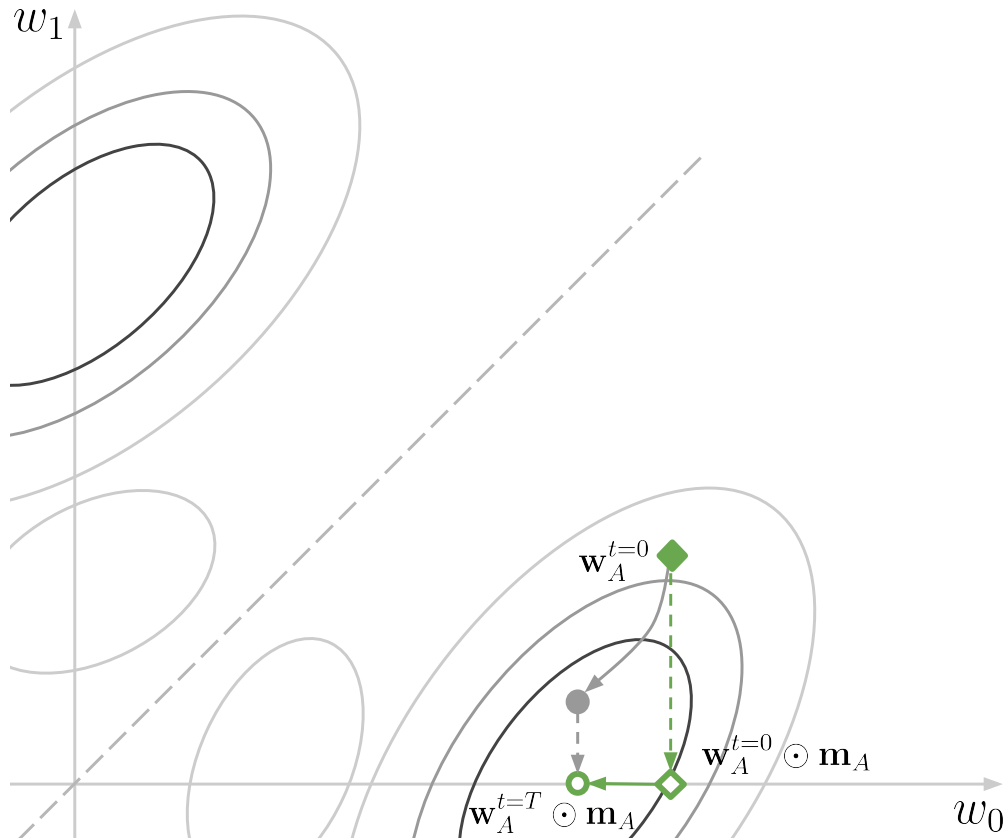


Figure 3.2: Visualization of LTH. Sourced from Yani Ioannou.

3.1.1.3 Our Method

In fig. 3.3, naively using the same mask to train a model, B , from a different random initialization will likely result in the initialization being far from a good solution. Permuting the mask to match the (symmetric) basin in which the new initialization is in will enable sparse training

3.2 Permutation Matching

[Ainsworth et al. \(2023\)](#) showed that the permutation symmetries in the weight space can be leveraged to align the basin of two models trained from different random initializations. The permutation mapping can be obtained

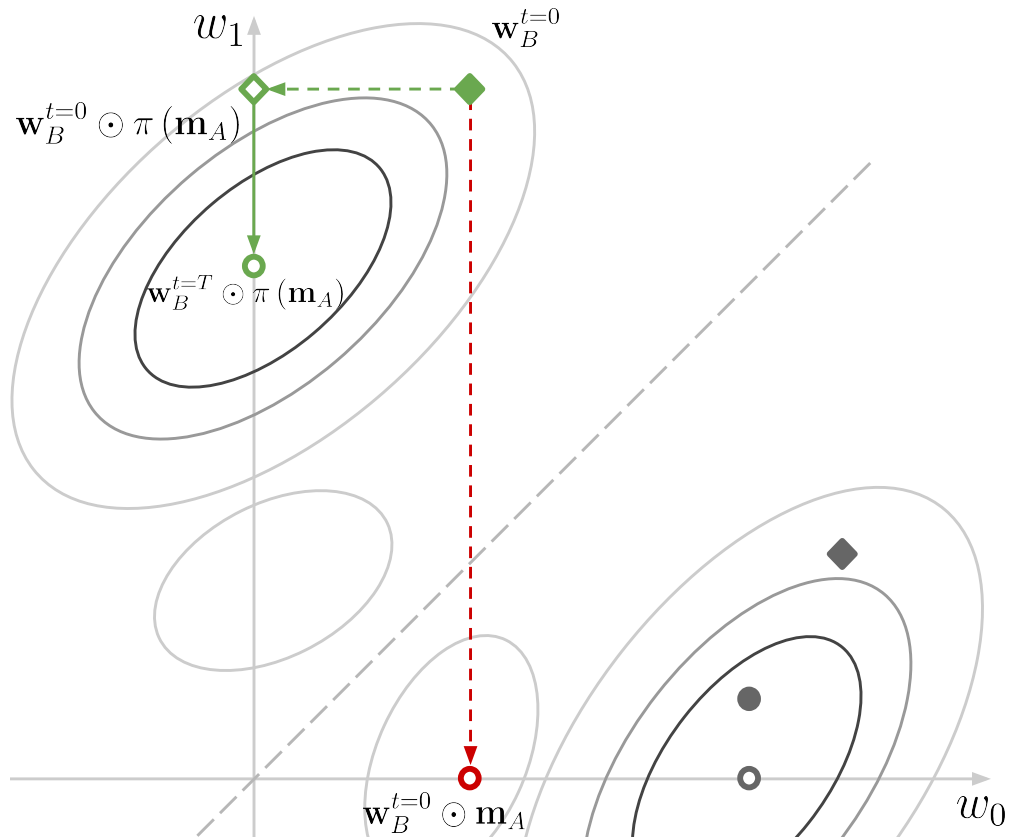


Figure 3.3: Visualization of the sparse training problem and how we aim to solve it. Sourced from Yani Ioannou.

by either matching activations or weights. In this work, we use activation matching to obtain the permutation mapping as it has been shown to be more stable in recent works (Sharma et al., 2024).

3.2.1 Evaluating Permutation Matching

Since LAP uses a greedy search to find an approximate solution, to ensure that the permuted model A and model B lie in the same basin, we evaluate the LMC, measure by the loss barrier between the two models as defined in 2.10.

To ensure that the permutation mapping, π , can closely match model A

and model B , we evaluate the loss barrier between the permuted model A and model B . However, aligning neurons alone is not sufficient to establish a low loss barrier due to variance collapse (Jordan et al., 2023). To overcome the variance collapse issue, we used REPAIR (Jordan et al., 2023) to correct the variance of the activations in the interpolated/merged model. As shown in fig. 4.7, the loss barrier after permutation matching and correcting the variance (REPAIR) is lower than the loss at random initialization, showing permutation mapping can match the models to bring them closer/in the loss basin.

3.2.2 Aligning Masks via Weight Symmetry

In contrast to previous works (Ainsworth et al., 2023), we are interested in permuting the mask obtained by LTH such that the optimization basin of the permuted sparse mask and the new random initialization is aligned. To validate our hypothesis, we train two dense models, $\mathbf{w}_A^{t=0}$ and $\mathbf{w}_B^{t=0}$, where t denotes the epoch, to convergence (trained for T epochs) and then use activation matching (Jordan et al., 2023) to find the permutation mapping π , such that the activations of $\pi(\mathbf{w}_A^{t=T})$ and $\mathbf{w}_B^{t=T}$ are aligned. Mask \mathbf{m}_A , obtained using IMP, is also permuted with the same permutation map π . The intuition is that the permuted mask aligns with the loss basin of the model $\mathbf{w}_B^{t=T}$, which is necessary for sparse training and, therefore, the sparse model can be more easily optimized (see fig. 3.4). We denote training with the permuted mask, $\pi(\mathbf{m}_A)$ as *permuted* and with the non-permuted mask, \mathbf{m}_A as *naive*.

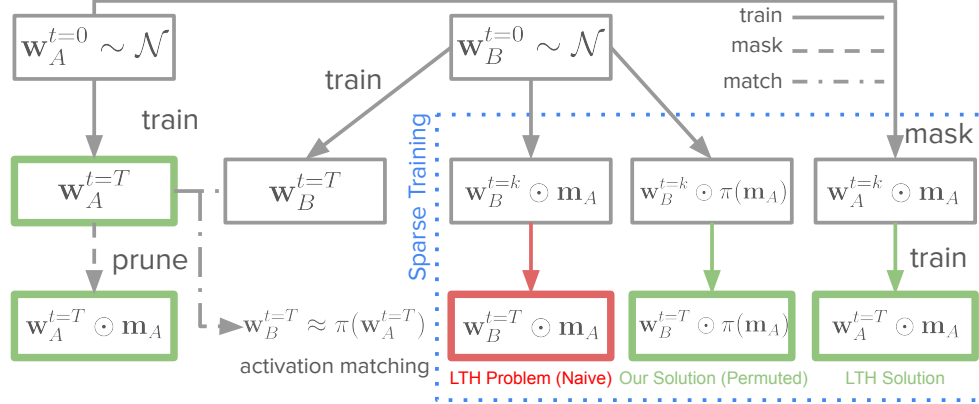


Figure 3.4: The overall framework of the training procedure, beginning with two distinct dense random weight initializations, $\mathbf{w}_A^{t=0}, \mathbf{w}_B^{t=0}$ sampled from a normal distribution, \mathcal{N} . The sparse training problem attempts to train the random initialization, $\mathbf{w}_B^{t=0}$ using the naive mask \mathbf{m}_A , found by pruning a dense trained model, $\mathbf{w}_A^{t=T}$. However, this results in poor generalization performance (Frankle et al., 2020b). We propose to instead train $\mathbf{w}_B^{t=k}$ at some rewind epoch k , equipped with a *permuted* mask $\pi(\mathbf{m}_A)$. We show that this achieves more comparable generalization to the pruned model/trained LTH solution, $\mathbf{w}_A^{t=T} \odot \mathbf{m}_A$. Sourced from Yani Ioannou.

3.3 Sparse Training

We first show that the dense solution $\mathbf{w}_A^{t=T}$ and the LTH solution obtained by training a model with sparse mask \mathbf{m}_A remain in the same linearly connected mode as defined in eq. (2.10) if one fixes the variance collapse identified by Jordan et al. (2023) by updating the activation statistics via the REPAIR method. We show in fig. 4.5a the error barrier after applying the REPAIR method remains considerably lower as we increase sparsity by iteratively pruning (IMP). These results extend the findings of Paul et al. (2023) to show that when variance collapse is taken into account, the LTH solution remains in the same linearly connected basin as the original dense solution.

In sparse training, the model is trained with a mask \mathbf{m} , masking some of the weights, during both forward and backward passes. To evaluate the

transferability of the permuted LTH mask we train, a different random initialization $\mathbf{w}_B^{t=0}$, the LTH sparse mask \mathbf{m}_A and permuted LTH mask $\pi(\mathbf{m}_A)$, which we denote the naive and permuted solution respectively. We also evaluate the LTH baseline, i.e., training model $\mathbf{w}_A^{t=0}$ with mask \mathbf{m}_A . Since LTH requires weight rewinding to an earlier point in training, we also use a rewind checkpoint from epoch $t = k \ll T$ for both the baselines and permuted solution. This rewind checkpoint is a snapshot of the network’s dense weight configuration at a designated earlier point in its training trajectory.

Chapter 4

Experiments and Results

To validate our hypothesis, we trained ResNet20 (He et al., 2016) and VGG11 (Simonyan and Zisserman, 2015) models on the CIFAR-10/100 datasets (Krizhevsky, 2009) (details in section 4.4.1) across different levels of sparsity ($S = 0.80, 0.90, 0.95, 0.97$).

Each of our results for a given rewind point, k , is averaged over 3 runs. We used ResNet20 with varying widths ($w = 1, 4, 8, 16$) to study the effect of increasing width on the permutation matching and, thereby, the performance of the permuted sparse model. We also demonstrate our hypothesis on the large-scale ImageNet dataset (Deng et al., 2009) using ResNet50, showing the efficacy of our method across different models and datasets of varying sizes.

4.1 Experimental Results

4.1.1 ResNet20/CIFAR-10 & CIFAR-100

We trained ResNet20 on the CIFAR-10/100 datasets. More details in section 4.4.1.1. As shown in figs. 4.1 and 4.2, the permuted solution outperforms the naive baseline across all model widths and rewind points. Since it is more difficult to train models with higher sparsity, the gap between naive and permuted solutions increases as sparsity increases, as shown in fig. 4.1d for width multiplier 1,4,8, and 16. Details about width multiplier implementation is outlined in section 4.3.1. It can also be observed that at higher sparsity increasing the rewind point improves both the LTH and permuted solution but not the naive solution. The improved performance of the permuted solution over naive supports our hypothesis and shows that misalignment of the LTH mask and loss basin corresponding to the new random initialization could explain why LTH masks do not transfer to different initializations. We also show accuracy vs. sparsity plots for $k = \{10, 25, 50, 100\}$ (details in section 4.4.2); as sparsity increases, the gap between permuted and naive solution increases for all rewind points. As illustrated in figure fig. 4.1, neither the LTH nor the permuted solution performs effectively with random initialization ($k = 0$) but improves on increasing the rewind point up to a certain point, beyond which it plateaus. Detailed results are presented in tables 4.7 to 4.10 in section 4.4. Overall, as the width increases, the gap between training from a random initialization with the permuted mask and the LTH/dense baseline decreases, unlike training with the non-permuted mask (naive), showing a model trained with the permuted mask generalizes

4. Experiments and Results

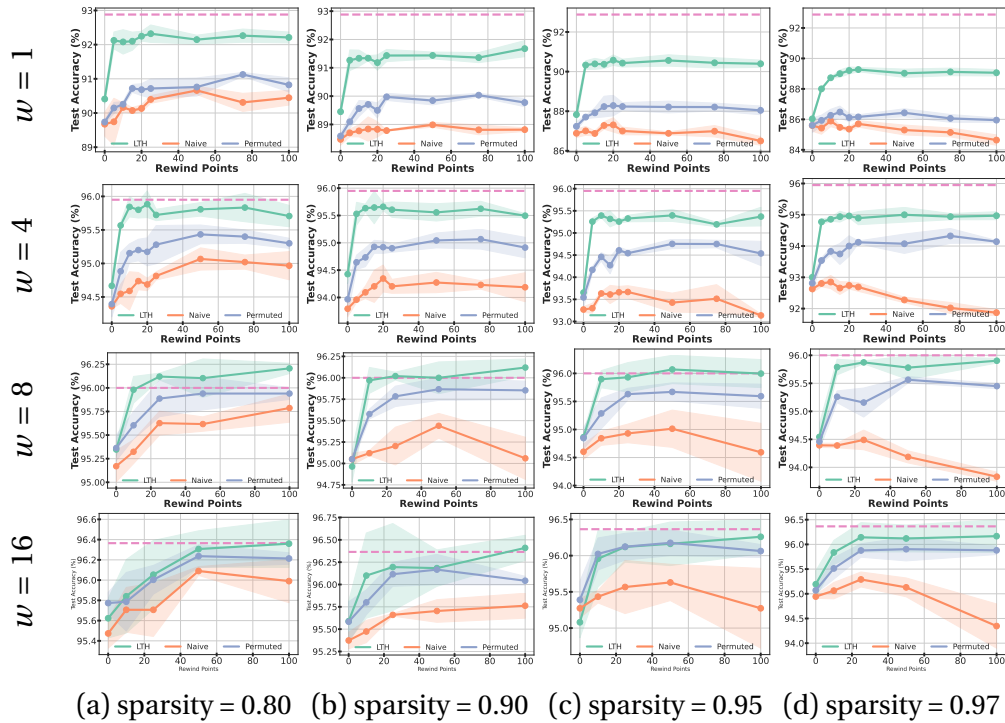
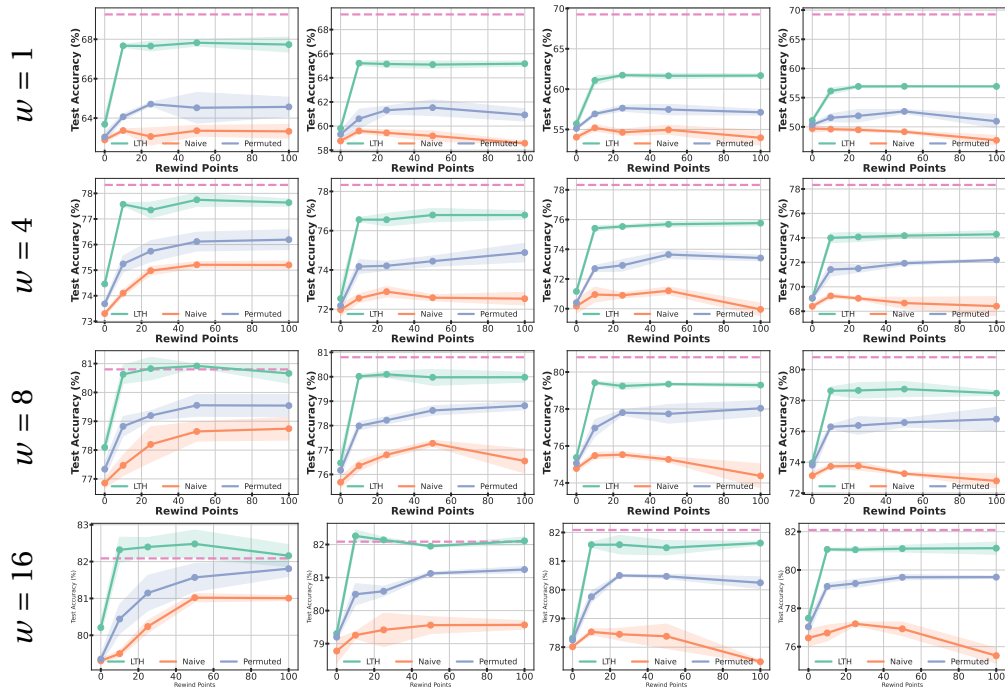


Figure 4.1: **ResNet20 \times { w }/CIFAR-10.** Test accuracy of sparse network solutions vs. increasing rewind points for different sparsity levels and widths, w . The effect of the rewind point on the test accuracy for different sparsities is shown. The dashed (- -) line shows the dense model accuracy.

better than naive. We observe that beyond a particular rewind point (i.e. epoch) as we achieve stability (Frankle et al., 2020b), increasing the rewind point does not significantly improve generalization performance.

We also validated our hypothesis on CIFAR-100 using ResNet20 with varying widths. As shown in fig. 4.2, the permuted solution consistently outperforms the naive solution, showing that our hypothesis holds true across different models and datasets. Similar to the CIFAR-10 dataset, as we increase the model width multiplier, the gap between the permuted and naive solution increases, showing the efficacy of our method. Detailed results are presented in tables 4.13 to 4.16 in section 4.4.



(a) sparsity = 0.80 (b) sparsity = 0.90 (c) sparsity = 0.95 (d) sparsity = 0.97

Figure 4.2: **ResNet20** $\times\{w\}$ /**CIFAR-100**. Test accuracy of sparse network solutions vs. increasing rewind points for different sparsity levels and widths, w . The effect of the rewind points on the test accuracy for different sparsities is shown. As the width increases, the gap between training from a random initialization with the permuted mask and the LTH/dense baseline (dashed line) decreases, unlike training with the non-permuted mask (naive), showing the model trained with the permuted model generalizes better than naive. The dashed (- -) line shows the dense model accuracy.

4.1.2 VGG11/CIFAR-10

For convolutional neural networks, we train a modified version of the standard VGG11 implemented by [Jordan et al. \(2023\)](#) on CIFAR-10. More details in section 4.4.1.1. Primary differences are:

- A single fully connected layer at the end which directly maps the flattened feature map output from the convolutional layers to the 10 classes for CIFAR-10 classification.

- The classifier is set up for CIFAR-10 with 10 output classes, as originally VGG11 was designed for ImageNet with 1000 output classes (Deng et al., 2009).

We observe that for a moderate sparsity (80%) in fig. 4.3a, the gap between the permuted and the naive baseline is not large, however for a higher sparsity level (90%), the permuted solution significantly outperforms the naive solution as shown in fig. 4.3b. For the VGG11 model, on increasing the rewind point, the permuted solution closely matches the accuracy of LTH, while the naive solution significantly plateaus and does not improve on increasing the rewind point. As shown in fig. 4.3b, we find that beyond $k \geq 10$, the naive solution plateaus, but the permuted solution persistently approaches the performance level of the LTH solution. For higher sparsities, the naive baseline was unstable in training as the modified VGG11 architecture does not have BatchNorm layers (Ioffe and Szegedy, 2015); we omit those results in the discussion for a fair comparison. Detailed results are presented in Table 4.11 in section 4.4. x

4.1.3 ResNet50/ImageNet

We utilize the standard ResNet50 implementation provided by torchvision and customize PyTorch’s distributed data parallel codebase (Contributors, 2024) for training models on the ILSVRC 2012 (ImageNet) dataset, which consists of 1.28 million images across 1,000 classes (Deng et al., 2009). More details in section 4.4.1.2. We used the ResNet50 model to evaluate the performance of the permuted mask at different sparsity levels. As observed in fig. 4.4, the permuted solution outperforms the naive solution across

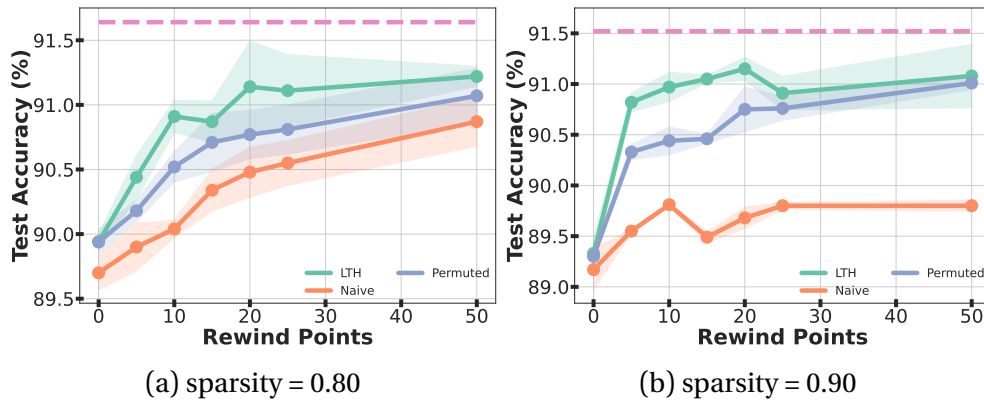


Figure 4.3: **VGG11 \times {1}/CIFAR-10**. Test accuracy of sparse network solutions at increasing rewind points for different sparsity levels. In fig. 4.3b, the permuted solution closely matches the LTH solution. We also see a more noticeable gap between the permuted and naive solutions compared to fig. 4.3a. The dashed (- -) line shows the dense model accuracy.

all sparsity levels, showing that our hypothesis holds true on large-scale datasets as well. While the permuted solution performs better than the naive solution, there is still a significant gap between LTH and the permuted solution in the case of the ImageNet dataset as compared to the CIFAR-10/100 dataset. This could be due to permutation matching not being accurate enough, as only a small subset of the training dataset was used for activation matching. This can also be visualized in terms of the loss barrier in fig. 4.7c between the permuted model A and model B ; the loss barrier after permutation is more prominent compared to the CIFAR dataset, figs. 4.7a and 4.7b. Thus, the permutation mapping π cannot match the models perfectly in the case of ImageNet since the permutation matching algorithm uses a greedy search algorithm to find the permutation mapping. However, given a better mapping, it may be possible to further improve the performance of the permuted solution as discussed in section 4.3.2. Upon increasing model capacity, we observe an improved ability to capture the complexity of the

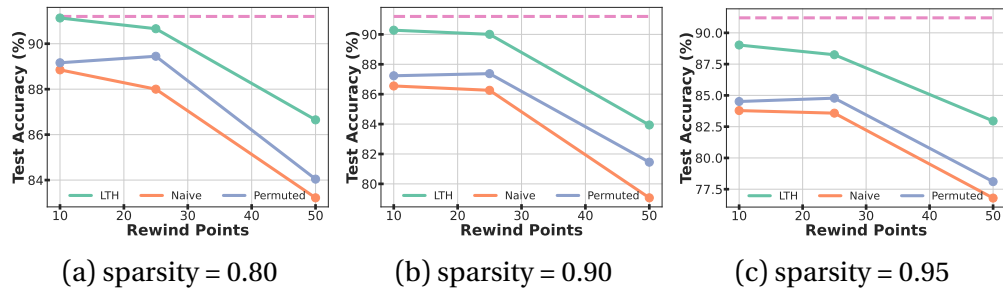


Figure 4.4: **ResNet50×{1}/ImageNet**. Top-5 test accuracy vs. rewind points of sparse network solutions at various sparsity levels. We observe the permuted solution consistently performing better than the naive solution for all sparsities. The dashed (- -) line shows the dense model accuracy.

input distribution, leading to improved LMC. This is reflected in a nearly 57% reduction in the loss barrier for ResNet50×{2}/ImageNet, as shown in fig. 4.7c. Detailed results are presented in table 4.12 in section 4.4. As demonstrated in table 4.12, the permuted solution outperforms the naive approach by nearly 2% at higher sparsity levels.

4.1.4 Early Permutation Matching

In our current methodology, we train both models A and B to convergence, resulting in the weight configurations $\mathbf{w}_A^{t=T}$ and $\mathbf{w}_B^{t=T}$, respectively. However, it has been observed in (Sharma et al., 2024) that it is possible to find permutation mapping earlier in training. In effort to reduce the computational cost associated with our approach, we aim to find a permutation π that allows us to suitably align the weights of model A at convergence ($\mathbf{w}_A^{t=T}$) with the weights of model B at an earlier training iteration $i \ll T$, $\mathbf{w}_B^{t=i}$. The initial results presented in 4.1 demonstrate the feasibility of obtaining a suitable permutation early in the training process. Specifically, the permuted solutions derived at early matching points, such as $t = 5, 20$, are sufficient

to achieve a level of generalization comparable to those obtained at later points in training, such as $t = 100$ and $t = 200$ (fully-converged).

4.1.4.1 Computational Cost of the Permuted Solution

The primary difference in computational complexity between the LTH, naive, and permuted solutions lies in the process of neuronal alignment, where weight/activation matching is used to locate permutations in order to bring the hidden units of two networks into alignment. To obtain the permuted solution, two distinct models must be trained independently to convergence, after which their weights or activations are aligned through a permutation-matching process. This alignment, though relatively efficient, adds a small computational overhead compared to LTH and naive solutions, which do not involve matching steps.

4.1.5 Analysis of Mask Reusage

In an additional experiment, we derived a sparsity mask from a model trained on the CIFAR-10 dataset and then applied this mask with a new random initialization to train models on the SVHN dataset. Across all tested sparsity levels, the model initialized with the permuted mask consistently outperformed the model initialized with the unpermuted (naive) mask.

4.1.5.1 Experimental Setup and Results

The experimental setup for this specific analysis, conducted on ResNet20 \times {1} using the SVHN dataset and with the rewinding point set to $k = 20$, is defined as follows:

4. Experiments and Results

Table 4.1: **ResNet20×{1}/CIFAR-10**. Results using the ResNet20×{1} trained on CIFAR-10, from a rewind point $k = 20$, using various methods of sparse training with sparsity S . The LTH and naive methods remain fixed, as they are independent of matching. For the permuted method, the permutation, π , is obtained by matching a fully trained dense model $\mathbf{w}_A^{t=T}$ and $\mathbf{w}_B^{t=i}$ at an early point in training, where $i \in \{5, 20, 50, 100\}$.

S	Method	Early Matching Point $t = i$				
		$t = 5$	20	50	100	200
80%	LTH			92.25 ± 0.14		
	naive			90.13 ± 0.11		
	perm.	90.49 ± 0.37	90.34 ± 0.63	90.42 ± 0.29	90.42 ± 0.25	90.68 ± 0.18
90%	LTH			91.18 ± 0.27		
	naive			88.83 ± 0.27		
	perm.	89.16 ± 0.51	89.23 ± 0.59	89.39 ± 0.69	89.31 ± 0.60	89.50 ± 0.27
95%	LTH			90.58 ± 0.26		
	naive			87.31 ± 0.36		
	perm.	87.37 ± 0.33	87.68 ± 0.77	87.43 ± 1.00	87.54 ± 0.43	88.29 ± 0.52
97%	LTH			89.21 ± 0.23		
	naive			85.36 ± 0.14		
	perm.	85.77 ± 0.44	85.93 ± 0.94	86.09 ± 0.51	85.88 ± 0.47	86.12 ± 0.27

1. Dense train $\mathbf{w}_A^{t=0}$ to convergence $\mathbf{w}_A^{t=T}$, we trained the model for $T = 200$ epochs on CIFAR-10.
2. Prune $\mathbf{w}_A^{t=T}$ via IMP-FT to obtain sparse mask, \mathbf{m}_A .
3. Dense train $\mathbf{w}_B^{t=0}$ to convergence $\mathbf{w}_B^{t=T'}$, we trained the model for $T' = 50$ epochs on SVHN.
4. Use activation matching to find a permutation, π , such that $\pi(\mathbf{w}_A^{t=T}) \approx \mathbf{w}_B^{t=T'}$ in weight-space, evaluated on SVHN.
5. **LTH solution:** Dense train $\mathbf{w}_A^{t=0}$ from Step 1. on SVHN for k epochs and then sparse train $\mathbf{w}_A^{t=k} \odot \mathbf{m}_A$ on SVHN for $T' - k$ epochs.
6. **Naive solution:** Load a rewind checkpoint from Step 3. at k , we sparse train $\mathbf{w}_B^{t=k} \odot \mathbf{m}_A$ on SVHN for $T' - k$ epochs.

7. **Permuted solution:** Load a rewind checkpoint from Step 3. at k , we sparse train $\mathbf{w}_B^{t=k} \odot \pi(\mathbf{m}_A)$ on SVHN for $T' - k$ epochs.

As demonstrated in table 4.2, our results indicate that across all sparsity levels (S) tested, the permuted solution consistently outperforms the naive solution and closely approaches the performance of the LTH solution. We observed statistically more significant improvements for the rewind point set at $k = 20$, suggesting enhanced stability in the sparse training of the permuted solution compared to the results obtained at $k = 10$.

Table 4.2: **ResNet20×{1}/SVHN**. Mask reuse results for different sparsity levels and rewind points k .

S	Method	Rewind $k = 10$	Rewind $k = 20$
80%	LTH	95.21 ± 0.06	95.23 ± 0.09
	naive	94.69 ± 0.04	94.09 ± 0.53
	perm.	94.45 ± 0.98	95.08 ± 0.10
95%	LTH	94.52 ± 0.20	94.53 ± 0.07
	naive	93.05 ± 0.80	92.61 ± 0.64
	perm.	92.94 ± 0.81	93.54 ± 0.99
97%	LTH	93.87 ± 0.08	93.75 ± 0.08
	naive	92.27 ± 1.22	90.69 ± 2.78
	perm.	92.34 ± 0.55	92.61 ± 0.11

4.2 Diversity Analysis of Permuted Models

4.2.1 Loss Landscape Analysis

A limitation of LTH is that it consistently converges to very similar solutions to the original pruned model (Evcı et al., 2022). Evcı et al. (2022)

speculate this occurs because the LTH is always trained with the same initialization/rewind point, and effectively relearns the same solution. Our hypothesis is that permuted LTH masks, trained with distinct initialization/rewind points and subject to approximation errors in permutation matching, may learn more diverse functions than the LTH itself. The 0-1 loss landscape of our permuted solution, as seen in 4.6a, converges to the same linearly connected mode as the LTH and dense solutions.

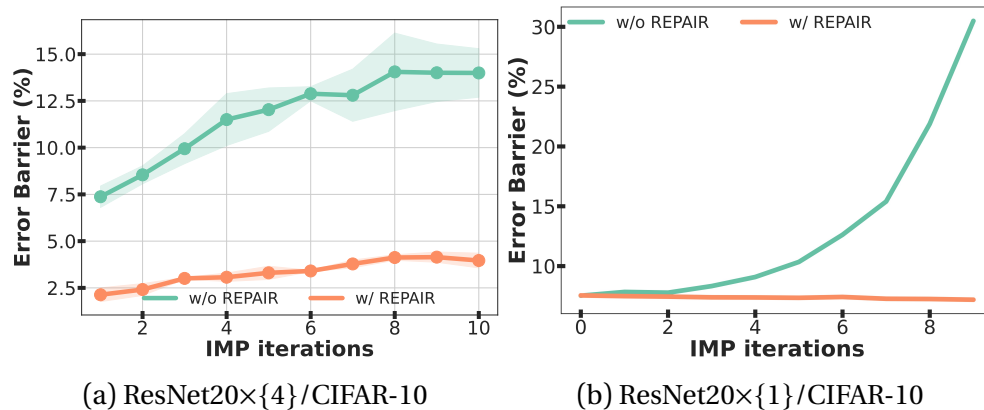
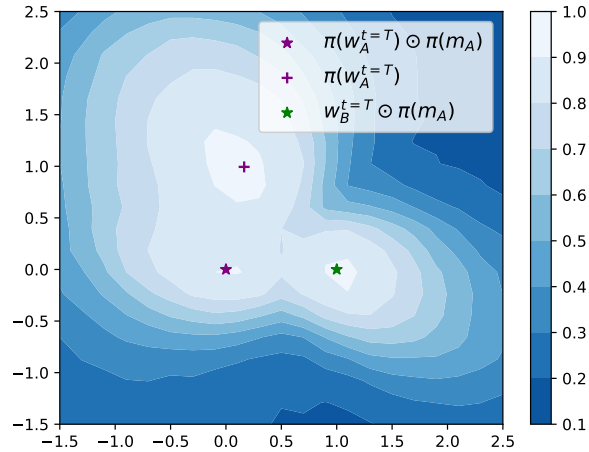


Figure 4.5: **LTH solution remains in the same linearly connected mode as the dense solution.** In fig. 4.5a we plot the error barrier between the dense solution and the sparse solution (y -axis) vs the IMP iteration corresponding to the sparse solution (x -axis), for 90% sparsity. We observe that after fixing variance collapse via the REPAIR method, the error barrier between the dense and the sparse solutions remains small, thus showing that the LTH solution remains in the same linearly connected mode as the dense solution. In fig. 4.5b, we demonstrate the same result as fig. 4.5a except we use IMP-FT and still see a small error barrier between the dense and pruned solution.

4.2.2 Permuted Ensemble Analysis

We analyze the functional diversity of sparse models trained at 90% sparsity, with either a permuted mask (permuted), the LTH mask (naive), LTH mask & initialization and the original pruned solution (i.e. IMP) on which the LTH is based. We follow the same analysis as [Evci et al. \(2022\)](#) and compare

the diversity of the resulting models, over five different training runs, using disagreement score, KL divergence and JS divergence. We also compare with an ensemble of five models trained independently with different random seeds.



(a) ResNet20×4/CIFAR-100: 0–1 loss landscape

Figure 4.6: We show that, modulo permutations, reusing the permuted mask leads to convergence in the same mode as the LTH solution. Hence, there is a small loss barrier between the permuted and LTH solutions, demonstrating they are within the same linearly connected mode. Sourced from Ekansh Sharma.

Table 4.3: **Ensemble Diversity Metrics for CIFAR-10/CIFAR-100**. Although the mean test accuracy of LTH is higher, the ensemble of permuted models achieves better test accuracy due to better functional diversity of permuted models.

Mask	Test Acc. (%)	Ensemble Acc. (%)	Disagreement	KL	JS
ResNet20×{1}/CIFAR-10					
none (dense)	92.76 ± 0.106	-	-	-	-
IMP	91.09 ± 0.041	93.25	0.093	0.352	0.130
LTH	91.15 ± 0.163	91.43	0.035	0.038	0.011
permuted	89.38 ± 0.170	91.75	0.107	0.273	0.091
naive	88.68 ± 0.205	91.07	0.113	0.271	0.089
ResNet20×{4}/CIFAR-100					
none (dense)	78.37 ± 0.059	-	-	-	-
IMP	74.46 ± 0.321	79.27	0.259	1.005	0.372
LTH	75.35 ± 0.204	75.99	0.117	0.134	0.038
permuted	72.48 ± 0.356	77.85	0.278	0.918	0.327
naive	71.05 ± 0.366	76.15	0.290	0.970	0.348

In table 4.3, we compare the functional similarity of the generated models using several metrics, including disagreement, which quantifies prediction differences (Fort et al., 2020), and Kullback–Leibler (KL) divergence and Jensen-Shannon (JS) divergence, which measure the dissimilarity between the output distributions of different models (Evcı et al., 2022). The results indicate that permuted masks achieve a level of diversity comparable to that of computationally intensive IMP solutions, consequently resulting in ensembles with a similar increase in generalization performance. In the case of IMP, which involves training independently over five different random seeds to obtain five distinct solutions with different sparse masks/topologies (i.e; $\mathbf{m}_1, \mathbf{m}_2, \mathbf{m}_3, \mathbf{m}_4, \mathbf{m}_5$), the process is computationally expensive and often impractical for creating large ensembles. The IMP baseline serves as an upper bound, as both training from different random initializations and

learning different topologies introduces a high level of stochastic noise to learn diverse solutions. The LTH ensemble, on the other hand, is generated by training with the same mask (\mathbf{m}_1) and weight initialization (\mathbf{w}_1) across five different training runs with different data orders. As noted in (Evcı et al., 2022) and our analysis in fig. 4.6, LTH ensembles do not typically exhibit high diversity because they are constrained by the same initial conditions and sparsity pattern, making them less suitable for ensemble learning. However, our proposed method allows us to reuse the LTH mask with different random initializations, introducing a greater source of randomness compared to the LTH baseline and thus improving the diversity of the solutions. While the diversity achieved with permuted masks (using the same mask \mathbf{m}_1 with five different permutations, i.e; $\pi_1, \pi_2, \pi_3, \pi_4, \pi_5$) is expected to be lower than that of IMP (due to the fixed topology), it offers a more computationally efficient way to improve diversity. As shown in table 4.3, the LTH ensemble does not significantly improve the accuracy compared to a single model, whereas the permuted ensemble significantly enhances the performance compared to a single model. Specifically, on CIFAR-100, the permuted ensemble achieves an accuracy of 77.85%, surpassing the LTH ensemble’s accuracy of 75.99%. This demonstrates that permuting the mask can help train more diverse solutions while being computationally more efficient than IMP.

4.3 Effect of Model Width Multiplier

4.3.1 Experimental Details

In our experiments, we mainly focused on varying the widths of the ResNet20 model, as VGG11 and ResNet50 are already vastly overparameterized. For residual neural networks, we train the standard ResNet20 on CIFAR-10 and CIFAR-100 with varying width. We implemented a scalar, w , that adjusts the number of channels in each convolutional and fully connected layer:

- **First Convolution Layer:** The number of output channels is scaled from 16 to $w \times 16$.
- **Layer 1,2,3:** The number of output channels for the convolutional blocks in these layers are scaled from 16, 32, and 64 to $w \times 16$, $w \times 32$, and $w \times 64$, respectively.
- **Fully Connected Layer:** The input dimension to the final linear layer is scaled to $w \times 64$.

4.3.2 Model Width Analysis

Permutation matching is an NP-hard problem; the activation matching algorithm proposed by [Ainsworth et al. \(2023\)](#) does not find the global optimum; rather, it uses a greedy search to explore a restricted solution space. Therefore, in practice permutation matching does not perfectly align two models. However, it has been observed that for wider models, the algorithm can more closely align two models ([Ainsworth et al., 2023](#); [Sharma et al., 2024](#)). To understand how the performance of the permuted

model is affected by the approximation error of the matching algorithm, we evaluated the LMC and the accuracy of the permuted solution on ResNet20 models with varying layer widths. As shown in fig. 4.7, on increasing the layer width, the loss barrier of the interpolated network reduces, showing that permutation mapping becomes more accurate and aligns two models better. Also, it can be observed in figs. 4.1 and 4.2 that the permuted solution becomes close to the LTH solution on increasing the model width, showing that as the permutation matching becomes more accurate, the gap between the LTH and the permuted solution reduces.

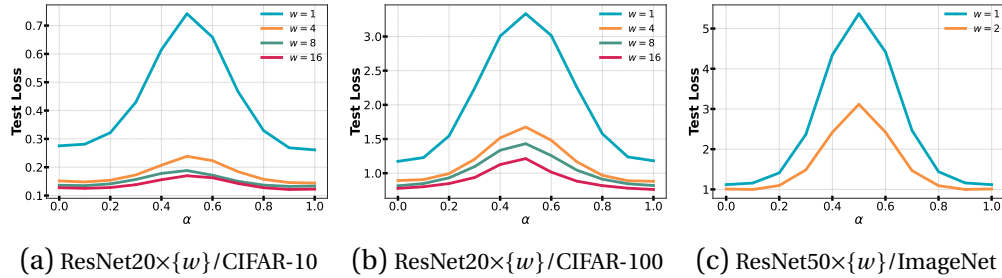


Figure 4.7: **Larger width exhibits better LMC.** Plots showing linear interpolation between $\pi(\mathbf{w}_A^{t=T})$ and $\mathbf{w}_B^{t=T}$ where π was obtained through activation matching between two dense models for varying widths, w . As the width of the model increases, the permutation matching algorithm gets more accurate, thereby reducing the loss barrier (i.e; better LMC), which is evaluated on the test set. This shows that the permutation matching can find a better mapping, π , for wider models, explaining why the permuted mask works better in the case of wider models.

4.4 Additional Results and Details

4.4.1 Architectural Details

4.4.1.1 ResNet20 & VGG11 on CIFAR-10/100

Datasets For our set of experiments, we used the CIFAR-10 and CIFAR-100 datasets (Krizhevsky, 2009). We apply the following standard data augmentation techniques to the training set:

- `RandomHorizontalFlip`: Randomly flips the image horizontally with a given probability (by default, 50%).
- `RandomCrop`: Randomly crops the image to a size of 32×32 pixels, with a padding of 4 pixels around the image.

Optimizers We use the following hyperparameters for ResNet20 and VGG11 trained on CIFAR-10/100, as outlined in table 4.4.

Hyperparameter	Value
Optimizer	SGD
Momentum	0.9
Dense Learning Rate	0.08
Sparse Learning Rate	0.02
Weight Decay	5×10^{-4}
Batch Size	128
Epochs (T)	200

Table 4.4: Hyperparameters for dense and sparse training of both ResNet20 and VGG11.

4.4.1.2 ResNet50 on ImageNet

Dataset For our set of experiments we used the ImageNet dataset (Deng et al., 2009). We apply the following standard data augmentation techniques to the training set:

- `RandomHorizontalFlip`: Randomly flips the image horizontally with a given probability (by default, 50%).
- `RandomResizedCrop`: Randomly crops a region from the image and resizes it to 224×224 pixels.

Optimizers We use the following hyperparameters for ResNet50 trained on ImageNet, as outlined in table 4.5.

Hyperparameter	Value
Optimizer	SGD
Momentum	0.9
Dense Learning Rate	0.4
Sparse Learning Rate	0.4
Weight Decay	1×10^{-4}
Batch Size	1024
Epochs (T)	80

Table 4.5: Hyperparameters for dense and sparse training of ResNet50.

4.4.2 Pruning Hyperparameters

Hyperparameter	ResNet20/VGG11	ResNet50
<code>train_epochs_per_prune</code>	50	20
Learning Rate	0.01	0.04

Table 4.6: Hyperparameters used for pruning ResNet20/VGG11 on CIFAR-10/100 and ResNet50 on ImageNet.

4. Experiments and Results

Table 4.8: **ResNet20×{4}/CIFAR-10**. Results using the ResNet20×{4} trained on CIFAR-10, from a rewind point k , using various methods of sparse training with sparsity S . LTH trains within the original dense/pruned solution basin, while naive/permutated train from a new random initialization. Note this table is the same setting as table 4.7 except $w = 4$.

S	Method	Rewind Epoch k								
		$k=0$	5	10	15	20	25	50	75	100
80%	LTH	94.67 ± 0.14	95.57 ± 0.05	95.84 ± 0.15	95.80 ± 0.12	95.88 ± 0.20	95.72 ± 0.09	95.81 ± 0.10	95.83 ± 0.21	95.71 ± 0.16
	naive	94.36 ± 0.04	94.55 ± 0.14	94.59 ± 0.29	94.74 ± 0.13	94.69 ± 0.09	94.81 ± 0.06	95.07 ± 0.17	95.02 ± 0.11	94.97 ± 0.21
	perm.	94.39 ± 0.19	94.88 ± 0.28	95.15 ± 0.14	95.20 ± 0.16	95.17 ± 0.21	95.28 ± 0.29	95.43 ± 0.14	95.40 ± 0.10	95.30 ± 0.08
90%	LTH	94.43 ± 0.17	95.53 ± 0.21	95.63 ± 0.07	95.65 ± 0.30	95.66 ± 0.07	95.61 ± 0.14	95.56 ± 0.16	95.62 ± 0.14	95.50 ± 0.04
	naive	93.79 ± 0.15	93.96 ± 0.05	94.09 ± 0.11	94.20 ± 0.29	94.35 ± 0.25	94.20 ± 0.13	94.27 ± 0.19	94.23 ± 0.08	94.19 ± 0.27
	perm.	93.97 ± 0.29	94.64 ± 0.13	94.73 ± 0.17	94.93 ± 0.12	94.92 ± 0.11	94.90 ± 0.07	95.04 ± 0.14	95.07 ± 0.18	94.91 ± 0.19
95%	LTH	93.65 ± 0.12	95.26 ± 0.08	95.39 ± 0.05	95.32 ± 0.18	95.26 ± 0.03	95.33 ± 0.07	95.40 ± 0.14	95.19 ± 0.05	95.37 ± 0.21
	naive	93.27 ± 0.07	93.30 ± 0.11	93.63 ± 0.04	93.61 ± 0.21	93.66 ± 0.13	93.67 ± 0.14	93.43 ± 0.21	93.51 ± 0.32	93.14 ± 0.03
	perm.	93.54 ± 0.24	94.17 ± 0.07	94.46 ± 0.10	94.27 ± 0.19	94.61 ± 0.07	94.54 ± 0.07	94.75 ± 0.11	94.75 ± 0.09	94.54 ± 0.27
97%	LTH	93.00 ± 0.11	94.77 ± 0.09	94.86 ± 0.06	94.94 ± 0.17	94.96 ± 0.06	94.89 ± 0.21	95.00 ± 0.24	94.94 ± 0.10	94.97 ± 0.13
	naive	92.63 ± 0.12	92.80 ± 0.10	92.85 ± 0.21	92.66 ± 0.21	92.74 ± 0.11	92.69 ± 0.14	92.28 ± 0.09	92.02 ± 0.18	91.87 ± 0.10
	perm.	92.81 ± 0.27	93.54 ± 0.08	93.83 ± 0.12	93.75 ± 0.34	94.00 ± 0.33	94.12 ± 0.04	94.07 ± 0.31	94.32 ± 0.24	94.14 ± 0.04

Detailed results for ResNet20×{ w }/CIFAR-10 are provided in tables 4.7 to 4.10, for VGG11×{1}/CIFAR-10 in table 4.11, for ResNet50×{1}/ImageNet in table 4.12, and for ResNet20×{ w }/CIFAR-100 in tables 4.13 to 4.16.

Table 4.7: **ResNet20×{1}/CIFAR-10**. Results using the ResNet20×{1} trained on CIFAR-10, from a rewind point k , using various methods of sparse training with sparsity S . LTH trains within the original dense/pruned solution basin, while naive/permutated train from a new random initialization.

S	Method	Rewind Epoch k								
		$k=0$	5	10	15	20	25	50	75	100
80%	LTH	90.41 ± 0.14	92.12 ± 0.25	92.08 ± 0.36	92.10 ± 0.27	92.25 ± 0.14	92.32 ± 0.26	92.15 ± 0.13	92.26 ± 0.19	92.21 ± 0.16
	naive	89.67 ± 0.35	89.74 ± 0.69	90.16 ± 0.14	90.07 ± 0.09	90.13 ± 0.11	90.40 ± 0.11	90.66 ± 0.12	90.31 ± 0.27	90.45 ± 0.22
	perm.	89.74 ± 0.05	90.15 ± 0.16	90.26 ± 0.08	90.72 ± 0.12	90.68 ± 0.18	90.72 ± 0.28	90.76 ± 0.27	91.13 ± 0.06	90.82 ± 0.21
90%	LTH	89.45 ± 0.10	91.27 ± 0.37	91.34 ± 0.29	91.34 ± 0.09	91.18 ± 0.27	91.43 ± 0.22	91.44 ± 0.12	91.36 ± 0.18	91.68 ± 0.28
	naive	88.47 ± 0.21	88.70 ± 0.14	88.77 ± 0.21	88.84 ± 0.43	88.83 ± 0.27	88.78 ± 0.02	88.99 ± 0.08	88.81 ± 0.17	88.82 ± 0.07
	perm.	88.59 ± 0.11	89.09 ± 0.22	89.56 ± 0.28	89.71 ± 0.12	89.50 ± 0.27	89.97 ± 0.13	89.84 ± 0.15	90.03 ± 0.07	89.77 ± 0.15
95%	LTH	87.83 ± 0.38	90.33 ± 0.22	90.39 ± 0.28	90.37 ± 0.21	90.58 ± 0.26	90.43 ± 0.20	90.56 ± 0.29	90.44 ± 0.26	90.40 ± 0.19
	naive	86.89 ± 0.21	87.01 ± 0.23	86.88 ± 0.13	87.28 ± 0.19	87.31 ± 0.36	87.00 ± 0.19	86.88 ± 0.08	86.99 ± 0.29	86.50 ± 0.22
	perm.	87.24 ± 0.22	87.70 ± 0.08	87.92 ± 0.25	88.23 ± 0.52	88.29 ± 0.52	88.24 ± 0.20	88.21 ± 0.30	88.21 ± 0.20	88.04 ± 0.22
97%	LTH	86.03 ± 0.22	88.00 ± 0.02	88.73 ± 0.05	89.00 ± 0.24	89.21 ± 0.23	89.27 ± 0.14	89.03 ± 0.27	89.12 ± 0.25	89.06 ± 0.21
	naive	85.60 ± 0.38	85.43 ± 0.40	85.89 ± 0.37	85.48 ± 0.13	85.36 ± 0.14	85.70 ± 0.21	85.30 ± 0.32	85.14 ± 0.29	84.64 ± 0.34
	perm.	85.61 ± 0.48	85.93 ± 0.34	86.26 ± 0.40	86.48 ± 0.39	86.12 ± 0.27	86.16 ± 0.14	86.43 ± 0.27	86.06 ± 0.26	85.95 ± 0.14

Refer to Figure 4.8 and Figure 4.9 for additional accuracy-vs-sparsity plots for ResNet20 on CIFAR-10 and CIFAR-100, respectively. Refer to Figure 4.10 for Top-1 accuracy vs. rewind points for ResNet50 on ImageNet.

4. Experiments and Results

Table 4.9: **ResNet20×{8}/CIFAR-10**. Results using the ResNet20×{8} trained on CIFAR-10, from a rewind point k , using various methods of sparse training with sparsity S . LTH trains within the original dense/pruned solution basin, while naive/permutated train from a new random initialization. Note this table is the same setting as table 4.7 except $w = 8$.

S	Method	Rewind Epoch k				
		$k=0$	10	25	50	100
80%	LTH	95.35 ± 0.07	95.98 ± 0.14	96.12 ± 0.04	96.10 ± 0.20	96.21 ± 0.06
	naive	95.17 ± 0.17	95.32 ± 0.13	95.63 ± 0.13	95.62 ± 0.08	95.79 ± 0.15
	perm.	95.36 ± 0.14	95.60 ± 0.15	95.89 ± 0.19	95.94 ± 0.17	95.94 ± 0.06
90%	LTH	94.96 ± 0.18	95.97 ± 0.15	96.02 ± 0.05	96.00 ± 0.19	96.12 ± 0.10
	naive	95.05 ± 0.07	95.12 ± 0.03	95.20 ± 0.22	95.44 ± 0.14	95.06 ± 0.25
	perm.	95.05 ± 0.05	95.58 ± 0.06	95.78 ± 0.12	95.87 ± 0.13	95.85 ± 0.11
95%	LTH	94.86 ± 0.08	95.90 ± 0.15	95.93 ± 0.26	96.07 ± 0.25	96.00 ± 0.25
	naive	94.60 ± 0.14	94.84 ± 0.13	94.93 ± 0.17	95.01 ± 0.33	94.59 ± 0.52
	perm.	94.85 ± 0.19	95.29 ± 0.27	95.63 ± 0.11	95.67 ± 0.16	95.59 ± 0.22
97%	LTH	94.54 ± 0.23	95.79 ± 0.14	95.87 ± 0.03	95.78 ± 0.21	95.90 ± 0.04
	naive	94.39 ± 0.04	94.39 ± 0.04	94.49 ± 0.18	94.19 ± 0.11	93.83 ± 0.08
	perm.	94.46 ± 0.14	95.26 ± 0.10	95.16 ± 0.26	95.56 ± 0.06	95.45 ± 0.05

Table 4.10: **ResNet20×{16}/CIFAR-10**. Results using the ResNet20×{8} trained on CIFAR-10, from a rewind point k , using various methods of sparse training with sparsity S . LTH trains within the original dense/pruned solution basin, while naive/permutated train from a new random initialization. Note this table is the same setting as table 4.7 except $w = 16$.

S	Method	Rewind Epoch k				
		$k=0$	10	25	50	100
80%	LTH	95.62 ± 0.19	95.84 ± 0.36	96.05 ± 0.34	96.31 ± 0.18	96.36 ± 0.24
	naive	95.47 ± 0.15	95.71 ± 0.22	95.71 ± 0.26	96.09 ± 0.04	95.99 ± 0.21
	perm.	95.77 ± 0.11	95.79 ± 0.29	96.00 ± 0.14	96.24 ± 0.11	96.21 ± 0.06
90%	LTH	95.59 ± 0.22	96.10 ± 0.48	96.19 ± 0.49	96.18 ± 0.20	96.41 ± 0.14
	naive	95.37 ± 0.09	95.47 ± 0.13	95.66 ± 0.01	95.70 ± 0.13	95.76 ± 0.14
	perm.	95.58 ± 0.22	95.80 ± 0.14	96.11 ± 0.13	96.17 ± 0.17	96.04 ± 0.05
95%	LTH	95.08 ± 0.21	95.96 ± 0.39	96.12 ± 0.21	96.16 ± 0.30	96.26 ± 0.23
	naive	95.27 ± 0.13	95.43 ± 0.09	95.57 ± 0.37	95.63 ± 0.25	95.27 ± 0.55
	perm.	95.39 ± 0.26	96.02 ± 0.22	96.12 ± 0.18	96.18 ± 0.18	96.06 ± 0.09
97%	LTH	95.19 ± 0.27	95.84 ± 0.25	96.14 ± 0.30	96.12 ± 0.27	96.17 ± 0.33
	naive	94.94 ± 0.04	95.06 ± 0.17	95.29 ± 0.15	95.13 ± 0.19	94.35 ± 0.45
	perm.	95.07 ± 0.06	95.51 ± 0.22	95.88 ± 0.14	95.90 ± 0.24	95.88 ± 0.09

Table 4.11: **VGG11 $\times\{1\}$ /CIFAR-10**. Results using the VGG11 trained on CIFAR-10, from a rewind point k , using various methods of sparse training with sparsity S . LTH trains within the original dense/pruned solution basin, while naive/permutated train from a new random initialization.

S	Method	Rewind Epoch k						
		$k = 0$	5	10	15	20	25	50
80%	LTH	89.94 \pm 0.06	90.44 \pm 0.17	90.91 \pm 0.12	90.87 \pm 0.16	91.14 \pm 0.28	91.11 \pm 0.08	91.22 \pm 0.08
	naive	89.70 \pm 0.13	89.90 \pm 0.18	90.04 \pm 0.07	90.34 \pm 0.16	90.48 \pm 0.19	90.55 \pm 0.17	90.87 \pm 0.19
	perm.	89.94 \pm 0.1	90.18 \pm 0.08	90.52 \pm 0.17	90.71 \pm 0.22	90.77 \pm 0.19	90.81 \pm 0.19	91.07 \pm 0.21
90%	LTH	89.33 \pm 0.16	90.82 \pm 0.09	90.97 \pm 0.14	91.05 \pm 0.04	91.15 \pm 0.11	90.91 \pm 0.17	91.08 \pm 0.31
	naive	89.17 \pm 0.2	89.55 \pm 0.02	89.81 \pm 0.02	89.49 \pm 0.05	89.68 \pm 0.11	89.80 \pm 0.03	89.80 \pm 0.05
	perm.	89.30 \pm 0.02	90.33 \pm 0.08	90.44 \pm 0.14	90.46 \pm 0.04	90.75 \pm 0.22	90.76 \pm 0.12	91.01 \pm 0.06

Table 4.12: **ResNet50 $\times\{1\}$ /ImageNet**. Top-1 and Top-5 Accuracies of ResNet50 $\times\{1\}$ trained on ImageNet, from a rewind point k , using various methods of sparse training with sparsity S .

S	Method	Top-1 Accuracy			Top-5 Accuracy		
		$k = 10$	25	50	$k = 10$	25	50
80%	LTH	72.87	72.16	65.23	91.13	90.66	86.65
	naive	69.13	68.94	60.30	88.85	88.1	83.22
	perm.	69.87	69.85	61.14	89.16	89.45	84.04
90%	LTH	71.40	70.74	60.62	90.27	90.00	83.94
	naive	65.49	64.77	54.46	86.55	86.26	79.07
	perm.	66.25	66.37	57.40	87.23	87.37	81.45
95%	LTH	68.61	68.07	59.83	89.03	88.25	82.96
	naive	61.39	60.77	51.78	83.79	83.58	76.79
	perm.	62.48	62.77	52.98	84.51	84.79	78.11

4. Experiments and Results

Table 4.13: **ResNet20×{1}/CIFAR-100**. Results using the ResNet20×{1} trained on CIFAR-100, from a rewind point k , using various methods of sparse training with sparsity S . LTH trains within the original dense/pruned solution basin, while naive/permutated train from a new random initialization.

S	Method	Rewind Epoch k				
		$k=0$	10	25	50	100
80%	LTH	63.69 ± 0.41	67.67 ± 0.08	67.66 ± 0.25	67.82 ± 0.17	67.73 ± 0.38
	naive	62.89 ± 0.16	63.37 ± 0.09	63.07 ± 0.44	63.36 ± 0.27	63.33 ± 0.35
	perm.	63.04 ± 0.24	64.07 ± 0.15	64.71 ± 0.10	64.52 ± 0.78	64.57 ± 0.49
90%	LTH	59.81 ± 0.29	65.21 ± 0.17	65.15 ± 0.28	65.10 ± 0.30	65.17 ± 0.21
	naive	58.77 ± 0.28	59.59 ± 0.18	59.44 ± 0.27	59.19 ± 0.41	58.58 ± 0.16
	perm.	59.32 ± 0.32	60.60 ± 0.79	61.32 ± 0.33	61.53 ± 0.65	60.93 ± 0.51
95%	LTH	55.71 ± 0.52	61.08 ± 0.54	61.73 ± 0.18	61.65 ± 0.37	61.68 ± 0.18
	naive	54.04 ± 0.29	55.20 ± 0.39	54.65 ± 0.38	54.96 ± 0.57	53.97 ± 0.91
	perm.	55.12 ± 0.17	56.93 ± 0.26	57.64 ± 0.36	57.47 ± 0.66	57.13 ± 0.34
97%	LTH	51.10 ± 0.34	56.14 ± 0.56	56.92 ± 0.25	56.94 ± 0.13	56.93 ± 0.06
	naive	49.70 ± 0.64	49.60 ± 0.25	49.49 ± 0.32	49.16 ± 0.21	47.70 ± 0.83
	perm.	50.34 ± 0.21	51.55 ± 0.69	51.88 ± 1.08	52.64 ± 0.34	50.96 ± 1.15

Table 4.14: **ResNet20×{4}/CIFAR-100**. Results using the ResNet20×{4} trained on CIFAR-100, from a rewind point k , using various methods of sparse training with sparsity S . LTH trains within the original dense/pruned solution basin, while naive/permutated train from a new random initialization. Note this table is the same setting as table 4.13 except $w = 4$.

S	Method	Rewind Epoch k				
		$k=0$	10	25	50	100
80%	LTH	74.46 ± 0.12	77.57 ± 0.06	77.35 ± 0.31	77.75 ± 0.26	77.64 ± 0.14
	naive	73.30 ± 0.08	74.10 ± 0.12	74.98 ± 0.17	75.21 ± 0.12	75.20 ± 0.16
	perm.	73.68 ± 0.09	75.24 ± 0.31	75.74 ± 0.41	76.12 ± 0.37	76.19 ± 0.39
90%	LTH	72.54 ± 0.57	76.56 ± 0.11	76.56 ± 0.32	76.80 ± 0.34	76.80 ± 0.21
	naive	71.97 ± 0.30	72.56 ± 0.22	72.89 ± 0.27	72.59 ± 0.15	72.54 ± 0.33
	perm.	72.18 ± 0.23	74.17 ± 0.35	74.21 ± 0.23	74.45 ± 0.27	74.89 ± 0.47
95%	LTH	71.16 ± 0.23	75.41 ± 0.18	75.53 ± 0.11	75.68 ± 0.17	75.76 ± 0.17
	naive	70.17 ± 0.47	70.95 ± 0.50	70.90 ± 0.18	71.21 ± 0.26	69.95 ± 0.42
	perm.	70.41 ± 0.07	72.70 ± 0.21	72.92 ± 0.39	73.65 ± 0.28	73.41 ± 0.18
97%	LTH	69.06 ± 0.03	74.00 ± 0.39	74.08 ± 0.37	74.18 ± 0.18	74.29 ± 0.31
	naive	68.40 ± 0.21	69.26 ± 0.19	69.06 ± 0.11	68.67 ± 0.47	68.42 ± 0.78
	perm.	69.08 ± 0.22	71.41 ± 0.54	71.49 ± 0.32	71.92 ± 0.17	72.20 ± 0.08

4. Experiments and Results

Table 4.15: **ResNet20×{8}/CIFAR-100**. Results using the ResNet20×{8} trained on CIFAR-100, from a rewind point k , using various methods of sparse training with sparsity S . LTH trains within the original dense/pruned solution basin, while naive/permutated train from a new random initialization. Note this table is the same setting as table 4.13 except $w = 8$.

S	Method	Rewind Epoch k				
		$k=0$	10	25	50	100
80%	LTH	78.09 ± 0.28	80.63 ± 0.32	80.83 ± 0.39	80.92 ± 0.06	80.66 ± 0.34
	naive	76.86 ± 0.17	77.47 ± 0.35	78.20 ± 0.61	78.65 ± 0.33	78.74 ± 0.39
	perm.	77.34 ± 0.26	78.82 ± 0.34	79.20 ± 0.16	79.55 ± 0.38	79.54 ± 0.39
90%	LTH	76.47 ± 0.43	80.02 ± 0.07	80.10 ± 0.13	79.98 ± 0.33	79.98 ± 0.20
	naive	75.68 ± 0.23	76.36 ± 0.21	76.80 ± 0.14	77.27 ± 0.12	76.55 ± 0.49
	perm.	76.17 ± 0.26	77.99 ± 0.17	78.22 ± 0.15	78.62 ± 0.19	78.82 ± 0.17
95%	LTH	75.38 ± 0.02	79.42 ± 0.06	79.24 ± 0.19	79.35 ± 0.06	79.29 ± 0.13
	naive	74.78 ± 0.15	75.48 ± 0.18	75.53 ± 0.15	75.27 ± 0.15	74.38 ± 0.65
	perm.	75.07 ± 0.14	76.97 ± 0.46	77.80 ± 0.14	77.74 ± 0.51	78.04 ± 0.42
97%	LTH	73.97 ± 0.21	78.63 ± 0.25	78.65 ± 0.50	78.74 ± 0.49	78.47 ± 0.16
	naive	73.13 ± 0.26	73.73 ± 0.12	73.76 ± 0.27	73.26 ± 0.07	72.79 ± 0.46
	perm.	73.81 ± 0.67	76.29 ± 0.14	76.38 ± 0.57	76.57 ± 0.29	76.79 ± 0.76

Table 4.16: **ResNet20×{16}/CIFAR-100**. Results using the ResNet20×{16} trained on CIFAR-100, from a rewind point k , using various methods of sparse training with sparsity S . LTH trains within the original dense/pruned solution basin, while naive/permutated train from a new random initialization. Note this table is the same setting as table 4.13 except $w = 16$.

S	Method	Rewind Epoch k				
		$k=0$	10	25	50	100
80%	LTH	80.21 ± 0.18	82.32 ± 0.34	82.40 ± 0.26	82.48 ± 0.38	82.16 ± 0.30
	naive	79.31 ± 0.06	79.50 ± 0.09	80.24 ± 0.17	81.02 ± 0.11	81.01 ± 0.07
	perm.	79.35 ± 0.11	80.44 ± 0.40	81.15 ± 0.48	81.57 ± 0.38	81.81 ± 0.21
90%	LTH	79.31 ± 0.16	82.26 ± 0.18	82.14 ± 0.08	81.95 ± 0.03	82.11 ± 0.12
	naive	78.78 ± 0.37	79.26 ± 0.11	79.42 ± 0.51	79.56 ± 0.26	79.57 ± 0.13
	perm.	79.20 ± 0.09	80.49 ± 0.32	80.59 ± 0.15	81.12 ± 0.05	81.24 ± 0.09
95%	LTH	78.32 ± 0.34	81.57 ± 0.09	81.57 ± 0.32	81.47 ± 0.25	81.63 ± 0.07
	naive	78.01 ± 0.02	78.53 ± 0.10	78.45 ± 0.21	78.38 ± 0.43	77.49 ± 0.06
	perm.	78.25 ± 0.20	79.76 ± 0.20	80.50 ± 0.04	80.47 ± 0.08	80.25 ± 0.21
97%	LTH	77.49 ± 0.27	81.07 ± 0.07	81.06 ± 0.11	81.11 ± 0.18	81.14 ± 0.32
	naive	76.46 ± 0.44	76.71 ± 0.41	77.19 ± 0.09	76.93 ± 0.36	75.53 ± 0.40
	perm.	77.04 ± 0.38	79.14 ± 0.17	79.30 ± 0.21	79.62 ± 0.14	79.63 ± 0.06

4. Experiments and Results

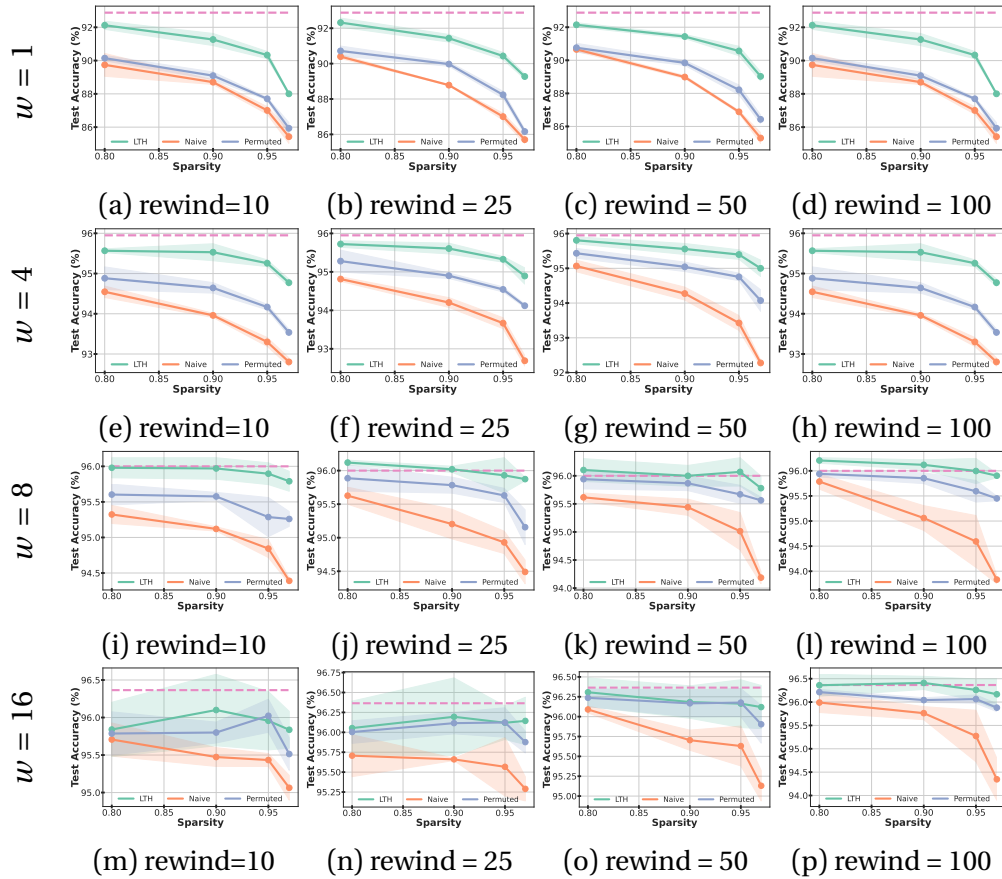


Figure 4.8: Accuracy vs sparsity trend for ResNet20 $\times\{w\}$ /CIFAR-10. As the width increases, the gap between permuted and naive solutions increases, showing permuted masks help with sparse training. With increased width, we observe a more significant gap seen throughout figs. 4.8d, 4.8h, 4.8l and 4.8p and the permuted solution approaches the LTH solution. The dashed (- -) line shows the dense model accuracy.

4. Experiments and Results

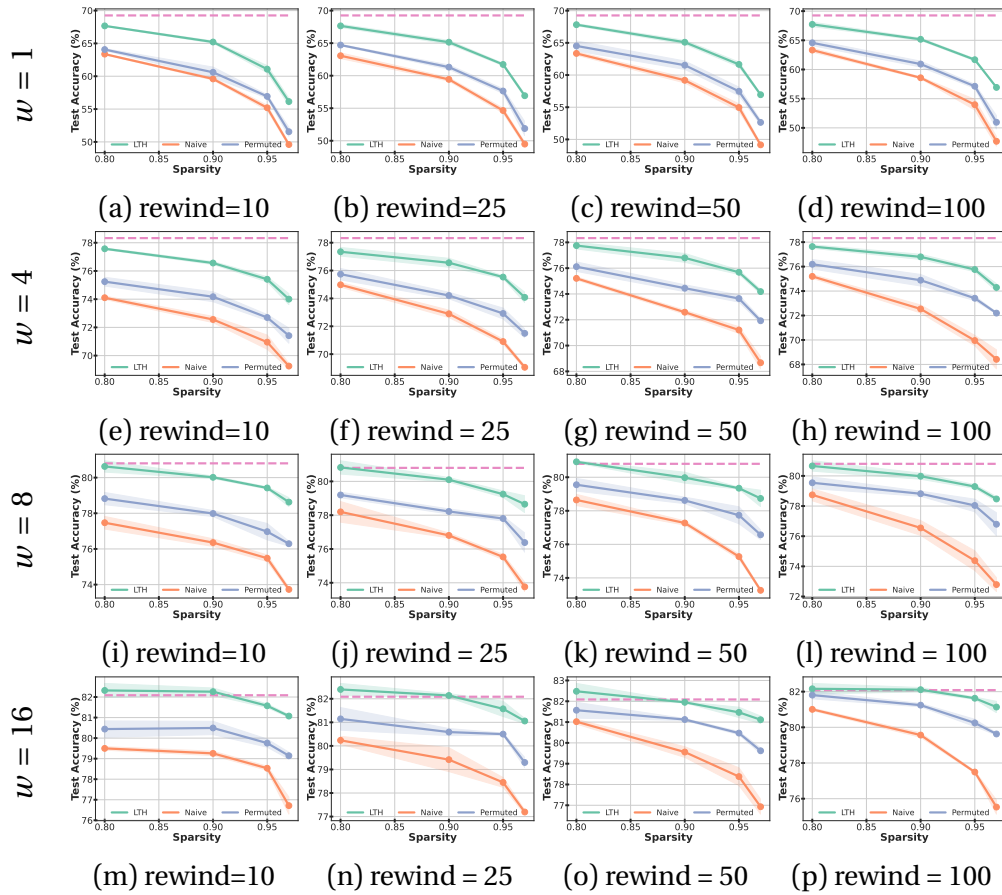


Figure 4.9: **Accuracy vs sparsity trend for ResNet20 \times { w }/CIFAR-100.** Similar to the phenomenon seen in fig. 4.8, with higher width the gap between permuted and naive solutions increases. As seen in figs. 4.9d, 4.9h, 4.9l and 4.9p and the permuted solution approaches the LTH solution. The dashed (- -) line shows the dense model accuracy.

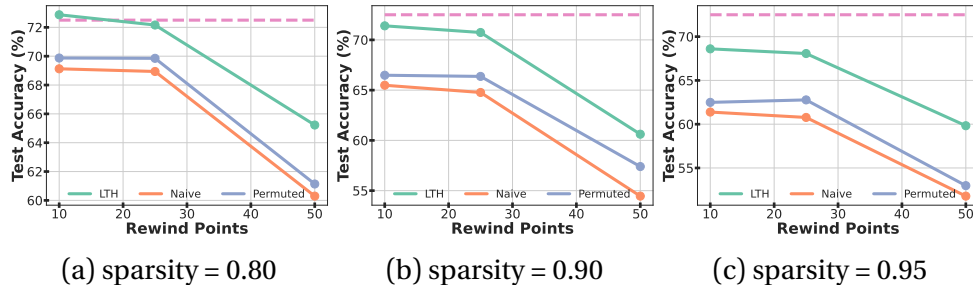


Figure 4.10: **ResNet50 \times {1}/ImageNet.** Top-1 test accuracy vs rewinds points of sparse network solutions at various sparsity levels. We observe the permuted solution consistently performing better than the naive solution for all sparsities. The dashed (- -) line shows the dense model accuracy.

Chapter 5

Conclusion

5.1 Discussion

The remarkable success of DNNs has revolutionized various fields, achieving significant breakthroughs in tasks ranging from image recognition, and medical imaging, to natural language processing and beyond. However, this success is often accompanied by vastly overparameterized, increasingly large, and complex models, demanding substantial computational resources for both training and inference. In resource-constrained environments, this presents a significant challenge, motivating the rise of SNN and the development of efficient training methodologies. A critical area of research in this domain has been the ability to train highly sparse SNN from scratch, with works such as the LTH suggesting the existence of sparse subnetworks within pre-trained dense models that can achieve comparable generalization performance to their dense counterparts. However, LTH does not solve the sparse training problem, as retraining a winning ticket mask with an arbitrary weight initialization often results in poor generalization

performance. In this work, we present new insights into sparse training from random initialization and the LTH by leveraging the inherent weight symmetries in DNNs. As shown in fig. 3.3, we demonstrate that by finding a permutation to align two arbitrary networks and subsequently permuting the mask to align with the loss basin in which the new initialization resides, effective sparse training can be enabled. Our empirical findings across various models and datasets support the hypothesis that misalignment between the mask and the loss basin prevents the effective use of LTH masks with new initializations. Although finding a permutation to align dense models can be computationally expensive, the primary goal of our work is to develop a deeper understanding of the limitations of LTH and explore how the winning ticket mask can be reused effectively, rather than to improve the efficiency of LTH itself. We hope that our work will spur future work in this direction and will be useful to the research community working in the realm of sparse training.

5.2 Limitations and Future Directions

- **Imperfect Neuronal Alignment:** Our procedure for neuronal alignment relies on activation matching, which is a greedy approximation. The stochasticity present in the data, coupled with the computational constraints of LAP, means that obtaining a perfect one-to-one correspondence between neurons is not always possible. Furthermore, the problem of finding a perfect permutation to achieve a completely zero loss barrier between two models after accounting for all possible permutation symmetries is classified as NP-hard. Potentially, using

more advanced neuronal alignment techniques could lead to identify stronger permutations, potentially enabling our permuted solutions to surpass the performance of the LTH baseline and approach dense.

- **Early Matching:** Throughout our study, we focused on training both models A and B to convergence to derive the permutation, which introduces a significant computational cost. While we presented some preliminary results in section 4.1.4, a more rigorous experimental investigation across a diverse set of models and datasets would be required to affirmatively establish the viability of early matching. Another compelling direction for future research would involve exploring techniques to effectively align a densely converged model (i.e., at $t = T$) with a dense model at its random initialization (i.e., at $t = 0$).
- **Larger Models:** Our current experimental design focused on CNNs and RNNs, supported by existing literature demonstrating their alignment in weight space due to the existence of permutation symmetries in these architectures. However, with the increasing prominence of Transformer models, investigating whether our hypothesis extends to these larger architectures would provide valuable insights. Most recently, [Zhang et al. \(2025\)](#) introduced “rotation symmetry” in Transformers, a novel type of parameter space symmetry that generalizes permutation symmetries. Potential future directions could involve further exploration of the optimization dynamics of these Transformer models and the development of methodologies to align them by leveraging these specific parameter space symmetries.

Bibliography

Samuel K. Ainsworth, Jonathan Hayase, and Siddhartha S. Srinivasa. Git re-basin: Merging models modulo permutation symmetries. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023.

Zeyuan Allen-Zhu, Yuanzhi Li, and Zhao Song. A convergence theory for deep learning via over-parameterization, 2019. URL <https://arxiv.org/abs/1811.03962>.

Yossi Arjevani and Michael Field. Annihilation of spurious minima in two-layer relu networks, 2022. URL <https://arxiv.org/abs/2210.06088>.

Carlo Baldassi, Clarissa Lauditi, Enrico M. Malatesta, Rosalba Pacelli, Gabriele Perugini, and Riccardo Zecchina. Learning through atypical phase transitions in overparameterized neural networks. *Phys. Rev. E*, 106:014116, Jul 2022. doi: 10.1103/PhysRevE.106.014116. URL <https://link.aps.org/doi/10.1103/PhysRevE.106.014116>.

Frederik Benzing, Simon Schug, Robert Meier, Johannes von Oswald, Yassir Akram, Nicolas Zucchet, Laurence Aitchison, and Angelika Steger. Random initialisations performing above chance and how to find them, 2022.

Dimitri Bertsekas. *Network optimization: continuous and discrete models*, volume 8. Athena Scientific, 1998.

Davis W. Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John V. Guttag. What is the state of neural network pruning? *ArXiv*, abs/2003.03033, 2020. URL <https://api.semanticscholar.org/CorpusID:212628335>.

Johanni Brea, Berfin Simsek, Bernd Illing, and Wulfram Gerstner. Weight-space symmetry in deep networks gives rise to permutation saddles, connected by equal-loss valleys across the loss landscape. *CoRR*, abs/1907.02911, 2019.

Cerebras. Harnessing the power of sparsity for large gpt ai models, 2022. URL <https://www.cerebras.ai/blog/harnessing-the-power-of-sparsity-for-large-gpt-ai-models>. Accessed: March 31, 2025.

Hongrong Cheng, Miao Zhang, and Javen Qinfeng Shi. A survey on deep neural network pruning: Taxonomy, comparison, analysis, and recommendations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 46:10558–10578, 2023. URL <https://api.semanticscholar.org/CorpusID:260887757>.

Anna Choromanska, Mikael Henaff, Michael Mathieu, Gerard Ben Arous, and Yann LeCun. The Loss Surfaces of Multilayer Networks. In Guy Lebanon and S. V. N. Vishwanathan, editors, *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics*, volume 38

of *Proceedings of Machine Learning Research*, pages 192–204, San Diego, California, USA, 09–12 May 2015. PMLR.

PyTorch Contributors. Pytorch examples, 2024. Accessed: 2024-11-26.

Yaim Cooper. The loss landscape of overparameterized neural networks. *ArXiv*, abs/1804.10200, 2018. URL <https://api.semanticscholar.org/CorpusID:13752493>.

Yann N. Dauphin, Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, Surya Ganguli, and Yoshua Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NeurIPS’14, page 2933–2941, Cambridge, MA, USA, 2014. MIT Press.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2009), 20-25 June 2009, Miami, Florida, USA*, pages 248–255. IEEE Computer Society, 2009. doi: 10.1109/CVPR.2009.5206848.

Laurent Dinh, Razvan Pascanu, Samy Bengio, and Yoshua Bengio. Sharp minima can generalize for deep nets. In *International Conference on Machine Learning*, 2017. URL <https://api.semanticscholar.org/CorpusID:7636159>.

Felix Draxler, Kambis Veschgini, Manfred Salmhofer, and Fred A. Hamprecht. Essentially no barriers in neural network energy landscape. In Jennifer G.

Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 1308–1317. PMLR, 2018.

Rahim Entezari, Hanie Sedghi, Olga Saukh, and Behnam Neyshabur. The role of permutation invariance in linear mode connectivity of neural networks. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022.

Utku Evci, Trevor Gale, Jacob Menick, Pablo Samuel Castro, and Erich Elsen. Rigging the lottery: Making all tickets winners, 2021. URL <https://arxiv.org/abs/1911.11134>.

Utku Evci, Yani Ioannou, Cem Keskin, and Yann N. Dauphin. Gradient flow in sparse neural networks and how lottery tickets win. In *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI 2022, The Twelveth Symposium on Educational Advances in Artificial Intelligence, EAAI 2022 Virtual Event, February 22 - March 1, 2022*, pages 6577–6586. AAAI Press, 2022. doi: 10.1609/AAAI.V36I6.20611.

Stanislav Fort, Huiyi Hu, and Balaji Lakshminarayanan. Deep ensembles: A loss landscape perspective, 2020.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

Jonathan Frankle, Gintare Karolina Dziugaite, Daniel M. Roy, and Michael Carbin. Pruning neural networks at initialization: Why are we missing the mark? *ArXiv*, abs/2009.08576, 2020a. URL <https://api.semanticscholar.org/CorpusID:221802286>.

Jonathan Frankle, Gintare Karolina Dziugaite, Daniel M. Roy, and Michael Carbin. Linear mode connectivity and the lottery ticket hypothesis. In *Proceedings of the 37th International Conference on Machine Learning, ICML'20*. JMLR.org, 2020b.

Jonathan Frankle, Gintare Karolina Dziugaite, Daniel M. Roy, and Michael Carbin. Pruning neural networks at initialization: Why are we missing the mark? In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.

Trevor Gale, Erich Elsen, and Sara Hooker. The state of sparsity in deep neural networks. *CoRR*, abs/1902.09574, 2019.

Timur Garipov, Pavel Izmailov, Dmitrii Podoprikin, Dmitry P. Vetrov, and Andrew Gordon Wilson. Loss surfaces, mode connectivity, and fast ensembling of dnns. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 8803–8812, 2018.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *International Confer-*

ence on Artificial Intelligence and Statistics, 2010. URL <https://api.semanticscholar.org/CorpusID:5575601>.

Charles Godfrey, Davis Brown, Tegan Emerson, and Henry Kvinge. On the symmetries of deep learning models and their internal representations. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022.

Ian J. Goodfellow and Oriol Vinyals. Qualitatively characterizing neural network optimization problems. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

Jake Grigsby, Kathryn A. Lindsey, and David Rolnick. Hidden symmetries of relu networks. *ArXiv*, abs/2306.06179, 2023. URL <https://api.semanticscholar.org/CorpusID:259137546>.

Fidel A. Guerrero-Peña, Heitor Rapela Medeiros, Thomas Dubail, Masih Aminbeidokhti, Eric Granger, and Marco Pedersoli. Re-basin via implicit sinkhorn differentiation. *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 20237–20246, 2022. URL <https://api.semanticscholar.org/CorpusID:255096607>.

Janek Haberer and Olaf Landsiedel. Activation sparsity and dynamic pruning for split computing in edge ai. In *Proceedings of the 3rd International*

Workshop on Distributed Machine Learning, DistributedML '22, page 30–36, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450399227. doi: 10.1145/3565010.3569066. URL <https://doi.org/10.1145/3565010.3569066>.

Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both weights and connections for efficient neural network. In Corinna Cortes, Neil D. Lawrence, Daniel D. Lee, Masashi Sugiyama, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 1135–1143, 2015.

Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.

Moritz Hardt, Ben Recht, and Yoram Singer. Train faster, generalize better: Stability of stochastic gradient descent. In Maria-Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of the 33rd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 1225–1234. JMLR.org, 2016.

Babak Hassibi and David G. Stork. Second order derivatives for network pruning: Optimal brain surgeon. In *Neural Information Processing*

Systems, 1992. URL <https://api.semanticscholar.org/CorpusID:7057040>.

Haowei He, Gao Huang, and Yang Yuan. Asymmetric valleys: Beyond sharp and flat local minima. In *Neural Information Processing Systems*, 2019. URL <https://api.semanticscholar.org/CorpusID:59599897>.

Kaiming He, X. Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1026–1034, 2015. URL <https://api.semanticscholar.org/CorpusID:13740328>.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 770–778. IEEE Computer Society, 2016. doi: 10.1109/CVPR.2016.90.

Robert Hecht-Nielsen. On the algebraic structure of feedforward network weight spaces, 1990.

Sepp Hochreiter and Jürgen Schmidhuber. Flat minima. *Neural Computation*, 9:1–42, 1997. URL <https://api.semanticscholar.org/CorpusID:733161>.

Torsten Hoefler, Dan Alistarh, Tal Ben-Nun, Nikoli Dryden, and Alexandra Peste. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks, 2021. URL <https://arxiv.org/abs/2102.00554>.

Kevin Lee Hunter, Lawrence Spracklen, and Subutai Ahmad. Two sparsities are better than one: unlocking the performance benefits of sparse–sparse networks. *Neuromorphic Computing and Engineering*, 2, 2021. URL <https://api.semanticscholar.org/CorpusID:245537213>.

Yani Ioannou. Training unstructured sparse neural networks. Slide presentation, 2022. Presented in 2022. Slides available upon request.

Sergey Ioffe and Christian Szegedy. Batch normalization: accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37, ICML'15*, page 448–456. JMLR.org, 2015.

Akira Ito, Masanori Yamada, and Atsutoshi Kumagai. Analysis of linear mode connectivity via permutation-based weight matching. *CoRR*, abs/2402.04051, 2024. doi: 10.48550/ARXIV.2402.04051.

Chi Jin, Rong Ge, Praneeth Netrapalli, Sham M. Kakade, and Michael I. Jordan. How to escape saddle points efficiently. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 1724–1732. PMLR, 2017.

Keller Jordan, Hanie Sedghi, Olga Saukh, Rahim Entezari, and Behnam Neyshabur. REPAIR: renormalizing permuted activations for interpolation repair. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023.

- Kedar Karhadkar, Michael Murray, Hanna Tseran, and Guido Montúfar. Mildly overparameterized relu networks have a favorable loss landscape, 2024. URL <https://arxiv.org/abs/2305.19510>.
- Kenji Kawaguchi. Deep learning without poor local minima. In Daniel D. Lee, Masashi Sugiyama, Ulrike von Luxburg, Isabelle Guyon, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 586–594, 2016.
- Kenji Kawaguchi, Leslie Pack Kaelbling, and Yoshua Bengio. Generalization in deep learning. *ArXiv*, abs/1710.05468, 2017. URL <https://api.semanticscholar.org/CorpusID:3520119>.
- Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.
- Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.
- Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60:84 – 90, 2012. URL <https://api.semanticscholar.org/CorpusID:195908774>.
- Harold W. Kuhn. The hungarian method for the assignment problem. In Michael Jünger, Thomas M. Lieblich, Denis Naddef, George L. Nemhauser,

- William R. Pulleyblank, Gerhard Reinelt, Giovanni Rinaldi, and Laurence A. Wolsey, editors, *50 Years of Integer Programming 1958-2008 - From the Early Years to the State-of-the-Art*, pages 29–47. Springer, 2010. doi: 10.1007/978-3-540-68279-0_2.
- Mike Lasby, Anna Golubeva, Utku Evci, Mihai Nica, and Yani Ioannou. Dynamic sparse training with structured sparsity, 2024. URL <https://arxiv.org/abs/2305.02299>.
- Yann LeCun, Bernhard Boser, John Denker, Donnie Henderson, Richard Howard, Wayne Hubbard, and Lawrence Jackel. Handwritten digit recognition with a back-propagation network. *Advances in neural information processing systems*, 2, 1989a.
- Yann LeCun, John S. Denker, and Sara A. Solla. Optimal brain damage. In *Neural Information Processing Systems*, 1989b. URL <https://api.semanticscholar.org/CorpusID:7785881>.
- Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning, 2015.
- Namhoon Lee, Thalaiyasingam Ajanthan, Stephen Gould, and Philip H. S. Torr. A signal propagation perspective for pruning neural networks at initialization. *ArXiv*, abs/1906.06307, 2019. URL <https://api.semanticscholar.org/CorpusID:189898449>.
- Bowen Lei, Dongkuan Xu, Ruqi Zhang, Shuren He, and B. Mallick. Balance is essence: Accelerating sparse training via adaptive gradient correction. *ArXiv*, abs/2301.03573, 2023. doi: 10.48550/arXiv.2301.03573.

- Dawei Li, Tian Ding, and Ruoyu Sun. Over-parameterized deep neural networks have no strict local minima for any continuous activations. *CoRR*, abs/1812.11039, 2018a.
- Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *ArXiv*, abs/1608.08710, 2016. URL <https://api.semanticscholar.org/CorpusID:14089312>.
- Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018b.
- Yixuan Li, Jason Yosinski, Jeff Clune, Hod Lipson, and John Hopcroft. Convergent learning: Do different neural networks learn the same representations? In Dmitry Storcheus, Afshin Rostamizadeh, and Sanjiv Kumar, editors, *Proceedings of the 1st International Workshop on Feature Extraction: Modern Questions and Challenges at NIPS 2015*, volume 44 of *Proceedings of Machine Learning Research*, pages 196–212, Montreal, Canada, 11 Dec 2015. PMLR.
- Derek Lim, Moe Putterman, Robin Walters, Haggai Maron, and Stefanie Jegelka. The empirical impact of neural parameter symmetries, or lack thereof. *ArXiv*, abs/2405.20231, 2024. URL <https://api.semanticscholar.org/CorpusID:270123043>.
- James Lucas, Juhan Bae, Michael R. Zhang, Stanislav Fort, Richard S. Zemel, and Roger Baker Grosse. On monotonic linear interpolation of neural net-

work parameters. In *International Conference on Machine Learning*, 2021. URL <https://api.semanticscholar.org/CorpusID:228300974>.

Rongrong Ma and Lingfeng Niu. A survey of sparse-learning methods for deep neural networks. *2018 IEEE/WIC/ACM International Conference on Web Intelligence (WI)*, pages 647–650, 2018. URL <https://api.semanticscholar.org/CorpusID:58005508>.

Decebal Constantin Mocanu, Elena Mocanu, Peter Stone, Phuong H. Nguyen, Madeleine Gibescu, and Antonio Liotta. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. *Nature Communications*, 9(1), June 2018. ISSN 2041-1723. doi: 10.1038/s41467-018-04316-3. URL <http://dx.doi.org/10.1038/s41467-018-04316-3>.

Hesham Mostafa and Xin Wang. Parameter efficient training of deep convolutional neural networks by dynamic sparse reparameterization, 2019. URL <https://arxiv.org/abs/1902.05967>.

Mustafa Munir, William Avery, and Radu Marculescu. Mobilevig: Graph-based sparse attention for mobile vision applications, 2023. URL <https://arxiv.org/abs/2307.00395>.

Vaishnavh Nagarajan and J. Zico Kolter. Uniform convergence may be unable to explain generalization in deep learning. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

Behnam Neyshabur, Srinadh Bhojanapalli, David McAllester, and Nati Srebro. Exploring generalization in deep learning. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5947–5956, 2017.

Quynh Nguyen and Matthias Hein. The loss surface of deep and wide neural networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 2603–2612. PMLR, 2017.

Michela Paganini and Jessica Zosa Forde. Streamlining tensor and network pruning in pytorch. *CoRR*, abs/2004.13770, 2020.

Mansheej Paul, Feng Chen, Brett W. Larsen, Jonathan Frankle, Surya Ganguli, and Gintare Karolina Dziugaite. Unmasking the lottery ticket hypothesis: What’s encoded in a winning ticket’s mask? In *The Eleventh International Conference on Learning Representations*, 2023.

Fabrizio Pittorino, Antonio Ferraro, Gabriele Perugini, Christoph Feinauer, Carlo Baldassi, and Riccardo Zecchina. Deep networks on toroids: Removing symmetries reveals the structure of flat regions in the landscape geometry. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvári, Gang Niu, and Sivan Sabato, editors, *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume

162 of *Proceedings of Machine Learning Research*, pages 17759–17781. PMLR, 2022.

Alex Renda, Jonathan Frankle, and Michael Carbin. Comparing rewinding and fine-tuning in neural network pruning. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

Alexander Shapiro and Arkadi Nemirovski. On complexity of stochastic programming problems. *Continuous optimization: Current trends and modern applications*, pages 111–146, 2005.

Ekansh Sharma, Devin Kwok, Tom Denton, Daniel M. Roy, David Rolnick, and Gintare Karolina Dziugaite. Simultaneous linear connectivity of neural networks modulo permutation. In *Machine Learning and Knowledge Discovery in Databases. Research Track*, pages 262–279, Cham, 2024. Springer Nature Switzerland.

Jun Shi, Jianfeng Xu, Kazuyuki Tasaka, and Zhibo Chen. Sasl: Saliency-adaptive sparsity learning for neural network acceleration. *IEEE Transactions on Circuits and Systems for Video Technology*, 31(5):2008–2019, 2021. doi: 10.1109/TCSVT.2020.3013170.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR*

2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings, 2015.

Berfin Simsek, François Ged, Arthur Jacot, Francesco Spadaro, Clément Hongler, Wulfram Gerstner, and Johanni Brea. Geometry of the loss landscape in overparameterized neural networks: Symmetries and invariances. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 9722–9732. PMLR, 2021.

Sidak Pal Singh and Martin Jaggi. Model fusion via optimal transport. In Hugo Larochelle, Marc’Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

Hidenori Tanaka, Daniel Kunin, Daniel L. K. Yamins, and Surya Ganguli. Pruning neural networks without any data by iteratively conserving synaptic flow. *ArXiv*, abs/2006.05467, 2020. URL <https://api.semanticscholar.org/CorpusID:219558821>.

N. Joseph Tatro, Pin-Yu Chen, Payel Das, Igor Melnyk, Prasanna Sattigeri, and Rongjie Lai. Optimizing mode connectivity via neuron alignment. In Hugo Larochelle, Marc’Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

- Tiffany J. Vlaar and Jonathan Frankle. What can linear interpolation of neural network loss landscapes tell us? In *ICML*, pages 22325–22341, 2022.
- Chaoqi Wang, Chaoqi Wang, Guodong Zhang, and Roger Baker Grosse. Picking winning tickets before training by preserving gradient flow. *ArXiv*, abs/2002.07376, 2020. URL <https://api.semanticscholar.org/CorpusID:211146532>.
- Mitchell Wortsman, Maxwell Horton, Carlos Guestrin, Ali Farhadi, and Mohammad Rastegari. Learning neural network subspaces. *ArXiv*, abs/2102.10472, 2021. URL <https://api.semanticscholar.org/CorpusID:231985570>.
- Zhuliang Yao, Shijie Cao, Wencong Xiao, Chen Zhang, and Lanshun Nie. Balanced sparsity for efficient dnn inference on gpu. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):5676–5683, July 2019. ISSN 2159-5399. doi: 10.1609/aaai.v33i01.33015676. URL <http://dx.doi.org/10.1609/aaai.v33i01.33015676>.
- Chengxi Ye, Grace Chu, Yanfeng Liu, Yichi Zhang, Lukasz Lew, and Andrew Howard. Robust training of neural networks at arbitrary precision and sparsity. *ArXiv*, abs/2409.09245, 2024. URL <https://api.semanticscholar.org/CorpusID:272689153>.
- Binchi Zhang, Zaiyi Zheng, Zhengzhang Chen, and Jundong Li. Beyond the permutation symmetry of transformers: The role of rotation for model fusion, 2025. URL <https://arxiv.org/abs/2502.00264>.

Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

Bo Zhao, Iordan Ganev, Robin Walters, Rose Yu, and Nima Dehmamy. Symmetries, flat minima, and the conserved quantities of gradient flow. *ArXiv*, abs/2210.17216, 2022. URL <https://api.semanticscholar.org/CorpusID:253237810>.

Bo Zhao, Robert Mansel Gower, Robin Walters, and Rose Yu. Improving convergence and generalization using parameter symmetries. *ArXiv*, abs/2305.13404, 2023. URL <https://api.semanticscholar.org/CorpusID:258840961>.

Haizhong Zheng, Xiaoyan Bai, Xueshen Liu, Z. Morley Mao, Beidi Chen, Fan Lai, and Atul Prakash. Learn to be efficient: Build structured sparsity in large language models, 2024. URL <https://arxiv.org/abs/2402.06126>.

Appendix

Mathematical Proof: Permutation Symmetries

Consider a simple neural network with:

- Input layer with n_0 neurons.
- One hidden layer with n_1 neurons.
- Output layer with n_2 neurons.
- The weights connecting the input layer to the hidden layer are represented by a matrix $W^{(1)} \in \mathbb{R}^{n_1 \times n_0}$, and the biases by $b^{(1)} \in \mathbb{R}^{n_1}$.
- The weights connecting the hidden layer to the output layer are represented by a matrix $W^{(2)} \in \mathbb{R}^{n_2 \times n_1}$, and the biases by $b^{(2)} \in \mathbb{R}^{n_2}$.

Let σ be the activation function applied element-wise. For an input $x \in \mathbb{R}^{n_0}$:

- The activation of the hidden layer is

$$h = \sigma \left(W^{(1)} x + b^{(1)} \right) \in \mathbb{R}^{n_1}.$$

- The output of the network is

$$y = W^{(2)} h + b^{(2)} \in \mathbb{R}^{n_2}.$$

Let $L(\theta; (x, t))$ be the loss function for a single data point (x, t) , where:

1. x is the input.
2. t is the target output.
3. $\theta = (W^{(1)}, b^{(1)}, W^{(2)}, b^{(2)})$ are the network parameters.

Theorem I: If a neural network with at least one hidden layer of size $n_1 > 1$ has a global minimum θ^* for a non-zero loss function, L , then \exists another global minimum $\theta' \neq \theta^*$ obtained by permuting the neurons in the hidden layer.

Lemma I: Let a neural network with parameters θ and a hidden layer of size n_1 have a loss function $L(\theta, (x, t))$ for an input x and target t . Consider a permutation π of the n_1 neurons in the hidden layer, represented by a permutation matrix, $\mathbf{P} \in \mathcal{P}$ of size $n_1 \times n_1$, where \mathbf{P} is orthogonal such that $\mathbf{P}^\top = \mathbf{P}^{-1}$ and $\mathbf{P} \neq I$. Define a new parameter configuration θ_π obtained by applying this permutation to the hidden layer neurons and their corresponding weights as follows:

$$\theta_\pi = (W_\pi^{(1)}, b_\pi^{(1)}, W_\pi^{(2)}, b_\pi^{(2)}) = (\mathbf{P}W^{(1)}, \mathbf{P}b^{(1)}, W^{(2)}\mathbf{P}^\top, b^{(2)})$$

Then, for any input x , the output of the network with parameters θ is the same as the output with parameters θ_π , and consequently, the loss function value remains invariant under this permutation:

$$L(\theta, (x, t)) = L(\theta_\pi, (x, t)).$$

Proof of Lemma I:

To prove the lemma, it suffices to show that the output of the neural network remains invariant under the described permutation of the hidden layer neurons.

Consider the activation of the hidden layer with the permuted parameters θ_π :

$$h_\pi = \sigma(W_\pi^{(1)}x + b_\pi^{(1)})$$

Substituting the definitions of the permuted weights and biases, we get:

$$h_\pi = \sigma(\mathbf{P}W^{(1)}x + \mathbf{P}b^{(1)})$$

Factoring out the permutation matrix \mathbf{P} :

$$h_\pi = \sigma(\mathbf{P}(W^{(1)}x + b^{(1)}))$$

Let $z = W^{(1)}x + b^{(1)}$ be the pre-activation of the hidden layer with the original parameters. Then the activation with the original parameters is $h = \sigma(z)$. Thus, we can write:

$$h_\pi = \sigma(\mathbf{P}z)$$

Since the activation function σ is applied element-wise, and the permutation matrix \mathbf{P} rearranges the elements of the vector z , the application of σ to the permuted vector $\mathbf{P}z$ results in a vector whose elements are the same as the permuted elements of $\sigma(z)$.

Hence, this can be expressed as:

$$h_\pi = \mathbf{P}h$$

Now, consider the activation of the output layer with the permuted parameters θ_π :

$$y_\pi = W_\pi^{(2)}h_\pi + b_\pi^{(2)}$$

Substituting the definitions of the permuted weights, biases, and the permuted hidden layer activation:

$$y_\pi = (W^{(2)}\mathbf{P}^\top)(\mathbf{P}h) + b^{(2)}$$

Using the associativity of matrix multiplication:

$$y_\pi = W^{(2)}(\mathbf{P}^\top\mathbf{P})h + b^{(2)}$$

Since \mathbf{P} is a permutation matrix, it is orthogonal, meaning $\mathbf{P}^\top\mathbf{P} = I$, where I is the identity matrix. Therefore:

$$y_\pi = W^{(2)}Ih + b^{(2)}$$

$$y_\pi = W^{(2)}h + b^{(2)}$$

The right-hand side of this equation is the output of the network with the original parameters θ , which we denoted as y . Thus, we have shown that $y_\pi = y$. Since the output of the network remains the same under the permutation, the value of the loss function for a given input-target pair (x, t) will also be the same:

$$L(\theta, (x, t)) = L(\theta_\pi, (x, t))$$

This holds for all data points in the training dataset, and therefore, the overall loss function remains invariant under the permutation:

$$L(\theta) = L(\theta_\pi)$$

■

Proof of Theorem I:

Let $\theta^* = (W^{*(1)}, b^{*(1)}, W^{*(2)}, b^{*(2)})$ be a weight configuration that achieves a global minimum loss, $L_{min} = L(\theta^*)$. Consider a permutation matrix, \mathbf{P} , of size $n_1 \times n_1$, where $\mathbf{P} \neq I$. We define a new parameter configuration θ' as:

$$\theta' = (\mathbf{P}W^{*(1)}, \mathbf{P}b^{*(1)}, W^{*(2)}\mathbf{P}^\top, b^{*(2)}).$$

By Lemma I, θ' must also be a global minimum. It suffices to show that $\theta' \neq \theta^*$ in the general case.

Assume for the sake of contradiction, $\theta' = \theta^* \iff$

1. $\mathbf{P}W^{*(1)} = W^{*(1)}$
2. $\mathbf{P}b^{*(1)} = b^{*(1)}$
3. $W^{*(2)}\mathbf{P}^\top = W^{*(2)}$

However, these conditions imply that the weights and biases are invariant under the permutation, \mathbf{P} . But for a general weight configuration, this raises a contradiction as we chose \mathbf{P} to be a non-identity permutation matrix, hence $\theta' \neq \theta^*$. Therefore, if a global minimum exists, it is not unique due to permutation symmetries and hence, there exist multiple global minima.

■