Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Measurement and Information Systems

Symmetries and Invariances in Overparameterized Neural Networks

Scientific Students' Association Report

Author:

Márk Penc

Advisor:

Bálint Daróczy
HUN-REN Institute for Computer Science and Control

2024

Contents

Kivonat

Az utóbbi években a mode connectivity tanulmányozása egyre népszerűbbé vált. Nemcsak lehetővé teszi a neurális hálózatok hatékony kombinálását, hanem betekintést nyújt a hibafelületek geometriájába is, például a globális minimumok összekapcsoltságába. A mode connectivity egyik speciális fajtája a lineáris mode connectivity, ahol a modellparaméterek lineáris kombinációját vizsgáljuk. A legújabb empirikus bizonyítékok azt mutatják, hogy az erősen túlparaméterezett neurális hálózatokban a barrier, a veszteség maximális különbsége a lineáris úton a két paraméterezés között általában kicsi, különösen akkor, ha más, a gépi tanulási modellekben található szimmetriákat és invarianciákat kihasználjuk.

Az egyik ilyen invariancia a permutációs szimmetria, ahol minden olyan paraméterkonfigurációhoz, amely nem esik egy szimmetria altérbe, létezik $m_1!m_2!\dots m_{L-1}!$ másik modell, ahol m_i az i-edik rétegben lévő neuronok számát jelöli, amelyek ugyanazt a függvényt valósítják meg. A szimmetria okozta azonos konfigurációk számának faktoriális skálázódása miatt nehéz megtalálni egy másikra leginkább hasonlító modellt, azonban számos heurisztikus algoritmust fejlesztettek ki. Az egyik ilyen módszer az ún. weight matching, ahol a permutációs szimmetriák felhasználásával minimalizáljuk a két modell paraméter vektorainak L_2 távolságát.

Ebben a munkában empirikus bizonyítékokkal alátámasztott elméleti eredményeket mutatunk be arra vonatkozóan, hogy miért keletkeznek akadályok, és hogy a weight matching alkalmazásakor a túlparaméterezettség miért segíthet csökkenteni a két modell közötti akadályt. A neurális hálózatokban előforduló invarianciák, például a zéró típusú neuronok segítségével megmutatjuk, hogy egy modell további neuronokkal való bővítésével a korlátot csökkenteni és növelni is lehet. Azzal, hogy lehetséges magyarázatot adunk a lineáris mode connectivity sikerességére a túlparaméterezett hálózatokban, egy lépéssel közelebb kerülünk annak megértéséhez, hogy hogyan lehet hatékonyan és eredményesen kombinálni modelleket.

Acknowledgements

I would like to express my gratitude to my advisor Bálint Daróczy for his relentless support, guidance and expertise throughout this work.

I am extremely grateful to Beatrix Benkő for providing the research topic and mentoring me, in addition to their insights and discussed ideas.

I would further like to thank Dániel Rácz for his insights, ideas and feedback.

This research was supported by the European Union project RRF-2.3.1-21-2022-00004 within the framework of the Artificial Intelligence National Laboratory.

Abstract

In recent years, the study of mode connectivity has soared in popularity. Not only does it allow us to combine neural networks efficiently, but it also gives us an insight into the geometry of the loss landscape, such as the connectedness of the global minima. One specific kind of mode connectivity is linear mode connectivity, where the linear combination of the model parameters are studied. Recent empirical evidence has shown that in highly overparameterized neural networks the barrier, the maximal difference in loss on the linear path between the two parametrizations, is usually small, especially when other symmetries and invariances, found in machine learning models, are leveraged.

One such invariance is permutation symmetry, where for each parameter configuration, that doesn‘t fall on a symmetry subspace, there exists $m_1!m_2!\dots m_{L-1}!$ other models, where m_i denotes the number of neurons in the i-th layer, that implement the same function. Due to the factorial scaling of the number of symmetry induced identical configurations it is difficult to find a model most resembling another, however many heuristic algorithms have been developed. One such method is weight matching, where the L_2 norm between two models is minimized, using permutation symmetries.

In this work we will present theoretical results, backed by empirical evidence as to why barriers arise and how overparameterization can help decrease the barrier between two models when employing weight matching. Using invariances in neural networks, such as zero-type neurons, we will show that by extending a model, with additional neurons, it is possible to both decrease and increase the barrier. By giving a possible explanation to the success of linear mode connectivity in overparameterized networks, we take one step closer to understanding how to efficiently and effectively combine models.

Chapter 1

Introduction

In recent years, the number of parameters in neural networks increased dramatically. Models can reach millions, if not billions of parameters, as is the case for generative models, such as GPT-3, that has 175 billion parameters [4]. Due to this drastic increase in the amount of learnable parameters, the cost, time, and infrastructure required to train deep neural networks experienced a rapid growth. This brought with it an increased effort to understand the loss landscape of the models, and to improve knowledge transfer algorithms and methods.

One such method is mode connectivity [8, 5], which studies the connectedness of converged optimal models. Recently, many advancements have been made in the understanding of mode connectivity.

One line of work studies the mathematics of mode connectivity and the connectedness of the global minima, such as [12], where they showed, that for well-trained networks with generic properties, such as dropout stability and noise stability, the solutions are mode connected. In [19], Şimşek et al. showed that overparameterized neural networks have a connected global minima due to permutation symmetries and other invariances.

There are plenty of ways neural networks stay invariant to the adjustments of weights. For a neural network that produces a function, there are a lot of other solutions in parameter-space that produce an identical one. Studying these symmetries and invariances is useful, since they give us an insight into the loss landscape and explain many phenomena we can observe during training. For example, the fact that starting from a random initialization almost always leads to finding a solution without needing to travel far in parameter space.

Another line of work focuses more on empirical research. Main advancements here have been made in linear mode connectivity, where the linear combination of models is studied. It has been empirically shown, that as the width of models increases, the barrier, the maximal difference in loss on the linear path between the two parametrizations, decreases [6]. Furthermore it has been suggested, that most SGD solutions belong to the same basin, up to permutation [6]. Due to this, multiple heuristic algorithms have been developed to leverage permutation invariance in neural networks, in order to decrease the barrier between linearly combined parameters, such as weight matching, where the L_2 distance between models is heuristically minimized [1].

Mode connectivity and linear mode connectivity have been further studied in terms of generalization [7], adversarial robustness [18], and ensembling [8].

In chapter 2 we will summarize the current state of research and introduce our problem setup. In chapter 3 we will attempt to bridge the gap between mathematical studies and

empirical research and give a possible theory, as to why neural networks in the overparameterized setting have permutations which decrease the barrier drastically. In chapter 4 we will show empirical evidence backing our findings. In chapter 5 we will focus on summarizing our findings and on future work.

My main contribution in this work is mainly presented in chapter 3, where I show how barriers arise, how they can be eliminated and show a formulating theory as to why overparameterization helps reduce the barrier.

This work is based on my joint work with Bálint Daróczy, Dániel Rácz, Beatrix Benkő found on https://github.com/PencMark/Esann_2025.

Chapter 2

Background and Problem Setup

2.1 Notation

We will now introduce some of the common notations, used throughout this work. Note that most of these are from [19], but there are slight changes here and there.

For a positive integer number m, we will denote the set of $\{1, 2, \dots, m\}$ by $[m]$.

Let $\theta \in \Theta$ be the parameters of a model. When discussing different aspects of models we will add different indices to it. Such as θ_m, which, in the case of two layered fully connected neural networks, will denote the amount of neurons, m, in the hidden layer. Or θ^*, which will represent a model, which reached a global minimum.

Our neural network will be the function $f : \mathbb{R}^P \times \mathbb{R}^{d_{\text{in}}} \to \mathbb{R}^{d_{\text{out}}}$, where P is the number of trainable parameters in a model.

The notation m_l will represent the amount of neurons in the l-th layer. Where $m_0 = d_{\text{in}}$ and $m_L = d_{\text{out}}$.

The vector $\mathbf{w}_i^l \in \mathbb{R}^{m_{l-1}}$ will denote the i-th weight vector of the l-th layer in an MLP style architecture. When writing $w_{i,j}^l \in \mathbb{R}$ we will refer to the weight going from the j-th neuron in the $(l-1)$-th layer to the i-th neuron if the l-th layer. We will denote $\mathbf{a}_i^l = [w_{1,i}^{l+1}, w_{2,i}^{l+1}, \dots, w_{m_{l+1},i}^{l+1}]$ and call it outgoing weights. A neuron has incoming weights $\mathbf{w}_i^l$ and outgoing weights $\mathbf{a}_i^l$. It will sometimes be useful to talk about these two vectors together, so we will denote $\vartheta_i^l = [\mathbf{w}_i^l, \mathbf{a}_i^l]$. By default we will consider each neuron biasless, unless explicitly stated.

In this work we will be mainly focusing on fully connected neural networks. Where, given an activation function $\sigma(x)$, we have the neural network output be

$$f(\mathbf{x}, \theta) = W^L \vec{\sigma}\left(W^{L-1}\vec{\sigma}\left(\dots \vec{\sigma}\left(W^1\mathbf{x} + \mathbf{b}^1\right)\dots\right) + \mathbf{b}^{L-1}\right) + \mathbf{b}^L, \tag{2.1}$$

where $\vec{\sigma}(x_1, x_2, \dots, x_n) = [\sigma(x_1), \sigma(x_2), \dots, \sigma(x_n)]$ is the activation function applied to each coordinate of a vector, $\mathbf{b}^l \in \mathbb{R}^{m_{l-1}}$ is the bias, if exists, and $W^l \in \mathbb{R}^{m_l \times m_{l-1}}$ is a matrix, where all $\mathbf{w}_i^l$-s for a given layer l are stacked on top of each other, such that the i-th row of W^l is $\mathbf{w}_i^l$.

For the dataset we have $\mathcal{D} = \{(\mathbf{x}_i, y_i); i = \{1, \dots, N\}\}$, and the loss function is

$$\mathcal{L}_{\mathcal{D}}(\theta) = \frac{1}{N}\sum_{i=1}^{N} l(f(\mathbf{x}_i, \theta), y_i), \tag{2.2}$$

where $l(f(\mathbf{x}_i, \theta), y_i) = \frac{1}{2}(f(\mathbf{x}_i, \theta) - y_i)^2$.

For a given data generating function $f^*(\mathbf{x}, \theta^*)$, and input data distribution μ we will call the **true loss** of a given parameter, θ, the following:

$$\mathcal{L}_\mu(\theta) := \mathbb{E}_{\mathbf{x}\sim\mu}[l(f(\mathbf{x}, \theta), f^*(\mathbf{x}, \theta^*))] \tag{2.3}$$

2.2 Invariances in Neural Networks

There are many types of invariances in neural networks. In this section we gathered four types, relevant to studying the global minima in overparameterized neural networks. Ones that arise due to permutation symmetries, due to the choice of the activation function, due to overparameterization, and due to a sampling bias. There are some other, more architecture specific invariances, like how in neural networks with normalization layers, such as batch normalization, there is a scale-invariance [9, 13]. However, in this work we will not focus on these specific invariances and will instead address more general concepts.

2.2.1 Permutation Symmetry Induced Invariances

It has been long known, and discussed as far back as the 90s [17], that MLP-s have an invariance due to permutation. On the other hand, many aspects of permutation symmetries weren't explored to their fullest until recently. In this section we will show some of the interesting behaviour that emerges due to permutation invariance.

So what exactly is permutation invariance? Given a fully connected neural network we can exchange any two neurons, that are in the same layer, by swapping ϑ_i^l with ϑ_j^l, where $i, j \in [m_l]$ and $i \neq j$.

A visualization of this is shown in figure 2.1, where we can see a hidden layer with two neurons. Swapping the two neurons is equivalent to switching both the incoming and outgoing weights.

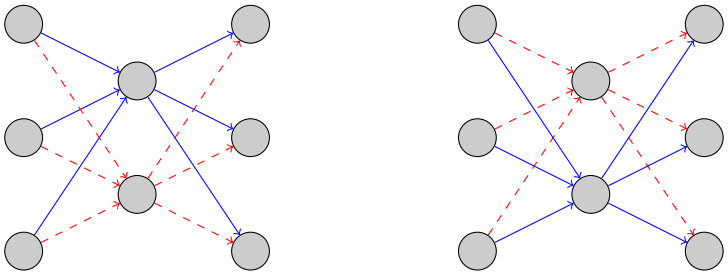


Figure 2.1: Two neural networks, that are permutations of each other. The weights colored blue represent ϑ_1^l, and colored in red ϑ_2^l. If we reorder the neurons, so that the top one is ϑ_2^l and the bottom one ϑ_1^l, we get the same function.

More formally, for each layer $l \in [L-1]$, there exists a symmetric group on the set of all possible permutations of $[m_l]$, denoted by S_{m_l}, under composition. For a given permutation $\pi_l \in S_{m_l}$, the map $\mathcal{P}_{\pi_l} : \mathbb{R}^{(m_{l+1}+m_{l-1})m_l} \to \mathbb{R}^{(m_{l+1}+m_{l-1})m_l}$ permutes the neurons of layer l in the following way:

$$(\vartheta_1^l, \vartheta_2^l, \ldots, \vartheta_{m_l}^l) \to (\vartheta_{\pi_l(1)}^l, \vartheta_{\pi_l(2)}^l, \ldots, \vartheta_{\pi_l(m_l)}^l). \tag{2.4}$$

This operation is equivalent to exchanging the rows of W_l and columns of W_{l+1}, according to the permutation. Since there is a permutation π_l for each layer $l \in [L-1]$, one can apply the permutation map $\mathcal{P}_{\pi_l}$ on each layer. From now on we will call $\pi = \{\pi_1, \pi_2, \ldots, \pi_{L-1}\}$ the permutation of a model and the map $\mathcal{P}_\pi : \mathbb{R}^P \to \mathbb{R}^P$ the permutation map of a model, which means applying $\mathcal{P}_{\pi_l}$ on all layers $l \in [L-1]$ according to π_l. In the rest of this work we will denote a model permutation as $\pi(\theta) := \mathcal{P}_\pi \theta$.

Note that the swapping of two neurons in a layer, doesn't change the function, that an MLP is producing. That is, for all $\theta \in \mathbb{R}^P$, $\mathbf{x} \in \mathbb{R}^{d_{\text{in}}}$, and for all π, $f(\mathbf{x}, \theta) = f(\mathbf{x}, \pi(\theta))$. In order to see why that is, let's focus on the l-th layer. And

$$\mathbf{z}^l := \vec{\sigma}\left(W^l \vec{\sigma}\left(W^{l-1} \vec{\sigma}\left(\ldots \vec{\sigma}\left(W^1 x\right)\ldots\right)\right)\right), \tag{2.5}$$

meaning the output of the neural network up to layer l. It is easy to see, that

$$\mathbf{z}^{l+1} = \vec{\sigma}(W^{l+1}\mathbf{z}^l) = \vec{\sigma}(\sum_{i=1}^{m_l} \mathbf{a}_i^l z_i^l). \tag{2.6}$$

Now let us consider permuting the neurons in layer l, according to π_l. Given we do this, the output of the l-th layer after permutation will be $z_i^{l'} = z_{\pi_l(i)}^l$, since we permute the incoming weights of these neurons. We further know, that $\mathbf{a}_i^{l'} = \mathbf{a}_{\pi_l(i)}^l$. This will mean, that the next layer will have an output of:

$$\mathbf{z}^{l+1} = \vec{\sigma}(\sum_{i=1}^{m_l} \mathbf{a}_{\pi_l(i)}^l z_{\pi_l(i)}^l), \tag{2.7}$$

which will be the same as before the neuron permutation, as we just sum in a different order.

The fact that there are permutation invariances in a neural network, means that each parameter in the parameter-space is has $m_1!m_2!\ldots m_{l-1}!$ number of permutation induced equivalent parameters. That is, for all inputs $\mathbf{x} \in \mathbb{R}^{d_{in}}$ they give the same output and for all $n \in \mathbb{N}$, $\frac{\partial^n \mathcal{L}_\mathcal{D}(\theta)}{\partial \theta^n}$ will be the same up to permutation. Which means that if we teach permuted models, all the first, second and n-th order optimizations will be the same, up to permutation, and find the same global minimum, up to permutation.

This permutation invariance also applies to a global minimum. Given there is a local or global minimum basin, there exists $m_1!m_2!\ldots m_{l-1}!$ other, connected, or unconnected global minima basins. This finding brought about a new wave of research which study how these global minima interact, and what this means for teaching neural networks. One line of research, which we will introduce in a later chapter is the conjecture that SGD solutions converge to the same loss basin, up to permutation. Due to the factorial nature of these symmetries finding an optimal permutation can be NP-hard, however many heuristic algorithms have been developed, such as those shown in [1].

In figure 2.2 we can see three different loss functions of simple neural networks. All of them implement the function $f(x, \theta^*) = \sigma(w_1 x) + \sigma(w_2 x)$, and have an optimal point at $w_1^* = 0.7, w_2^* = 0.1$. Notice how the loss functions are completely symmetric to w_1 and w_2 being swapped. We can further observe how each global minimum is part of a set of global minima that arises due to permutation invariance. Here we have two neurons, so we have two global minima due to permutations.

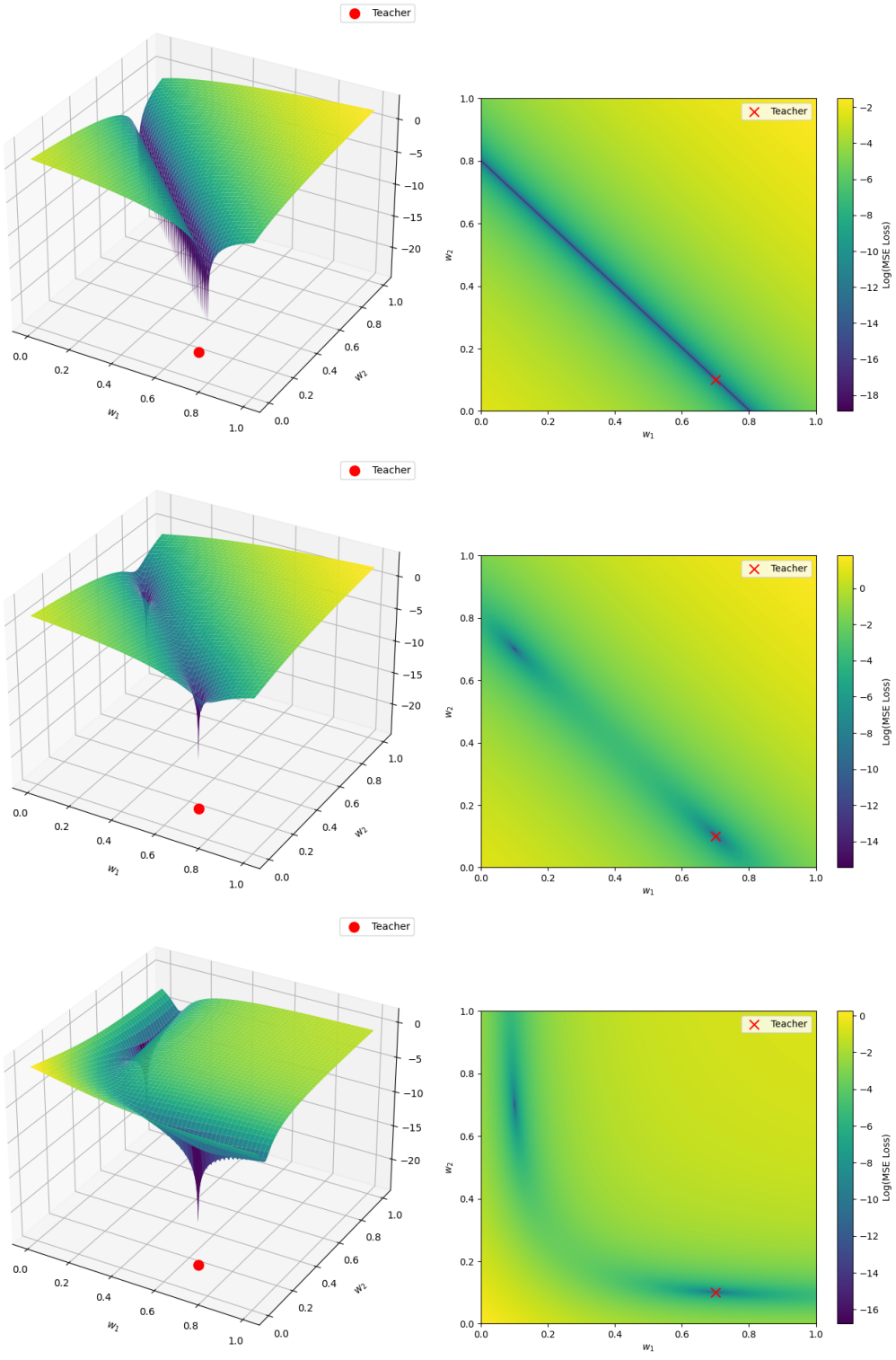


Figure 2.2: Logarithm of permutation symmetric mean square error loss functions of $w_1^* = 0.7, w_2^* = 0.1$ for $f(x, \theta^*) = \sigma(w_1 x) + \sigma(w_2 x)$. The loss functions from top to bottom are: $ReLU(x)$; $\sigma_{softplus}(x) + \sigma_{sigmoid}(x)$ and $\tanh(x)$. Notice how the loss function is symmetric on the axis $w_1 = w_2$, and how for every configuration of parameters there exists another, by switching w_1 and w_2, that has the same loss value.

2.2.2 Activation Function Induced Invariances

Another type of invariance is caused by the choice of the activation function $\sigma(x)$. There are a few invariances here as well, we will demonstrate two of them, one that arises from the parity of the function, and one that arises from the linearity of the activation function.

First we will focus on the parity. Let's consider the activation function

$$\sigma(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}. \tag{2.8}$$

The tanh function is an odd activation function. Meaning, that $-\sigma(-x) = \sigma(x)$. The output of the i-th neuron at layer l is $z_i^l = \sigma(\mathbf{w}_i^l \mathbf{z}^{l-1})$. If we take $\mathbf{w}_i^{l'} := -\mathbf{w}_i^l$, we will get $z_i^{l'} = \sigma(\mathbf{w}_i^{l'} \mathbf{z}^{l-1}) = -z_i^l$. To negate the minus sign, one needs to set the out weights to their negative as well, so $\mathbf{a}_i^{l'} := -\mathbf{a}_i^l$. With the change of these two vectors the function $f(\mathbf{x}, \theta)$ will be equal to $f(\mathbf{x}, \theta')$. This can be done to any neuron out of the $m_1 + m_2 + \ldots + m_{L-1}$ number of them, and brings in $2^{m_1+m_2+\ldots+m_{L-1}}$ number of functions which are equivalent to a given parameter. The same reasoning can be made for even functions, where $\sigma(x) = \sigma(-x)$, such as the rarely seen absolute value activation function [10]. Here only the change $\mathbf{w}_i^{l'} := -\mathbf{w}_i^l$ is to be made in order to obtain an equivalent model.

The parity induced invariances in the loss function can bee seen in figure 2.3. Here we can see a neural network implementing the function $f(x, \theta) = w_2 \sigma(w_1 x)$. We show on the loss function how multiple optimal solutions arise due to the activation function being odd or even.

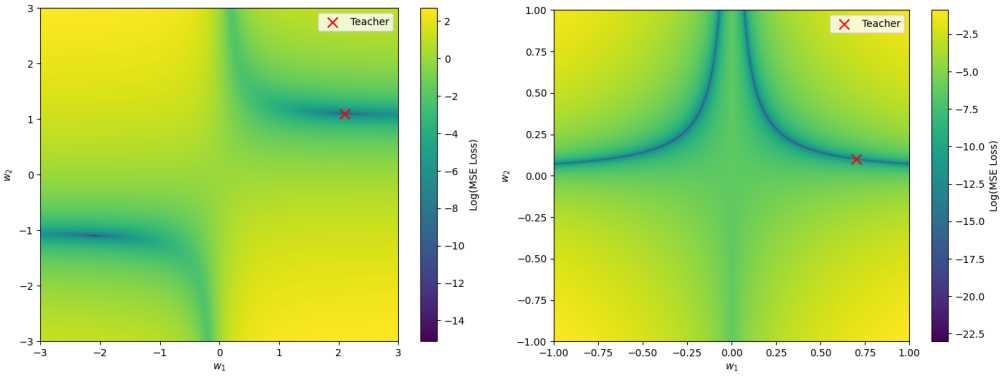


Figure 2.3: Logarithmic MSE loss of the odd function $f(x, \theta) = w_2 \tanh(w_1 x)$. Note, that there are only two global minimums, at $[2.1, 1.1]$ and $[-2.1, -1.1]$. *Right:* Logarithmic MSE loss of the even activation function $w_2|w_1 x|$, the parity induced invariance for the teacher, $[0.7, 0.1]$ is $[-0.7, 0.1]$. Note that here the whole valley achieves zero loss. This is because of the linear scale invariance, explained with ReLU.

Other types of functions, which are piece-wise linear, have a different kind of invariance. These types of function include the widely used $ReLU(x) = \max(x, 0)$ [15] and $LeakyReLU(x) = max(x, \epsilon x)$ [14] activation functions. These functions are scale invariant, meaning that for all $y \in \mathbb{R}^{++}$ the following holds:

$$ReLU(x) = \frac{1}{y} ReLU(yx). \tag{2.9}$$

This creates a whole valley of parameters, which are functionally equivalent to each other. We just need to change the weights of a given neuron to $\mathbf{w}_i^{l'} := y\mathbf{w}_i^l$ and $\mathbf{a}_i^{l'} := \frac{1}{y}\mathbf{a}_i^l$.

On figure 2.4 we can see how the function $\frac{1}{y}ReLU(yx)$ is invariant to $y \in \mathbb{R}^{++}$. We can also see how this creates a whole path of invariance, where the value of the loss function is zero.

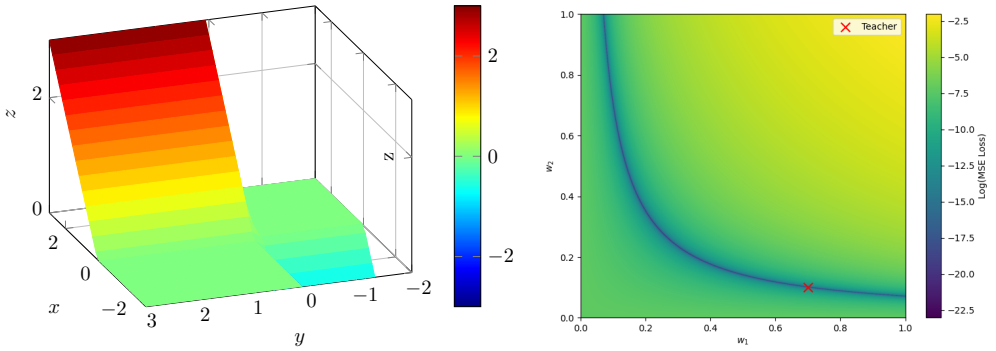


Figure 2.4: *Left:* Plot of the function $z = \frac{1}{y}\max(yx, 0)$, as can be seen, it is invariant along $y \in]0, \infty]$. *Right:* Logarithmic plot of the MSE loss of a ReLU network trained on a teacher network, with optimal weights $w_1^* = 0.7, w_2^* = 0.1$, of the function $f(x, \theta^*) = w_2^* ReLU(w_1^* x)$. Notice the valley achieving zero loss. While this looks similar to the permutation invariance seen in figure 2.2, this is a completely different type of invariance.

2.2.3 Overparameterization Induced Invariances

Here we will introduce the main idea presented in [19]. They focused on how overparameterized neural networks have a connected global minima, that comes from extending a smaller network with additional invariances. Let's consider 2 layered neural networks for this section, such that

$$f(\mathbf{x}, \theta) = \sum_{i=1}^{m} \mathbf{a}_i \sigma(\mathbf{w}_i \mathbf{x}), \tag{2.10}$$

where $m = m_1$ and $\mathbf{a}_i = \mathbf{a}_i^1$ and $\mathbf{w}_i = \mathbf{w}_i^1$. We say that a network has a minimal width of r^*, if there exists a parameter $\theta_{r^*}^*$, such that $\mathcal{L}_{\mathcal{D}}(\theta_{r^*}^*) = 0$, but for all $m < r^*$ all θ_m^*-s have $\mathcal{L}_{\mathcal{D}}(\theta_m^*) > 0$. We will call such a network, with minimal width of r^* an irreducible neural network [19].

Now let us consider expanding an irreducible neural network from width r^* into another with width $m > r^*$, as per definitions 3.2 and 3.3 in [19]. We need to have an expansion, such that $f(\mathbf{x}, \theta_m) = f(\mathbf{x}, \theta_{r^*}^*)$, for all $\mathbf{x} \in \mathbb{R}^{d_{\text{in}}}$. Given that we do not take the other invariances, presented in subsections 2.2.1 and 2.2.2, into consideration, we can have two types of expansions. One of them is splitting a neuron into k other neurons, the other is having b number of neurons cancel each other out.

For this example we will use a neuron in the first hidden layer, but this works for any hidden layer if you replace $\mathbf{x}$ with $\mathbf{z}^l$. Given a neuron, which contributes to the next layer $\mathbf{a}\sigma(\mathbf{w}\mathbf{x})$, we can split the neuron into k duplicates of the neuron, which when combined, create the same behaviour. In the same layer, create neurons $1, 2, \ldots, k$, such that for all $i \in [k]$, $\mathbf{w}_i = \mathbf{w}$ and $\sum_{i=1}^{k} \mathbf{a}_i = \mathbf{a}$. This way, when the neurons are summed up, they contribute to the next layer the same as the original, as seen below, in equation 2.11. In this the rest of this work we will call these type of neurons **duplicated neurons**,

and when there are k number of duplicates, we will specifically call them **k-duplicated neurons**.

$$\sum_{i=1}^{k} \mathbf{a}_i \sigma(\mathbf{w}_i \mathbf{x}) = \sum_{i=1}^{k} \mathbf{a}_i \sigma(\mathbf{w}\mathbf{x}) = (\sum_{i=1}^{k} \mathbf{a}_i)\sigma(\mathbf{w}\mathbf{x}) = \mathbf{a}\sigma(\mathbf{w}\mathbf{x}) \tag{2.11}$$

The other type of invariance, which extra neurons can introduce is when they cancel each other out. If we have $1, 2, \ldots, b$ neurons, such that all the incoming weights are the same, $\forall i, j \in [b]$: $\mathbf{w}_i = \mathbf{w}_j$, and the out weights sum up to zero, $\sum_{i=1}^{b} \mathbf{a}_i = \mathbf{0}$, then these neurons, when summed up will contribute exactly $\mathbf{0}$ to the next layer, as seen in equation 2.12. In the rest of the document we will refer to these neurons as **zero-type neurons**. Furthermore sometimes we will specifically call them **b-zero-type neurons**, if there are b neurons, that cancel each other out.

$$\sum_{i=1}^{b} \mathbf{a}_i \sigma(\mathbf{w}_i \mathbf{x}) = \sum_{i=1}^{b} \mathbf{a}_i \sigma(\mathbf{w}\mathbf{x}) = (\sum_{i=1}^{b} \mathbf{a}_i)\sigma(\mathbf{w}\mathbf{x}) = \mathbf{0}\sigma(\mathbf{w}\mathbf{x}) = \mathbf{0} \tag{2.12}$$

These two types of extensions can be taken into account regardless the choice of activation function. Note, that any model can be extended, even if it is not in the overparameterized setting. However it is particularly interesting to look at models, which have already achieved zero loss, and say something about their extensions, since we know that they too achieve zero loss.

This means that if we have a neural network, that can solve a problem with a width of r^* and has an optimal solution of $\theta^*_{r^*}$, we can have solutions in $m > r^*$, which also create the same function as $f(\mathbf{x}, \theta^*_{r^*})$. If we have, for example, $m = r^* + 1$, an optimal solution could be achieved by setting the first r^* neurons to the same incoming and outgoing weights and having the m-th neuron have any incoming weight, but an outgoing weight of zero, $\mathbf{a}_m = \mathbf{0}$. Leveraging the invariances presented above yields us a whole subspace of optimal solutions in the parameter-space.

In figure 2.5, we can see examples of zero loss subspaces. One that arises due to a duplicated neuron and one that is created by having a zero-type neuron. In both examples there is an optimal solution of $r^* = 1$, but our models have more parameters. However this is not an issue, as we can express the same function using the invariances presented. In both images the zero loss basins achieve the same function as the optimal model, $\theta^*_{r^*}$.

In Definition 3.2 of [19], they introduced the **affine subspace** $\Gamma_s(\theta^*_{r^*})$ of an irreducible point $\theta^*_{r^*}$. We too will define it below.

Definition 1. [19] Let $\theta^*_{r^*}$ be an optimal irreducible neural network. Given that we have m neurons, we can achieve the same function as $f(\theta^*_{r^*}, \mathbf{x})$, by having duplicated neurons and zero-type neurons. Let k_i be the number of times the i-th neuron was duplicated, and b_i represent a number of b_i-zero-type neurons. The affine subspace is determined by the number of duplicated neurons and zero-type neurons, which we will represent by $s = (k_1, \ldots, k_{r^*}, b_1, \ldots, b_j)$. This is an $(r^* + j)$ tuple of integers, where $k_i \geq 1$ and $b_i \geq 1$ and $\sum_{i=1}^{r^*} k_i + \sum_{i=1}^{j} b_i = m$. Furthermore we will represent the incoming weights of zero-type neurons by $\mathbf{w}'_i$ for $i \in [j]$ and the outgoing weights by α_i-s. The **affine subspace**

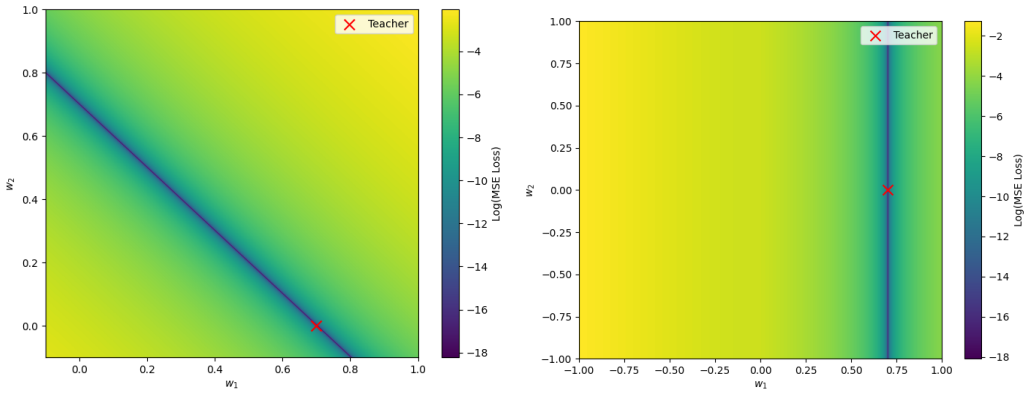


Figure 2.5: *Left:* Loss function evaluated on a database created from a teacher neural network of $f(x,\theta) = w_1 ReLU(0.7x)$. The plot is of the logarithm of the loss of a model $w_1 ReLU(0.7x) + w_2 ReLU(0.7x)$. *Right:* Loss function evaluated on a database from a teacher neuron of $ReLU(w_1 x)$. The plot is of the logarithm of the loss of model $ReLU(w_1 x) + 0ReLU(w_2 x)$.

$\Gamma_s(\theta^*_{r^*})$ is

$$\Gamma_s(\theta^*_{r^*}) := \{(\underbrace{\mathbf{w}_1,\dots,\mathbf{w}_1}_{k_1},\dots,\underbrace{\mathbf{w}_{r^*},\dots,\mathbf{w}_{r^*}}_{k_{r^*}},\underbrace{\mathbf{w}'_1,\dots,\mathbf{w}'_1}_{b_1},\dots,\underbrace{\mathbf{w}'_j,\dots,\mathbf{w}'_j}_{b_j},$$
$$\mathbf{a}^1_1,\dots,\mathbf{a}^{k_1}_1,\dots,\mathbf{a}^1_{r^*},\dots,\mathbf{a}^{k_{r^*}}_{r^*},\alpha^1_1,\dots,\alpha^{b_1}_1,\dots,\alpha^1_j,\dots,\alpha^{b_j}_j)$$
$$|\forall t \in [r^*] : \sum_{i=1}^{k_i} \mathbf{a}^i_t = \mathbf{a}_t \text{ and } \forall t \in [j] : \sum_{i=1}^{k_i} \alpha^i_t = \mathbf{0}\}. \tag{2.13}$$

The last part of the definition is just the equation, which clarifies, that these are indeed duplicated and zero-type neurons, grouped by t. The minimal loss paths seen in figure 2.5, are an example of these affine subspaces.

An example of a $\theta^*_{r^*}$ and all their affine subspaces, as defined in definition 1, is shown in figure 2.6.

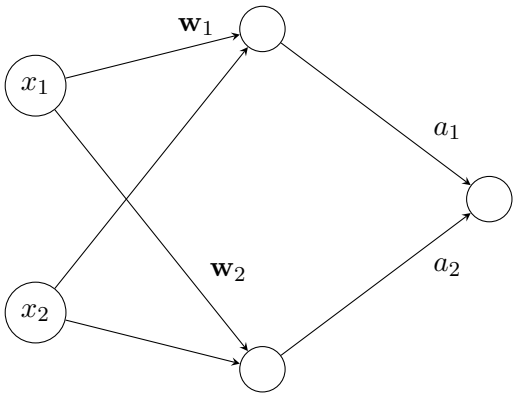
Furthermore, the network is invariant to any neuron permutation as per subsection 2.2.1, so we can take any permutation of the neurons in an affine subspace introduced in definition 1. This means that each zero-loss affine subspace is present $m!$ times in the parameter space. We will denote the permutations of a subspace as in [19], by

$$\mathcal{P}_\pi \Gamma_s(\theta^*_{r^*}) := \{\mathcal{P}_\pi \theta_m : \theta_m \in \Gamma_s(\theta^*_{r^*}), \pi \in S_m\}. \tag{2.14}$$

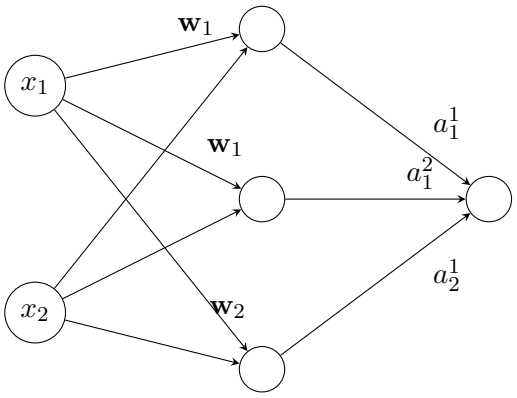
They further expanded upon this idea and looked at the union of these affine subspaces over all possible permutations, and all possible variations of duplicated and zero-type neurons. This union of affine subspaces gives us a set of global minima in the parameter-space, and allows us to talk about the elements of this set, which we know to be optimal parametrizations.

Definition 2. [19] The **expansion manifold** of an irreducible point $\theta^*_{r^*}$ is

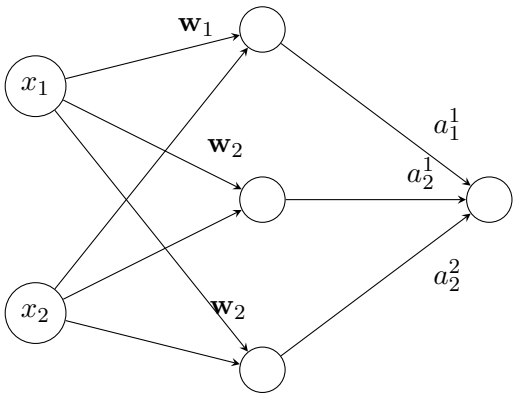
$$\Theta_{r^* \to m}(\theta^*_{r^*}) := \bigcup_{\substack{s=(k_1,\dots,k_{r^*},b_1,\dots,b_j) \\ \pi \in S_m}} \mathcal{P}_\pi \Gamma_s(\theta^*_{r^*}). \tag{2.15}$$



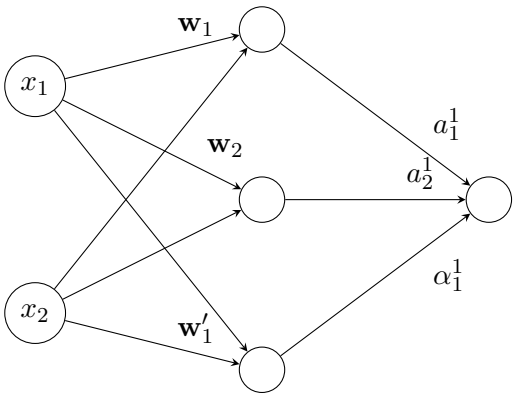
(a) The optimal solution.



(b) $s = (k_1 = 2, k_2 = 1)$, $a_1^1 + a_1^2 = a_1$, and $a_2^1 = a_2$



(c) $s = (k_1 = 1, k_2 = 2)$, $a_1^1 = a_1$, and $a_2^1 + a_2^2 = a_2$



(d) $s = (k_1 = 1, k_2 = 1, b_1 = 1)$, $a_1^1 = a_1$, $a_2^1 = a_2$, $\alpha_1^1 = 0$, and $\mathbf{w}_1' \in \mathbb{R}^2$

Figure 2.6: An optimal solution $\theta^*_{r^*}$, where $r^* = 2$ and their three possible affine subspaces, $\Gamma_s(\theta^*_{r^*})$, for $m = 3$.

Note, that the expansion manifold is not a manifold by strict mathematical terms, however it is used in the scientific literature by this name. We will follow these conventions in this work.

The expansion manifold is defined by overparameterization and permutation induced invariances. It is the union of all possible permutations and extra invariant neurons. One could also take into account activation function induced invariances into the definition. For example in the case of odd activations, like tanh, each weight and out weight could get a -1 or 1 assigned to them. In the case of scale invariant activation functions it is possible to extend the definition by assigning a scale factor to each in and out weight.

2.2.4 Sampling Induced Invariances

There is another type of invariance that can be found in neural networks, that can arise from a sampling bias in the input.

Let us consider a dataset like Mnist. Mnist consist of handwritten characters from zero to nine. The numbers are of a shade of white on a black background. Due to the nature of the dataset, and that of other similar ones, an invariance can arise. Let us consider the scenario, that a pixel in the top, left corner is always black, this can translate to $\forall \mathbf{x} \in \mathcal{D} : x_1 = 0$, for the first value of the input. In case there is a sampling bias in the dataset, like the one presented, each weight, $\mathbf{w}_i$, in the first layer, can have any value for $w_{i,1}$, since $w_{i,1}x_1 = w_{i,1}0 = 0$, for all $\mathbf{x} \in \mathcal{D}$.

This type of invariance can also arise in other hidden layers, if some neuron in layer l always has $z_i^l = 0$, for all $\mathbf{x} \in \mathcal{D}$. This is most likely to happen, when $\mathbf{w}_i = \mathbf{0}$. This, however is far less likely then the dataset having redundancy, unless regularization techniques are used. If we use a regularization technique such as L_1 or L_0 regularization, this is a more probable scenario. Although, if for all j, $w_{i,j} = 0$, we may as well just prune the neuron, as it has zero contribution to the network. We also know, that these neurons will not be present in an irreducible neural network, since with pruning we can get a smaller network, implementing the same function.

Furthermore, this type of invariance is not only limited to a given neuron or input feature being zero, but can also arise if there exists any linear correspondence between two neuron outputs or two input features, for all inputs. For example $x_2 = cx_1$ for some $c \in \mathbb{R}$ and all $\mathbf{x} \in \mathcal{D}$.

In figure 2.7 we will show how a sampling bias can create invariances in a neural network. We will show two cases, one where an x_i is 0 for all $\mathbf{x} \in \mathcal{D}$. The other case will be an example where there is a linear correlation between two inputs.

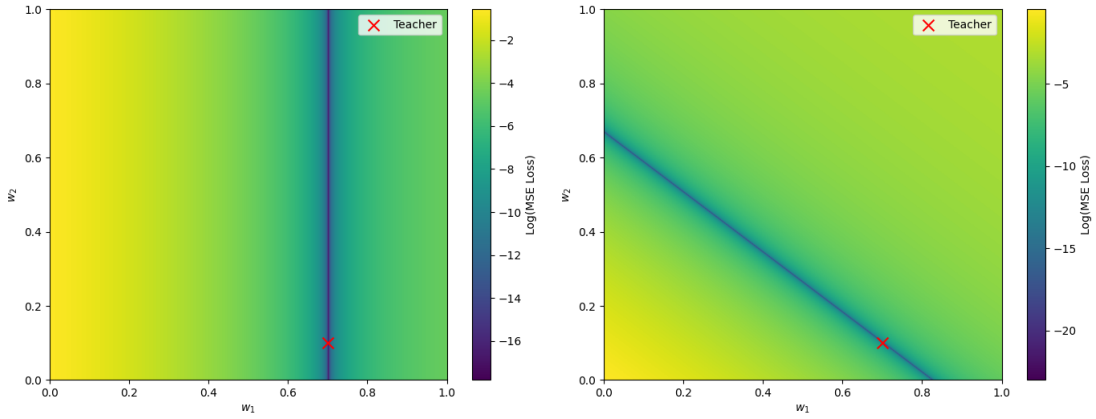


Figure 2.7: *Left:* The logarithmic loss of a neural network with the function of $f(\mathbf{x}, \theta) = \tanh(w_1x_1 + w_2x_2)$, where the input data for x_1 was generated using uniform distribution from $[-3, 3]$ and $x_2 = 0$. *Right:* The logarithmic loss of the same function but with the input data for x_2 generated as $x_2 = 1.23x_1$.

While this type of invariance can happen in neural networks we will not consider it in the rest of this work, as it is dataset specific.

2.3 Mode Connectivity

In this section we will introduce mode connectivity. Mode connectivity has been defined in many different ways over the years. We will give our own definitions here, which combines parts from [1], [3], and [12]. Furthermore we will write in two subsections regarding mode

connectivity, one in the classical sense and one about linear mode connectivity. In both sections we will talk about two sets of parameters in the parameter-space, which we will denote by ${}^{A}\theta$ and ${}^{B}\theta$.

2.3.1 Classical Mode Connectivity

Mode connectivity was first introduced in [8], they looked at if there exists a path, where models can be combined, such that along the path the increase in the loss is negligible or non-existent.

Definition 3. [12, 3] The **barrier** along a path $\gamma(\lambda) : \mathbb{R} \to \Theta$, where $\gamma(0) = {}^{B}\theta$ and $\gamma(1) = {}^{A}\theta$ is defined by

$$\mathcal{B}(\gamma(\lambda)) := \max_{\lambda \in [0,1]} \{\mathcal{L}_{\mathcal{D}}(\gamma(\lambda)) - ((1-\lambda)\mathcal{L}_{\mathcal{D}}(\gamma(0)) + \lambda \mathcal{L}_{\mathcal{D}}(\gamma(1))\}. \tag{2.16}$$

▪

Intuitively this is the loss of the models along the path, where the interpolation between the losses at the two endpoints is subtracted.

Definition 4. [12] We say that ${}^{A}\theta$ and ${}^{B}\theta$ are **mode connected** if there exists a path $\gamma(\lambda) : \mathbb{R} \to \Theta$, where $\gamma(0) = {}^{B}\theta$ and $\gamma(1) = {}^{A}\theta$, where $\mathcal{B}(\gamma(\lambda)) \leq 0$. ▪

As previously stated, there are many different kinds of definitions for mode connectivity, some do not subtract the interpolation of the endpoint losses, some subtract it with $\lambda = 0.5$, and there are even some which subtract the maximum of the endpoint losses. Regardless, the point of these are all the same, which is to understand if two models are connected along a path with small or no increase in the loss function. We can also define **ϵ-mode connectivity** [12], where we can say that two solution are ϵ-mode connected, if $\mathcal{B}(\gamma(\lambda)) \leq \epsilon$.

There have been many mathematical achievements in understanding mode connectivity between global minimum points. One such achievement is found in [12], where they showed that assuming generic properties, such as dropout stability or noise stability, well-trained networks are mode connected.

Another big achievement is found in [19], where they showed that the expansion manifold, $\Theta_{r^* \to m}(\theta^*_{r^*})$, of an irreducible point, $\theta^*_{r^*}$, is connected. This means that in overparameterized networks the global minima has a subset, where every two points are mode connected.

2.3.2 Linear Mode Connectivity

In contrast to classical mode connectivity, **linear mode connectivity** focuses on the special case, where

$$\gamma(\lambda) = \lambda {}^{A}\theta + (1-\lambda) {}^{B}\theta. \tag{2.17}$$

Linear mode connectivity is good for us in the sense, that if two models are, for a relatively small ϵ, **ϵ-linearly mode connected**, we can efficiently combine the two models. It could, for example, be used to combine large language models, which have a lot of parameters.

There have been empirical studies, which theorise, that up to neuron permutations, sufficiently overparameterized solutions are ϵ-linearly mode connected, such that ϵ is fairly small. One of these studies is by [1], where they developed multiple algorithms for permuting neurons in models, in order to reduce the barrier between their linear combination. One if these, introduced in their paper, in section 3.2, is weight matching.

In figure 2.8 we can see an example of the loss values along a linear path of two zero loss solutions. Note, that a barrier of 1.12 is considered fairly large.

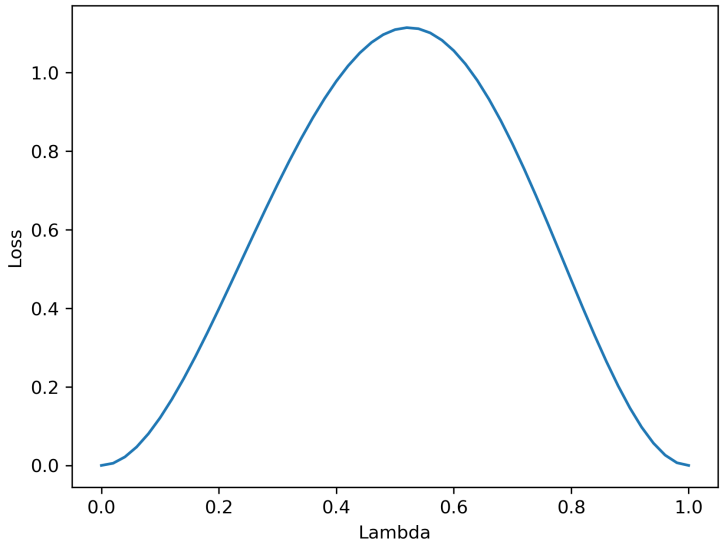


Figure 2.8: Loss values between the linear combination of two converged models with two layers and sixteen neurons in the hidden layer. The models were taught on a database created by a teacher network with four neurons. The barrier in this case is 1.12.

If two models are ϵ-linearly mode connected, it can mean, that the loss landscape between the two models is relatively flat. This can also give us an idea of how well a given solution generalizes. Generally speaking when training a neural network we are searching for solutions that are noise-invariant. This is usually the case when the basin we find is sufficiently large and flat, as opposed to a small, sharp basin, which usually does not generalize as well.

2.4 Weight Matching

Weight matching tries to minimize the L_2 distance between two models in parameter-space. Note, that, as stated in [1], this problem is NP-hard for $L > 2$. This is due to the fact that for each layer weight matching permutes both the rows in the current layer and the columns in the next layer, and the minimization should happen for all layers combined.

The algorithm of weight matching fixes all layers, but one. Then takes a step, where it minimizes the L_2 distance in that layer. according to a neuron permutation. Note that the minimization happens for the next layer as well, but for the outgoing weights of the model. It does this minimization algorithm for all other layers and then repeats the whole process, until convergence.

Note that we will mainly focus on $L = 2$, where the problem is not NP-hard. We will not go into further detail about the algorithm of weight matching, as it in itself is not too important to this thesis.

As for results, there have been many in recent years, where weight matching reduced or even eliminated the barrier between models. It has been theorized that it is due to the decrease in L_2 distance. However in many occurrences the decrease in L_2 distance does

not sufficiently explain the phenomenon. To our knowledge the reduction of the barrier in weight matching has not been fully understood as of yet.

In figure 2.9 we show how the barrier can decrease after weight matching. The models have the same parameters as in figure 2.8, the only difference is that one of the models was permuted to the other, according to the weight matching algorithm. We can see a drastic decrease in the barrier.

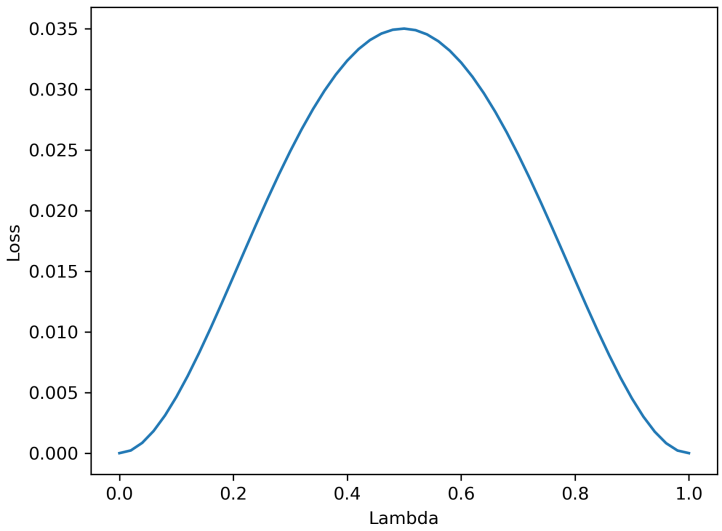


Figure 2.9: After weight matching the two models from figure 2.8 become ϵ-mode connected, for $\epsilon = 0.035$. The parameters we get from their linear combination are quite good at fitting the data. The optimal permutation, minimizing the L_2 distance is ${}^{B}\pi = [2, 12, 4, 11, 15, 6, 3, 8, 13, 16, 10, 5, 14, 1, 7, 9]$.

Chapter 3

Permutation in the Overparameterized Setting

In this section we will attempt to give an explanation as to why linear mode connectivity works in the overparameterized regime. We will mainly work in the setting introduced in [19]. The possible explanation will consist of four parts, and will be backed by empirical evidence. At some places, where indicated, we will need further rigorous proofs in future work. From here on, we will only consider two layered neural networks.

3.1 Uniqueness of the Irreducible Solution

In [19], they assumed that there exists a $\theta^*_{r^*}$, that is unique up to permutation. In this section we will familiarize the reader with three papers, which studied the necessary conditions, for irreducible neural networks to be unique up to invariances.

In [17], Sussmann showed that for a two layer irreducible neural network with tanh activation function, the optimal parameters are uniquely determined up to a finite group of invariances. More specifically those introduced in subsections 2.2.1 and 2.2.2. The main argument behind both this and the next paper is that under the condition of irreducibility, and activation function invariances, the functions, which the neurons produce, are linearly independent of each other.

In [2], they tried a more general approach. They looked at odd functions in both biasless, and biased neural networks, and gathered conditions under which the activation function has an *independence property* or a *weak independence property*. They proved, that if an odd activation function satisfies the independence property, then $\theta^*_{r^*}$ is uniquely determined, up to invariances. In case of biasless neural networks it was enough for activation functions to satisy the weak independence property, set by the authors.

In [16], they explored the uniqueness of irreducible networks with *ReLU* activation functions. They discovered, that in these types of two layered neural networks there is another type of non-trivial invariance, that is different from scale invariance, and permutation invariance. They further proved an irreducible solution is unique, up to these three invariances.

For reasons explained in the next section, we will focus on the activation function $\sigma(x) = \sigma_{soft}(x) + \sigma_{sigm}(x)$, where $\sigma_{soft}(x) = \ln(1 + e^x)$ is the softplus and $\sigma_{sigm}(x) = \frac{1}{1+e^{-x}}$ is the sigmoid activation function. In all of our empirical experiments $\theta^*_{r^*}$ was uniquely determined with this loss function, up to permutation, however in future work it will be

important to prove if a similar argument can be made about this activation function, to the odd functions, tanh and $ReLU$. Note, that our explanation does not necessarily rely on the choice of activation function, however for each function, different invariances would need to be considered. For now, we will assume that $\theta^*_{r^*}$ is unique up to invariances, as assumed in [19].

3.2 Convergence Into the Expansion Manifold

Next we will discuss if, in an overparameterized neural network, convergence into the expansion manifold is guaranteed. Meaning, that $\forall \theta^*_m : \theta^*_m \in \Theta_{r^* \to m}(\theta^*_{r^*})$.

In [19] theorem 4.2. they proved that given we have an, up to permutation, unique $\theta^*_{r^*}$, and an activation function σ, that is C^∞ and $\sigma(0) \neq 0$, and for infinitely many odd and even n, the n-th derivative of the loss function, $\sigma^{(n)}(0) \neq 0$. For $m > r^*$, all $\theta^*_m \in \Theta_{r^* \to m}(\theta^*_{r^*})$, where $\mathcal{L}_\mu(\theta^*_m) = 0$. Meaning that convergence into the expansion manifold is guaranteed, if the model achieves zero true loss.

The activation function $\sigma(x) = \sigma_{soft}(x) + \sigma_{sigm}(x)$ (figure 3.1) satisfies the criteria set up in the previous paragraph [19]. Thus we know that if we have an overparameterized neural network, our model will converge into one of the affine subspaces $\mathcal{P}_\pi \Gamma_s(\theta^*_{r^*})$.

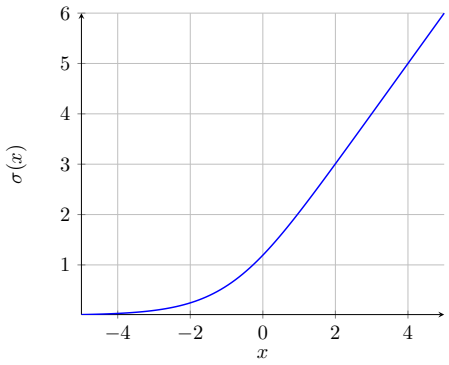


Figure 3.1: Plot of the activation function $\sigma(x) = \sigma_{soft}(x) + \sigma_{sigm}(x)$.

The authors of [19] chose this activation function because it guarantees that there aren't any activation function induced invariances. In remark B.3. they mentioned that for the tanh activation function the theorem holds, as long as an extended expansion manifold is considered, where each in and out weight pair may get a positive or negative sign.

3.3 Linear Combination of Converged Models

Now that we have the necessary background and problem setup, we will delve into the linear combination of overparameterized neural networks, and show how the error arises between converged models.

Theorem 1 (Models in the Same Affine Subspace are Linearly Mode Connected). Let us consider two models from the same affine subspace, ${}^A\theta^*_m, {}^B\theta^*_m \in \Gamma_s(\theta^*_{r^*})$, determined by $s = (k_1, \ldots, k_{r^*}, b_1, \ldots, b_j)$, where $k_i \geq 1$ and $b_i \geq 1$, and $\sum_{i=1}^{r^*} k_i + \sum_{i=1}^{j} b_i = m$. Let the linear combination of the models be ${}^\lambda\theta_m := \lambda {}^A\theta^*_m + (1-\lambda) {}^B\theta^*_m$. With these conditions ${}^\lambda\theta_m \in \Gamma_s(\theta^*_{r^*})$, for all λ and thus also achieves zero loss. ∎

Proof. (Theorem 1)

We will prove the theorem in multiple parts. First of all, we will see, that if there are k_i-duplicated neurons in both model A and B, which are copies of the incoming weight, $\mathbf{w}_i$, $i \in [r^*]$ of a neuron from $\theta^*_{r^*}$, and $\sum_{j=1}^{k_i} {}^A\mathbf{a}_i^j \sigma(\mathbf{w}_i\mathbf{x}) = \mathbf{a}_i\sigma(\mathbf{w}_i\mathbf{x}) = \sum_{j=1}^{k_i} {}^B\mathbf{a}_i^j \sigma(\mathbf{w}_i\mathbf{x})$, then their linear combination is also equal to $\mathbf{a}_i\sigma(\mathbf{w}_i\mathbf{x})$. The linear combination of the two sub-models is:

$$\sum_{j=1}^{k_i}(\lambda {}^A\mathbf{a}_i^j + (1-\lambda) {}^B\mathbf{a}_i^j)\sigma((\lambda\mathbf{w}_i + (1-\lambda)\mathbf{w}_i)\mathbf{x}) \tag{3.1}$$

We can eliminate the λ from inside the activation function, since in both sub-models have the same $\mathbf{w}_i$ neuron duplicated k_i times.

$$\sum_{j=1}^{k_i}(\lambda {}^A\mathbf{a}_i^j + (1-\lambda) {}^B\mathbf{a}_i^j)\sigma((\lambda\mathbf{w}_i + (1-\lambda)\mathbf{w}_i)\mathbf{x}) = \sum_{j=1}^{k_i}(\lambda {}^A\mathbf{a}_i^j + (1-\lambda) {}^B\mathbf{a}_i^j)\sigma(\mathbf{w}_i\mathbf{x}) \tag{3.2}$$

Now it is possible to take $\sigma(\mathbf{w}_i\mathbf{x})$ out of the sum.

$$\sum_{j=1}^{k_i}(\lambda {}^A\mathbf{a}_i^j + (1-\lambda) {}^B\mathbf{a}_i^j)\sigma(\mathbf{w}_i\mathbf{x}) = (\sum_{j=1}^{k_i}(\lambda {}^A\mathbf{a}_i^j + (1-\lambda) {}^B\mathbf{a}_i^j))\sigma(\mathbf{w}_i\mathbf{x}) \tag{3.3}$$

Next we can rewrite the sum for ${}^A\mathbf{a}_i^j$ and ${}^B\mathbf{a}_i^j$ separately, and take λ and $(1-\lambda)$ out of the sum:

$$(\sum_{j=1}^{k_i}(\lambda {}^A\mathbf{a}_i^j + (1-\lambda) {}^B\mathbf{a}_i^j))\sigma(\mathbf{w}_i\mathbf{x}) = (\lambda\sum_{j=1}^{k_i} {}^A\mathbf{a}_i^j + (1-\lambda)\sum_{j=1}^{k_i} {}^B\mathbf{a}_i^j)\sigma(\mathbf{w}_i\mathbf{x}) \tag{3.4}$$

We can now replace $\sum_{j=1}^{k_i} {}^A\mathbf{a}_i^j$ with $\mathbf{a}_i$, and replace $\sum_{j=1}^{k_i} {}^B\mathbf{a}_i^j$ with $\mathbf{a}_i$ as well, since we know these are k_i-duplicated neurons.

$$(\lambda\sum_{j=1}^{k_i} {}^A\mathbf{a}_i^j + (1-\lambda)\sum_{j=1}^{k_i} {}^B\mathbf{a}_i^j)\sigma(\mathbf{w}_i\mathbf{x}) = (\lambda\mathbf{a}_i + (1-\lambda)\mathbf{a}_i)\sigma(\mathbf{w}_i\mathbf{x}) = \mathbf{a}_i\sigma(\mathbf{w}_i\mathbf{x}) \tag{3.5}$$

Next we will prove that the linear combination between two b-zero-type neurons is itself a b-zero-type neuron. We have $\sum_{j=1}^{b_i} {}^A\alpha_i^j \sigma({}^A\mathbf{w}_i'\mathbf{x}) = \mathbf{0} = \sum_{j=1}^{b_i} {}^B\alpha_i^j \sigma({}^B\mathbf{w}_i'\mathbf{x})$. The linear combination of the two sub-models is:

$$\sum_{j=1}^{b_i}(\lambda {}^A\alpha_i^j + (1-\lambda) {}^B\alpha_i^j)\sigma((\lambda {}^A\mathbf{w}_i' + (1-\lambda) {}^B\mathbf{w}_i')\mathbf{x}) \tag{3.6}$$

Since the part inside the activation function does not depend on j, we can take the multiplication out of the summation. Furthermore we will denote ${}^\lambda\mathbf{w}_i' := (\lambda {}^A\mathbf{w}_i' + (1-\lambda) {}^B\mathbf{w}_i')$.

$$\sum_{j=1}^{b_i}(\lambda {}^A\alpha_i^j + (1-\lambda){}^B\alpha_i^j)\sigma({}^\lambda\mathbf{w}_i'\mathbf{x}) = (\sum_{j=1}^{b_i}(\lambda {}^A\alpha_i^j + (1-\lambda){}^B\alpha_i^j))\sigma({}^\lambda\mathbf{w}_i'\mathbf{x}) \tag{3.7}$$

We can do something similar as to before, take the summation apart in the equation and take the λ and $(1-\lambda)$ out of the summation.

$$(\sum_{j=1}^{b_i}(\lambda {}^A\alpha_i^j + (1-\lambda){}^B\alpha_i^j))\sigma({}^\lambda\mathbf{w}_i'\mathbf{x}) = (\lambda\sum_{j=1}^{b_i}{}^A\alpha_i^j + (1-\lambda)\sum_{j=1}^{b_i}{}^B\alpha_i^j)\sigma({}^\lambda\mathbf{w}_i'\mathbf{x}) \tag{3.8}$$

We further know, that these are b_i-zero-type neurons. Thus $\sum_{j=1}^{b_i}{}^A\alpha_i^j$ is the same as $\mathbf{0}$, and $\sum_{j=1}^{b_i}{}^B\alpha_i^j$ is also equal to $\mathbf{0}$.

$$(\lambda\sum_{j=1}^{b_i}{}^A\alpha_i^j + (1-\lambda)\sum_{j=1}^{b_i}{}^B\alpha_i^j)\sigma({}^\lambda\mathbf{w}_i'\mathbf{x}) = \mathbf{0}\sigma({}^\lambda\mathbf{w}_i'\mathbf{x}) = \mathbf{0} \tag{3.9}$$

We are now ready to prove theorem 1. We know, that the linear combination of b_i-zero-type neurons, when matched with other b_i-zero-type neurons is also a b_i-zero-type neuron. We also know that the linear combination of k_i-duplicated neurons of $\mathbf{w}_i$ remain k_i-duplicated neurons of $\mathbf{w}_i$, when matched with k_i-duplicated neurons of $\mathbf{w}_i$ from the other model. Since both models come from the same affine subspace each k_i-duplicated neuron of $\mathbf{w}_i$ will be interpolated with a k_i-duplicated neuron of $\mathbf{w}_i$ and b_i-zero-type with a b_i-zero-type neuron, and with this we have proven the theorem.

□

Theorem 2 (Linear Mode Connectivity After Permutation). Let us consider two models, which have the same number of k_i-duplicated neurons and b_i-duplicated neurons, but not in the order defined by $s = (k_1, \ldots, k_{r^*}, b_1, \ldots, b_j)$. More specifically let them be in two different permutations, ${}^A\pi$ and ${}^B\pi$, of the same affine subspace. This way ${}^A\theta_m^* \in \mathcal{P}_{({}^A\pi)}\Gamma_s(\theta_{r^*}^*)$, and ${}^B\theta_m^* \in \mathcal{P}_{({}^B\pi)}\Gamma_s(\theta_{r^*}^*)$. There exists an optimal permutation $\pi^* = {}^A\pi({}^B\pi^{-1})$, where π^{-1} is the inverse permutation, so that their linear combination ${}^\lambda\theta_m := \lambda {}^A\theta_m^* + (1-\lambda)\pi^*({}^B\theta_m^*)$ achieves zero loss, and ${}^\lambda\theta_m \in \mathcal{P}_{({}^A\pi)}\Gamma_s(\theta_{r^*}^*)$ for all $\lambda \in [0,1]$.

Proof. (Theorem 2)

With the help of theorem 1 proving this will not be difficult. Let us permute the neurons of ${}^A\theta_m^*$ according to ${}^A\pi^{-1}$, and also permute ${}^B\theta_m^*$ according to ${}^B\pi^{-1}$. We know that this way

$${}^A\pi^{-1}({}^A\theta_m^*), {}^B\pi^{-1}({}^B\theta_m^*) \in \Gamma_s(\theta_{r^*}^*). \tag{3.10}$$

According to theorem 1, their linear combination in this state is also in the affine subspace $\Gamma_s(\theta_{r^*}^*)$, that is for all λ,

$${}^A\pi^{-1}({}^\lambda\theta_m) = \lambda {}^A\pi^{-1}({}^A\theta_m^*) + (1-\lambda){}^B\pi^{-1}({}^B\theta_m^*) \in \Gamma_s(\theta_{r^*}^*). \tag{3.11}$$

Let us now take the permutation ${}^A\pi$ of everything. This will yield us, that for all λ,

$${}^\lambda\theta_m = \lambda {}^A\theta_m^* + (1-\lambda){}^A\pi({}^B\pi^{-1}({}^B\theta_m^*)) \in \mathcal{P}_{({}^A\pi)}\Gamma_s(\theta_{r^*}^*). \tag{3.12}$$

With this we have proven, as long as the models are in the same affine subspace, up to permutation, there exists an optimal permutation $\pi^* = {}^A\pi({}^B\pi^{-1})$, where after applying π^* to model B, it becomes linearly mode connected with model A.

□

Theorem 3 (The Error, if Models in Different Subspaces Are Combined). Let us now consider models in different subspaces. Let us have two models that have the same number of k_i-duplicated neurons and b_i-zero-type neurons, except for k_1 and k_2, where there is one more k_1-duplicated neuron in model A, and one more k_2-duplicated neuron in model B. More specifically, let $s = (k_1, k_2, \ldots, k_{r^*}, b_1, \ldots, b_j)$, where $k_i \geq 1$, $k_1 \geq 2$, and $b_i \geq 1$ and let ${}^Bs = ({}^Bk_1, {}^Bk_2, k_3 \ldots, k_{r^*}, b_1, \ldots, b_j)$, where ${}^Bk_1 = k_1 - 1$ and ${}^Bk_2 = k_2 + 1$. Our two optimal models are, ${}^A\theta^*_m \in \Gamma_s(\theta^*_{r^*})$ and ${}^B\theta^*_m \in \Gamma_{{}^Bs}(\theta^*_{r^*})$. Let their linear combination be ${}^\lambda\theta_m := \lambda {}^A\theta^*_m + (1-\lambda){}^B\theta^*_m$.

The error between the optimal function and the linear interpolation, $f(\mathbf{x}, \theta^*) - f(\mathbf{x}, {}^\lambda\theta_m)$, is:

$$\lambda {}^A\mathbf{a}_1^{k_1}(\sigma(\mathbf{w}_1\mathbf{x}) - \sigma((\lambda\mathbf{w}_1 + (1-\lambda)\mathbf{w}_2)\mathbf{x})) + (1-\lambda){}^B\mathbf{a}_2^1(\sigma(\mathbf{w}_2\mathbf{x}) - \sigma((\lambda\mathbf{w}_1 + (1-\lambda)\mathbf{w}_2)\mathbf{x})) \tag{3.13}$$

▪

Proof. (Theorem 3)

Next, we will prove this theorem. Let us first consider the different ways $f(\mathbf{x}, \theta^*)$ can be expressed.

$$f(\mathbf{x}, \theta^*) = \sum_{i=1}^{r^*} \mathbf{a}_i \sigma(\mathbf{w}_i, \mathbf{x}) = \sum_{t=1}^{r^*} \sum_{i=1}^{k_t} \mathbf{a}_t^i \sigma(\mathbf{w}_t\mathbf{x}) + \sum_{t=1}^{j} \sum_{i=1}^{b_t} \alpha_t^i \sigma(\mathbf{w}'_t\mathbf{x}) \tag{3.14}$$

For simplicity, let us introduce $SubNetwork_{p\to q}$ as the part of the neural network from neuron p to neuron q. We further know, that equation 3.14 is true for both s and ${}^Bs$. For ${}^A\theta^*_m$ it can be expressed like so:

$${}^ASubNetwork_{(k_1+k_2)\to m} = \sum_{t=3}^{r^*} \sum_{i=1}^{k_t} {}^A\mathbf{a}_t^i \sigma(\mathbf{w}_t\mathbf{x}) + \sum_{t=1}^{j} \sum_{i=1}^{b_t} {}^A\alpha_t^i \sigma({}^A\mathbf{w}'_t\mathbf{x}) \tag{3.15}$$

$$f(\mathbf{x}, {}^A\theta^*_m) = \sum_{i=1}^{k_1} {}^A\mathbf{a}_1^i \sigma(\mathbf{w}_1\mathbf{x}) + \sum_{i=1}^{k_2} {}^A\mathbf{a}_2^i \sigma(\mathbf{w}_2\mathbf{x}) + {}^ASubNetwork_{(k_1+k_2)\to m} \tag{3.16}$$

And for ${}^B\theta^*_m$ like so:

$${}^BSubNetwork_{(k_1+k_2)\to m} = \sum_{t=3}^{r^*} \sum_{i=1}^{k_t} {}^B\mathbf{a}_t^i \sigma(\mathbf{w}_t\mathbf{x}) + \sum_{t=1}^{j} \sum_{i=1}^{b_t} {}^B\alpha_t^i \sigma({}^B\mathbf{w}'_t\mathbf{x}) \tag{3.17}$$

$$f(\mathbf{x}, {}^B\theta^*_m) = \sum_{i=1}^{{}^Bk_1} {}^B\mathbf{a}_1^i \sigma(\mathbf{w}_1\mathbf{x}) + \sum_{i=1}^{{}^Bk_2} {}^B\mathbf{a}_2^i \sigma(\mathbf{w}_2\mathbf{x}) + {}^BSubNetwork_{(k_1+k_2)\to m} \tag{3.18}$$

Next, we can express the linear combination of these two functions.

$$f(\mathbf{x}, \theta^*) = \lambda f(\mathbf{x}, {}^A\theta_m^*) + (1-\lambda) f(\mathbf{x}, {}^B\theta_m^*) \tag{3.19}$$

As for the linear combination of the model parameters, the function, that they create, can be expressed as:

$$\begin{aligned} f(\mathbf{x}, {}^\lambda\theta_m) = &\sum_{i=1}^{k_1-1} (\lambda^A \mathbf{a}_1^i + (1-\lambda)^B \mathbf{a}_1^i)\sigma(\mathbf{w}_1\mathbf{x}) + \\ &+ (\lambda^A \mathbf{a}_1^{k_1} + (1-\lambda)^B \mathbf{a}_2^1)\sigma((\lambda\mathbf{w}_1 + (1-\lambda)\mathbf{w}_2)\mathbf{x}) + \\ &+ \sum_{i=1}^{k_2} (\lambda^A \mathbf{a}_2^i + (1-\lambda)^B \mathbf{a}_2^{i+1})\sigma(\mathbf{w}_2\mathbf{x}) + \\ &+ \sum_{t=3}^{r^*} \sum_{i=1}^{k_t} (\lambda^A \mathbf{a}_t^i + (1-\lambda)^B \mathbf{a}_t^i)\sigma(\mathbf{w}_t\mathbf{x}) + \\ &+ \sum_{t=1}^{j} \sum_{i=1}^{b_t} (\lambda^A \alpha_t^i + (1-\lambda)^B \alpha_t^i)\sigma((\lambda^A \mathbf{w}_t' + (1-\lambda)^B \mathbf{w}_t')\mathbf{x}) \end{aligned} \tag{3.20}$$

For the previous theorem we proved, that the linear combination of b-zero-type neurons is itself an b-zero-type neuron. So we know, that the last line in equation 3.20 is zero. We further know, that this is true for $\lambda f(\mathbf{x}, {}^A\theta^*)$ and $(1-\lambda) f(\mathbf{x}, {}^B\theta^*)$. The last part of this proof is to express $f(\mathbf{x}, \theta^*) - f(\mathbf{x}, {}^\lambda\theta_m)$ and cancel out a lot of stuff. Let us take (equation 3.19 − equation 3.20). We know that the zero-type neurons sum to zero in both. The line before the last in equation 3.20 also cancels out with the linear combination of the functions. This leaves us with:

$$\begin{aligned} f(\mathbf{x}, \theta^*) - f(\mathbf{x}, {}^\lambda\theta_m) = &\lambda(\sum_{i=1}^{k_1} {}^A\mathbf{a}_1^i \sigma(\mathbf{w}_1\mathbf{x}) + \sum_{i=1}^{k_2} {}^A\mathbf{a}_2^i \sigma(\mathbf{w}_2\mathbf{x})) + \\ &+ (1-\lambda)(\sum_{i=1}^{k_1-1} {}^B\mathbf{a}_1^i \sigma(\mathbf{w}_1\mathbf{x}) + \sum_{i=1}^{k_2+1} {}^B\mathbf{a}_2^i \sigma(\mathbf{w}_2\mathbf{x})) + \\ &+ (-1) \sum_{i=1}^{k_1-1} (\lambda^A \mathbf{a}_1^i + (1-\lambda)^B \mathbf{a}_1^i)\sigma(\mathbf{w}_1\mathbf{x}) + \\ &+ (-1)(\lambda^A \mathbf{a}_1^{k_1} + (1-\lambda)^B \mathbf{a}_2^1)\sigma((\lambda\mathbf{w}_1 + (1-\lambda)\mathbf{w}_2)\mathbf{x}) + \\ &+ (-1) \sum_{i=1}^{k_2} (\lambda^A \mathbf{a}_2^i + (1-\lambda)^B \mathbf{a}_2^{i+1})\sigma(\mathbf{w}_2\mathbf{x}) \end{aligned} \tag{3.21}$$

From the fist line in equation 3.21 we can remove the the k_1-th neuron from the first sum. In the second line we can remove the first element of the second sum, the neuron corresponding to $\mathbf{a}_2^1$.

$$
\begin{aligned}
f(\mathbf{x},\theta^*) - f(\mathbf{x},{}^{\lambda}\theta_m) = \sum_{i=1}^{k_1-1} \lambda^A \mathbf{a}_1^i \sigma(\mathbf{w}_1\mathbf{x}) + \lambda^A \mathbf{a}_1^{k_1}\sigma(\mathbf{w}_1\mathbf{x}) + \sum_{i=1}^{k_2} \lambda^A \mathbf{a}_2^i \sigma(\mathbf{w}_2\mathbf{x}) + \\
+ \sum_{i=1}^{k_1-1} (1-\lambda)^B \mathbf{a}_1^i \sigma(\mathbf{w}_1\mathbf{x}) + (1-\lambda)^B \mathbf{a}_2^1 \sigma(\mathbf{w}_2\mathbf{x}) + \sum_{i=1}^{k_2} (1-\lambda)^B \mathbf{a}_2^{i+1} \sigma(\mathbf{w}_2\mathbf{x}) + \\
+(-1)\sum_{i=1}^{k_1-1} (\lambda^A \mathbf{a}_1^i + (1-\lambda)^B \mathbf{a}_1^i)\sigma(\mathbf{w}_1\mathbf{x}) + \\
+(-1)(\lambda^A \mathbf{a}_1^{k_1} + (1-\lambda)^B \mathbf{a}_2^1)\sigma((\lambda\mathbf{w}_1 + (1-\lambda)\mathbf{w}_2)\mathbf{x}) + \\
+(-1)\sum_{i=1}^{k_2} (\lambda^A \mathbf{a}_2^i + (1-\lambda)^B \mathbf{a}_2^{i+1})\sigma(\mathbf{w}_2\mathbf{x}) \qquad (3.22)
\end{aligned}
$$

We can further reorganize the sums in the top two rows in equation 3.22, according to $\sigma(\mathbf{w}_1\mathbf{x})$ and $\sigma(\mathbf{w}_2\mathbf{x})$. Doing this they will cancel out with rows three and five respectively. This leaves us with:

$$
\begin{aligned}
f(\mathbf{x},\theta^*) - f(\mathbf{x},{}^{\lambda}\theta_m) = \lambda^A \mathbf{a}_1^{k_1}\sigma(\mathbf{w}_1\mathbf{x}) + (1-\lambda)^B \mathbf{a}_2^1 \sigma(\mathbf{w}_2\mathbf{x}) + \\
+(-1)(\lambda^A \mathbf{a}_1^{k_1} + (1-\lambda)^B \mathbf{a}_2^1)\sigma((\lambda\mathbf{w}_1 + (1-\lambda)\mathbf{w}_2)\mathbf{x}) \qquad (3.23)
\end{aligned}
$$

By reorganizing the multiplication by factoring out ${}^A\mathbf{a}_1^{k_1}$ and ${}^B\mathbf{a}_2^1$ we get the equation given in theorem 3.

□

We used an example in theorem 3, where there was only one mismatched neuron between the linear combination of the two models. This mismatch was between a neuron of $\mathbf{w}_1$ and $\mathbf{w}_2$. This theorem and proof can be done for any indices and any number of neuron mismatches. Each mismatch will induce a similar error term. Furthermore, mismatches between zero-type neurons with themselves and zero-type neurons with duplicated neurons can also be considered.

This error will induce an increase and a barrier in the loss function. Since we know, that $y_i = f(\mathbf{x},\theta^*)$, for the loss function, on a finite dataset $\mathcal{D}$, we have:

$$
\mathcal{L}_{\mathcal{D}}(\theta) = \frac{1}{N}\sum_{i=1}^{N} \frac{1}{2}(f(\mathbf{x}_i,\theta^*) - f(\mathbf{x}_i,{}^{\lambda}\theta_m))^2 \qquad (3.24)
$$

To summarize this section. We showed, that there is no mismatch if we converged to the same affine subspace, up to permutation. We also expressed, where the increase in the loss function comes from if we converged to different subspaces. They come from the neurons, which could not be matched with the same type of neurons.

3.4 Eliminating the Barrier

Now that we know, where the barrier comes from, we can talk about how to eliminate it. We will discuss two methods, one where knowledge of $s = (k_1, k_2, \ldots, k_{r^*}, b_1, \ldots, b_j)$ is needed, and one where it is not. We will further discuss pruning and extension.

If we have knowledge of $s = (k_1, \ldots, k_{r^*}, b_1, \ldots, b_j)$, we can prune the network to the irreducible form of $\theta^*_{r^*}$. To achieve this, we need to remove zero-type neurons and combine duplicated neurons. If we do this to both models, we can combine them, after permutation, with zero loss. Note, that this method might not work for every activation function, for example for tanh we might need to do some invariant transformations beforehand, in order to make the signs align in both models.

We can also extend the two models with extra neurons. If we know ${}^A s$ and ${}^B s$, we can extend them both to a new subspace, defined by ${}^C s$, in a higher dimension. A possible subspace to extend the models into is defined by

$$ {}^C s = (max({}^A k_1, {}^B k_1), \ldots, max({}^A k_{r^*}, {}^B k_{r^*}), max({}^A b_1, {}^B b_1), \ldots, max({}^A b_j, {}^B b_j)). \quad (3.25) $$

To extend into this subspace we can add neurons with respective incoming weights, and zero outgoing weights. If ${}^A k_i < {}^B k_i$, we can add neurons to model A, ${}^B k_i - {}^A k_i$ number of them, with incoming weights $\mathbf{w}_i$ and out going weights $\mathbf{a}_i = \mathbf{0}$. We can do the same for model B, and for zero-type neurons. This will result in them being in the, up to permutation, same affine subspace, where we have proven in theorem 1, that up to permutation they are linearly mode connected. This type of extension brings in $\frac{|{}^A s' - {}^B s'|_1}{2}$ number of new neurons, where s' is the vector created from the tuple s.

If we do not have knowledge of s, we can do a greedy extension. Let us copy all neurons from model A to model B, for the copies, we will take the same incoming weights as from where we copied, but will have an out going weight of zero. If we do this for model A as well, then we will be guaranteed to be, with both points up to permutation, in the subspace, defined by

$$ {}^C s = ({}^A k_1 + {}^B k_1, \ldots, {}^A k_{r^*} + {}^B k_{r^*}, {}^A b_1 + {}^B b_1, \ldots, {}^A b_j + {}^B b_j), \quad (3.26) $$

where we know them to be linearly mode connected. This extension however is quite expensive, as it doubles the number of neurons in our hidden layer.

3.5 How Overparameterization Might Help

Here we will present a working theory as to why overparameterization might help. In theorem 3 we have shown, that the loss arises due to mismatched neurons. We have further seen, that at every mismatch the loss arising is proportional to the outgoing weights of the mismatched neurons. In this section we will focus on parts of the expansion manifold with only duplicated neurons, and no zero-type neurons.

First we will discuss the number of mismatches, and the way we can calculate how it changes with the width of the model. To calculate it, we can think of the different subspaces as a node in a graph. Each subspace is connected to another by a weighted edge, representing the number of mismatches between the two subspaces after an optimal permutation. We know that the order in which the neurons are present in the graph does not matter, as we can permute them to best match with each other, so we can simplify our graph. In figure 3.2 we show how we can represent the problem with a graph.

In figure 3.3 we can see an example of the ratio of subspace pairs with a given number of neuron mismatches, to all subspace pairs. In figure 3.4 we show how the average number

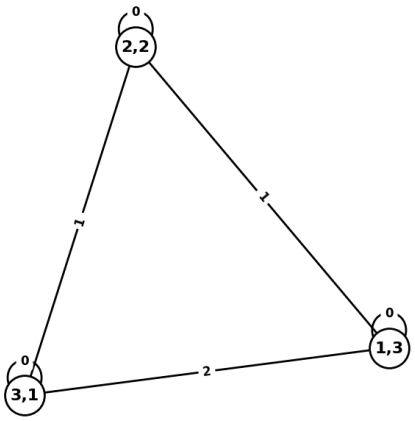


Figure 3.2: A representation of the number of mismatches as a graph. Here r^* is two and m is four. We represent $s = (1, 3)$, $s = (2, 2)$ and $s = (3, 1)$ as a node in the graph, and write the number of mismatches as weights between the nodes. We further know, that each node in the graph has $\frac{m!}{k_1!...k_{r^*}!}$ number of affine subspaces inside it due to permutation [19]. The problem this way is reduced to calculating the number of edges for each weight.

of mismatched neurons between two optimal parametrizations changes with the scale of overparameterization. We can see that it increases, but the rate of increase slows down the more overparameterized we get.

This is one part of the story, we have shown, that the more overparameterized the model, the more likely it is that there are neurons which cannot be paired with each other. This would suggest, that as the scale of overparameterization increases, the loss and barrier does too, since there will be more and more error terms in the loss of the linear interpolation. However, as previously discussed, the other factor which has a big impact on the loss are the outgoing weights. We believe, that the scale of the outgoing weights will decrease more rapidly then the number of mismatches increases. One example would be if they are all equally distributed, then it would be a linear decrease.

This is still a formulating theory, and more work needs to be done, in order to be assured that this is indeed the reason as to why in the overparameterized regime the barrier of linear mode connectivity of well trained solutions tends to decrease as the width of models increases.

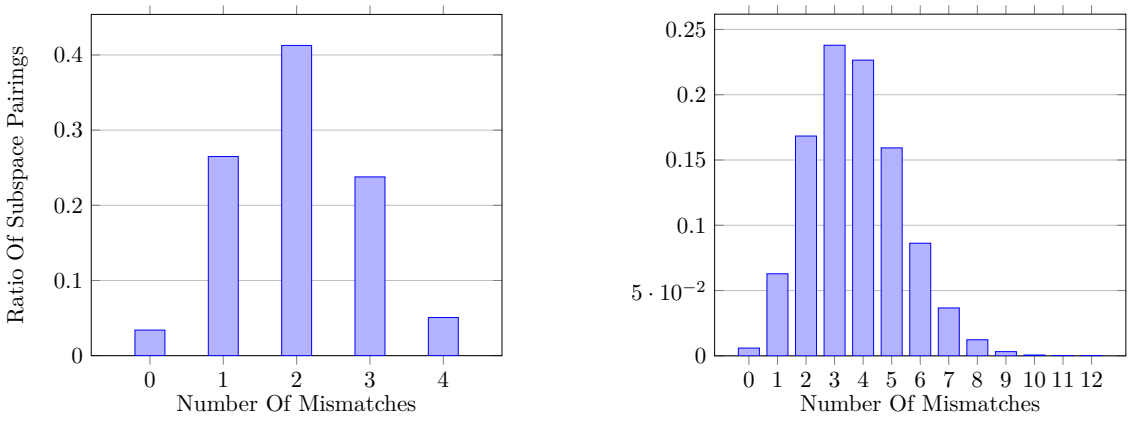


Figure 3.3: *Left:* The ratio of subspace pairings to the number of mismatches there are between them. Here $r^* = 4$ and $m = 8$. *Right:* The same ratio but with $m = 16$, notice how there are, on average, more mismatches, however they are more concentrated at smaller numbers.

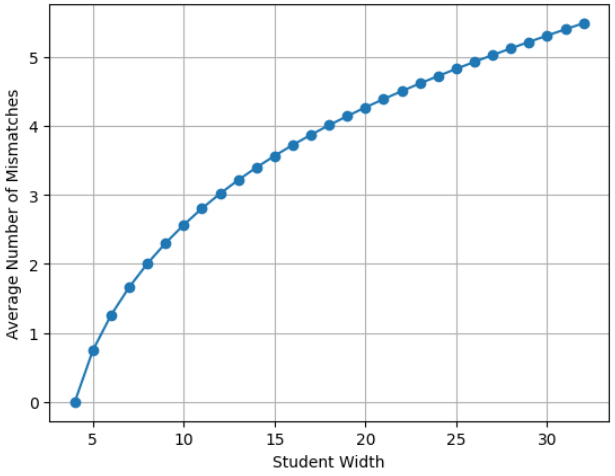


Figure 3.4: Using the data from figure 3.3, we can calculate the average number of mismatches between two subspaces for a given width. Here we calculated up to 32 neurons, and the minimal width is $r^* = 4$. The rate of change decreases, the more overparameterized the model.

Chapter 4

Empirical Experiments

4.1 Experimental Setup

In our setup we considered two layered neural networks, $f(\mathbf{x}, \theta) = \sum_i^m \mathbf{a}_i \sigma(\mathbf{w}_i \mathbf{x})$, where the activation function is $\sigma(x) = \sigma_{soft}(x) + \sigma_{sigm}(x)$. The output dimension, $d_{\text{out}} = 1$ and input dimension $d_{\text{in}} = 2$. For the minimal width we chose $r^* = 4$, with the unique, up to permutation, teacher model, ${}^T\theta^*_{r^*}$, weights being:

$$\mathbf{w}_1 = [0.5, 0.6], \mathbf{w}_2 = [-0.5, 0.5], \mathbf{w}_3 = [-0.3, -0.5], \mathbf{w}_4 = [0.7, -0.6] \text{ and } \forall i \in [r^*] : \mathbf{a}_i = 1$$

To create the database, we evaluated the function $f(\mathbf{x}, \theta^*_{r^*})$ on a grid of x_1 and x_2 going from -5 to 5 with a step size of 0.25. This is the $\mathcal{D} = \{(\mathbf{x}_i, f(\mathbf{x}_i, \theta^*_{r^*})) | i \in [1681]\}$ on which the models were trained.

The database can be seen in figure 4.1.

This is a regression problem, for the loss function we used mean square error, and took gradient descent steps with Adam optimizer [11], with a learning rate of 0.0005. We trained the models for 40000 or, when m was sufficiently large (16 neurons), 50000 epochs. We always saved the parameters achieving the best loss value and only accepted a model as converged, when $\mathcal{L}_{\mathcal{D}}(\theta^*_m) < 10^{-7}$.

4.2 Overparameterization Decreases the Barrier After Permutation

In this experiment we wanted to explore how the width m affects the barrier after permutation. We know, that between irreducible networks, the barrier is zero after permutation, because they are unique up to permutation. We also know that an error in the function can arise if duplicated neurons are not matched with duplicated neurons and b-zero-type neurons with b-zero-type neurons. Furthermore, empirical evidence suggests that after permutation the barrier decreases as the width increases. In order to test this we trained models on eight widths, for each width we trained eight models, differing only in the initialized parameters. We checked the maximum loss between the interpolation of models $(1, 2), (3, 4), (5, 6)$ and $(7, 8)$. We omitted checking more connections, because then the measurements would not have been independent. Note that because the models are converged and endpoints basically achieve zero loss, the maximum loss is more or less equivalent to the barrier.

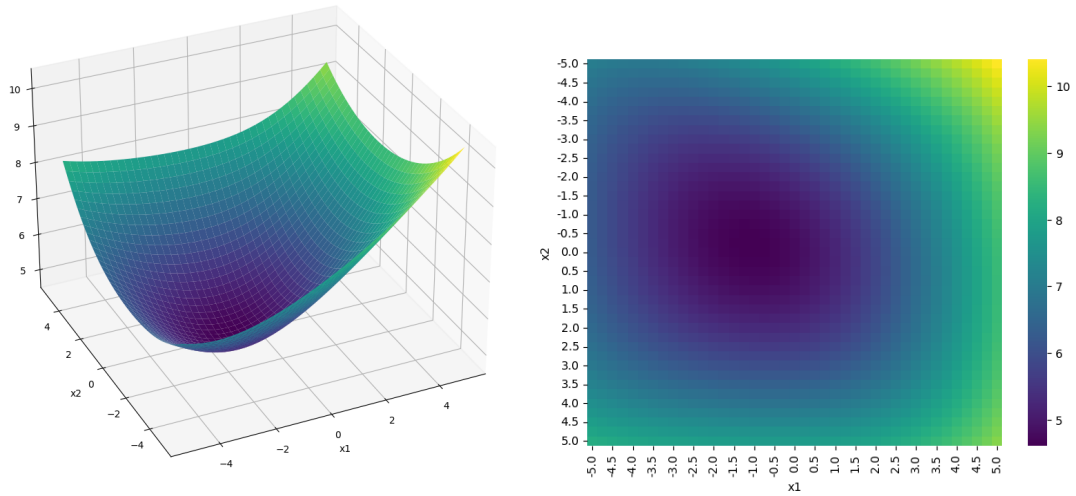


Figure 4.1: This is the function, which the neural network has to learn. The database consists of each grid-point that can be seen and their respective value, shown by color, and on the left figure.

In figure 4.2 we can see the barrier between models of different widths before permutation. In figure 4.3 we can see the barrier after matching the weights of one model to another. Notice the drastic decrease in barriers, and the difference between mildly and largely overparameterized models.

Note, that the decrease in the barrier is not explainable by the decrease in L_2 distance as can be seen in figure 4.4

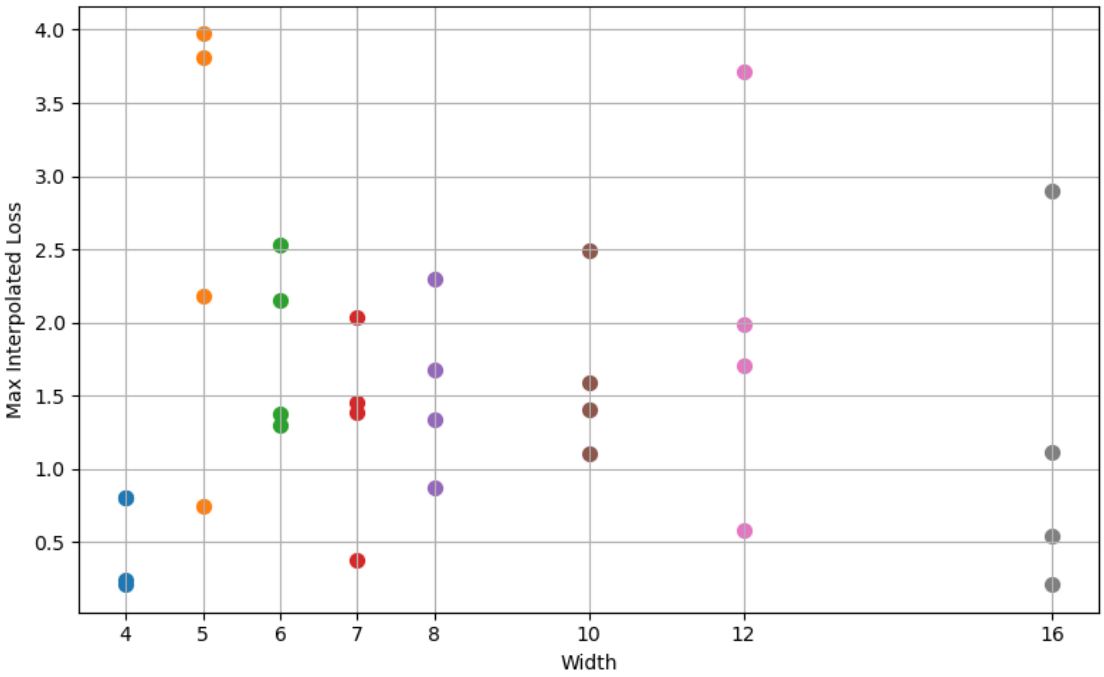


Figure 4.2: In this figure each dot represents the maximum of the loss value of the interpolated, trained, models. This is before weight matching.

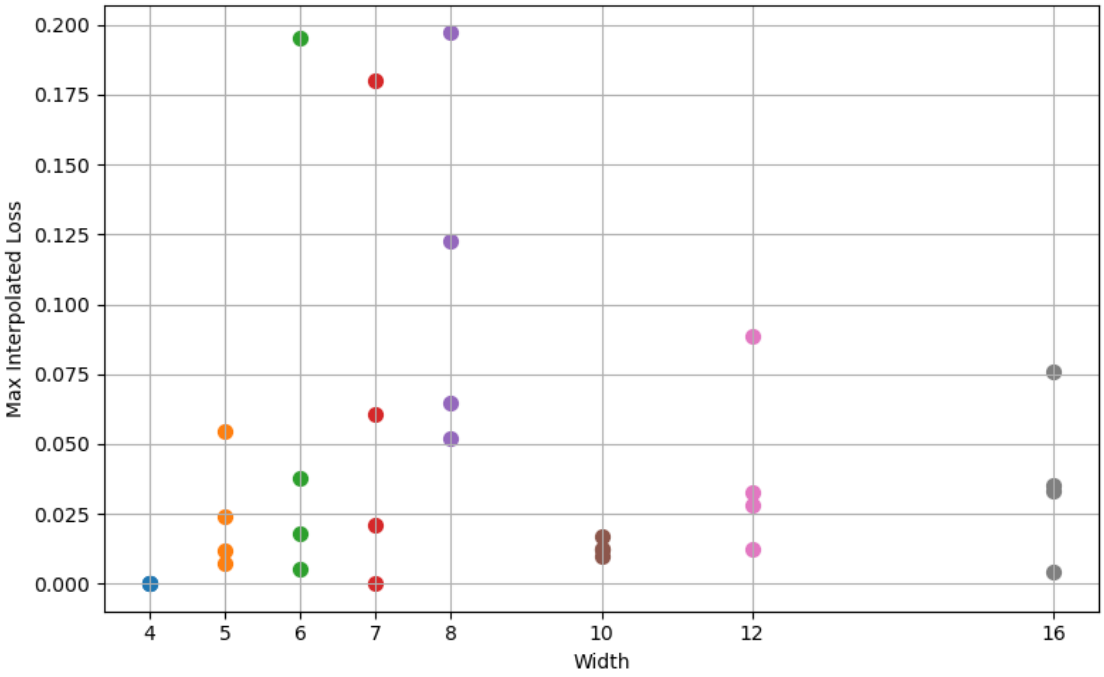


Figure 4.3: As can be seen on the scale of the y-axis, the barriers decreased by a lot after weight matching. However, there is still quite a bit of deviation between the barriers in each width. Generally speaking models from width 10 achieved smaller losses.

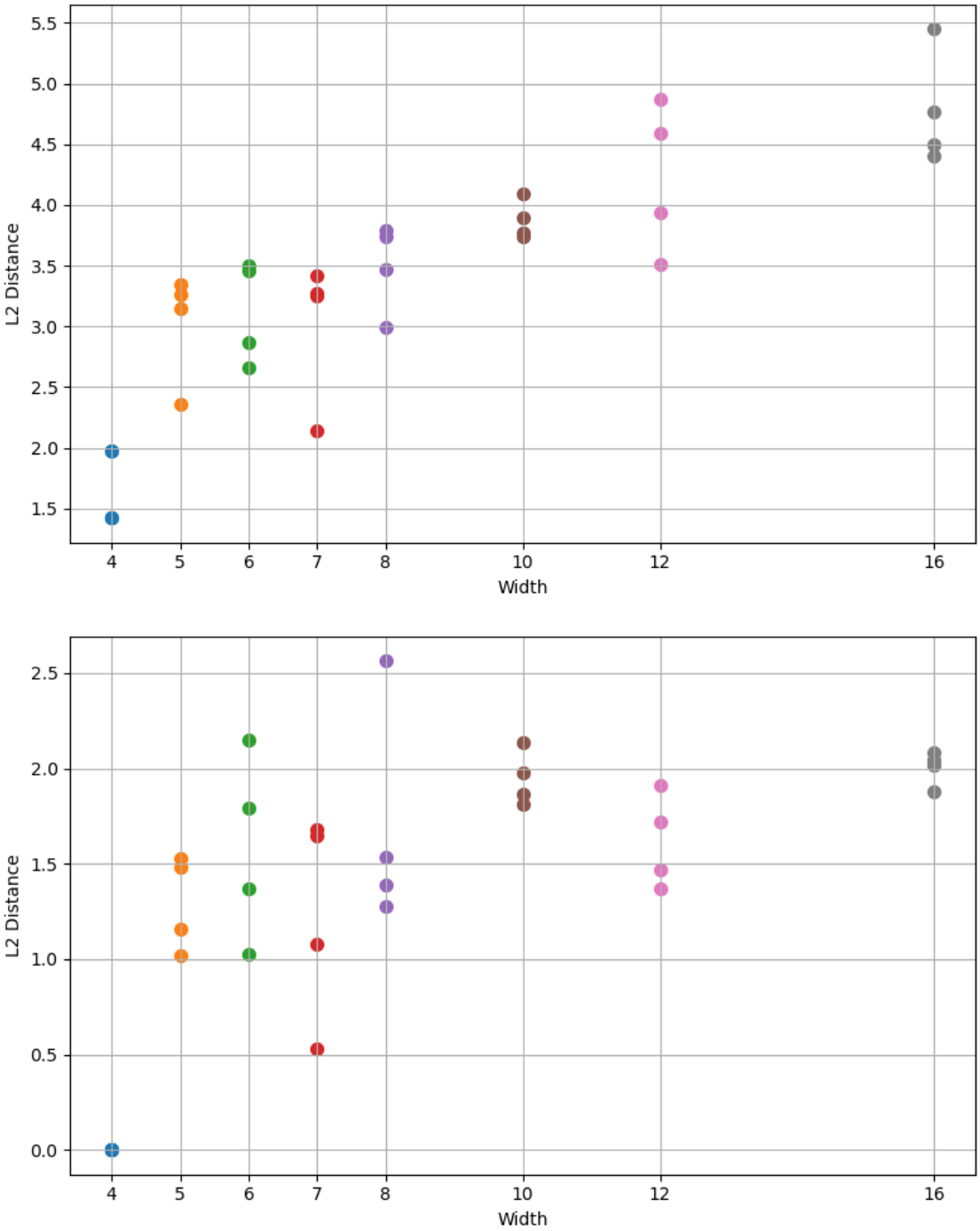


Figure 4.4: *Top:* L_2 distances between the above models before permutation. *Bottom:* L_2 distances between the above models after weight matching. The distance between models did not decrease, however, due to the increase in the parameter-space dimension this is not unsurprising.

4.3 Converged Models

In this section we will give examples to how in an overparameterized setting the model converged into the expansion manifold.

First of all though, we want to empirically show, that there does not exist a model, that achieves zero loss and has fewer neurons then the teacher outlined in section 4.1, this can be seen in figure 4.5.

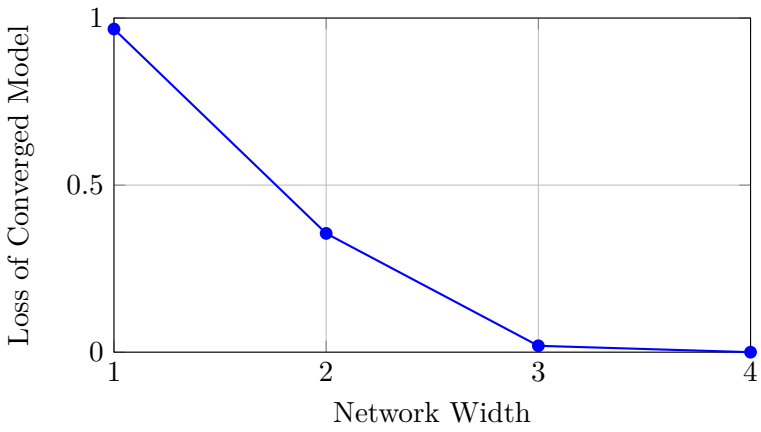


Figure 4.5: Models with less than four neurons cannot solve the regression problem. Even though the model with three neurons seems to have achieved quite well, the loss was still 0.019. This experiment empirically validates the minimality of $r^* = 4$.

Next we will show, that $\theta^*_{r^*}$ converges to the teacher network, up to permutation, and how weight matching permutes one model to the other (figure 4.6).

So how exactly would weight matching work here? In figure 4.7 we have another converged model and their permutation to the one shown in figure 4.6.

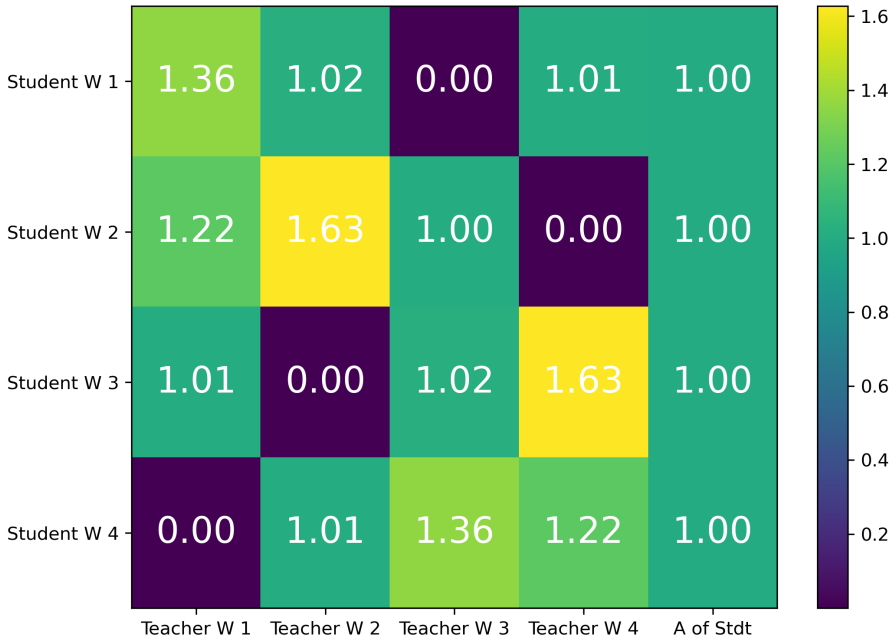


Figure 4.6: In this figure in the first $m \times r^*$ (4×4 in this case) matrix the L_2 distance between the teacher incoming weights, as defined in section 4.1, and student network incoming weights can be seen. The student network converged to the teacher weights, each row and column has one square, where the distance is 0. The last column shows the out weights of each neuron. These also converged to the weight, $\mathbf{a}_i = 1$, of the teacher network. In this example $\theta^*_{r^*} = \mathcal{P}_\pi({}^T\theta^*_{r^*})$, where $\pi = [3, 4, 2, 1]$.

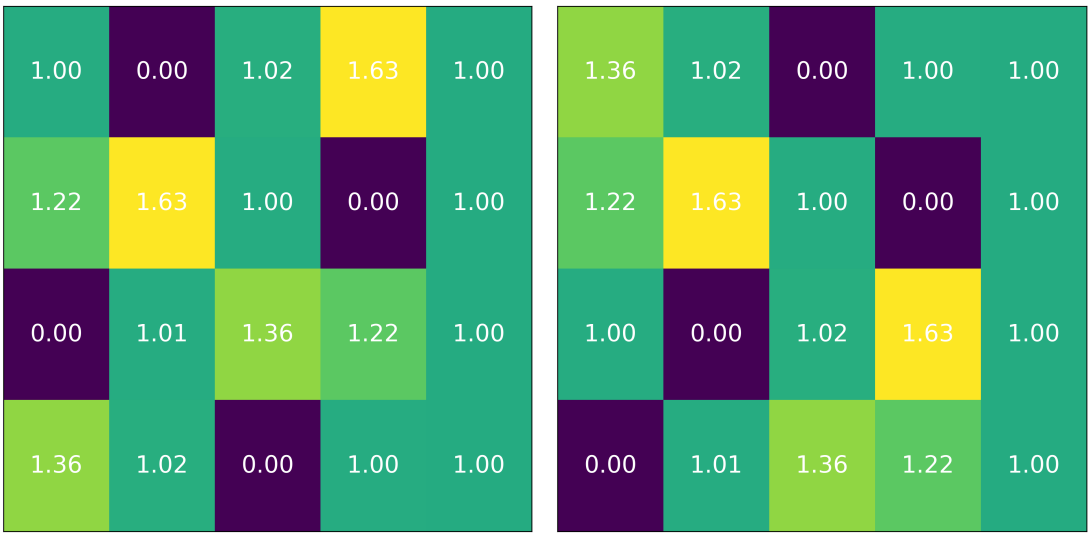


Figure 4.7: *Left:* Another optimal model, which converged to the same global minimum up to permutation. In this example $\theta^*_{r^*} = \mathcal{P}_\pi({}^T\theta^*_{r^*})$, where $\pi = [2, 4, 1, 3]$. *Right:* The model after weight matching to the parameters in figure 4.6. Notice how it took up the same order of neurons.

Next we will present an example of duplicated, and zero type neurons in the converged models with $m \geq r^*$. Note that here there are small errors in bigger networks, that arise from only converging to $\mathcal{L}_{\mathcal{D}}(\theta^*_m) < 10^{-7}$. Still, the neurons are mostly identifiable.

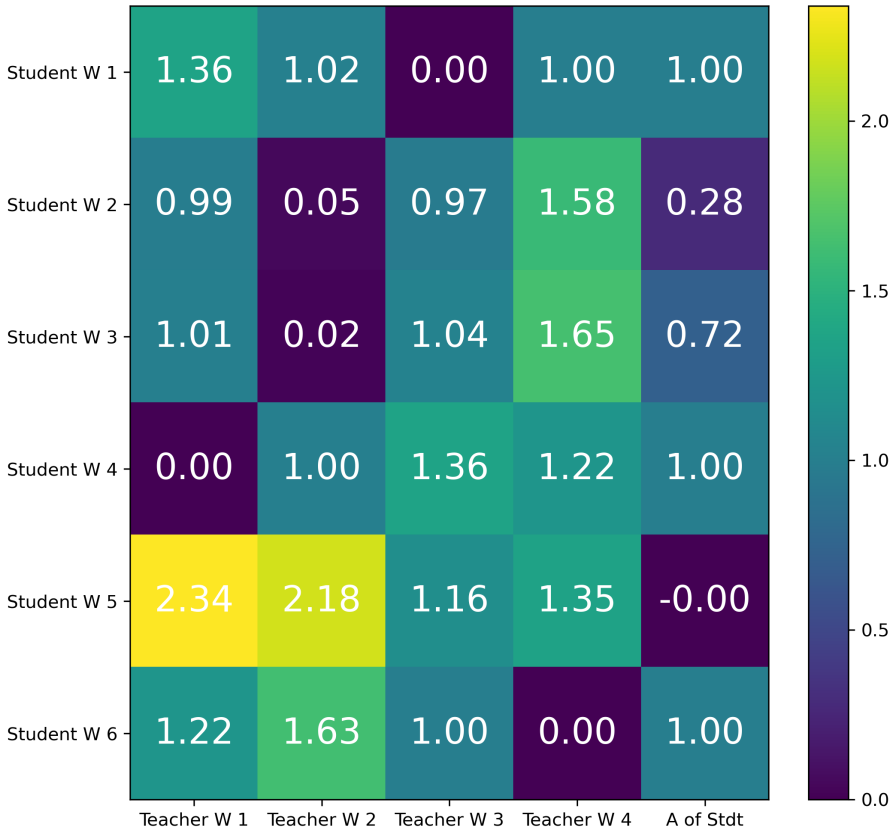


Figure 4.8: In this example a student network with six neurons was trained until convergence. Here weights one, four and six are copies of the teacher. We can see a duplicated neuron in the student weights two and three, where they converged to the weight of teacher neuron two, and their out weight sums up to one. We can also see a zero-type neuron, which has weights, that are not of the teacher, but also an out weight of zero. In this example $\theta^*_m \in \mathcal{P}_\pi \Gamma_s({}^T\theta^*_{r^*})$, where $m = 6$, $s = (k_1 = 1, k_2 = 2, k_3 = 1, k_4 = 1, b_1 = 1)$ and $\pi = [4, 2, 3, 1, 6, 5]$.

Chapter 5

Future Work

In this work we have discussed the symmetries and invariances, that can be found in neural networks. We have identified parts of the parameter-space, where there exists linear mode connectivity between models, both before and after permutation. We have also shown where the loss comes from when interpolating, and how one can reduce or even eliminate the barrier. We have also presented a working theory on the success of linear mode connectivity in the overparameterized regime. We further backed our findings with empirical evidence.

One important part will be further delving into the uniqueness of irreducible neural networks. In future work we need to prove or disprove the uniqueness of an irreducible neural network with the activation function $\sigma(x) = \sigma_{soft}(x) + \sigma_{sigm}(x)$.

Another line of work would be inspecting linear mode connectivity for different, more realistic, activation functions, such as for tanh or $ReLU$. In order to do this we would first need to identify all the invariances, under which a network with minimal width is unique. Next we would need to see if any new global minima can arise in an overparameterized network, or if convergence into the expansion manifold, extended with additional invariances, is guaranteed. Then we could discuss the linear combination of models, and express the error term.

We have given a possible theory as to why overparameterization might help reduce the barrier after permutation. In order to further examine this we would need to understand how the outgoing weights get distributed when the width of the model increases.

Bibliography

[1] Samuel K. Ainsworth, Jonathan Hayase, and Siddhartha Srinivasa. Git re-basin: Merging models modulo permutation symmetries, 2023. URL https://arxiv.org/abs/2209.04836.

[2] Francesca Albertini and Eduardo Sontag. Uniqueness of weights for neural networks. 04 1993.

[3] Gül Sena Altıntaş, Devin Kwok, and David Rolnick. The butterfly effect: Tiny perturbations cause neural network training to diverge. In *High-dimensional Learning Dynamics 2024: The Emergence of Structure and Reasoning*, 2024. URL https://openreview.net/forum?id=T1urv73edU.

[4] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020. URL https://arxiv.org/abs/2005.14165.

[5] Felix Draxler, Kambis Veschgini, Manfred Salmhofer, and Fred Hamprecht. Essentially no barriers in neural network energy landscape. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1309–1318. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/draxler18a.html.

[6] Rahim Entezari, Hanie Sedghi, Olga Saukh, and Behnam Neyshabur. The role of permutation invariance in linear mode connectivity of neural networks, 2022. URL https://arxiv.org/abs/2110.06296.

[7] Jonathan Frankle, Gintare Karolina Dziugaite, Daniel M. Roy, and Michael Carbin. Linear mode connectivity and the lottery ticket hypothesis, 2020. URL https://arxiv.org/abs/1912.05671.

[8] Timur Garipov, Pavel Izmailov, Dmitrii Podoprikhin, Dmitry Vetrov, and Andrew Gordon Wilson. Loss surfaces, mode connectivity, and fast ensembling of dnns. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 8803–8812, Red Hook, NY, USA, 2018. Curran Associates Inc.

[9] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015. URL https://arxiv.org/abs/1502.03167.

[10] Animesh Karnewar. AANN: Absolute artificial neural network, 2018. URL https://openreview.net/forum?id=rkhxwltab.

[11] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017. URL https://arxiv.org/abs/1412.6980.

[12] Rohith Kuditipudi, Xiang Wang, Holden Lee, Yi Zhang, Zhiyuan Li, Wei Hu, Sanjeev Arora, and Rong Ge. Explaining landscape connectivity of low-cost solutions for multilayer nets, 2020. URL https://arxiv.org/abs/1906.06247.

[13] Zhiyuan Li and Sanjeev Arora. An exponential learning rate schedule for deep learning, 2019. URL https://arxiv.org/abs/1910.07454.

[14] Andrew L. Maas, Awni Y. Hannun, and Andrew Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proceedings of the 30th International Conference on Machine Learning (ICML), Workshop on Deep Learning for Audio, Speech, and Language Processing*, 2013.

[15] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, page 807–814, Madison, WI, USA, 2010. Omnipress. ISBN 9781605589077.

[16] Henning Petzka, Martin Trimmel, and Cristian Sminchisescu. Notes on the symmetries of 2-layer relu-networks. *Proceedings of the Northern Lights Deep Learning Workshop*, 1:6, 02 2020. DOI: 10.7557/18.5150.

[17] Héctor J. Sussmann. Uniqueness of the weights for minimal feedforward nets with a given input-output map. *Neural Networks*, 5(4):589–593, 1992. ISSN 0893-6080. DOI: https://doi.org/10.1016/S0893-6080(05)80037-1. URL https://www.sciencedirect.com/science/article/pii/S0893608005800371.

[18] Pu Zhao, Pin-Yu Chen, Payel Das, Karthikeyan Natesan Ramamurthy, and Xue Lin. Bridging mode connectivity in loss landscapes and adversarial robustness, 2020. URL https://arxiv.org/abs/2005.00060.

[19] Berfin Şimşek, 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, 2021. URL https://arxiv.org/abs/2105.12221.