Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Artificial Intelligence and Systems Engineering

Symmetries and Invariances in Overparameterized Neural Networks

Bachelor's Thesis

Author
Márk Penc

Advisor
Bálint Daróczy
HUN-REN Institute for
Computer Science and Control

Dr. Gábor Hullám
BME VIK Department of Artificial
Intelligence and Systems Engineering

December 6, 2024

Contents

HALLGATÓI NYILATKOZAT

Alulírott *Penc Márk*, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy autentikált felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Budapest, 2024. december 6.

Penc Márk
hallgató

Kivonat

Az elmúlt években a túlparaméterezett neurális hálózatok tanulmányozása egyre népszerűbbé vált. A túlparaméterezettség, vagyis a tanító adatpontokhoz való illeszkedéshez szükségesnél több paraméterrel való rendelkezés, számos előnnyel jár. Például javíthat az általánosítási képességen, jobb optimalizálási garanciákat biztosíthat, és egyszerűsítheti a modellméret megválasztását [5]. Fontos kutatási terület a túlparaméterezett neurális hálózatok hibafelületének megértése, különös tekintettel a globális minimumpontok halmazára. Korábbi munkák kimutatták, hogy ezekben a hálózatokban az azonos függvényt eredményező különböző paraméterezések miatt létezik egy összefüggő globális minimumhalmaz, ahol bármely két pontot egy olyan út köti össze, amely mentén a veszteség értéke nulla [38].

Ebben a munkában néhány olyan invarianciát fogunk kifejteni, amelyek mellett a hálózat függvénye változatlan marad. Ezt követően azonosítjuk, hogy mikor lehet a paramétertérben lineárisan kombinálni ezeket a függvénybeli invariáns hálózatokat úgy, hogy azok kombinációja ne eredményezze a veszteségfüggvény növekedését. Ezután megkeressük azokat az eseteket, amikor lineáris kombinációjuk mégis a veszteség növekedését eredményezi, és kifejezzük a növekedés értékét. Megmutatjuk továbbá, hogy a modell dimenziójának bővítése extra neuronok hozzáadásával hogyan csökkentheti a veszteség növekedését két paraméterezés között. Továbbá bemutatunk egy elméletet arra vonatkozóan, hogy a túlparaméterezés miért segíthet csökkenteni a két paraméterezés lineáris kombinációjának hibáját. Úgy gondoljuk, hogy ez a munka segíthet áthidalni a legújabb empirikus eredmények és a matematikai elmélet közötti szakadékot, mélyebb megértést nyújtva a modern gépi tanulási modellek alapjául szolgáló elvekről.

Acknowledgements

I would like to express my deepest gratitude to Bálint Daróczy for his relentless support, insightful guidance, and invaluable expertise throughout this work. His encouragement and thoughtful advice have been instrumental not only in shaping the direction of this thesis but also in helping me navigate challenges with confidence.

I sincerely thank Dr. Gábor Hullám for their guidance and support during my studies, which have greatly influenced my academic growth, and for their role in facilitating this thesis.

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 their insights, ideas and feedback on the thesis.

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 overparameterized neural networks soared in popularity. Overparameterization, i.e. having more parameters than necessary to fit the training data, comes with several advantages. For example, it can improve generalization properties, provide better optimization guarantees, and simplify the selection of model sizes [5]. An important research area is understanding the loss landscape of overparameterized neural networks, with special focus to the set of global minima. Previous work has shown that in these networks, due to different parameterizations that produce the same function, there is a connected set of global minima, where any two points are connected by a path, along which the loss is zero [38].

In this work we will express some of the invariances, under which the function implemented by a network remains the same. We will then identify when functionally invariant networks can be linearly combined in the parameter space, so that their combination does not result in an increase of the loss function. Next we will find the cases where their linear combination does result in an increase of the loss and express the value of the increase. We will further show how expanding the dimension of a model, by adding specifically chosen extra neurons can reduce the increase in loss between two parameterizations. Furthermore, we present a working theory as to why overparameterization might help reduce the error of the linear combination of two parameterizations. We believe that this work can help bridging the gap between recent empirical findings and mathematical theory, providing a deeper understanding of the principles underlying modern machine learning 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, which has 175 billion parameters [7]. Due to this drastic increase in the amount of learnable parameters, the cost, time, and infrastructure required to train deep neural networks experienced 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 for better model combinations.

In this thesis we will focus on how the same function can be expressed by different parameterizations of neural networks and on how and when models can be linearly combined so that their combination continues to perform well on a given task.

1.1 Machine Learning and Neural Networks

In contrast to traditional algorithmic approach, machine learning tasks focus not on creating algorithms for a given problem, but on creating algorithms that, when given data, can gain some information from it. In this thesis we will focus on the problem of function approximation, where given some input and output data, we wish to find the function that approximates the data the best from a given set of possible hypothesis. A simple example of this problem would be a linear regression task.

1.1.1 Linear Regression

In linear regression the problem can be framed as finding a hyperplane that best approximates our data. Our dataset, consisting of N datapoints, is

$$\mathcal{D} = \{(\mathbf{x}_i, y_i) | i = \{1, \dots, N\}, \mathbf{x}_i \in \mathbb{R}^{d_{\text{in}}}, y_i \in \mathbb{R}\}, \tag{1.1}$$

where an $(\mathbf{x}_i, y_i)$ pair is an observation of an input and an output. Our goal is to find the **parameters**, $\theta = (\mathbf{w}^T, b)^T$, of a linear function of the form

$$f(\mathbf{x}, \mathbf{w}, b) = \mathbf{w}^T \mathbf{x} + b, \tag{1.2}$$

so that it fits the data the best. In order to measure how well a given **hypothesis**, $f(\mathbf{x}, \mathbf{w}, b)$, performs we can introduce the **loss function** to measure the difference between the prediction of our hypothesis and the expected output in the elements of our dataset. One commonly used loss function is the **mean squared error** (MSE) [14], where, for all observations $(\mathbf{x}_i, y_i) \in \mathcal{D}$, we measure the squared difference between the prediction of our hypothesis, $f(\mathbf{x}_i, \mathbf{w}, b)$, and the expected output, y_i:

$$\mathcal{L}_{\mathcal{D}}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{2} (f(\mathbf{x}_i, \mathbf{w}, b) - y_i)^2 \tag{1.3}$$

In the above loss function, we divide by $\frac{1}{N}$ in order to normalize by the size of the dataset. In addition, division by $\frac{1}{2}$ is a common practice to make the derivative of the loss function simpler to work with. Our goal is now reduced to finding the **optimal parameters** under which $\mathcal{L}_{\mathcal{D}}(\theta)$ takes up a global minimum. One of the most common methods is the first order optimization technique, **gradient descent**. Here, we calculate the gradient of the loss function at point θ and update the parameters of the model by taking an **update step** in the opposite direction:

$$\theta^{t+1} := \theta^t - \mu(t) \nabla_\theta \mathcal{L}_{\mathcal{D}}(\theta^t) \tag{1.4}$$

In the above update rule, θ^t denotes our parameters after t number of updates steps, furthermore, $\mu(t)$ denotes the **learning rate** at step t. If we repeat the above update rule, with a sufficiently chosen learning rate, until we don't see an improvement in the loss, we solved our linear regression task. There are plenty of strategies for choosing an optimal learning rate and for improving optimization algorithms, but the goal is always to find the minimum of the loss [14].

1.1.2 Neural Networks

In the linear regression problem we approximated the data using a linear function. What if our data, however, has a more complicated structure that cannot be described by a simple linear correspondence? In order to approximate with more diverse functions, we would need to expand our possible set of hypotheses, to encompass different kinds of functions, not only linear. One possible solution is to work with neural networks. There are many different kinds of neural networks, in this thesis we will focus on biasless **fully connected neural networks**.

Our neural network is composed of **neurons** [28, 35], which are computational units defined by the function $z : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$. The output of a neuron is given by $z(\mathbf{x}, \mathbf{w}) = \sigma\left(\mathbf{w}^T \mathbf{x}\right)$, where $\mathbf{x} \in \mathbb{R}^n$ represents the input, $\mathbf{w} \in \mathbb{R}^n$ are learnable parameters, and $\sigma : \mathbb{R} \to \mathbb{R}$ is called the **activation function**.

Fully connected neural networks, with L layers, consists of the input layer ($l = 0$), hidden layers ($l \in \{1, \ldots, L-1\}$) and an output layer ($l = L$). In each layer there are multiple neurons, we will denote the number of neurons in the l-th layer as m_l. A neuron in the i-th position of the l-th layer, $l \in \{1, \ldots, L-1\}$, has an output of $z_i^l = \sigma\left(\mathbf{w}_i^{l^T} \mathbf{z}^{l-1}\right)$, where the function σ is a non-linear activation function, and $\mathbf{z}^{l-1} \in \mathbb{R}^{m_{l-1}}$ are the outputs of the neurons in the previous layer, stacked on top of each other and $\mathbf{w}_i^l \in \mathbb{R}^{m_{l-1}}$ are the learnable parameters, furthermore, $\mathbf{z}^0 = \mathbf{x}$ is the input of our neural network. In our setup the output layer has a linear activation, so $z_i^L = y_i = \mathbf{w}_i^{L^T} \mathbf{z}^{L-1}$. Our fully connected neural network implements the function:

$$f(\mathbf{x}, \theta) = W^L \vec{\sigma}\left(W^{L-1}\vec{\sigma}\left(\dots\vec{\sigma}\left(W^1\mathbf{x}\right)\dots\right)\right), \tag{1.5}$$

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, 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 $\mathbf{w}_i^l$.

Training fully connected neural networks means finding the parameters that minimize a given loss function. This is most commonly done by gradient descent, but there are many other optimization algorithms to choose from [14]. The loss surface is usually not convex, and due to this sometimes we only find a local minimum.

Due to the nonlinearity of the activation functions, neural networks can represent a wide variety of functions. In fact, multiple studies have demonstrated that neural networks are universal function approximators [16, 12], meaning they can approximate any function to an arbitrarily small error, given sufficient capacity.

1.2 Overparameterized Neural Networks

During training, we only see a handful of data points drawn from an underlying distribution that we want to approximate with our function. A big question in statistical learning theory is how well we can generalize to unseen data. The main framework for this problem, **empirical risk minimization** (ERM), was introduced by Vapnik in [31].

In the previous section we briefly and informally mentioned the set of possible hypotheses we can chose from. We can call this set the **hypothesis set**, and denote it by $\mathcal{H}$. In order to discuss how versatile this set is, we can introduce the **capacity** [6, 5] of the hypothesis set, *cap*($\mathcal{H}$). This measure can, for example, be the Vapnik-Chervonenkis dimension [33, 32], which quantifies the largest set of points that can be shattered, i.e. classified correctly under all possible labelings, by functions chosen from the hypothesis set.

A well known property of neural networks is that as the capacity of the hypothesis set increases, the generalization bounds based on the capacity of the hypothesis set creates a *U-shape* [31]. This is because the risk bound can be decomposed to the empirical risk measured on the training set, which decreases as the capacity increases, and to a capacity term, which accounts to the complexity of our function and increases with the capacity.

A U-shaped generalization bound can be seen in figure 1.1. According to this classical understanding, finding an optimal model requires balancing the capacity of $\mathcal{H}$.

Before long though, it became clear that this was not the whole picture. As far back as 1990's researchers saw that trained, heavily parameterized models performed better then would be expected by ERM [11, 36, 5]. This observation brought the study of overparameterized neural networks, networks that have more parameters then necessary to fit the training data, to the forefront, and led to the development of new theoretical frameworks, which seek to explain how overparameterized networks are able to avoid overfitting and exhibit good generalization properties despite the large capacity of their hypothesis set.

As of our modern understanding, the core reason as to why heavily parameterized models can generalize well relates to the *smoothness* of the function [5]. Some algorithms, like those used in the training of neural networks, tend to find solutions where the produced

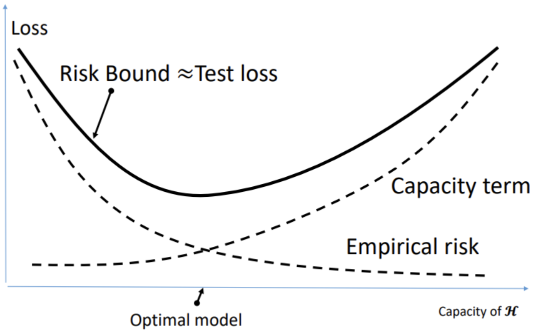


Figure 1.1: A U-shaped generalization curve. As the capacity of the hypothesis set increases, we can fit the data, provided during training, better, however, after a while the predictions to unseen data worsen. The decrease in performance to unseen data, while achieving good performance on the training set, is called **overfitting** [31]. (Figure from: [5]).

function is smooth [5]. This led to a new, **overparameterized regime** to develop, with many differences to the classical machine learning regime. Some advantages to overparameterization can include, but are not limited to better generalization properties, easier finding of the optimal model capacity and better convergence properties [5].

In figure 1.2, we can see our current understanding on how the risk changes, as the capacity of our hypothesis set increases. After being able to fit the training data perfectly, the risk starts decreasing.

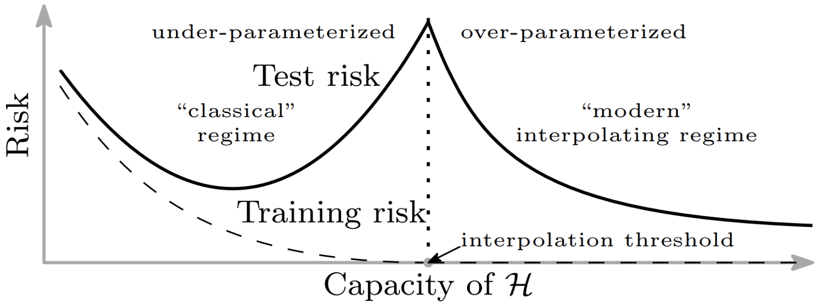


Figure 1.2: Our modern understanding of how the capacity of $\mathcal{H}$ relates to generalization. After reaching the interpolation threshold, where the network can fit the data perfectly, the generalization begins to improve and neural networks start predicting well again to unseen data. (Figure from: [6]).

In figure 1.3, we can see how the smoothness of the function can lead to better generalization after the interpolation threshold, where we achieve zero loss on our training data.

Many of the models commonly used today, such as transformers [34] and large ResNets [15] are considered to be overparameterized. Understanding the global minima of these models and how to effectively combine them is especially important in recent years, as it can save us some computing capacity.

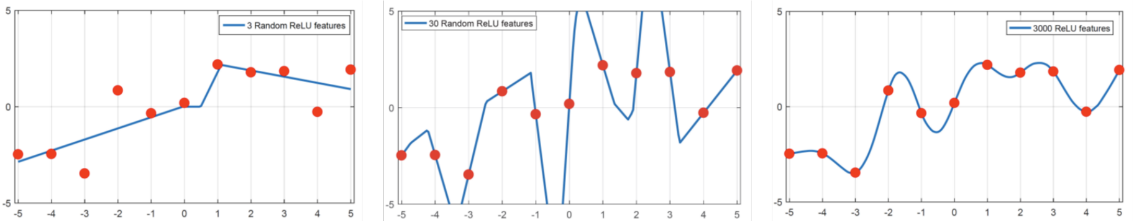


Figure 1.3: *Left:* A classical underparameterized neural network. *Middle:* A neural network, with slightly higher capacity then at the interpolation threshold. *Right:* A highly overparameterized neural network, able to fit the data more smoothly. (Figure from: [5]).

1.3 Model Combination

In this work when we talk about the combination of models, we refer to their combination in the parameter space. Our main motivation is mode-connectivity that is if we have two parameterizations, the question is if there exists a path between them, where the error of the predictions along the path does not increase much.

If we have $\theta_A, \theta_B \in \mathbb{R}^P$ as our two parameterizations, where P is the number of parameters in the model, we can look at a path, a continuous function mapping the interval $[0, 1]$ into the parameter space, $\gamma : [0, 1] \to \mathbb{R}^P$, where $\gamma(0) = \theta_B$ and $\gamma(1) = \theta_A$. Each point along this path will give us a valid parameterization of our neural network. We can then measure the loss function at the points. Our goal will be to see when this loss does not increase by much, i.e. the function $f(\mathbf{x}, \gamma(\lambda))$ performs well on the task, for $\lambda \in [0, 1]$.

We will mainly be focusing on the linear combination of models, here the points along the path will be defined as $\theta_\lambda := \lambda\theta_A + (1 - \lambda)\theta_B$, $\lambda \in [0, 1]$.

In figure 1.4 we can see the cross section of a ResNet. The image shows on a real world example to what we mean by model combination.

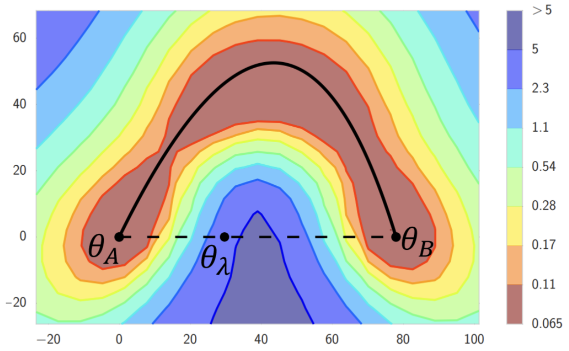


Figure 1.4: Two low loss parameterizations of a ResNet-164. The image is of the loss values along a planar cross section of the parameter space. By model combination we mean, for example, the two paths shown on the image. On the linear one, the loss increases by a lot, however the parameterizations on the other path perform well. (Figure from: [13]).

Looking at the combination of these models is motivated by multiple examples. One of them is understanding how the minima points are located in the parameter space, and understanding the geometry of the loss landscape. Another is the hope that we can say something about how well optimal points generalize. A third one is the hope that we can transfer knowledge between models.

1.4 Related Work

One line of work studies the mathematics of mode connectivity and the connectedness of global minima, such as [20], where they showed that for well-trained networks with generic properties, such as dropout stability and noise stability, the solutions are mode connected. In [38], they showed that overparameterized neural networks have connected global minima sets 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 implements a function there may be a lot of other solutions in the parameter space that produce an identical one. For example, fully connected neural networks can produce the same function when neurons from a layer are permuted [29]. Studying these functional invariances is useful since they give us 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 the parameters is studied. It has been empirically shown that as the width of the model increases the barrier, the maximal difference in the loss along a linear path between the two parametrizations, decreases [9]. Furthermore, it has been suggested that most stochastic gradient descent (SGD) [27] solutions belong to the same basin up to permutation [9]. 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 [2].

Mode connectivity and linear mode connectivity have been further studied in terms of generalization [10], adversarial robustness [37], and ensembling [13].

1.5 Thesis Structure

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 working 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, sections 3.3, 3.4 and 3.5, and in chapter 4, 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 a continuation of my Scientific Students' Association Report titled Symmetries and Invariances in Overparameterized Neural Networks [25]. Some changes include an extended introduction, new and more detailed explanations and images, rephrasing of the main ideas of the work, and general formatting changes.

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

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 [38], but there are slight changes.

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

Let $\theta \in \mathbb{R}^P$ 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 a two layered fully connected neural networks, will denote the amount of neurons, m, in the hidden layer, or θ^*, which will represent parameters, which reached a global minimum by some optimization method.

Our neural network will be the function $f : \mathbb{R}^{d_{\text{in}}} \times \mathbb{R}^P \to \mathbb{R}^{d_{\text{out}}}$, 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 a fully connected neural network. $w_{i,j}^l \in \mathbb{R}$ will refer to the weight going from the j-th neuron in the $(l-1)$-th layer to the i-th neuron of the l-th layer. We will denote $\mathbf{a}_i^l = \left(w_{1,i}^{l+1}, w_{2,i}^{l+1}, \ldots, w_{m_{l+1},i}^{l+1}\right)^T$ and call them 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 $\vartheta_i^l = \left(\mathbf{w}_i^{l\,T}, \mathbf{a}_i^{l\,T}\right)^T$.

In this work we will be focusing on biasless fully connected neural networks with linear output layer. Where, given an activation function $\sigma : \mathbb{R} \to \mathbb{R}$, we have the neural network output be

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

where $\vec{\sigma}(x_1, x_2, \ldots, x_n) = (\sigma(x_1), \sigma(x_2), \ldots, \sigma(x_n))^T$ is the activation function applied to each coordinate of a vector, 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 $\mathbf{w}_i^l$.

In this thesis, we will discuss networks with a varying number of neurons. For a two layered fully connected neural network, and input dimension of d_{in} and output dimension d_{out}, the **network dimension** depends on the number of neurons in the hidden layer, m.

The network dimension is defined as $\text{ndim}(m) := m(d_{\text{in}} + d_{\text{out}})$ and it is the dimension of the parameter space. When we do not have different parameter spaces with different dimensions, we will sometimes refer to it as P.

For the dataset we have $\mathcal{D} = \{(\mathbf{x}_i, y_i) | i = \{1, \dots, N\}, \mathbf{x}_i \in \mathbb{R}^{d_{\text{in}}}, y_i \in \mathbb{R}\}$, 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 $l(f(\mathbf{x}_i, \theta), y_i) = \frac{1}{2}(f(\mathbf{x}_i, \theta) - y_i)^2$. In this thesis we will assume that the loss surface is a smooth manifold [1].

In this work we will be focusing on networks that can reach zero loss on a given dataset. In order to create such a dataset we will use a **teacher** network. This teacher network will allow us to generate the data, which we want to fit to, by sampling the output of the network for given inputs. Note that this network trivially achieves zero loss.

For a given data generating function, i.e. output of a teacher network, $f(\mathbf{x}, \theta^*)$, and input data distribution μ with support in $\mathbb{R}^{d_{in}}$ 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 parameterizations under which the function implemented by a neural network remains invariant. In this section we gathered four types of invariances, 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, for example, in neural networks with normalization layers, such as batch normalization, there exists a scale-invariance [17, 21]. 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 [29] that neural networks have an invariance due to neuron permutations. On the other hand, many aspects of permutation symmetries were not explored to their fullest until recently. In this section we will show some of the interesting behavior that emerges due to permutation invariance.

Permutation invariance means that 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.

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

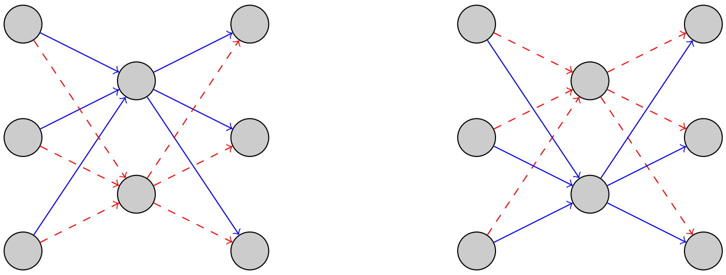


Figure 2.1: Two neural networks that are permutations of each other. The weights colored blue represent ϑ_1^l, and colored in red ϑ_2^l. If we reorder the neurons, so that the top one is ϑ_2^l and the bottom one ϑ_1^l, we get the same function.

l in the following way:

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

This operation is equivalent to exchanging the rows of W_l and columns of W_{l+1}, according to the permutation, π_l. Since there can be 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 a neural network is implementing. 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, $l \in [L-1]$. The output of the fully connected neural network at the l-th layer is

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

It is easy to see that

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

Now let us consider permuting the neurons in layer l, according to π_l. 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 means that the next layer has an output of

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

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

The fact that there can be permutation invariances in a neural networks means that each parameter in the parameter space 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. This means that if we teach permuted models, all the first, and higher order optimizations will be the same, up to permutation, and find the same global minimum, up to permutation.

This finding inspired a new wave of research which studies how global minima interact with their permutations, and what consequences this has in training neural networks. One line of research, which we will introduce in a later chapter is the conjecture that SGD solutions converge to the same minima basin, up to permutation. Due to the factorial nature of these symmetries finding an optimal permutation, by some measure, can be NP-hard, however many heuristic algorithms have been developed, such as those shown in [2].

In figure 2.2 we can see three different loss functions of simple neural networks. All of them implement the function $f(x, \theta) = \sigma(w_1 x) + \sigma(w_2 x)$, $x \in \mathbb{R}$, 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.

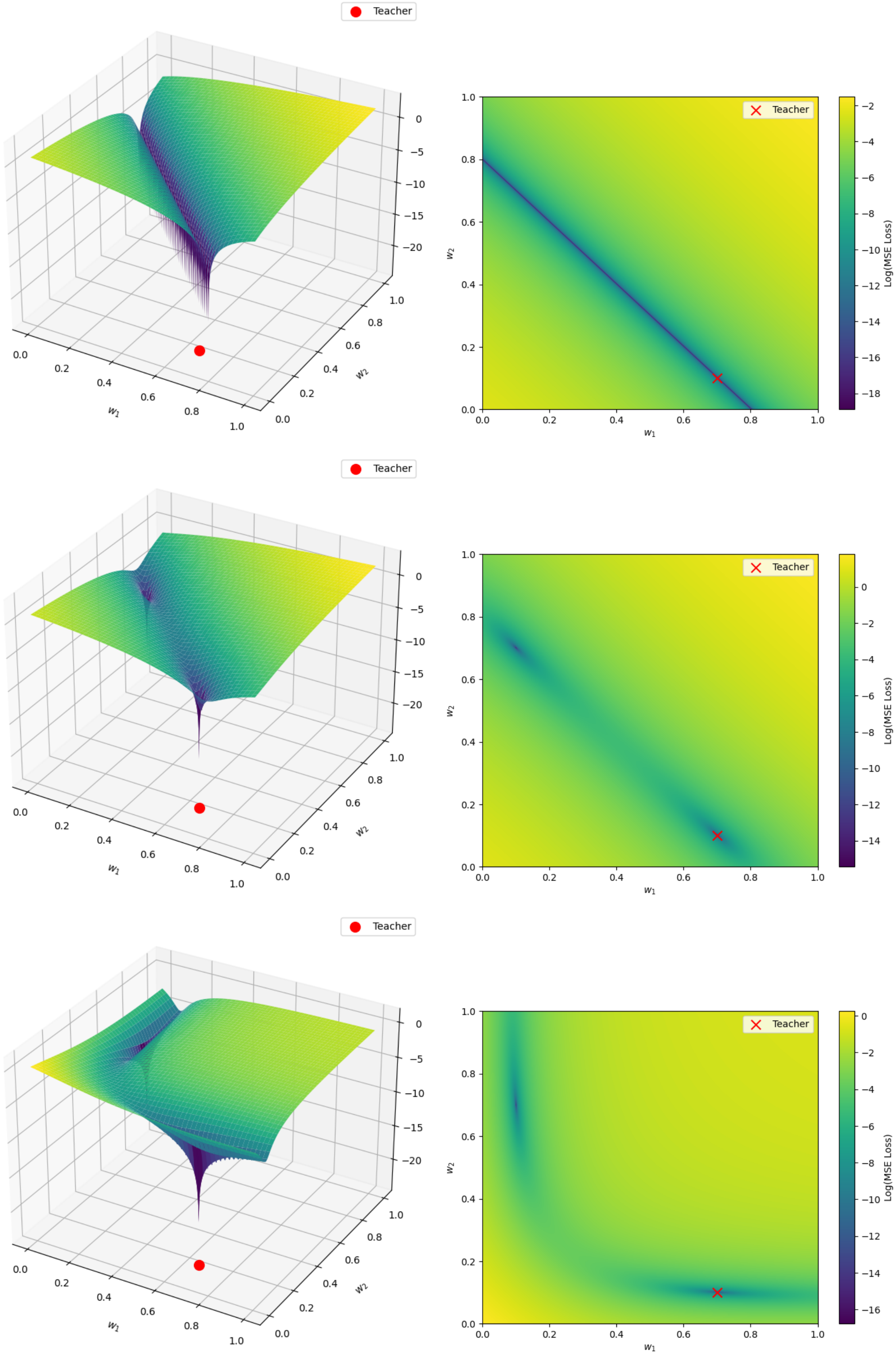


Figure 2.2: Logarithm of permutation symmetric mean squared error loss function of $f(x, \theta) = \sigma(w_1 x) + \sigma(w_2 x)$, where the teacher network had weights of $w_1^* = 0.7, w_2^* = 0.1$. The activation functions from top to bottom are: $\text{ReLU}(x)$, $\sigma_{\text{softplus}}(x) + \sigma_{\text{sigmoid}}(x)$ and $\tanh(x)$. Notice how the loss function is symmetric along the line $w_1 = w_2$, 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 piece-wise 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 $z_i^l = \sigma({\mathbf{w}_i^l}^T \mathbf{z}^{l-1})$. If we take $\mathbf{w}_i^{l'} := -\mathbf{w}_i^l$, we will get $z_i^{l'} = \sigma({\mathbf{w}_i^{l'}}^T \mathbf{z}^{l-1}) = -z_i^l$. 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 [18]. 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 how multiple optimal solutions arise due to the activation function being odd or even.

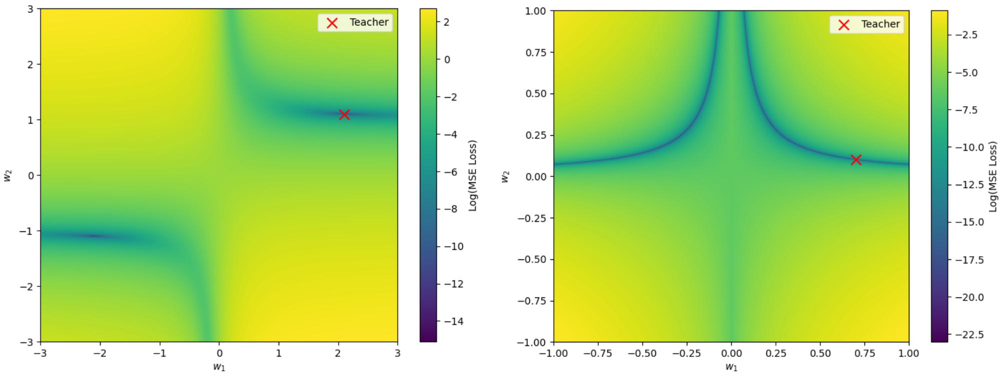


Figure 2.3: Logarithmic MSE loss of the odd function $f(x, \theta) = w_2 \tanh(w_1 x)$ on a dataset from a teacher with weights $w_1^* = 2.1$ and $w_2^* = 1.1$. Note that there are only two global minima, at $(2.1, 1.1)$ and $(-2.1, -1.1)$. *Right:* Logarithmic MSE loss of the even activation function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

Other types of functions which are piece-wise linear, have a different kind of invariance. These types of function include the widely used $\mathrm{ReLU}(x) = \max(x, 0)$ [23] and $\mathrm{LeakyReLU(x)} = \max(x, \epsilon x)$ activation functions, where $0 < \epsilon << 1$ [22]. These functions are scale invariant, meaning that for all $y \in \mathbb{R}^{++}$ the following holds:

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

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 $\frac{1}{y}\mathrm{ReLU}(yx)$ 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.

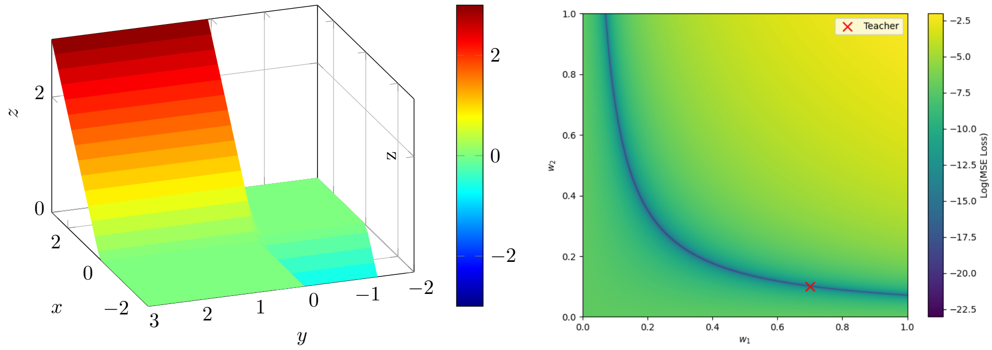


Figure 2.4: *Left:* Plot of the function $z = \frac{1}{y}\max(yx, 0)$, as can be seen, it is invariant along $y \in]0, \infty]$. *Right:* Logarithmic plot of the MSE loss of a ReLU network trained on a dataset from a teacher network, with optimal weights $w_1^* = 0.7, w_2^* = 0.1$, of the function $f(x, \theta^*) = w_2^*\mathrm{ReLU}(w_1^* x)$. 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 [38]. 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

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

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 $\theta_{r^*}^* \in \mathbb{R}^{\mathrm{ndim}(r^*)}$, 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** [38]. We will call networks with widths $m > r^*$ **overparameterized**.

Now we will introduce how to expand an irreducible neural network from width r^* into another with width $m > r^*$, as per definitions 3.2 and 3.3 in [38]. We need to have an expansion, such that $f(\mathbf{x}, \theta_m) = f(\mathbf{x}, \theta_{r^*}^*)$, for all $\mathbf{x} \in \mathbb{R}^{d_{\mathrm{in}}}$. 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 one is having b number of neurons cancel each other out.

Duplicated Neurons: Given a neuron which contributes to the next layer $\mathbf{a}\sigma(\mathbf{w}^T\mathbf{x})$, we can split the neuron into k duplicates of the neuron, which when combined, create the same behaviour. We need to 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 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**.

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

In figure 2.5, we can see how a neuron can be split into k neurons, such that they implement the same function.

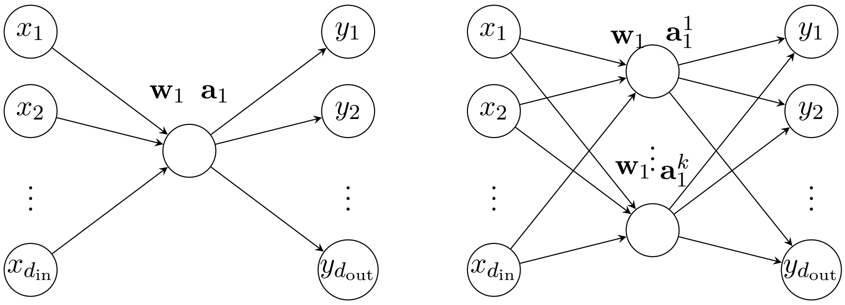


Figure 2.5: *Left:* A figure of a neuron, it contributes $\mathbf{a}_1\sigma(\mathbf{w}_1^T\mathbf{x})$ to the next layer. *Right:* If $\sum_{i=1}^{k} \mathbf{a}_1^k = \mathbf{a}_1$, we have k-duplicated neurons of the neuron on the left.

Zero-Type Neurons: 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, will contribute exactly $\mathbf{0}$ to the next layer, as seen in equation 2.12. In the rest of this work, 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.

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

In figure 2.6, we can see how there can be b number of neurons which do not contribute to the next layer. In the figure, and the rest of this work, we will denote the outgoing weights of zero-type neurons with α.

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

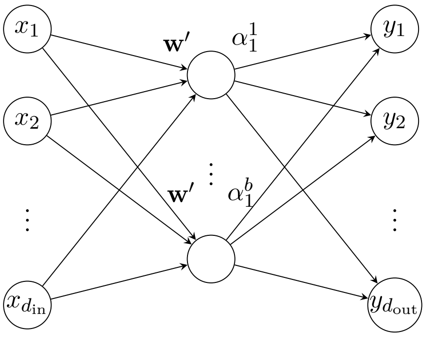


Figure 2.6: If $\sum_{i=1}^{b} \alpha_1^b = \mathbf{0}$, these neurons contribute exactly zero to the next layer, and we call them b-zero-type neurons.

as in the original network, 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 a whole subspace of optimal solutions in the parameter space.

In figure 2.7, we can see examples of global minima, where all points achieve zero loss. 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 set of parameters with zero loss achieve the same function as the optimal model, $\theta^*_{r^*}$.

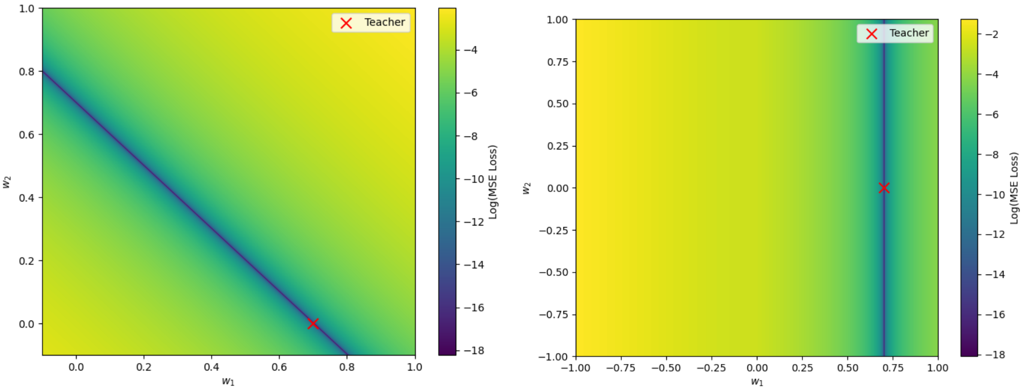


Figure 2.7: *Left:* Loss function evaluated on a dataset created from a teacher neural network of $f(x, \theta^*) = 0.7\text{ReLU}(0.7x)$. The plot is of the logarithm of the loss of a model $w_1\text{ReLU}(0.7x) + w_2\text{ReLU}(0.7x)$. *Right:* Loss function evaluated on a dataset from a teacher network of $f(x, \theta^*) = 0.7\text{ReLU}(0.7x)$. The plot is of the logarithm of the loss of model $0.7\text{ReLU}(w_1 x) + 0\text{ReLU}(w_2 x)$.

In Definition 3.2 of [38], for a given parameter space determined by m neurons they introduced the **affine subspace** $\Gamma_s(\theta^*_{r^*}) \subset \mathbb{R}^{\text{ndim}(m)}$ of an irreducible point $\theta^*_{r^*} \in \mathbb{R}^{\text{ndim}(r^*)}$. We too will define it below.

Definition 1. [38] Let $\theta^*_{r^*} = (\mathbf{w}_1^T, \dots, \mathbf{w}_{r^*}^T, \mathbf{a}_1^T, \dots, \mathbf{a}_{r^*}^T)^T$, $\theta^*_{r^*} \in \mathbb{R}^{\text{ndim}(r^*)}$ be an irreducible neural network, achieving zero loss. Given that we have m neurons, we can achieve the same function as $f(\mathbf{x}, \theta^*_{r^*})$, 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, $\Gamma_s(\theta^*_{r^*}) \subset \mathbb{R}^{\text{ndim}(m)}$ is determined by the number of duplicated neurons and zero-type neurons, which we will denote 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** $\Gamma_s(\theta^*_{r^*})$ is defined as

$$\Gamma_s(\theta^*_{r^*}) := \{\Big(\underbrace{\mathbf{w}_1^T, \ldots, \mathbf{w}_1^T}_{k_1}, \ldots, \underbrace{\mathbf{w}_{r^*}^T, \ldots, \mathbf{w}_{r^*}^T}_{k_{r^*}}, \underbrace{\mathbf{w}_1'^T, \ldots, \mathbf{w}_1'^T}_{b_1}, \ldots, \underbrace{\mathbf{w}_j'^T, \ldots, \mathbf{w}_j'^T}_{b_j},$$

$$\mathbf{a}_1^{1T}, \ldots, \mathbf{a}_1^{k_1 T}, \ldots, \mathbf{a}_{r^*}^{1\ T}, \ldots, \mathbf{a}_{r^*}^{k_{r^*} T}, \alpha_1^{1T}, \ldots, \alpha_1^{b_1 T}, \ldots, \alpha_j^{1T}, \ldots, \alpha_j^{b_j T}\Big)^T$$

$$|\forall t \in [r^*] : \sum_{i=1}^{k_i} \mathbf{a}_t^i = \mathbf{a}_t \text{ and } \forall t \in [j] : \sum_{i=1}^{b_i} \alpha_t^i = \mathbf{0}\}. \tag{2.13}$$

▪

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 set of zero loss parameterizations seen in figure 2.7, are an example of these affine subspaces.

The extension, by adding or splitting extra neurons, can be determined for a model architecture with a fixed width. The affine subspaces include the parameters of this extension into a higher dimension.

An example of a $\theta^*_{r^*} \in \mathbb{R}^6$ and all their affine subspaces, as defined in definition 1, is shown in figure 2.8.

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 affine subspace is present $m!$ times in the parameter space. We will denote the permutations of a subspace, $\mathcal{P}_\pi \Gamma_s(\theta^*_{r^*}) \subset \mathbb{R}^{\text{ndim}(m)}$, as in [38], by

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

In [38] 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.

For simplicity, we will introduce the set of all possible variation of k_i-duplicated and b_i-zero-type neurons that an expansion from a network with width r^* to a network with width m can cause. $\Upsilon_{r^* \to m} := \{(k_1, \ldots, k_{r^*}, b_1, \ldots, b_j) : k_i \geq 1, b_i \geq 1, \sum_{i=1}^{r^*} k_i + \sum_{i=1}^{j} b_i = m\}$.

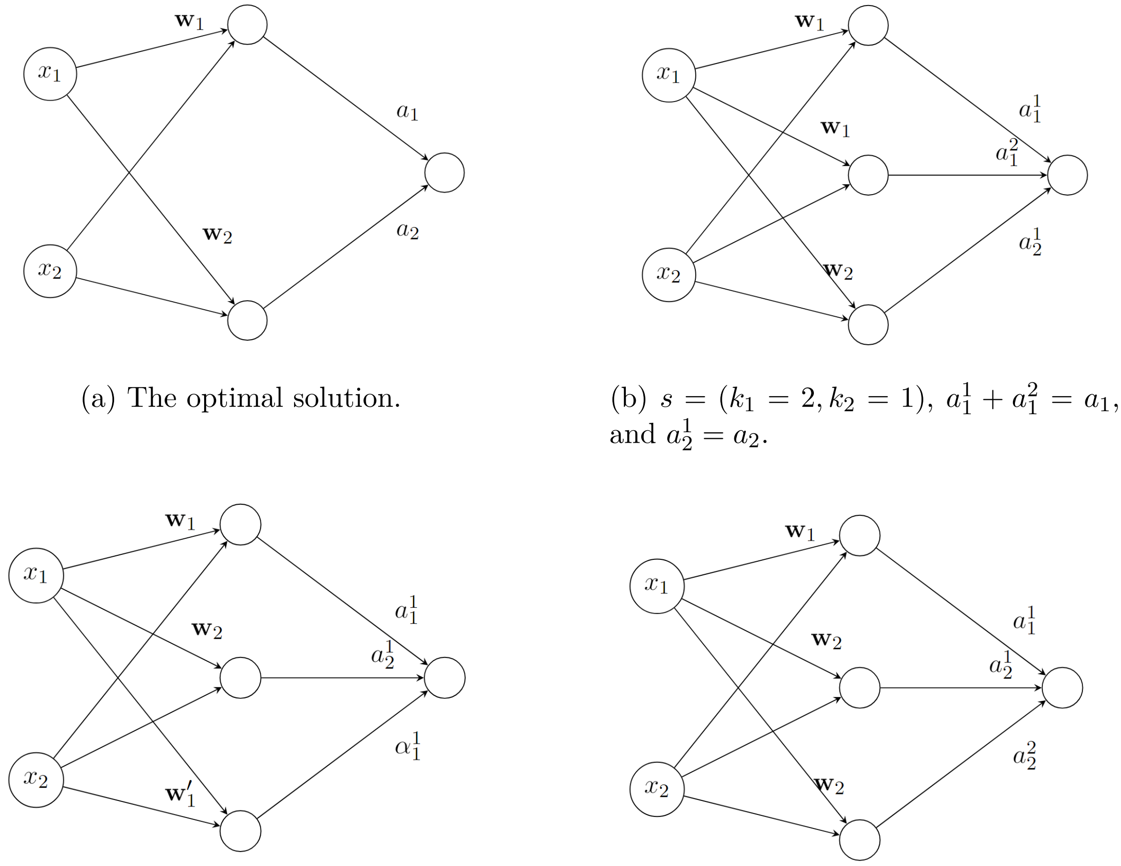
Definition 2. [38] The **expansion manifold**, $\Theta_{r^* \to m}(\theta^*_{r^*}) \subset \mathbb{R}^{\text{ndim}(m)}$, of an irreducible point $\theta^*_{r^*} \in \mathbb{R}^{\text{ndim}(r^*)}$ is

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

▪

Note that the expansion manifold is not necessarily 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 possible additions of



(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 2.8: An optimal solution $\theta_{r^*}^* \in \mathbb{R}^6$, where $r^* = 2$ and their three possible affine subspaces, $\Gamma_s(\theta_{r^*}^*) \subset \mathbb{R}^9$, for $m = 3$.

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. This is not necessarily an invariance in the function itself, certain restrictions need to apply to the dataset or neuron outputs.

Let us consider a dataset like MNIST [8]. It consists 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 an invariance can arise. Let us consider the scenario that a pixel in the top left corner is always black, i.e. $\forall \mathbf{x} \in \mathcal{D} : x_1 = 0$, 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 can happen for many activation functions, when $\mathbf{w}_i = \mathbf{0}$, which can occur when regularization techniques are used, such as L^1 [30] or L^0 [24] regularization. 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 by 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 2.9 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.

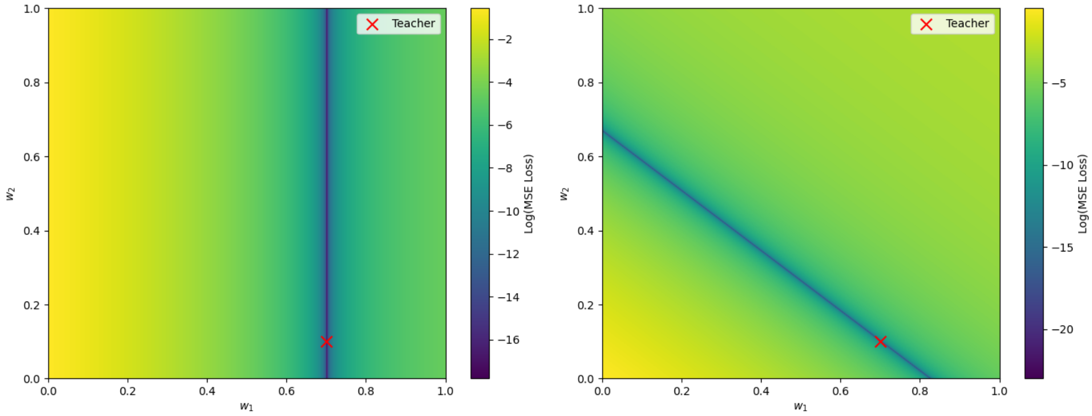


Figure 2.9: *Left:* The logarithmic loss of a neural network with the function of $f(\mathbf{x}, \theta) = \tanh(w_1 x_1 + w_2 x_2)$, where the input data for x_1 was generated using uniform distribution from $[-3, 3]$ and $x_2 = 0$. *Right:* The logarithmic loss of the same function but with the input 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. It can, however, partially explain some differences between real world data and our mathematical setup.

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 [2], [4], and [20]. 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 θ^A and θ^B.

2.3.1 Classical Mode Connectivity

Mode connectivity was first introduced in [13], they looked at if there exists a path in the parameter space, between two parameterizations, $\theta^A \in \mathbb{R}^P$ and $\theta^B \in \mathbb{R}^P$, such that along the path the increase in the loss is negligible or non-existent.

Definition 3. [20, 4] The **barrier** along a path, with arc-length parametrization, $\gamma : [0,1] \to \mathbb{R}^P$, where $\gamma(0) = \theta^B$ and $\gamma(1) = \theta^A$ 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. We will briefly mention the **barrier curve** between two parameterizations, which is the same as equation 2.16, without the $\max_{\lambda \in [0,1]}$.

Definition 4. [20] We say that θ^A and θ^B are **mode connected** if there exists a path $\gamma : [0,1] \to \mathbb{R}^P$, where $\gamma(0) = \theta^B$ and $\gamma(1) = \theta^A$, 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** [20], 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 [20], 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 [38], 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

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

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 [2], where they developed multiple algorithms for permuting neurons in models, in order to reduce the barrier between their linear combination. One of these, introduced in their paper, in section 3.2, is weight matching.

In figure 2.10 we can see an example of how the loss of the linear combination between two parameterizations changes with λ. Note that in this example the linear path is not the best path. We could take a curved one to achieve ϵ-mode connectivity, for a smaller ϵ.

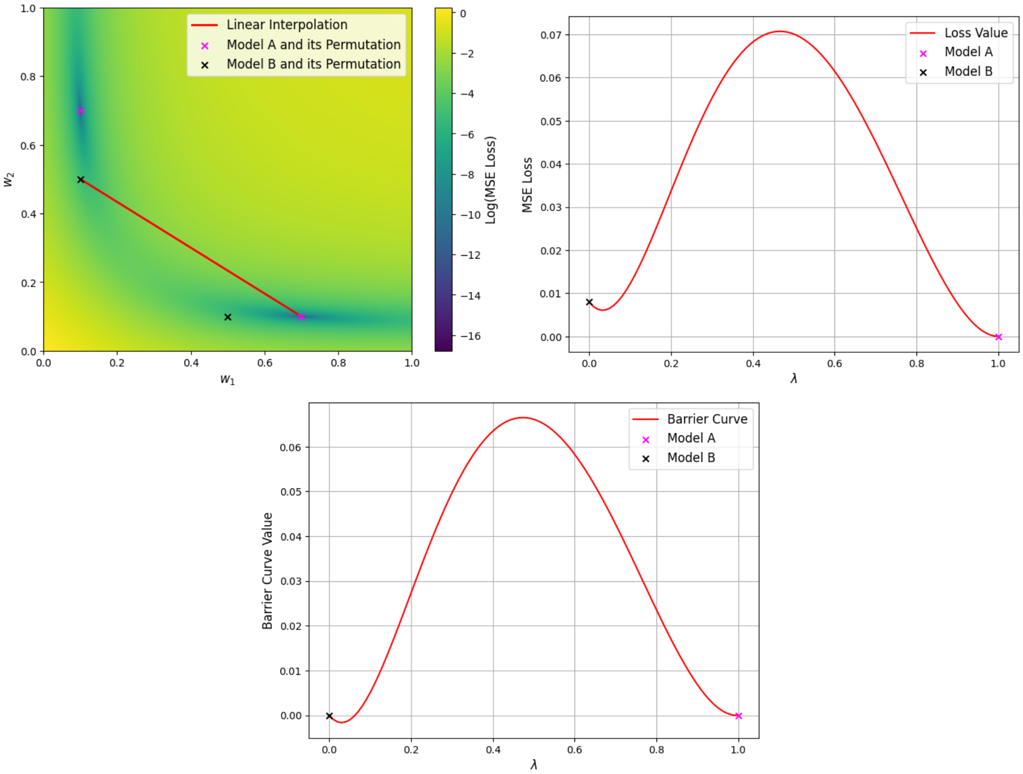


Figure 2.10: *Top Left:* The loss surface from figure 2.2. The red line represents the linear combination, γ, of the two parameterizations. *Top Right:* The value of the loss function along the path. *Bottom Middle:* The barrier curve between the two parameterizations. The maximum of the curve is the barrier (0.066), hence these two models are 0.066-linearly mode connected.

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 parameterizations in the parameter space, using neuron permutations. Note that, as stated in [2], 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 [2]. 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 the 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 2.11 we permuted one of our parameterizations from figure 2.10 to the other, which resulted in a smaller L^2 distance, and the two models to become linearly mode connected.

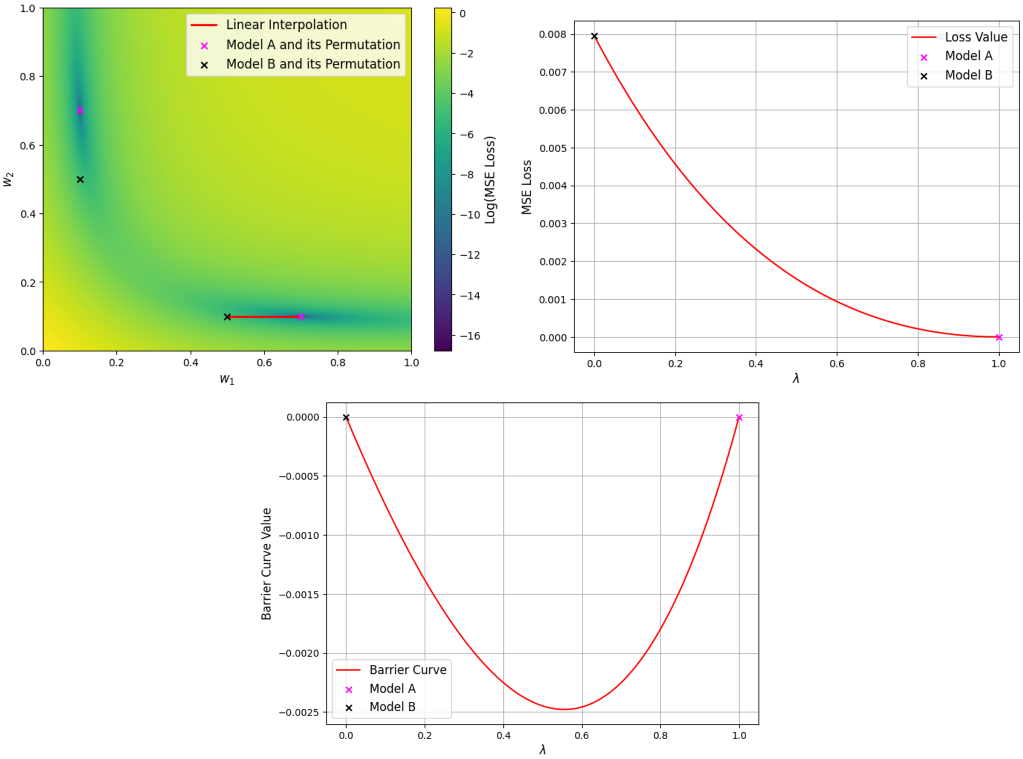


Figure 2.11: *Top Left:* The loss surface from figure 2.2. The red line represents the linear combination, γ, of the two parameterizations. *Top Right:* The value of the loss function along the path. *Bottom Middle:* The barrier curve between the two parameterizations. The maximum of the curve is the barrier (0.0), hence these two models are linearly mode connected.

In figure 2.12 we can see an example of two trained networks and the loss values between their interpolation before and after weight matching. Since the endpoints achieved zero loss, the curve is also the barrier curve.

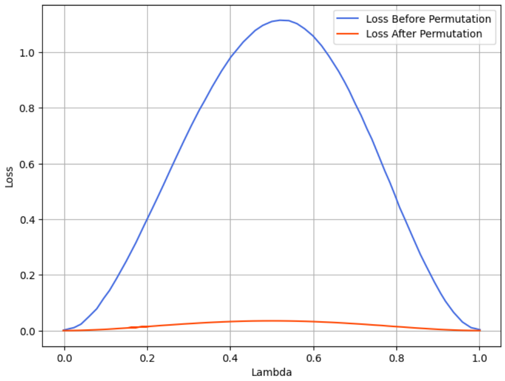


Figure 2.12: *Blue:* Loss values between the linear combination of two converged networks, with a barrier of 1.12. *Orange:* After weight matching the two models become 0.035-linearly mode connected.

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 [38]. 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 [38], 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 [29], 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 [3], the authors tried a more general approach. They looked at odd functions in both biasless, and biased neural networks, and developed the 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 [26], 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 $\sigma(x) = \sigma_{\text{soft}}(x) + \sigma_{\text{sigm}}(x)$, where $\sigma_{\text{soft}}(x) = \ln(1 + e^x)$ is the softplus and $\sigma_{\text{sigm}}(x) = \frac{1}{1+e^{-x}}$ 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 [38].

3.2 The Global Minima in the Expanded Network

Next we will discuss if the expansion of an irreducible network to a higher dimension contains all global minima points, i.e. $\forall \theta^*_m : \theta^*_m \in \Theta_{r^* \to m}(\theta^*_{r^*})$.

In [38, Theorem 4.2], the author shows that under strong conditions all global minima on $\mathcal{L}_\mu(\theta^*_m)$ are elements of the expansion manifold. These conditions are: up to permutation unique $\theta^*_{r^*}$; an activation function σ that is C^∞, $\sigma(0) \neq 0$; and for infinitely many odd and even n the n-th derivative of the loss function, $\sigma^{(n)}(0) \neq 0$.

The activation function $\sigma(x) = \sigma_{\text{soft}}(x) + \sigma_{\text{sigm}}(x)$ (figure 3.1) satisfies the criteria set up in the previous paragraph [38]. Hence we know that if we have an overparameterized neural network with zero loss, our model will be an element of one of the affine subspaces $\mathcal{P}_\pi \Gamma_s(\theta^*_{r^*})$.

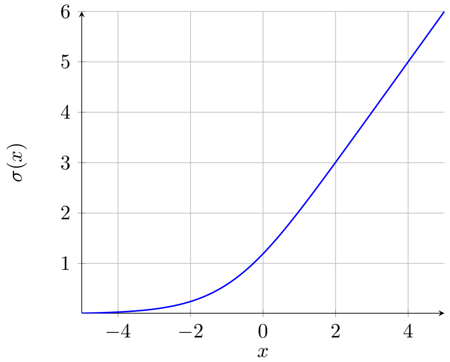


Figure 3.1: Plot of the activation function $\sigma(x) = \sigma_{\text{soft}}(x) + \sigma_{\text{sigm}}(x)$.

The authors of [38] 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 Zero Loss 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.

In our calculations, we will talk about the linear combination of two parameterizations, θ^A_m and θ^B_m, that are elements of the expansion manifold. For the weights of these models we will introduce a new notation. What was for one model, θ_m, $\mathbf{a}^j_i$, will now be denoted for θ^A_m as $[\mathbf{a}^A]^j_i$ and for θ^B_m as $[\mathbf{a}^B]^j_i$. The same convention will be used for the incoming and outgoing weights of zero-type neurons.

As a quick refresher, if we have an irreducible network, with parameters $\theta^*_{r^*} \in \mathbb{R}^{\text{ndim}(r^*)}$, for $m > r^*$, we have the affine subspaces $\mathcal{P}_\pi \Gamma_s(\theta^*_{r^*}) \subset \mathbb{R}^{\text{ndim}(m)}$, for which all elements produce the same function as the irreducible network. An affine subspace like that is determined by $s = (k_1, \dots, k_{r^*}, b_1, \dots, b_j)$, storing the information of how many times each neuron in our original network is duplicated (k_i), how many groups of zero-type neurons there are (j), and how many neurons cancel each other out in each zero-type group (b_i). A given s tuple needs to have for all i, $k_i \geq 1$, $b_i \geq 1$, and since we have m neurons in the extension $\sum_{i=1}^{r^*} k_i + \sum_{i=1}^{j} b_i = m$. The subspace is further determined by the order that the neurons take up (π).

In Theorem 1 we will first prove that two parameterizations from the same affine subspace $\Gamma_s(\theta^*_{r^*}) \subset \mathbb{R}^{\text{ndim}(m)}$, are linearly mode connected. An illustration of the theorem setup can be seen in figure 3.2. Note that while the coloring is the same for both networks, they are not neccessarily the same network, as both the duplicated and zero-type neurons can have different weight configurations.

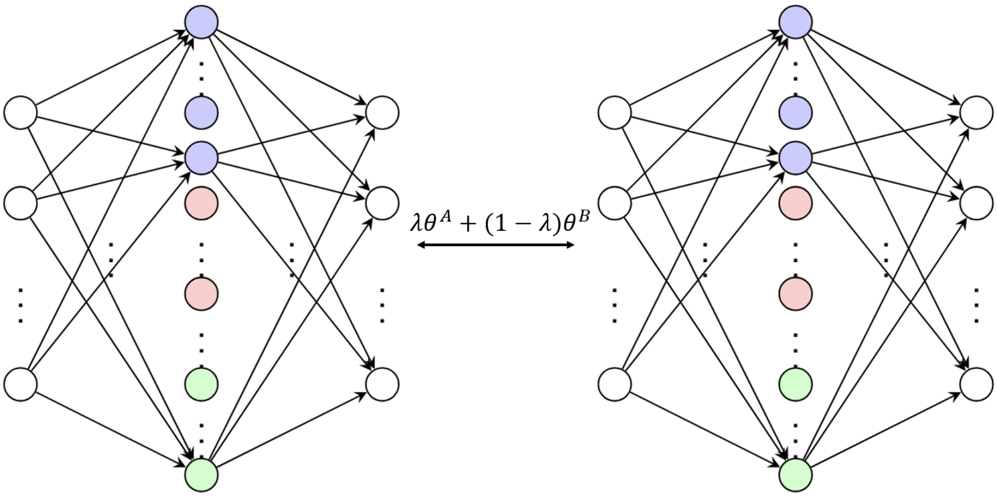


Figure 3.2: In this figure the colors of the neurons represent which group they belong to. For example, blue neurons would be considered duplicated neurons of $\mathbf{w}_1$, there being k_1 number of them. Our two parameterizations, θ^A (left) and θ^B (right) have the same number of dupicated neurons and zero-type neurons in the same order. We want to know what the loss of their linear combination is.

Theorem 1. Let us consider two parameterizations from the same affine subspace, $\theta^A_m, \theta^B_m \in \Gamma_s(\theta^*_{r^*})$, then their linear combination $\theta^\lambda_m = \lambda\theta^A_m + (1-\lambda)\theta^B_m$ is also in the subspace $\Gamma_s(\theta^*_{r^*})$, hence it has zero loss. ▪

Proof.

We will prove the theorem in multiple steps.

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

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

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

$$\begin{aligned}&\sum_{j=1}^{k_i} \left(\lambda[\mathbf{a}^A]_i^j + (1-\lambda)[\mathbf{a}^B]_i^j\right) \sigma \left((\lambda \mathbf{w}_i + (1-\lambda)\mathbf{w}_i)^T \mathbf{x}\right) \\ &= \sum_{j=1}^{k_i} \left(\lambda[\mathbf{a}^A]_i^j + (1-\lambda)[\mathbf{a}^B]_i^j\right) \sigma \left(\mathbf{w}_i^T \mathbf{x}\right)\end{aligned} \tag{3.2}$$

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

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

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

$$\begin{aligned}&\left(\sum_{j=1}^{k_i} \left(\lambda[\mathbf{a}^A]_i^j + (1-\lambda)[\mathbf{a}^B]_i^j\right)\right) \sigma \left(\mathbf{w}_i^T \mathbf{x}\right) \\ &= \left(\lambda \sum_{j=1}^{k_i}[\mathbf{a}^A]_i^j + (1-\lambda)\sum_{j=1}^{k_i}[\mathbf{a}^B]_i^j\right) \sigma \left(\mathbf{w}_i^T \mathbf{x}\right)\end{aligned} \tag{3.4}$$

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

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

Step 2: Next we will prove that the linear combination between two b-zero-type neurons is itself a b-zero-type neuron. We have $\sum_{j=1}^{b_i}[\alpha^A]_i^j\sigma\left({[\mathbf{w}^A]_i'}^T\mathbf{x}\right) = \mathbf{0} = \sum_{j=1}^{b_i}[\alpha^B]_i^j\sigma\left({[\mathbf{w}^B]_i'}^T\mathbf{x}\right)$. The linear combination of the two sub-models is:

$$\sum_{j=1}^{b_i} \left(\lambda[\alpha^A]_i^j + (1-\lambda)[\alpha^B]_i^j\right) \sigma \left(\left(\lambda[\mathbf{w}^A]_i' + (1-\lambda)[\mathbf{w}^B]_i'\right)^T \mathbf{x}\right) \tag{3.6}$$

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

$$\begin{aligned} &\sum_{j=1}^{b_i}\left(\lambda[\alpha^A]_i^j + (1-\lambda)[\alpha^B]_i^j\right)\sigma\left([\mathbf{w}^\lambda]_i'^T\mathbf{x}\right) \\ &= \left(\sum_{j=1}^{b_i}\left(\lambda[\alpha^A]_i^j + (1-\lambda)[\alpha^B]_i^j\right)\right)\sigma\left([\mathbf{w}^\lambda]_i'^T\mathbf{x}\right) \end{aligned} \tag{3.7}$$

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.

$$\begin{aligned} &\left(\sum_{j=1}^{b_i}\left(\lambda[\alpha^A]_i^j + (1-\lambda)[\alpha^B]_i^j\right)\right)\sigma\left([\mathbf{w}^\lambda]_i'^T\mathbf{x}\right) \\ &= \left(\lambda\sum_{j=1}^{b_i}[\alpha^A]_i^j + (1-\lambda)\sum_{j=1}^{b_i}[\alpha^B]_i^j\right)\sigma\left([\mathbf{w}^\lambda]_i'^T\mathbf{x}\right) \end{aligned} \tag{3.8}$$

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

$$\left(\lambda\sum_{j=1}^{b_i}[\alpha^A]_i^j + (1-\lambda)\sum_{j=1}^{b_i}[\alpha^B]_i^j\right)\sigma\left([\mathbf{w}^\lambda]_i'^T\mathbf{x}\right) = \mathbf{0}\sigma\left([\mathbf{w}^\lambda]_i'^T\mathbf{x}\right) = \mathbf{0} \tag{3.9}$$

Step 3: We are now ready to prove Theorem 1. We know from step 2 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 from step 1 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 linearly combined 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.

□

In Theorem 1 we looked at the linear combination of models in the same affine subspace. What if, however, the neurons are in a different order? In the following theorem we will see that if there are the same amount of k_i-duplicated neurons and b_i-duplicated neurons in both parameterizations θ_m^A and θ_m^B, but not necessarily in the order of s, but a permutation of it, i.e. $\theta_m^A \in \mathcal{P}_{(\pi_A)}\Gamma_s(\theta_{r^*}^*) \subset \mathbb{R}^{\mathrm{ndim}(m)}$ and $\theta_m^B \in \mathcal{P}_{(\pi_B)}\Gamma_s(\theta_{r^*}^*) \subset \mathbb{R}^{\mathrm{ndim}(m)}$, then there exists a permutation, where after applying the permutation to one of the parameterizations, they become linearly mode connected.

An example of two models from subspaces determined by the same s, but also by different permutations can be seen on figure 3.3.

Theorem 2. Let us consider two parameterizations that have the same number of k_i-duplicated neurons and b_i-duplicated neurons, but in different permutations, $\theta_m^A \in \mathcal{P}_{(\pi_A)}\Gamma_s(\theta_{r^*}^*)$, and $\theta_m^B \in \mathcal{P}_{(\pi_B)}\Gamma_s(\theta_{r^*}^*)$. After applying the permutation $\pi^* = \pi_A\left(\pi_B^{-1}\right)$ to

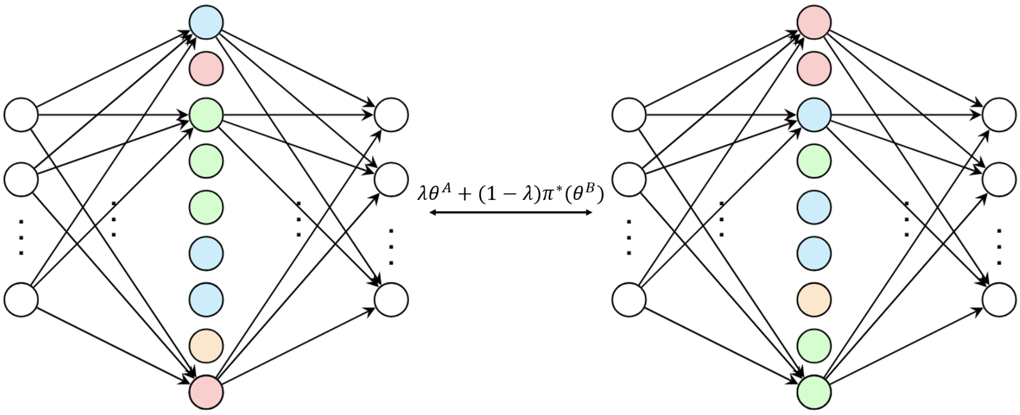


Figure 3.3: Here we can see an example of two networks that are in different permutations of an affine subspace determined by s, $\theta_m^A \in \mathcal{P}_{(\pi_A)}\Gamma_s(\theta_{r^*}^*)$ and $\theta_m^B \in \mathcal{P}_{(\pi_B)}\Gamma_s(\theta_{r^*}^*)$. We will show that we can permute one network to the other, so that they become linearly mode connected.

θ_m^B, the linear combination of the models, $\theta_m^\lambda = \lambda\theta_m^A + (1-\lambda)\pi^*(\theta_m^B)$ is in the subspace $\mathcal{P}_{(\pi_A)}\Gamma_s(\theta_{r^*}^*)$. ▪

Proof.

With the help of Theorem 1 proving this will not be difficult. Let us permute the neurons of θ_m^A according to π_A^{-1}, and also permute θ_m^B according to π_B^{-1}. We know that this way

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

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,

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

Let us now take the permutation π_A of everything. This will yield us that for all λ,

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

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

□

In the following section we will identify where the increase in the loss comes from, when linearly combining parameters. In Theorem 3 we will express the difference between the function created by the irreducible network, $f(\mathbf{x}, \theta_{r^*}^*)$ and the function created by the linear combination between the parameterizations seen in figure 3.4. These networks differ in that they are from two subspaces that are determined by two different s, i.e. they have a different structure with regards to duplicated and zero-type neurons.

Theorem 3. Let us have two parameterizations, $\theta_m^A \in \Gamma_s(\theta_{r^*}^*)$ and $\theta_m^B \in \Gamma_{s^B}(\theta_{r^*}^*)$, where $s = (k_1, k_2, \ldots, k_{r^*}, b_1, \ldots, b_j)$, and $s^B = (k_1^B, k_2^B, k_3 \ldots, k_{r^*}, b_1, \ldots, b_j)$, and $k_1^B = k_1 - 1$

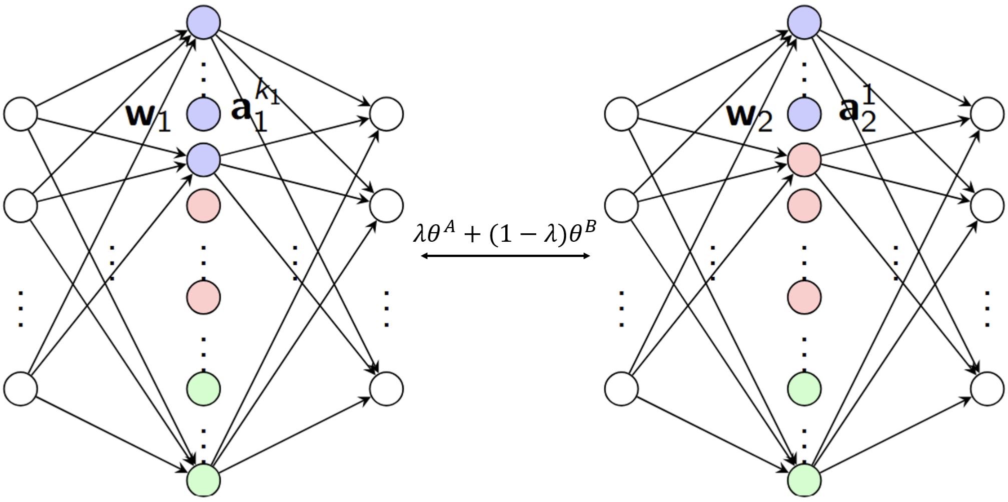


Figure 3.4: In this figure we have two networks, $\theta_m^A \in \Gamma_s(\theta_{r^*}^*)$ (left) and $\theta_m^B \in \Gamma_{s^B}(\theta_{r^*}^*)$ (right). The only difference between s and s^B is that in s the k_1-th neuron is a duplicated neuron of $\mathbf{w}_1$, while in s^B it is a duplicated neuron of $\mathbf{w}_2$.

and $k_2^B = k_2 + 1$, i.e. there is one neuron that cannot be matched with a neuron of the same type. The error between the function created by the irreducible network and the linear combination of the parameterizations, $f(\mathbf{x}, \theta^*) - f(\mathbf{x}, \theta_m^\lambda)$, where $\theta_m^\lambda = \lambda\theta_m^A + (1-\lambda)\theta_m^B$, is

$$\begin{aligned} f(\mathbf{x}, \theta^*) - f(\mathbf{x}, \theta_m^\lambda) = \lambda [\mathbf{a}^A]_1^{k_1} \left(\sigma(\mathbf{w}_1^T \mathbf{x}) - \sigma\left((\lambda \mathbf{w}_1 + (1-\lambda)\mathbf{w}_2)^T \mathbf{x} \right) \right) \\ + (1-\lambda)[\mathbf{a}^B]_2^1 \left(\sigma(\mathbf{w}_2^T \mathbf{x}) - \sigma\left((\lambda \mathbf{w}_1 + (1-\lambda)\mathbf{w}_2)^T \mathbf{x} \right) \right) \end{aligned} \tag{3.13}$$

▪

Proof.

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

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

For simplicity, let us introduce $\text{SubNetwork}_{p\to q}$ 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 s^B. For θ_m^A it can be expressed like so:

$$\text{SubNetwork}_{(k_1+k_2)\to m}^A = \sum_{t=3}^{r^*} \sum_{i=1}^{k_t} [\mathbf{a}^A]_t^i \sigma\left(\mathbf{w}_t^T \mathbf{x}\right) + \sum_{t=1}^{j} \sum_{i=1}^{b_t} [\alpha^A]_t^i \sigma\left([\mathbf{w}^A]_t'^T \mathbf{x}\right) \tag{3.15}$$

$$f(\mathbf{x}, \theta_m^A) = \sum_{i=1}^{k_1} [\mathbf{a}^A]_1^i \sigma\left(\mathbf{w}_1^T \mathbf{x}\right) + \sum_{i=1}^{k_2} [\mathbf{a}^A]_2^i \sigma\left(\mathbf{w}_2^T \mathbf{x}\right) + \text{SubNetwork}_{(k_1+k_2)\to m}^A \tag{3.16}$$

And for θ_m^B like so:

$$\text{SubNetwork}^B_{(k_1+k_2)\to m} = \sum_{t=3}^{r^*}\sum_{i=1}^{k_t}[\mathbf{a}^B]^i_t\sigma\left(\mathbf{w}_t^T\mathbf{x}\right) + \sum_{t=1}^{j}\sum_{i=1}^{b_t}[\alpha^B]^i_t\sigma\left([\mathbf{w}^B]'^T_t\mathbf{x}\right) \tag{3.17}$$

$$f(\mathbf{x},\theta^B_m) = \sum_{i=1}^{k_1^B}[\mathbf{a}^B]^i_1\sigma\left(\mathbf{w}_1^T\mathbf{x}\right) + \sum_{i=1}^{k_2^B}[\mathbf{a}^B]^i_2\sigma\left(\mathbf{w}_2^T\mathbf{x}\right) + \text{SubNetwork}^B_{(k_1+k_2)\to m} \tag{3.18}$$

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

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

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

$$\begin{aligned}
f(\mathbf{x},\theta^\lambda_m) = &\sum_{i=1}^{k_1-1}\left(\lambda[\mathbf{a}^A]^i_1 + (1-\lambda)[\mathbf{a}^B]^i_1\right)\sigma\left(\mathbf{w}_1^T\mathbf{x}\right)\\
&+\left(\lambda[\mathbf{a}^A]^{k_1}_1 + (1-\lambda)[\mathbf{a}^B]^1_2\right)\sigma\left((\lambda\mathbf{w}_1 + (1-\lambda)\mathbf{w}_2)^T\mathbf{x}\right)\\
&+\sum_{i=1}^{k_2}\left(\lambda[\mathbf{a}^A]^i_2 + (1-\lambda)[\mathbf{a}^B]^{(i+1)}_2\right)\sigma\left(\mathbf{w}_2^T\mathbf{x}\right)\\
&+\sum_{t=3}^{r^*}\sum_{i=1}^{k_t}\left(\lambda[\mathbf{a}^A]^i_t + (1-\lambda)[\mathbf{a}^B]^i_t\right)\sigma\left(\mathbf{w}_t^T\mathbf{x}\right)\\
&+\sum_{t=1}^{j}\sum_{i=1}^{b_t}\left(\lambda[\alpha^A]^i_t + (1-\lambda)[\alpha^B]^i_t\right)\sigma\left(\left(\lambda[\mathbf{w}^A]'_t + (1-\lambda)[\mathbf{w}^B]'_t\right)^T\mathbf{x}\right)
\end{aligned} \tag{3.20}$$

For the previous theorem we proved that the linear combination of b-zero-type neurons is itself an b-zero-type neuron. Therefore we know that the last line in equation 3.20 is zero. We further know that this is true for $\lambda f(\mathbf{x},\theta^A_m)$ and $(1-\lambda)f(\mathbf{x},\theta^B_m)$. The last part of this proof is to express $f(\mathbf{x},\theta^*) - f(\mathbf{x},\theta^\lambda_m)$. Let us substract equation 3.20 out of equation 3.19. 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:

$$\begin{aligned}
f(\mathbf{x},\theta^*) - f(\mathbf{x},\theta^\lambda_m) = &\lambda\left(\sum_{i=1}^{k_1}[\mathbf{a}^A]^i_1\sigma\left(\mathbf{w}_1^T\mathbf{x}\right) + \sum_{i=1}^{k_2}[\mathbf{a}^A]^i_2\sigma\left(\mathbf{w}_2^T\mathbf{x}\right)\right)\\
&+(1-\lambda)\left(\sum_{i=1}^{k_1-1}[\mathbf{a}^B]^i_1\sigma\left(\mathbf{w}_1^T\mathbf{x}\right) + \sum_{i=1}^{k_2+1}[\mathbf{a}^B]^i_2\sigma\left(\mathbf{w}_2^T\mathbf{x}\right)\right)\\
&-\sum_{i=1}^{k_1-1}\left(\lambda[\mathbf{a}^A]^i_1 + (1-\lambda)[\mathbf{a}^B]^i_1\right)\sigma\left(\mathbf{w}_1^T\mathbf{x}\right)\\
&-\left(\lambda[\mathbf{a}^A]^{k_1}_1 + (1-\lambda)[\mathbf{a}^B]^1_2\right)\sigma\left((\lambda\mathbf{w}_1 + (1-\lambda)\mathbf{w}_2)^T\mathbf{x}\right)\\
&-\sum_{i=1}^{k_2}\left(\lambda[\mathbf{a}^A]^i_2 + (1-\lambda)[\mathbf{a}^B]^{(i+1)}_2\right)\sigma\left(\mathbf{w}_2^T\mathbf{x}\right)
\end{aligned} \tag{3.21}$$

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 $\mathbf{a}_2^1$.

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

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

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

By reorganizing the multiplication by factoring out $[\mathbf{a}^A]_1^{k_1}$ and $[\mathbf{a}^B]_2^1$ 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:

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

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 $s = (k_1, k_2, \dots, k_{r^*}, b_1, \dots, b_j)$ is needed, and one where it is not. We will further discuss pruning and extension.

If we have knowledge of $s = (k_1, \dots, k_{r^*}, b_1, \dots, b_j)$, 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 s^A and s^B, we can extend them both to a new subspace, determined by s^C, in a higher dimension. A possible subspace to extend the models into is defined by

$$s^C = \left(\max(k_1^A, k_1^B), \dots, \max(k_{r^*}^A, k_{r^*}^B), \max(b_1^A, b_1^B), \dots, \max(b_j^A, b_j^B)\right). \tag{3.25}$$

To extend into this subspace we can add neurons with respective incoming weights, and zero outgoing weights. If $k_i^A < k_i^B$, we can add neurons to model A, $k_i^B - k_i^A$ 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 same affine subspace, up to permutation, where we have proven in Theorem 2 that, up to permutation, they are linearly mode connected. This type of extension brings in $\frac{|{s'}^A - {s'}^B|_1}{2}$ 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 outgoing 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

$$s^C = \left(k_1^A + k_1^B, \dots, k_{r^*}^A + k_{r^*}^B, b_1^A + b_1^B, \dots, b_j^A + b_j^B\right), \tag{3.26}$$

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 3.5 we show how we can represent the problem with a graph.

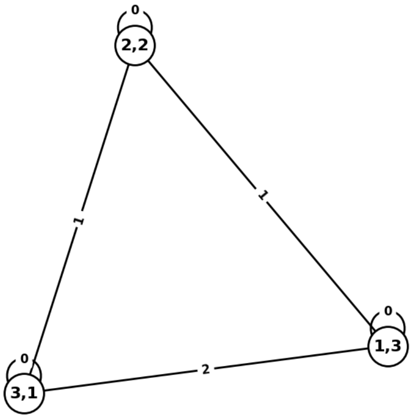


Figure 3.5: A representation of the number of mismatches as a graph. Here r^* is two and m is four. We represent $s = (1, 3)$, $s = (2, 2)$ and $s = (3, 1)$ as a node in the graph, and write the number of mismatches as weights between the nodes. We further know that each node in the graph has $\frac{m!}{k_1!...k_{r^*}!}$ number of affine subspaces inside it due to permutation [38]. The problem this way is reduced to calculating the number of edges for each weight.

In figure 3.6 we can see an example of the ratio of subspace pairs with a given number of neuron mismatches, to all subspace pairs. In figure 3.7 we show how the average number 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 do 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 an inverse-proportional 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.

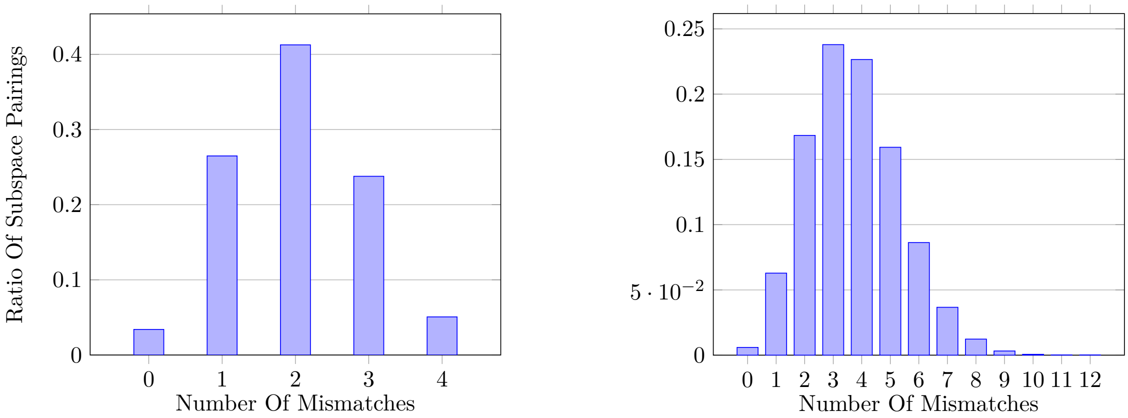


Figure 3.6: *Left:* The ratio of subspace pairings to the number of mismatches there are between them. Here $r^* = 4$ and $m = 8$. *Right:* The same ratio but with $m = 16$, notice how there are, on average, more mismatches, however they are more concentrated at smaller numbers.

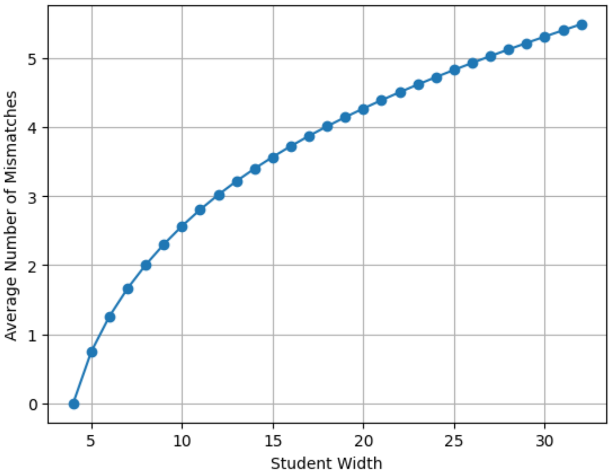


Figure 3.7: Using the data from figure 3.6, we can calculate the average number of mismatches between two subspaces for a given width. Here we calculated up to 32 neurons, and the minimal width is $r^* = 4$. The rate of change decreases, the more overparameterized the model.

Chapter 4

Experiments

4.1 Experimental Setup

In our setup we considered two layered neural networks, $f(\mathbf{x}, \theta) = \sum_i^m \mathbf{a}_i \sigma(\mathbf{w}_i^T \mathbf{x})$, where the activation function is $\sigma(x) = \sigma_{\text{soft}}(x) + \sigma_{\text{sigm}}(x)$. 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, $\theta^*_{r^*}$, 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 dataset, 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 dataset can be seen in figure 4.1.

This is a regression problem, for the loss function we used mean squared error, and took gradient descent steps with Adam optimizer [19], 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.

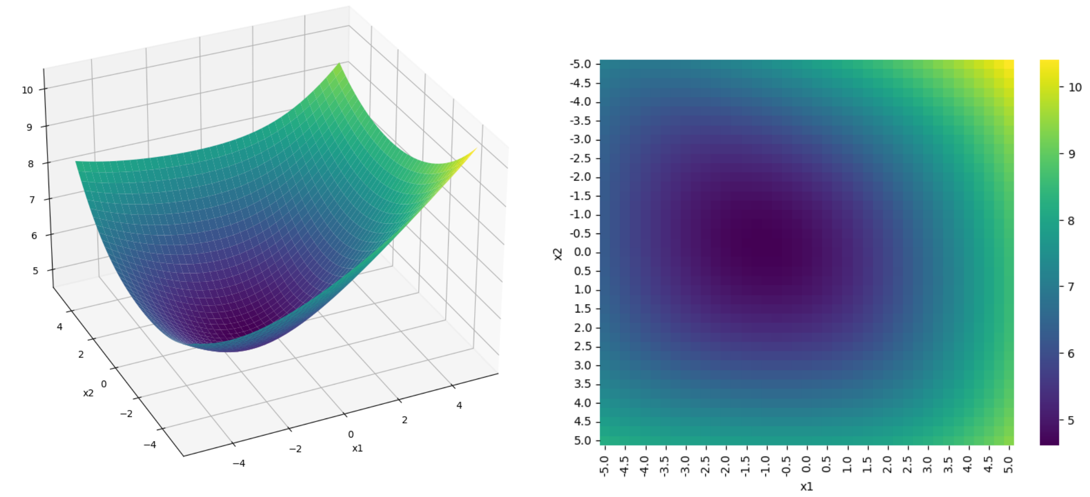


Figure 4.1: This is the function, which the neural network has to learn. The dataset consists of each grid-point that can be seen and their respective value, shown by color, and on the left figure.

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

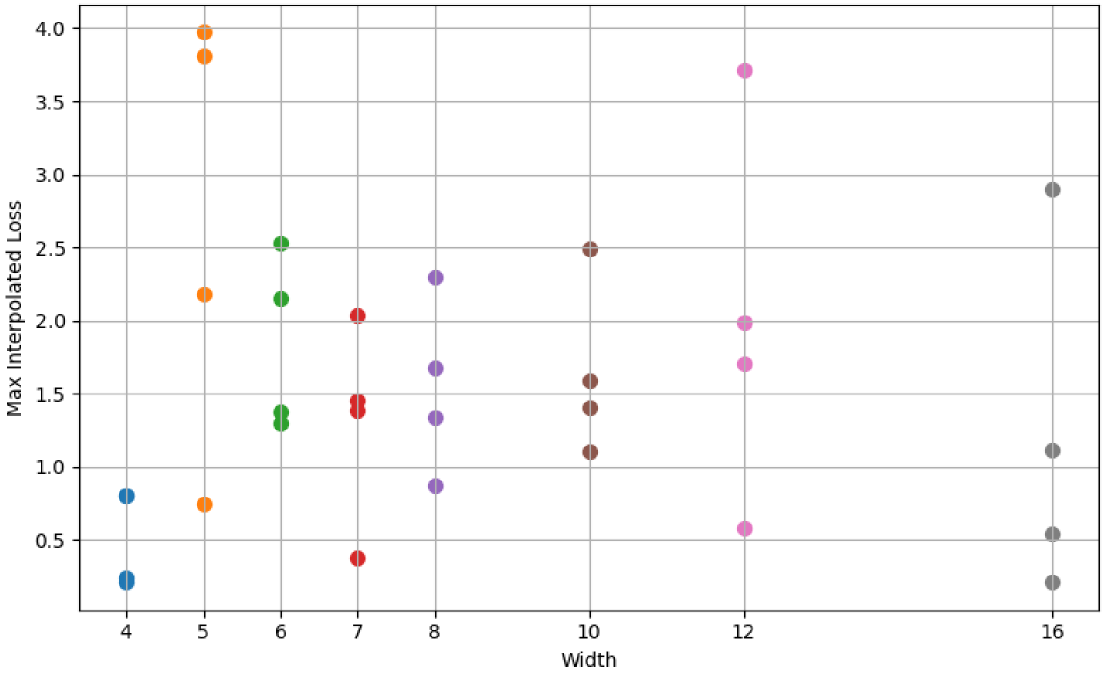


Figure 4.2: In this figure each dot represents the maximum of the loss value of the interpolated, trained, models. This is before weight matching.

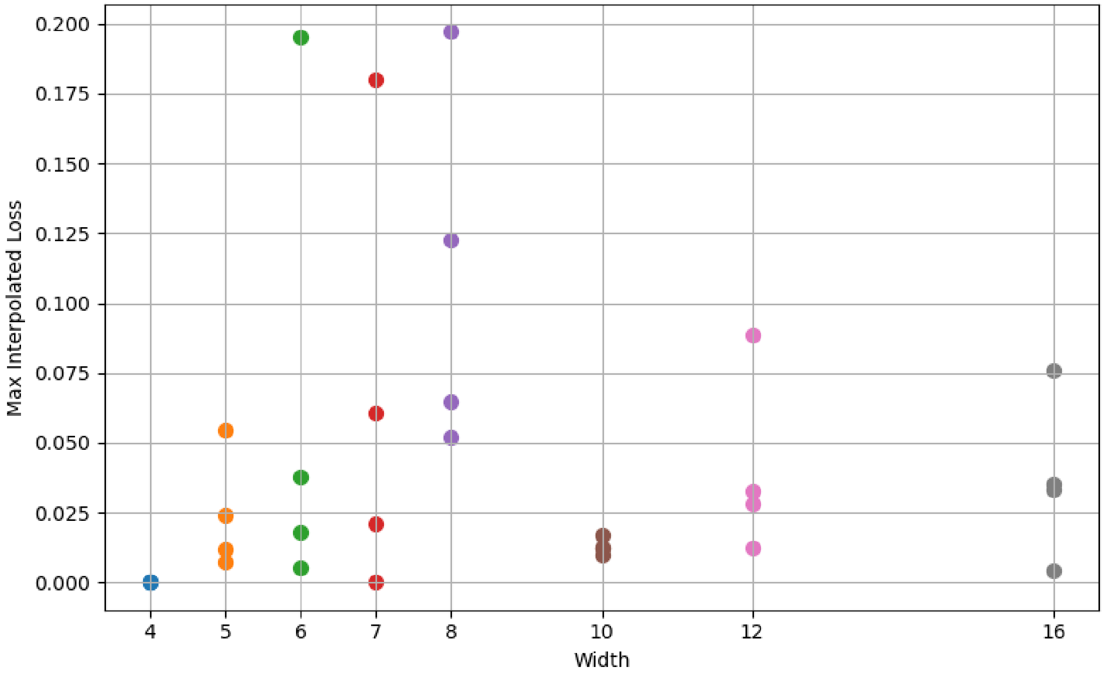


Figure 4.3: As can be seen on the scale of the y-axis, the barriers decreased by a lot after weight matching. However, there is still quite a bit of deviation between the barriers in each width. Generally speaking models from width 10 achieved smaller losses.

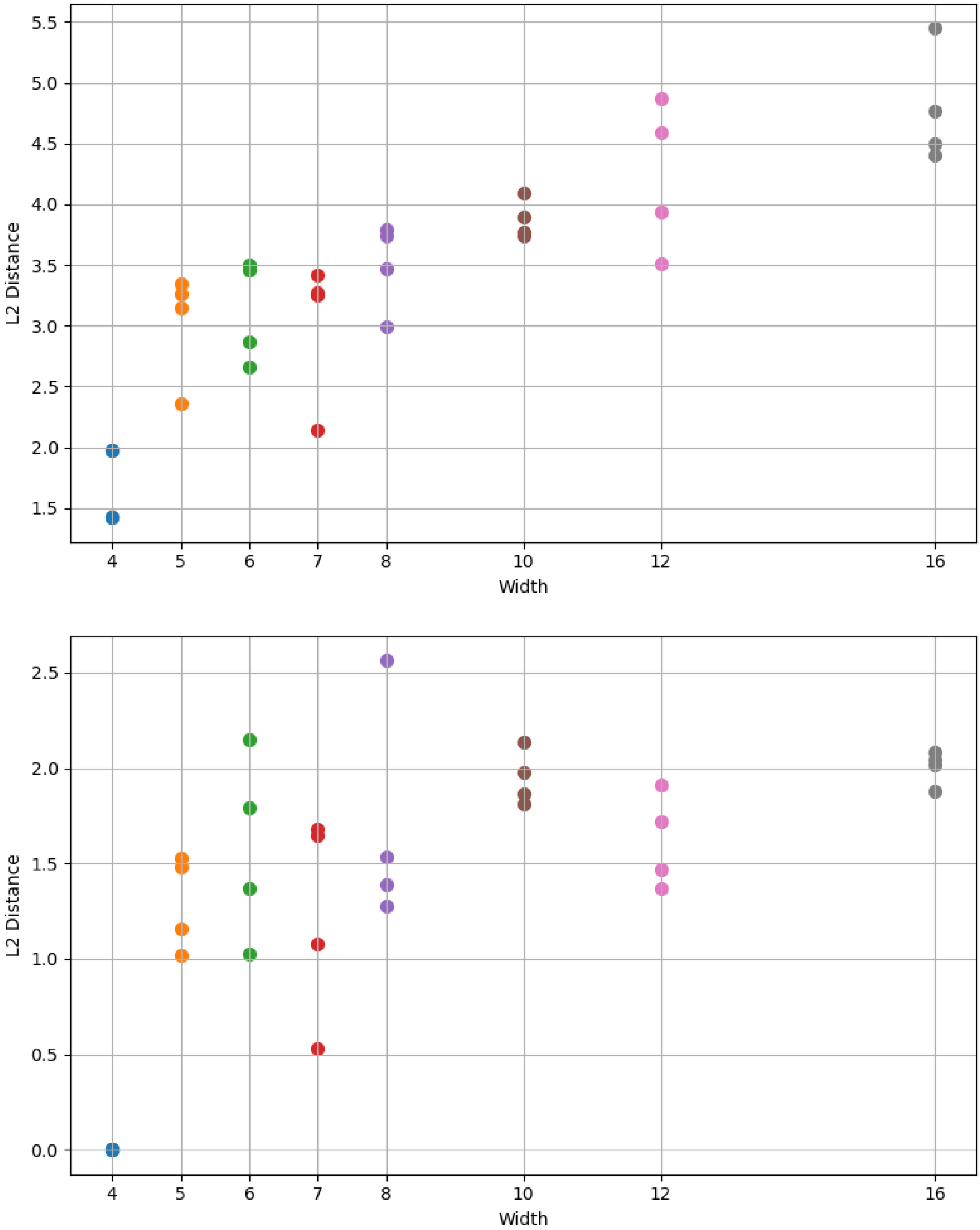


Figure 4.4: *Top:* L^2 distances between the above models before permutation. *Bottom:* L^2 distances between the above models after weight matching. The distance between models did not decrease, however, due to the increase in the parameter-space dimension this is not unsurprising.

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.

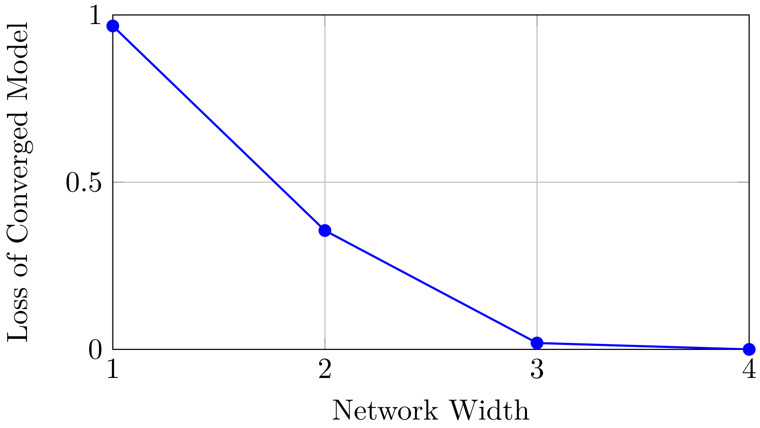


Figure 4.5: Models with less than four neurons cannot solve the regression problem. Even though the model with three neurons seems to have achieved quite well, the loss was still 0.019. This experiment empirically validates the minimality of $r^* = 4$.

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

In figure 4.7 we have another converged model and their permutation to the one shown in figure 4.6.

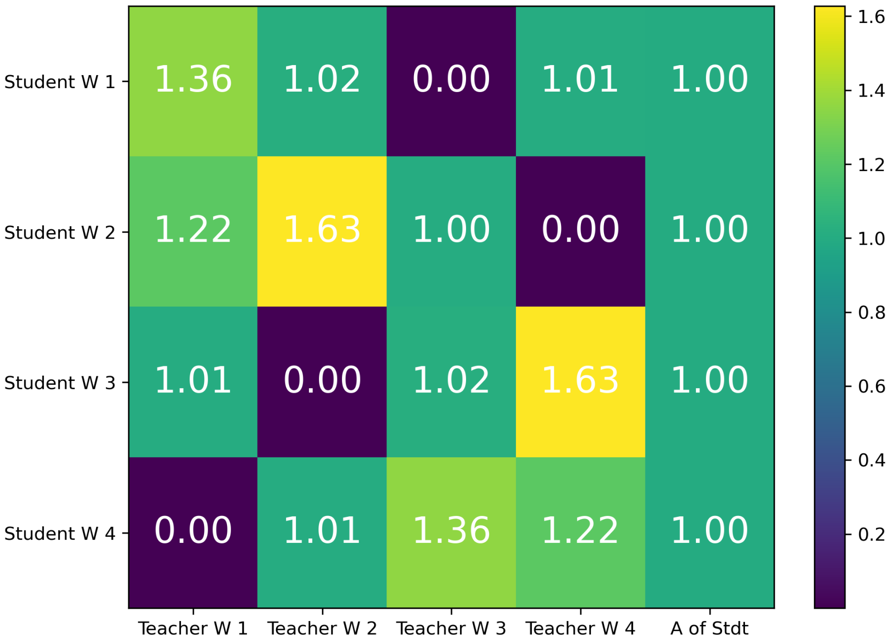


Figure 4.6: In this figure in the first $m \times r^*$ (4×4 in this case) matrix the L^2 distance between the teacher incoming weights, as defined in section 4.1, and student network incoming weights can be seen. The student network converged to the teacher weights, each row and column has one square, 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 $\theta_{r^*} = \mathcal{P}_\pi(\theta^*_{r^*})$, where $\pi = [3, 4, 2, 1]$.

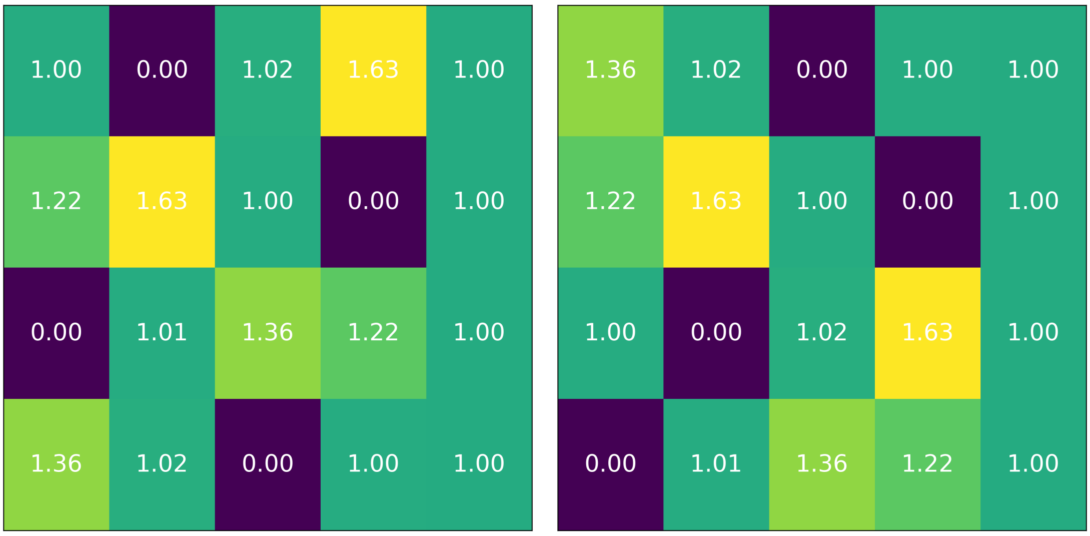


Figure 4.7: *Left:* Another optimal model, which converged to the same global minimum up to permutation. In this example $\theta_{r^*} = \mathcal{P}_\pi(\theta^*_{r^*})$, where $\pi = [2, 4, 1, 3]$. *Right:* The model after weight matching to the parameters in figure 4.6. Notice how it took up the same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

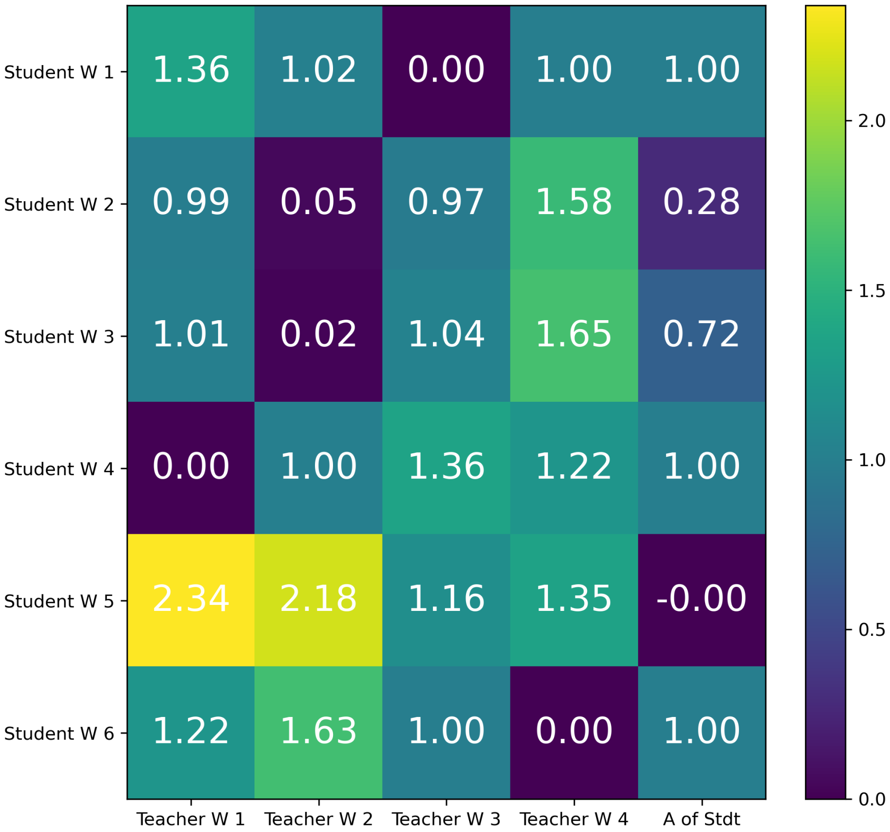


Figure 4.8: In this example a student network with six neurons was trained until convergence. Here weights one, four and six are copies of the teacher. We can see a duplicated neuron in the student weights two and three, where they converged to the weight of teacher neuron two, and their out weight sums up to one. We can also see a zero-type neuron which has weights that are not of the teacher, but also an outgoing weight of zero. In this example $\theta_m \in \mathcal{P}_\pi \Gamma_s(\theta^*_{r^*})$, 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 some 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 combining, 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 $\sigma(x) = \sigma_{\text{soft}}(x) + \sigma_{\text{sigm}}(x)$.

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] P.-A. Absil, R. Mahony, and R. Sepulchre. *Optimization Algorithms on Matrix Manifolds*. Princeton University Press, Princeton, NJ, 2008. ISBN 978-0-691-13298-3.

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

[3] Francesca Albertini, Eduardo Sontag, and Vincent Maillot. Uniqueness of weights for neural networks. In Richard J. Mammone, editor, *Artificial Neural Networks with Applications in Speech and Vision*, pages 113–125, 04 1993.

[4] 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`.

[5] Mikhail Belkin. Fit without fear: remarkable mathematical phenomena of deep learning through the prism of interpolation. *Acta Numerica*, 30:203–248, 2021. DOI: `10.1017/S0962492921000039`.

[6] Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, 116(32):15849–15854, 2019. DOI: `10.1073/pnas.1903070116`. URL `https://www.pnas.org/doi/abs/10.1073/pnas.1903070116`.

[7] 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`.

[8] Li Deng. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012. DOI: `10.1109/MSP.2012.2211477`.

[9] 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`.

[10] 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.

[11] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997. ISSN 0022-0000. DOI: https://doi.org/10.1006/jcss.1997.1504. URL https://www.sciencedirect.com/science/article/pii/S002200009791504X.

[12] Ken-Ichi Funahashi. On the approximate realization of continuous mappings by neural networks. *Neural Networks*, 2(3):183–192, 1989. ISSN 0893-6080. DOI: https://doi.org/10.1016/0893-6080(89)90003-8. URL https://www.sciencedirect.com/science/article/pii/0893608089900038.

[13] 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.

[14] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[15] 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)*, pages 770–778, 2016. DOI: 10.1109/CVPR.2016.90.

[16] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989. ISSN 0893-6080. DOI: https://doi.org/10.1016/0893-6080(89)90020-8. URL https://www.sciencedirect.com/science/article/pii/0893608089900208.

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

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

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

[20] 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.

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

[22] 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.

[23] 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.

[24] B. K. Natarajan. Sparse approximate solutions to linear systems. *SIAM Journal on Computing*, 24(2):227–234, 1995. DOI: 10.1137/S0097539792240406. URL https://doi.org/10.1137/S0097539792240406.

[25] Márk Penc and Bálint Daróczy. Symmetries and invariances in overparameterized neural networks, 2024.

[26] 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.

[27] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, 22(3):400–407, 1951. ISSN 00034851. URL http://www.jstor.org/stable/2236626.

[28] F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, 65:386–408, 1958. DOI: 10.1037/h0042519.

[29] 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.

[30] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, 58(1):267–288, 1996. ISSN 00359246. URL http://www.jstor.org/stable/2346178.

[31] V. Vapnik. *The Nature of Statistical Learning Theory*. Information Science and Statistics. Springer New York, 1999. ISBN 9780387987804. URL https://books.google.hu/books?id=sna9BaxVbj8C.

[32] V. N. Vapnik and A. Ya. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability & Its Applications*, 16(2):264–280, 1971. DOI: 10.1137/1116025. URL https://doi.org/10.1137/1116025.

[33] Vladimir Vapnik. On the uniform convergence of relative frequencies of events to their probabilities. In *Doklady Akademii Nauk USSR*, volume 181, pages 781–787, 1968.

[34] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[35] Bernard Widrow and Marcian E. Hoff. (1960) bernard widrow and marcian e. hoff, "adaptive switching circuits," 1960 ire wescon convention record, new york: Ire, pp.

96-104. In *Neurocomputing, Volume 1: Foundations of Research*. The MIT Press, 04 1988. ISBN 9780262267137. DOI: 10.7551/mitpress/4943.003.0012. URL https://doi.org/10.7551/mitpress/4943.003.0012.

[36] David H. Wolpert, editor. *The Mathematics of Generalization*. CRC Press, 1st edition, 1995. DOI: 10.1201/9780429492525.

[37] 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.

[38] 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.