Generalization in deep learning: Bilateral synergies in MLP learning

M.W. Theunissen

orcid.org / 0000-0002-7456-7769

Thesis accepted in fulfilment of the requirements for the degree Doctor of Philosophy in Computer and Electronic Engineering at the North-West University

Promoter: Prof. E. Barnard

Co-promoter: Prof. M. H. Davel

Graduation: August 2021

Student number: 22721339

Declaration

I, Marthinus Wilhelmus Theunissen hereby declare that the thesis entitled "Generalization in deep learning: Bilateral synergies in MLP learning" is my own original work and has not already been submitted to any other university or institution for examination.

M.W. Theunissen

Student number: 22721339

Signed on the 28^{th} day of June 2021 at Potchefstroom, North West, South Africa.

Acknowledgements

First, I would like to thank my supervisors, Prof. Etienne Barnard (EB) and Prof. Marelie H Davel (MD), for introducing me to the topic of generalization in machine learning and guiding me to become an ever improving academic researcher. I am especially thankful to EB for helping me see the forest for the trees. To MD I am thankful for planting the seeds (and growing some branches to be honest) of the early stages of this work and for her relentless critique of my technical writing.

To my parents, Marnus and Cathy Theunissen, I am thankful for supporting me in my decision to be a lifelong student, for helping me financially where needed, and feigning interest in my work.

I would also like to thank Ulrike Janke for managing me, taking a load of admin duties off my shoulders, and understanding why I drop off the face of the earth sometimes.

To Deon Knoetze I am thankful for the food and moral support while I wrote my thesis. A deceptively vital contribution to the completion of this document.

Finally, to Christine Knoetze. You have now seen me through two postgrad degrees. You keep me grounded, you keep me human, you keep me breathing. Without you there is no me (and, by extension, no thesis).

Abstract

We present an investigation of how simple artificial neural networks (specifically, feedforward networks with full connections between each successive pair of layers) generalize to out-of-sample data. By emphasizing the substructures formed within these networks we are able to shed light on several phenomena and relevant open questions in the literature. Specifically, we show that hidden units with piecewise linear activation functions are optimized on the train set in a distributed manner, meaning each subunit is only optimized to reduce the loss of a specific subpopulation of the train set. This mechanism gives rise to a type of modularity that is not often considered in investigations of artificial neural networks and generalization.

We are able to uncover informative regularity in subunit behavior and elucidate known phenomena such as: different artificial neural networks tend to prioritize similar samples, overparametization does not necessarily lead to poor generalization, artificial neural networks are able to interpolate large amounts of noise and still generalize appropriately, and generalization error as a function of representational capacity undergoes a second descent beyond the point of interpolation (a.k.a the double descent phenomenon).

We motivate a perspective of generalization in deep learning that is less focused on the complexity of hypothesis spaces, and looks to substructures and the manner by which training data is compartmentalized as a method of understanding the observed ability of these networks to generalize. This perspective contradicts classical ideas of generalization and complexity under certain conditions.

Keywords: *Deep learning, Generalization, Learning theory, Interpolation*

Contents

List of Figures

List of Tables

List of Algorithms

List of Acronyms

AI Artificial Intelligence

ML Machine Learning

DL Deep Learning

MLP Multilayer Perceptron

ANN Artificial Neural Network

DNN Deep Neural Network

SLT Statistical Learning Theory

ReLU Rectified Linear Unit

PCA Principle Component Analysis

MSE Mean Squared Error

VC Vapnik Chervonenkis

CNN Convolutional Neural Network

RNN Recurrent Neural Network

CE Cross Entropy

SGD Stochastic Gradient Descent

NLL Negative Log Likelihood

Notation

Unless stated otherwise, we consistently make use of the notation described here.

Table 0.1: Distinguishing between types of values.

Description	Notation	Example
a scalar	a lower-case letter	x
a vector	a bold lower-case letter	$\mathbf{x}$
a matrix	a bold upper-case letter	$\mathbf{X}$
a set	an upper-case letter	X

We make frequent references to specific subcomponents (weights, nodes, layers etc.) of a Multilayer Perceptron (MLP) architecture. For clarity we use the following simple system of subscripts and superscripts to distinguish subcomponents. Eq. 0.1 depicts how we refer to a specific weight parameter and Fig. 0.1 provides an accompanying illustration.

$$w^{s}_{l,j,i}. \tag{0.1}$$

Each subscript or superscript refers to a different component of an MLP or data, respectively. For this example, these are defined below:

- l refers to the l^{th} hidden layer in the network.
- j refers to the j^{th} node in layer l.
- i refers to the i^{th} node in the preceding layer $l-1$.
- s refers to a specific data sample.

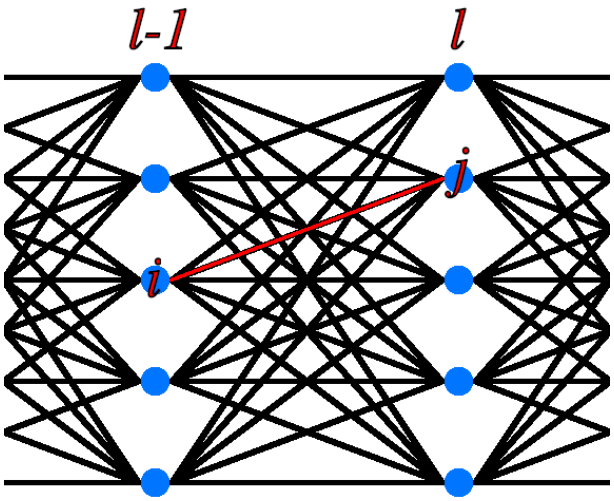

Figure 0.1: Illustration of the notation we use to refer to a parameter in an MLP. The weight parameter $w_{l,j,i}$, highlighted in red, corresponds to the subscripts used in Eq. 0.1.

In general, we refer to weights with the letter w, the inputs of a network with x, the predicted outputs with $\widehat{y}$, the label data with y, a sample with s, a class with c, a node pre-activation value with z, and a node post-activation value with h.

The following is a list of frequently used values:

- $\mathbf{h}_l$ refers to the activation pattern of hidden layer l.
- $\mathbf{W}_l$ refers to the weight matrix of layer l
- $h_{l,j}$ refers to the activation value at node j in layer l.
- $z_{l,j}$ refers to the pre-activation weighted sum at node j in layer l.
- S^c refers to the set of all samples belonging to class c.

Chapter 1

Introduction

In this chapter we introduce our subject matter and motivate its importance. We also clearly define our contributions and limit the scope of our empirical evidence.

1.1 Generalization in Deep Learning (DL)

acknowledge that methods that lend themselves a bit more to the symbolic (or interpretable) approach such as models based on Bayesian statistics, Markov models, and Decision trees also have a strong presence in the Artificial Intelligence (AI) community [1]. However, models related to the Artificial Neural Network (ANN) have become the *go-to* method for scenarios with vast amounts of data and complexity with little hope of maintaining interpretability.

DL systems form the backbone of modern Machine Learning (ML) applications that involve a high level of complexity and an abundance of data. These systems often include many different submodules which work together to form an end-to-end pipeline specifically designed for the specific task. One thing all DL systems have in common is their use of

ANNs. These architectures come with several advantageous characteristics such as good scalability to high-dimensional data, effective optimization, and strong generalization [2, 3], the last of which is the focus of this thesis. Generalization is the ability of an ML algorithm to perform well on data that it was not optimized for. It is a somewhat overloaded term. We, therefore, provide the following simple definition of the context we investigate it in:

> Given a set of training and testing samples of the form $(\mathbf{x} \in X, \mathbf{y} \in Y)$ sampled from some distribution $\mathbb{P}(X, Y)$. A learning algorithm produces a function $f : \mathbf{x}^{train} \rightarrow \mathbf{y}^{train}$. Generalization refers to $E(|f(\mathbf{x}^{test}) - \mathbf{y}^{test}|)$, where the superscripts, *train* and *test*, refer to whether the sample belongs to the train or test set, respectively.

The advantages of ANN models come at the cost of interpretability. These models use vector calculus in high-dimensional spaces which are not intuitively understandable. Without the ability to intuitively interpret an algorithm's decision rules, an alternative framework is used to reason about generalization. According to Statistical Learning Theory (SLT) [4], function fitting can maintain generalization by means of a bias-variance tradeoff. We discuss this principle in Section 2.2, but in short: A system with insufficient parametric flexibility is biased by those limitations, causing poor performance on both train and test sets, whereas a system with excessive flexibility will be highly variable across random variations (e.g. different drawings of the train set), leading to poor test set generalization. Proper generalization is achieved when the parametric flexibility is appropriate for the task complexity.

SLT was, and to a large extent still is, the prevailing paradigm with which to investigate generalization in ML systems. This can be seen in modern regularizing techniques (e.g. drop-out [5,6], early stopping, and weight decay [3]), which are often interpreted i.t.o their effect on model capacity. It can also be seen in the common theme of *inherent* complexity control in investigations of generalization [7–9]. Despite its common acceptance, capacity control has never been shown to be necessary for generalization, only sufficient [10]. It has also been noted that classic measures of complexity (e.g. Vapnik Chervonenkis (VC) dimension and Rademacher complexity [4]) are usually too conservative to be practically

useful [11, 12].

Contemporary practical results and targeted experimentation [13] have shown that Deep Neural Networks (DNNs) can exhibit a level of generalization that is not accounted for by classical SLT. In spite of their extreme representational capacity and expressive power, these models are able to generalize even in cases where the model has enough capacity to completely memorize random noise. These paradoxical results cast doubt on whether it is always important to balance bias and variance in an attempt to improve generalization. In addition to this, it has become a common intuition in practice to use overparameterized models, first, before regularizing accordingly. Classically, a model is considered overparameterized if it has more learnable parameters than training samples [14].

Finding an alternative or adjusted framework with which to characterize generalization is an ongoing topic of research in ML. By "characterize" we mean identifying the necessary properties to enable good generalization in general circumstances. Apart from limiting the hypothesis space, most modern approaches involve investigating the stability or robustness with which the model is able to make accurate predictions. Stability refers to being insensitive to variations in the datapoints from specific datasets, and robustness refers to being insensitive to variations in the entire input space [10]. One popular approach is to analyze the geometry of the loss landscape at the optimum [15–17]. Another approach involves approximating classification margins, which have been successfully used to predict generalization in simpler algorithms [18–20]. In Chapter 2 we will detail these various perspectives on characterizing generalization in DL.

Despite the abundance of perspectives and approaches none of them are able to conclusively capture the essential conditions necessary to allow ANNs to generalize. This gap in our theoretical understanding is of great practical significance. Without accurate and principled metrics to predict generalization, there are limitations on how well we can predict a model's performance on unseen data. We are forced to resort to imperfect heuristics and extensive (in many cases also expensive) empirical testing to ensure an acceptable level of generalization. In addition to this, a strong understanding of the properties governing generalization would also help guide the development of future algorithms.

1.2 Contributions

At a high level, the goal of this thesis is to motivate the idea that the structures formed within ANN architectures are an untapped resource in investigations of generalization and DL. By this we mean that the subcomponents (e.g. hidden layers, nodes, or weight vectors etc.), and the interaction among them, are often overlooked when trying to predict generalization. This idea is a departure from the classical notion of characterizing generalization, where a worst case approach is taken to determine the highest level of global complexity an algorithm can model. We assert that only looking at the mapping from input to output will never provide a complete perspective on the ability of overparameterized networks to generalize. It is very common for much of the network to only be applied to specific portions of the train set, and investigating these subcomponents (or groups of subcomponents) separately, provides enlightening insights about how generalization is allowed even if the global model contains a staggering amount of complexity.

We motivate our view with three overarching themes. The first two themes correspond to already published work. The work from the last theme is currently being prepared for publication.

1. **Subunits model subpopulations:** This theme is discussed in Chapters 3 and 4 and corresponds to [21,22]. Here we show that subunits (hidden nodes with piecewise linear activation functions) are only optimized to reduce the loss of specific sets of training samples, for which they activate. By tracking the activation patterns of these subunits we can estimate class specificity, uncover certain redundancies, and even make accurate predictions on test samples.

2. **Overparameterization allows noise isolation:** This theme is discussed in Chapter 5 and corresponds to [23,24]. Here we show that a result of the modular manner by which the train set is fitted, as described in point 1, is that certain kinds of noise can be prevented from affecting prediction values on test data without the same noise. We call this the *modular fitting* hypothesis and it is only possible with a large representational capacity.

3. **Sample priority affects generalization:** This theme is discussed in Chapter 6. Here we investigate the order in which samples are fitted and the resulting generalization at different training iterations. We find that the order by which training samples are fitted is fairly consistent across architectures and has a clear effect on generalization. Additionally, we speculate that the same modularity enabled by the use of subunit cooperation allows spurious functions that are fitted later in training from affecting those fitted earlier. We argue this is why the double descent phenomenon (see Section 2.4) exists.

The findings from each of these themes, together, motivate an alternative perspective of generalization and DL: one which emphasizes the empirical exploration of substructures and their relevance to subpopulations of the train set, instead of stringent theoretical analyses of hypothesis spaces. We hope that, by showing how these kinds of investigations can provide explanation for commonly observed phenomena, others will adopt this approach to further research and rethink generalization in DL; more on this in Chapter 7.

1.3 Scope of empirical results

The multitude of design choices and techniques available to practitioners, implementing DL systems, complicates attempts at developing a theoretical perspective that is general enough to be accurate across implementation domains and constraints. We have, therefore, limited our empirical results to a simple framework that (hopefully) maintains the most important aspects common to virtually all ANNs. We delimit this framework here, and provide motivation for its generality in Section 3.6. Unless stated otherwise, all experimental results have the following design choices:

- A fully connected feedforward network architecture (a.k.a an MLP).
- Every hidden node uses a Rectified Linear Unit (ReLU) activation function.

- Every hidden layer contains the same number of nodes.
- A ten class classification task is to be performed.
- Parameters are updated according to their gradients (calculated by backpropagation) w.r.t a loss value measured on a mini-batch of randomly selected training samples.

This framework is analytically convenient and uses fewer assumptions, regarding the data, than more specialized DL implementations. By that we mean that while Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) assume spatial and temporal structures, respectively, an MLP makes no such assumption. Despite its simplicity and lack of implicit inductive bias, standard MLPs are still able to exhibit remarkable generalization performance under extremely overparameterized conditions.

Throughout our empirical analyses we make use of four datasets, as described in Appendix C.2. Note that all four are image classification datasets. However, seeing as we flatten the image features prior to using them as input to the MLP, and considering the fact that MLPs are invariant to permutations of the input features (provided that the input features of all samples are permuted in the same way), we expect our findings to not be limited to grid-like image data.

1.4 Thesis Overview

In **Chapter 2** we discuss related work on the topic of generalization and DL. In **Chapter 3** we introduce a theoretical perspective of MLP training that highlights the manner by which subunits are optimized, both independently and in cooperation. This is supported by experimental results in **Chapter 4**. **Chapter 5** presents an investigation of overparameterized DNNs with specific focus on how subunits can isolate certain kinds of noise. In **Chapter 6** we take a closer look at the *double descent* phenomenon through the lens of subunit cooperation. In **Chapter 7** we recap and consolidate our findings to clearly emphasize the implications our theoretical perspective has for generalization in DL.

Chapter 2

Related work

In this chapter we discuss relevant works and paradigms found in modern approaches to characterizing generalization in DL. These concepts are necessary to interpret some of the novel work presented in subsequent chapters.

2.1 Overview

In Section 2.2 we start by explaining the theory and intuitions behind the bias-variance tradeoff and SLT. Section 2.3 discusses how data representation relates to generalization and how it is commonly investigated. A recent (*double descent*) phenomenon, observed when comparing generalization across various representational capacities, is discussed in Section 2.4. In the Section 2.5 we discuss research involving possible ways certain features are prioritized during training and how it relates to generalization.

2.2 The bias-variance tradeoff in ML

As mentioned in Section 1.1, SLT and the bias-variance tradeoff have become the prevailing paradigm in characterizing generalization. In this section we explain why, we highlight some fundamental principles, and present some shortcomings.

2.2.1 Intuition

The intuition behind using a bias-variance tradeoff to ensure generalization in ML is based on compactness. If an ML model is able to approximate an appropriate function with a small number of parameters (relative to the amount of training data) that function is more likely to be applicable to data outside of the train set. This intuition implicitly assumes that most naturally occurring functions we would like to approximate have simple underlying decision rules that the model should hopefully learn when the number of trainable parameters are limited.

There are analogous concepts in related fields. One example is data compression in information or signal theory. It has been shown that proper compression can lead to improved robustness and, consequently, better generalization [25]. Another example is assigning low likelihoods to events with complicated requirements to limit suspicious coincidences when using Bayesian predictive models [1, 26]. These intuitions strongly embody the theme of Occam's razor. This behavior has even been referred to as Occam's hill [27] due to poor performance observed for models that are too simple or too complex.

2.2.2 The U-curve

A common manifestation of the bias-variance tradeoff is the U-shaped curve that is typical of generalization error as a function of parametric model capacity. See Fig. 2.1 for an illustration. For models with insufficient capacity the performance on both the train and test set is poor. For models with too much capacity the performance on test data suffers.

There exists an optimal capacity where generalization is maximized.

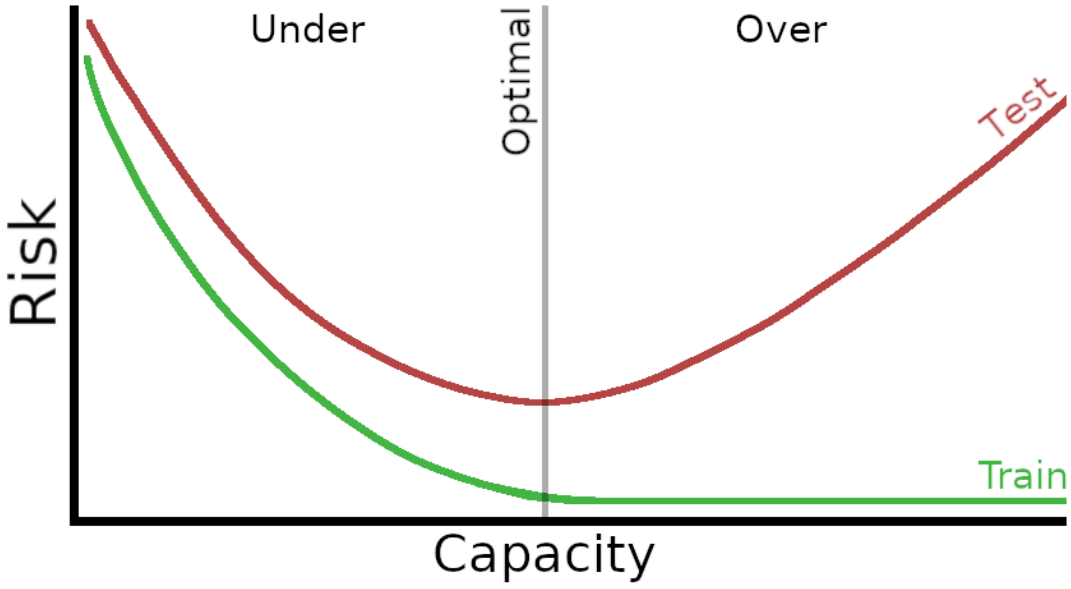

Figure 2.1: Illustration of the bias-variance U-curve. Risk and capacity are abstract terms used to refer to the amount of error in output predictions and parametric flexibility, respectively. Examples of risk include misclassification rate and MSE. Examples of capacity include the number of trainable parameters or the number of hidden units.

To the left of the optimal point models are in the *underfitting* regime where high bias results in erroneous predictions. This kind of error is a result of the models' inability to learn the appropriate function descriptors. With increased capacity bias reduces and variance increases, but the overall effect is improved performance. To the right of the optimal point models are in the *overfitting* regime where high variance results in erroneous predictions on the test set. A high variance means that the models fit complicated, and spurious, function descriptors that are unlikely to pertain to the task being performed [3]. These spurious function descriptors are often thought to be noise, outlier values, or inconsequential fluctuations in feature values.

A similar U-curve can be observed when considering the generalization error across training iterations at a constant representational capacity [28]. This leads to the notion of *effective capacity*: the representational capacity limited by imperfections in the optimization algorithm. While a model's representational capacity can be very high, its effective capacity is limited by many factors such as the optimization process or regularizing ef-

fects; implicit or explicit. With additional training iterations its effective capacity usually increases because it is capable of learning more complex function descriptors in order to minimize a cost function [2]. However, an exact measure of this effective capacity remains an unsolved problem in the literature.

2.2.3 Classic measures of complexity

There are many proposed measures of model capacity (or complexity) originating from SLT. These measures are typically defined by specific upper bounds on the type of functions a given model can fit. Using these measures of capacity, related bounds can be developed to guarantee a certain level of generalization. In other words, if the model is unable to approximate a function more complex than a certain threshold, the model is guaranteed to have a certain level of generalization. It is common to make these bounds dependent on parameter norms [29], model depth [30], or the number of parameters [31,32]. Many of these bounds are based on two well-known measures: VC dimension and Rademacher complexity. We will, therefore, be discussing these two measures next.

The **VC dimension** was first proposed in 1971 [33] but its practical use expanded greatly in the early 1990*s* [4]. It is a measure of the richness, complexity, expressiveness, diversity, flexibility, or capacity of the hypothesis space (all possible functions that can be approximated) of an ML model. Within the context of SLT and ANNs, all of these concepts are functionally equivalent. The prototypical concept of the VC dimension is defined for a parametric binary classification problem [3].

Let us say some learning algorithm F is to approximate a function $f(\mathbf{x}, \theta)$ that maps an input vector $\mathbf{x} \in \mathbb{R}^D$ to an output value $y \in \{0, 1\}$, using a set of training samples S. In this case we can say that the parameter set is $\theta \in \mathbf{\Theta}$, where $\mathbf{\Theta}$ is the hypothesis space of F. The VC dimension is defined as the maximum train set size $|S|$ for which there exists $\theta \in \mathbf{\Theta}$ that can *shatter* the input space. *Shattering* refers to being able to map the entire train set to any arbitrary labeling [34]. Formally we can define the VC dimension with Eq. 2.1.

$$\mathrm{VC}_{\mathrm{dim}} = \max_{|S|} \big(\forall f : S \to \{0,1\} \; \exists \; \theta \in \Theta\big). \tag{2.1}$$

At an increased VC dimension a system will be able to approximate a larger variety of complex functions which, intuitively, will result in a higher probability of finding solutions with poor generalization beyond the train set [11]. Being distribution independent, bounds developed from the VC dimension have the advantage of being generally applicable over any distribution of the data. However these bounds are based on a worst case scenario (shattering the input space) and so, they are often too conservative. Additionally, since they were developed for a binary classifier, creating bounds for more complex ML tasks require additional conditions [31].

The **Rademacher complexity** [35] is a distribution-dependent alternative metric that extends to real-valued and multi-class functions. If we keep the same scenario and notation we used to explain the VC dimension, the Rademacher complexity can be defined with Eq. 2.2.

$$\mathrm{RC} = E_F\big[E_S\big[\sup_{\theta\in\Theta}\big(\frac{1}{|S|}\sum_{s\in S}\sigma_s f(s,\theta)\big)\big]\big]. \tag{2.2}$$

where $\sigma_s \sim \mathcal{U}\{-1,1\}$. By using the supremum over the entire hypothesis space, we are effectively finding the function that would best correlate the Rademacher values (σ_s) and the model outputs. The inner expectation E_S then measures the ability of F to fit random noise w.r.t a specific set S. This is called the *empirical* Rademacher complexity. The outer expectation E_F measures the Rademacher complexity over all possible datasets drawn from the true underlying data distribution [36].

An increased Rademacher complexity indicates that the model is better able to fit random noise which indicates increased model capacity. Because the Rademacher complexity considers the actual distribution of the function space, bounds developed from this metric can often be more tight than the VC dimension. Finding the best correlation with random values (σ_s) also means that this measure is based on a worst case scenario, albeit within

the context of a specific distribution.

Metrics of this kind have provided a basis for developing many bounds on generalization, which have several factors in common. These are listed below:

- They are based on the intuition that limiting the complexity of the hypothesis space will lead to better generalization.
- They usually provide upper bounds based on worst case scenarios.
- Iterating over the entire hypothesis space of modern ML systems is often intractable which leads to limitations being placed on the hypothesis space within the context of specific domains in an attempt to produce non-vacuous bounds.
- In order to promote generality, these bounds often focus on higher level properties of function approximation with little consideration for the interaction among substructures which are ubiquitous in modern ML systems.

2.2.4 The apparent paradox

In spite of the works mentioned in the previous section, regarding capacity and generalization, no theoretically rigorous framework for the robust prediction of generalization error in a general setting has been developed. DNNs in particular have consistently produced results that are not predicted by strictly capacity-related generalization bounds and in many cases even contradict the intuitions suggested by these bounds. This is exemplified in the seminal work by Zhang et al. in 2016 [13], and re-released in 2021 [37]. By means of extensive experimentation with MLPs and CNNs (Inception net [38] and Alexnet [39]) they showed that:

- Common DNN architectures have enough representational capacity to easily fit various types of random noise.
- In spite of the extreme representational capacities these models can still perform very well on several natural datasets.

- Explicit regularization is not strictly necessary to enable good generalization, and even had little effect on the observed generalization ability.

These findings suggest a so-called "apparent paradox" [7, 10]. If these models have representational capacities large enough to memorize noise, how are their generalization performance on natural data not impeded by the obvious overparameterization as the bias-variance tradeoff suggests?

A generally applicable explanation for this behavior will doubtlessly lead to a deeper understanding of generalization in ML and its relationship with model capacity. This realization caused an explosion of research with the aim of either consolidating classical notions of generalization or proposing more relevant alternatives. Some of the more prominent examples of these works are discussed in the remaining sections of this chapter.

2.3 Data representation

Many recent works in the literature, regarding generalization and DL, adopt a representational view. The idea is that some attribute of the way DNNs represent training data allows them to generalize even when the parameters outnumber the training samples – sometimes by orders of magnitude. These approaches consider the data dependence of generalization more than typical capacity-based approaches do. In this section we discuss existing research with this viewpoint.

2.3.1 An example

It would be prudent to start with a simple concrete example. Refer to Fig. 2.2. In the example a binary classification task is to be performed on two-dimensional samples.

The solution in the center plot can be considered to be in the underfitting regime. It uses a single decision boundary and fails to correctly classify several samples in the train set.

This decision boundary can be implemented with a single hidden node and bias node. In this case every training sample is represented by a one-dimensional feature $h \in \mathbb{R}$, determined by how far to one side or the other the sample is located. It is then trivial to assign one class or the other based on which side of the decision boundary the sample falls.

If we use a combination of many decision boundaries (e.g. by means of many hidden nodes and non-linear activation functions) we can construct a decision boundary like the one presented in the right-hand plot. This is typical of an overfitted model. This model is able to correctly classify all the training samples but such a complicated decision boundary is not likely to be representative of the true underlying distribution of the training data. In this case every training sample is represented by a hidden feature vector $\mathbf{h} \in \mathbb{R}^D$, where D is the number of hidden nodes.

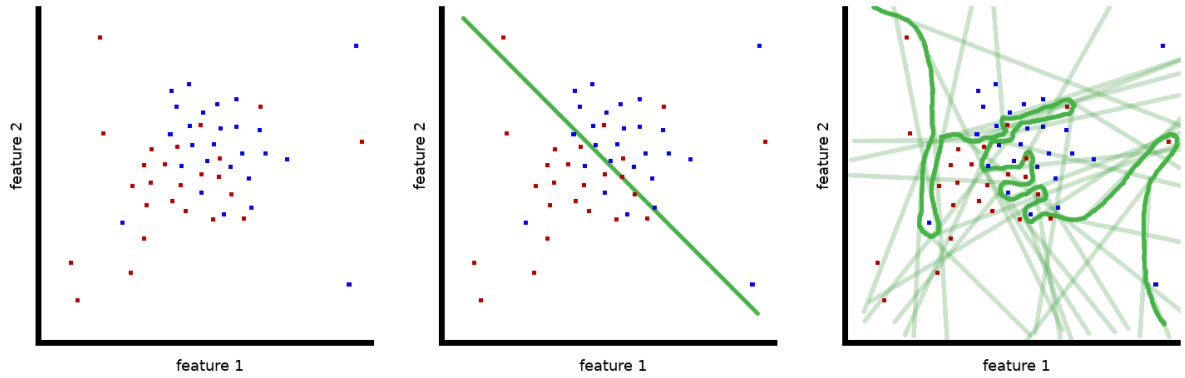

Figure 2.2: Illustration of the relationship between representation and capacity. The *left* plot shows a train set of samples of the form $\mathbf{x} \in \mathbb{R}^2$, with the colors representing which of two classes each sample belongs to. The *center* plot shows the same data with a single linear decision boundary. The *right* plot shows many decision boundaries.

From the perspective of classical SLT an optimal number of hidden nodes between these two extremes is necessary to ensure that the model will generalize. For low dimensional problems this intuition would probably (depending on the true underlying data distribution) be correct. However, more practical problems have thousands of input features and we tend to use even more hidden units, still. With such severe overparameterization we find that DNNs consistently find solutions that generalize well.

2.3.2 Distributed representation

One defining characteristic of ANN-like models is that they use a distributed representation [40] of data. This means that concepts (i.e. underlying features and patterns expressed in samples) are represented as patterns of activity that are distributed over many subunits instead of having a single unit for every concept. This also means that every subunit is related to many different concepts. Here, a subunit refers to some underlying computing element such as a neuron in a neural network (distributed representation) or a latent variable in a mixture model (local representation).

As a simple example, say we want to represent the state of a group of individuals' health. We can observe each individual and record whether they are healthy or not. This would be a local representation. Our representation would grow in size the more individuals we observe, and it would not directly help us determine the healthiness of unobserved individuals. Alternatively, we can represent each individual as a set of features, say: hair color, height, weight, and body temperature. A given individual is then represented as a vector of four values. This would be a distributed representation. The concepts relevant to each individual is distributed over four features. Immediately we can observe some similarities between individuals possibly relating to health.

The notion of distributed representations, while abstract, is strongly related to the work presented in this document. There are several advantages to using this type of representation. As discussed in [41], models using distributed representations have relatively low computational cost both during training and inference. This is because the patterns of activity are created by many interconnected subunits. With the focus being on updating the connections, not the concepts themselves, no sequential search is necessary to compare each subunit with every other. In our example we would be trying to determine the relationships between the features and health instead of comparing individuals directly. This allow these models to take advantage of parallel hardware.

These models also *naturally* generalize to novel inputs. An input value is not required to correspond exactly to an existing concept. The most related subunits will activate for

the new input and the resulting pattern of activity will represent a new concept that is created by many related concepts. For examples, if we observe a new individual that is very heavy and very short we might correctly conclude that the person is unhealthy even though we have never observed the individual before.

This leads to the third advantage: new (potentially informative) concepts can be created. Individuals that are very tall and very heavy might correlate with slightly elevated body temperature. This might indicate that the person is a male. Individuals that have gray hair and low body temperature might indicate old age. Both of these concepts are not included in the base features but can be deduced from how some features correlate and both can be useful in predicting overall health.

Distributed representations are usually thought of i.t.o word embeddings in natural language processing applications [42], however, it will become clear from later results in Chapters 3 through 5 that they are fundamentally linked to how DNNs generalize.

2.3.3 Classification margins

Refer back to the example in Fig. 2.2. If we focus on the model with the single decision boundary in the center plot, an intuitive measure of generalization that is not directly related to capacity is the expected distance between samples and the decision boundary. This distance is referred to as the *classification margin*. If samples tend to be located far away from the decision boundary it suggests more "certainty" w.r.t which side of the decision boundary those samples belong. Consequently the model should be less sensitive to small fluctuations in feature values. For such a simple linear model, calculating this distance is relatively simple and exact [43]. However, calculating this distance for DNNs is intractable. This is because of several factors such as the curse of dimensionality, non-linearities, and architectural structures.

Several works endeavor to approximate such classification margins for DNNs and then show, empirically, that maximizing the metric correlates with generalization. Notable approaches include: using a spectral norm of the Jacobian matrix [44]; linearization [18];

lower bounding the linear regions of training samples [19]; and using a first-order Taylor approximation [20]. These approaches show promising results under carefully constructed experimental conditions, however, a generally applicable measure of classification margin for DNNs and a causal link between such a metric and generalization has not been proven yet.

2.3.4 Loss landscape geometry

Another tool with which generalization is investigated in DL is the geometry of the loss landscape. By measuring the overall training loss of the model at various parameterizations around the solution found by the learning algorithm, we can estimate characteristics such as the sharpness, flatness, and curvature of the solution. The idea is that models with smooth and flat loss landscapes will generalize better. This idea is based on similar intuitions to the bias-variance tradeoff. Models that have low curvature in the loss landscape will require less information to describe, have low variance, and be less sensitive to small fluctuations in feature values [15].

Refer to Fig. 2.3 for an illustration. While all three presented models have similarly low train set loss at their respective solutions, as indicated by the dark region around their parameterization, we expect that they will have very different levels of generalization. The model represented in the left-hand plot will have the best generalization of the three because of a relatively gradual descent to, and wide basin around, the final minimum. The model in the center plot will have slightly poorer generalization because there is a steep descent and narrow basin. The model in the right-hand plot has a steep descent, a narrow basin, and high curvature in general which suggests very poor generalization ability.

A practical problem with this approach is that it suffers heavily from the curse of dimensionality. This means that it is difficult to obtain an unbiased and consistent perspective on the loss landscape in high dimensional parameter spaces, which is the case in virtually all practical DNNs. The surface is typically mapped with dimensionality reductions,

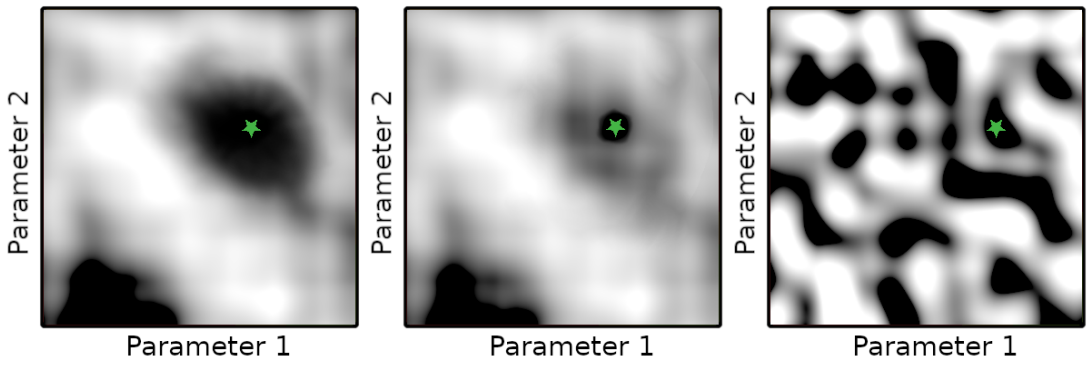

Figure 2.3: Illustration of several loss landscapes. The three plots show the loss landscapes of three two-parameter learning algorithms. Bright and dark regions indicate high and low train loss values, respectively. The green star icon indicates the resulting parameterization.

random searches [17], or heuristic searches [45]. A conceptual problem is that the loss value is easily manipulated by weight scaling. For example, Dinh et al. [46] showed that a minimum can be made arbitrarily sharp or flat with no effect on generalization, by exploiting simple symmetries in the weight scales of networks with rectified units.

2.4 Double descent

Recent research has demonstrated a curious phenomenon where parametric ML algorithms undergo two distinct phases of generalization behavior as a function of capacity. The phenomenon is called *double descent*, because in addition to the classical U-shaped curve (see Fig. 2.1) a second descent is observed in the test risk for models with capacities larger than what is necessary to interpolate the training data. It is of particular interest within the context of DNNs as it most clearly manifests for extremely large models. It was formally presented and defined in [47] but, to our knowledge, it was first noticed and reported on in [8]. See Fig. 2.4 for a visual illustration of the phenomenon.

Note that the double descent curve appears to encompass the classical U-shaped bias-variance curve. This suggests that it could prove very useful in bridging the gap between SLT and modern DL. This phenomenon, the second descent in particular, garners a lot of attention, [48–51], due to its implications for our understanding of generalization and

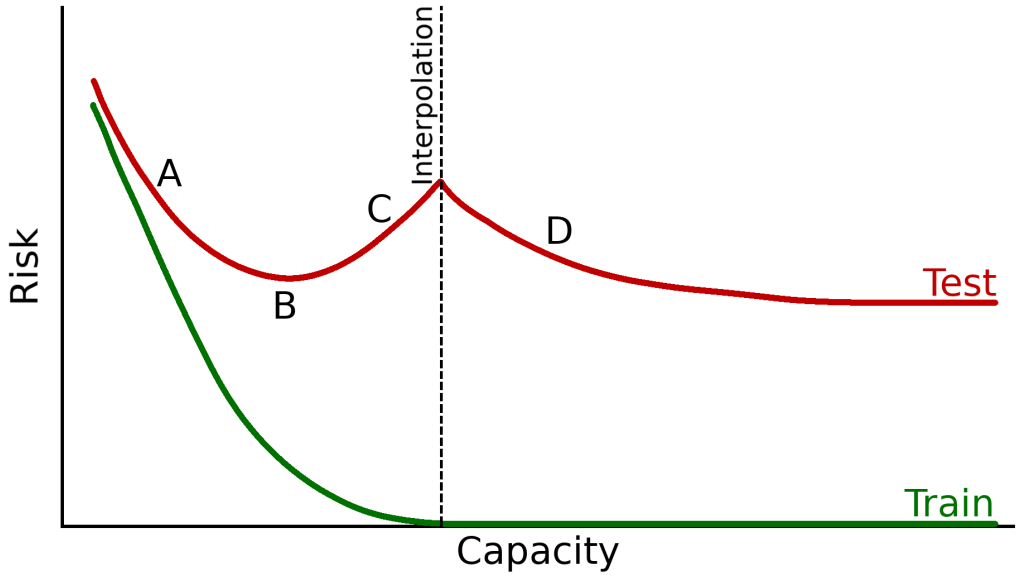

Figure 2.4: Illustration of double descent. A) Classic underfitting regime, B) Classic optimal bias-variance tradeoff, C) Classic overfitting regime, D) Second descent.

model capacity. In a similar way that Zhang et al. [13] drew attention to the fact that most DL models are large enough to memorize noise and still generalize, Belkin et al. [47] highlighted the fact that, in general circumstances, increased parametric flexibility can even improve performance. These findings are an even stronger contradiction to the idea of balancing bias and variance, which increases the necessity of finding a theoretically principled framework to describe, and potentially predict, generalization in DL.

Despite the clear connection with generalization and the bias-variance tradeoff, no generally accepted account of the origin of the double descent phenomenon exists. Some propositions include: the notion of *effective model complexity*, of which double descent is a function [48]; suggesting that with increased capacity both bias and variance decrease [52]; or a phase transition similar to the jamming transition of granular media that occurs around the critical point [53]. In Chapter 6 we present an alternative proposition based on extensive empirical investigation.

As a final note, we want to point out that double descent is also observed as a function of training iterations [48, 50]. This suggests that it is related to the *effective capacity* described in Section 2.2.2 and not just the representational capacity. These two mani-

festations are clearly related, however, we can investigate them separately. We refer to double descent as a function of representational capacity as *model-wise* double descent and double descent as a function of training iterations as *epoch-wise* double descent.

2.5 Feature and sample priority

2.5.1 Feature priority

The kind of features that are prioritized during the training of a DNN has been the topic of several works. In [9], using Fourier analyses, it was proposed that DNNs have a *spectral bias* that causes lower frequency functions to be fitted before higher frequencies. These lower frequency functions are more robust to perturbations which is connected to improved generalization.

In [54] a "stiffness" measure is proposed to estimate whether the features, being learned, are likely to generalize. This measure is based on how much parameter updates in favor of one sample would affect the loss of other samples, with positive stiffness values indicating that parameter updates reduce the loss of several samples simultaneously and are more likely to generalize. They found that stiffness values for within-class samples tend to be higher early in training when generalization is still improving.

2.5.2 Sample priority

In [55] it was empirically shown that ANNs fit training samples in a very similar order. The phenomenon seems to be most prominent when the models share a specific architecture, however it was found to still exist even when comparing models with different architectures. Their results indicated that stronger architectures tend to learn the same samples a less powerful model would, before continuing to fit additional samples. It was also shown that certain artificially manipulated datasets, such as randomly shuffling the training labels [13], prevented the phenomenon. This lead the authors to speculate that

the phenomenon is a result of how ANNs discover structure in natural datasets.

In [56] it was found that measuring the variance of per-sample gradient updates during training is an effective way to estimate the difficulty of a training sample. This idea explicitly assumes that the gradients i.t.o easier samples will stabilize early and the variance of gradients for those samples will be less. In other words: more difficult samples are expected to be fitted later.

In our work we refer to samples that tend to require more parameter updates and more parametric flexibility to fit as *difficult* samples (by definition). In Section 6.3 we go into more detail on this topic. Here, we clarify this definition by stating that some samples tend to be fitted later in training. We consider these samples more difficult to fit than those that are fitted earlier. The underlying mechanism that causes these samples to be fitted later relates to the number of similar samples present in the train set. Therefore, the difficulty of a sample is not related to its own input feature structure in isolation, but rather it relates to its feature structure w.r.t the others in the train set.

2.5.3 Coherent gradients

Updating parameters according to the gradients of a cost function w.r.t each parameter is currently the optimization method of choice for the vast majority of modern DL systems. Many variations of gradient descent exist in the literature, however, the fundamental principle remains the same. This lead to the *coherent gradient hypothesis* as a method of accounting for strong generalization exhibited by ANNs trained on natural datasets [57]. This hypothesis can be summarized as follows:

1. Training samples that have similar input features produce similar gradients.
2. Seeing as the global cost is usually added over many samples the gradients from similar samples will overlap producing stronger gradients in certain directions of parameter space.
3. Optimizing parameters in this way will lead to faster updates in directions where

per-sample gradients overlap.

4. The overall effect is that parameters are updated to simultaneously reduce the cost of many samples as opposed to a few.

Much in the theme of algorithmic stability [58], the obtained representation for the train set distribution does not overly rely on specific samples, and will therefore generalize better.

2.6 Discussion

While a full review of all the approaches and perspectives of generalization and DL currently being researched is impractical, we have highlighted some notable ones. We have also focused on those that are most relevant to the work presented in this thesis. The bias-variance tradeoff (discussed in Section 2.2) provides principled guidelines to facilitate the study of generalization in ML. We have discussed its shortcomings in the modern ML landscape, with a focus on ANNs. In the following chapters we hope to show how classical measures, such as VC dimension, Rademacher complexity, and their derivatives, are unable to capture the nuances of subunit interactions and their effect on generalization.

In Section 2.3 we discussed the way ANNs represent training data and the possible effect on generalization. This is inherently connected to finding structure in the training data. In Chapters 3, 4, and 5 we show how such structures can be found and modularized to facilitate good generalization in an overparameterized and noisy setting. In Section 2.4 we discussed the double descent phenomenon; and in Section 2.5 we mentioned several other phenomena regarding the way features are fitted and samples are prioritized in ANNs. While these phenomena seem quite unrelated to the double descent phenomenon, in Chapter 6 we show how the way features are prioritized (in addition to the modularity we propose in Chapter 5) results in the double descent phenomenon as a natural conclusion.

Chapter 3

Subunit cooperation

In this chapter we discuss a viewpoint of MLP learning, through gradient-based learning, that is useful to explain a novel perspective on generalization in DL models.

3.1 Overview

This chapter contains the bulk of our theoretical motivations. In Section 3.2 we thoroughly define the MLP architecture, explain the general training procedure of such a model, and clearly identify exactly how we train our experimental models. This information is not novel, however it provides necessary context to interpret our practical results and understand the novel contributions in the following sections.

In Section 3.3 we derive equations to show that subunits are trained to fit measurably similar samples using *two systems*, one continuous and one discrete. This novel perspective of fitting subpopulations of the train set in a distributed manner by the interplay between a continuous and discrete system of behaviors is utilized throughout the rest of the study.

Then in Section 3.4 we explain how each hidden layer produces a feature space using the

mechanisms introduced in Section 3.3. Section 3.5 defines and highlights the significance of *sample sets*, which will be used to develop informative metrics for much of the following chapters. In Section 3.6 we present a discussion on how applicable the proposed view of subunit cooperation is to more specialized ANN models.

3.2 The multilayer perceptron

We begin by defining an MLP. This is a basic ANN architecture that consists of a set of sequential hidden layers followed by an output layer. All layers are fully connected feedforward layers.

3.2.1 Architecture

Suppose the MLP is to find a function $f : \mathbf{x} \to \mathbf{y}$ where $\mathbf{x} \in \mathbb{R}^D$ and $\mathbf{y} \in \mathbb{R}^K$. This is the general setting for an MLP. The dimensionality D equals the number of input features and K is the number of desired output values. Common choices include $K = 1$ or $K = 2$ for a binary classification problem, $K = 1$ for a single-valued regression problem, and $K = 10$ for a ten-class classification problem. Refer to Fig. 3.1 for an illustration of this MLP with N hidden layers.

Notice that there is a single bias node connected to the first hidden layer. A bias node has a constant feature value (typically set to 1) which allows each node in the following layer more flexibility by "biasing" the node pre-activation value towards a learned value when all input features are zero [3]. It is unclear if there is any advantage in adding bias nodes beyond the first layer, as it is possible for the network to construct nodes in subsequent layers that behave like bias nodes, if they are required. This can be achieved by increasing a positive weight value from the bias node (in the earlier layer) to a subsequent node while suppressing the other weight values to the same node. Preliminary experiments (not shown) suggest that the omission of bias nodes at hidden layers beyond the first one has no discernible effect on the performance of MLPs in the context within which

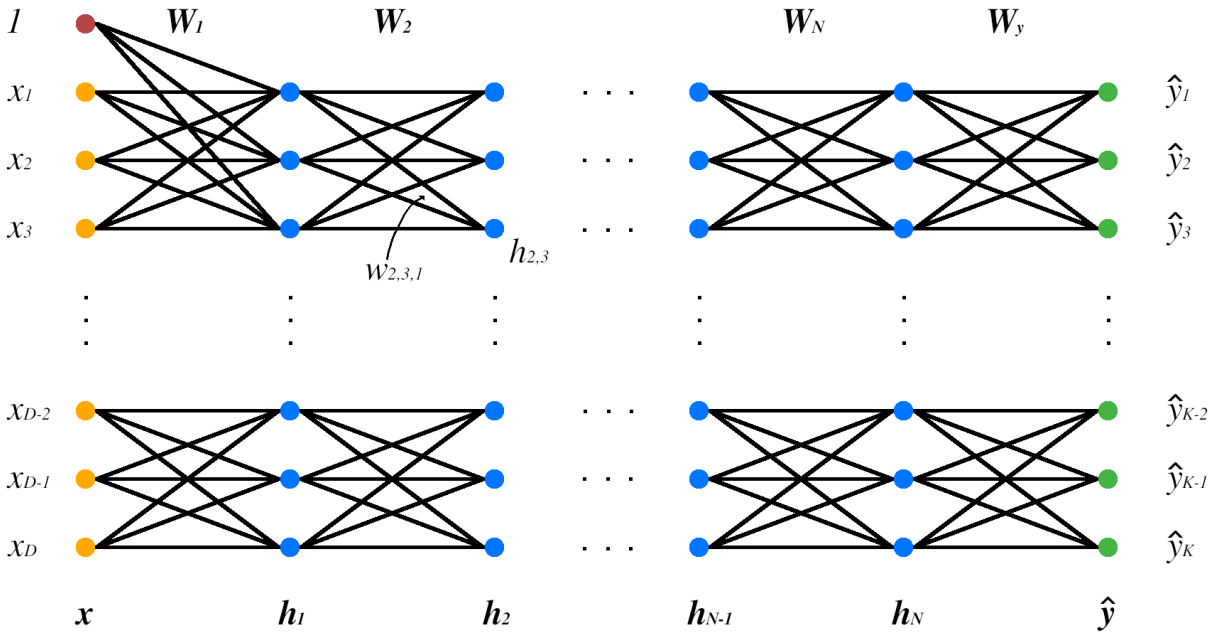

Figure 3.1: Illustration of an MLP. The input, bias, hidden, and output units are represented by orange, red, blue, and green, respectively. $\widehat{\mathbf{y}}$ denotes the output of the network and not the label data $\mathbf{y}$.

we study them. Consequently, we use a single bias node by default, in our experiments, in order to allow the models sufficient flexibility while retaining an extremely compact analytic form when describing model updates.

A common practice in analytical investigations of feedforward networks is to absorb the bias terms into the input feature vector and first hidden layer weights [59–61]. The bias node is included in $\mathbf{x}$ as an additional entry (that is always equal to 1) and the bias weights are included in the relevant weight matrix as an additional column. By assuming this viewpoint we can safely omit explicit reference to the bias parameters in our theoretical investigations.

3.2.2 Activation function

With regard to Fig. 3.1, if we define $\mathbf{x}$ as $\mathbf{h}_0$, every layer $l \in [1, N]$ performs the following function:

$$\mathbf{h}_l = \alpha_l\big(\mathbf{W}_l \cdot \mathbf{h}_{l-1}\big), \tag{3.1}$$

where α_l refers to an element-wise activation function. Historically, a sigmoid function was used but modern ANNs typically use a ReLU [62] or a variation thereof [63–65]. The element-wise ReLU function, applied to a pre-activation value z, can be defined as:

$$\text{ReLU}(z) = \max(0, z). \tag{3.2}$$

These activation functions allow the ANN to learn non-linear functions. If a linear activation function is used the model can only fit linear functions regardless of model depth or width. There is also some empirical evidence of improved optimization and generalization arising from using activation functions such as the ReLU as opposed to more classic activation functions that have smooth gradients, e.g. the sigmoid or hyperbolic tangent function [62].

The output layer usually uses a different task-related activation function. For example, a model performing a classification task can use a softmax activation function to convert the output values into a normalized probability distribution over the possible classes. However, it is not strictly necessary to use an activation function at the output layer.

3.2.3 Optimization

A gradient-based algorithm is generally used to optimize the weight parameters. Some variation of gradient descent is used in virtually all DL systems. The following list defines how gradient-based learning is typically used to update weight values in an ANN, and a discussion of each step follows.

1. A **set of training samples** is passed through the network.
2. A **loss function** is used to measure the training risk w.r.t these samples.

3. **Backpropagation** is used to calculate the the gradient of the training risk w.r.t each parameter.

4. A gradient-based **update rule** is used to change each weight parameter.

5. Items 1 to 4 are repeated until some **stopping criterion** is met.

Set of training samples

The set of training samples used for weight updates can range from the entire train set (*batch learning*) to a single sample (*on-line learning*). However, due to various empirical and theoretical motivations we tend to use a number of samples between these two extremes, called a *mini-batch* [66–68]. A simple way to select this subset is to sample data randomly from the train set, hence the label Stochastic Gradient Descent (SGD). Usually, this ensures that the mini-batch is a reasonable approximation of the train set. Factors such as class imbalances, limited training data, and undersized mini-batches can necessitate more advanced techniques such as smart sampling [69], cross-validation [36], or specialized loss functions [70]. Our experiments are conducted on widely used benchmark classification datasets that do not significantly suffer such problems. Therefore, we sample our training data randomly.

Loss function

The goal of the loss function is to measure the discrepancy between the predicted output and the ground truth label data. Ideally we want this function to be differentiable and ensure that the function estimated by the overall optimization process has low bias and variance, and is also *consistent*. This means that it becomes a better approximation of the risk w.r.t the true underlying data distribution as the train set becomes bigger [3]. Simple choices of loss function include Mean Squared Error (MSE) and Cross Entropy (CE) loss. Next we discuss these functions for a set of training samples B.

We would like to preface this discussion by stating that we do not use any activation

function at the output layer. Our output layer, therefore, performs a linear transformation into a K dimensional space, where K is the number of classes. This means that, in order to determine the output class of a sample s, we simply use an argmax on the raw unnormalized output vector $\widehat{\mathbf{y}}^s$.

The MSE loss function measures the expected squared difference between output vectors and label vectors. This is a natural choice for regression problems where the output value has strong numerical significance, however it can still be used to perform classification tasks if $\mathbf{y}$ is a one-hot encoded class indicator. If the label data in B is of the form $\mathbf{y} \in \mathbb{R}^K$, the MSE loss can be calculated with Eq. 3.3.

$$\text{MSE}(B) = \frac{1}{|B|} \sum_{s \in B} \sum_{c \in K} \left(\widehat{\mathbf{y}}^s[c] - \mathbf{y}^s[c]\right)^2, \tag{3.3}$$

where $\widehat{\mathbf{y}}^s[c]$ is the value at the output unit corresponding to class c, in response to sample s, and $\mathbf{y}^s[c]$ is the corresponding label data.

In a classification-specific context where the label data in B is of the form $y \in [1, K]$, we follow the tendency of modern toolkits (such as pytorch) to calculate a variation of CE loss. This loss function combines a softmax activation and a Negative Log Likelihood (NLL) to measure the dissimilarity between the distribution of the training data and the output. It is defined as:

$$\text{CE}(B) = -\frac{1}{|B|} \sum_{s \in B} \log\left(\sigma(\widehat{\mathbf{y}}^s[y^s])\right), \tag{3.4}$$

where, $\sigma(\widehat{\mathbf{y}}^s[y^s])$ is the value at the output unit corresponding to the index y^s after an elementwise softmax is applied. The softmax normalizes $\widehat{\mathbf{y}}^s$ into a probability distribution over classes and, for an output unit $\widehat{\mathbf{y}}[j]$, is defined as:

$$\sigma(\widehat{\mathbf{y}}[j]) = \frac{e^{\widehat{\mathbf{y}}[j]}}{\sum_{c \in K} e^{\widehat{\mathbf{y}}[c]}}. \tag{3.5}$$

We make use of both MSE and CE loss functions in our experiments. When we use the MSE loss we use one-hot encoded vectors as label data. In our experiments, when we use CE loss, we use the version as defined above. The actual values obtained from the final layer are therefore *without applying an activation function* even though such function is used during training. This difference is important to understand for the discussions that follow. This modified CE function has a similar effect to standard CE loss during gradient updates, as is shown in Appendix B.1.

Backpropagation

Once an empirical risk value is obtained, the gradient w.r.t each weight parameter needs to be calculated. This is achieved with backpropagation [71]. The exact expression for calculating the gradient for each weight is determined by several factors such as the loss function, activation functions, and network architecture. However, the general procedure is to differentiate the loss function L w.r.t the output units $\widehat{y}_j \in \widehat{\mathbf{y}}$ to obtain $\frac{\partial L}{\partial \widehat{y}_j}$ and then, using the chain rule, differentiate the loss function w.r.t units in the preceding hidden layer $h_{N,i} \in \mathbf{h}_N$ to obtain $\frac{\partial L}{\partial h_{N,i}} = \sum_j \frac{\partial L}{\partial \widehat{y}_j} \frac{\partial \widehat{y}_j}{\partial h_{N,i}}$. This calculation can be extended backwards through the network to obtain the gradient $\frac{\partial L}{\partial h_{l,j}}$ w.r.t any hidden unit j in layer l.

In order to determine the gradient i.t.o a specific weight, say the weight connecting the j^{th} node in hidden layer l with the i^{th} node in the preceding layer, we can use the chain rule one final step; noting that $\mathbf{h}_l[j] := h_{l,j}$. This can be calculated with Eq. 3.6.

$$\frac{\partial L}{\partial w_{l,j,i}} = \frac{\partial L}{\partial h_{l,j}} \frac{\partial h_{l,j}}{\partial w_{l,j,i}}. \tag{3.6}$$

Update rule

The next step is to adapt the weight parameters in accordance with their gradients. By doing this, we are moving the overall loss value closer to a local minimum. The most basic update rule (seen in Eq. 3.7) is to subtract the gradient, multiplied by a constant

scalar value, to produce a new weight value.

$$\Delta w_{l,j,i} = -\eta \frac{\partial L}{\partial w_{l,j,i}}. \tag{3.7}$$

The scaling constant η is referred to as the *learning rate* because it determines the size of steps in parameter space, and consequently the rate of learning. Optimizing the weight parameters of an ANN with such a basic update rule is often very difficult. There is a multitude of complications that arise as a result of the high level of non-linearity and curvature in the function space. Some of these complications include saddle points [72], cliffs, plateaus [73], and general ill-conditioning of the Hessian matrix. In order to lessen the effect of such phenomena additional scaling terms can be introduced into the update rule. Examples include: momentum terms [74], learning rate decay, gradient clipping [73], or even estimates of first- and second order moments (e.g. the Adam optimizer [75]).

Stopping criteria

The stopping criterion used to terminate the optimization of an ANN is an important matter. A common practice is to repeat parameter updates for a number of epochs (a single series of updates that utilizes the entire train set). If only a train set is available we can either stop optimizing after an arbitrarily chosen number of epochs or after the model is able to interpolate (correctly predict all the samples in) the training data.

These methods provide no guarantee that the model performs well on any data not in the train set. For this reason a separate validation set is often used as a measure of performance during training. Measuring the model performance on the validation set provides an estimate of how well the model is able to generalize from the training data. It is then natural to stop optimizing when the model has achieved a minimum error rate on the validation set. This is referred to as *early stopping*.

In our experiments we use a third dataset as 'evaluation set'. This is yet another set of samples on which model performance is only measured after the training procedure

has concluded and model selection has taken place. This prevents our experimental procedures from indirectly optimizing on the test data, which is common when calibrating hyperparameters under early stopping conditions.

3.2.4 Regularization

Another important component of training an ANN is regularization. This is yet another overloaded term. Some define it as any technique that reduces the out-of-sample risk [3], and others define it as any technique that penalizes a complex hypothesis space so as to improve generalization [76]. From the perspective of SLT these two definitions are equivalent. This is further complicated by the fact that improving optimization efficiency (effectively reducing the number of parameter updates required to fit an equivalent number of training samples) generally improves generalization as well. This could be because of algorithmic stability earlier in training [58, 77].

Examples of well known regularizers include: weight-decay, which limits the parameter norms to prevent models from approximating excessively complex functions; dropout [5,6], which prevents subunits from co-adapting to complex function descriptors; data augmentation [78], which creates larger and richer datasets to reduce overfitting; early stopping, which simply stops training when overfitting is empirically detected; and ensemble learning [79], which reduces variance by employing multiple learning algorithms and a voting system.

Less explicit factors thought to have a regularizing effect include: using smaller minibatches [66], sparsity in representations [62], sparse connectivity and parameter sharing in CNNs [80, 81], and batch normalization [82].

None of these techniques have been shown to be a necessary requirement to facilitate generalization. Unless stated otherwise, we do not use any explicit regularizing techniques in our experiments. This is so that our results more closely reflect the inherent ability of ANNs to generalize. For experiments where we want to investigate the "typical" subunit behavior for a well-optimized ANN we might use early stopping, and if we are investigating

the effect of adding noise to the representation of the training data we do not. This will always be clarified on a per-experiment basis.

3.3 Optimizing subunit parameters

Now that we have defined the MLP and how it is optimized through gradient-based learning, we present an analytical description of its training process through the lens of subunits (hidden nodes with piecewise linear activation functions). We will continue using the notation presented in previous sections.

3.3.1 Node-supported cost

If we define $\widehat{\mathbf{y}}$ as $\mathbf{h}_{N+1}$, and assuming one-hot encoded class indicators for MSE, the derivative of both MSE and CE w.r.t the output units reduces to a scaled difference between the target and output values. See Appendix B.1 for details. We define this difference as:

$$\boldsymbol{\lambda} = \nabla_{\mathbf{h}_{N+1}} L \propto \mathbf{h}_{N+1} - \mathbf{y}. \tag{3.8}$$

By applying the chain rule to Eq. 3.6 the gradient of a weight parameter $w_{l,j,i}$, on a per sample basis, can be written as:

$$\frac{\partial L}{\partial w_{l,j,i}} = \frac{\partial L}{\partial h_{l,j}} \frac{\partial h_{l,j}}{\partial z_{l,j}} \frac{\partial z_{l,j}}{\partial w_{l,j,i}}, \tag{3.9}$$

where $h_{l,j}$ and $z_{l,j}$ are the post- and pre-activation values at node j in layer l. The third factor is always equal to the activation value at node i in layer $l-1$. The other two factors depend on whether $l = N + 1$. For this reason we label the first term with $\phi_{l,j}$ and the second term with $\theta_{l,j}$ to produce the following equation.

$$\frac{\partial L}{\partial w_{l,j,i}} = \phi_{l,j}\theta_{l,j}h_{l-1,i}. \tag{3.10}$$

If $l = N+1$ then j is an output unit with no activation function so $\phi_{l,j} = \lambda_j$ and $\theta_{l,j} = 1$. However, if $l \neq N+1$ then layer l uses a ReLU activation function, and:

$$\theta_{l,j} = \begin{cases} 1 & \text{if } h_{l,j} > 0, \\ 0 & \text{else,} \end{cases} \tag{3.11}$$

and $\phi_{l,j}$ is the sum of the gradients of the loss function w.r.t all the weight parameters which fan out from node j. Each of these gradients can then also be the calculated with Eq. 3.10 up to the output layer where $\phi_{l,j} = \lambda_j$. In practice we do this calculation recursively, however we can construct an iterative expression with the following equation [21, 22]:

$$\phi_{l,j} = \sum_{p \in P_{l,j}} \big(\big(\prod_{w \in p} w \big) \lambda_p \big), \tag{3.12}$$

where:

- $P_{l,j}$ is the set of all active paths from node j to the output layer, $\widehat{\mathbf{y}}$ or $\mathbf{h}_{N+1}$. An active path refers to a set of subsequent weights that are connected by nodes where $\theta_{l,j} = 1$.
- λ_p is the derivative of the loss function w.r.t the output unit at the end of path p.

We refer to $\phi_{l,j}$ as the *node-supported cost* of node j in layer l. Fig. 3.2 provides an illustration to clarify Eq. 3.12. The significance of the node-supported cost is that it indicates the portion of the the discrepancy between the predicted output and ground truth value that can be attributed to the active paths branching out from node j. It also suggests that j is related to the other nodes along the active paths. This is because the gradients reaching j also affect the other nodes.

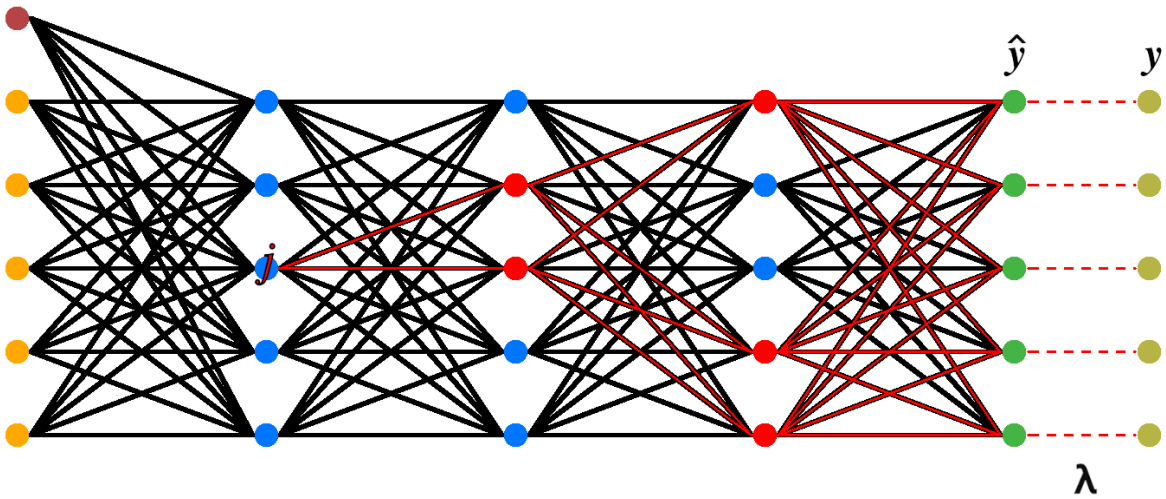

Figure 3.2: Illustration of how node-supported cost is determined in an MLP. Active paths between node j and output units are highlighted in red. The node-supported cost is the λ values multiplied by all weights in the path connected to node j, summed over all active paths.

3.3.2 Parameter-specific updates

By combining Eq. 3.7 and 3.10, and inserting the definition of node-supported cost, defined in Eq. 3.12, we obtain a weight-specific update rule:

$$\Delta w_{l,j,i} = -\eta \theta_{l,j} h_{l-1,i} \sum_{p \in P_{l,j}} \big(\big(\prod_{w \in p} w \big) \lambda_p \big). \tag{3.13}$$

In this equation $\theta_{l,j} = 1$ because the active paths necessitate that node j is also active. The active paths will be dependent on the sample for which the gradient is calculated. If we sum over the subset $S_{l,j,i} \subset B$ for which both nodes are active, we get a complete update rule for weight $w_{l,j,i}$:

$$\Delta w_{l,j,i} = -\eta \sum_{s \in S_{l,j,i}} h^s_{l-1,i} \sum_{p \in P^s_{l,j}} \big(\big(\prod_{w \in p} w \big) \lambda^s_p \big). \tag{3.14}$$

For an alternative derivation of this update rule see [21, 22]. There are a few key insights obtained from this update rule. First, parameters are updated according to specific discrete sets of samples. Additionally, the gradient of a parameter cannot distinguish

between a node-supported cost that is added over large, but polarizing, per-path terms, and a node-supported cost that is added over small but similarly valued terms. Finally, in general, a large node-supported cost will result in a reduction in activation value for node j upon parameter update.

3.3.3 Additional scaling factors

We defined the update rule, given in Eq. 3.14, for the simplest case where there is a constant global learning rate and each parameter is only updated according to its gradient at a specific iteration. However, as mentioned in Section 3.2.3, we use additional terms to aid optimization in practice.

Learning rate decay and gradient clipping would not significantly affect the behavior of our update rule. The former will merely scale η in Eq. 3.14 based on how many training iterations have been completed. The latter will scale the entire update rule based on whether the resulting gradient step is larger than some threshold. Therefore, the gradients are still calculated according to the same active paths and sets of specific samples. Momentum uses an exponentially decaying average gradient step. This can be represented with Eq. 3.15.

$$\Delta w_{l,j,i} = \epsilon \Delta w'_{l,j,i} - \eta \sum_{s \in S_{l,j,i}} h^s_{i-1,i} \sum_{p \in P^s_{l,j}} \big(\big(\prod_{w \in p} w \big) \lambda^s_p \big). \tag{3.15}$$

Each parameter update adds the previous parameter update $\Delta w'_{l,j,i}$ scaled by a constant ϵ. $\Delta w'_{l,j,i}$ can potentially be calculated with completely different paths, samples, and node-supported costs. During any iteration all parameters are synchronously updated. This means that each parameter is updated with the assumption that all other parameters will remain unchanged. The implication of this is that $\Delta w'_{l,j,i}$ will be very different to the current gradient step during early iterations and both will be approximately zero during convergence. Our analysis focuses on how backpropagation and ReLU activation functions disregard some samples when updating specific parameters during a single iter-

ation of training. We, therefore, focus on $\Delta w_{l,j,i}$ and $\Delta w'_{l,j,i}$ separately to obtain a good understanding of basic SGD; leaving an extension to momentum terms to future work.

The Adam optimizer is a slightly more sophisticated optimizer than SGD but operates on the same principles. From Eq. 3.14 we can define the unscaled gradient of weight $w_{l,j,i}$ with Eq. 3.16.

$$g_{l,j,i} = \sum_{s \in S_{l,j,i}} h^s_{l-1,i} \sum_{p \in P^s_{l,j}} \big((\prod_{w \in p} w) \lambda^s_p \big). \tag{3.16}$$

Adam uses an estimated corrected first moment of this gradient across training iterations, as defined in Eq. 3.17. This measures an exponentially decaying average of the gradient, but is corrected to compensate for initialization bias.

$$m_{l,j,i} = \frac{\beta_1 m'_{l,j,i} + (1 - \beta_1) g_{l,j,i}}{1 - \beta_1^t}. \tag{3.17}$$

Similarly, a second moment is calculated with Eq. 3.18. This measures an exponentially decaying uncentered variance of the gradient.

$$v_{l,j,i} = \frac{\beta_2 v'_{l,j,i} + (1 - \beta_2) g^2_{l,j,i}}{1 - \beta_2^t}. \tag{3.18}$$

In equations 3.17 and 3.18 β_1 and β_2 are constants that determine the respective decay rates of each moment and t refers to the number of completed training iterations. The final update rule is then calculated with Eq. 3.19, where δ is a small positive constant for numerical stability.

$$\Delta w_{l,j,i} = -\eta \frac{m_{l,j,i}}{\sqrt{v_{l,j,i}} + \delta}. \tag{3.19}$$

In a similar way to our description of how momentum affects our update rule, we acknowledge that Adam adds a level of complexity that is not considered by our analysis,

however both SGD and Adam measure $g_{l,j,i}$ to determine parameter updates which is the fundamental mechanism that we seek to understand.

As a final note, we acknowledge that our update rule does not account for the random sampling of mini-batches from the train set. SGD works with the assumption that you do not need to calculate the gradient w.r.t all samples in the train set to estimate a good direction of the slope. In our analysis we, similarly, assume that the relative relationship between subunits can be analyzed from a per-iteration perspective because the mini-batches are sufficient approximations of the train set as a whole.

3.3.4 Two systems

It should be clear by now that parameter updates, in a ReLU activated network, are heavily affected by the specific samples for which the connected nodes are active. It is helpful to think of this dynamic "selection" of weights (via paths) and samples (via node activations), to determine the node supported costs, as an interplay between two synergistic systems.

During the forward pass (items 1 and 2 in Section 3.2.3) nodes are activated to create binary valued node-sample pairs. This gives rise to a **discrete system** that is fully specified by these binary values. During the backward pass (items 3 and 4 in Section 3.2.3) the set of samples that are active at each node are used to adjust the parameter values feeding into that node. This **continuous system** is specified for each node separately as it uses the continuously valued weights, and node supported costs to update the weight vector feeding into each node [21, 22].

In short, the discrete system creates a per-sample binary *mask* over subunits which dictates whether each weight vector will be affected by that sample. The continuous system updates each weight vector only for samples with "activated" status in their respective mask at that node.

3.4 Feature spaces

Now that we have motivated how subunits are affected by discrete sets of samples during training, we will show how samples from these sets are measurably similar and describe how they are used to create more class-related feature spaces with subsequent hidden layers.

From Eq. 3.1 we can define the activation value at node j in layer l as:

$$h_{l,j} = \alpha_l\big(\mathbf{w}_{l,j} \cdot \mathbf{h}_{l-1}\big), \tag{3.20}$$

where $\mathbf{w}_{l,j}$ is the row in matrix $\mathbf{W}_l$ connecting layer $l-1$ and node j. The product of the two vectors can be rewritten to create the following equation:

$$h_{l,j} = \alpha_l\big(\|\mathbf{w}_{l,j}\|\|\mathbf{h}_{l-1}\|\cos\theta\big), \tag{3.21}$$

where θ is the angle between the weight vector and the activation vector in the preceding layer. The pre-activation node value is determined by the cosine similarity of the two vectors, scaled by the product of the norm of the activation vector (in the previous layer) and the norm of the relevant weight vector (in the current layer). As a result, if the activation function is a ReLU the angle between the activation vector and the weight vector has to be $\in (-90°, 90°)$ for the sample information to be propagated by the node. In other words, the node is activated for samples that produce an activation pattern in the preceding layer with a cosine similarity larger than 0 i.t.o the weight vector. This criterion holds regardless of the activation or weight strengths [23, 24]. This means that all samples that are active at node j will share a half-space (defined by $\mathbf{w}_{l,j}$) in the feature space of layer $l-1$.

In doing so, the network is creating a new feature space in the current layer l that is constructed on a per-dimension basis and each dimension is only optimized for a set of samples that are similar to each other. Half-spaces in high dimensional space tend to overlap a great deal. The result is that the network creates subsequent feature spaces with hidden layers that become more and more class-related, groups similar samples into

per-node sets, and reduces extreme specificity due to large overlap. See Fig. 3.3 for an example.

We say that feature spaces become more class-related partly based on empirical observations. Samples from different classes are more separated in the feature spaces of deeper layers. This will be very evident in the next chapter. However, it is also based on logic. In the input space there is no guarantee that samples from different classes will be well-separated. For a model that can correctly classify many train samples, the feature space at the output layer necessarily needs to represent samples from different classes differently. This is because all samples from a given class will need to have maximum values at the output unit corresponding to that class. Therefore, a transition from possibly low class separation to high class separation across depth is required. The nature of this transition is the subject of much of this thesis. See Section 4.6 for a more detailed discussion on how individual nodes can become more class related during training.

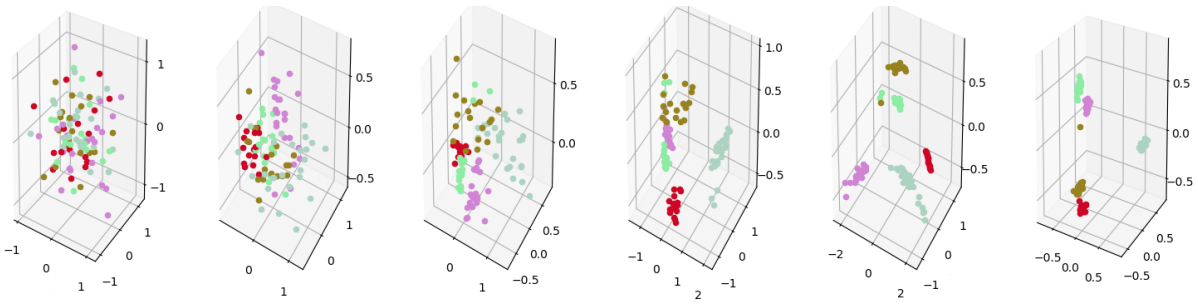

Figure 3.3: An illustration of feature spaces [from left to right: input; 4 × hidden layers; output layer] in a trained MLP. The model was trained to perform a 5-class classification task of 100 randomly generated 50 dimensional input vectors. Note that PCA is used to reduce the dimensionality of the actual feature spaces to 3 for this visual depiction.

3.5 Sample sets

For much of the empirical work in the following chapters, we investigate the *sample set* of nodes. This is simply defined as the set of samples for which a node is activated. A node is activated if its pre-activation value, for a specific sample, is larger than zero. Within the context of a ReLU activated MLP this means that any node outputting a non-zero value

is activated and gradients can be backpropagated to its corresponding weight vector.

It is possible to define an approximated sample set for other activation functions such as a sigmoid function. For example in [21, 22, 83] a pre-activation value above a threshold of zero was used to label a sigmoid-activated node as activated. This approximation is not as theoretically motivated as the ReLU activated case, but similar empirical results were obtained. As an analogy: such activation functions would create non-linear weighted contributions of all samples and not definite discrete sets like the ReLU function.

3.6 Broader applicability

In this section we discuss the extent to which our idea of subunit cooperation is applicable to more specialized ANNs. The following is a list of factors that might limit the applicability of our analyses. For each point we provide a brief discussion on how our findings might relate to models with alternative design choices. Take note that, without experimentation, these motivations are speculative.

1. **Fully connected feed forward layers**: For this type of architecture all hidden nodes are connected to all nodes in the preceding and subsequent layers, and to no other nodes. This is certainly not the case for more specialized ANNs which use methods such as weight sharing, convolution, pooling layers, recurrent connections, attention mechanisms, or skip connections, based on the type of data and problem task [3]. In spite of these variations almost all ANNs use a collection of, at least partially, interconnected subunits that connect the input and output. This is the main assumption we make w.r.t network architecture. The exact mechanisms that determine whether a sample is part of a sample set (see Eq. 3.21) would differ. However, the question of whether sample sets are constructed is more related to the activation function and interconnected nature of nodes than the specific network architecture.

2. **Gradient-based learning**: The idea of subunits being optimized on subpopula-

tions of the train set is heavily dependent on how the gradients propagate backwards through the network during gradient-based learning methods. There are many alternative optimization techniques, e.g. swarm optimization or genetic algorithms. However, gradient-based learning has seen the most practical success in the type of DL applications we study, by far.

3. **ReLU activations**: Our analyses show that subunits are optimized to reduce the loss for specific groups of training samples. ReLU activation functions provide a convenient way to distinguish when the gradients w.r.t a specific sample will affect a weight parameter. We suspect that it would be possible to produce analogous indications of membership to a sample set for activation functions that do not have the convenient piecewise linearity of ReLUs. This could take the form of a hard threshold as mentioned in Section 3.5 or assigning a "fuzzy" membership to all nodes.

4. **Classification problem**: We have chosen to investigate classification problems for their analytical convenience. The exact problem task is more related to the input and output features than the hidden layers of an ANN. For example an MLP performing a single-valued regression task would have a single output feature, but the hidden units will be trained in the same way. We would not be able to compare class-specific behaviors of hidden units, however sample sets will still be generated. For semantic segmentation there can be an equal number of input and output features [84], an ANN-based language model typically produces word embeddings with an empirically chosen number of features [85], and a generative adversarial network consists of two ANNs with output features specifically chosen so that the generator can produce good approximations of an input distribution [86]. In spite of all these variations there are usually a set of (or a few sets of) hidden layers that perform a function approximation with little information of the problem task being performed by the global DL system.

5. **No explicit regularization**: Most practical DL systems make use of some regularizing techniques such as batch normalization layers [82], drop-out [5,6], or weight decay [3]. These methods can be helpful to reduce overfitting, but they are not a

requirement for generalization [13]. Most of our experiments do not use explicit regularization to focus on the inherent generalization performance of ANNs precisely in those conditions where good performance is *not* due to efficient regularization.

3.7 Discussion

The work presented in this chapter proposes a view of MLP learning that highlights the fact that subunits model subpopulations of the train set while "cooperating" with other subunits to reduce the overall training risk of the model. From this viewpoint one can think of an MLP as an ensemble of subpredictors each trained on a dynamically sampled train set all within a multitask learning paradigm. That is, multitask learning, in the sense that many tasks are being learned in parallel and use a shared representation [87]. The different tasks correspond to each individual node but also to each separate class. Representations are shared because each node, locally, identifies appropriate samples and the overall model, globally, distinguishes between multiple classes. It is well known that ensemble methods and multitask learning can improve generalization.

These findings suggest that subunits, individually and as part of the whole, can be researched to obtain insights on the generalization behavior of ANNs that are unlikely to be found if we focus only on global measures of function complexity and available parametric flexibility.

In subsequent chapters we will look at how informative subunits are as individual predictors, and in cooperation. We will also show how they tend to have varying levels of specializations i.t.o class information and noise, providing insights on the generalization behavior of overparameterized ANNs.

Chapter 4

Regularities in subunit behavior

In this chapter we investigate how subunit behaviors change during training, how class relevance is managed across model depth and width, and how subunits can be analyzed as individual classifiers. A large part of this chapter consists of work presented in [21, 22]

4.1 Overview

It is easy to think of high dimensional hidden representations in an MLP i.t.o one of two extremes. Either we oversimplify the representation and think of it as two- or three-dimensional feature spaces where we are strongly guided by (often faulty) intuitions. Or we forgo any intuition and think of it as one giant hidden state without any order we can comprehend, resorting to carefully selected metrics to reason about its behavior. In this chapter we show that, by regarding each subunit as a semi-independent component and observing its behavior w.r.t class information, we uncover remarkable regularity across models of various architectures and random initialization.

In Section 4.2 we train models of varying width and depth to be analyzed in subsequent

sections. In Section 4.3 we look at the class sensitivity of subunits in some of these models, and how it is managed for models of varying size. In Section 4.4 we use perplexity to investigate the binary activation patterns of samples at various layers in models of varying size. In doing so, we uncover some obvious redundancy in later layers. In Section 4.5 we support our proposition of the two synergistic systems (see Section 3.3.4) by measuring the power of subunits to generalize from information available to each system separately.

Parts of the research in Section 4.5 were described in [21, 22] and were also utilized in a related Masters dissertation [88].

4.2 Generalization over depth and width

We start by optimizing several MLPs of various widths and depths to perform a classification task on the MNIST and FMNIST datasets. The performance of the trained models are shown in Fig. 4.1. See Appendix C.3 for details on our experimental setup.

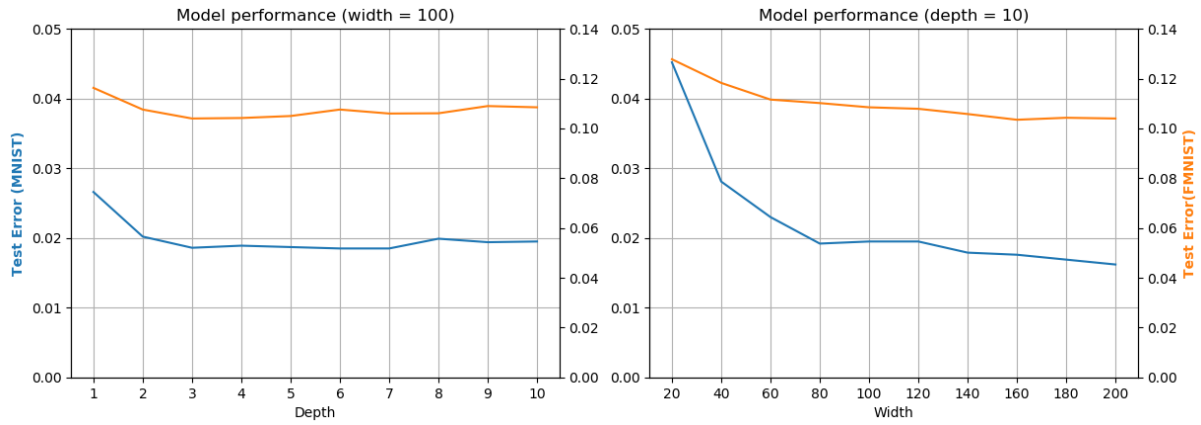

Figure 4.1: Generalization error for models of varying depth at a width of 100 (left), and varying width at a depth of 10 (right). The models represented by the blue curves are trained on MNIST data and correspond to the left vertical axis. The models represented by the orange curves are trained on FMNIST data and correspond to the right vertical axis.

At increasing depth, models trained on both datasets initially achieve decreasing generalization error, but the performance quickly saturates. Increasing depth (and thus

the number of parameters) does not seem to degrade performance, even when as many as twenty hidden layers are used. Selecting a model using the same grid search for a 20×100 architecture, we obtained a final evaluation error of 0.0198 on MNIST data. Results here shown up to a depth of 10. For a depth of 10 layers and increasing width, we see a similar saturation in performance. These models represent typical examples of well-optimized MLPs for good generalization performance on these two benchmark classification tasks [89–91]. They serve as basis for the majority of the analyses in this chapter.

The issue of vanishing or exploding gradients is sometimes a concern in such deep networks. We found that using an appropriate initialization scheme (He [92] for ReLU) in combination with scaling factors such as learning rate decay, momentum, or Adam's inherent dynamic scaling of parameter updates, prevented most issues with exploding or vanishing gradients. See Section 3.3.3. The reader will notice from Appendix C that for experiments where we use SGD as opposed to Adam, our initial learning rates are often larger. Apart from general hyperparameter tuning, we found that this is sometimes necessary to prevent initial parameter updates from being too small in deep networks. With very deep networks the early gradients can be vanishingly small, resulting in the optimizer being unable to move closer to a useful point in parameter space.

4.3 Subunit class sensitivity

For our first experiment we ask the question: If subunits are optimized on sample sets, how class-specific are these sets and how does this change after training? Fig. 4.2 shows a phenomenon that we have observed in many ReLU-activated MLPs of various sizes, trained with different algorithms, different hyperparameters, and on different classification tasks.

We observe that most nodes in a randomly initialized network activate for more-or-less random portions of the samples from each class. Later layers (meaning further away from the input layer) have slightly more nodes with extreme per-class activation ratios.

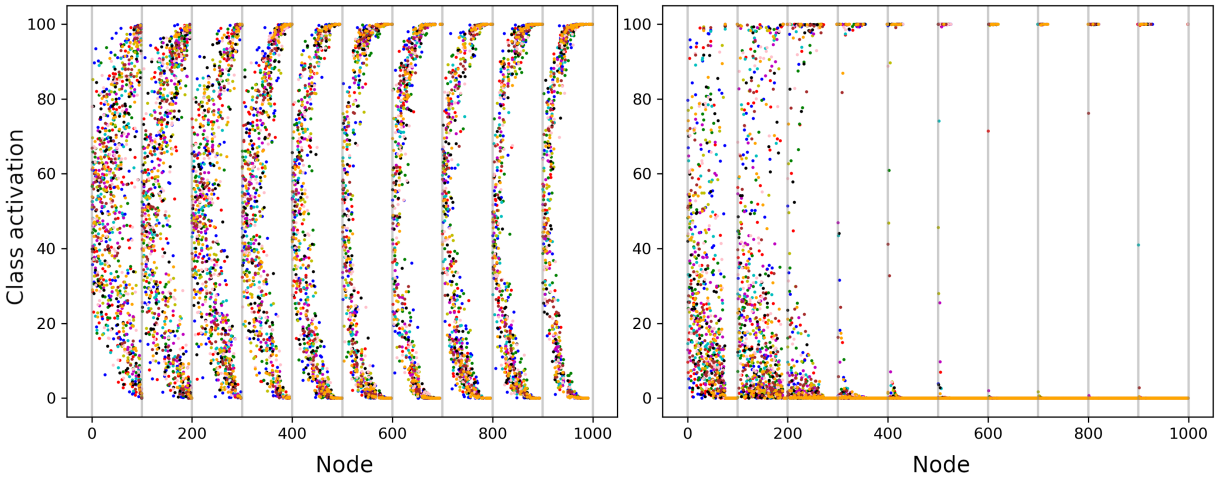

Figure 4.2: Percentage of class-related MNIST samples that activate each node in a 10×100 model at initializations (left) and after training (right). Each point refers to a node-class pair. Classes are indicated by separate colors and nodes are ordered from left to right according to which layer they belong to, from input to output. Note that we have also ordered nodes within each layer according to how far away from the 50% mark they are, along the vertical axis. These results are measured on the train set, but the same tendencies are observed when measured on a test set. See Appendix C.3 for experimental details.

By "extreme" we mean that the nodes activate for closer to 0% or 100% of all samples. This is to be expected because of the structure of the network. Due to the random initialization of weights, there is no reason for any node to be particularly attuned to any class-related features, but strong activations will be amplified in subsequent layers by weights that happen to be initialized with large positive values. Similarly weights that happen to be initialized with large negative values will result in zero-valued activations in the subsequent layers. This effect cascades over depth to result in later nodes that are activated (or not) for almost all samples.

After training, the nodes in the layers closest to the input are quite insensitive to any given class: these nodes tend to activate for some samples from most classes. Further away from the input, nodes tend to become selective: each node is activated by either none of the samples in a class or virtually all the samples in the class. This regular pattern occurs over a wide range of conditions, as long as the network has sufficiently many layers and nodes, and arises despite the random initialization of weights. Note that this effect is different to the one described w.r.t untrained networks, because the extreme

per-class activation ratios are much more numerous and relates to particular classes. It therefore seems to indicate a fundamental aspect of the way a DNN arranges itself to perform classification.

Earlier work on the complexity analysis of DNNs [93–95] has observed that hidden units in later layers produce many additional distinct linear regions in feature space; and that with depth, layer behavior becomes more abstract and class-specific. However, the observed transition with depth is strikingly sharp, and not spread out over available depth as one would expect.

By thinking of every hidden node as a kind of binary "class detector" we can measure the precision and recall of each node-class pair with the following equations:

$$p(j,c) = \frac{|S_j^c|}{|S_j|}; \quad r(j,c) = \frac{|S_j^c|}{|S^c|}, \tag{4.1}$$

where S_j^c is the sample set of node j limited to class c, S_j is the sample set of node j across all classes, and S^c is the set of all samples from class c. We measure the general class sensitivity of a node by using a weighted average recall over classes. This is presented in Eq. 4.2.

$$s(j) = \sum_{c \in K} p(j,c) r(j,c), \tag{4.2}$$

where K is the set of all classes. Note that the contribution of each class is weighted by the node's precision w.r.t that class, to capture the intuition that the relevance of a class to a node is proportional to the node's precision for the class. We can then get a per-layer value by disregarding *dead nodes* and averaging over all the remaining nodes of each layer. Dead nodes are those that do not activate for any sample and therefore contribute no information toward differentiating among classes.

Fig. 4.3 shows the per-layer class sensitivity of the models with 10 hidden layers and varying layer widths in Fig. 4.1, as measured on the evaluation set. As expected, later layers tend to have higher class sensitivity. We also observe that wider models tend to have lower class sensitivity in earlier layers suggesting a tendency for wide networks to rely less on specific subunits as class indicators.

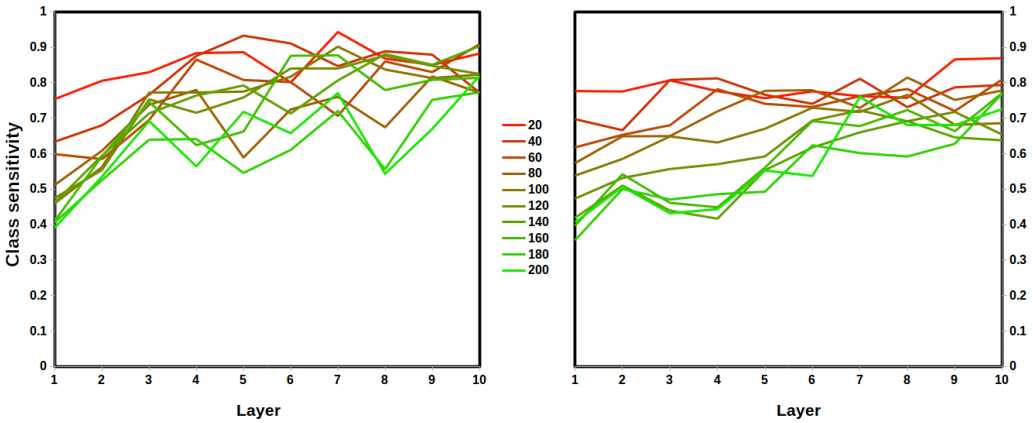

Figure 4.3: Per-layer class sensitivity for networks trained on MNIST (left) and FMNIST (right). Each color refers to the width of the 10 hidden layers each model uses.

4.4 Layer encodings

Continuing with our investigation of the discrete dynamics of hidden layers and class information, we look at the number of different binary activation patterns (from here referred to as *encodings*) that occurs at each hidden layer. Each encoding consists of a vector of binary values indicating whether each node in the layer is active for a given input sample, or not. If the total number of occurrences of each encoding for a layer l as a response to all samples from a class c is given by the set Q_l^c, then the entropy of the encodings for class c at layer l can be defined as:

$$H(c,l) = -\sum_{q \in Q_l^c} \frac{q}{|S^c|} ln\big(\frac{q}{|S^c|}\big), \tag{4.3}$$

where S^c is set of all samples belonging to class c; and the perplexity of the class c at layer l is defined as:

$$P(c,l) = e^{H(c,l)}. \tag{4.4}$$

In this context, entropy defines the average information content in the set of possible encodings and their frequencies, and perplexity provides an estimate of the total amount of information related to the encodings used by layer l to represent all the samples in class c. Minimal information is indicated by a perplexity of 1, which implies that the layer represents every sample of the class as an identical encoding. Maximal information

is indicated by a perplexity value equal to the total number of samples in the class: this happens when every sample is represented by a unique encoding at the current layer.

Fig. 4.4 shows the mean per-layer perplexities of the models presented in Fig. 4.1, with mean values obtained by averaging over all classes. The perplexities are measured w.r.t the evaluation set samples: with the focus of our analyses being on generalization, we are interested in encodings that are applicable during categorization of unseen samples, and not only those created for optimization purposes.

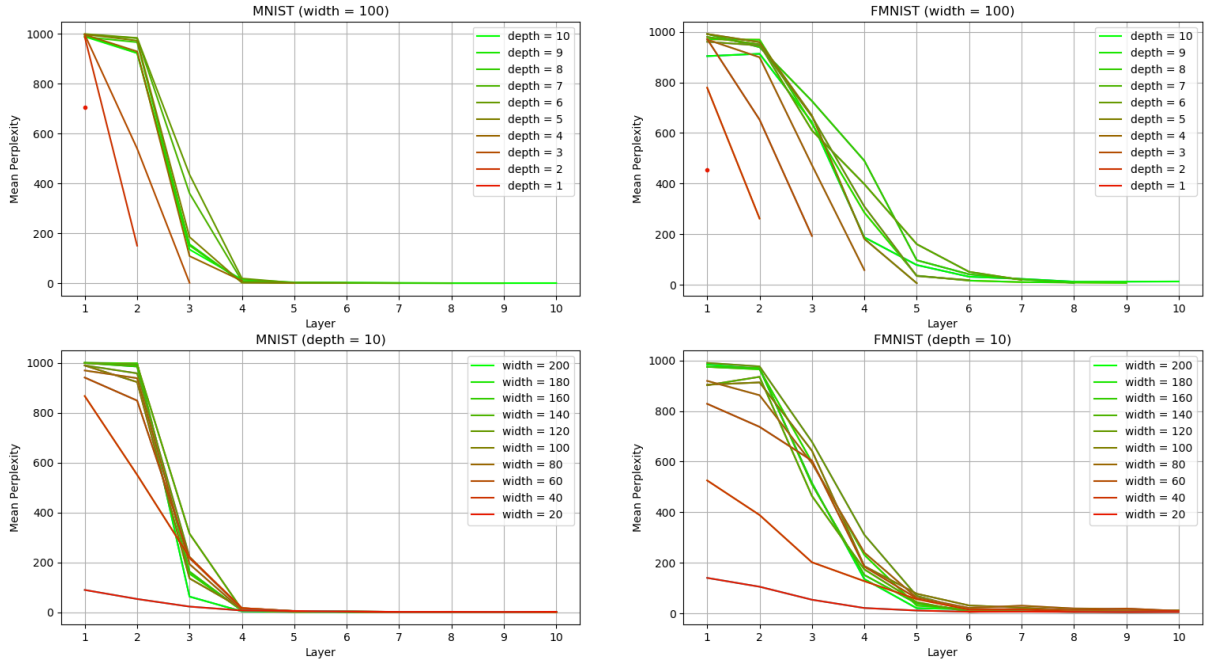

Figure 4.4: Mean per-layer perplexities for the models presented in Fig. 4.1. The models in the top plots have varying depth, and the models in the bottom plots have varying width. The models to the left and right are trained on MNIST and FMNIST, respectively.

There are several interesting observations that can be made from these graphs. Notice the relatively sharp drop in perplexity values for all networks, and take note that the drop is more gradual for the FMNIST models and sharper for wider networks. Additionally, for the networks with sufficient depth and width (coincidentally corresponding to most of the green curves):

- The perplexity in the later layers is very near 1. That is, all encodings have become fully class-specific.

- The perplexity values in the first 2 layers are almost equal to the total number of samples for each class (approximately 1 000 for the evaluation sets for both MNIST and FMNIST), which means that an individual encoding is created per sample.
- The transition from high perplexity to low perplexity is very similar across networks.

Lastly, notice that if the network width is below some threshold, the perplexity values in the earlier layers reduce accordingly. This cannot be due to a lack of representational power seeing as even the smallest layer (20 nodes) can represent more patterns than required by the number of training samples, i.e. $2^{20} > 55\ 000$. Taking into account their lower evaluation error, this suggests that wider networks represent sample information in a way that is more conducive to good generalization. This phenomenon was explored by Brutzkus et al. [96] where it is attributed to better weight exploration and a small number of observed prototype weight vectors.

Provided the network is large enough, there seems to be a range of earlier layers within which the nodes have high (virtually maximal) perplexity, and a corresponding range of later layers where the nodes have relatively low (virtually minimal) perplexity. Furthermore, the transition from the former behavior to the latter is consistent across all networks, irrespective of size, as long as they are deeper and wider than a task-specific threshold. After this transition, the class-specific discrete behavior in the excess layers is relatively trivial. (Perplexity is already at a minimum.)

The nodes in the earlier layers appear to perform most of the information processing required to produce a feature space that supports the ability to differentiate among samples relating to different classes. In this setup, the later layers effectively produce no new benefits and merely propagate the information forward through the network. The forward propagation of information, at this point, takes the form of a class-specific encoding, which is unique to each layer. By varying either width or depth, the same message emerges: a task-specific threshold exists w.r.t both width and depth, beyond which network behavior is strikingly regular and similar, irrespective of network size.

Recently, similar research on the activation patterns of ReLU-activated networks was

conducted [97]. The authors used the term *neural code* to refer to an analogous concept to our *encodings*, albeit measured over all layers. Similar to encodings in the first few layers, it was observed that almost all samples are represented by a unique neural code. It was proposed that neural networks form hash encoders with favorable properties such as determinism and categorization, meaning every encoding corresponds to exactly one sample and it is fairly easy to classify these encodings with simple algorithms, respectively. It is interesting to note that the determinism property, in our results, diminishes so rapidly in later layers. (The fact that results in [97] were also measured on CNN-based models such as ResNet [98] and VGG [99], supports the idea that our findings might translate to non-MLP models.)

4.5 Subunits as classifiers

In the previous two sections we have shown that individual subunit binary activations tend to be class-related, especially in later layers, and specific combinations of subunits also indicate class-related information. In Section 3.3.4 we described how the training of subunits can be seen as the interaction between two systems, one discrete and one continuous. The discrete system is defined by which nodes are active when applying different samples to the network. The continuous system updates the weight parameters based on the loss values (by means of the node-supported cost) of the samples identified by the discrete system at each subunit.

One way in which to determine the extent to which the discrete and continuous systems each exists in own right, is to analyze the classification ability of each system individually. We ask how well each system would be able to classify unseen samples, given either the discrete information available per sample (which nodes are on or off) or the continuous information per sample (pre-activation values at each node).

We now interpret each node as a classifier, implicitly estimating $P\big(z|y_c\big)$, where z is the pre-activation value and y_c a class. A discrete, continuous and combined estimate of this value is created at each node:

- *discrete*: if $z > 0$, $P\big(z|y_c\big)$ is estimated as the ratio of class c training samples with positive activation values w.r.t all class c training samples; 1 minus this value otherwise.
- *continuous*: the estimate provided by a kernel density estimator trained using all class c training data activation values observed at this node.
- *combined*: using the discrete estimate if $z \leq 0$, the continuous estimate otherwise.

This estimate is combined with the prior probability $P(y_c)$ of a class being observed to estimate the posterior $P(y_c|z)$:

$$P(y_c|z) = \frac{P(z|y_c)P(y_c)}{\sum_K P(z|y_k)P(y_k)}. \tag{4.5}$$

We view the nodes as independent classifiers (we ignore possible dependence) and multiply the probability estimates per class over all the nodes in a layer, to obtain a layer-specific probability estimate for each of the three systems. (In practice, the log probabilities are summed.) These probability estimates can then be used directly to classify samples based on maximum probability, creating three layer-specific classifiers for each layer in the network: a continuous, a discrete and a combined classifier. While no subcomponent of the network uses these probabilities directly, the performance of classifiers based on them, indicate how informative node behaviors are within the context of the discrete and (or) continuous systems. By analyzing the resulting classification ability at different layers and training iterations, we can provide insight into the importance and interaction of the continuous and discrete systems.

In Fig. 4.5, we demonstrate the performance of these three systems for the 6×100 network trained on FMNIST in Fig. 4.1; the behavior of this model during training expresses the overall tendencies for all the analyzed models very well. The most striking observation is that, at the later hidden layers, the accuracies of the three systems are virtually identical. In the first layer, the accuracy of the combined system is higher than both the discrete and continuous systems. This difference in classification accuracy among the three systems becomes smaller at later layers in the network, until it disappears. We expect the combined

system to consistently outperform the continuous and discrete systems, however in later layers these two systems separately achieve accuracies comparable to that of the combined system and global model as well.

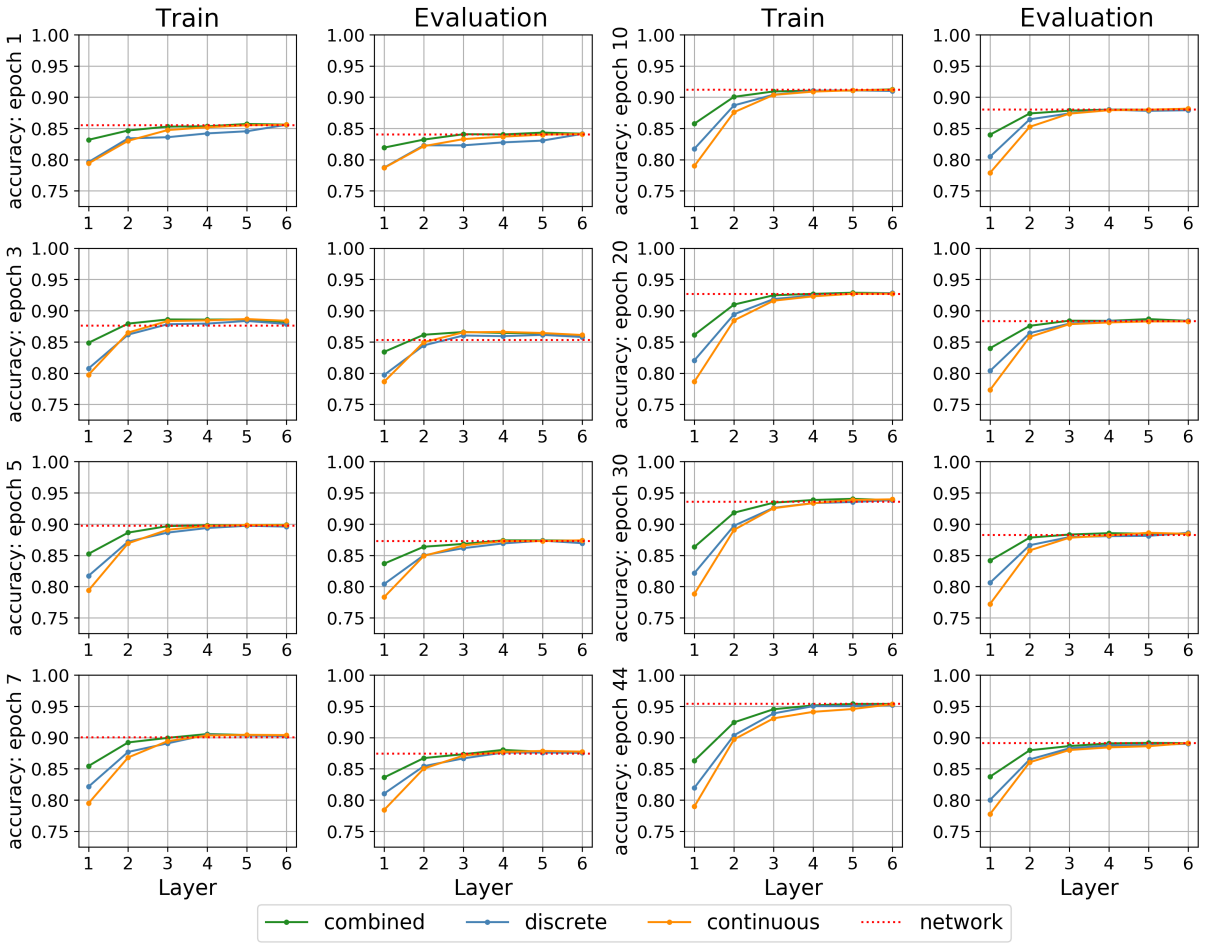

Figure 4.5: Classification accuracies of per-layer classifiers constructed using the three methods defined in Section 4.5. These classifiers are based on the train set activation patterns of a 6×100 model at four of the first epochs while training on FMNIST. Note that the dotted red line indicates the accuracy of the global model, at each epoch.

Additionally, it can be seen that the accuracies in the later layers improve visibly over iterations of learning while the performance of the earlier layers improves less. This is consistent with what we observe in Fig. 4.2. It appears that the feature spaces produced by earlier layers are meant to group subsets of the training data into informative groups, which are not necessarily related to classes, and the later layers use these groups to create very class specific feature spaces. Note how close the performance of the three systems, in later layers, are to the overall model performance at all of the presented training epochs.

This suggests that these systems are at least correlated with the underlying mechanisms dictating the change in generalization during training.

At the end of the first epoch, significant training has already occurred. In this case 859 parameter updates have taken place. We therefore also look at how the performance of these systems changes during mini-batch updates in the first epoch, as shown in Fig. 4.6. Note how poorly the continuous system performs initially (relative to the discrete system), until the training process stabilizes and the previously discussed trends emerge. It is also worth observing that the earlier layers outperform the later layers in this very-early stage of training.

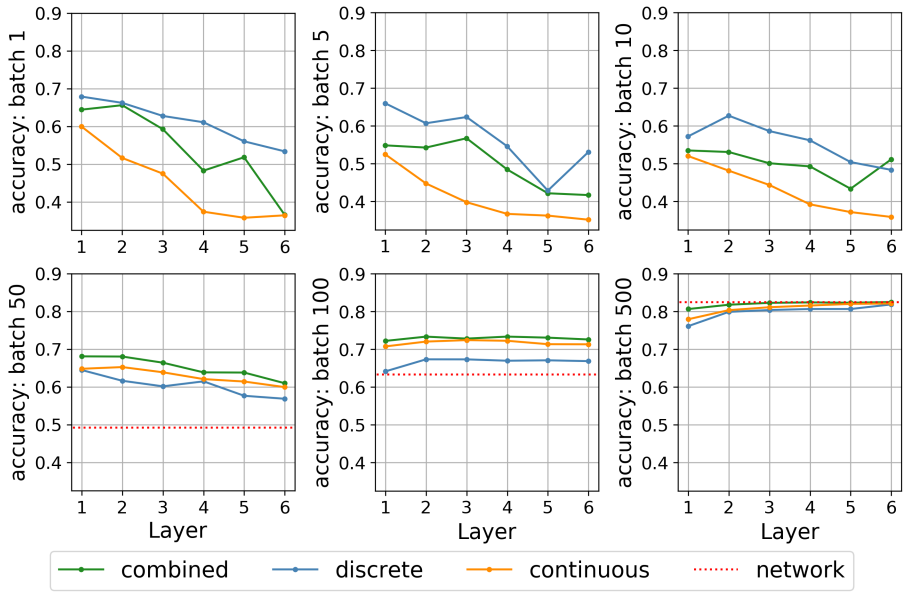

Figure 4.6: Evaluation accuracies of per-layer classifiers constructed using the three methods defined in Section 4.5. These classifiers are based on the train set activation patterns of a 6×100 model at several of the earliest iterations while training on FMNIST. Note that the dotted red line indicates the evaluation accuracy of the global model, at each iteration.

Similar trends are observed when changing either network depth (Fig. 4.7) or width (Fig. 4.8). It is striking to note that the three systems start overlapping when sufficient depth becomes available, but struggle to beforehand. Similarly, when the network layers lack width, the earlier layers underperform significantly. This is especially true for the discrete system. As expected, there is a clear increase in accuracy (across all systems) in the later layers with an increase in width. Curiously, the continuous performance appears

to reduce with an increase in width in the first layers.

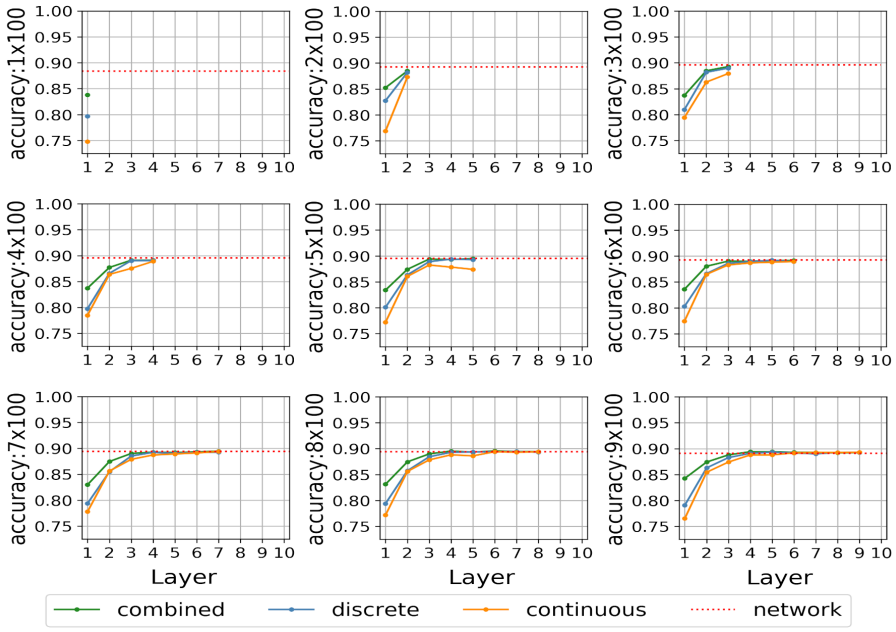

Figure 4.7: Evaluation accuracies of per-layer classifiers constructed using the three methods defined in Section 4.5. These classifiers are based on the train set activation patterns of models, with depths ranging from 1 to 9, after training on FMNIST. Note that the dotted red line indicates the evaluation accuracy of the global model.

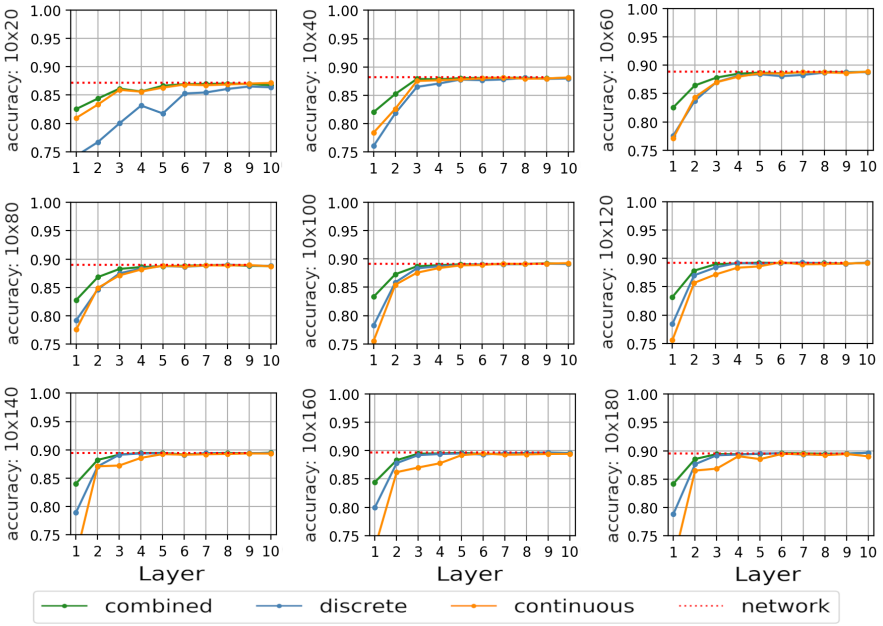

Figure 4.8: Evaluation accuracies of per-layer classifiers constructed using the three methods defined in Section 4.5. These classifiers are based on the train set activation patterns of models, with widths ranging from 20 to 180, after training on FMNIST. Note that the dotted red line indicates the evaluation accuracy of the global model.

In this section we only presented results for the models trained on FMNIST. This was done because the transition from low to high performance is much clearer for FMNIST models than for MNIST models. While not shown, we summarize the only differences the MNIST models produced: (1) the three systems and overall model attained higher accuracies in general; (2) the layer at which the three systems converge is earlier; (3) the three systems reach good performance faster.

The piecewise linear nature of ReLUs might naturally lead to clear discrete behavior. Therefore, in the interest of generality, we repeat the analysis from this section on a network with sigmoid activation functions (see Fig. 4.9). In order to obtain binary "activations" we regard a sigmoid-activated node as active when the activation value is larger than 0.5. This is a crude approximation, but it makes intuitive sense as this is the point at which the sigmoid function has maximal gradient and activation values are expected to diverge away from this value toward 0 or 1. Somewhat surprisingly, the discrete system again emerges very clearly. We see that the two systems in the sigmoid-activated network behave similarly to those in the ReLU-activated networks, except that the continuous system outperforms the discrete system by a small margin in later layers. Other trends remain.

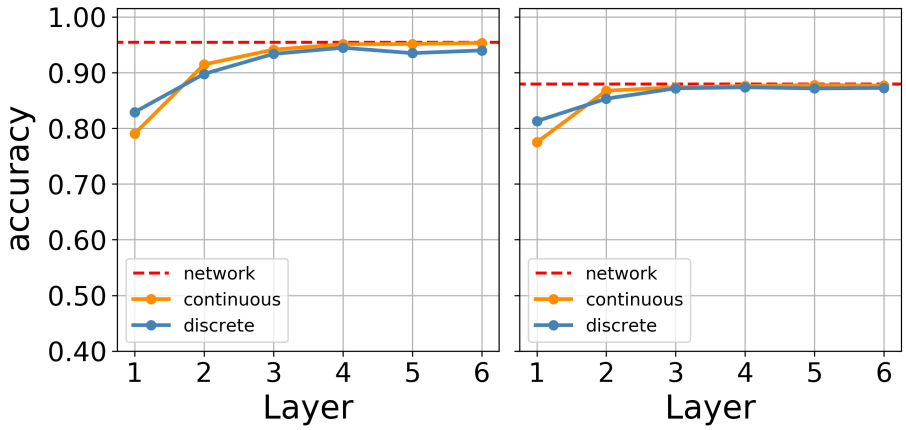

Figure 4.9: Train (left) and evaluation (right) accuracies of per-layer classifiers using the *continuous* and *discrete* systems, defined in Section 4.5. These classifiers are based on the train set activation patterns of a 6×100 model, using sigmoid activation functions, after training on FMNIST. Note that the dotted red line indicates the classification accuracy of the global model. See Appendix C.3 for experimental details.

4.6 Discussion

In this chapter we first showed that binary subunit activations, in a trained ANN, have varying levels of class sensitivity with earlier layers quite insensitive and later layers often reaching extremes of $P\big(z|y_c\big) = 1$ or $P\big(z|y_c\big) = 0$. We also showed that the specific binary activation patterns, at early hidden layers, remain deterministic w.r.t individual samples, but quickly become deterministic w.r.t classes in later layers, instead. For both of these observations, the transition from class-agnostic to class-specific behavior, across depth, seems to be task-specific. That is, the transition:

- is not spread out over available depth as is often envisioned, analogous to the levels of hierarchical abstraction of CNN features; and
- is fairly consistent over network widths and depths, assuming sufficient width and depth are available.

This finding suggests that an overparameterized ANN "squeezes" the class discrimination into the earlier layers, with the later layers merely propagating the class-deterministic information.

We now make use of the mechanisms defined in the previous chapter to explain how subunits can become more sensitive to a specific class.

- The weight vector is only optimized to reduce the loss for its sample set, which contains samples that are locally similar in the preceding feature space – see Eq. 3.21.
- With a large positive node-supported cost the weights that have strong activation values in the preceding layer are expected to reduce after a gradient update. See Eq. 3.13. So if some of the features in the preceding layer are very common for samples from some class, the corresponding weights connected to the current node will reduce for most samples from that class. The result is that the node will have lower pre-activation values for many samples from that class, possibly resulting in these samples being dropped from the sample set.

- With a large negative node-supported cost, the exact opposite will happen. The weights associated with features in the preceding layer that are common for the class will increase and the node will have higher pre-activation values for many samples from that class, possibly resulting in these samples being added to the sample set.
- Only when (or if) the node-supported cost moves closer to zero will the node's class sensitivity stabilize.

Initially the input features have a higher probability of being common to specific classes than the randomly initialized subsequent layers. So the nodes in the first hidden layer that happen to have negative node-supported costs will become attuned to classes before the later layers. This is reflected in Fig. 4.6 where we see that during the first few training updates, the earlier layers perform as better classifiers than the later ones. However, the more attuned the early features become to classes the more the features in subsequent layers too will become attuned, with the later layers quickly overtaking the earlier ones.

All of these results indicate that subunits have a remarkable amount of regularity in their activation behavior, across hyperparameters and random intitializations, suggesting the potential for developing a measure of generalization from the interaction among them. This is a view that is often overlooked in favor of a more top-down perspective where a global measure of complexity is sought.

Chapter 5

Overparameterization and noise

In this chapter we present results that show how concepts introduced in Chapter 3 can account for good generalization in spite of noise interpolation. A large part of this chapter consists of work presented in [23, 24].

5.1 Overview

It is a widely accepted notion that an ANN trained in the presence of a specific type of noise is expected to become insensitive to that type of noise [100, 101]. Intuitively, if the model performance is insensitive to random perturbations in the feature or weight space the model is expected to more easily generalize to out-of-sample data. This could be due to several reasons such as smoother functions being approximated [101], or larger and richer train sets (i.e. *data augmentation*) [78]. Whatever the reason, there is a limit to the amount of useful noise. For example, a model trained on pure Gaussian noise has no hope of learning any useful features, and a model trained with an excessively high drop-out [5, 6] constant lacks the expressivity to fit a sufficiently complex function.

An influential paper by Zhang et al. [13] showed that ANNs have representational capacities large enough to easily memorize various kinds of noise. The goal with their work was not to show how noise can improve generalization but to show that common ANN models are able to completely fit noise and still generalize when trained on natural datasets, leading to the "apparent paradox" described in Section 2.2.4. The noise they investigated were specifically chosen, not to improve generalization, but force the model to map complicated artificial function descriptors.

In this chapter we investigate similar types of training noise with a larger emphasis on how models generalize to noiseless test data when trained on varying levels of such noise. Our goal is to show how subunits manage noise by modularizing the representations of the training distribution. This enables noise to be separated from true task information. Generalization is then less affected (in some cases unaffected) by large amounts of deliberately spurious function descriptors being interpolated.

5.2 Training noise

We have chosen to analyze three kinds of noise: Label corruption, Gaussian input corruption, and Structured input corruption. These kinds of noise were chosen to encompass those investigated in [13], place more emphasis on varying the noise, and still maintain the analytically convenient per-sample implementation of the analysis framework. The following subsections describe each kind of noise, in detail. Note that $\mathcal{B}(1, P)$ depicts a single sample from a binomial distribution with a probability P of success.

5.2.1 Label corruption

The training label of an affected sample is replaced with an alternative selected uniformly from all other possibilities. This type of noise is identical to the "partially corrupted labels" in [13] except for the selection of alternative labels. Instead of selecting a random label uniformly, we select a random alternative (not including the true label) uniformly.

This results in a dataset that is corrupted by exactly the probability value P that determines corruption levels (see Algorithm 1).

For $P = 0$ no samples are corrupted. For $P = 1$ all samples will have training labels that are entirely unrepresentative of the original class distribution. We, therefore, expect extreme levels of class-related overlap in the input space with higher levels of label corruption.

Algorithm 1: Label corruption

Input: A train set of labeled samples ($X^{(train)}$, $Y^{(train)}$), a set of possible labels C, and a probability value P

Output: A train set of corrupted samples ($X^{(train)}$, $\widehat{Y}^{(train)}$)

```
for y in Y^(train) do
    if B(1, P) then
        ŷ ~ U[C\{y}]
    else
        ŷ ← y
```

Take note that if there are class-specific feature structures, this kind of noise is almost guaranteed to reduce generalization. That is because the model is forced to learn feature descriptors that are not only unrelated to the problem task, but actively adversarial to it. We, therefore, refer to label corruption as noise rather loosely. Nevertheless, it is useful to measure the extent to which generalization degrades at varying levels of label corruption (Chapter 5) and at varying model sizes (Chapter 6). The distinct differences in generalization behavior in the presence of the three kinds of noise presented in this section will become clear in Section 5.3.

5.2.2 Gaussian input corruption

All the input features of an affected sample are replaced with Gaussian noise with a mean and standard deviation equal to that of the uncorrupted samples. This type of noise is

similar to the "Gaussian" in [13], with the difference being that instead of replacing input data with Gaussian noise selected from a variable with identical mean and variance to the dataset, we determine the mean and variance of the Gaussian i.t.o the specific sample being corrupted, as in Algorithm 2. In other words, the corrupted sample contains features that are sampled from a Gaussian distribution that has a mean and standard deviation equal to those of the features of the uncorrupted sample.

This kind of noise completely destroys any structure in the input features of the corrupted samples and prevents any dependable correlation in feature values between corrupted samples and uncorrupted samples as well as corrupted samples and other corrupted samples. For $P = 1$ all samples will have random input features preventing any form of generalization. We note that the mean feature value for a sample before and after corruption will be unaffected by this type of noise, which might still be used as an indication of class. However, we have not observed this having any effect on generalization in practice.

Algorithm 2: Gaussian input corruption

Input: A train set of labeled samples ($X^{(train)}$, $Y^{(train)}$), a set of possible labels C, and a probability value P

Output: A train set of corrupted samples ($\widehat{X}^{(train)}$, $Y^{(train)}$)

```
for x in X^(train) do
    if B(1, P) then
        x̂ ~ N[μ_x, σ_x]
    else
        x̂ ← x
```

$\mu_{\mathbf{x}}$ and $\sigma_{\mathbf{x}}$ refer to the mean and standard deviation of all feature values in $\mathbf{x}$, respectively

5.2.3 Structured input corruption

For every affected sample, its input features are replaced by a set of transformed alternatives that is completely distinguishable from the true input but consistent per class. This

type of noise replaces input data with alternatives that are completely different to any in the true dataset but still structured in a way that is predictable. This is accomplished by rotating the sample 90° counter-clockwise about the center (prior to flattening the image data), followed by an inversion of the feature values. Inversion refers to subtracting the feature value from the maximum feature value in the sample – see Algorithm 3.

This kind of noise does not completely destroy any structure in the input features, like the Gaussian input corruption, therefore, there should still be strong class-related correlations in feature values between corrupted samples and other corrupted samples but very little between corrupted samples and uncorrupted samples. For $P = 1$ all samples are transformed in such a way that their class-related feature descriptors no longer relate to the problem task. It is similar to effectively training on an entirely different dataset.

Algorithm 3: Structured input corruption

Input: A train set of labeled samples ($X^{(train)}$, $Y^{(train)}$), a set of possible labels C, and a probability value P

Output: A train set of corrupted samples ($\widehat{X}^{(train)}$, $Y^{(train)}$)

```
for x in X^(train) do
    if B(1, P) then
        x̂ ← invert(rotate(x))
    else
        x̂ ← x
```

rotate is a 90° rotation counter clockwise about the origin

invert is an inversion of all values in the vector

5.3 Noise interpolation

We trained several models on three datasets with varying levels of the three types of noise described in the previous section. Fig. 5.1 shows the resulting generalization errors. See Appendix C.4 for details on our experimental setup. All models are heavily overparam-

eterized with 10 hidden layers of 512 nodes each. That is 5 120 subunits and 2 766 336 trainable parameters. All models were able to easily fit the noisy training data, corroborating the findings of [13]. The results are averaged over three random initializations. The standard error over initializations at interpolation was very close to zero for all models; hence the negligible error bars.

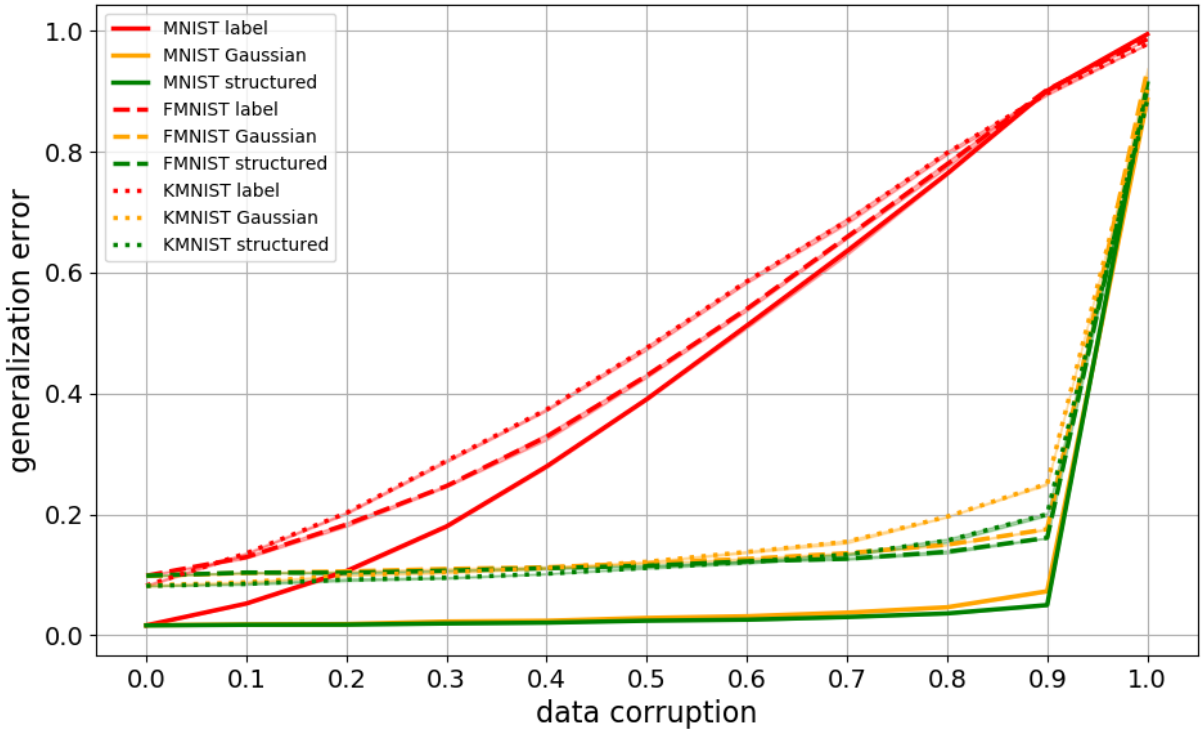

Figure 5.1: The generalization error for models trained on MNIST (solid lines), FMNIST (dashed lines), and KMNIST (dotted lines) at varying levels of three types of noise. These noise types are label corruption (red lines), Gaussian input corruption (orange lines), and structured input corruption (green lines). The horizontal axis represents the probability of any given training sample having been corrupted for the relevant model. All models are overparameterized and have perfect performance on the training data. All values are averaged over 3 random initializations. See Appendix C.4 for experimental details.

Notice that, when analyzing label corruption, there is a strong inverse correlation between the amount of label noise and the model's ability to generalize to unseen data. This suggests that either:

1. the models are memorizing sample-specific input-output relationships and a certain portion of the unseen data is similar enough to a corresponding portion of uncorrupted training samples to facilitate appropriate generalization; or

2. the global approximated function is somehow compartmentalized to contain fundamental rules about the task in some regions and ad hoc rules with which to correctly classify the corrupted samples in other regions.

Observe from the results of the two input corruptions (Gaussian and structured) that noise in the input data has an almost negligible effect on generalization up to the point where there is an insufficient amount of true data in the set with which to learn. This threshold is expected to change with more difficult classification tasks (more class variance and overlap), datasets containing fewer samples in total, and models with less parameter flexibility. This robustness to noise in the input space was not mentioned in [13]. We expect that is because the relevant experiments were aimed at showing that the models are large enough to memorize absolute input noise easily and still generalize from noiseless training data without considering the ability to generalize in the presence of noise. However, note that Zhang et al. [13] did vary the label noise, specifically, and observed a similar degradation in generalization to our results.

The fact that these types of input noise do not result in a linear reduction in generalization ability still supports both the previous propositions. If the first postulate is true, then the samples with corrupted input data are memorized, but no samples in the evaluation set are similar enough to them to incur additional generalization error. If the second postulate is true, then the regions in the approximated function that were determined by the corrupted input samples are simply never used for classifying the uncorrupted evaluation set.

It is also worth noting that the Gaussian and structured input corruptions have very similar influence on generalization. The models are therefore able to generalize in the presence of input noise regardless of whether the corrupted samples have informative (i.e. regular class-related feature patterns) structures in the input space.

5.4 Generalizing by modularizing

The cosine similarity ($\cos\theta$) in Eq. 3.21 can be used as an estimate of how much the representation of a sample in a preceding layer is directionally similar to that of the set of samples for which a node tends to be active. By measuring the average cosine similarity of samples in a node's sample set w.r.t the determinative weight vector ($\mathbf{w}_{l,j}$ in Eq. 3.21) and averaging over nodes in a layer, it is possible to identify layers where samples tend to be grouped together convincingly. That is, the samples are closely related (in the preceding feature space) and the resulting activation values tend to be large. Using Eq. 3.21, the *mean sample set cosine similarity* per layer l (over all weight and active sample pairs at every node) can be calculated as

$$\mu_{\text{cosine}}(l) = \frac{1}{|J_l|} \sum_{j \in J_l} \frac{1}{|S_{l,j}|} \sum_{s \in S_{l,j}} \frac{h^s_{l,j}}{\|\mathbf{w}_{l,j}\| \|\mathbf{h}^s_{l-1}\|}, \tag{5.1}$$

where J_l is a set of all the nodes in layer l and $S_{l,j}$ is the sample set of node j in layer l. Dead nodes are omitted from this calculation as they do not contribute to the global function and merely reduce the dimensionality of the feature space in the corresponding layer by one.

Fig. 5.2 shows this metric for the models presented in Fig. 5.1. It can be observed that, in general, earlier layers tend to have lower mean sample set cosine similarity than later layers. It appears that label corruption and structured input corruption results in the depth at which high mean sample set cosine similarities are obtained being later in the noise-corrupted networks compared to the baseline models, while Gaussian input corruption results in the opposite. The effects of structured input corruption are much more subtle than the other two types of noise. These findings suggest that the coherence of input features of training data noise plays an important part in the depth at which a convincing representation of sample information is formed.

The noise in the datasets is introduced probabilistically on a per sample basis (see Algorithm 1 to 3). This provides a convenient way to investigate the composition of sample sets w.r.t noise. Fig. 5.3 shows how sample sets can consist of different ratios of true and

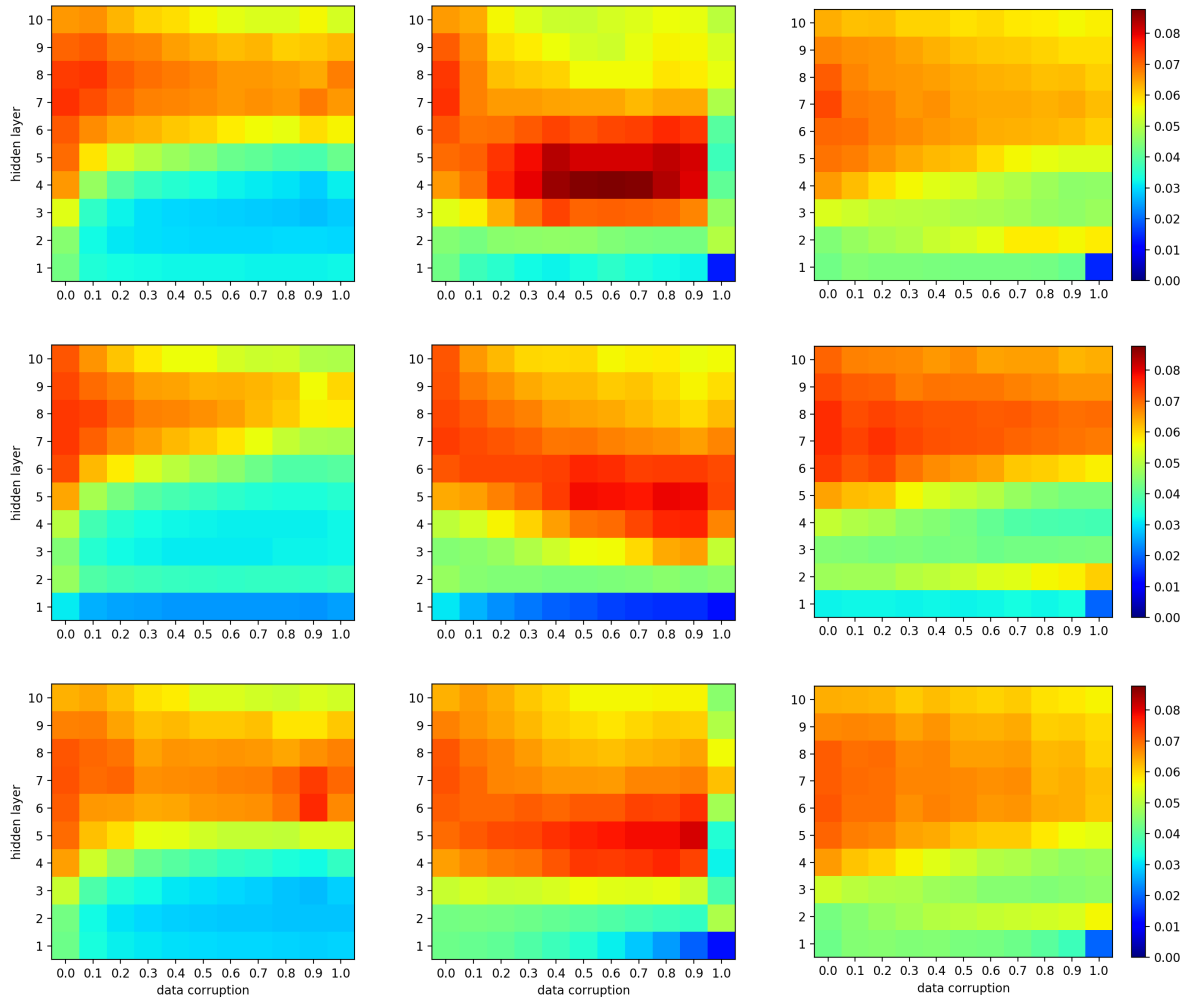

Figure 5.2: The mean sample set cosine similarities for models trained on MNIST (top), FMNIST (center), and KMNIST (bottom) at varying levels of three types of noise. These noise types are label corruption (left), Gaussian input corruption (center), and structured input corruption (right). These results are averaged over 3 random initializations and correspond to the models presented in Fig. 5.1.

corrupted sample information.

We define the *polarization* of a node-class pair as the amount the sample set favors either the corrupted or uncorrupted samples of a class. For a node j and class c this is defined as follows:

$$\text{polarization}(c, j) = 2 \left| \frac{1}{2} - \frac{|\widetilde{S_j^c}|}{|S_j^c|} \right|, \tag{5.2}$$

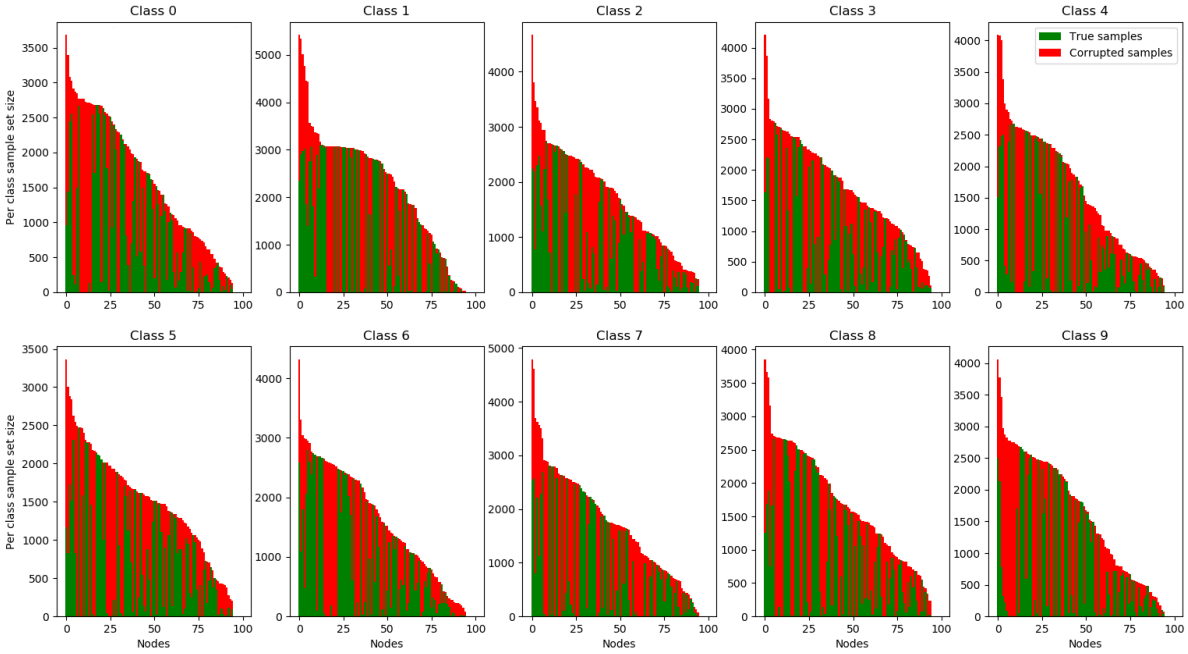

Figure 5.3: Per-class sample set corruption ratios for the first hidden layer of a 3×100 MLP fitting MNIST training samples, including structured input corruptions at a probability of 0.5. The nodes have been arranged in descending order of sample set size. The true and corrupted portions of the sample sets are presented in green and red, respectively.

where S_j^c is the sample set of node j in response to samples from class c, and $\widetilde{S_j^c}$ is a corresponding set limited to corrupted samples. A polarization of 1 means that the entire sample set consists of either corrupted or uncorrupted data. A polarization of 0 means that there is an equal number of corrupted and uncorrupted samples in the set. By averaging over nodes and classes, a per-layer mean polarization value can be obtained with the following equation:

$$\mu_{\text{polarization}}(l) = \frac{1}{|K||J_l|} \sum_{c\in K, j\in J_l} \text{polarization}(c,j), \tag{5.3}$$

where K is a set of all classes. The polarization metric indicates how much the sample sets formed within a layer are polarized between true class information and corrupted class information, for any given class.

The relevant polarization values are provided in Fig. 5.4. The main observation is that sample sets for specific classes tend to be highly in favor of true **or** corrupted sample information. This is especially prevalent for Gaussian input corruption where there is

very little feature correlations to be found in the corrupted samples.

The label corruption produces lower polarization earlier in the model, but this is to be expected seeing as the input data, from both corrupted and uncorrupted samples, has strong coherence based on correlations in class-related input structures. We observe that when the training data consists of 50% label corruption the earlier layers find it the most difficult to separate it from true task information.

The structured input corruption is well-separated from the uncorrupted samples early in the networks. This could be a result of the large difference between input feature structures from corrupted and uncorrupted samples along with the relatively coherent within-class input structures for both the corrupted and uncorrupted samples. These findings support the second postulate in Section 5.3. It appears that sub-regions in the function space are dedicated to processing different kinds of training data.

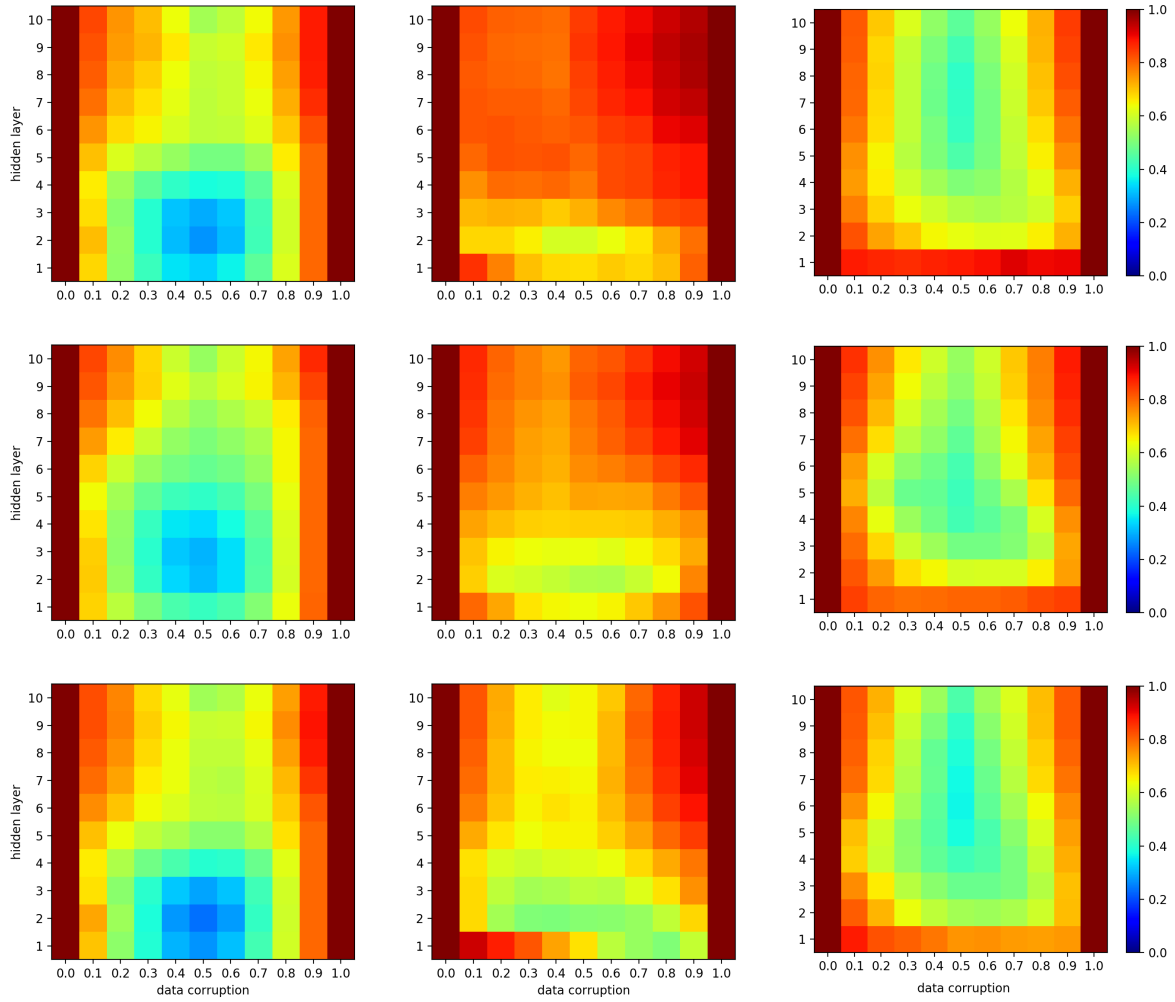

Figure 5.4: The mean per-layer polarization for models trained on MNIST (top), FMNIST (center), and KMNIST (bottom) at varying levels of three types of noise. These noise types are label corruption (left), Gaussian input corruption (center), and structured input corruption (right). These results are averaged over 3 random initializations and correspond to the models presented in Fig. 5.1.

5.5 Discussion

In this chapter we have shown that several overparameterized DNNs with no explicit regularization are able to generalize well with evident spurious input-output relationships present in the training data. We have used empirically evaluated metrics to show that sample sets in later layers tend to have higher within-set similarities, and the layer at which this occurs is affected by the amount and type of noise in the training data. Additionally, these sample sets are highly polarized between samples containing true task information

and samples without.

If we adopt the viewpoint that nodes in hidden layers act as collaborating feature differentiators (separating samples based on feature criteria that are unique to each node) to generate informative feature spaces, then each layer also acts as a mixture model fitting samples based on their representation in the preceding layer. Model components (referring to a node and its corresponding fan-in weight vector) are optimized on a specific subset of the population as determined by the activation patterns in the preceding layer. And, as we have observed, these subpopulations can and tend to be composed of true task information or false task information. In this sense, some model components of the network are dedicated to correctly classifying uncorrupted samples, and others are dedicated to corrupted samples.

To generalize this observation to training scenarios without explicit data corruption, it can be observed that in most datasets samples from a specific class have varied representations. Without defining some representations as noise, they are still processed in the same way the structured input corruption data is processed, hence the strong similarity between the baseline models and those containing structured input noise. This is also why it is possible to perform some types of multitask learning. One example would be combining MNIST, FMNIST, and KMNIST. In this scenario the train set will contain 180 000 examples with consistent training labels, but three distinct representations in the input space. For example, class 6 will be correctly assigned to the written number 6, a shirt, and a certain Japanese character. We empirically confirmed (see Appendix A) that such a model can easily generalize to any of the three separate evaluation sets as if it was trained on only one of the train sets separately.

To summarize, we present the *modular fitting* hypothesis to account for the observed ability of ANNs to generalize in spite of interpolating large amounts of non-generalizable feature descriptors:

> The subunits in a ReLU-activated ANN are only fitted to subpopulations of the train set that are similar in their respective preceding feature spaces. When out-of-sample data is to be processed, the regions most similar to the unseen data are used to make predictions, thereby preventing the model components

> fitted to sufficiently dissimilar sample information from affecting generalization.

According to this hypothesis, the ANNs do overfit on the noise, albeit in a benign fashion. The vast representational capacity and non-linearities enable subcomponents of the network to be dedicated to processing subpopulations of the training data. In effect, contradictory training samples are separated from one another and modeled independently, each associated with more closely related samples, only if such samples exist. This ensures that (within reasonable bounds) neither noise nor the memorization of a very large number of samples necessarily influence a given prediction.

This sheds light on why excessive capacity does not necessarily lead to overfitting. At least in the investigated framework, additional parameters imply an increased number of subcomponents with which to separate potentially detrimental feature descriptors from those that are related to the true underlying data distribution. These findings suggest that the capacity of an ANN should not be thought of as the complexity of the functions it will approximate but rather in its ability to partition data into signal and noise. This is akin to the processing of noise in a relational database, where attributes can have a varying number of possible values but an increased number of columns or rows will not result in less accurate responses to queries. It is also easy to see the link with the concept of distributed representations discussed in Section 2.3.2. An appropriate output is generated because only the features most relevant (i.e. having a weight vector similar to the activation pattern in the preceding layer) to the input sample are activated.

Chapter 6

Sample priority and double descent

In this chapter we track the order in which samples are fitted. In doing so we highlight novel aspects of how ANNs find structure in training data. Our findings shed light on the double descent phenomenon when classifying natural datasets.

6.1 Overview

The question we would like to address in this chapter is: Why do ANNs exhibit the double descent phenomenon when trained on natural datasets? By "natural" datasets we are referring to naturally occurring data: features and labels where we expect certain features (or groups of features) to have a definite relation to others. This is usually characterized by various levels of correlation among features, a variable amount of noise, and redundancy in the input space.

As defined in Section 2.4, double descent is a phenomenon observed in out-of-sample risk as a function of model capacity. In addition to the classic bias-variance U-curve (see Fig. 2.2.2) we observe that generalization improves for capacities larger than that which is

necessary to interpolate the training data. The point at which the second descent occurred is often referred to as the interpolation threshold and the region of poor generalization around it is called the critically parameterized regime [48].

We make use of two hypotheses w.r.t how ANNs generalize and discuss several phenomena (some known and some novel) observed when training ANNs, in order to provide a coherent account of the existence of the double descent phenomenon in ANNs. The first hypothesis is the coherent gradients hypothesis as discussed in Section 2.5. The second hypothesis is the modular fitting hypothesis defined in Section 5.5.

The main phenomena we discuss are as follows:

1. *ANNs share sample fitting priority.* (That is, they tend to fit samples in the same order during training.) This is a known behavior to which we provide a plausible explanation using the coherent gradients hypothesis. This work is presented in Section 6.3.

2. *Under certain conditions ANNs effectively refit training samples.* This a fairly evident behavior, but the implications for generalization error have not been explored extensively. We present such an exploration in Section 6.4, showing that substantial levels of systematic sample refitting late in training can be detrimental to generalization.

3. *The epoch-wise double descent phenomenon.* In Section 6.5 we explain how epoch-wise double descent is a result of some subunits not being heavily affected by the refitting behavior and maintaining generalization. We propose that the modular fitting hypothesis defines the main mechanism enabling this to occur.

4. *The model-wise double descent phenomenon.* In Section 6.6 we combine our findings from the previous three sections to account for model-wise double descent and define four novel and informative regimes of generalization behavior as a function of representational capacity. This encapsulates the known model-wise double descent curve.

This chapter fits into the theme of the thesis by showing that the distributed training of subunits and the resulting modularity of the hidden representation (i.e. the modular fitting hypothesis) elucidates the existence of the double descent phenomenon.

6.2 On double descent

Before starting our discussion of the link between sample priority and double descent we would like to highlight some observations that can be made from other's work. Since it was presented and defined [47], the double descent descent has been reproduced in many works, not limited to ANN-specific investigations. While not representing a formal meta-analysis we discuss some consistency in the conditions under which double descent is observed. We limited our research to studies with a focus on ANN-based algorithms. These observations are also reflected in *our* results. See Sections 6.4, 6.5, and 6.6.

Clear instances of **model-wise double descent in classification error** usually require some form of label noise or datasets with a high level of inter-class overlap. For example: in [8, 47–49, 102–104] we observe several instances of model-wise double descent in classification error for models (including MLPs and CNNs such as ResNet [98]) trained on CIFAR10 or CIFAR100 [105]. Many of these models have label noise added. In [47] and [53] we see this kind of double descent for MLPs trained on clean MNIST samples, which has much less inter-class overlap, although both instances make use of non-standard optimization practices. In [47] the authors make use of a "weight reuse" scheme. This means that larger models are initialized with the trained weight vectors of smaller models. In their supplementary material they repeated the experiment without weight reuse, which resulted in a much less convincing model-wise double descent. In [53], PCA was used to downsample the input features in the MNIST dataset, after which the dataset is divided into two classes (even and odd numbers). We expect such a dataset to produce significant inter-class overlap, resulting in clear model-wise double descent in classification error.

Model-wise double descent in average sample loss is not investigated as often as

the previously discussed type, but it typically produces much more convincing ascent and descent in the critically parameterized regime, even when there is little increase in generalization error. This suggests that critically parameterized models are obtaining extreme loss values for specific evaluation samples and not just poorer performance on the entire evaluation set as a whole. We refer the reader to [106] for a detailed investigation of this.

Clear instances of **epoch-wise double descent in classification error** usually require some form of label noise or datasets with a high level of inter-class overlap, even more so than its model-wise counterpart. For example: in [7, 24, 48, 50, 107] we observe several instances of epoch-wise double descent in classification error for models (including MLPs and CNNs such as ResNet [98] or VGG [99]) trained on MNIST, FMNIST, KMNIST, CIFAR10, CIFAR100 [105], or SVHN [108]; all of which have added label noise. In [53] we also see this kind of double descent for MLPs trained on MNIST, but we have already motivated that this is a non-standard compilation of MNIST samples with a high degree of inter-class overlap as a result. It is also worth noting that the second descent in classification error over training epochs is usually rather small when compared to what is observed as a function of representational capacity.

According to our knowledge no instance of **epoch-wise double descent in average sample loss** has been reported in the literature. In [109] it was proposed to artificially "flood" training losses with adversarial gradient steps when training losses become too small. This does induce an epoch-wise second descent in average sample loss. However, the proposed method artificially manipulates the training loss, making the optimization process very different to the expected behavior of an ANN trained on natural datasets.

An additional point to note, is that double descent, in general, does not occur under optimal early stopping conditions. This is supported by several works such as [8, 48, 110]. The dependence, of double descent, on factors such as inter-class overlap, training iterations, and the discrepancy between classification error and average sample loss suggests that it is linked to the way features are fitted earlier and later in training. This motivated us to investigate the order in which samples are fitted as a means of uncovering the origin of

the double descent phenomenon.

6.3 On sample priority

6.3.1 Priority by similarity

As mentioned in Section 2.5.2, the order in which a set of training samples are learned tends to be similar across different ANNs with similar architectures, and to an extent across different types of architectures as well. It was also reported that powerful algorithms tend to learn the samples a less powerful model would, before continuing to learn additional samples. Additionally, measuring the variance of sample-specific gradients during training can be used to estimate the "difficulty" of fitting training samples. So why do ANNs prioritize similar examples in spite of random weight initialization and differences in architectures?

We begin by addressing sample similarity. There are many metrics available to measure the similarity between input features of training samples, e.g. an Lp-norm, cross-correlation, mutual information etc. We expect samples of the same class to be more similar than samples of different classes, but this is not always the case. Furthermore, it is easy to find samples in common benchmark datasets where some samples are much more similar than others even within the same class. See Fig. 6.1 where we use an extremely crude measure of dissimilarity (Euclidean distance) to identify clear differences between within-class samples.

At initialization, weight vectors are not related to any task and it is then reasonable to assume that only loss gradients for samples, from the same class, with a large degree of similarity will significantly overlap (cohere). This can be made clearer by referring back to Eq. 3.14, repeated here for convenience:

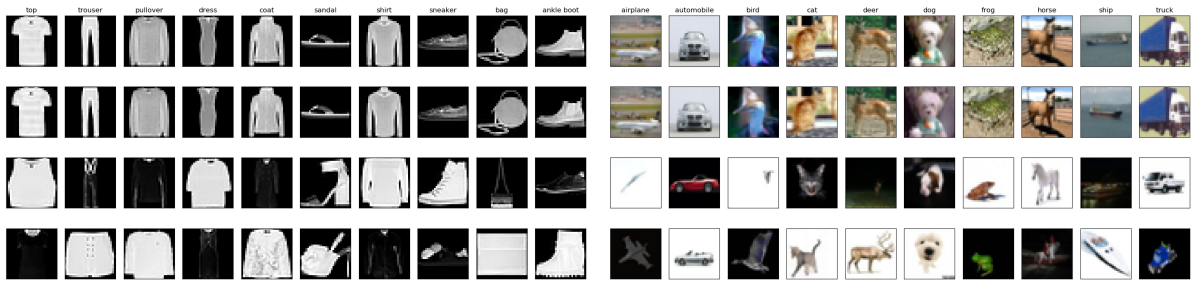

Figure 6.1: Illustrating extreme within-class dissimilarity in FMNIST (left) and CIFAR10 (right). Each column refers to a class. The top two rows and the bottom two rows show the two training samples with the smallest and largest Euclidean distance between flattened input features, respectively.

$$\Delta w_{l,j,i} = -\eta \sum_{s \in S_{l,j,i}} h^s_{l-1,i} \sum_{p \in P^s_{l,j}} \big((\prod_{w \in p} w) \lambda^s_p \big). \tag{6.1}$$

If many samples activate the feature represented by node i, the size of $S_{l,j,i}$ will generally be larger and the resulting parameter update, added over samples, will also be larger. At initialization, we expect this observation to be valid for all weight parameters because node j will not be overly sensitive to any local features in layer $l-1$.

According to the coherent gradients hypothesis [57] this will result in larger steps in parameter space to reduce the loss of those samples. Consequently, the samples containing many of these common features will experience faster reductions in their loss value and these samples will consistently be fitted early in training. This proposition is supported by a variety of common phenomena observed when training ANNs on natural datasets. Examples include:

- We tend to observe a rapid reduction in training loss values during early training, after which it stabilizes. Large parameter updates are being made that are relevant to a large subset of the train set.
- We observe that early reductions in training loss are usually accompanied by similar reductions on data outside of the training data. The features that create coherent gradients also necessarily need to be shared by many samples thereby increasing the

probability that they are applicable to the true underlying data distribution and not just the train set.

- Many different ANNs seem to prioritize similar samples during optimization and stronger architectures learn the samples that weaker architectures learn before continuing to learn new samples [55]. The task ambiguity of the randomly initialized weights results in only strong general similarities dominating across architectures.

After the early overlapping gradients have been sufficiently reduced the relative contribution of less coherent gradients are expected to increase. With sufficient parametric flexibility, parameter updates from such gradients should not significantly affect the representation obtained for the samples that are fitted as a result of the early coherent gradients. This is because any update made to reduce the loss i.t.o the features with non-overlapping gradients, that would increase the loss of the samples already fitted, would increase the loss of many samples simultaneously. This is consistent with the modular fitting hypothesis. In short: overlapping gradients from input features shared by many samples dominate parameter updates early, and later parameter updates will be unlikely to undo such parameter updates unless they too produce significant gradient overlap.

6.3.2 Evidence

The perspective proposed in the previous subsection strongly suggests that samples that tend to be fitted earlier contribute more to generalization than those fitted later. In addition to common empirical evidence that, almost universally, show initial improvement in performance on out-of-sample data we present a direct estimate of per-sample contributions to generalization.

Fig. 6.2 (top) provides a scatter plot of all the training samples in the MNIST dataset. The location of each data point is averaged over 75 models (with the relevant standard errors included as error bars). All models were able to attain a training error of 0.0 and differ in depth, width, or initialization seed. Details on the experimental setup are included in Appendix C.5. We have chosen to use the entire MNIST train set to ensure reproducibility

and comparability over training seeds, and because MNIST samples provide an intuitive illustration of various syntactically similar samples of each class.

The location of each sample on the horizontal axis is determined by the iteration at which the sample was *permanently* fitted. More specifically, it is the number of parameter updates that was required for the sample to not be misclassified for the rest of the training iterations. The location of each sample on the vertical axis is a measure of how much reductions in loss value w.r.t the sample correlates with the expected reduction on evaluation samples. To estimate this value for a sample s we use $g(s)$ defined in Eq. 6.2.

$$g(s) = \frac{1}{|D_{\text{eval}}|} \sum_{d \in D_{\text{eval}}} R(\mathbf{l}_{\text{diff}}(s), \mathbf{l}_{\text{diff}}(d)), \tag{6.2}$$

where D_{eval} refers to the entire evaluation set, R refers to the Pearson correlation coefficient, and $\mathbf{l}_{\text{diff}}(s)$ is a vector containing the difference between loss values of the same sample s at subsequent iterations. $\mathbf{l}_{\text{diff}}(s)$, therefore, has a length of $n-1$, where n refers to the number of iterations that was required to fit all the training samples. Note that, due to computation and storage constraints, we do not include all the iterations in this calculation. Instead, we use a logarithmically spaced sampling of the training iterations. See Appendix C.5 for details on this approximation.

From Fig. 6.2 (top) we see that, in accordance with our expectations, particular samples are consistently fitted earlier and reductions in their losses are relatively well correlated with expected reductions in evaluation set sample losses. The samples that are fitted later tend to correlate less with the evaluation set but their spread across the vertical axis is also larger. This is to be expected considering the large range of models that were averaged over as well as the idea that gradient overlap is not as strongly governed by task-ambiguous similarity in later training, because the weight vectors become more sensitive to class related local features.

The (center) and (bottom) plots in Fig. 6.2 overlay the input features onto the scatter plot for the samples that were generally fitted first and last, respectively. Take note that the samples fitted first contain strong visual similarity between samples from the same class. Additionally, the samples fitted first appear to be mostly from class *0* and class *1*.

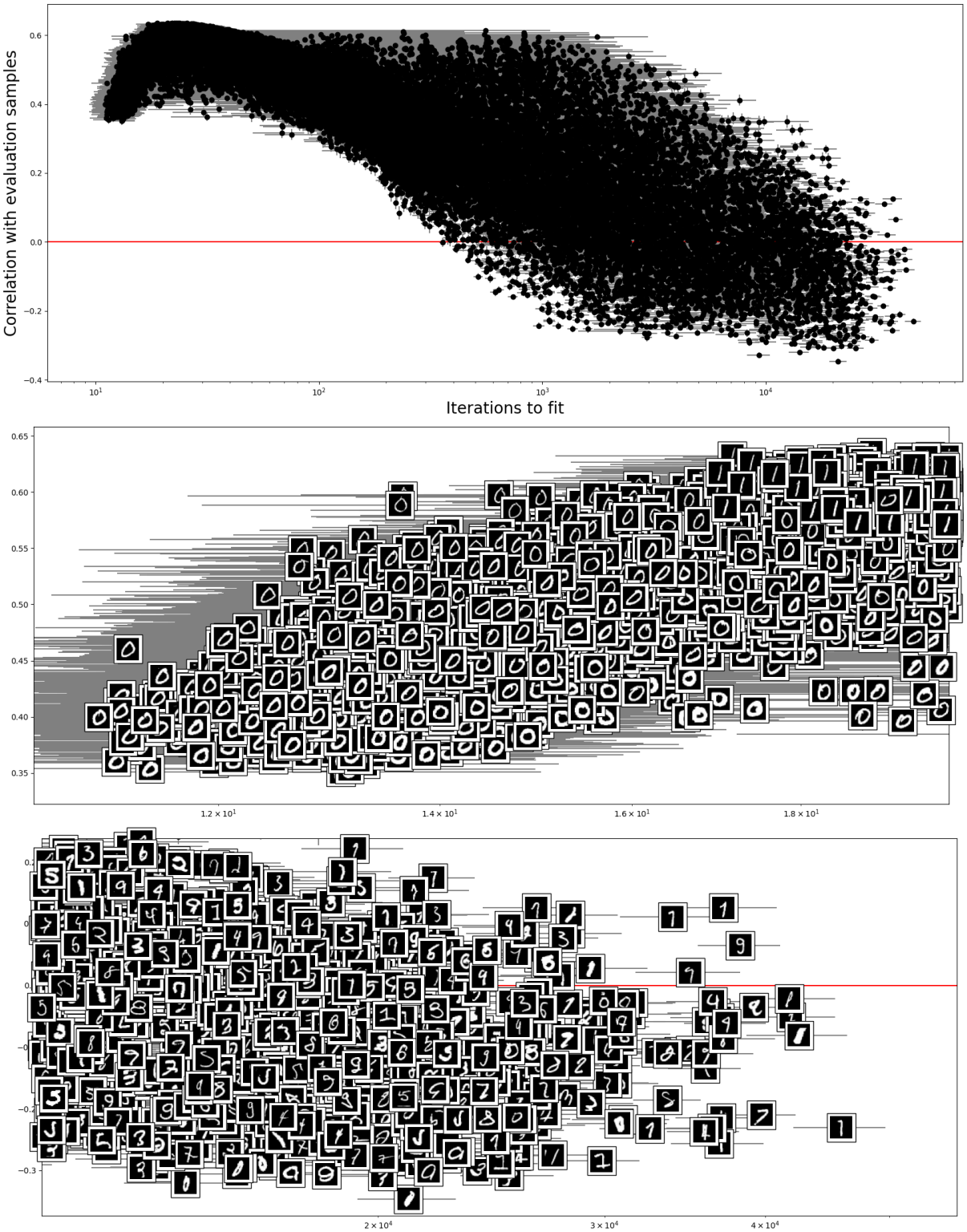

Figure 6.2: MNIST sample priority. (top) A scatter plot of $g(s)$ as defined in Eq. 6.2 and the iteration at which a sample is fitted. (center) A cropping of the first samples fitted with input features overlaid. (bottom) Another cropping of the last samples fitted.

These two classes, in particular these stereotypical renditions of them, are numerous and share little common features with the other eight classes in the dataset. In contrast to this, the samples which tend to be fitted last appear to be atypical versions of each class, often being heavily skewed or containing odd artifacts. These samples are not expected to produce gradients that are well aligned with the other samples of their respective classes. These findings support the notion that samples with coherent gradients are prioritized consistently and representations found early on (according to coherent gradients) are not lost by the point of interpolation. That is because, in Fig. 6.2, they are fitted early and never unfitted during layer training.

Fig. 6.3 shows the same experiment conducted on MNIST data that has been corrupted with one of the three mechanisms defined in Section 5.2. All three of the datasets have been corrupted by replacing a randomly selected 50% of the training samples with corrupted instances. Note that the evaluation set is left uncorrupted. See Appendix C.5 for the differences between the experimental setup for these models and those in Fig. 6.2.

There are a number of interesting observations that can be made w.r.t these three plots. First, notice that the samples that have structured input corruption are fitted in parallel with the clean samples, but the Gaussian input corruption and label corruption are consistently fitted very late in training. This corroborates our notion that input feature similarity and the resulting coherent gradients are driving forces in determining the order used to fit samples. Two samples that are similar will still be similar after structured input corruption is applied to both, because each sample is corrupted using the same transformation. The gradients from a large group of structurally corrupted "1's" will still overlap. In contrast to this, the Gaussian input corrupted and label-corrupted samples have very low similarity between within-class samples because the former prevents any measurable common input structures, and the latter reduces the probability of structurally similar samples belonging to the same class.

As expected, the corrupted samples tend to correlate less with the uncorrupted evaluation set. This indicates that features being fitted to reduce the loss on corrupted samples adds no power to generalize to the evaluation set. In the case of the label corruption,

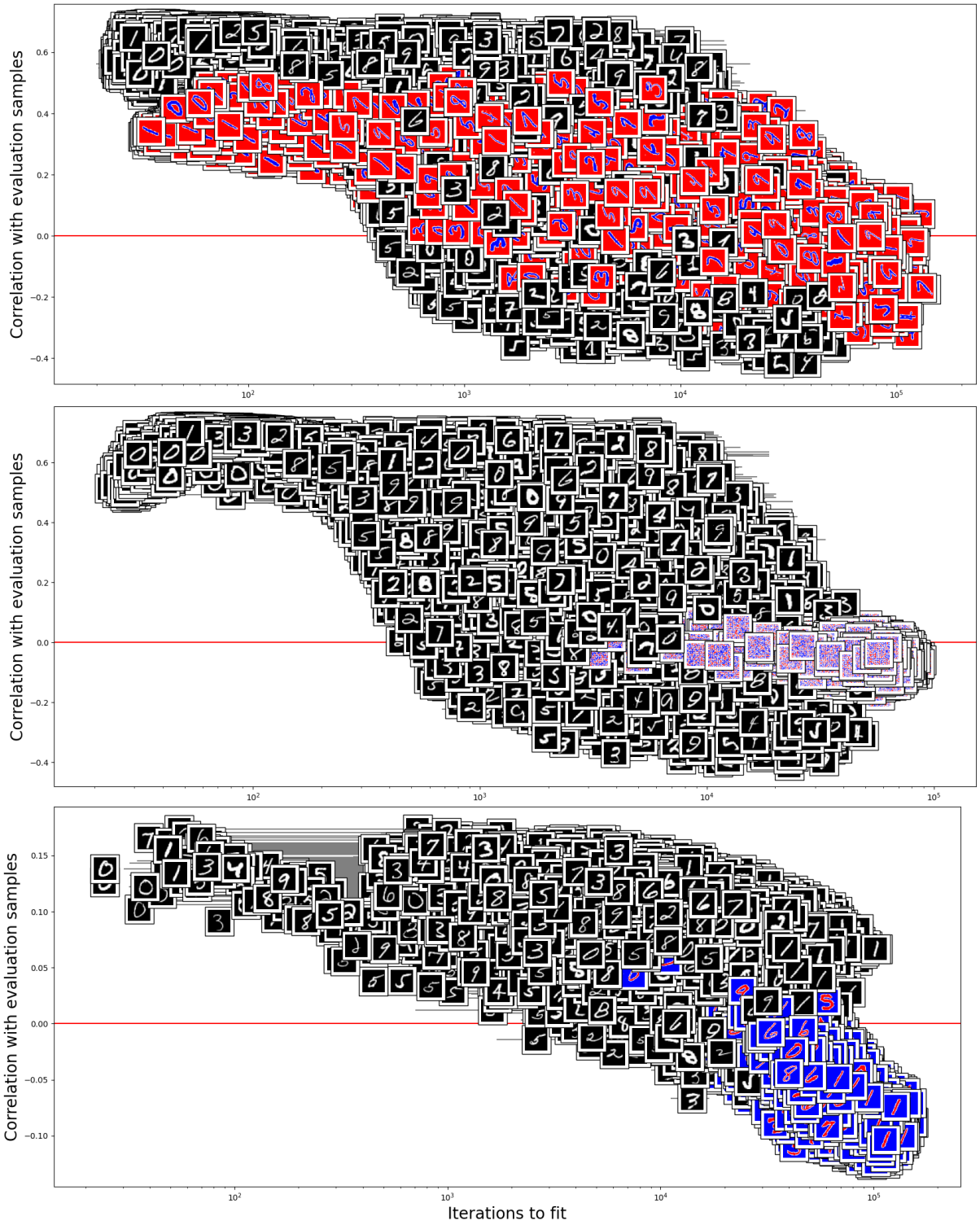

Figure 6.3: Corrupted MNIST sample priority. (top) MNIST samples with 50% structured input corruption. (center) MNIST samples with 50% Gaussian input corruption. (bottom) MNIST samples with 50% label corruption. Note that the alternative colormap for corrupted samples is just for visual distinction.

specifically, it is worth noting that the correlation tends to be negative. This suggests that when features are being fitted to reduce the loss on label corruption, it has a definite adverse effect on the model's ability to generalize.

A final observation, which is also unique to the label-corrupted dataset, is that a large portion of the syntactically similar clean samples that are usually fitted early are being fitted very late at approximately the same time that the label corruption is being fitted. This appears to suggest that our idea of similarity and sample priority is flawed. However, in the next section we will show that these samples are being fitted early and then re-fitted towards the end when the label corruption is optimized for.

It was suggested that it makes more sense to measure $g(s)$ in Eq. 6.2 in terms of the class to which sample s belongs. In Appendix A we repeat the results presented in this subsection on a class-specific basis. All our observations remain valid.

6.4 On refitting samples

In this section we investigate to what extent samples are refitted and how this affects the generalization performance of the model being trained. To do this, we make use of graphs indicating the sample refitting dynamics for several of the models trained in the previous section. See Fig. 6.4 for an example. The black curve indicates the evaluation error at each iteration. The other metrics are defined below (with clean and corrupted samples being represented by the same metrics but on separate plots):

- *newfit count* is the total number of training samples that are correctly classified in the current sampled iteration, while never having been correctly classified at a prior sampled iteration.
- *unfit count* is the total number of training samples that were correctly classified in the preceding sampled iteration and incorrectly classified in the current one.
- *refit count* is the total number of training samples that were incorrectly classified

in the preceding sampled iteration and correctly classified in the current one, while already having been correctly classified at a prior sampled iteration.

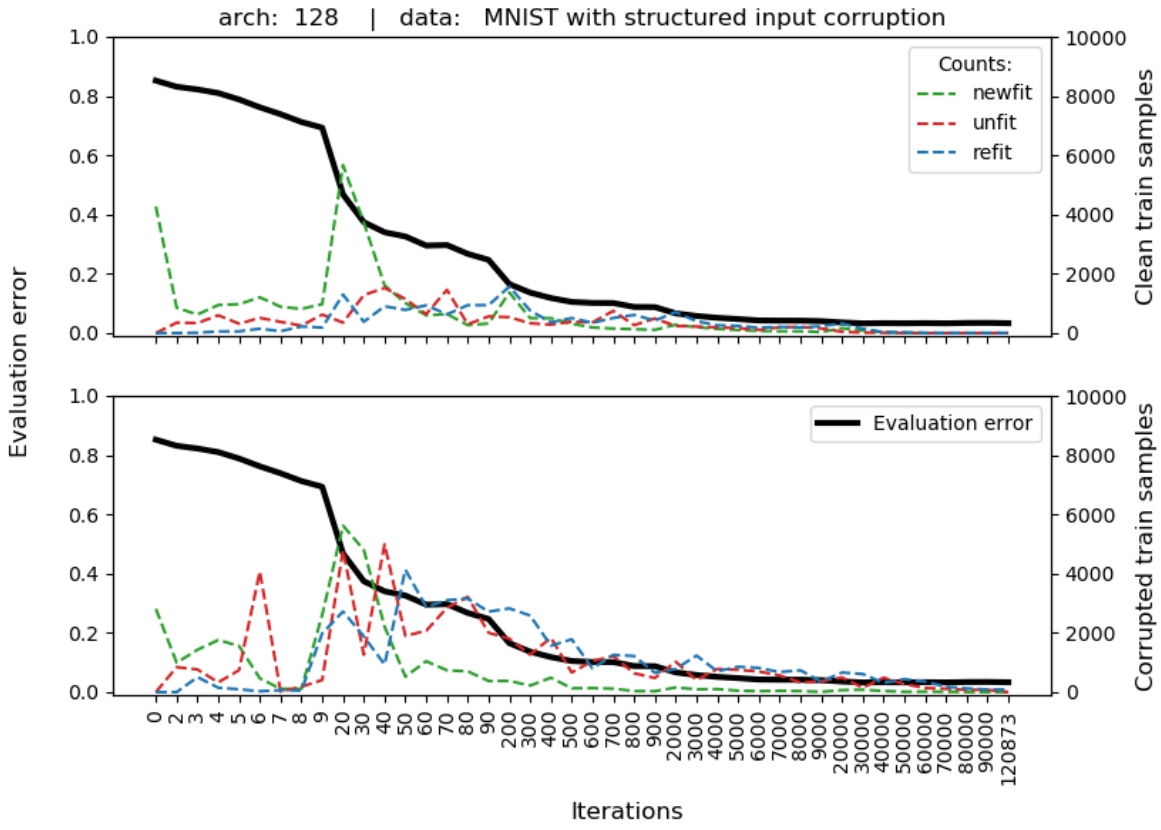

Figure 6.4: Evaluation error (left vertical axis), and several metrics regarding train sample refitting dynamics (right vertical axis) as a function of sampled (log-scaled) training iterations.

This particular graph corresponds to one of the models trained on MNIST with 50% structured input corruption. For brevity we will not present the graphs for other models trained on clean MNIST and MNIST with structured input corruption. All of these models produced graphs with similar trends to the graph presented in Fig. 6.4, with the obvious exception that clean MNIST shows no corrupted counts. Most samples are fitted and refitted very early, after which very few samples are refitted as generalization stabilizes.

Fig. 6.5 shows the refitting dynamics for a selection of the models trained on MNIST with 50% Gaussian input corruption. The model initializations with the lowest evaluation error at interpolation are selected. Notice that, in contrast to Fig. 6.4, we see that the samples with Gaussian input corruption are still being fitted well beyond the point where

the generalization has stabilized. Importantly, the clean samples do not undergo any significant refitting during the later stages of training.

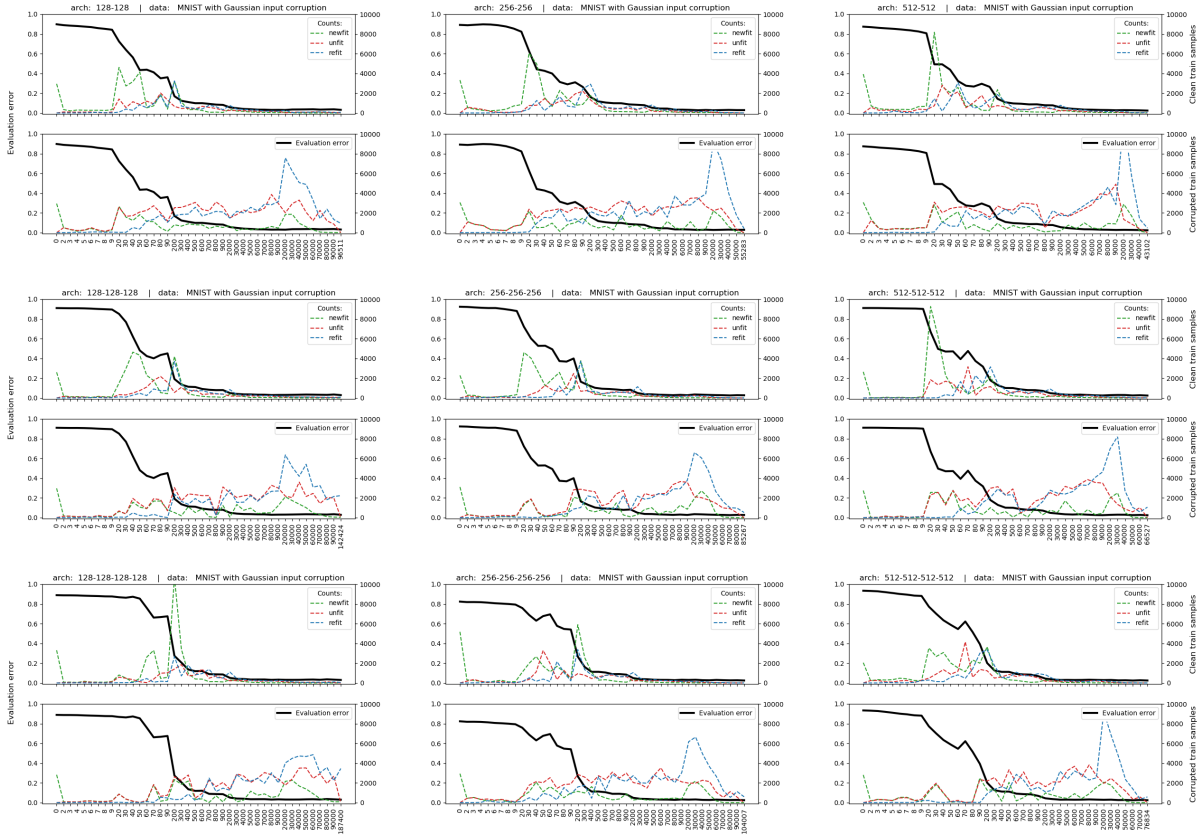

Figure 6.5: Refitting dynamics of models trained on MNIST with 50% Gaussian input corruption. From left to right models have hidden-layer widths of 128, 256, and 512. From top to bottom models have depths of 2, 3, and 4.

The corresponding refitting dynamics of a selection of the label-corrupted models, from the previous section, are presented in Fig. 6.6. The model initializations with the lowest evaluation error at interpolation are selected. Take note that in accordance with the clean, structured input corrupted, and Gaussian input corrupted models, the clean samples are fitted and refitted early in training. However, there then appears to be a period of very few refittings, with the clean samples being refitted at a slightly higher frequency than the corrupted samples. At the point where the generalization starts to degrade, we observe that there is a very noticeable increase in new corrupted samples being fitted and refitting of both clean and corrupted samples. This behavior is different to any of the clean, structured input corrupted, and Gaussian input corrupted models, in that a

large portion of the clean samples are being refitted very late and we see a corresponding increase in evaluation error.

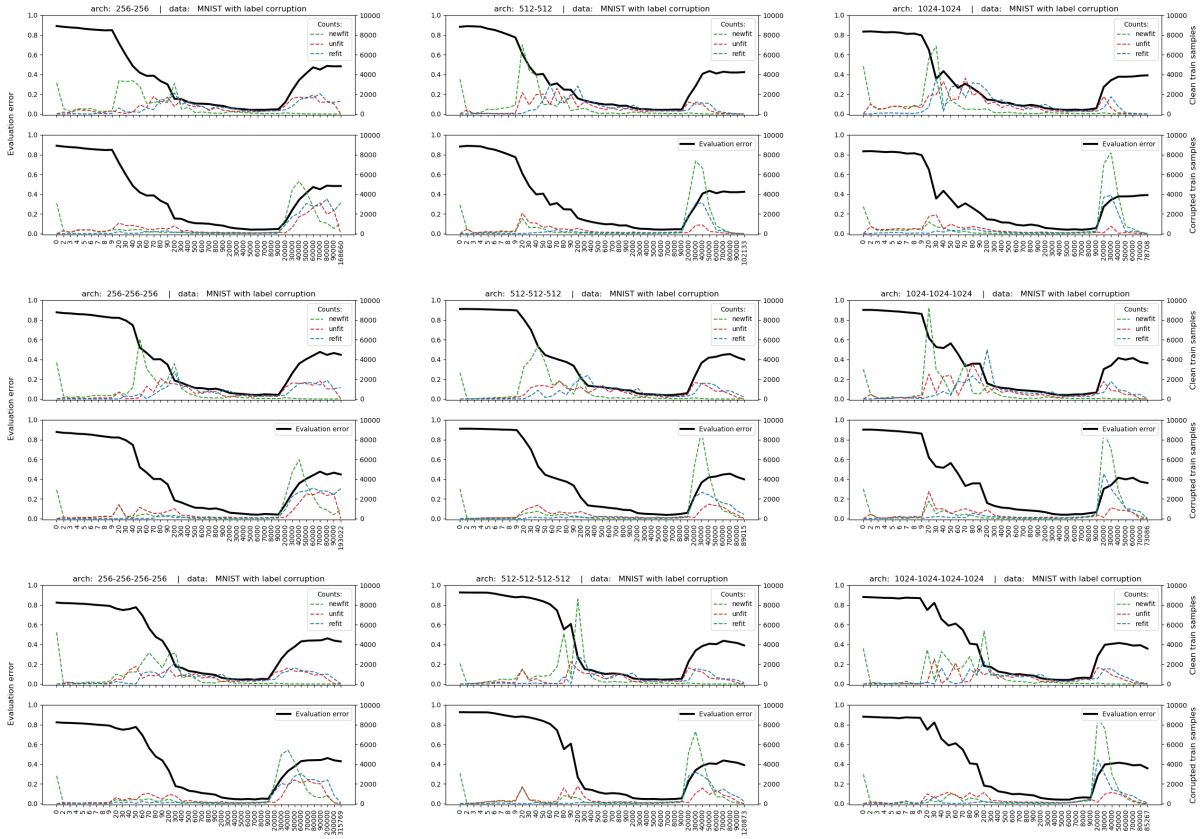

Figure 6.6: Refitting dynamics of models trained on MNIST with 50% label corruption. From left to right models have hidden layers widths of 256, 512, and 1024. From top to bottom models have depths of 2, 3, and 4.

So what differentiates the type of refitting that does not negatively affect generalization (see Fig. 6.4 and 6.5) and the type that does (Fig. 6.6)? It is evident from all three figures that refitting clean samples early in training does not degrade generalization, in fact, it tends to improve it.

Refitting Gaussian input corrupted samples tends to have little effect on the generalization performance of our models. This is consistent with the idea that the ReLU activation functions enable the subunits optimized for corrupted samples to not activate when processing the uncorrupted evaluation set. This is reflected in Fig. 5.4 where the subunits of models trained on datasets with a portion of Gaussian input corruption shows the highest level of polarization between clean and corrupted samples, when compared to models

trained with the other two types of corruption.

In all of our graphs of refitting dynamics for label-corrupted models there is a period of very low *refit count* followed by a clear "spike" in clean sample refittings at the iterations where generalization degrades. This is consistent with the idea of sample priority and coherent gradients introduced in Section 6.3. The highly coherent gradients from the clean samples dominate initially and need to be reduced to near-zero in order for the non-coherent gradients from the label-corrupted samples to have a relatively significant effect on the overall parameter updates.

While the gradients from both Gaussian input corrupted and label-corrupted samples are expected to have low coherence, the label-corrupted samples are not as easily separable from clean samples as Gaussian corrupted samples are. We suspect that this is because the Gaussian corrupted samples have very dissimilar input feature structures to any of the clean samples but the label-corrupted samples still maintain similar structures to the clean samples even though the classes are assigned randomly. This is also supported by Fig. 5.4 where the subunits of models trained on datasets with a portion of label corruption shows lower levels of polarization between clean and corrupted samples, especially in the shallower layers.

In summary: Sample refitting that occurs early in training tends to be governed by coherent gradients from samples with similar input feature structures and is usually part of the function fitting process. When there is a relatively large increase in refitting, late in training, and a model is not able to separate (modularize) useful features from others, this can be destructive to the generalization performance.

Before continuing we would like to address our use of the log-spaced sampling of training iterations. By not considering every iteration the metrics we use to visualize the refitting dynamics are approximated values. This is especially true for the later iterations. As a sanity check, in Fig. 6.7, we present the refitting dynamics for a label-corrupted model where all of the training iterations are considered. See Appendix C.5 for details on the hyperparameters used to train this model.

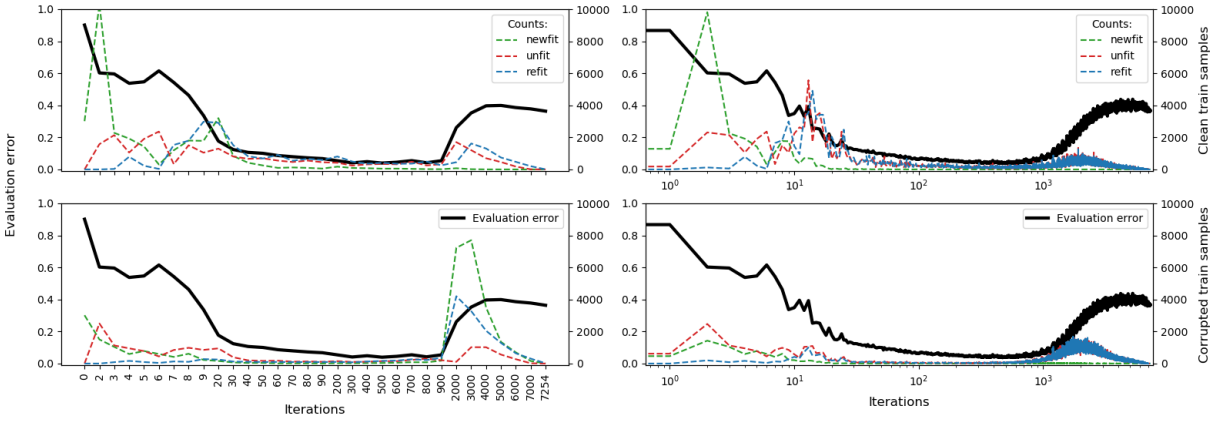

Figure 6.7: Refitting dynamics of a 3×1024 model trained on MNIST with 50% label corruption. A log-spaced sampling of iterations are considered in the left-hand plots, and all iterations are considered in the right-hand plots.

Notice, from the right-hand plots, that we still observe that unfit and refit counts increase noticeably when the generalization error increases. However, the newfit counts for corrupted samples appear to be almost negligible in contrast to our approximation (in left-hand plots). This discrepancy is caused by the fact that newfit counts are unique, meaning that each sample in the train set will only contribute one newfit count throughout training (the area under the green curve is always equal to the total number of clean or corrupted samples in the set). When using our approximations the "spike" in newfit counts is analogous to adding the unique newfit counts that occurred between the last sampled iterations and the current one. While we use "snapshots" of the refitting dynamics, the trends we observe are, therefore, still valid.

6.5 Epoch-wise double descent

In the previous two sections we showed that similar samples are consistently prioritized, and why. We also showed that with significant inter-class overlap in the input space (of which label corruption is an extreme example), a large portion of the clean samples are refitted later in training and this could degrade generalization. In this section we will show how epoch-wise double descent occurs when a sufficient portion of the subunits is unaffected by this refitting phenomenon.

6.5.1 Sensitivity to class corruption

For label-corrupted models, we expect that the weight vectors for the subunits that are least attuned to the label corruption right before the label corruption is fitted will be least affected by the drastic change in representation. That is because those weight vectors would have been fitted by the coherent gradients from the clean samples. To measure the sensitivity of a weight vector to label corruption we use a weighted sum of recall values over classes (after label corruption). This metric, named *sensitivity to class corruption*, is defined for a subunit j in Eq. 6.3.

$$s_{\text{corr}}(j) = \sum_{k \in K} p_{\text{corr}}(j,k) r_{\text{corr}}(j,k), \tag{6.3}$$

where K is the set of all classes, and $p_{corr}(j,k)$ and $r_{corr}(j,k)$ are the precision and recall of the node j w.r.t corrupted class k samples, respectively. These two values are defined in Eq. 6.4.

$$p_{\text{corr}}(j,k) = \frac{|S_j^{k_{\text{corr}}}|}{|S_j|}; \quad r_{\text{corr}}(j,k) = \frac{|S_j^{k_{\text{corr}}}|}{|S^{k_{\text{corr}}}|}, \tag{6.4}$$

where $S_j^{k_{\text{corr}}}$ is the set of corrupted samples from class k for which node j is activated, S_j is the sample set of j, and $S^{k_{\text{corr}}}$ is the set of all corrupted samples from class k. Note the similarity between these two equations and equations 4.1 and 4.2. Where the equations in Chapter 4 measure the general sensitivity to classes, the equations used here measure the sensitivity to the corrupted instances of classes, specifically. We use this sum of recall values over classes because measuring the sensitivity by a simple ratio of corrupted samples in the relevant sample set is generally uninformative. The global model is trained to differentiate between classes, not between corrupted and uncorrupted samples. We weigh the contribution of each class by the precision value so that the relevance of the class to the node is proportional to how much the node selectively activates for the corrupted instances of that class. A high *sensitivity to class corruption* means that the relevant node is both sensitive and selective w.r.t the corrupted instances of class samples, particularly.

In Fig. 6.8 we present the sensitivity to class corruption for a label-corrupted model. Note that very early in training the sensitivity seems to be random and then some subunits become more sensitive and others less sensitive as optimal generalization is approached. This transition appears somewhat systematic with highly sensitive subunits generally remaining sensitive around the point of optimal generalization and beyond. While informative, our main interest in this metric is in using it to determine the order of subunits during analysis and visualization. In the next subsection we will investigate the changes in weight vectors for subunits that are sensitive or insensitive to label corruption as defined in this section.

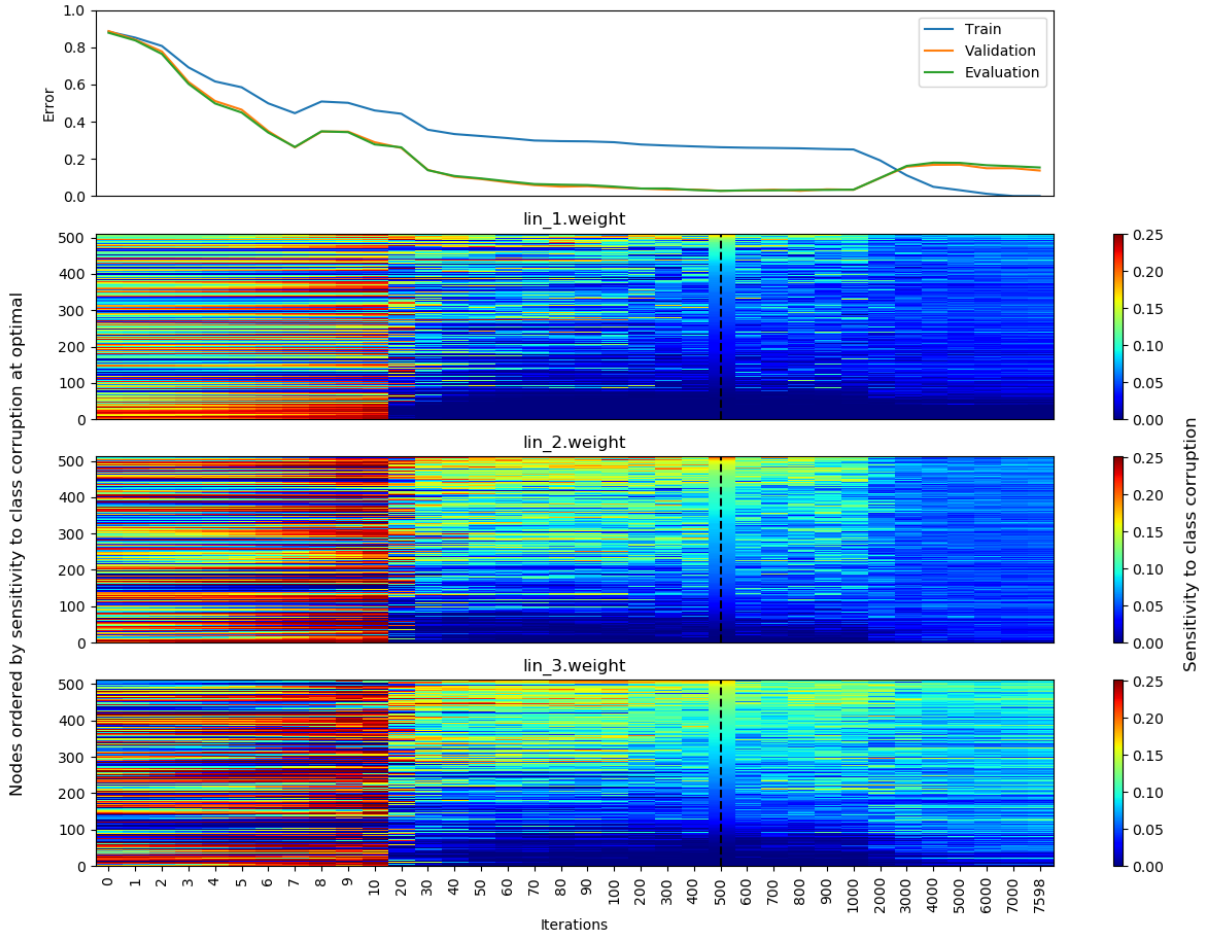

Figure 6.8: Sensitivity to class corruption for all the subunits in a 3×512 model trained on a 25% label-corrupted MNIST train set, throughout training. The learning curves are presented in the top panel and nodes in the the heatmaps have been ordered by their value at the iteration where optimal generalization is obtained. This is indicated by the black dotted line. See Appendix C.5 for details on the experimental setup.

6.5.2 Changes in weight vectors

In this subsection our goal is to show how some subunits, in an overparameterized ANN, are less affected by the refitting phenomenon discussed in the previous section, and how these subunits tend to be those that are insensitive to label corruption. For the same model depicted in Fig. 6.8 we plot the cosine similarity between each weight vector and the corresponding weight vector at the optimal iteration. This is presented in Fig. 6.9. We order the subunits in each hidden layer by their sensitivity to class corruption in ascending order. We limit the colormap to cosine similarity values between 0.6 and 1.0. This results in any value below 0.6 being depicted as 0.6. We empirically observed that instances below this value are rare, and are usually masked by the interpolation used to plot the heatmaps with a manageable number of pixels.

Take note that many weight vectors are fairly close to their optimal state very early in training, indicating that they do not change much while generalization is still improving. Most of the weight vectors change drastically after the first ascent in generalization error, often being more dissimilar to the optimal than they were at initialization. The most important result, we would like to highlight, is that the weight vectors that correspond to subunits with the least sensitivity to class corruption (lower-most nodes) tend to remain fairly close to their optimal value after generalization degrades. This indicates that the subunits that are fitted early by coherent gradients from clean samples maintain their optimal states even when the refitting phenomenon changes most of the nodes sensitive to class corruption. In the next subsection we will repeat this analysis on several models of varying widths to clearly show the connection between these highly consistent subunits and the epoch-wise double descent.

6.5.3 Refitting, capacity, and double descent

From Fig. 6.6 we see that larger models tend to display the epoch-wise double descent phenomenon more convincingly. This has been observed previously [48]. To investigate the change in subunit behavior at varying levels of epoch-wise double descent we trained

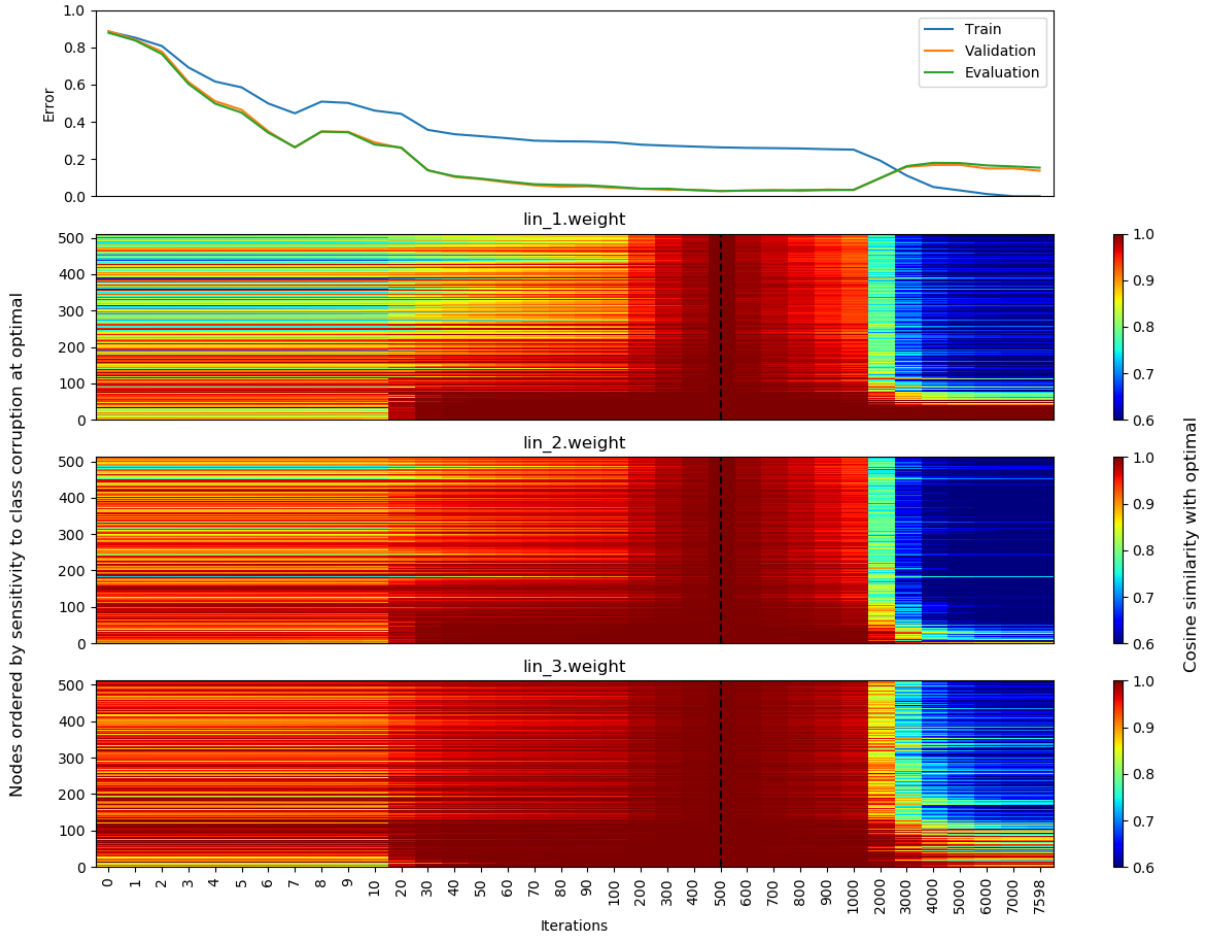

Figure 6.9: Cosine similarity between weight vectors and their corresponding state at the iteration where optimal generalization is obtained. This model corresponds with the one presented in Fig. 6.8 and the nodes are ordered identically.

eight models, for which the resulting performances are presented in Fig. 6.10. The four smallest models (widths of 16, 32, 64, and 128) were unable to interpolate the entire train set. Take note of the poor generalization performance around the 3×128 model. This architecture obtained a training error of 0.00022 and was, therefore, just barely insufficient to fit the train set completely. This is typical of model-wise double descent.

We first look at the changes in weight vectors for the four models with insufficient representational capacity to interpolate the train set. This case is presented in Fig. 6.11. We see that the weight vectors in these models at initialization tend to be further away from their optimal state than for the larger model in Fig. 6.9. This is likely because, with limited capacity, more changes are necessary to find an optimal partitioning of the training

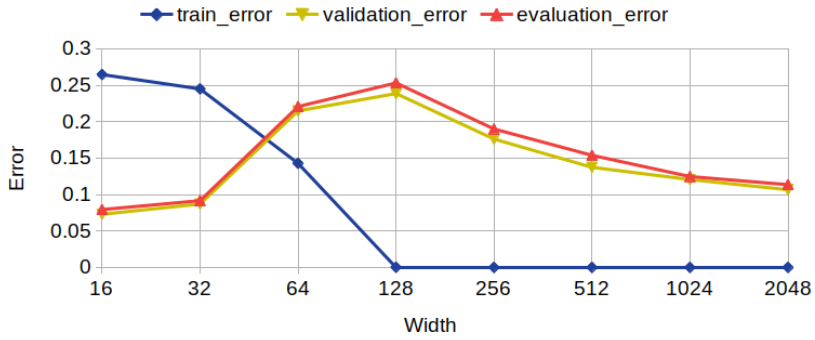

Figure 6.10: Performance at interpolation (or 500 epochs) for eight models with three hidden layers and varying width, trained on MNIST with 25% label corruption. See Appendix C.5 for details on the experimental setup.

data. We also see that a large portion of the weight vectors in the two smallest models stay close to their optimal value after the best generalization is obtained. Correspondingly, we do not see much degradation in the generalization performance in later training. This is likely because these models are unable to fit much of the label corruption and little refitting occurs. See from Fig. 6.10 that the final train errors for these two models are near 25%. For the larger two models label corruption is definitely being fitted (see Fig. 6.10) and, as expected, the cosine similarities after the optimal iteration are generally much further away from their optimal state. This is reflected in the rise in evaluation error beyond that point.

Next we look at the four models with sufficient representational capacity to interpolate the train set. This is presented in Fig. 6.12. Here, the opposite trend occurs. The later weight vectors in the smaller two of the four models tend to be much further away from their optimal state. For the two larger models we see more weight vectors remaining close to their optimal state, and a reduced increase in generalization error and more convincing second descent. In all four of these models there is a clear trend for the weight vectors corresponding to subunits that are the least sensitive to label corruption, to obtain their optimal state early and remain closer to their optimal state when label corruption is fitted.

In the widest of the eight investigated models we see that the cosine similarities tend to be very high early on and then reduce drastically during overfitting. To account for this, we propose that the wider models do not need to change any given weight vector too

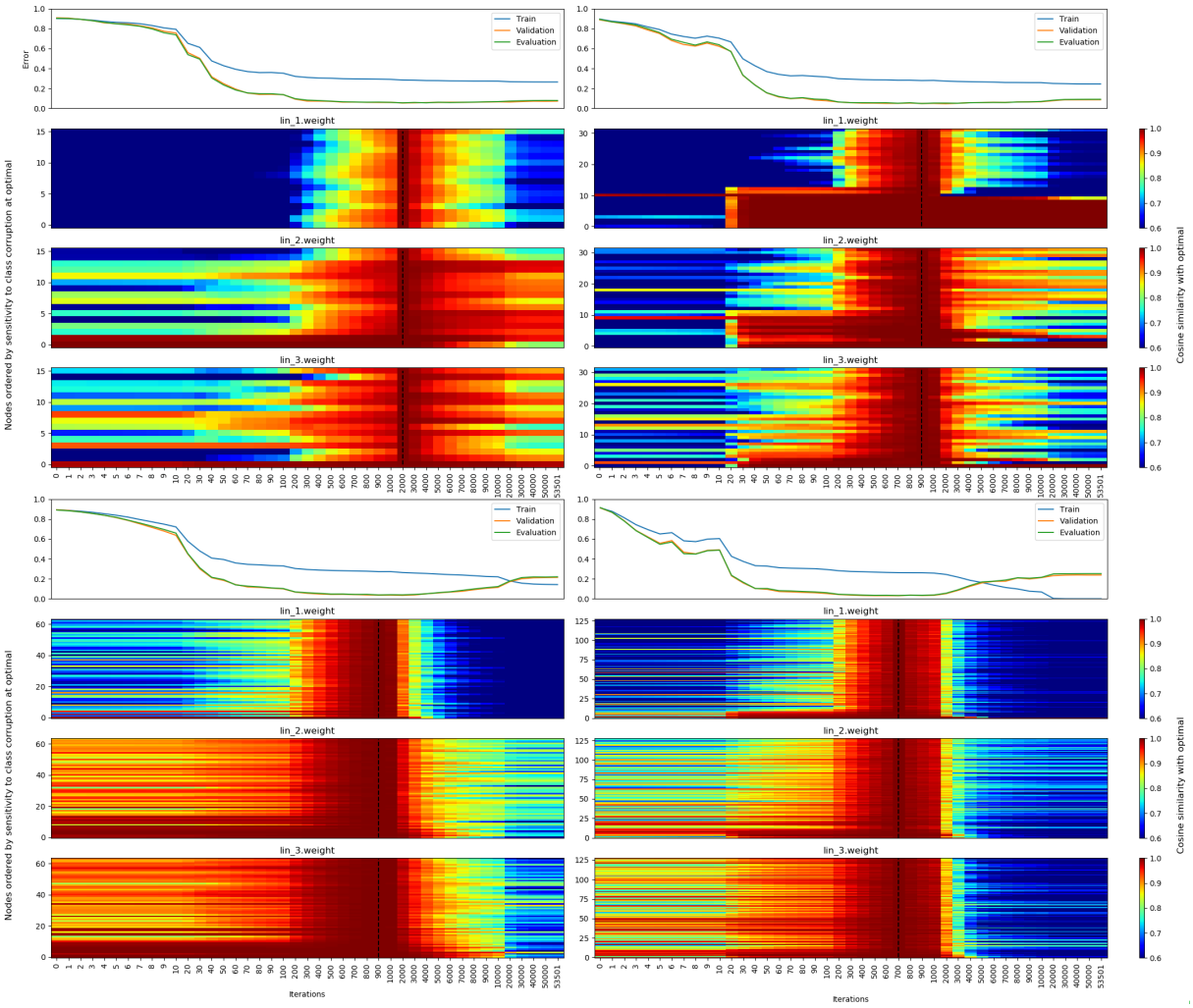

Figure 6.11: Cosine similarity between weight vectors and their corresponding state at the iteration where optimal generalization is obtained. From left to right then top to bottom each panels refers to the model with a width of 16, 32, 64, and then 128 from Fig. 6.10.

much in order to fit the simple early features that tend to contribute to generalization. Only when the more complex features are fitted to reduce the loss on label corruption do the weight vectors need to be changed to a significant degree.

These findings strongly suggest that the reason for the epoch-wise second descent is that overparameterized models have the parametric flexibility to separate the non-generalizable features that cause samples to be refitted, from those that generalize. Therefore, the good representations found early are less likely to be destroyed. After refitting occurs, these generalizable features continue to be incorporated in the function approximating process which can lead to slight improvement again. The modular fitting hypothesis provides an

answer as to how it is possible for some representations found earlier to remain unchanged throughout the epoch-wise ascent and descent.

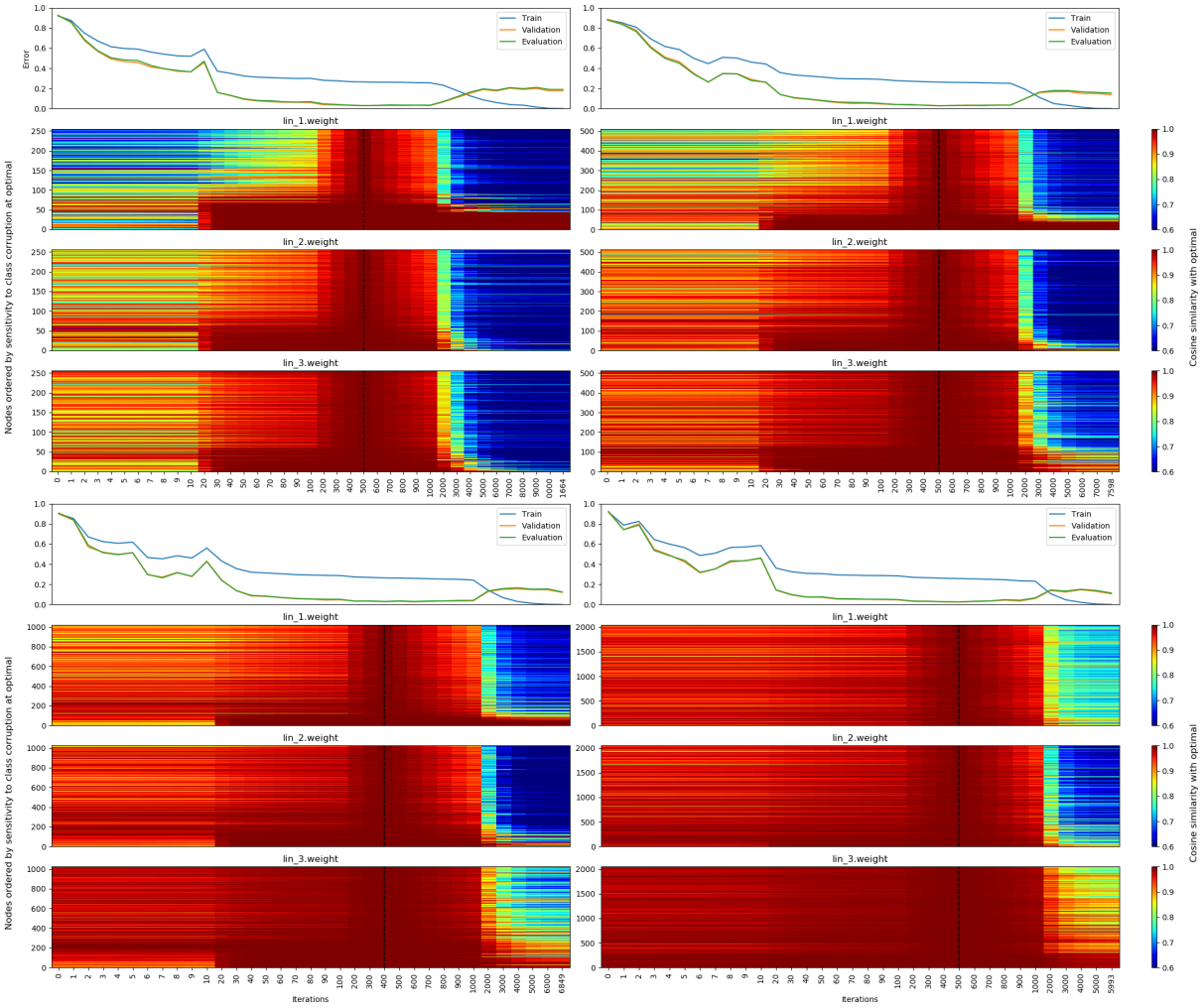

Figure 6.12: Cosine similarity between weight vectors and their corresponding state at the iteration where optimal generalization is obtained. From left to right then top to bottom each panels refers to the model with a width of 256, 512, 1024, and then 2048 from Fig. 6.10.

6.6 Model-wise double descent

In order to address the question of why double descent occurs we have had to describe and motivate several aspects of the way ANNs fit natural datasets when optimized with gradient based learning. A brief summary follows:

- ANNs are biased to make parameter updates that would reduce the loss of many training samples simultaneously. This is enabled by samples having common features, activating the same subunits, and gradients overlapping to produce large steps in parameter space. At initialization, only samples that have many features in common will produce coherent gradients. This is why ANNs, with different random initialization and architectures, prioritize similar samples early in training. A side-effect of the ubiquity of these shared features is that they are more likely to generalize.

- After the early coherent gradients have been sufficiently reduced, features that do not result in gradients that overlap as much begin to affect parameter updates. If the samples containing these features are sufficiently dissimilar to most of the already fitted samples, the modularity of the hidden representations allows them to not affect the already fitted features. However, if they are similar enough to cause extensive inter-class overlap in the input space, the resulting parameter updates can detrimentally change the representations found earlier. This is indicated by the refitting phenomenon.

- With increased representational capacity fewer alterations are necessary to accommodate the more difficult samples, leaving more subunits closer to their early state. This is why epoch-wise double descent occurs.

Model-wise double descent can be seen as the natural result of the behavior explained above. Models with very limited representational capacities can only fit features from coherent gradients and as a result no signs of classical "overfitting" are seen. Models closer to the critically parameterized regime are able to fit more features from non-coherent gradients resulting in potentially detrimental changes in earlier representations. Moving far enough beyond the critically parameterized regime allows more of the earlier representations to remain intact. Hence, the model-wise second descent. Using Fig. 6.13 as reference we define four distinct regimes of generalization behavior based on our findings in this chapter.

- **A**: *hypoparameterized* regime. The model's representational capacity is only large enough to fit features from early coherent gradients. Any increase in capacity (while still remaining in this regime) will result in improved generalization upon convergence. This is because more samples will be able to be fitted and a richer representation of the true data distribution will be obtained. Any form of early stopping is unlikely to improve performance. Models in this regime are identifiable by the fact that they are unable to interpolate the train set and show no signs of a classic U-shaped overfitting curve in out-of-sample risk as a function of training iterations.

- **B**: *pre-critically parameterized* regime. The model's representational capacity is sufficiently large to fit features from early coherent gradients and some features from non-coherent gradients. Adding more capacity can reduce generalization as more and more features from non-coherent gradients are able to be fitted, at the cost of earlier representations. Any form of early stopping is likely to improve performance for models in this regime, as optimal generalization tends to be obtained before features from non-coherent gradients begin to be fitted. Models in this regime are identifiable by the fact that they are unable to interpolate the train set and show strong signs of a classic U-shaped overfitting curve in out-of-sample risk as a function of training iterations.

- **C**: *post-critically parameterized* regime. The model's representational capacity is sufficiently large to fit features from early coherent gradients and enough features from non-coherent gradients to fit the entire train set. Adding more capacity will improve generalization as larger models will be better able to fit the features from non-coherent gradients without affecting the earlier representations. For the same reason as in the previous regime, models from this regime will usually gain from early stopping. Models in this regime are identifiable by the fact that they are able to interpolate the train set and show strong signs of a classic U-shaped overfitting curve in out-of-sample risk as a function of training iterations.

- **D**: *hyperparameterized* regime. The model's representational capacity is sufficiently large to fit features from early coherent gradients and enough features from non-coherent gradients to fit the entire train set without affecting earlier representations.

Increases in capacity still tend towards improved generalization, albeit without linear returns. Early stopping is not required in this regime, because generalization improves monotonically. Models in this regime are identifiable by the fact that they are able to interpolate the train set and show no signs of a classic U-shaped overfitting curve in out-of-sample risk as a function of training iterations.

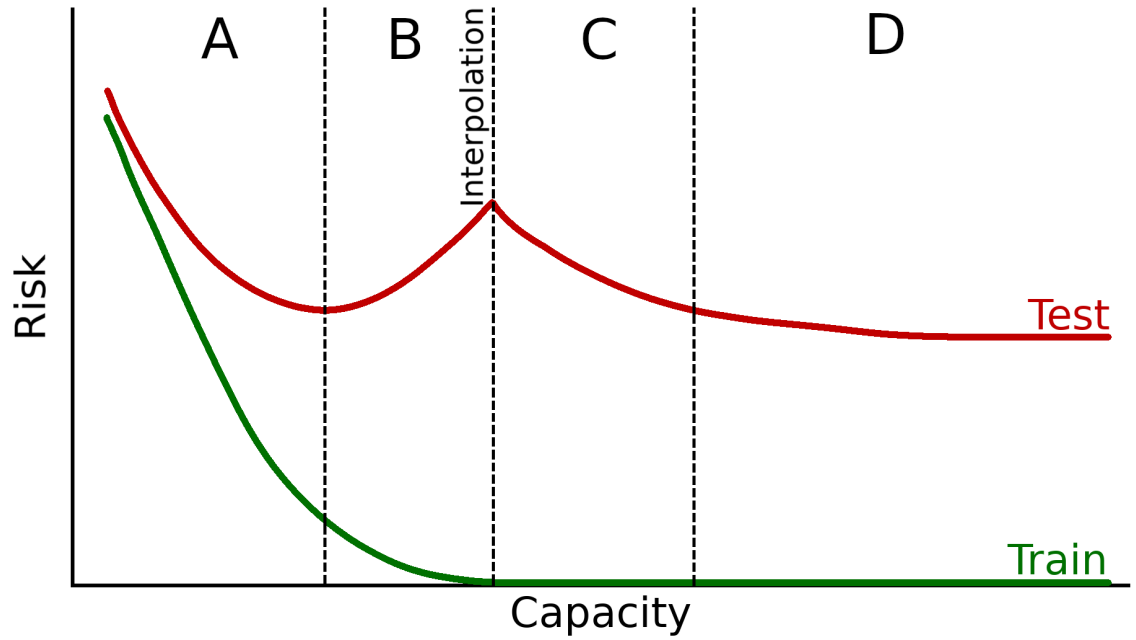

Figure 6.13: Illustrating four regimes of representational capacity.

As empirical evidence for these four regimes, we trained numerous single hidden layer MLPs at varying layer widths, on three datasets. We observe the validation error as a function of the hidden layer width as well as training epochs. While these four regimes are part of a continuum it is easy to see when a model strongly embodies a specific regime.

Fig. 6.14 shows these curves for models trained on CIFAR10. Notice the slight but visible critically parameterized regime around a width of 80. The five smallest architectures (2, 4, 6, 8, and 20) appear to be in the hypoparameterized regime. This can be seen by the downwards slope in the top graph and the corresponding monotonically improving curves in the bottom graph. The next three smallest architectures (40, 60, and 80) are in the pre-critically parameterized regime, as indicated by the rise in validation error over epochs.

The architectures with a width of 200 and 400 are in the post-critically parameterized regime, with the latter model only showing the slightest increase in validation error over epochs. The rest of the architecture are clearly in the hyperparameterized regime. These models again show monotonically improving generalization error over training epochs.

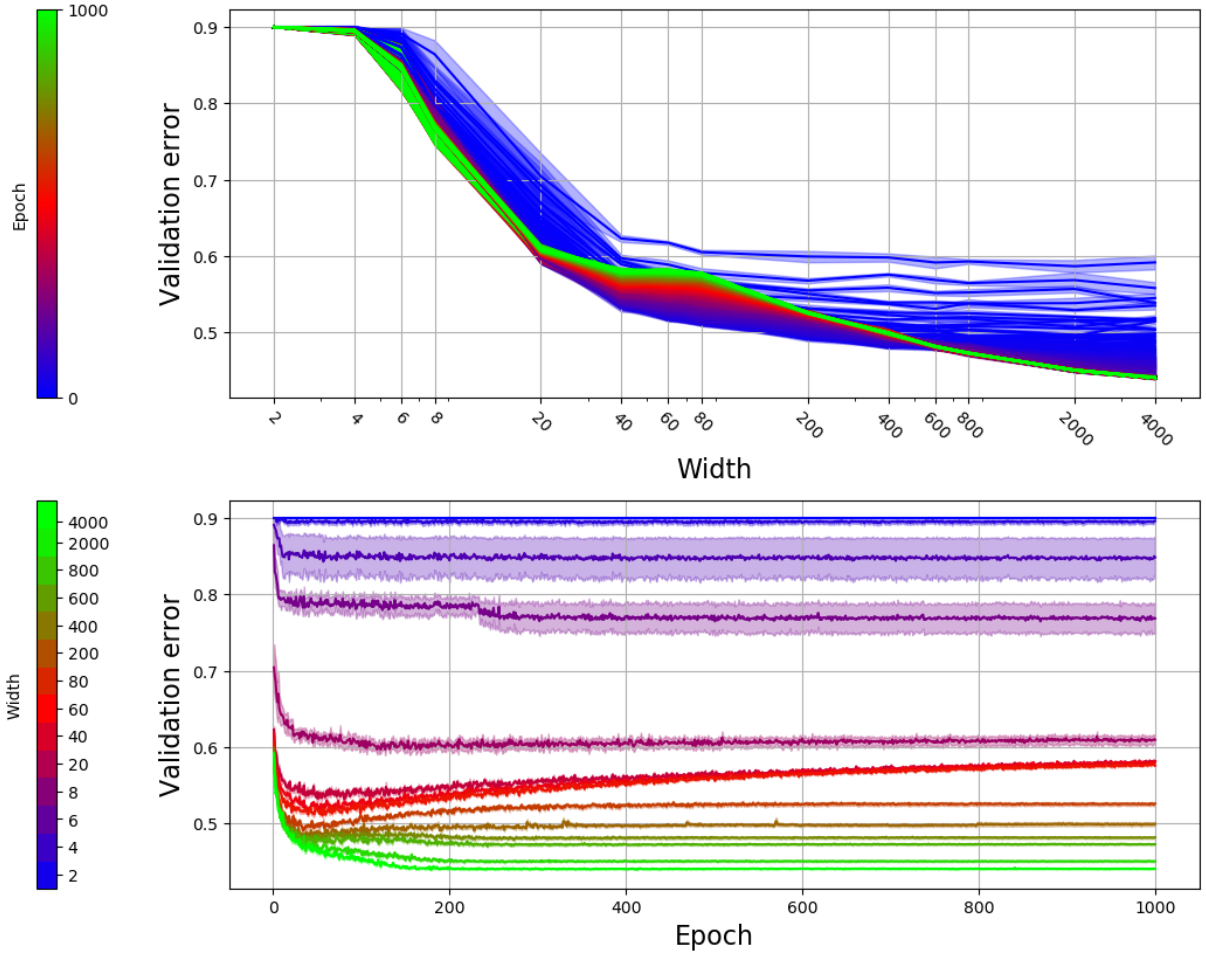

Figure 6.14: CIFAR10: Validation error curves as a function of hidden layer width (top) and training epochs (bottom). The colors represent the training epochs and hidden layer widths in the top and bottom plots, respectively. Note that all measures are averaged over several random initializations. See Appendix C.5 for details on the experimental setup.

Fig. 6.15 shows the validation error for models trained on MNIST. There is no visible model-wise double descent for this dataset. This is unsurprising when considering how little the training samples overlap in the input space: as mentioned in Appendix C.2 a large part of MNIST is linearly separable. It is also predicted by the lack of refittings that occur late in training for models trained on clean MNIST data. At first it appears

that these models jump from the hypo- to the hyperparameterized regime. However, if we zoom in on the largest models (see the bottom plot in Fig. 6.15) we can still see signs of the pre- and post-parameterized regimes, albeit to a much lesser extent.

Fig. 6.16 shows the validation error for models trained on FMNIST with 20% label corruption. As is typical with the addition of label noise, there is a very clear model-wise double descent curve. As expected, we also see instances of epoch-wise double descent for the larger models (see the bottom plot). While the first three regimes are clearly visible, we do not observe a model in the hyperparameterized regime. Instead, the larger models seem to approach the hyperparameterized regime asymptotically: the rise in validation error, over epochs, is lower and lower with the addition of more hidden units, but the improvement is also reduced with each jump in width. The $1 \times 10\,000$ model is the largest we are able to analyze under our computational constraints.

Fig. 6.17 provides a compilation of the validation errors (left) and corresponding average sample validation loss (right) for the three datasets shown in Fig. 6.14, 6.15, and 6.16. As mentioned in Section 6.2 the model-wise double descent is much clearer in average validation loss, even for clean MNIST samples. We also see that there is no visible instance of epoch-wise double descent in average validation loss.

6.7 Discussion

Seeing as we have already recapped and discussed the findings in this chapter in the previous section, and also the fact that the next chapter serves as a "wrap up" of previous chapters, we will omit a dedicated discussion section for this chapter.

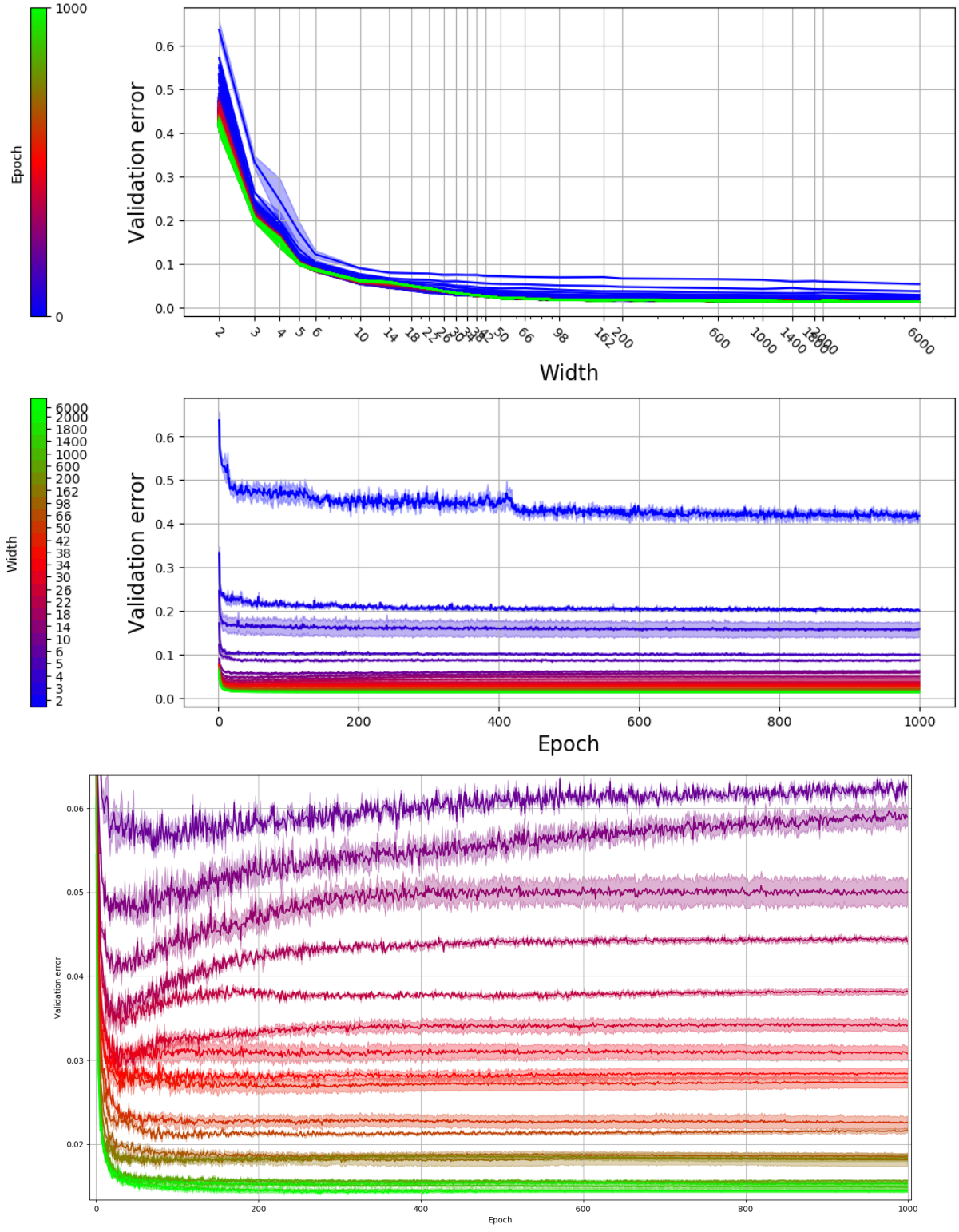

Figure 6.15: MNIST: Validation error curves as a function of hidden layer width (top) and training epochs (center). The colors represent the training epochs and hidden layer widths in the top and bottom plots, respectively. The bottom plot is a cropping of the validation error over epochs for the largest models. Note that all measures are averaged over several random initializations. See Appendix C.5 for details on the experimental setup.

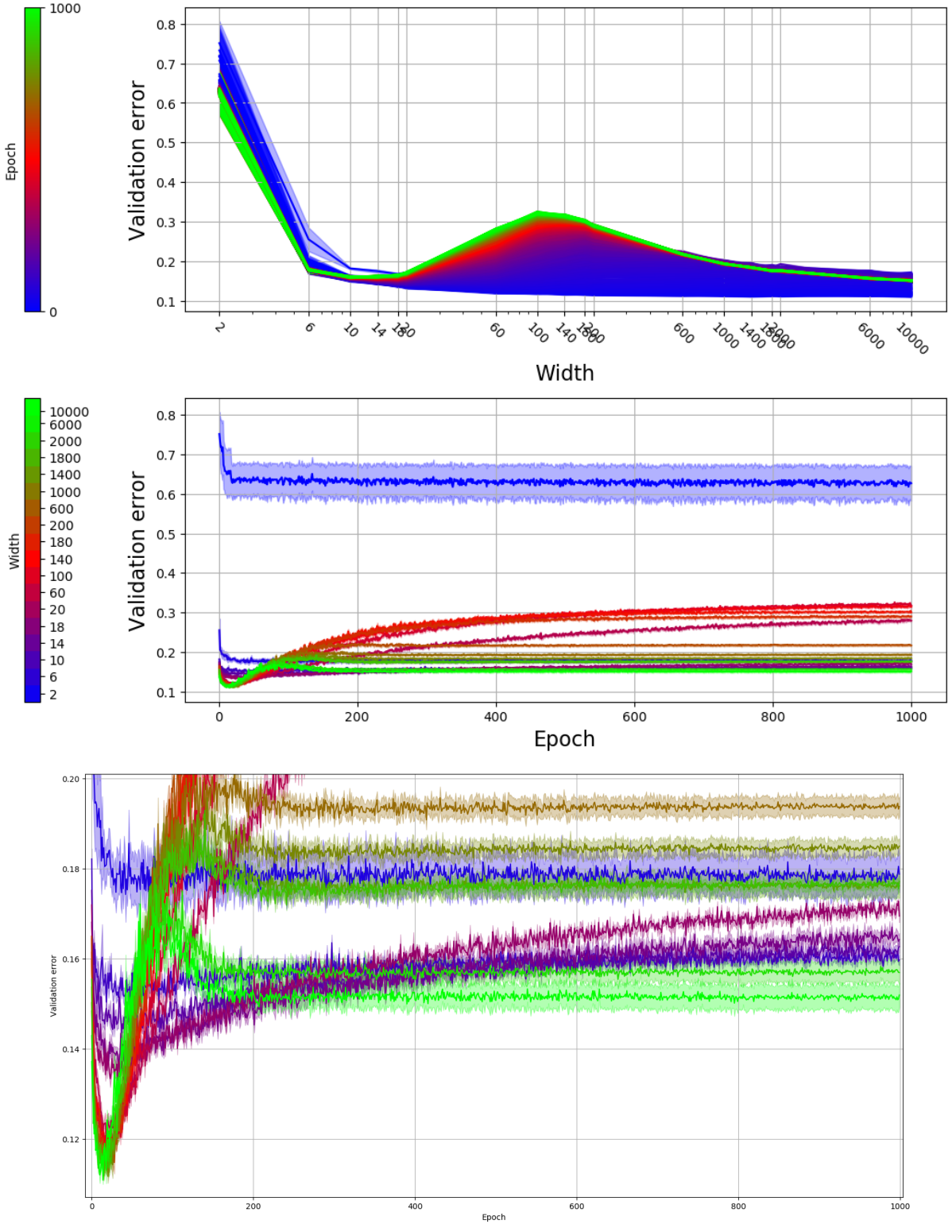

Figure 6.16: FMNIST with 20% label corruption: Validation error curves as a function of hidden layer width (top) and training epochs (bottom). The colors represent the training epochs and hidden layer widths in the top and bottom plots, respectively. The bottom plot is a cropping of the validation error over epochs for the largest models. Note that all measures are averaged over several random initializations. See Appendix C.5 for details on the experimental setup.

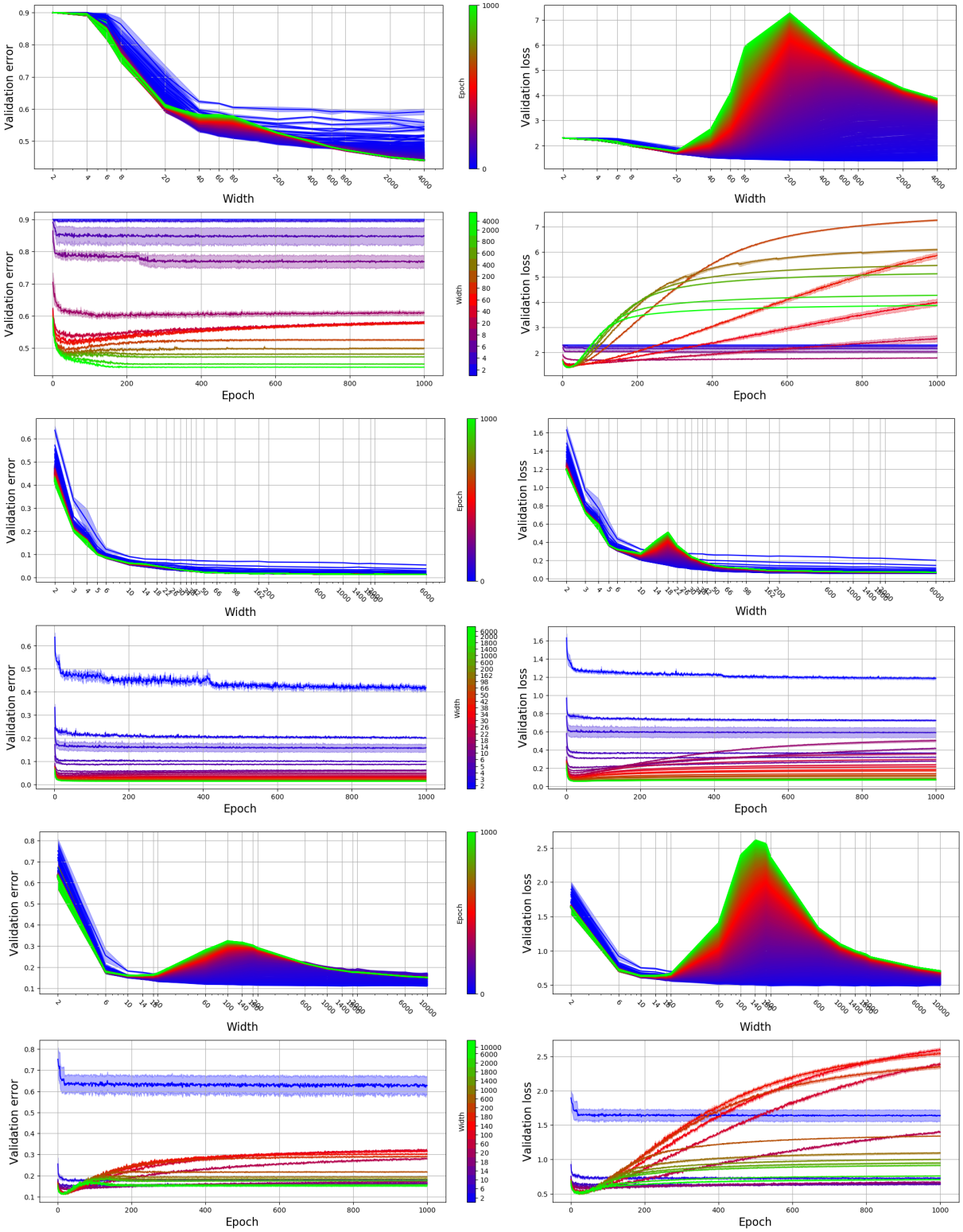

Figure 6.17: Validation error (left) and corresponding average sample validation loss (right) for the three datasets: CIFAR10 (top), MNIST (center), and FMNIST with 20% label corruption (bottom).

Chapter 7

Implications for generalization in DL

In this chapter we consolidate the findings presented in preceding chapters. We aim to clarify their relevance to the subject matter and produce a coherent perspective on generalization in MLPs w.r.t model substructures.

7.1 Overview

As stated in Section 1.2, our principal goal is to motivate the point that substructures in an ANN are underutilized in investigations of generalization in DL. We have explained that this approach is different to classical notions of generalization where the focus is on the mapping from input to output as a whole. In this chapter we aim to combine our findings into a coherent perspective of the role of substructure (by means of subunits) in MLP training and clarify how this is a promising shift in paradigms regarding generalization and DL.

As we are discussing many interrelated concepts in this chapter, it might be prudent to clearly define what we mean with *subunits* and *substructures*. We use the term "subunits"

to refer to hidden nodes with a piecewise linear activation function. We use these subunits as a method of investigating substructure in the network. Substructure is a less concrete term we use to refer to the formation of related regions of the hidden representation and the network itself. For example, some specialized subunits only activate for a select few samples in the train set. We can think of this group as a substructure within the network. Classical measures of capacity fail to distinguish such substructures from the whole.

In Section 7.2 through 7.4 we recap our findings, describing how subunits can shed light on both the learning process and a number of previously observed phenomena related to generalization in DL. In Section 7.5 we describe how this leads to a novel perspective on the role of subunits w.r.t generalization in an overparameterized MLP. In Section 7.6 we discuss how the overall proposal (placing more emphasis on substructure) could advance the field, and we propose future avenues of research to continue with this approach in Section 7.7.

7.2 Investigating subunits separately

Throughout the thesis we analyze subunit behavior by regarding them as semi-independent components. This is not how ANNs are typically analyzed. Even contemporary works tend to investigate hidden representations with metrics measured over the entire model and train set. See Chapter 2 for some notable examples.

In Section 3.3 we provide theoretical support for the perspective of subunits as distinct entities by proving that, with piecewise linear activation functions, subunits are only ever optimized to reduce the loss of a set of samples that are measurably similar. This results in a distributed optimization procedure governed by the interplay between two systems: the continuous system (each weight vector is updated according to a node-specific cost associated with a particular sample set) and the discrete system (sample set membership is identified based on feature similarity).

7.3 Investigating the learning process through subunit behaviors

By analyzing various aspects of subunit behavior throughout training, and at inference time, we note several novel systematic behaviors indicated by regularities across models with sufficient parametric flexibility. We summarize these behaviors here:

- The transition from class-ambiguous to class-specific behavior across hidden layer depth is consistent over architectures or initializations, and this transition is task-specific. This is explicitly demonstrated in Section 4.3 and 4.4, where we show that models that are deeper and wider than a task-specific threshold have very similar: generalization performance, transitions in class sensitivity across hidden layers, and transitions in mean per-layer perplexity values. The 'task' in this case refers to a specific dataset, not classification in general.

- The number of linear regions representing each class tends to be reduced to one, very early in available model depth, making later layers less instrumental in performing the classification task. In Section 4.4 we show that binary activation patterns (each corresponding to a distinct linear region) in later layers become deterministic w.r.t class membership, suggesting that earlier layers transform sample information into class information which is simply propagated by later layers. Here, the additional depth contributes little to the actual task being performed and suggests a level of redundancy which could be linked to why additional capacity does not necessarily hurt generalization ability. This, still, does not mean that excess depth does not aid in the necessary optimization process in order to achieve this level of generalization.

- Subunit activation patterns on the train set are regular enough to contain sufficient information to make strikingly accurate predictions using the equivalent evaluation set activation patterns. In Section 4.5 we demonstrate the existence of the continuous and discrete system by constructing per-layer classifiers, based on train set activation patterns, which incorporate the information available to each system. We find that such classifiers can generalize to out-of-sample data at a level comparable

to the actual model from which they are constructed. These classifiers, themselves, exhibit regularity when compared across factors such as training iterations, depth, and width.

- Subunits tend to separate task information based on feature similarity. This allows models to generalize even when interpolating non-generalizable features. In Chapter 5 we experimentally investigate the composition of sample sets under various noisy conditions. The noise is introduced probabilistically on a per sample basis. In doing so, we find that ANNs can modularize their hidden representation of training data so that some subunits only activate for certain kinds of training data and the grouping of training samples among subunits tends to separate true task information and noise, provided that the noise is sufficiently dissimilar to the true task information. This means that many subunits only model clean data and others only noisy data. In addition, if out-of-sample data is sufficiently dissimilar to the type of noise, the subunits modeling the noise are never activated. Therefore, even after completely interpolating noise during training, the model is able to generalize to out-of-sample data. We generalized this concept to define the "modular fitting" hypothesis in Section 5.5. This type of modularity in the hidden representation can typically only exist if the model is overparameterized, meaning that there are many more trainable parameters than training samples. This modularity sheds light on why overparameterized ANNs do not necessarily lead to poorer generalization and why they are able to generalize after interpolating explicit spurious sample information.

While not directly predicting generalization, these regularities empirically indicate that subunits can be analyzed as separate entities to uncover informative insights w.r.t how the hidden representation might relate to generalization.

7.4 Interpreting various phenomena through subunits

Continuing with our analyses of subunits we identify several underlying mechanisms of previously observed, but never acceptably explained, phenomena. We summarize the

main phenomena and our conclusions here:

- Different ANNs share the order with which training samples are fitted. In Section 6.3 we provide a plausible explanation for this behavior using the coherent gradients hypothesis and the modular fitting hypothesis. In essence: Gradients from input features shared by many samples overlap at initialization, resulting in those samples being prioritized. The modularity of the hidden representation results in the fitted samples remaining fitted for the rest of training.

- Epoch-wise double descent. In Section 6.2 we motivate that epoch-wise double descent tends to only manifest clearly for large models and some degree of inter-class overlap. This is usually induced with label noise. By directly measuring refitting dynamics of training samples we confirm that label corruption results in a noticeable increase in samples that tend to be learned early, being refitted later in training. In our experiments, this exclusively occurs at the iterations where the label corruption is fitted and generalization error starts to increase (see Section 6.4). By measuring the cosine similarity, in Section 6.5, between weight vectors and their state at the iteration of optimal generalization, we show that models exhibiting epoch-wise double descent have the representational capacity to maintain "general" subunits throughout the excessive refitting phenomenon.

- Model-wise double descent. Using the insights from Section 6.5, in Section 6.6 we explain that models in the critically parameterized regime fit less generalizable feature descriptors later in training at the cost of good generalizable feature descriptors already fitted. Below the critically parameterized regime models are unable to fit significant amounts of these "bad" feature descriptors and generalization improves. Above the critically parameterized regime models are able to fit these "bad" feature descriptors at less of a cost of earlier feature descriptors, and generalization improves again.

Our explanation of these three phenomena are based on the idea that models are able to modularize their hidden representation, as discussed in the previous section. If the

model was unable to separate feature descriptors, based on similarity, the reconfiguring of representations that occur during the later parameter updates would not be able to maintain the generalizable feature descriptors fitted early in training. Most literature on generalization and DL places little focus on substructure (and the resulting modularity) in ANNs, hence the fact that the link between sample priority and double descent has not been explored like this.

7.5 A perspective on subunit synergy

When considering all of these results together, a novel perspective on the role of subunits in MLPs becomes clear. It emphasizes the importance of finding subpopulations of the train set that are more or less important at different stages and within different substructures in the network. A well-optimized MLP is not one big hidden state with globally measurable properties related to complexity and generalization. Instead, it is the synergy of many subunits each trained to a specific region of the training data distribution but also working in cooperation to reduce the global loss. A balance needs to be struck between the local similarity within subgroups (which enables the modularity conducive to good generalization) and the utility of such subgroups in global training loss reduction which is necessary to learn appropriate functions. If too much of the local structures are destroyed in order to reduce the global loss, generalization is lost.

For models above the interpolation threshold: with more parametric flexibility this balance is much easier to maintain because fewer substructures need to be significantly changed in an attempt to interpolate the train set. Only with enough parametric flexibility can we obtain bilateral synergy between subgroups that are useful to reduce the training loss and subgroups that have the correct type of modularity in the hidden representation for appropriate generalization. For models below the interpolation threshold, the same balance needs to be upheld but smaller models are inherently less able to reduce the training loss. Therefore, the subgroups that have the correct type of modularity are not counteracted upon, as much, by those that are necessary to reduce the loss completely.

7.6 Outlook

Works that *do* focus on substructures in ANNs are usually aimed at interpreting the decision making process [111–113]. For CNNs in particular, it has been shown that intermediate layers produce hierarchical structures that transition from general sample information to more specific class information in later layers [114, 115]. However, using this perspective as a means of investigating the fundamental principles by which ANNs generalize is not often undertaken. We speculate that this is because of a general perception of DL models as high-dimensional "black-box" models, disregarding substructures as viable sources of information that can pertain to function fitting in general.

Some notable explorations of substructure follow. In [116], linear classifiers were trained using the features produced by hidden layers in popular CNN models to estimate the utility (i.t.o linear separability) of feature representations at individual layers. It was shown that linear separability increases monotonically with depth. In [93] the advantages of depth in DNNs were investigated by looking at how later layers reuse computations from earlier layers and how the resulting compositional structures aid generalization. In [94] the type of functions that can be fitted by ANNs, based on their structural properties, was investigated. It was found that the complexity of computable functions increases exponentially with depth, and earlier weights are much less robust to noise. None of these three works primarily focused on generalization, but their findings contribute to our general understanding of the function fitting process. We hope that our contributions, which *do* focus on generalization, motivate further study using these methodologies.

If a bias-variance tradeoff is, in fact, the fundamental principle governing generalization in ML, we will need to rethink the concept of capacity and complexity so that it considers the distributed manner by which training data is fitted. Most functions that are able to be fitted by an ANN, will never be fitted because there is an inductive bias that groups similar samples together. A worst case estimate of the complexity of the functions being approximated will tend to miss this fact. We propose a larger emphasis on empirical investigations of substructures in ANNs both during training and at inference time. It

is quite clear that the whole is greater than the sum of the parts and without studies investigating the interactions between subcomponents we have little hope of determining exactly how generalization and capacity are related in practice.

7.7 Future work

An additional contribution from our work is that it opens up a variety of promising themes of research that are not often considered. On a conceptual level our findings suggest that the number of trainable parameters, specifically, is a very poor metric to predict generalization. The number of substructures might be much more informative. Finding a theoretically principled estimate of the number of substructures in a trained network would probably provide more insight on its ability to generalize than the number of parameters. A natural next step would be to determine how independent such substructures are, what level of redundancy they carry, and if there are any substructures shared by independently trained models at various degrees of generalization.

On a more practical note, for further research, we would like to:

- Investigate the extent to which the findings presented in this thesis extend to more specialized ANN architectures such as modern CNNs or RNNs. For example, it is widely accepted that CNNs, in particular, utilize depth better than fully-connected alternatives. We are referring to the empirical observation that generalization often improves for CNNs of increasing depth [39,117]. It would be interesting to determine how class sensitivity, activation patterns, or noise polarization vary across depth in a CNN when compared to MLPs.
- Look at concepts from ensemble learning and multitask learning as a way of determining properties of subunits that could be conducive to good generalization. For example, it is known that "diversity" in subpredictors aids in the generalization of an ensemble [118].
- Create a practical measure of the destruction of earlier structures. This could help

develop optimization algorithms that automatically promote the stability of generalizable features fitted earlier in training.

- Explore the construction of sample sets throughout training. If we can determine subunit behaviors common to models that tend to generalize well, this might provide clues about the effect of the modularity of the hidden representation and its relation to model capacity. An intuitive example is to determine the amount of redundancy (e.g. using information theoretical methods or set theory) among sample sets. If many subunits model redundant aspects of the train set distribution, this could indicate that the available capacity is not optimally used to modularize the hidden representation.

- Eventually, we would like to create a framework that enables practitioners to more accurately estimate the ability of ANNs to generalize to unseen data.

7.8 Final remark

We believe that our findings sufficiently motivate our perspective on substructure and we hope that others will also regard ANNs as something more than a nebulous "black box" to be probed from a distance. Only when we get to grips with the way data is compartmentalized by substructures will we have the necessary tools to rethink generalization in DL.

Bibliography

[1] K. Murphy, *Machine Learning: A Probabilistic Perspective*, ser. Adaptive Computation and Machine Learning series. MIT Press, 2012. [Online]. Available: https://books.google.co.za/books?id=NZP6AQAAQBAJ

[2] T. Mitchell, *Machine Learning*, ser. McGraw-Hill International Editions. McGraw-Hill, 1997. [Online]. Available: https://books.google.co.za/books?id=EoYBngEACAAJ

[3] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[4] V. Vapnik, "An overview of statistical learning theory," *IEEE transactions on neural networks*, vol. 10, no. 5, pp. 988–999, 1999.

[5] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *ArXiv*, vol. abs/1207.0580, 2012.

[6] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014. [Online]. Available: http://jmlr.org/papers/v15/srivastava14a.html

[7] D. Arpit, S. Jastrzebski, N. Ballas, D. Krueger, E. Bengio, M. S. Kanwal, T. Maharaj, A. Fischer, A. C. Courville, Y. Bengio, and S. Lacoste-Julien, "A closer look at memorization in deep networks," in *Proceedings of the 34th*

International Conference on Machine Learning, ser. Proceedings of Machine Learning Research, vol. 70. PMLR, 2017, pp. 233–242. [Online]. Available: http://proceedings.mlr.press/v70/arpit17a.html

[8] B. Neyshabur, R. Tomioka, and N. Srebro, "In search of the real inductive bias: On the role of implicit regularization in deep learning." in *International Conference on Learning Representations (Workshop)*, 2015. [Online]. Available: http://arxiv.org/abs/1412.6614

[9] N. Rahaman, A. Baratin, D. Arpit, F. Draxler, M. Lin, F. Hamprecht, Y. Bengio, and A. Courville, "On the spectral bias of neural networks," in *Proceedings of the 36th International Conference on Machine Learning*, vol. 97. PMLR, 2019, pp. 5301–5310. [Online]. Available: http://proceedings.mlr.press/v97/rahaman19a.html

[10] K. Kawaguchi, L. P. Kaelbling, and Y. Bengio, "Generalization in deep learning," in *Mathematics of Deep Learning, Cambridge University Press, to appear. Prepint avaliable as: MIT-CSAIL-TR-2018-014, Massachusetts Institute of Technology*, 2018. [Online]. Available: https://lis.csail.mit.edu/wp-content/uploads/kklpkyb18.pdf

[11] D. Cohn and G. Tesauro, "How tight are the vapnik-chervonenkis bounds?" *Neural Computation*, vol. 4, pp. 249–269, 1992.

[12] S. Holden and M. Niranjan, "On the practical applicability of VC dimension bounds," *Neural Computation*, vol. 7, pp. 1265–1288, 1995.

[13] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals, "Understanding deep learning requires rethinking generalization," in *International Conference on Learning Representations*, 2017. [Online]. Available: https://arxiv.org/abs/1611.03530

[14] S. Oymak and M. Soltanolkotabi, "Toward moderate overparameterization: Global convergence guarantees for training shallow neural networks," *IEEE Journal on Selected Areas in Information Theory*, vol. 1, pp. 84–105, 2020.

[15] S. Hochreiter and J. Schmidhuber, "Flat minima," *Neural Computation*, vol. 9, pp. 1–42, 1997.

[16] P. Chaudhari, A. Choromanska, S. Soatto, Y. LeCun, C. Baldassi, C. Borgs, J. T. Chayes, L. Sagun, and R. Zecchina, "Entropy-SGD: Biasing gradient descent into wide valleys," in *In International Conference on Learning Representations*, 2017.

[17] N. Keskar, J. Nocedal, P. Tang, D. Mudigere, and M. Smelyanskiy, "On large-batch training for deep learning: Generalization gap and sharp minima," in *5th International Conference on Learning Representations*, 2017.

[18] G. F. Elsayed, D. Krishnan, H. Mobahi, K. Regan, and S. Bengio, "Large margin deep networks for classification." in *NeurIPS*, 2018, pp. 850–860. [Online]. Available: http://dblp.uni-trier.de/db/conf/nips/nips2018.html#ElsayedKMRB18

[19] F. Croce, M. Andriushchenko, and M. Hein, "Provable robustness of ReLU networks via maximization of linear regions," in *the 22nd International Conference on Artificial Intelligence and Statistics*. PMLR, 2019, pp. 2057–2066.

[20] Y. Jiang, D. Krishnan, H. Mobahi, and S. Bengio, "Predicting the generalization gap in deep networks with margin distributions," in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=HJlQfnCqKX

[21] M. Davel, M. W. Theunissen, A. M. Pretorius, and E. Barnard, "DNNs as layers of cooperating classifiers," *ArXiv*, vol. abs/2001.06178, 2020.

[22] M. H. Davel, M. W. Theunissen, A. M. Pretorius, and E. Barnard, "DNNs as layers of cooperating classifiers," in *Proc. Thirty-Fourth AAAI Conference on Artificial Intelligence*, 2020.

[23] M. W. Theunissen, M. H. Davel, and E. Barnard, "Insights regarding overfitting on noise in deep learning." in *South African Forum for Artifical Intelligence Research*, 2019, pp. 49–63.

[24] ——, "Benign interpolation of noise in deep learning," *South African Computer Journal*, vol. 32, no. 2, pp. 80–101, 2020.

[25] D. Sculley and C. Brodley, "Compression and machine learning: a new perspective on feature space vectors," *Data Compression Conference (DCC'06)*, pp. 332–341, 2006.

[26] D. MacKay, "Bayesian interpolation," *Neural Computation*, vol. 4, pp. 415–447, 1992.

[27] C. E. Rasmussen and Z. Ghahramani, "Occam's razor," *Advances in neural information processing systems*, pp. 294–300, 2001.

[28] G. Montavon, G. Orr, and K. Müller, *Neural Networks: Tricks of the Trade*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012. [Online]. Available: https://books.google.co.za/books?id=M5O6BQAAQBAJ

[29] B. Neyshabur, R. Tomioka, and N. Srebro, "Norm-based capacity control in neural networks," in *Conference on Learning Theory*. PMLR, 2015, pp. 1376–1401.

[30] S. Sun, W. Chen, L. Wang, X. Liu, and T.-Y. Liu, "On the depth of deep neural networks: A theoretical view," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 30, no. 1, 2016.

[31] M. Anthony and P. L. Bartlett, *Neural Network Learning: Theoretical Foundations*. Cambridge University Press, 1999.

[32] S. Shalev-Shwartz and S. Ben-David, *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, 2014.

[33] V. N. Vapnik and A. Y. Chervonenkis, *On the Uniform Convergence of Relative Frequencies of Events to Their Probabilities*. Cham: Springer International Publishing, 2015, pp. 11–30. [Online]. Available: https://doi.org/10.1007/978-3-319-21852-6_3

[34] P. L. Bartlett and W. Maass, "Vapnikchervonenkis dimension of neural nets," in *The handbook of brain theory and neural networks (2nd.* MIT Press, 2003, pp. 1188 – 1192.

[35] V. Koltchinskii and D. Panchenko, "Empirical margin distributions and bounding the generalization error of combined classifiers," *Annals of Statistics*, vol. 30, pp. 1–50, 2002.

[36] M. Mohri, A. Rostamizadeh, and A. Talwalkar, "Foundations of machine learning," in *Adaptive computation and machine learning*, 2012.

[37] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals, "Understanding deep learning (still) requires rethinking generalization," *Commun. ACM*, vol. 64, no. 3, pp. 107–115, Feb. 2021. [Online]. Available: https://doi.org/10.1145/3446776

[38] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[39] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, pp. 1097–1105, 2012.

[40] G. E. Hinton, "Learning distributed representations of concepts," in *Parallel distributed processing: Implications for psychology and neurobiology.* Clarendon Press/Oxford University Press, 1986, pp. 46 – 61.

[41] D. E. Rumelhart and J. L. McClelland, *Parallel distributed processing, explorations in the microstructure of cognition: foundations.* MIT Press, 1986, vol. 1.

[42] Y. Bengio, R. Ducharme, P. Vincent, and C. Janvin, "A neural probabilistic language model," *The journal of machine learning research*, vol. 3, pp. 1137–1155, 2003.

[43] V. N. Vapnik, *The Nature of Statistical Learning Theory.* Berlin, Heidelberg: Springer-Verlag, 1995.

[44] J. Sokolic, R. Giryes, G. Sapiro, and M. Rodrigues, "Robust large margin deep neural networks," *IEEE Transactions on Signal Processing*, vol. 65, pp. 4265–4280, 2017.

[45] H. Li, Z. Xu, G. Taylor, and T. Goldstein, "Visualizing the loss landscape of neural nets," in *Conference on Neural Information Processing Systems*, 2018.

[46] L. Dinh, R. Pascanu, S. Bengio, and Y. Bengio, "Sharp minima can generalize for deep nets," in *International Conference on Machine Learning*, 2017.

[47] M. Belkin, D. Hsu, S. Ma, and S. Mandal, "Reconciling modern machine-learning practice and the classical biasvariance trade-off," *Proceedings of the National Academy of Sciences*, vol. 116, pp. 15 849 – 15 854, 2019.

[48] P. Nakkiran, G. Kaplun, Y. Bansal, T. Yang, B. Barak, and I. Sutskever, "Deep double descent: Where bigger models and more data hurt," in *International Conference on Learning Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=B1g5sA4twr

[49] P. Nakkiran, P. Venkat, S. M. Kakade, and T. Ma, "Optimal regularization can mitigate double descent," in *International Conference on Learning Representations*, 2021. [Online]. Available: https://openreview.net/forum?id=7R7fAoUygoa

[50] X. Zhang and D. Wu, "Rethink the connections among generalization, memorization and the spectral bias of dnns," *ArXiv*, vol. abs/2004.13954, 2020.

[51] J. Ba, M. Erdogdu, T. Suzuki, D. Wu, and T. Zhang, "Generalization of two-layer neural networks: An asymptotic viewpoint," in *International conference on learning representations*, 2019.

[52] B. Neal, S. Mittal, A. Baratin, V. Tantia, M. Scicluna, S. Lacoste-Julien, and I. Mitliagkas, "A modern take on the bias-variance tradeoff in neural networks," *ArXiv*, vol. abs/1810.08591, 2018.

[53] S. Spigler, M. Geiger, S. d'Ascoli, L. Sagun, G. Biroli, and M. Wyart, "A jamming transition from under- to over-parametrization affects generalization in deep learning," *Journal of Physics A: Mathematical and Theoretical*, vol. 52, no. 47, 2019. [Online]. Available: https://doi.org/10.1088/1751-8121/ab4c8b

[54] S. Fort, P. Nowak, S. Jastrzebski, and S. Narayanan, "Stiffness: A new perspective on generalization in neural networks," *ArXiv*, vol. abs/1901.09491, 2019.

[55] G. Hacohen, L. Choshen, and D. Weinshall, "Lets agree to agree: Neural networks share classification order on real datasets," in *International Conference on Machine Learning*. PMLR, 2020, pp. 3950–3960.

[56] C. Agarwal and S. Hooker, "Estimating example difficulty using variance of gradients," *ArXiv*, vol. abs/2008.11600, 2020.

[57] S. Chatterjee, "Coherent gradients: An approach to understanding generalization in gradient descent-based optimization," in *International Conference on Learning Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=ryeFY0EFwS

[58] O. Bousquet and A. Elisseeff, "Stability and generalization," *The Journal of Machine Learning Research*, vol. 2, pp. 499–526, 2002.

[59] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006.

[60] C. C. Aggarwal *et al.*, "Neural networks and deep learning," *Springer*, vol. 10, pp. 978–3, 2018.

[61] S. Skansi, *Introduction to Deep Learning: From Logical Calculus to Artificial Intelligence*. Springer, 2018.

[62] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, 2011, pp. 315–323.

[63] D. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (ELUs)," in *4th International Conference on Learning Representations, ICLR 2016*, 2016. [Online]. Available: http://arxiv.org/abs/1511.07289

[64] D. Hendrycks and K. Gimpel, "Gaussian error linear units (gelus)," *arXiv preprint arXiv:1606.08415*, 2016.

[65] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *in ICML Workshop on Deep Learning for Audio, Speech and Language Processing*, 2013.

[66] D. R. Wilson and T. Martinez, "The general inefficiency of batch training for gradient descent learning," *Neural networks : the official journal of the International Neural Network Society*, vol. 16 10, pp. 1429–51, 2003.

[67] N. Cesa-Bianchi, A. Conconi, and C. Gentile, "On the generalization ability of online learning algorithms," *IEEE Transactions on Information Theory*, vol. 50, pp. 2050–2057, 2004.

[68] T. Breuel, "The effects of hyperparameters on sgd training of neural networks," *ArXiv*, vol. abs/1508.02788, 2015.

[69] D. Loyola, M. Pedergnana, and S. Gimeno-Garcia, "Smart sampling and incremental function learning for very large high dimensional data," *Neural networks : the official journal of the International Neural Network Society*, vol. 78, pp. 75–87, 2016.

[70] C. H. Sudre, W. Li, T. Vercauteren, S. Ourselin, and M. J. Cardoso, "Generalised dice overlap as a deep learning loss function for highly unbalanced segmentations," in *Deep learning in medical image analysis and multimodal learning for clinical decision support*. Springer, 2017, pp. 240–248.

[71] D. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, 1986.

[72] Y. N. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, and Y. Bengio, "Identifying and attacking the saddle point problem in high-dimensional non-convex optimization," in *Advances in Neural Information Processing Systems*, vol. 27. Curran Associates, Inc., 2014. [Online]. Available: https://proceedings.neurips.cc/paper/2014/file/17e23e50bedc63b4095e3d8204ce063b-Paper.pdf

[73] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *International conference on machine learning*. PMLR, 2013, pp. 1310–1318.

[74] N. Qian, "On the momentum term in gradient descent learning algorithms," *Neural networks : the official journal of the International Neural Network Society*, vol. 12 1, pp. 145–151, 1999.

[75] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference for Learning Representations*, 2015. [Online]. Available: http://arxiv.org/abs/1412.6980

[76] S. Russell, S. Russell, and P. Norvig, *Artificial Intelligence: A Modern Approach*, ser. Pearson series in artificial intelligence. Pearson, 2019. [Online]. Available: https://books.google.co.za/books?id=koFptAEACAAJ

[77] M. Hardt, B. Recht, and Y. Singer, "Train faster, generalize better: Stability of stochastic gradient descent," in *International Conference on Machine Learning*. PMLR, 2016, pp. 1225–1234.

[78] J. Wang, L. Perez *et al.*, "The effectiveness of data augmentation in image classification using deep learning," *Convolutional Neural Networks Vis. Recognit*, vol. 11, 2017.

[79] T. G. Dietterich, "Ensemble methods in machine learning," in *Multiple Classifier Systems*, 2000.

[80] S. d'Ascoli, L. Sagun, G. Biroli, and J. Bruna, "Finding the needle in the haystack with convolutions: on the benefits of architectural bias," in

Advances in Neural Information Processing Systems, vol. 32. Curran Associates, Inc., 2019. [Online]. Available: https://proceedings.neurips.cc/paper/2019/file/124c3e4ada4a529aa0fedece80bb42ab-Paper.pdf

[81] B. Neyshabur, "Towards learning convolutions from scratch," in *Advances in Neural Information Processing Systems*, vol. 33. Curran Associates, Inc., 2020, pp. 8078–8088. [Online]. Available: https://proceedings.neurips.cc/paper/2020/file/5c528e25e1fdeaf9d8160dc24dbf4d60-Paper.pdf

[82] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on machine learning*. PMLR, 2015, pp. 448–456.

[83] A. M. Pretorius, E. Barnard, and M. H. Davel, "Relu and sigmoidal activation functions," in *South African Forum for Artifical Intelligence Research*, 2019.

[84] E. Shelhamer, J. Long, and T. Darrell, "Fully convolutional networks for semantic segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, pp. 640–651, 2017.

[85] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, vol. 30. Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[86] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, ser. NIPS'14. MIT Press, 2014, pp. 2672–2680.

[87] R. Caruana, "Multitask learning," *Machine learning*, vol. 28, no. 1, pp. 41–75, 1997.

[88] A. M. Pretorius, "Activation functions in deep neural networks," Master's thesis, North-West University Faculty of Engineering, 2020.

[89] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[90] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *ArXiv*, vol. abs/1708.07747, 2017.

[91] R. Novak, Y. Bahri, D. A. Abolafia, J. Pennington, and J. Sohl-Dickstein, "Sensitivity and generalization in neural networks: an empirical study," in *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=HJC2SzZCW

[92] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," *IEEE International Conference on Computer Vision*, pp. 1026–1034, 2015.

[93] G. F. Montufar, R. Pascanu, K. Cho, and Y. Bengio, "On the number of linear regions of deep neural networks," in *Advances in Neural Information Processing Systems*, vol. 27. Curran Associates, Inc., 2014. [Online]. Available: https://proceedings.neurips.cc/paper/2014/file/109d2dd3608f669ca17920c511c2a41e-Paper.pdf

[94] M. Raghu, B. Poole, J. Kleinberg, S. Ganguli, and J. Sohl-Dickstein, "On the expressive power of deep neural networks," in *International conference on machine learning*. PMLR, 2017, pp. 2847–2854.

[95] R. Eldan and O. Shamir, "The power of depth for feedforward neural networks," in *Conference on learning theory*. PMLR, 2016, pp. 907–940.

[96] A. Brutzkus and A. Globerson, "Over-parameterization improves generalization in the xor detection problem," *ArXiv*, vol. abs/1810.03037, 2018.

[97] F. He, S. Lei, J. Ji, and D. Tao, "Neural networks behave as hash encoders: An empirical study," *ArXiv*, vol. abs/2101.05490, 2021.

[98] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[99] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *International Conference on Learning Representations*, 2015.

[100] C. M. Bishop, "Training with noise is equivalent to tikhonov regularization," *Neural Computation*, vol. 7, pp. 108–116, 1995.

[101] G. An, "The effects of adding noise during backpropagation training on a generalization performance," *Neural Computation*, vol. 8, pp. 643–674, 1996.

[102] W. J. Maddox, G. M. Benton, and A. Wilson, "Rethinking parameter counting in deep models: Effective dimensionality revisited," *ArXiv*, vol. abs/2003.02139, 2020.

[103] Z. Yang, Y. Yu, C. You, J. Steinhardt, and Y. Ma, "Rethinking bias-variance trade-off for generalization of neural networks," in *International Conference on Machine Learning*. PMLR, 2020, pp. 10 767–10 777.

[104] S. dAscoli, M. Refinetti, G. Biroli, and F. Krzakala, "Double trouble in double descent: Bias and variance(s) in the lazy regime," in *International Conference on Machine Learning*. PMLR, 2020, pp. 2280–2290.

[105] A. Krizhevsky, "Learning multiple layers of features from tiny images," Department of Computer Science, University of Toronto, Tech. Rep., 2009.

[106] A. E. Venter, M. W. Theunissen, and M. H. Davel, "Pre-interpolation loss behavior in neural networks," in *Southern African Conference for Artificial Intelligence Research*. Springer, 2021, pp. 296–309.

[107] B. Han, G. Niu, X. Yu, Q. Yao, M. Xu, I. Tsang, and M. Sugiyama, "Sigua: Forgetting may make learning with noisy labels more robust," in *International Conference on Machine Learning*. PMLR, 2020, pp. 4006–4016.

[108] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," in *Conference on Neural Information Processing Systems (workshop)*, 2011.

[109] T. Ishida, I. Yamane, T. Sakai, G. Niu, and M. Sugiyama, "Do we need zero training loss after achieving zero training error?" in *Proceedings of the 37th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 119. PMLR, 2020, pp. 4604–4614. [Online]. Available: http://proceedings.mlr.press/v119/ishida20a.html

[110] M. S. Advani and A. M. Saxe, "High-dimensional dynamics of generalization error in neural networks," *Neural Networks*, vol. 132, pp. 428 – 446, 2020.

[111] S. Bach, A. Binder, G. Montavon, F. Klauschen, K.-R. Müller, and W. Samek, "On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation," *PloS one*, vol. 10, no. 7, 2015.

[112] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in *Advances in Neural Information Processing Systems*, vol. 30. Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper/2017/file/8a20a8621978632d76c43dfd28b67767-Paper.pdf

[113] G. Montavon, S. Lapuschkin, A. Binder, W. Samek, and K. Müller, "Explaining nonlinear classification decisions with deep taylor decomposition," *Pattern Recognit.*, vol. 65, pp. 211–222, 2017.

[114] C. Ma, J.-B. Huang, X. Yang, and M.-H. Yang, "Hierarchical convolutional features for visual tracking," *2015 IEEE International Conference on Computer Vision (ICCV)*, pp. 3074–3082, 2015.

[115] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *European conference on computer vision*. Springer, 2014, pp. 818–833.

[116] G. Alain and Y. Bengio, "Understanding intermediate layers using linear classifier probes," in *5th International Conference on Learning Representations, Workshop*

Track Proceedings, 2017. [Online]. Available: https://openreview.net/forum?id=HJ4-rAVtl

[117] D. C. Ciresan, U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber, "Flexible, high performance convolutional neural networks for image classification," in *Twenty-second international joint conference on artificial intelligence*, 2011.

[118] L. L. Minku, A. White, and X. Yao, "The impact of diversity on online ensemble learning in the presence of concept drift," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, pp. 730–742, 2010.

[119] T. Clanuwat, M. Bober-Irizar, A. Kitamoto, A. Lamb, K. Yamamoto, and D. Ha, "Deep learning for classical Japanese literature," *ArXiv*, vol. abs/1812.01718, 2018.

[120] P. Simard, D. Steinkraus, and J. C. Platt, "Best practices for convolutional neural networks applied to visual document analysis," *Seventh International Conference on Document Analysis and Recognition, 2003. Proceedings.*, pp. 958–963, 2003.

Appendix A

Additional results

In this chapter we provide supplementary results.

In Section 5.5 we state that it is possible to fit three datasets simultaneously and then appropriately, and separately, generalize to their evaluation sets. We support this statement with the empirical results presented in Fig. A.1. We trained three random initialization of 3×1024 architectures on one of the seven train sets. We then measured the evaluation error on each of the seven evaluation sets. The seven sets represent every possible combination of the MNIST, FMNIST, and KMNIST datasets. For composite datasets (e.g. M+F) we simply group the train and evaluation sets into two larger supersets. This is possible because all three of the datasets have the same input dimensions, and number of classes. For experimental details see Section C.6.

Take note, from the bottom row, that models fitted to a composite of all three datasets (M+F+K) are able to generalize to any of the seven datasets at comparable levels to the models that were fitted to only the relevant train sets, as seen on the diagonal. As expected models lacking one of the three core datasets in their respective train set are unable to generalize to evaluation sets that only contain samples of such core datasets. This is seen in the evaluation errors highlighted in red, which are close to random guessing.

		generalize to						
		M	F	K	M+F	M+K	F+K	M+F+K
trained on	M	0.0156	0.9261	0.8933	0.4709	0.4545	0.9097	0.6117
	F	0.8617	0.1008	0.9232	0.4813	0.8925	0.5120	0.6286
	K	0.9497	0.8069	0.0753	0.8783	0.5125	0.4411	0.6106
	M+F	0.0163	0.1001	0.9387	0.0582	0.4775	0.5194	0.3517
	M+K	0.0152	0.8065	0.0786	0.4109	0.0469	0.4425	0.3001
	F+K	0.9381	0.0975	0.0823	0.5178	0.5102	0.0899	0.3726
	M+F+K	0.0172	0.1005	0.0862	0.0589	0.0517	0.0933	0.0680

Figure A.1: Evaluation error when generalizing from various composite MNIST-like datasets. Rows refer to the training data and columns refer to the evaluation data being generalized to. All models achieved zero training error. 'M', 'F', and 'K' refer to MNIST, FMNIST, and KMNIST, respectively. We present the average performance over three random initializations. The maximum standard error for all results is 0.0076006. The colors are only for visual illustration.

In Section 6.3.2, a reviewer suggested that it makes more sense to measure a sample's contribution to generalization in terms of the class to which it belongs. In Fig. A.2 we present similar results to those presented in Fig. 6.2 and 6.3. The difference is that for these results the location of a given sample on the vertical axis (as defined by $g(s)$ in Eq. 6.2) is calculated on a per-class basis. Meaning, we only calculate the correlations with samples in the evaluation set that are labeled with the same class as the current training sample. As expected, we observe higher maximum correlations overall. Other than that our findings remain the same.

For further clarity we also present these results for classes 0, 5, and 9, separately in Fig. A.3. Consequently, these plots contain approximately 6 000 samples each instead of the full 60 000. The refitting phenomenon is very evident for label corruption in these results.

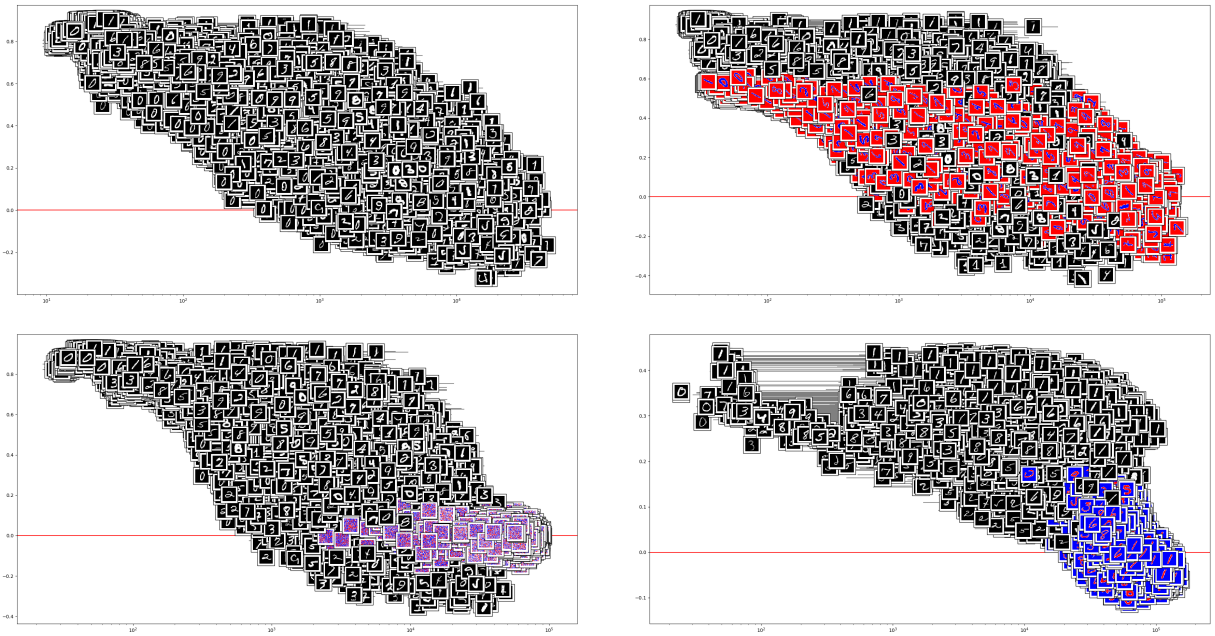

Figure A.2: Sample priority on a per-class basis. In order: Unaltered MNIST training samples, MNIST samples with 50% structured input corruption, MNIST samples with 50% Gaussian input corruption, and MNIST samples with 50% label corruption. Note that the alternative colormap for corrupted samples is just for visual distinction.

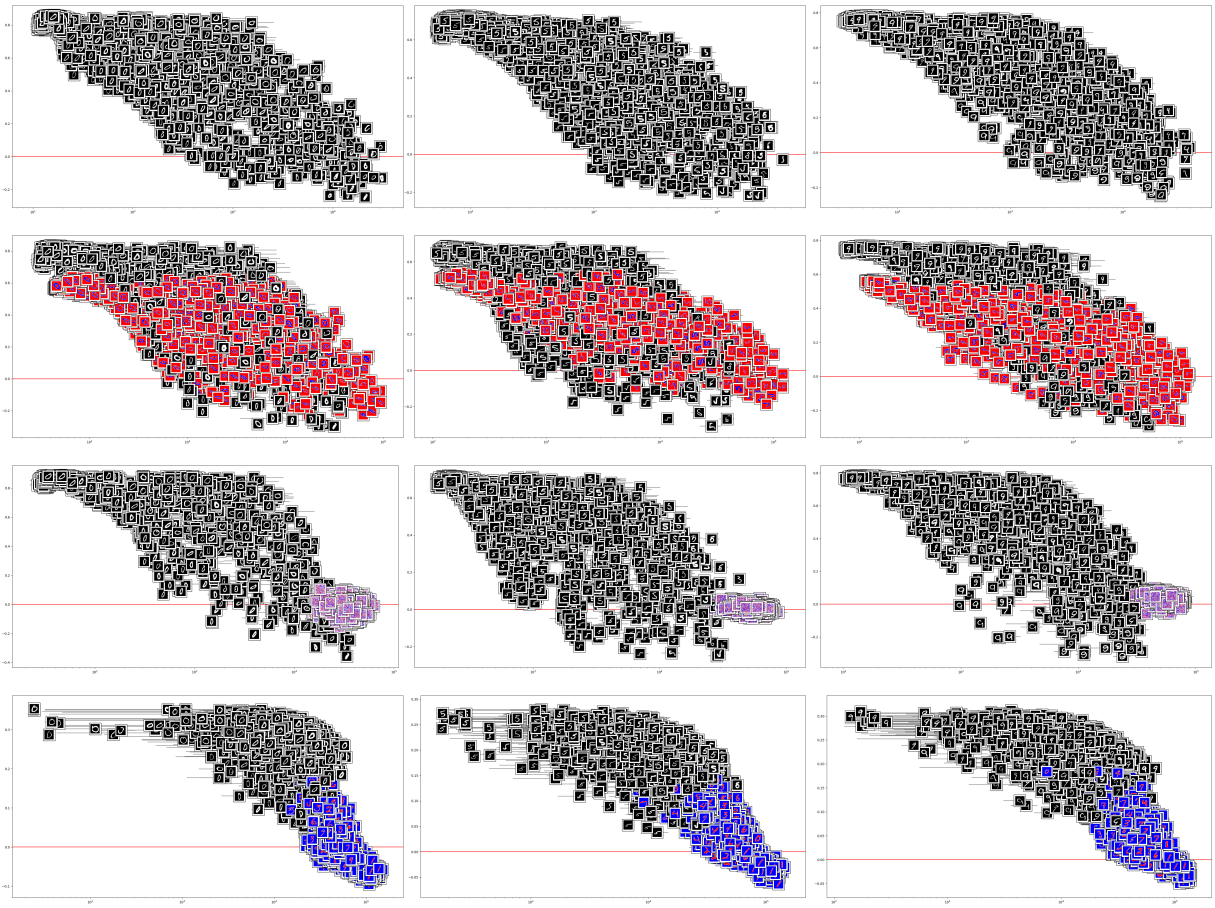

Figure A.3: Sample priority on a per-class basis for three classes separately: 0 (left), 5 (center), and 9 (right). From top to bottom: Unaltered MNIST training samples, MNIST samples with 50% structured input corruption, MNIST samples with 50% Gaussian input corruption, and MNIST samples with 50% label corruption. Note that the alternative colormap for corrupted samples is just for visual distinction.

Appendix B

Additional derivation

In this chapter we provide mathematical derivations of some statements made during the main body of text.

B.1 Differentiating loss functions

B.1.1 Mean Squared Error (MSE)

In accordance with Eq. 3.3 the per-sample loss value at output units $\widehat{\mathbf{y}} \in \mathbb{R}^K$ w.r.t one-hot encoded target values $\mathbf{y} \in \mathbb{R}^K$ is:

$$L(\widehat{\mathbf{y}}, \mathbf{y}) = \sum_{c \in K} \big(\widehat{\mathbf{y}}[c] - \mathbf{y}[c]\big)^2. \tag{B.1}$$

The partial derivative of the loss w.r.t output unit j is then defined by:

$$\frac{\partial L}{\partial \widehat{y}_j} = 2\big(\widehat{y}_j - y_j\big). \tag{B.2}$$

It is clear from this short description that the derivative of an MSE loss function w.r.t the output units is proportional to the difference between output values and target values.

B.1.2 Cross Entropy (CE)

In accordance with Eq. 3.4 the per-sample loss value at output units $\widehat{\mathbf{y}} \in \mathbb{R}^K$ w.r.t a target value $y \in [1, K]$ is:

$$L(\widehat{\mathbf{y}}, y) = -\log\big(\frac{e^{\widehat{\mathbf{y}}[y]}}{\sum_{c \in K} e^{\widehat{\mathbf{y}}[c]}}\big), \tag{B.3}$$

which can be rewritten as:

$$L(\widehat{\mathbf{y}}, y) = -\widehat{\mathbf{y}}[y] + log\big(\sum_{c \in K} e^{\widehat{\mathbf{y}}[c]}\big). \tag{B.4}$$

The partial derivative of the loss i.t.o output unit j is then defined by:

$$\frac{\partial L}{\partial \widehat{y}_j} = \begin{cases} -1 + \frac{e^{\widehat{\mathbf{y}}[j]}}{\sum_{c \in K} e^{\widehat{\mathbf{y}}[c]}} & \text{if } j = y \\ \frac{e^{\widehat{\mathbf{y}}[j]}}{\sum_{c \in K} e^{\widehat{\mathbf{y}}[c]}} & else. \end{cases} \tag{B.5}$$

From this we see that the derivative of L, w.r.t each output unit that is not at an index equal to the target value, is the predicted probability of that class, while the the derivative w.r.t the "correct" output unit is the predicted probability minus 1. Minimizing this loss function will result in a probability distribution over output units that resembles a one-hot encoding of the target value. This means that the derivative of CE w.r.t the output units is also proportional to the difference between output values and target values, although the target value is implicitly one-hot encoded and the output values are automatically normalized after a softmax has been applied.

Appendix C

Detailed experimental setups

In this chapter we provide more details regarding the experimental setups used throughout the thesis.

C.1 Terminology

The term **depth** refers to the number of hidden layers included in the MLP. The term **width** refers to the number of nodes per hidden layer. Unless stated otherwise, assume that there are an equal number of nodes for every hidden layer of a particular model. When presenting a **learning rate** in the form $lr * (\gamma/e)$, lr refers to the initial learning rate, γ is the decay constant, and e is the step size (in epochs) for learning rate decay. An **epoch** is one cycle through all the available training data. An **iteration** is one parameter update. Usually the number of iterations per epoch can be calculated by dividing the train set size by the batch size, and rounding down. When we define the data *splits* we use the following notation: train set size/validation set size/evaluation set size. For weight *initialization* we are referring to the *He* initialization scheme [92].

C.2 Datasets

The classification datasets that are used for empirical investigations are MNIST [89], FMNIST [90], and KMNIST [119], and CIFAR10 [105]. The first three are drop-in replacements for each other but have varying levels of difficulty.

C.2.1 MNIST

This dataset consists of 60 000 training samples and 10 000 test samples. Each sample has 28×28 input features and a class indicator (between zero and nine). The input features correspond to a grayscale image of a handwritten digit between zero and nine corresponding to the class indicator. Before feeding the input features into an MLP they are flattened to produce a vector of 784 features ranging from 0 to 1. Fig. C.1 shows the distribution of class membership. Notice that there is only the slightest class imbalance. The expected level of generalization for a well optimized simple MLP architecture trained on this dataset, without data augmentation, is a test error of approximately 0.016 [120]. Seeing as a simple linear classifier can achieve a test error of 0.12 [89], a large portion of this dataset is thought to be linearly separable.

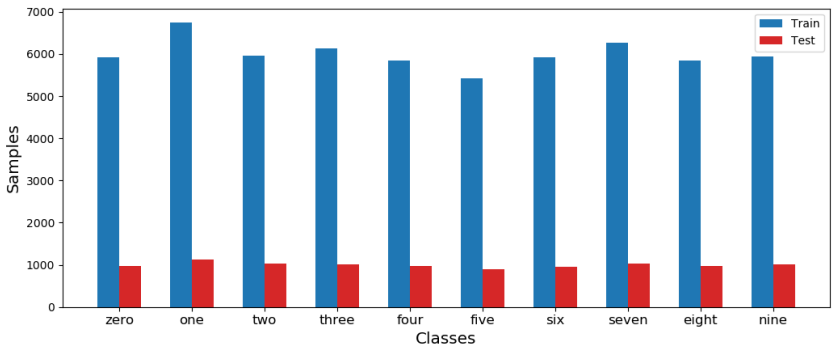

Figure C.1: MNIST class membership count.

C.2.2 FMNIST

This dataset has the same number of samples, input features, and classes as MNIST. It was created as a more difficult alternative to MNIST. The input features correspond to a grayscale image of one of ten types of accoutrement corresponding to the class indicator. These input features are also flattened before feeding it to the MLP. Fig. C.2 shows the distribution of class membership. Notice that there is no class imbalance. The expected level of generalization for a well optimized simple MLP architecture trained on this dataset, without data augmentation, is a test error of approximately 0.1 [91].

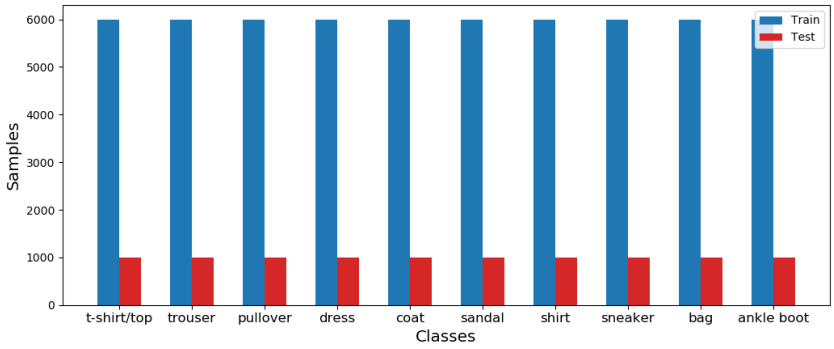

Figure C.2: FMNIST class membership count.

C.2.3 KMNIST

This dataset is, yet another, more difficult alternative to MNIST. The input features correspond to a grayscale image of one of ten hand written Japanese characters corresponding to the class indicator. These input features are also flattened before feeding it to the MLP. Fig. C.3 shows the distribution of class membership. Notice that there is no class imbalance. This is a less popular and newer dataset than the other three. Consequently, we were unable to find a credible expected generalization error for MLP models trained on this dataset. However, from our own experiments we note that performance tends to be similar to that of FMNIST.

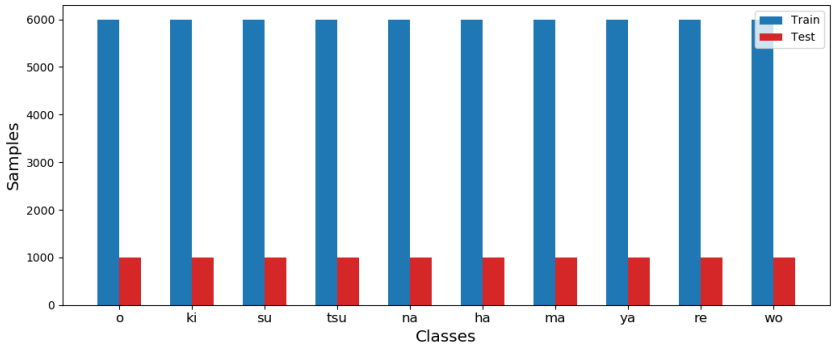

Figure C.3: KMNIST class membership count.

C.2.4 CIFAR10

This dataset consists of 50 000 training samples and 10 000 test samples. Each sample has $3 \times 32 \times 32$ input features and a class indicator. The input features correspond to a color image of a number of real-world objects corresponding to the class indicator. Before feeding the input features into an MLP they are flattened to produce a vector of 3 072 features. Fig. C.4 shows the distribution of class membership. There are no class imbalances. The expected level of generalization for a well optimized simple MLP architecture trained on this dataset, without data augmentation, is a test error of approximately 0.45 [91].

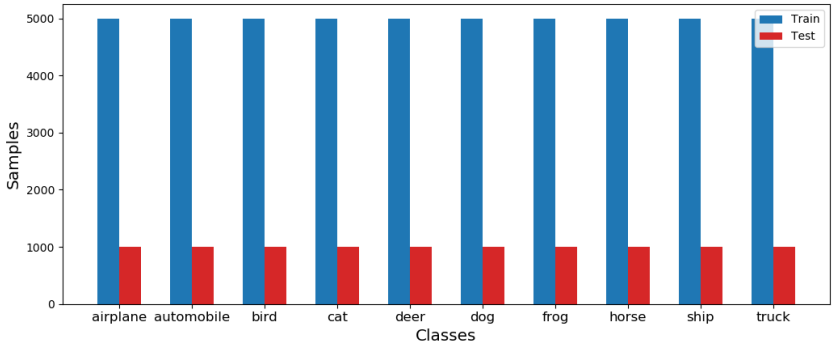

Figure C.4: CIFAR10 class membership count.

C.3 Regularities in subunit behavior

The models in Fig. 4.1 were optimized with a small grid search over learning rates and had the common hyperparameters presented in Table C.1. For every architecture we trained three random initializations with three learning rates (0.00055, 0.0007, and 0.00085) for 100 epochs with early stopping on the validation error. If the best validation error was found within the last 30% of the total epochs, 30% more epochs were added to the training process. This was repeated until the best validation error was found before the last 30% of the total epochs. From the resulting 9 models (3 initialization and 3 learning rates) we selected the one that had the best evaluation set performance.

Table C.1: Hyperparameters for Fig. 4.1

data	splits	loss_f	optim	learn_r	batch size
MNIST or FMNIST	55k/5k/10k	MSE	Adam(0.9/0.999)	optimized	64

The model in Fig. 4.2 was trained for 400 epochs with the hyperparameters presented in Table C.2. Early stopping on the validation error was used which resulted in a final train error of 0.00122, a validation error of 0.016, and an evaluation error of 0.0197.

Table C.2: Hyperparameters for Fig. 4.2

data	splits	loss_f	optim	learn_r	batch size
MNIST	55k/5k/10k	MSE	Adam(0.9/0.999)	0.001	64

The model in Fig. 4.9 was optimized with a small grid search over learning rates with the hyperparameters presented in Table C.3. We trained three random initializations with three learning rates (0.01, 0.001, and 0.0001) for 300 epochs with early stopping on the validation error. The learning rate was decayed with a constant of 0.99 every epoch. We selected the model that performed the best on the evaluation set.

Table C.3: Hyperparameters for Fig. 4.9

data	splits	loss_f	optim	learn_r	batch size
FMNIST	55k/5k/10k	CE	Adam(0.9/0.999)	optimized	64

C.4 Overparameterization and noise

All models in Fig. 5.1 were trained with the common hyperparameters presented in Table C.4. We trained three random initializations of each model for 1 500 epochs or up to the point of interpolation (train error of 0.0), whichever occurred first. 293 out of the 297 models (3 initializations, 3 datasets, 3 data corruptions, and 11 levels of corruption) attained a training error of 0.0. The four exceptions all had a training error lower than 0.00005. The generalization errors are measured on the evaluation set.

Table C.4: Hyperparameters for Fig. 5.1

data	splits	loss_f	optim	learn_r	batch size
MNIST, FMNIST, KMNIST	55k/5k/10k	CE	SGD	0.01*(0.99/10)	64

C.5 Sample priority and double descent

All models in Fig. 6.2 and 6.3 were trained, to interpolation, with the common hyperparameters presented in Table C.5. For the clean MNIST dataset (Fig. 6.2), five random initializations of depths ranging from 1 to 3, and five widths (32, 64, 128, 256, 512) were trained. This is 75 models in total. For the 50% corrupted datasets (Fig. 6.3) we added larger models because some of the smaller architectures proved insufficient to efficiently fit the entire training set.

For the MNIST with structured input corruption dataset we trained an identical spread of architectures, to the clean MNIST dataset, except for the addition of 4×32 and omission of 1×32 architectures, respectively. For the MNIST with Gaussian input corruption dataset we included five initializations of depths ranging from 1 to 4 with a width of 512, five initializations of depths ranging from 1 to 5 with widths of 256 and 128, three initializations of a 3×64 architecture, two initializations of a 4×64 architecture, and two initializations of a 5×64 architecture. This resulted in a total of 77 models. The MNIST with label corruption dataset required even more representational capacity. In this case we included five initializations of the following architectures: A width of 256 and

depths ranging from 2 to 5, widths of 512 and 1024 with depths ranging from 1 to 5, and a 1×2048 architecture. This resulted in a total of 75 models.

Table C.5: Hyperparameters for Fig. 6.2 and 6.3

data	splits	loss_f	optim	learn_r	batch size
MNIST	60k/0k/10k	CE	SGD	0.01*(0.99/5)	64

In order to log the appropriate per-sample fitting dynamics, a measure was taken with a logarithmically scaled sampler. This means that not every iteration is included in the calculation of $g(s)$ (see Eq. 6.2). The reason for this is our limited time, computational power, and storage. We decided that the loss of resolution is worth the saving in time and memory for two reasons. The first is that very little variation in parameter updates occur at later iterations and lower resolution is acceptable. The second reason is that absolute per-model resolution becomes less important if the metrics are averaged over 75 models, which is the case for this experiment.

The model presented Fig. 6.7 is trained, to interpolation, with the hyperparameters presented in Table C.6 on MNIST data with 50% label corruption.

Table C.6: Hyperparameters for Fig. 6.7

data	splits	loss_f	optim	learn_r	batch size
MNIST	60k/0k/10k	CE	SGD	0.2*(0.99/1)	512

All models in Fig. 6.8, 6.9, and 6.10 were trained, on MNIST data with 25% label corruption, with the common hyperparameters presented in Table C.7. For each of the eight architectures a small grid search over three random initialization and three learning rates (0.2, 0.1, or 0.05 with a decay constant of 0.99 every epoch) was conducted. The model that obtained the best validation error at interpolation or 500 epochs (which ever occurred first) was selected for investigation.

Table C.7: Hyperparameters for Fig. 6.10

data	splits	loss_f	optim	learn_r	batch size
MNIST	55k/5k/10k	CE	SGD	optimized	512

The models in Fig. 6.14 are trained with the common hyperparameters presented in

Table C.8. Five random initialization were trained for 1 000 epochs. An average over initializations is plotted with the standard error shown with the shaded error bars.

Table C.8: Hyperparameters for Fig. 6.14

data	splits	loss_f	optim	learn_r	batch size
CIFAR10	50k/0k/10k	CE	SGD	0.01*(0.99/5)	64

The models in Fig. 6.15 are trained with the common hyperparameters presented in Table C.9. Three random initialization were trained for 1 000 epochs. An average over initializations is plotted with the standard error shown with the shaded error bars.

Table C.9: Hyperparameters for Fig. 6.15

data	splits	loss_f	optim	learn_r	batch size
MNIST	55k/5k/10k	CE	SGD	0.01*(0.99/5)	64

The models in Fig. 6.16 are trained with 20% label corruption and the common hyperparameters presented in Table C.10. Five random initialization were trained for 1 000 epochs. An average over initializations is plotted with the standard error shown with the shaded error bars.

Table C.10: Hyperparameters for Fig. 6.16

data	splits	loss_f	optim	learn_r	batch size
FMNIST	55k/5k/10k	CE	SGD	0.01*(0.99/5)	64

C.6 Additional results

The models in Fig. A.1 are trained with the common hyperparameters presented in Table C.11. Three random initialization of 3×1024 architectures were trained to interpolation.

Table C.11: Hyperparameters for Fig. A.1

data	splits	loss_f	optim	learn_r	batch size
MNIST, FMNIST, KMNIST	60k/0k/10k	CE	SGD	0.2*(0.99/1)	512