# Activation functions in deep neural networks


## AM Pretorius

 orcid.org/ 0000-0002-6873-8904


Dissertation accepted in fulfilment of the requirements for the degree Master of Engineering in Computer and Electronic Engineering at the North West University

| | |
|---|---|
| Supervisor: | Prof MH Davel |
| Co-Supervisor: | Prof E Barnard |

| | |
|---|---|
| Graduation: | May 2020 |
| Student number: | 25022563 |

# Declaration

I, Arnoldus Mauritius Pretorius hereby declare that the dissertation entitled "Activation functions in deep neural networks" is my own original work and has not already been submitted to any other university or institution for examination.

A.M. Pretorius

Student number: 25022563

Signed on the 20th day of November 2019 at Potchefstroom.

# Acknowledgements

This research was performed within the Multilingual Speech Technologies (MuST) research group of the North-West University, which is a member of the Centre for Artificial Intelligence Research (CAIR) of the Department of Science and Innovation. It was supervised by Professors Marelie Davel and Etienne Barnard and is a product of a new research endeavor initiated by the group in 2018: the theory and application of machine learning, and understanding generalization in deep neural networks. The experience of working with world-class supervisors and distinguished researchers is one that has left me with confidence, curiosity, and extreme gratitude.

I would like to thank:

- Ulrike Janke and Laurene Jacobs, who made the administrative workload non-existing, and who always made sure I had everything I needed to do good work.
- Tian Theunissen, who started this journey with me, and who has been a colleague, mentor, and friend during these two years.
- The MuST group for unrestricted use of the *Teapot* server, without which none of the work would be possible, as well as regular visits to the Hermanus lab. I also acknowledge the Centre for High Performance Computing (CHPC), for providing computational resources to my research.
- My supervisors, Profs. Marelie and Etienne. Thank you for your guidance, patience and admirable dedication to the growth of your students. Your work ethic, knowledge, and goodwill are held in high esteem.

Lastly, I would like to thank my friends and family who have supported me in this endeavor. Thank you to my parents for motivating me to make the most of every opportunity; and to Esma, for every supportive phone call, letter and distraction.

# Abstract

The ability of machine learning algorithms to generalize is arguably their most important aspect as it determines their ability to perform appropriately on unseen data. The impressive generalization abilities of deep neural networks (DNNs) are not yet well understood. In particular, the influence of activation functions on the learning process has received limited theoretical attention, even though phenomena such as vanishing gradients, node saturation and sparsity have been identified as possible contributors when comparing different activation functions.

In this study, we present findings based on a comparison of several DNN architectures trained with two popular activation functions, and investigate the effect of these activation functions on training and generalization. We aim to determine the principal factors that contribute towards the superior generalization performance of rectified linear networks when compared with sigmoidal networks. We investigate these factors using fully-connected feedforward networks trained on three standard benchmark tasks.

We find that the most salient differences between networks trained with these activation functions relate to the way in which class-distinctive information is separated and propagated through the network. We find that the behavior of nodes in ReLU and sigmoidal networks shows similar regularities in some cases. We also find that there are relationships in the ability of hidden layers to accurately use the information available to them and the capacity (specifically depth and width) of the models. The study contributes towards open questions regarding the generalization performance of deep neural networks, specifically giving an informed perspective on the role of two historically popular activation functions.

***Keywords:*** *Deep neural network, Generalization, Non-linear activation function, Activation distribution, Node activity*

# Contents

**List of Figures** **ix**

**List of Tables** **xiii**

**List of Acronyms** **xiv**

**1 Introduction** **1**

1.1 Background . . . . . 1

1.2 Problem statement . . . . . 6

1.3 Project scope . . . . . 6

1.4 Research questions . . . . . 7

1.5 Objectives of the study . . . . . 7

1.6 Research methodology . . . . . 8

1.7 Dissertation overview . . . . . 9

1.8 Publications . . . . . 10

**2 Background** **11**

2.1 Introduction . . . . . 11

2.2 Literature Study . . . . . 12

2.2.1 A brief overview of deep neural networks . . . . . 12

2.2.2 Network optimization . . . . . 15

2.2.3 Activation functions and similar studies . . . 18
2.2.4 Sparsity . . . 20
2.3 Candidate datasets . . . 22
2.4 DNN tools (mustnet) . . . 23
2.5 Conclusion . . . 23

**3 Trained Models** **24**
3.1 Introduction . . . 24
3.2 Experimental setup . . . 25
3.2.1 Dataset preparation . . . 25
3.2.2 DNN architecture configuration . . . 26
3.3 Optimization . . . 27
3.3.1 Choosing hyperparameters . . . 27
3.4 Comparing accuracy . . . 28
3.4.1 MNIST . . . 29
3.4.2 FMNIST . . . 32
3.4.3 CIFAR10 . . . 35
3.5 Discussion . . . 38
3.6 Conclusion . . . 39

**4 Node Distributions** **41**
4.1 Introduction . . . 41
4.2 Node activation distributions . . . 42
4.2.1 Sigmoid . . . 43
4.2.2 ReLU . . . 47
4.3 CIFAR10 . . . 52

4.4 Discussion . . . 53
4.5 Conclusion . . . 54

**5 Sparsity, Node Specialization and Dead Nodes 56**

5.1 Introduction . . . 56
5.2 Theta values . . . 57
5.3 Node activity . . . 58
5.3.1 Activity of hidden layers . . . 58
5.3.2 Batch normalization . . . 60
5.4 Sparsity and dead nodes . . . 64
5.5 Discussion . . . 68
5.6 Conclusion . . . 69

**6 Nodes as Classifiers 71**

6.1 Introduction . . . 71
6.2 DNNs as layers of cooperating classifiers . . . 72
6.2.1 Theoretical hypothesis . . . 73
6.2.2 Probabilities based on distributions . . . 74
6.3 Probabilistic comparison of ReLU and sigmoid . . . 76
6.3.1 Layer accuracy . . . 76
6.3.2 Learning process . . . 81
6.3.3 Discussion . . . 83
6.4 Conclusion . . . 85

**7 Conclusion 87**

7.1 Introduction . . . 87
7.2 Key findings and implications . . . 87

7.3 Contributions . . . . . 90
7.4 Future work . . . . . 90
7.5 Conclusion . . . . . 91
References . . . . . 92

**A Supplemental Figures** **96**

A.1 Appendix: Chapter 4 . . . . . 96
A.2 Appendix: Chapter 6 . . . . . 97

# List of Figures

1.1 Function shape comparison of sigmoid and ReLU. . . . . 3

1.2 Example of ReLU vs. sigmoid train and validation accuracy. . . . . 4

1.3 Example of the average ReLU vs. sigmoid validation accuracy over 10 random training seeds for selected learning rates (lr). . . . . 5

2.1 Architecture of feedforward neural network with relevant notation, reproduced from [11], with permission. . . . . 14

2.2 Example of parameter sparsity in a simple linear model. Matrix **A** represents the parameterization of a model. . . . . 21

2.3 Example of representational sparsity in a simple linear model. Vector **h** is a sparse representation of an input **x**. . . . . 21

3.1 Learning curves and loss of a 4x200 network trained on MNIST with ReLU activations. The model parameters are saved at the epoch that achieves the highest validation accuracy. . . . . 28

3.2 Comparison of training and validation curves of ReLU and sigmoid networks with increase in depth (left to right) and a fixed width of 200 nodes. The top row of networks are trained without batch normalization while the bottom row is trained with batch normalization. . . . . 30

3.3 The same comparison as in Figure 3.2 with the exception of a fixed width of 800 nodes per layer. . . . . 31

3.4 A summary of the average evaluation accuracy for each network configuration trained on the MNIST dataset. . . . . 32

3.5 Comparison of training and validation curves of ReLU and sigmoid networks with increase in depth and a fixed width of 200 nodes per layer trained on the FMNIST dataset. . . . . 33

3.6 Comparison of training and validation curves of ReLU and sigmoid networks with increase in depth and a fixed width of 800 nodes per layer trained on the FMNIST dataset. . . . . 34

3.7 A summary of the average evaluation accuracy for each network configuration trained on the FMNIST dataset. . . . . 35

3.8 Comparison of training and validation curves of ReLU and sigmoidal networks with increase in depth and a fixed width of 200 nodes per layer trained on the CIFAR10 dataset. . . . . 36

3.9 Comparison of training and validation curves of ReLU and sigmoidal networks with increase in depth and a fixed width of 800 nodes per layer trained on the CIFAR10 dataset. . . . . 37

3.10 A summary of the average evaluation accuracy for each network configuration trained on the CIFAR10 dataset. . . . . 38

4.1 Activation distributions of generic nodes in shallow (top row) and deeper (bottom row) layers before sigmoid activation function is applied. (MNIST) 44

4.2 Activation distributions of generic nodes in shallow (top row) and deeper (bottom row) layers after sigmoid activation function is applied. (MNIST) 45

4.3 Medians of activation distributions for each class at every node in a 4x200 network. Generated for a untrained model with sigmoid activation functions. (MNIST) . . . . . 46

4.4 Medians of activation distributions for each class at every node in a 4x200 network. Generated for a trained model with sigmoid activation functions. (MNIST) . . . . . 47

4.5 Medians of activation distributions for each class at every node in a 4x200 network. Generated for a trained model with sigmoid activation functions and batch normalization. (MNIST) . . . . . 48

4.6 Activation distributions of generic nodes in shallow (top row) and deeper (bottom row) layers before ReLU activation function is applied. (MNIST) . 48

4.7 Activation distributions of generic nodes in shallow (top row) and deeper (bottom row) layers after the ReLU activation function is applied. (MNIST) 49

4.8 Medians of activation distributions for each class at every node in a 4x200 network. Generated for an untrained model with ReLU activation functions. (MNIST) . . . . . 50

4.9 Medians of activation distributions for each class at every node in a 4x200 network. Generated for trained model with ReLU activation functions. (MNIST) . . . . . 51

4.10 Medians of activation distributions for each class at every node in a 4x200 network. Generated for trained model with ReLU activation functions and batch normalization. (MNIST) . . . . . 52

4.11 CIFAR10: Median values of activation distributions for each class at every node in a 4x200 network, trained with sigmoid (left) and ReLU (right) activations. . . . . 53

5.1 Activity per node for a 4x200 ReLU network trained on MNIST. . . . . 60

5.2 Activity per node for a 4x200 sigmoid network trained on MNIST. . . . . 61

5.3 Activity per node for a 8x200 ReLU network trained on MNIST. . . . . 62

5.4 Activity per node for a 8x200 sigmoid network trained on MNIST. . . . . 63

5.5 Activity per node for a 8x200 network trained with batch normalization. Trained using ReLU (left) and sigmoid (right) activation functions. (MNIST, training set) . . . . . 63

5.6 Class-specific activity per node for a 4x200 network trained with ReLU activation functions. Brighter colors indicate higher activation counts while red lines indicate dead nodes. (MNIST) . . . . . 65

5.7 Class-specific activity per node for a 8x200 network trained with ReLU activation functions. Brighter colors indicate higher activation counts while red lines indicate dead nodes. (MNIST) . . . . . 66

5.8 Class-specific activity per node for a 4x200 network trained with sigmoidal activation functions. Brighter colors indicate higher activation counts while red lines indicate dead nodes. (MNIST) . . . . . 66

5.9 Class-specific activity per node for a 8x200 network trained with sigmoidal activation functions. Brighter colors indicate higher activation counts while red lines indicate dead nodes. (MNIST) . . . . . 67

6.1 Example of applied kernel density estimation to pre-activation distributions of ReLU (left) and sigmoidal (right) trained nodes. . . . . 75

6.2 Discrete, continuous and combined system train and test accuracies per layer for ReLU networks with varied depth (2-8) and width of 200 nodes. (FMNIST) . . . . . 77

6.3 Discrete and continuous system train and test accuracies per layer for sigmoidal networks with varied depth (2-8) and width of 200 nodes. (FMNIST) 78

6.4 Discrete, continuous and combined system train and test accuracies per layer for ReLU networks with varied depth (2-8) and width of 200 nodes. (MNIST) . . . . . 79

6.5 Discrete and continuous system train and test accuracies per layer for sigmoidal networks with varied depth (2-8) and width of 200 nodes. (MNIST) 80

6.6 ReLU: Train and test accuracies of the discrete, continuous and combined systems as measured on an FMNIST 6x100 DNN. System performance is shown after specific epochs. . . . . 81

6.7 Sigmoid: Train and test accuracies of the discrete and continuous systems as measured on an FMNIST 6x100 DNN. System performance is shown after specific epochs. . . . . 82

A.1 Medians of activation distributions for each class at every node in a layer. Generated for a 8x200 sigmoidal trained network. (MNIST) . . . . . 96

A.2 Medians of activation distributions for each class at every node in a layer. Generated for a 8x200 ReLU trained network. (MNIST) . . . . . 97

A.3 Discrete, continuous and combined system train and test accuracy's per layer for ReLU networks with varied depth (2-8) and width of 800 nodes. (FMNIST) . . . . . 97

A.4 Discrete and continuous system train and test accuracy's per layer for sigmoidal networks with varied depth (2-8) and width of 800 nodes. (FMNIST) 98

A.5 Discrete, continuous and combined system train and test accuracy's for ReLU networks with varied width (20-200) and a constant depth of 10 hidden layers. (FMNIST) . . . . . 98

A.6 Discrete and continuous system train and test accuracy's for sigmoidal networks with varied depth (1-8) and a constant width of 100 nodes. (FMNIST) 99

# List of Tables

2.1 Network architecture notation, based on [11]. . . . . 14

5.1 Percentage of dead nodes for networks trained with ReLU and sigmoid activations without and with batch normalization. All layers have a width of 200 nodes. . . . . 67

5.2 Layer sparsity for networks trained with ReLU and sigmoid activations without and with batch normalization. Sparsity here refers to the average percentage of samples per node that are inactive for a hidden layer (training set). We show the mean sparsity in a layer with standard error over three random training seeds. . . . . 68

# List of Acronyms

**DNNs** deep neural networks

**ReLU** rectified linear unit

**SGD** Stochastic gradient descent

**KDE** Kernal Density Estimator

**MSE** mean squared error

**MLPs** multilayer perceptrons

**CLT** Computational Learning Theory

# Chapter 1

# Introduction

---

*In this chapter we introduce the study and discuss why it is relevant and interesting. We discuss what forms part of the scope, as well as the research questions and objectives.*

---

## 1.1 Background

Generalization is the most important property of machine learning algorithms; it represents the ability of such algorithms to perform appropriately on unseen samples, based on their exposure to a corpus of training data. Towards the end of the twentieth century, a group of theoretical approaches jointly known as "Computational Learning Theory (CLT)" [1] was developed, providing a basis for understanding generalization in machine learning. However, recent developments with deep neural networks (DNNs) have demonstrated that CLT, at least in its naive form, fails to explain how these systems achieve their excellent generalization abilities [2].

Deep learning describes a range of machine learning techniques that allow models to contain multiple layers with non-linear processing units and has undergone some promising

transformation in the last couple of years. Despite the fact that deep learning models have shown great performance in fields such as computer vision, speech recognition, speech translation and natural language processing, their impressive generalization capabilities are still not well understood and theorized [2].

In machine learning, there is a distinction between supervised and unsupervised learning algorithms. Supervised learning algorithms are algorithms that learn to associate some input with some output given a labeled set of examples [3]. Unsupervised learning refers to models that extract information from a dataset without requiring annotated examples. We consider deep learning in the supervised context only.

Deep neural networks in their simplest form are called multilayer perceptrons (MLPs) [3]. MLPs consist of an input layer, one or several hidden layers and an output layer. Each node in a previous layer is connected to all the nodes in the following layer by a weight vector. These weight values are adjusted in the learning process so that the network output matches the labeled example, minimizing some loss function. To create a non-linear representation, and allow the network to train, each node is followed by an activation function that effectively "squishes" or rectifies the output of each node. The two most popular activation functions for deep neural networks, especially MLPs, are the sigmoidal function and rectified linear units (ReLUs). Figure 1.1 shows the difference between the sigmoidal and ReLU activation function and how the input is transformed.

One of the reasons that rectified linear units (ReLUs) [4]–[6] are easy to optimize is because of their similarity to linear units, apart from ReLU units outputting zero across half of their domain. This fact allows the gradient of a rectified linear unit to remain not only large, but also constant whenever the unit is active. A drawback of using ReLUs however is that they cannot learn via gradient-based methods with zero value activations, meaning that when the output at a node is 0 or less, the gradient is equal to zero [3].

Prior to the introduction of ReLUs, most DNNs used activation functions called logistic sigmoid activations or hyperbolic tangent activations. Sigmoidal units saturate across most of their domain: they saturate to a high value (usually 1) when the input is large and

positive and saturate to a low value (usually 0) when the input is large and negative [7]. The fact that a sigmoidal unit saturates over most of its domain can make gradient-based learning difficult [3]. The gradient of a sigmoidal function has a maximum value of 0.25 and tapers off to 0 when saturating. This causes a "vanishing gradients" problem when training deep networks that use these activation functions, because fractions get multiplied over several layers and gradients end up nearing zero.

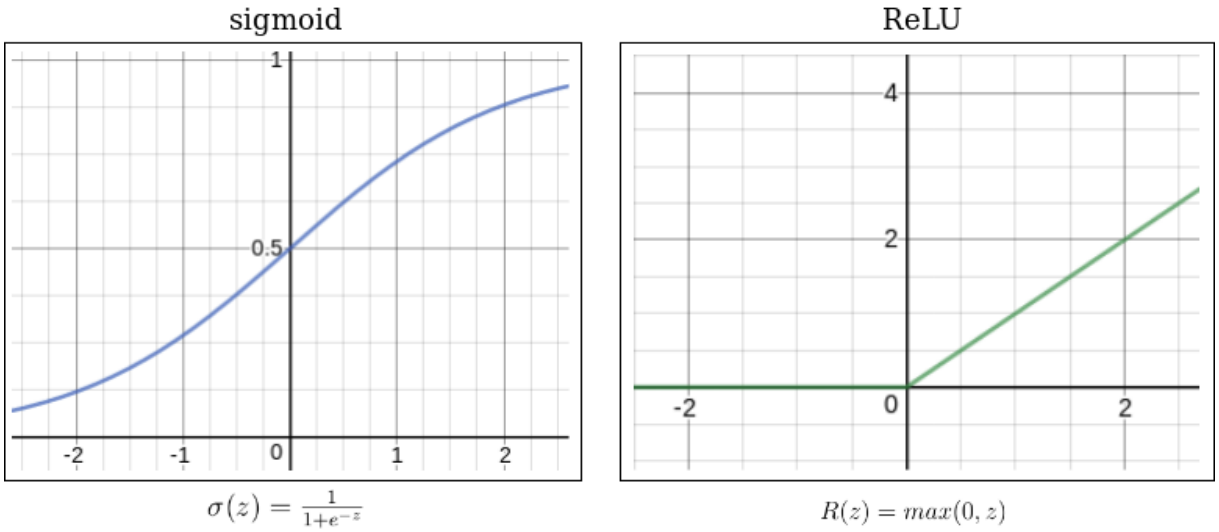


Figure 1.1: Function shape comparison of sigmoid and ReLU.

Networks developed with different activation functions produce different generalization results. For example, see Figure 1.2, where we compare the training and test accuracy of two similar networks with different activation functions on the same task. For each network, its hyperparameters are optimized individually (to obtain the best trained network per type). Both networks reach the same training accuracy just after the 50th epoch whereas the test accuracy (which is tested on a set of examples that the network has not seen before) of the ReLU network is higher than for the network using sigmoid activations.

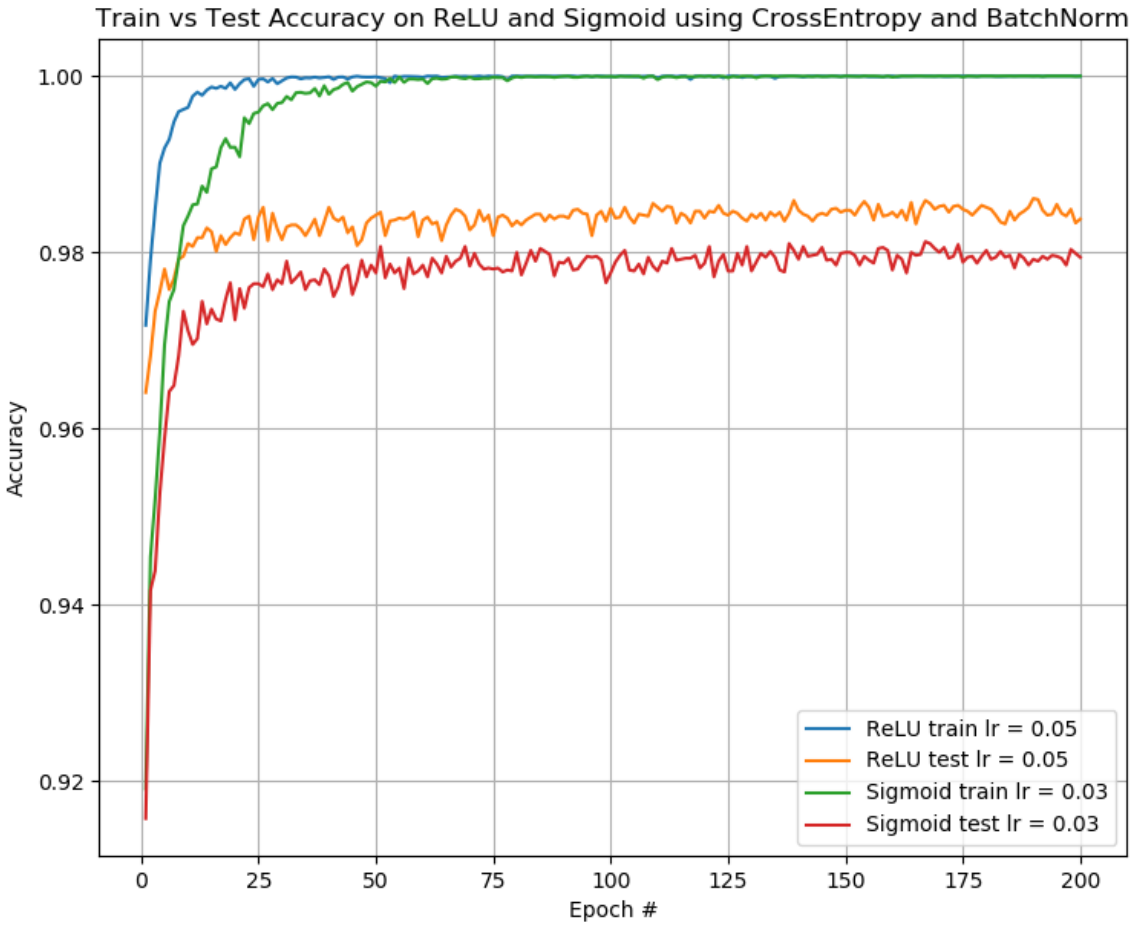


Figure 1.2: Example of ReLU vs. sigmoid train and validation accuracy.

These same two networks were compared over 10 different training seeds to ensure consistent results. The average test accuracy of the ReLU network over the 10 training seeds was still higher than the sigmoid network, as seen in Figure 1.3.

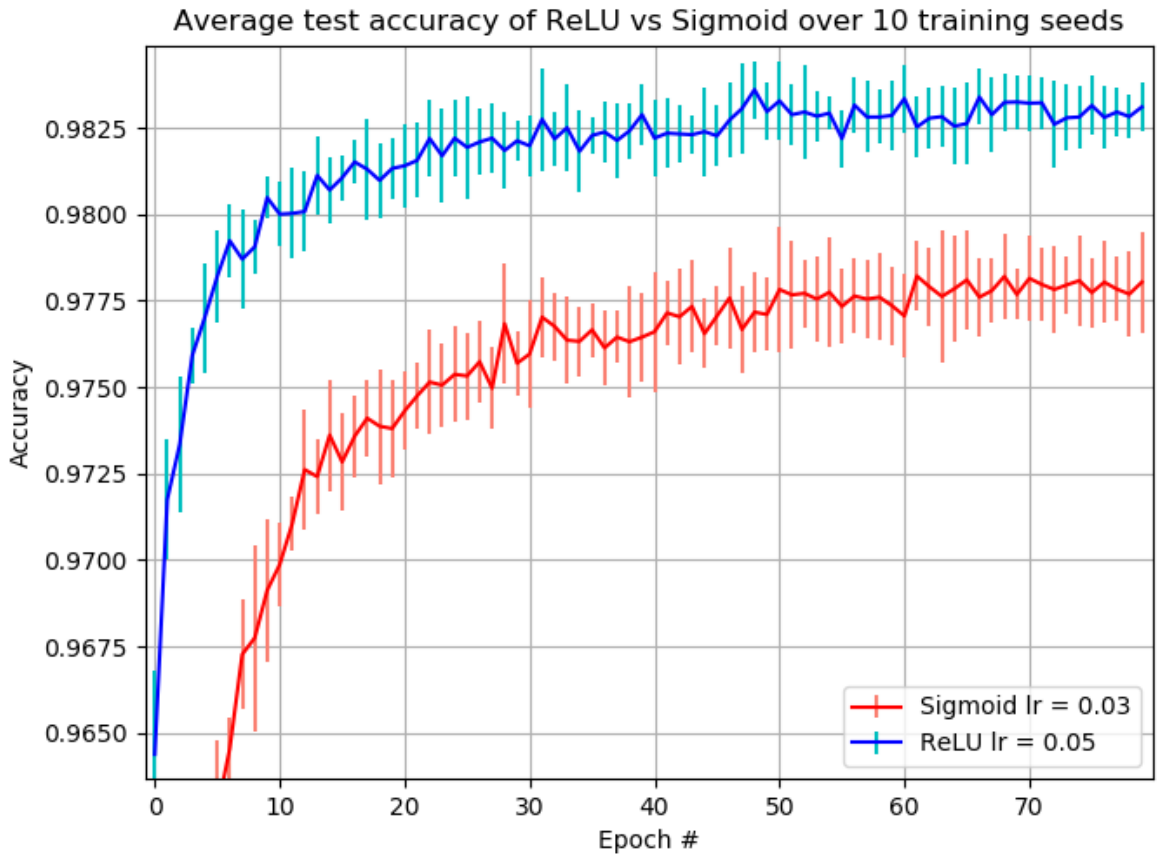


Figure 1.3: Example of the average ReLU vs. sigmoid validation accuracy over 10 random training seeds for selected learning rates (lr).

## 1.2 Problem statement

The generalization capabilities of DNNs are still not well understood, and limited work uses activation functions specifically to analyze the learning process. Certain network characteristics, specifically node saturation and sparsity [6], have been identified as important factors when comparing the effect of different activation functions on the network training process. Our goal is to better understand how these and other network characteristics shed light on the training and generalization process in deep neural networks, specifically when comparing models trained with different activation functions.

## 1.3 Project scope

Given the problem statement above, we restrict the scope of the research to a classification task and a limited number of datasets, architecture types and activation functions:

- **Datasets:** The datasets we have selected are MNIST, FashionMNIST and CIFAR10. These standard, benchmarked datasets have varying complexity, from relatively simple (MNIST) to fairly complex (CIFAR10). This provides more than one perspective when analyzing DNNs.
- **Architecture types:** We focus only on fully connected MLPs with varying depth and width.
- **Activation functions:** The two main activation functions we investigate are sigmoidal functions and ReLUs.

Our purpose is to probe how activation functions effect network behavior; we understand and regard this as being only a single piece of the puzzle regarding the generalization of DNNs. We aim to learn something from this study regarding the role of activation functions specifically.

## 1.4 Research questions

With the aim of investigating the generalization capabilities of a DNN in terms of its activation functions the following research questions are formulated:

- How do networks with different activation functions and varying width and depth (trained on specific datasets) compare with each other i.t.o. training and test accuracy?
- Which, if any, network characteristics (such as network sparsity, node activity or activation distributions) are likely candidates for analyzing the effect of the activation function on the learning process?
- For different network architectures, how do these characteristics compare across activation functions?
- How do these characteristics affect the training process and the generalization capabilities of a network?
- Can we supply a statement on the reason for the difference in generalization ability of networks trained with different activation functions?

## 1.5 Objectives of the study

With the above research questions in mind, the objectives of the study are the following:

- Determine experimental architecture configurations and train several deep neural networks on different datasets to find optimal hyperparameters and compare to benchmark performance results.
- Compare performance of different DNN architectures with different activation functions on different datasets.

- Investigate the effect of the characteristics of different activation functions on the DNNs by doing in-depth analysis on network properties (such as the activation distributions of specific nodes) and determine if and to what extent these characteristics influence the generalization capabilities of the network.
- Evaluate and discuss the implications of these findings on the generalization capabilities of deep neural networks.

## 1.6 Research methodology

This study consists of applied, quantitative and exploratory research. A significant part of the study consists of empirical experiments and the analysis thereof using the mustnet codebase (see Section 2.4). The following will form part of the study:

- **Literature review:** Gain a proper understanding of deep neural networks, focusing on the training process, activation functions and network generalization ability. Investigate recent research on deep neural networks i.t.o. generalization and the characteristics of different activation functions.
- **Codebase development:** Contribute to the mustnet codebase that was developed in-house to MuST. This codebase is still being developed in a team effort to configure, train, evaluate and analyze deep neural networks.
- **Experimental development:** Using the mustnet codebase we will configure and train a set of DNNs to:
  - Identify specific hyperparameters that we want to optimize for each DNN architecture and search for the optimum values for these hyperparameters.
  - Determine how the training and generalization performance of optimized DNNs with different activation functions compare to each other.
  - Analyze the DNNs and investigate the effect of the activation function on network characteristics. We specifically plan to investigate characteristics such as

activation distributions, node saturation, sparsity and general node behavior.

- **Assess and discuss the experimental findings:** After running a set of experiments the results will be assessed and discussed in detail.

## 1.7 Dissertation overview

This dissertation aims to give a perspective on the training and generalization abilities of ReLU and sigmoidal trained DNNs. The study consists mostly of an empirical investigation, resulting in some new theoretical perspectives. The dissertation is structured as follows:

- In Chapter 2 we review existing literature and give necessary background information regarding DNN generalization and activation functions.
- In Chapter 3 we describe the experimental setup that is used to train and evaluate several DNN architectures. We compare the training and generalization performance of these architectures with the activation functions of interest.
- In Chapter 4 we investigate the node behavior after applying a specific activation function. We investigate the continuous information available at each node (in the form of activation distributions) by looking at how nodes effectively separate and propagate class-distinctive information.
- In Chapter 5 we investigate the discrete behavior of ReLU and sigmoidal networks by choosing relative thresholds for switching from an "on" state to an "off" state for any sample-node pair. The discrete node behavior of DNNs are used to calculate a measure of sparsity for hidden layers in networks.
- In Chapter 6 we theorize about the continuous and discrete subsystems of DNNs, and how these systems use the continuous and discrete information available at each node to solve the classification task. We measure the accuracy of each system by creating a postulation of nodes and layers as individual classifiers. We compare the

accuracy per layer of ReLU and sigmoidal trained networks and discuss the possible implications of results.

- In Chapter 7 we discuss how objectives were met, we summarize the key findings and their implications and we discuss future work.

## 1.8 Publications

Some of the content of this dissertation is repeated in two papers that have since been published. These papers are titled "Sigmoid and ReLU activation functions" [8] and "DNNs as layers of cooperating classifiers" [9]. These papers were published at FAIR 2019 and AAAI 2020, respectively.

# Chapter 2

# Background

---

*In this chapter we give background information from relevant literature and introduce several concepts that are essential in this study. We investigate similar studies and identify areas where further empirical results are required.*

---

## 2.1 Introduction

In this chapter we provide background information on several key concepts and related studies. A brief overview of the field is given and the mathematical notation that is used throughout this study is introduced. The relevance and importance of non-linear activation functions is discussed and the progress and short-comings of existing studies are investigated. We then research concepts that are essential to completing the objectives stated in Chapter 1, such as methods for effectively training and analyzing DNNs.

A catalyst for much of the current research around generalization in DNNs stems from a paper written by Zhang et al. [10]. In this paper it was demonstrated, in contradiction to classical CLT frameworks [1], that very large networks with excessive capacity are able

to generalize very well to unseen data while simultaneously memorizing randomly labeled data or completely unstructured random noise. The findings from this paper are a strong motivation for the need to better understand DNN generalization.

## 2.2 Literature Study

### 2.2.1 A brief overview of deep neural networks

The goal of a feedforward neural network, and in particular a multilayer perceptron (MLP), is to approximate some function $f^*$ (being the true function) for a certain classification or regression problem [3]. In the case of classification (our focus from this point on-wards), this function $\hat{y} = f(\overline{x}; \boldsymbol{\theta})$ maps some input $\overline{x}$ to a category $y$ by learning the values for parameters $\boldsymbol{\theta}$ that results in the best approximation of this function. These parameters are also referred to as weights due to their nature of adding more or less value (or *weight*) to the outcome/value at a specific node. For classification, $\overline{y}$ is the one-hot encoded output vector with $y$ being the actual class label.

A deep neural network, in its most simple form, can be described as having 5 core components, see Figure 2.1 and Table 2.1 for all notation and dimensions: an input vector $\overline{x}$ that takes values from some high dimensional input (such as an image), hidden layers $\overline{h}_i$ that each consists of several nodes (or *neurons*), weights $w_{i,j,k}$ connecting any node in a hidden layer to any node in the previous layer, an output layer $\overline{y}$ that is one-hot encoded and corresponds to a specific class and a loss function $L(\hat{y}; \overline{y})$. The loss function calculates the difference between the predicted output vector $\hat{y}$ and the correct output vector $\overline{y}$ and gives this as a loss value. The objective of the network is to minimize the loss value, thus effectively minimizing the number of errors that occur when classifying data. A bias $b$ is an extra term that is summed to the output of each node before applying the non-linear activation function. Following the derivation in [11], the output at any node $h_{i,j}$ is a function of the weighted inputs from the previous layer nested within a non-linear

activation function $T$ so that:

$$h_{i,j} = T(\sum_{k=0}^{s_{(i-1)}} w_{i,j,k} h_{i-1,k}) \qquad 1 \leq i \leq N \tag{2.1}$$

with $w_{i,j,k}$ being the weight from node $k$, layer $i-1$ to node $j$, layer $i$ and with $h_{i-1,k}$ the output value at node $j$, layer $i-1$ and $N$ the number of network layers.

We only add a bias term to the input layer. This simplifies theoretical analysis, and does not hurt performance: if required for the task, an MLP with sufficient hidden nodes is able to create a "pseudo-bias" in any layer, as long as a bias exists in an earlier layer. In practice, if necessary, the optimization process is able to strengthen the weight between the true bias and pseudo-bias, while weakening all other weights to the pseudo-bias.

If we consider the bias added to the input layer, the output of the first hidden layer is somewhat different from the rest of the layers:

$$h_{1,j} = T(\sum_{k=0}^{s_{(0)}} w_{1,j,k} h_{i-1,k} + b_{0,k}) \qquad i = 1 \tag{2.2}$$

The purpose of the activation function $T$ is to generate non-linear mappings from inputs to the outputs so that the network can represent and learn complex, non-linear tasks from data. The form of the activation function should be differentiable so that we can perform back-propagation and compute the gradients of the loss function with regard to the network weights, then adjust these weights to minimize the loss value [3]. Activation functions will be described in further detail in Section 2.2.3.

Equation 2.1 and 2.2 can be simplified to have the output of each hidden layer as:

$$\overline{h}_i = T(\boldsymbol{w_i}\overline{h}_{i-1}) \qquad 1 \leq i \leq N \tag{2.3}$$

$$\overline{h}_1 = T(\boldsymbol{w_1}\overline{h_0} + \overline{b}_0) \qquad i = 0 \tag{2.4}$$

with $\boldsymbol{w_i}$ the weight matrix of layer $i$, assuming that the activation function $T$ is applied exactly the same way for all nodes within each layer $h_i$.

The mathematical notation used in this section is applied throughout the dissertation, unless specifically stated otherwise or rewriting/re-parameterizing an expression.

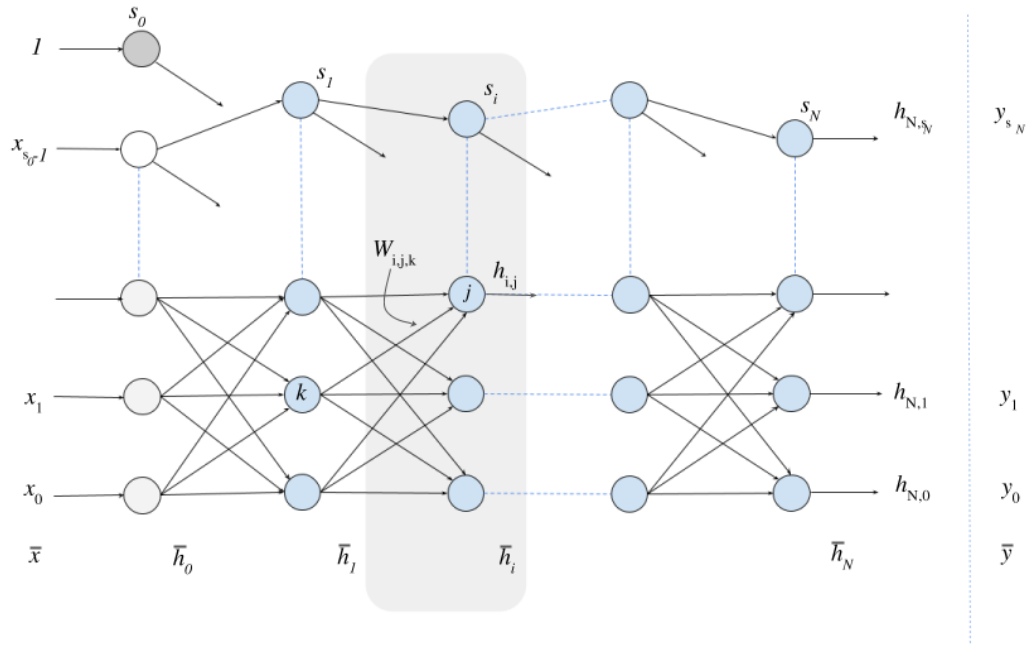


Figure 2.1: Architecture of feedforward neural network with relevant notation, reproduced from [11], with permission.

| | **Description** | **Size** |
|---|---|---|
| $N$ | number of network layers | scalar |
| $s_i$ | highest node index, layer $i$ | scalar |
| $\overline{x}$ | input vector | vector of size $s_{0-1}$ |
| $\overline{y}$ | target vector | vector of size $s_{N+1}$ |
| $\overline{h_i}$ | output vector, layer $i$ | vector of size $s_i + 1$ |
| | $\overline{h}_0$ equals $\overline{x}$ with 1 appended | |
| $\boldsymbol{w_i}$ | weights matrix, layer $i$ | matrix of size $(s_i + 1) \times (s_{i-1} + 1)$ |
| $\boldsymbol{\theta}$ | network parameters/weights | matrix of size $N \times (s_i + 1) \times (s_{i-1} + 1)$ |
| $w_{i,j,k}$ | weight from node $k$, layer i-1 to node $j$, layer $i$ | scalar |
| $h_{i,j}$ | output value at node $j$, layer $i$ | scalar |
| $b$ | bias term added to first layer | scalar |
| $D$ | dimension of input space after bias applied | scalar |
| | $D = s_0 + 1$ | |

Table 2.1: Network architecture notation, based on [11].

### 2.2.2 Network optimization

In this section we investigate and briefly introduce concepts that are related to choosing and optimizing hyperparameters for effective network training.

According to Goodfellow et al. [12], the iterative nature of training algorithms for DNNs require the networks to be well initialized so that a good "starting point" is specified. The initialization strategy can determine how fast a neural network converges—or if it converges at all. The correct initialization strategy introduces initial stability for training whereas a poorly chosen strategy would cause numerical difficulties and could cause convergence to fail altogether. When learning does converge, the initial point could determine if the network converges to a low or high cost, directly influencing the generalization capabilities of the model due to the fact that it determines the starting position in the solution space; although points of comparable cost can have varying generalization errors. For feedforward networks, it is generally accepted to use Xavier initialization [7] when using sigmoid activations and He initialization [13] when using ReLU activations.

Stochastic gradient descent (SGD) is a well known optimization technique and according to Goodfellow et al. [14], it is one of the most important algorithms behind nearly all of deep learning. The use of SGD made it possible to use large training sets that generalize well, compared to the computational inefficiency of normal gradient descent.

When using classical gradient descent (as opposed to SGD) to optimize network parameters, each parameter is updated in proportion to the derivative of the error function with regard to the parameter at the specific training point. This weight update rule can be written as:

$$\Delta w_{i,j,k} = -\eta \frac{\partial E}{\partial w_{i,j,k}} \tag{2.5}$$

with $E$ the error and $\eta$ a (possibly adaptive) learning rate. The usefulness of stochastic gradient descent, comes from the fact that the gradient used by the optimizer is an estimate and not an exact value. SGD uses several randomly chosen samples compiled into a mini-batch to approximate this expectation of the gradient [14]. There exist several variations of the SGD algorithm, but it seems that one of the most commonly used

variations is Adam [15]. The effectiveness and popularity of Adam comes from its adaptive estimates of lower-order moments compared to normal SGD.

The network loss function compares one or more predicted values to the real labeled targets and determines the cost associated with those values. The objective of the loss function (often called objective function) is to calculate the error between approximated values and real values. The optimizer then minimizes this loss function to effectively minimize the number of classification errors. Two of the most popular loss functions are cross-entropy loss and mean squared error (MSE). While cross-entropy [16] and MSE are well-known principles from information theory and statistics respectively, their use in deep learning has historical significance. Cross-entropy loss is based on maximum likelihood whereas MSE is based on minimizing a Euclidean distance. According to Goodfellow et al. [17], MSE was more popular in the 1980s and 1990s, but due to the spreading ideas around the principle of maximum likelihood, it was gradually replaced by the cross-entropy loss. The application of cross-entropy losses improved the performance of models trained with sigmoidal and softmax units, which was less compatible with MSE—resulting in saturation and slow convergence. The softmax function is typically used after the last layer when using the cross-entropy loss function, as it normalizes an input vector (thus vector from output layer) to a probability distribution that represents the probability of potential classes.

According to Goodfellow et al. [3], when a feedforward network is used to accept an input $\overline{x}$ to produce a predicted output $\hat{y}$, the information provided by $\overline{x}$ propagates up to hidden units at each layer and finally produces $\hat{y}$. This is called forward propagation. During training then, forward propagation continuous until it produces a scalar cost/loss $L(\boldsymbol{\theta})$ (with $\hat{y}$ a function of $\boldsymbol{\theta}$ and $L$ a function of $\hat{y}$). Although back-propagation [18] allows the network to propagate information regarding the loss backward through the network, the term back-propagation is often misunderstood as meaning the whole learning algorithm for DNN training. The back-propagation rule allows the network to compute the gradient of the loss function with regard to the network parameters. The optimizer (such as SGD) then effectively learns a better parameterization of the model given this gradient.

**General back-propagation rule:**

To outline the general back-propagation rule [11], assume a generic activation function $a$ with $a_{i,j}$ the activation result at layer $i$ for node $j$. Similarly assume a general error function $E$, and use $z_{i,j}$ to describe the sum of the input to node $j$ in layer $i$. Using back-propagation as derived in [11], the derivative from eq.(2.5) can then be calculated as:

$$\frac{\partial E}{\partial w_{i,j,k}} = \beta_{i,j} a_{i-1,k} \tag{2.6}$$

where

$$\beta_{i,j} = \begin{cases} \dfrac{\partial a_{i,j}}{\partial z_{i,j}} \dfrac{\partial E}{\partial a_{i,j}} & \text{if } i = N \text{ (output layer)} \\ \\ \dfrac{\partial a_{i,j}}{\partial z_{i,j}} \displaystyle\sum_n w_{i+1,n,j} \beta_{i+1,n} & \text{if } i \neq N \text{ (inner layer)} \end{cases} \tag{2.7}$$

and $n$ counts through all the forward connections from node $j$ to the next layer.

This is the update rule when evaluating the error produced by a single training sample and is used recursively from last hidden layer to first. When using stochastic gradient descent (SGD) these updates are averaged over a batch of random samples, before effecting an actual parameter update. Note that the form of $\frac{\partial a_{i,j}}{\partial z_{i,j}}$ is often different at the last layer (when $i = N$) when compared to the inner layers.

Although Zhang et al. [10] have shown that deep neural networks are able to generalize without explicit regularization, the importance and relevance of regularization techniques is central to training DNNs that give good generalization performance. Regularization often refers to the set of techniques that results in an increase of test accuracy, even if this is at the expense of training accuracy. There are several of these techniques, such as $L_1$ and $L_2$ norm penalties, that each achieves a regularizing effect in their own way. Some of the more relevant regularization techniques for this study is that of early stopping and batch normalization [19].

Batch normalization provides an effective way of reparameterizing most DNN architectures, and this reparameterization reduces the problem of coordinating updates across

many layers and adds stability to the training process [20]. According to Goodfellow et al. [20], batch normalization can be applied to any input or hidden layer in a feedforward network. With $\boldsymbol{H}$ a mini-batch of activations of the layer to normalize, that is arranged as a design matrix, with the activations for each sample appearing in a row of the matrix. To normalize $\boldsymbol{H}$, we replace it with:

$$\boldsymbol{H'} = \frac{\boldsymbol{H} - \boldsymbol{\mu}}{\boldsymbol{\sigma}} \tag{2.8}$$

where $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ are vectors that respectively contain the mean and standard deviation of each unit. When using an affine transform however, $\boldsymbol{H'}$ is multiplied with an $\alpha$ value and subtracted with a $\beta$ value that are both optimized by the SGD optimizer. Although the original design intent of batch normalization was not to explicitly regularize the learning process, it has a substantial indirect regularizing effect. This regularizing effect is achieved by introducing stochastic elements in the form of the mini-batch mean and standard deviation that is respectively subtracted and divided from the activation value at each node in a layer. This fact in addition to the fact that each mini-batch is randomly shuffled adds regularizing noise to the training process, similar to that of dropout [20]. Batch normalization is typically added as a extra layer that normalizes the output of the previous linear layer in the manner discussed above. The batch normalization layer has extra learnable parameters and already includes a bias when an affine transform is used.

Several of the concepts and hyperparameters discussed in this section are used to optimize the neural networks presented in this study.

### 2.2.3 Activation functions and similar studies

As discussed in Section 2.1, to create a non-linear representation and allow networks to learn complex non-linear problems, each node is followed by an activation function that effectively "squishes" or rectifies the output of the node. Two historically popular activation functions for deep neural networks are the established sigmoidal function and the widely used rectified linear unit (ReLU) [5], [6]. Various other activation functions have been proposed that are mostly variations of the ReLU function [21], [22], but none

are clearly superior to these functions; we therefore investigate only these two functions.

Although several researchers have compared the performance of non-linear activation functions in deep models [6], [7], [23], [24] and the respective difficulties of training DNNs with these activation functions have been established, a more concrete understanding of their effect on the training and generalization process is lacking.

A historical perspective given by Goodfellow et al. [3] highlights some of the uncertainty surrounding the matter:

> "The other major algorithmic change that has greatly improved the performance of feedforward networks was the replacement of sigmoid hidden units with piecewise linear hidden units, such as rectified linear units. Rectification using the $max(0; z)$ function was introduced in early neural network models and dates back at least as far as the Cognitron and Neocognitron (Fukushima,1975). These early models did not use rectified linear units, but instead applied rectification to nonlinear functions. Despite the early popularity of rectification, rectification was largely replaced by sigmoids in the 1980s, perhaps because sigmoids perform better when neural networks are very small. As of the early 2000s, rectified linear units were avoided due to a somewhat superstitious belief that activation functions with non-differentiable points must be avoided. This began to change in about 2009. Jarrett et al. (2009) observed that "using a rectifying nonlinearity is the single most important factor in improving the performance of a recognition system" among several different factors of neural network architecture design." [3]

It is widely considered that ReLU networks [4]–[6] are easy to optimize because of their similarity to linear units, apart from ReLU units outputting zero across half of their domain. This fact allows the gradient of a rectified linear unit to remain not only large, but also constant whenever the unit is active (allowing network training not to suffer from the "vanishing gradients" problem). A drawback of using ReLUs, however, is that they cannot learn via gradient-based methods when the node output is 0 or less, since the gradient is zero. [3].

Prior to the introduction of ReLUs, most DNNs used activation functions called logistic sigmoid activations or hyperbolic tangent activations. Sigmoidal units saturate across most of their domain: they saturate to a value of 1 when the input is large and positive, and

saturate to a value of 0 when the input is large and negative [7]. The fact that a sigmoidal unit saturates over most of its domain can make gradient-based learning difficult [3]. The gradient of an unscaled sigmoidal function is always less than 1 and tapers off to 0 when saturating. This causes a "vanishing gradients" problem when training deep networks that use these activation functions, because fractions get multiplied over several layers and gradients end up nearing zero.

Thus, both of the popular activation functions face certain difficulties during training, and remedies have been developed to cope with these challenges. The general consensus is that ReLU activations are empirically preferable to sigmoidal units, but the evidence in this regard is not overwhelming and theoretical motivation for their superiority is weak.

### 2.2.4 Sparsity

A critical paper by Glorot et al. [6] discussed the advantages and characteristics of rectifier networks vs sigmoidal networks. This paper, however, makes a claim that the use of rectifier *neurons* over sigmoidal units is biologically inspired. In contrast, for Goodfellow et al. [3], this inspiration from neuroscience is less important. Specifically, they state:

> "We know that actual neurons compute very different functions than modern rectified linear units, but greater neural realism has not yet led to an improvement in machine learning performance. Also, while neuroscience has successfully inspired several neural network architectures, we do not yet know enough about biological learning for neuroscience to offer much guidance for the learning algorithms we use to train these architectures." [3]

In this study, we distance ourselves from the biological inspirations for DNNs and deep rectifier networks and draw no line of relevance from this concept.

The paper by Glorot et al. [6] relies heavily on the concepts and advantages surrounding sparsity. We make a distinction between parameter sparsity, where the term refers to the fact that some parameters have an optimal value of zero, and representational sparsity or "sparse interaction". Representational sparsity describes a representation where many of the elements in the representation are zero (or close to zero). [3]

$$\underset{\mathbf{y} \in \mathcal{R}^{\mathbf{m}}}{\begin{bmatrix} 18 \\ 5 \\ 15 \\ -9 \\ -3 \end{bmatrix}} = \underset{\mathbf{A} \in \mathcal{R}^{\mathbf{mxn}}}{\begin{bmatrix} 4 & 0 & 0 & -2 & 0 & 0 \\ 0 & 0 & -1 & 0 & 3 & 0 \\ 0 & 5 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & -1 & 0 & -4 \\ 1 & 0 & 0 & 0 & -5 & 0 \end{bmatrix}} \underset{\mathbf{x} \in \mathcal{R}^{\mathbf{n}}}{\begin{bmatrix} 2 \\ 3 \\ -2 \\ -5 \\ 1 \\ 4 \end{bmatrix}}$$

Figure 2.2: Example of parameter sparsity in a simple linear model. Matrix **A** represents the parameterization of a model.

$$\underset{\mathbf{y} \in \mathcal{R}^{\mathbf{m}}}{\begin{bmatrix} -14 \\ 1 \\ 19 \\ 2 \\ 23 \end{bmatrix}} = \underset{\mathbf{B} \in \mathcal{R}^{\mathbf{mxn}}}{\begin{bmatrix} 3 & -1 & 2 & -5 & 4 & 1 \\ 4 & 2 & -3 & -1 & 1 & 3 \\ -1 & 5 & 4 & 2 & -3 & -2 \\ 3 & 1 & 2 & -3 & 0 & -3 \\ -5 & 4 & -2 & 2 & -5 & -1 \end{bmatrix}} \underset{\mathbf{h} \in \mathcal{R}^{\mathbf{n}}}{\begin{bmatrix} 0 \\ 2 \\ 0 \\ 0 \\ -3 \\ 0 \end{bmatrix}}$$

Figure 2.3: Example of representational sparsity in a simple linear model. Vector **h** is a sparse representation of an input **x**.

In the first expression (Figure 2.2) we show an example of a sparsely parameterized model, while in the second expression (Figure 2.3) we show a model with a sparse representation of the data [3].

The element-wise application of rectified linear units to the output of a layer creates true zero activations and in doing so creates a sparse representation of the layer. Having many hidden layers with ReLU activations then effectively creates a sparse representation for the model [6]. The authors then list several advantages of sparse representation in DNNs, which according to them are:

- Information disentangling: making the model more robust to small changes in input.

- Efficient variable-size representation: allows the model to control the effective dimensionality of the representation for a given input.
- A likeliness for the representation to be more linearly separable.
- Distributed but sparse representations: where the representational efficiency is not as rich as dense distributed representations, but presents a good trade-off between sparsity and distributed representations.

This paper goes on to compare the training and generalization performance of sparse rectified neurons to that of continuous non-linear activation functions such as the hyperbolic tangent function. Overall, we suspect that there are more factors that need to be considered when comparing activation functions, specifically regarding the behavior of hidden nodes.

## 2.3 Candidate datasets

To investigate the performance of different DNN architectures, we first establish appropriate tasks to train and evaluate our models. As computer vision tasks have become one of many standardized ways to benchmark the performance of DNNs, we train and evaluate different network architectures on three specific datasets. These datasets include:

- MNIST, which is a corpus of 70 000 handwritten digits having 10 targets [25].
- FMNIST (70 000 samples), which is more complex than MNIST and consists of images of pieces of clothing, including jackets, trousers and shoes; having a total of 10 targets [26].
- CIFAR10, which is considered (for standard MLPs) a complex problem consisting of 60 000 images of 10 different target classes, varying from airplanes and ships to cats and dogs [27].

We determine benchmark performance for MNIST [28], [29], FMNIST [30], [31] and CIFAR10 [32] to ensure that networks trained in this study have comparable performance.

## 2.4 DNN tools (mustnet)

To investigate the research questions asked in this study, several tools are used to ensure that results are trustworthy, reproducible and can be effectively presented. To achieve this we use the open-source PyTorch [33] library that was developed to allow users to use machine learning and deep learning techniques to easily create applications such as computer vision or speech recognition systems. Functional tools such as tensor creation, auto-differentiation and GPU acceleration is available to use for deep learning research and application. We use the Python and Bash scripting languages to develop a codebase that can effectively configure, train, evaluate and analyze deep neural networks. This codebase is developed in-house to the research group.

## 2.5 Conclusion

This chapter provided an overview of deep neural networks and a mathematical notation was introduced. Background information regarding activation functions and similar studies were presented. Key concepts such as vanishing gradients and network sparsity were discussed. Optimization strategies were investigated. Candidate datasets were identified and some of the development tools were introduced. In Chapter 3 we use the techniques described here to train and evaluate several neural network architectures.

# Chapter 3

# Trained Models

---

*In this chapter we describe the experimental setup used to train deep neural networks, how these networks were optimized and then compare the performance of different network architectures, trained with different activation functions.*

---

## 3.1 Introduction

Based on the literature in Chapter 2, we configure and train several DNN architectures. We use standard techniques to optimize networks and choose hyperparameters that lead to convergence. We then compare trained networks to evaluate the generalization capabilities of networks with different activation functions trained on different datasets. We make sure that networks are sufficiently trained by looking at the train loss over epochs as well as the training and validation accuracy.

The two main activation functions of interest are the rectified linear unit (ReLU) and the sigmoidal unit, as motivated in Chapter 1. We investigate deep feedforward neural networks only on classification problems, and specifically on standardized computer vision

tasks. Convolutional networks are more commonly used on these tasks and achieve higher accuracies, but we limit our attention to fully connected networks in order to investigate the essential components of generalization in DNNs.

## 3.2 Experimental setup

In this section we describe the experimental setup that was used in this study. We specifically describe the tools used, preparation of datasets, as well as the architecture configurations used. Note that in the rest of this dissertation we often demonstrate trends and concepts related to the behavior and activity of hidden nodes trained with ReLU and sigmoidal activation functions. In those cases we show individual networks. When we are contrasting values (as we are doing in this chapter) we obtain results over multiple random seeds, and report on mean and standard error across seeds, in order to determine the consistency of results.

### 3.2.1 Dataset preparation

The performance of different neural network architectures is analyzed on three specific datasets, these datasets are: MNIST, FMNIST (fashion MNIST) and CIFAR10, (see Section 2.3).

We use the torchvision pre-packaged versions [34] and the official test set (also referred to as the evaluation set) in all experiments. We split the data into a train and validation set, and use the official test set. The data is split so that the majority of the data is used to train the model, a smaller percentage of the data is used to validate the training accuracy and then a test set is used to ultimately evaluate the model and calculate the classification accuracy on unseen data. The validation set is also considered as unseen data, but this is the set with which we optimize hyperparameters. We choose hyperparameters that yield the highest validation accuracy. The generalization error is then the difference between the training accuracy and the final test accuracy with the model parameters that yield

the highest validation accuracy. For all three datasets we choose a validation set size of 5 000 samples, a test set size of 10 000 samples and the rest of the data, which is 55 000 samples for MNIST and FMNIST and 45 000 for CIFAR10, to train the model.

Datasets are prepared using PyTorch utilities that make the process of obtaining, preparing and using a dataset simple and repeatable. Datasets are downloaded and samples are cast to tensors. The dataset is combined with a sampler called a "dataloader" that effectively iterates through the dataset and feeds samples to the model in batches.

In this study a mini-batch size of 64 is used, meaning that the network calculates the loss for 64 randomly shuffled samples and updates the parameters accordingly using back-propagation.

### 3.2.2 DNN architecture configuration

We select different architectures to probe, for example, how deeper networks generalize with different activation functions vs. shallower networks, and also for wider networks vs. narrow networks. For each dataset we choose several architectures that we are interested in, namely:

- Network depths of 2, 4, 6 and 8 layers.
- Networks widths of 200 and 800 nodes.
- With batch normalization layers and without batch normalization layers.

When not adding batch normalization we add a bias to the first layer; when using batch normalization we use an affine transform at each layer. The batch normalization layer has extra learnable parameters and already includes a bias when an affine transform is used, adding an extra bias term to the weights is redundant. As previously mentioned, we only consider and evaluate networks with rectified linear units and networks with sigmoid activation functions.

## 3.3 Optimization

In this section we describe the optimization strategies that are used, as well as how hyperparameters are chosen and tuned.

### 3.3.1 Choosing hyperparameters

When choosing hyperparameters to optimize, we consider those that greatly affect model convergence and generalization performance. Values for these hyperparameters are selected after several initial experiments to determine combinations of hyperparameters that give the best convergence with the highest validation accuracy. For hyperparameters that do not have a large effect on performance, we choose values, after some initial testing, that are suited and likely to result in good convergence.

The hyperparameters that greatly affect convergence are: the optimizer, learning rate, train seed and weight initialization. The optimizer used to train the neural networks is Adam [15], due to its adaptive estimates of lower-order moments compared to normal SGD. We constantly optimize over three random training seeds when searching for hyperparameter values, to increase the robustness of our results (we do not optimize for seeds, but rather use random seeds as a way to ensure that sound hyperparameters are chosen). When choosing learning rates for Adam, we choose three initial learning rates that differ from each other with one order of magnitude. (The initial learning rates are 0.01, 0.001 and 0.0001.) We then use iterative grid search to determine appropriate learning rate values, and let the learning rate decay with a factor of 0.99 after every epoch using a learning rate scheduler.

We let all models train for 300 epochs, with one epoch being a full pass of the whole dataset. To regularize the network training, early stopping is used. This means that the model parameters are stored at the epoch that reached the highest validation accuracy. No other explicit form of regularization is added to the networks, such as L1 and L2 norm penalties. We do however recognize the regularizing effect of batch normalization [19].

We track training loss to ensure convergence. Figure 3.1 shows an example of how to monitor that the train loss decreases while the train and validation accuracy increases.

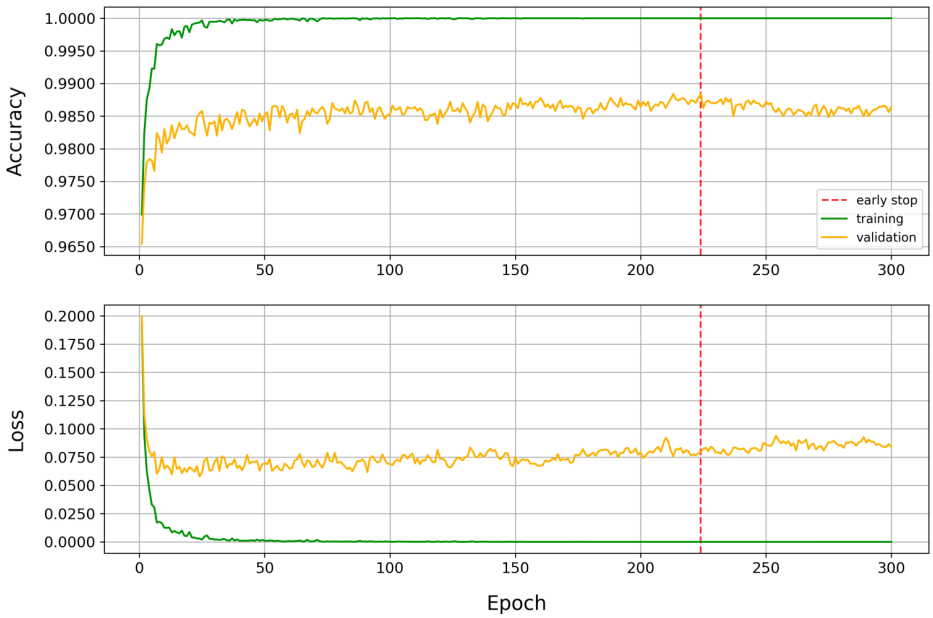


Figure 3.1: Learning curves and loss of a 4x200 network trained on MNIST with ReLU activations. The model parameters are saved at the epoch that achieves the highest validation accuracy.

We use Xavier initialization [7] when using sigmoid activations and He initialization [13] when using ReLU activations. Cross-entropy is used as loss function with a softmax layer at the output layer. The softmax layer normalizes the output values to form a probability distribution. Maximum likelihood estimation is used to determine predicted target values. The use of cross-entropy over mean squared error (MSE) was an empirical choice, but it is also supported in literature [17] to use cross-entropy, specifically when using sigmoid or softmax functions.

## 3.4 Comparing accuracy

In this section, we present the results achieved on each of the three datasets, after training several models and optimizing hyperparameters for different architectures.

Figure 3.2 is an example of how we present the average training and validation accuracy

per epoch for networks trained with ReLU and sigmoid activation functions. The columns show the learning curves for networks increasing in depth while the top row shows networks trained without batch normalization (bn_0) and the bottom row shows networks trained with batch normalization (bn_1). The average accuracy over three seeds is shown with error bars indicating the standard error of the mean. The blue curve represents the training accuracy of networks trained with ReLU activations, while the red curve shows the training accuracy of networks trained with sigmoid activations. The green curve shows the validation accuracy of networks trained with ReLU activations while the orange curve shows the validation accuracy of the sigmoidal networks. The same layout is presented for the performance of wider networks with a constant width of 800 nodes, as seen in Figure 3.3. Note that in some cases a training accuracy of 100.0% might not be the network that generalizes the best, this is why we use early stopping to ensure good generalization while also achieving good training accuracy.

### 3.4.1 MNIST

As MNIST is the easiest of the three problems, we expect the performance of the ReLU and sigmoid activation functions to be the most comparable of the three datasets.

From Figure 3.2 it is seen from the validation curves that when we average over random seeds the ReLU networks clearly outperform the sigmoid networks in each architecture configuration. This difference in validation accuracy is larger for networks trained with batch normalization. From Figure 3.3 this same behavior is seen, but with the difference in validation accuracy (between ReLU and sigmoid) for the wider networks even smaller when trained without batch normalization.

Depending on number of epochs trained, the number of layers and the number of nodes in each layer, the validation accuracy of the ReLU networks vary between 98.5% and 99.0%, while the sigmoid networks only ever reach validation accuracy equal or close to 98.5%. Both the ReLU and sigmoid networks achieve a training accuracy of 100%, or very close to it.

It is also observed from the training curve that while both networks seem to converge very quickly (after the first 50 epochs) the ReLU networks seem to converge slightly earlier when trained with batch normalization. The standard error over networks are relatively small for ReLU and sigmoid networks, meaning that there is little variance in the average accuracy per epoch, with the exception of the 8 layer sigmoidal network with batch normalization in Figure 3.3. The training and validation curves of deeper networks seem more unstable and noisy than those of shallower networks.

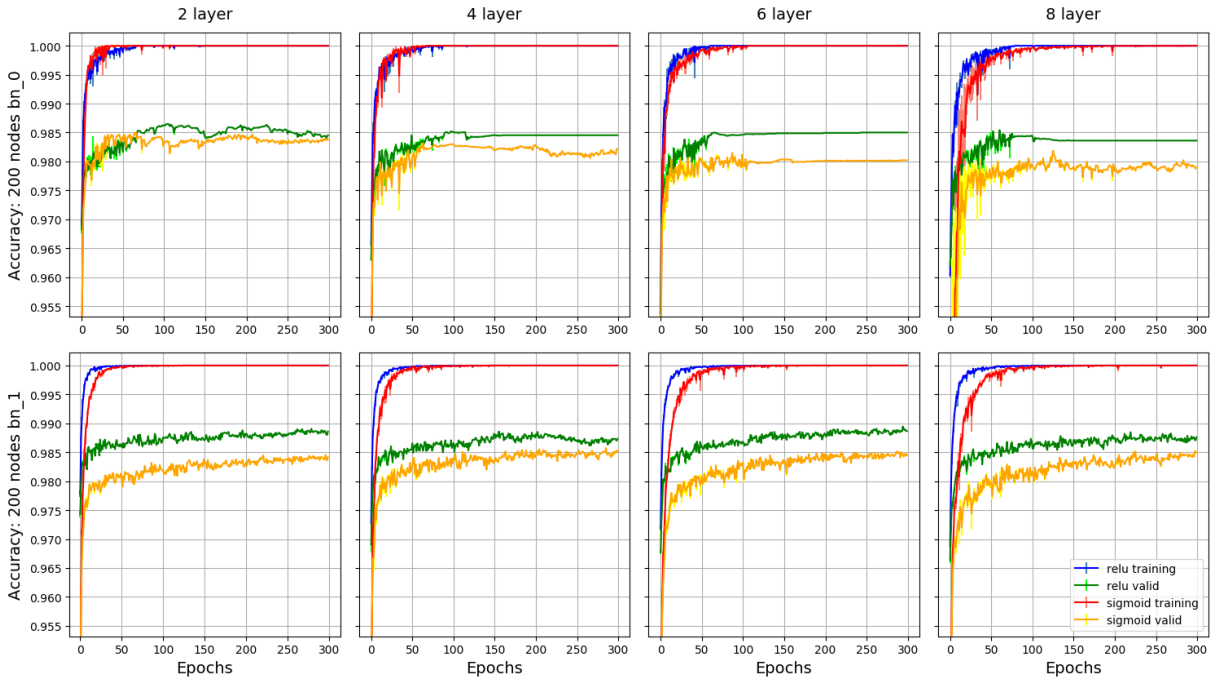


Figure 3.2: Comparison of training and validation curves of ReLU and sigmoid networks with increase in depth (left to right) and a fixed width of 200 nodes. The top row of networks are trained without batch normalization while the bottom row is trained with batch normalization.

From Figure 3.3 it is seen that the wider networks seem to perform slightly better than the networks with 200 nodes per layer, with the ReLU networks still outperforming the sigmoidal networks.

Figure 3.4 shows the evaluation accuracy of each architecture averaged over three random seeds with standard error shown with error bars showing the standard error. We observe that when we evaluate the networks on the 10 000 completely unseen samples of the test set, the ReLU networks tend to generalize better than the sigmoidal networks. It is

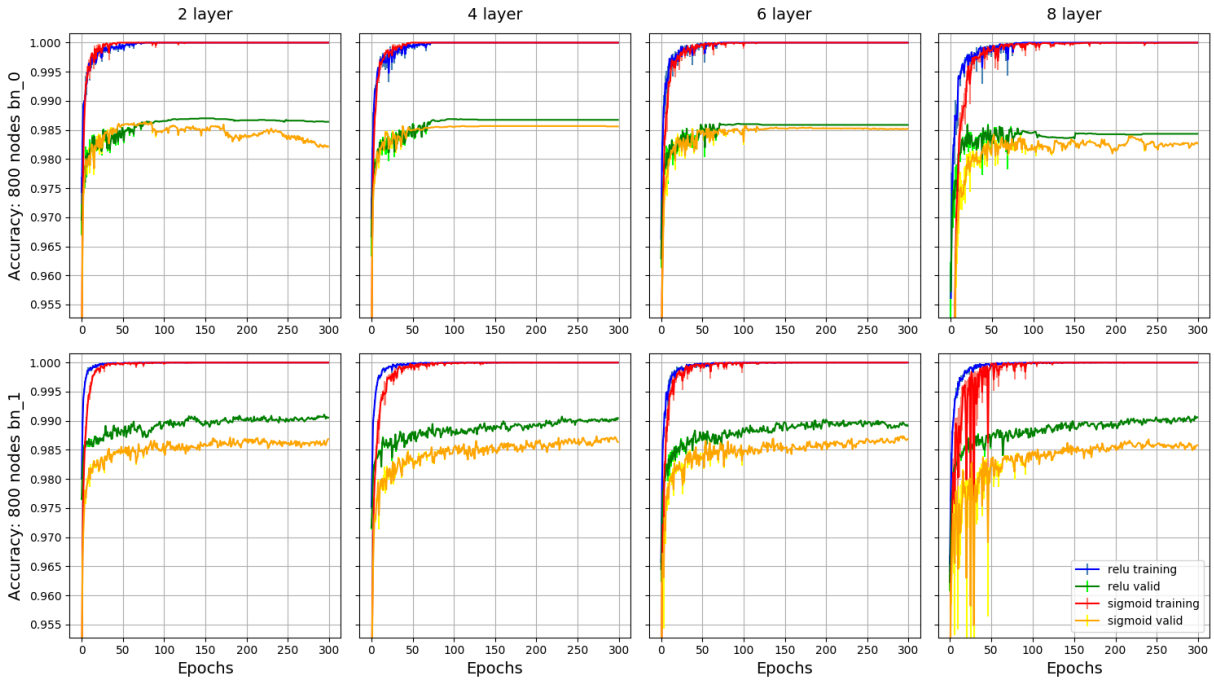


Figure 3.3: The same comparison as in Figure 3.2 with the exception of a fixed width of 800 nodes per layer.

observed that an increase in the number of hidden layers yields similar performance to shallow networks with some deeper networks generalizing slightly worse. We also observe however, that with a width of 200 nodes, the evaluation accuracy of the sigmoidal networks increases with depth when using batch normalization and decreases when not using batch normalization. Possible reasons for this observation are discussed later in this chapter. We should note that evaluation accuracy here is very high, and that the difference in accuracy is due to the misclassification of approximately 20 samples out of 10 000.

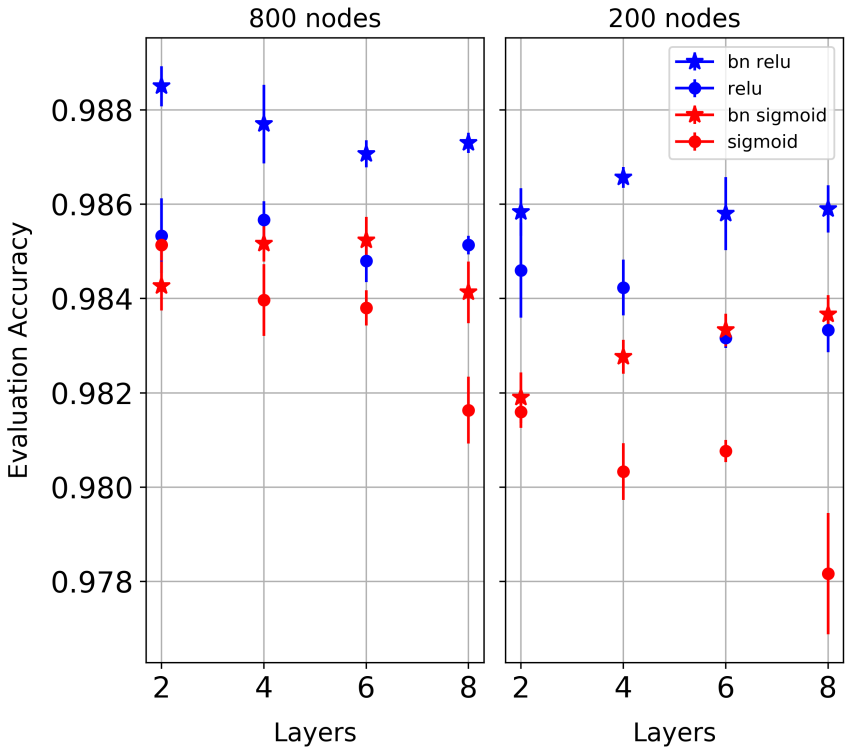


Figure 3.4: A summary of the average evaluation accuracy for each network configuration trained on the MNIST dataset.

### 3.4.2 FMNIST

From Figure 3.5, which shows the same information as Figure 3.2 (but for the FMNIST dataset), we can see that the ReLU networks again outperform the sigmoidal networks in each architecture configuration. The difference in validation accuracy is again larger for networks trained with batch normalization. From Figure 3.6 we observe that there is very little difference in the validation accuracy of wider networks trained without batch normalization. The validation curves of ReLU and sigmoidal networks are very similar with ReLU only slightly outperforming the sigmoid networks with more hidden layers. When using batch normalization with the wider networks we again see that the ReLU configurations are superior. An interesting observation we make is that in the case of both the 200 width and 800 width networks, the sigmoid networks seem to perform better without batch normalization. This difference is more clearly observed in shallower networks, but also visible in deeper networks.

Depending on the number of epochs trained, the number of layers and the number of nodes in each layer, the accuracy of the ReLU networks vary between 90.0% and 90.1%, while the sigmoid networks only ever reach validation accuracy equal to or slightly better than 90.0% when they are wide enough. Both the ReLU and sigmoid networks achieve a training accuracy of 100% or very close to it.

The training curves of the sigmoid networks seem to converge later when increasing the number of hidden layers. The standard error over epochs is relatively small for both ReLU and sigmoid networks.

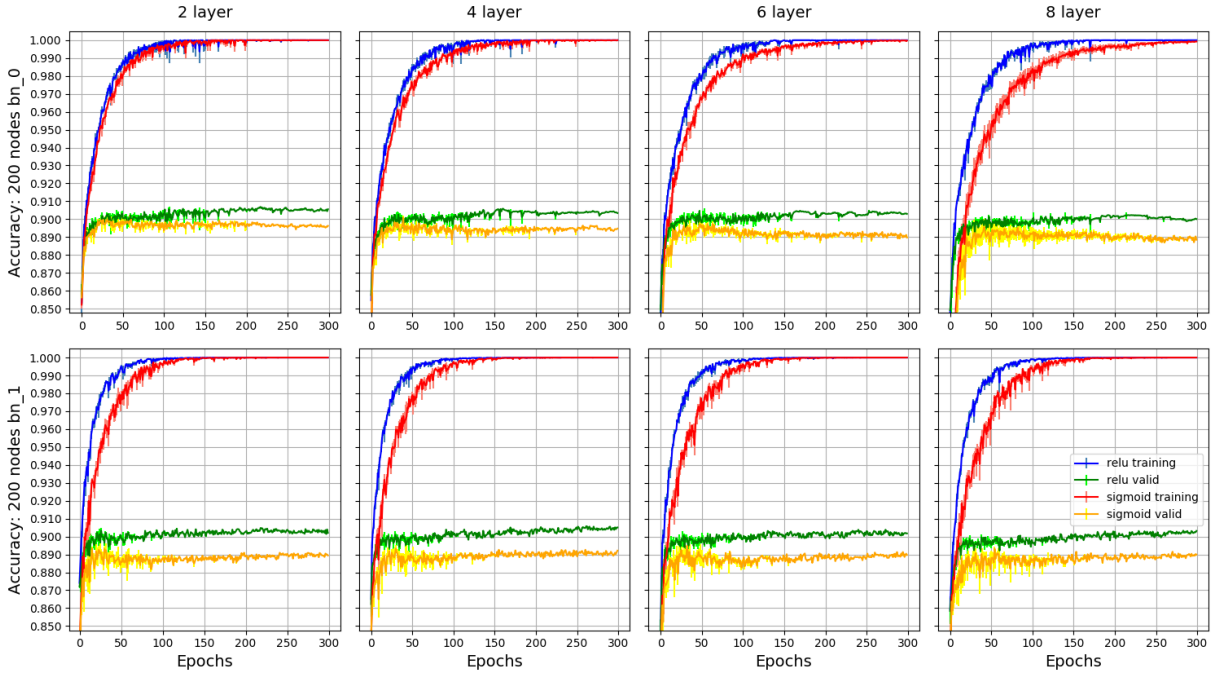


Figure 3.5: Comparison of training and validation curves of ReLU and sigmoid networks with increase in depth and a fixed width of 200 nodes per layer trained on the FMNIST dataset.

Figure 3.7 shows the average evaluation accuracy of different network architectures over different seeds trained on the FMNIST dataset. We again observe that when networks are evaluated on the 10 000 unseen samples, the ReLU networks tend to generalize better than the sigmoid networks, with the exception of shallower networks that have a width of 800 nodes. It is observed that an increase in the number of hidden layers usually causes the networks to generalize slightly worse, except for networks trained with batch normalization that have a width of 200 nodes. We observe that the sigmoid networks

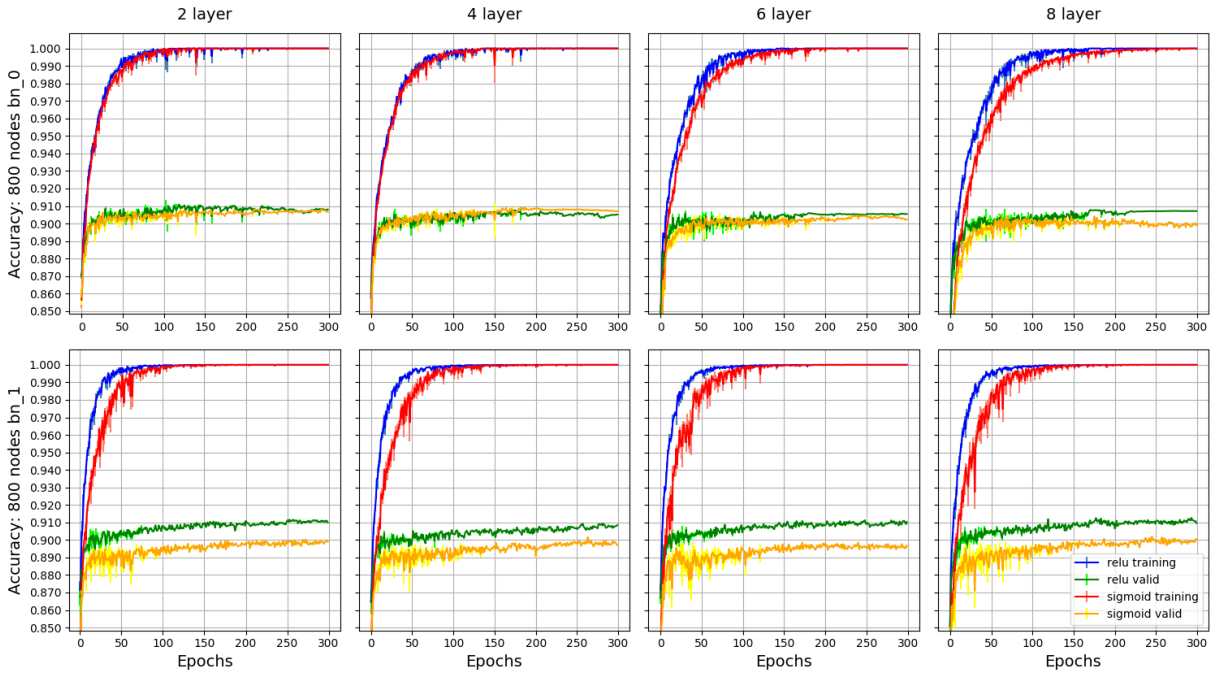


Figure 3.6: Comparison of training and validation curves of ReLU and sigmoid networks with increase in depth and a fixed width of 800 nodes per layer trained on the FMNIST dataset.

trained without batch normalization generalized better than those trained with batch normalization. We discuss possible reasons for these discrepancies in Section 3.5.

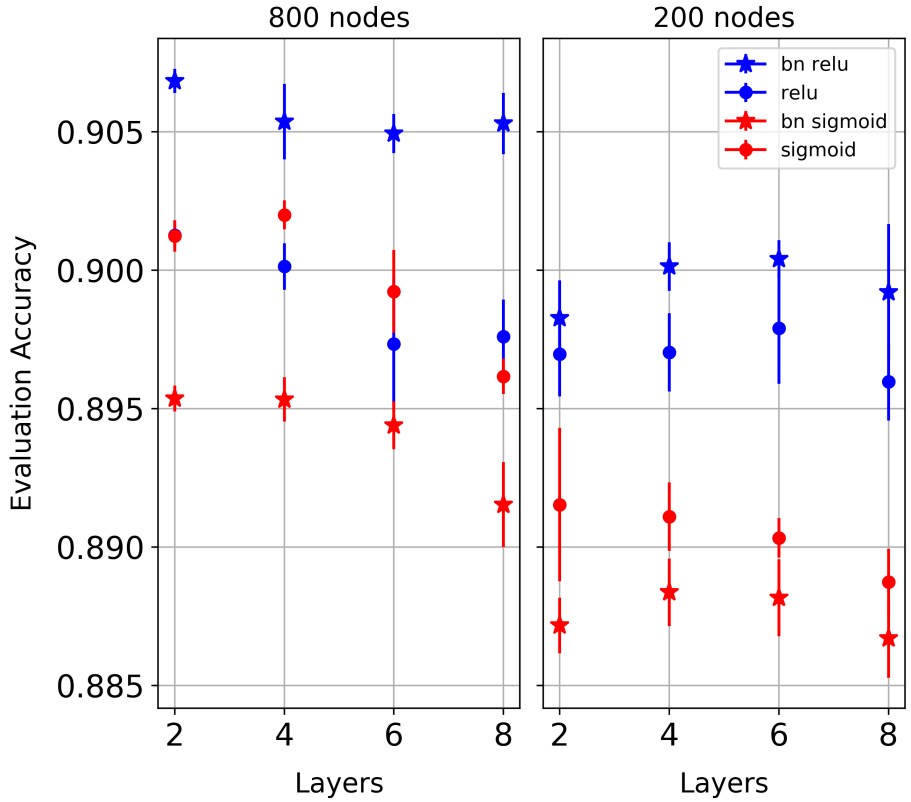


Figure 3.7: A summary of the average evaluation accuracy for each network configuration trained on the FMNIST dataset.

### 3.4.3 CIFAR10

The CIFAR10 dataset is the most complex of the three tasks. There are numerous contrasting features between classes and even class-specific samples. Samples have 3 channels of 1 024 pixels that translate to an input layer of 3 072 features for an MLP. CIFAR10 is used more commonly as a dataset to benchmark convolutional neural networks. We therefore expect fully connected networks not to perform nearly as well compared to MNIST and FMNIST. The differences in accuracy vary more when comparing architecture configurations due to the increased complexity of the dataset.

From Figure 3.8 we observe that after 300 epochs, the networks with 2 hidden layers trained without batch normalization struggle to fit the dataset. When the networks have sufficient parameters, as in the case of 4 hidden layers and deeper, the ReLU networks can fit the training data and converge to a 100% training accuracy while the sigmoidal

networks struggle to learn the training set with more hidden layers. When using batch normalization, both ReLU and sigmoid networks fit the data appropriately. In all configurations, except the 6- and 8-layer ReLU networks trained with batch normalization, the highest validation accuracy is achieved in the first couple of epochs, before starting to overfit thereafter. The shallower sigmoidal networks have comparable validation accuracy to ReLU networks when batch normalization is not used. When batch normalization is used, the ReLU networks outperform the sigmoid networks in shallow and deeper networks.

From Figure 3.9 we see that the wider networks of 800 nodes can fit the training data much better and earlier than the networks in Figure 3.8. When trained without batch normalization, we again see that the sigmoid networks have comparable performance to the ReLU networks with fewer hidden layers. When trained using batch normalization, the wider ReLU networks all outperform the sigmoid networks. Using batch normalization on the wider networks lessens the initial overfitting as seen in all other configurations, especially on the ReLU networks.

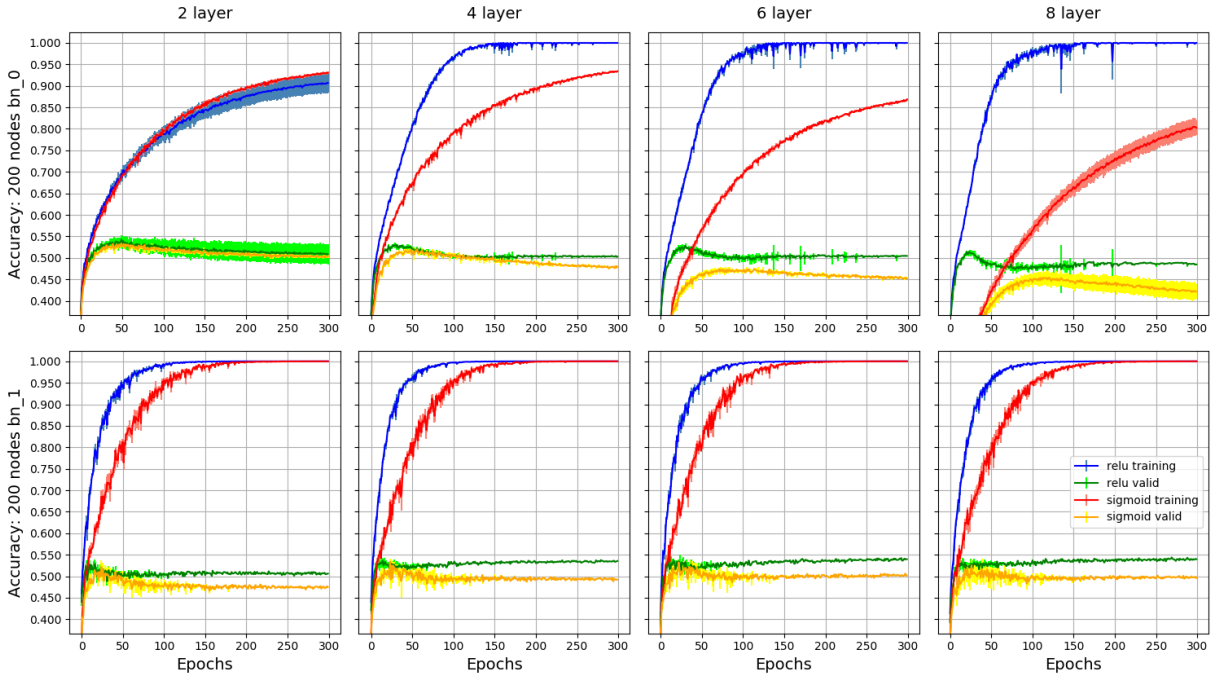


Figure 3.8: Comparison of training and validation curves of ReLU and sigmoidal networks with increase in depth and a fixed width of 200 nodes per layer trained on the CIFAR10 dataset.

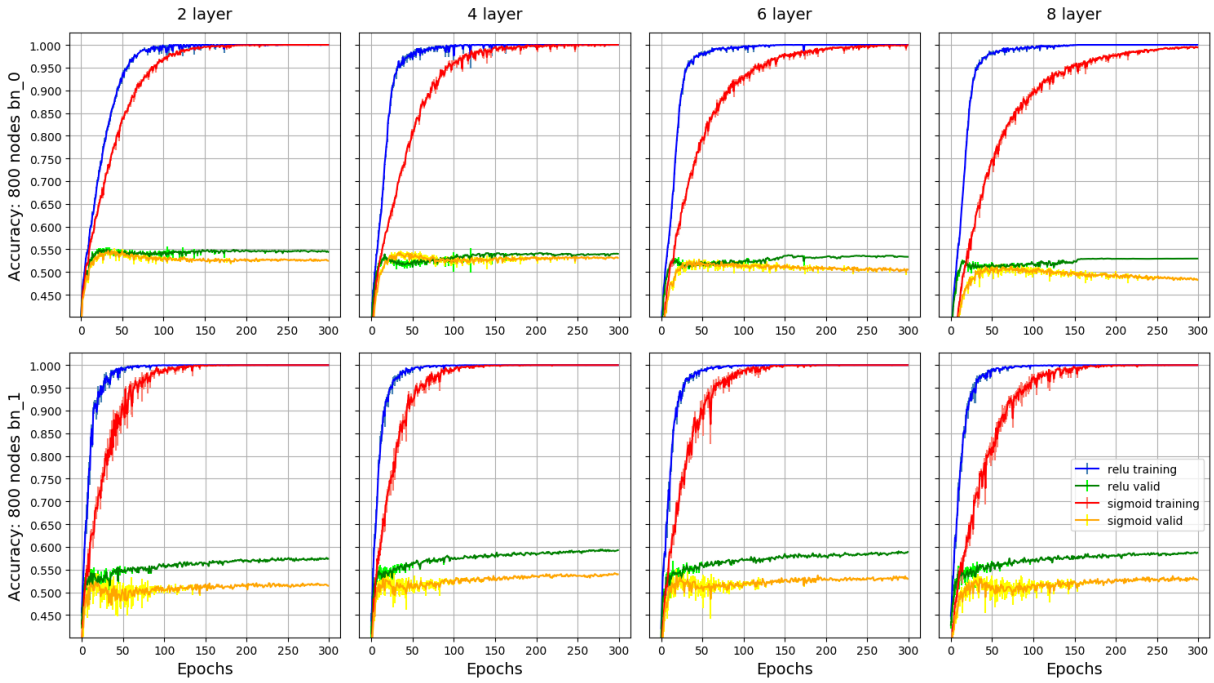


Figure 3.9: Comparison of training and validation curves of ReLU and sigmoidal networks with increase in depth and a fixed width of 800 nodes per layer trained on the CIFAR10 dataset.

Figure 3.10 shows the average evaluation accuracy on the test set of each architecture configuration. When not trained with batch normalization, the ReLU and sigmoid networks generalize less well with an increase in depth. The sigmoid networks perform similarly to ReLU networks in all configurations where batch normalization is not used, except for the 6- and 8-layer networks that are 200 nodes wide. The sigmoid networks generalize relatively poorly with these two configurations compared to the other configurations. This poor generalization could be attributed to the vanishing gradient problem since these two network architectures struggle to fit the training set. In contrast, we see a general increase in evaluation accuracy when increasing the number of hidden layers while training with batch normalization.

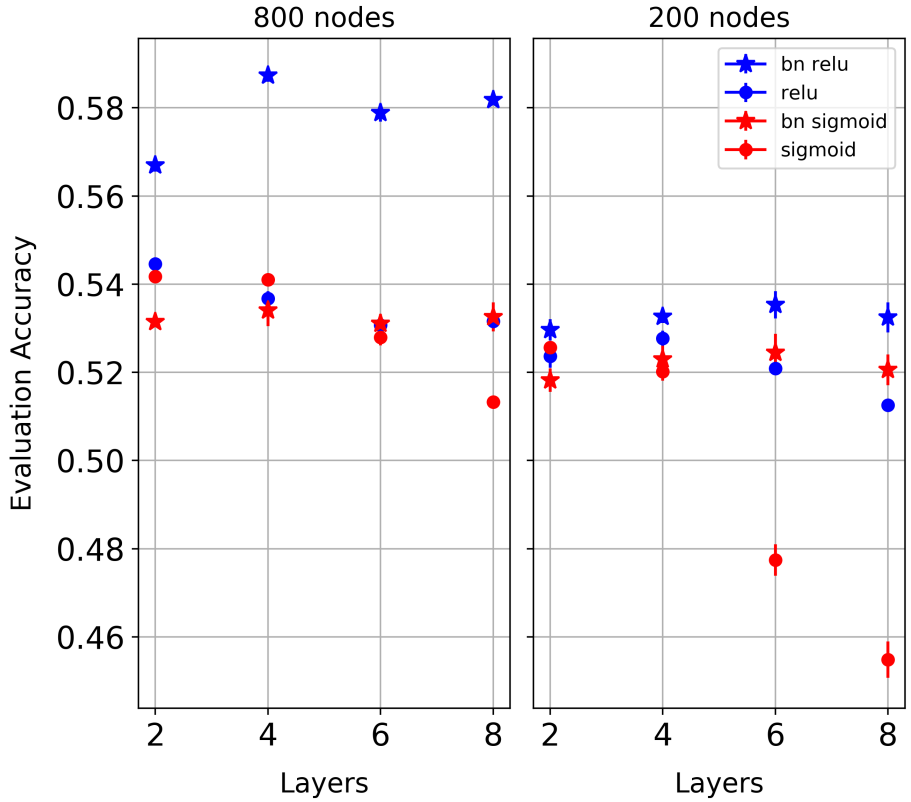


Figure 3.10: A summary of the average evaluation accuracy for each network configuration trained on the CIFAR10 dataset.

## 3.5 Discussion

In this section possible reasons for discrepancies and differences in results are discussed.

A possible reason for the drop in performance in Figures 3.4 and Figure 3.7 with an increase in the number of hidden layers could be attributed to the over-parameterized models causing the networks to overfit on the relatively easy MNIST and FMNIST tasks. Another reason for the decrease in performance could be that over-parameterization makes it much harder to find a set of parameters that will generalize well, in the large solution space created. The high dimensionality of deeper networks could generally make initial convergence harder because of all the available directions when navigating the loss landscape.

The increase in training and generalization performance when using batch normalization could be contributed to its regularizing and stabilizing effect [19]. The regularizing effect

is achieved by introducing stochastic elements in the form of the mini-batch standard deviation and mean that is respectively divided and subtracted at each node in a layer. This fact in addition to that each mini-batch is randomly shuffled adds regularizing noise to the training process, similar to that of dropout [20].

There is no concrete evidence for the improved generalization performance for the sigmoid networks trained without batch normalization vs. those trained with batch normalization on the FMNIST dataset. A possible reason for this observation could be that, for the complexity of the specific task, the normalization of outputs is disadvantageous due to the calculation of the mean and standard deviation from a uniform distribution while the distribution of activation values in sigmoid networks are highly non-uniform; this is investigated further in Section 4.2.

## 3.6 Conclusion

In this chapter several DNN architecture configurations were optimized, trained and evaluated to investigate the difference in the training and generalization performance of networks trained with ReLU and sigmoidal activation functions.

From the results shown in Section 3.4 it was observed that:

- Networks trained with ReLU activation functions tend to outperform sigmoidal networks when not trained with batch normalization, even though the difference in performance is small, especially for shallower networks.
- ReLU networks trained with batch normalization always outperform sigmoidal networks trained with batch normalization, leading us to believe that batch normalization is more advantageous for networks trained with ReLU activations.
- Wider networks trained without batch normalization have more comparable performance between ReLU and sigmoidal networks.

- When using batch normalization, the increase in depth generally increases the generalization performance, specifically on the complex CIFAR10 dataset. This is not always the case for simpler tasks; reasons for deeper networks performing less well are discussed in Section 3.5.

- From Section 3.5, we suspect that over-parameterization and many hidden layers hurt the performance of sigmoid networks more so than networks with ReLU activations.

- An interesting observation made is that, with the intermediate difficulty of the FMNIST dataset, sigmoid networks trained without batch normalization have better validation and evaluation accuracy than those trained with batch normalization.

In summary, then, we observe that optimal generalization is obtained when batch normalization is used to train wide ReLU networks; for CIFAR10, network depth provides a small additional benefit. In contrast to [7] we cannot ascribe these benefits to the vanishing gradients problem, since all our networks, apart from the 200-width sigmoid networks from Figure 3.8, can train to virtually perfect classification of the training set. An alternative explanation is therefore required, and the next chapter investigates a number of clues related to such an explanation.

# Chapter 4

# Node Distributions

---

*In this chapter we introduce the concept of node activation distributions and describe how they differ for networks trained with ReLU and sigmoidal activation functions. The concepts in this chapter lays the groundwork for the rest of the dissertation.*

---

## 4.1 Introduction

In Chapter 3 we discussed and compared the difference in performance of networks trained with ReLU and sigmoidal activation functions on different datasets and network architectures. To better understand the effect of activation functions on the training and generalization performance of DNNs, we investigate the behavior of nodes after the activation function is applied. We specifically look at the activation values for each class at each node to investigate how class-distinctive information is separated and propagated throughout the network.

We choose to show results only for the 4-layer networks as they have moderate depth and similar trends are observed in deeper and wider networks. Results reported in this chapter

are generated for a network with 4 hidden layers with a constant width of 200 nodes trained on the MNIST dataset, unless stated otherwise. We also show the difference in node behavior for networks trained on the more complex CIFAR10 dataset. In addition, we compare the difference between networks trained with and without batch normalization, as the application of batch normalization to a network changes the behavior of nodes with both ReLU and sigmoid activation functions. The accuracy of these 4-layer networks are reported in Figure 3.4 for MNIST: reaching evaluation accuracy of 98.6% for ReLU and 98.18% for sigmoid when not using batch normalization. For CIFAR10 the evaluation accuracy is reported in Figure 3.10: reaching accuracy of up to 51.04% for ReLU and 52.5% for sigmoid when not using batch normalization. For networks trained with batch normalization on the MNIST dataset, ReLU networks reached an accuracy of 98.71% vs 98.36% for sigmoid. For the CIFAR set, batch normalization networks reached 54.03% for ReLU networks and 53.03% for sigmoid networks.

## 4.2 Node activation distributions

The term "activation distribution" is used to refer to the distribution of activation values for the samples of a class at a hidden node after the activation function has been applied. When relevant, the term "pre-activation distribution" is used to refer to the class information at the input of a node.

At each hidden node $h_{i,j}$ we calculate the class activation distributions $(a_{c,i,j})_{x_{c_1}}^{x_{c_M}}$ for each class $c$ after applying the activation function $T$ so that:

$$a_{c_m,i,j} = T(\sum_{k=0}^{s_{(i-1)}} w_{i,j,k} h_{i-1,k}) \qquad 1 \leq i \leq N \tag{4.1}$$

with sample $x_{c_m}$ as input, so that:

$$(a_{c,i,j})_{x_{c_1}}^{x_{c_M}} = \{a_{c_1,i,j}, a_{c_2,i,j}, a_{c_m,i,j}, ..., a_{c_M,i,j}\} \qquad 1 \leq i \leq N \qquad 1 \leq c \leq C \tag{4.2}$$

where $c$ is the class index, $i$ the layer index, $j$ the node index and $s$ the number of nodes in a layer. $x_{c_1}$ is the first sample in a class $c$ and $x_{c_M}$ the last sample with $C$ the number

of classes. For each class activation distribution $(a_{c,i,j})_{x_{c_1}}^{x_{c_M}}$ at each node we calculate the median of the distribution.

When training a DNN with non-linear activation functions, there exists points where the learning process is saturated and gradients reach zero or near-zero values, depending on the activation function. For ReLU this saturation point is at 0.0, where values equal to or below this point have zero gradient and learning subsides for that specific sample at that node. The output of a node, after the sigmoid function is applied, has two of these saturation points, at 0.0 and 1.0 respectively. For the sigmoid function, gradients never completely reach zero when nearing these saturation points, but they become very small and any weight update with this gradient has almost no effect. The position of the median with regard to the saturation points, and with regard to medians from activation distributions of other classes at a node, provides an indication of how the class information is separated among different classes. At each node we are specifically interested in seeing the relative position of the range of activation values for one class with respect to to the saturation points as well as the range of activation values of another class. The use of the median of the activation values is suited as a rough measurement for this purpose as it does not entirely matter if distributions are multi-modal or mono-modal (although activation distributions caused by the sigmoid and ReLU activations are mostly mono-modal, with sigmoid activation distributions having some bi-modal tendencies).

### 4.2.1 Sigmoid

Figure 4.1 shows the pre-activation distributions for generic nodes in the hidden layers of a sigmoid network. Nodes are randomly selected and are meant as generic representations of nodes in earlier and deeper layers, they show typical behavior of nodes in those layers. These activation distributions represent the information flowing into a node before the activation function is applied.

From Figure 4.1 it is observed that the pre-activation distributions of nodes in the first hidden layer (top row) tend to overlap, while pre-activation distributions flowing into

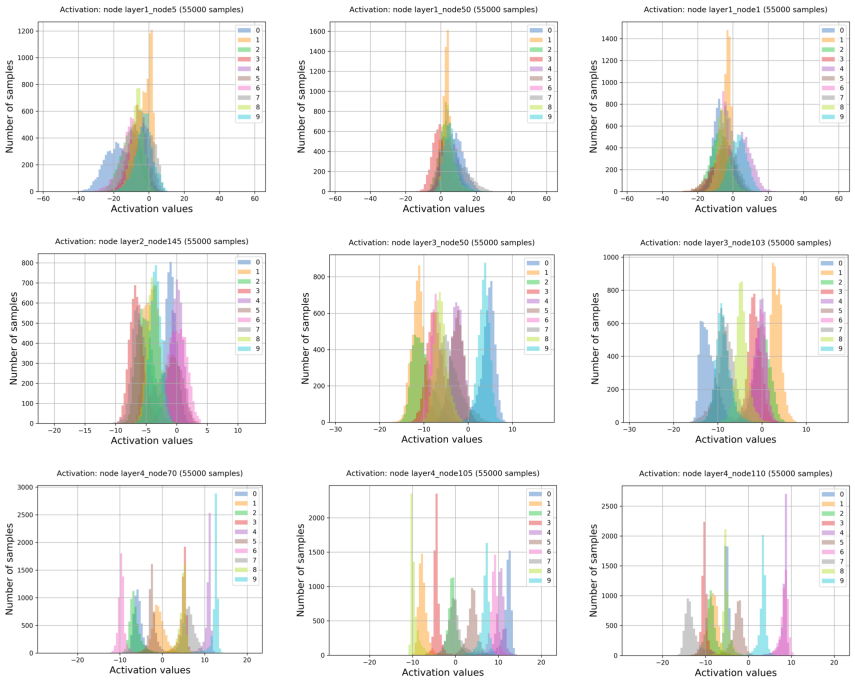


Figure 4.1: Activation distributions of generic nodes in shallow (top row) and deeper (bottom row) layers before sigmoid activation function is applied. (MNIST)

nodes of the second and third hidden layer tend to be more separated from one another. This is especially true for nodes in the last hidden layer as seen from the bottom row of Figure 4.1.

Figure 4.2 shows the activation distributions of specific hidden nodes after the sigmoid activation function has been applied. It is observed that the activation distributions are highly non-Gaussian and are skewed towards the 0.0 and 1.0 saturation points. Interestingly the outputs of the second hidden layer seems to be less saturated with higher variance. (This phenomenon is observed at the second layer for all deeper networks as well.) The bottom row shows typical activation distributions at nodes in the last hidden layer. These distributions are heavily skewed towards the saturation points, with some distributions almost being point distributions with very low variance. We observe that when class information is separated, activation distributions still overlap each other near the saturation points.

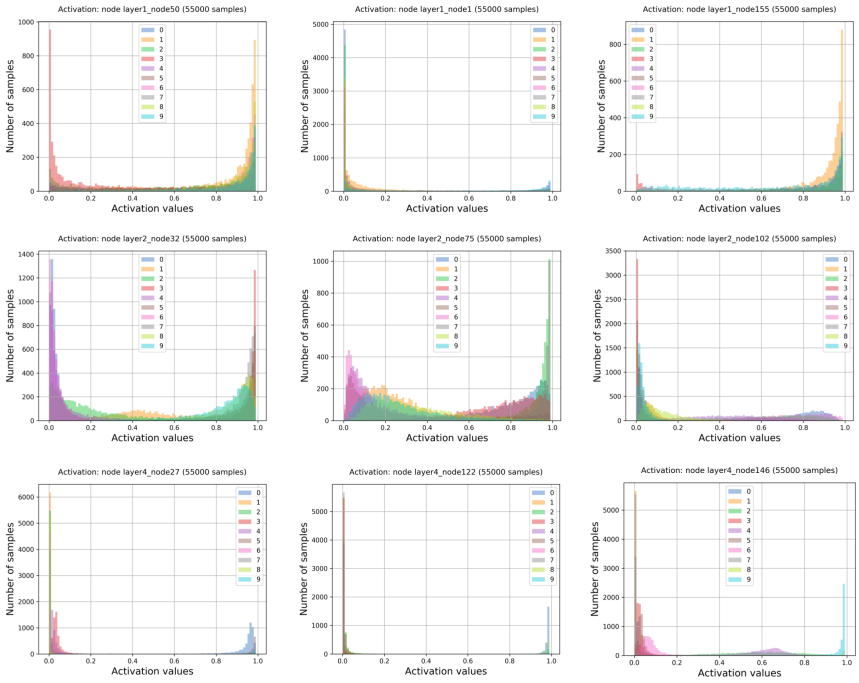


Figure 4.2: Activation distributions of generic nodes in shallow (top row) and deeper (bottom row) layers after sigmoid activation function is applied. (MNIST)

### Distribution divergence: untrained vs. trained

To understand how class information is separated and propagated, we plot the median of the activation distributions of each class at each node. Figures 4.3 through 4.5 shows the median values for the distributions after the activation function has been applied. The x-axis indexes the nodes in the networks. For a 4x200 network this means that nodes 1-200 make up layer one, nodes 201-400 layer two, etc. Since the activation distributions of the sigmoid networks are highly non-Gaussian, we track the position of activation distributions by plotting the median of the distributions. By using the median values, we regard the distribution as a whole while not focusing on the location of the distribution mass.

Looking at Figure 4.3, if we take the untrained model with initialized weights, it is observed that the medians are mostly all clustered around 0.5 for the first hidden layer. The medians in the second hidden layer (nodes 200-400) are slightly more saturated towards 1.0 and 0.0 and the median values are starting to overlap more for classes at the

same node. From nodes 400 to 600 it is observed that the medians of the distributions overlap significantly, meaning that with the untrained weights the nodes are not able to distinguish activation values of one class from the others. In the last hidden layer (nodes 600-800) it is observed that the activation distributions lay almost directly on top of each other and class information cannot be separated.

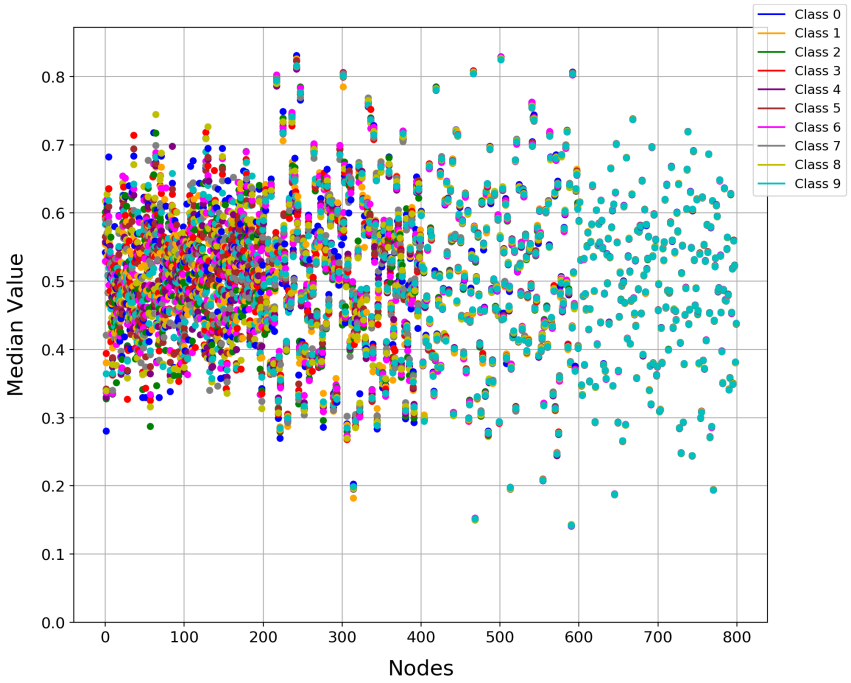


Figure 4.3: Medians of activation distributions for each class at every node in a 4x200 network. Generated for a untrained model with sigmoid activation functions. (MNIST)

From Figure 4.4 we see that when the model is well trained, class information is separated by saturating medians towards 0.0 and 1.0. This effect is more clear at deeper hidden layers (nodes 400-599 and 600-800), while more nodes in earlier layers have activation distributions with median values slightly closer to 0.5. This observation possibly indicates that deeper layers are more effective in separating class-distinctive information in a trained sigmoidal network.

Figure 4.5 shows the medians of activation distributions for the same network trained with batch normalization. It is observed from the relative position of median values, that activation distributions are not as saturated towards 0.0 and 1.0 compared to Figure 4.4. Class information is still separated, albeit less effectively than the network trained without batch normalization. We observe that the overlap of median values near the saturation

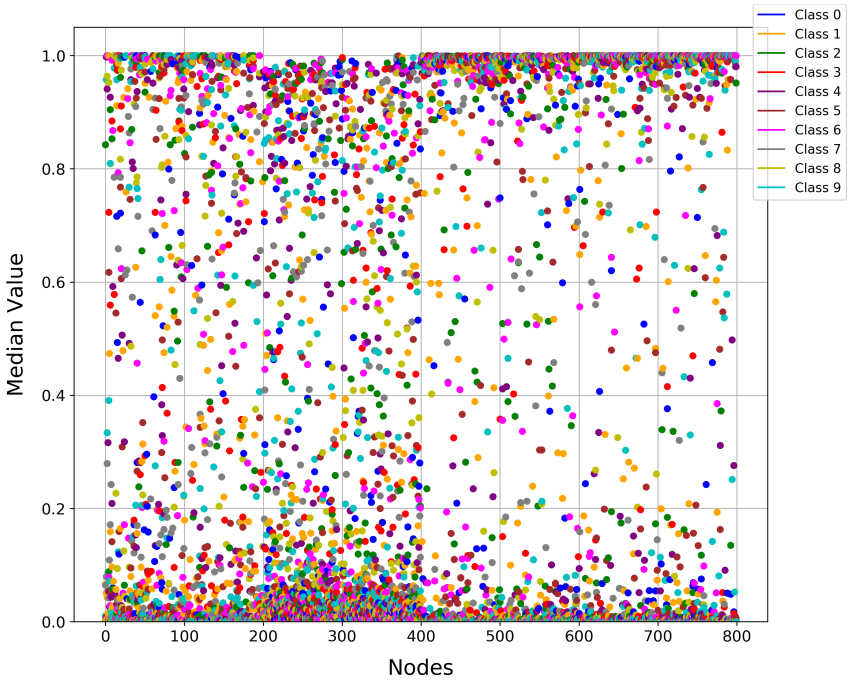


Figure 4.4: Medians of activation distributions for each class at every node in a 4x200 network. Generated for a trained model with sigmoid activation functions. (MNIST)

points are less when training with batch normalization. The network trained with batch normalization achieved higher training and test accuracy. Possible reasons for the effect of batch normalization on node behavior is discussed in Section 4.4.

### 4.2.2 ReLU

Figure 4.6 shows the pre-activation distributions for specific nodes in the hidden layers of a trained ReLU network. The nodes shown are randomly chosen to show the typical behavior of nodes in those layers.

From Figure 4.6 it is observed that the pre-activation distributions of nodes in the first layer have Gaussian shape and overlap near 0. The pre-activation distributions of specific classes at nodes of the second, third and fourth hidden layer seem to be separated to a greater extent while other class distributions overlap more. These class distributions, specifically in the last hidden layer, are being pushed towards values in the positive domain (i.e. values bigger than 0) while the overlapping distributions are at values closer to 0.0

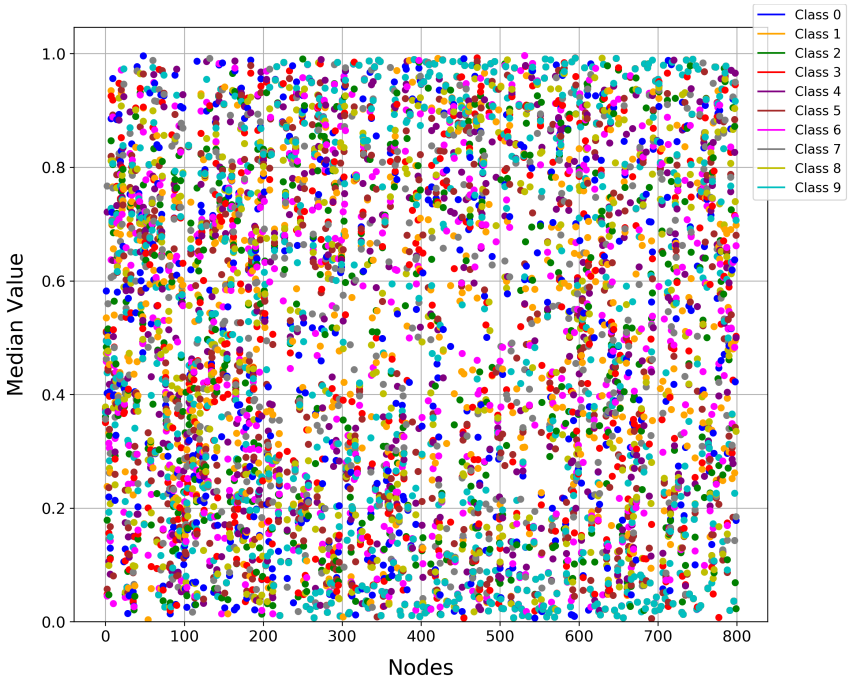


Figure 4.5: Medians of activation distributions for each class at every node in a 4x200 network. Generated for a trained model with sigmoid activation functions and batch normalization. (MNIST)

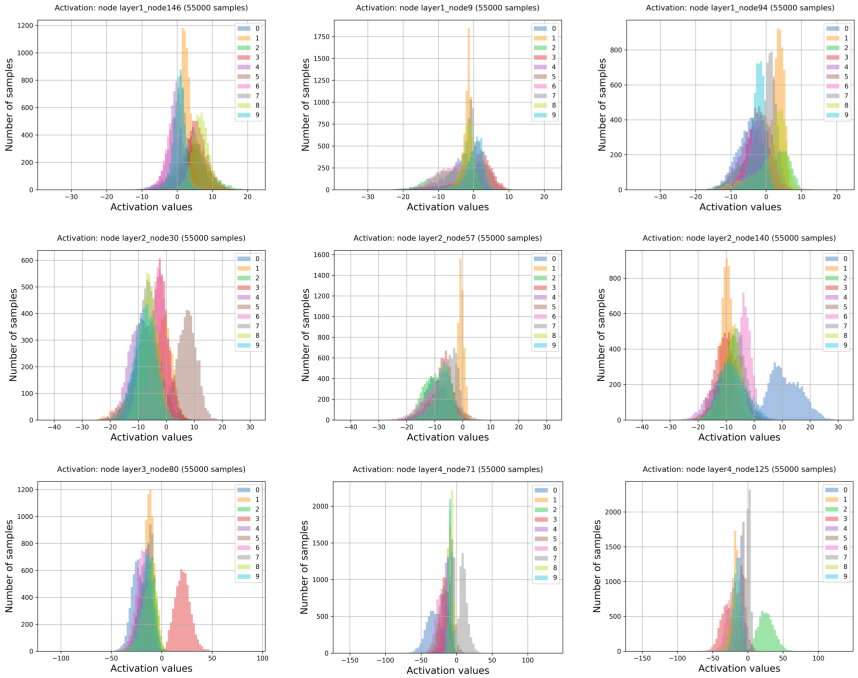


Figure 4.6: Activation distributions of generic nodes in shallow (top row) and deeper (bottom row) layers before ReLU activation function is applied. (MNIST)

or in the negative domain.

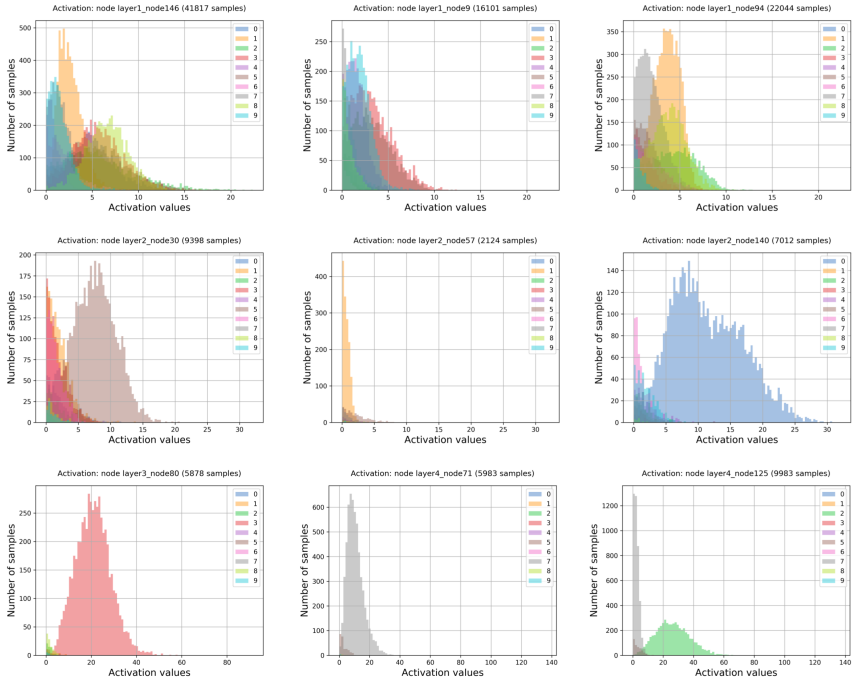


Figure 4.7: Activation distributions of generic nodes in shallow (top row) and deeper (bottom row) layers after the ReLU activation function is applied. (MNIST)

Figure 4.7 shows the typical output of specific hidden nodes in the network after the ReLU activation function has been applied at the node. It is clear that the activation distributions are cut off at 0.0 since activation values below 0.0 are rectified while the positive values retain their value. The distributions in the positive domain retain some of the same Gaussian shape, depending on how many samples have negative activation values. In the positive domain, the activation distributions are not saturated and do not overlap as heavily for different classes such as the output of nodes in Figure 4.2. Not only do the nodes in deeper layers become more specialized towards one class, they activate for fewer samples than the nodes in earlier layers. This confirms the ability of ReLU trained networks to be sparse. This effect is further investigated and discussed in Chapter 5.

### Distribution divergence: untrained vs. trained

For ReLU networks the mean and the median are almost identical due to the Gaussian shape of their activation distributions. Figure 4.8 shows the untrained model for the network trained with ReLU activation functions. The median values of node distributions are less saturated in earlier layers and saturate more towards 0.0 for deeper layers. It is important to note that the activation distributions in deeper layers do not overlap as strongly compared to the sigmoid network in Figure 4.3. The ability of the ReLU network to completely suppress class information and not activate for specific samples allows the network to still separate distributions of different classes in deeper layers. Even when distributions are not separated in a meaningful (trained) way, the inherent sparse structure that the ReLU activations introduce suggests better separation of class information. There is not yet a clear link between this observation and the generalization ability of ReLU networks.

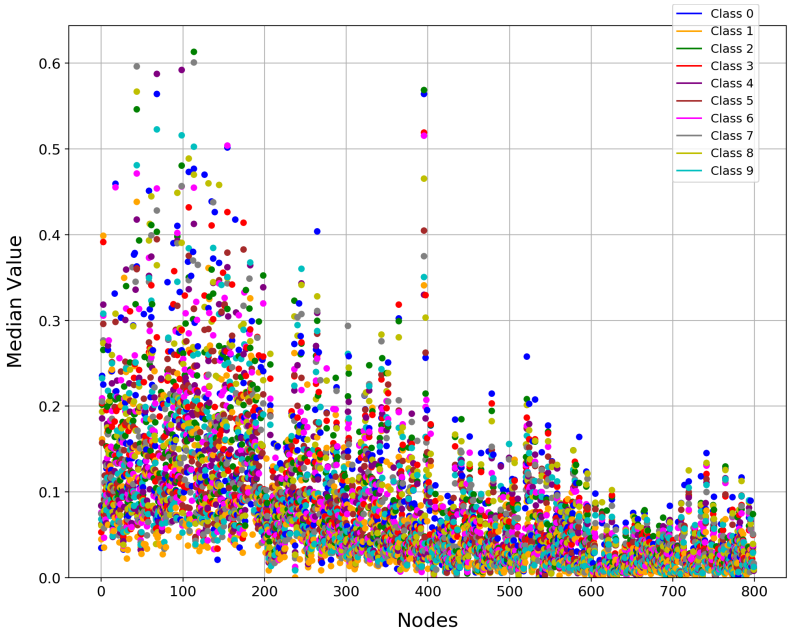


Figure 4.8: Medians of activation distributions for each class at every node in a 4x200 network. Generated for an untrained model with ReLU activation functions. (MNIST)

From Figure 4.9 it is observed that when a ReLU network is well trained, the activation distributions in earlier layers have lower activation values and class information is suppressed with overlap of activation distributions of classes. This observation with the

behavior seen in Figure 4.7 indicates that nodes in earlier layers remain relatively agnostic towards classes. The nodes in deeper layers have less overlap and nodes become more specialized towards specific classes. We also see that once the model is trained, activation values are increased significantly, meaning that if the node is active for a sample, it activates more strongly compared to the untrained model.

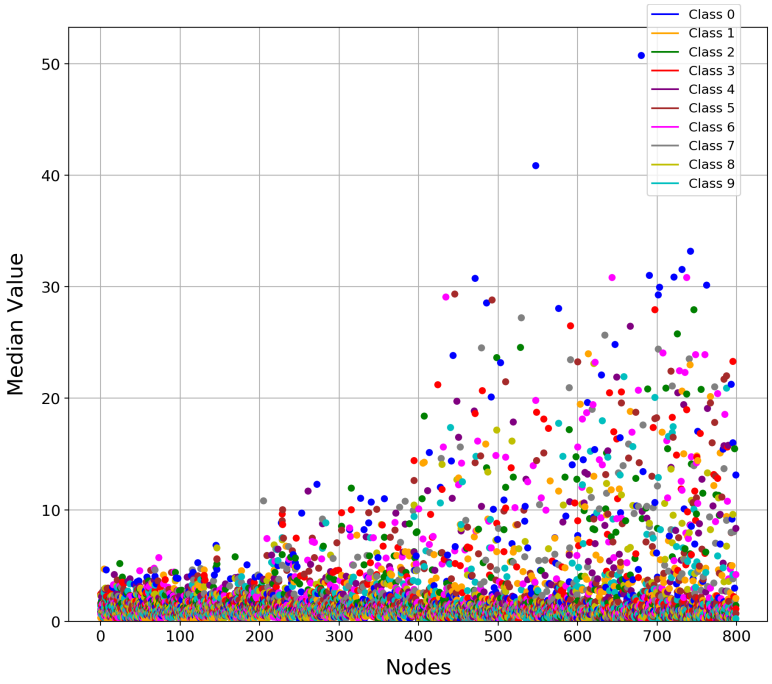


Figure 4.9: Medians of activation distributions for each class at every node in a 4x200 network. Generated for trained model with ReLU activation functions. (MNIST)

Figure 4.10 shows the same network trained with batch normalization. Here it is observed that the activation distributions at nodes in earlier layers have higher relative median values compared to the network trained without batch normalization. The distributions are off-set from 0.0 in nodes 0-200, suggesting that batch normalization forces the nodes in the first layer to be more active towards all classes and not suppress class information. The nodes in deeper layers also seem less saturated but still become more specialized towards specific classes. The effect of normalizing outputs with the mean and variance of mini-batches also has the indirect effect that weights are kept smaller and overall activation values are smaller than when trained without batch normalization.

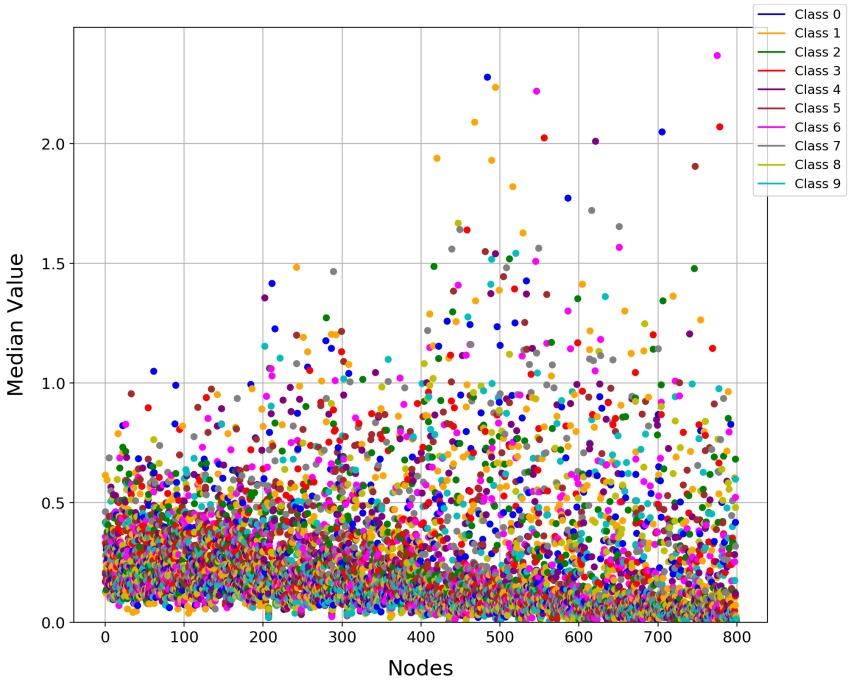


Figure 4.10: Medians of activation distributions for each class at every node in a 4x200 network. Generated for trained model with ReLU activation functions and batch normalization. (MNIST)

## 4.3 CIFAR10

To investigate the behavior of nodes and activation distributions on a more complex task, we illustrate the median values of a 4x200 network trained with ReLU and sigmoidal activation functions on CIFAR10.

Figure 4.11 shows the median values of activation distributions of networks trained on the CIFAR10 dataset. If the network cannot effectively fit the more complex data, class information cannot be effectively separated and the median values look more similar to the initialized models in Figure 4.3 and Figure 4.8.

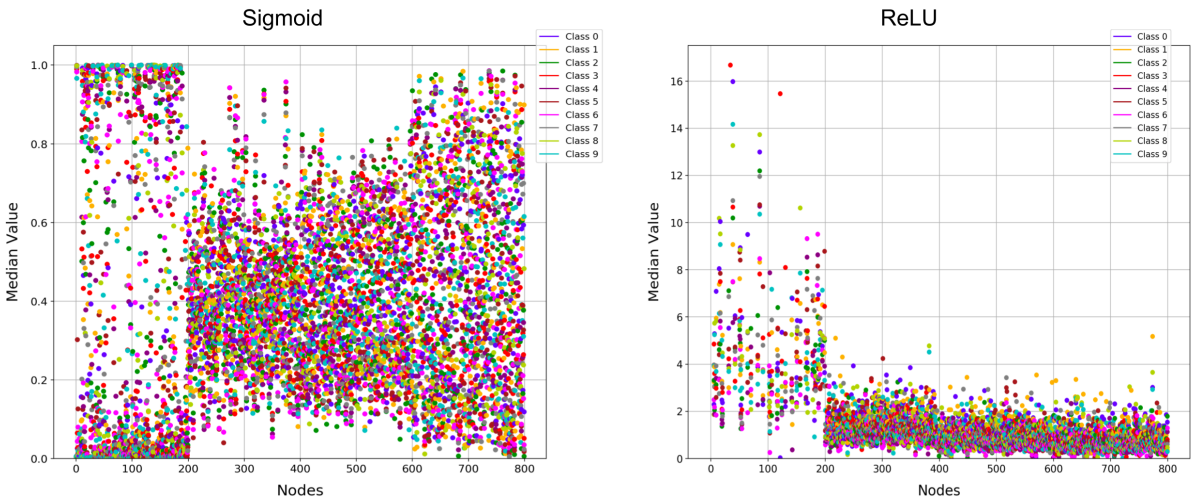


Figure 4.11: CIFAR10: Median values of activation distributions for each class at every node in a 4x200 network, trained with sigmoid (left) and ReLU (right) activations.

## 4.4 Discussion

In Figures 4.1 and 4.6 it is clear that class-distinctive information is better separated in later layers vs. earlier layers. The information flowing into the first hidden layer is a linear representation of the input features as no activation function has been applied yet. This is why class information is not effectively separated and pre-activation distributions in the first hidden layer overlap heavily. The outputs from the first hidden layer give the following layers a non-linear representation of the input sample due to the activation function being applied to the outputs of these nodes. The pre-activation distributions of nodes in deeper layers are separated more effectively by the collective non-linear outputs of previous layers.

From Figure 4.10 and Figure 4.5 it was observed that when networks are trained with batch normalization, the median values are on average further away from the saturation points for ReLU and sigmoid than networks trained without batch normalization. In both cases the networks trained with batch normalization outperformed the networks trained without batch normalization. This could suggest that the over-saturation of nodes is harmful for class discrimination. Goodfellow et al. [20] also discusses the benefits of

additive stochastic elements in DNNs.

For the ReLU network, it is observed from Figure 4.9 and Figure 4.10 that there is a difference in the scale of activation values of nodes. We suggest that this scaling does not affect the discriminatory ability of the network, but rather that the magnitude of activation values with regard to each other is what mainly contributes to the separation of class information.

## 4.5 Conclusion

In this chapter we investigated the behavior of nodes in hidden layers by looking at class activation distributions and how class-distinctive information is separated and propagated through the network for ReLU and sigmoidal activation functions. It was observed that:

- ReLU and sigmoid networks both are able to effectively separate class information, but they do this in fundamentally different ways.
- The sigmoid networks saturate class information towards 0.0 and 1.0 where there is plenty of overlap between activation distributions.
- The ReLU networks saturate class information towards 0.0 for earlier layers with moderate overlap of class distributions, while nodes in later layers separate class information effectively. We suggest that this makes nodes in earlier layers more conservative towards class discrimination while nodes in later layers become more specialized towards single classes.
- When networks are untrained, distributions of sigmoid networks overlap almost exactly while the sparse structure of ReLU networks allow class activations to still be separated, however in an unstructured way.
- Training networks with batch normalization lessens the effect of saturation and nodes have activation distributions that are more similar for classes.

- When networks are trained on the more complex CIFAR10 dataset, class information is not effectively separated and the node behavior looks more similar to that of the untrained model.

Interesting observations were discussed in Section 4.4 and possible reasons for these observations were given. Overall we find that node saturation seems less pertinent than the way in which class-distinctive information is separated and propagated through the networks, and how node behavior differs for ReLU and sigmoid trained networks. Results in this chapter seem to indicate that the ability of ReLU-trained networks to not have excessive overlap of activation distributions in deeper layers is advantageous compared to sigmoid networks that almost always have overlap of class distributions when trained. The impact of activation distributions on generalization performance is investigated further in Chapter 6, but first we consider the other side of the coin: the importance of the discrete behavior of nodes, when we are not concerned with the *size* of an activation, but simply whether it is active or not.

# Chapter 5

# Sparsity, Node Specialization and Dead Nodes

---

*In this chapter we give an introduction to the discrete dynamics of DNNs as we investigate the node activity and sparsity of DNNs trained with the ReLU and sigmoid functions. We theorize on the implications of results in terms of network training and generalization.*

---

## 5.1 Introduction

In Chapter 4 it was investigated how class-distinctive information is separated and propagated throughout a network. This was achieved by plotting the median values of class activation distributions at nodes in hidden layers. These distributions provide a perspective on the available information at each node and how the application of different activation functions either suppresses or passes the information to later layers. We now propose a discrete way to look at node activity by determining the activation *status* at any given node for a sample, rather than an activation value. DNN analysis is simplified by regarding the activation status of a node as either "on" or "off" for any given input

sample. We use this discrete perspective to investigate representation sparsity, as defined in Section 2.2.4, and node specialization in DNNs.

## 5.2 Theta values

When calculating the activation status for a node using sigmoid or ReLU activation functions, it is important to again consider the structural differences of these activation functions. The nodes in sigmoid networks have continuous activation values and are never truly inactive for any sample. When a sigmoid node has a negative input value, it (at the very least) has a positive activation value close to zero. The ReLU network has a discrete piecewise linear structure where nodes are completely inactive for samples that yield input values below or equal to zero; see Figure 1.1. This true zero activation result allows for the sparse representations discussed in Section 5.4. We postulate similar switching behavior for sigmoid networks based on observations made in Chapter 4. Note that while not effectively present in sigmoid networks, sigmoid nodes are also treated as discrete switches to investigate how the discrete behavior of sigmoid and ReLU networks compares during training and generalization.

We define a function $\Theta$ that determines whether a node is considered to be switched "on" or "off" for a given input (note that this theta function is not the same as the matrix $\boldsymbol{\theta}$ defined in Section 2.1). For ReLU this threshold is the point where the activation value is equal to zero, that is, for any zero or negative input at the node. We define $\Theta_{ReLU}(T(h_{i,j}))$ as a version of the unit stepwise function:

$$\Theta_{ReLU}(T(h_{i,j})) = \begin{cases} 1 & \text{if } T(h_{i,j}) > 0 \\ 0 & \text{if } T(h_{i,j}) \leq 0 \end{cases} \tag{5.1}$$

From Chapter 4, we know that activation values of sigmoid networks tend to saturate towards 0.0 and 1.0 when optimized from the untrained state (as also shown in Figure 4.2). We select a threshold activation value where the node is deemed "off" when below this threshold and "on" when above the threshold. We utilize 0.5 as the switching threshold

as it symmetrically divides the sigmoid's activation domain. Since this threshold yields maximum gradient, activation distributions are expected to diverge from this point when the network is training. Therefore when the activation function is a sigmoid we define $\Theta_{sigmoid}(T(h_{i,j}))$ as:

$$\Theta_{sigmoid}(T(h_{i,j})) = \begin{cases} 1 & \text{if } T(h_{i,j}) > 0.5 \\ 0 & \text{if } T(h_{i,j}) \leq 0.5 \end{cases} \tag{5.2}$$

This discrete method of network analysis is an approximation of node activity, especially in the case of the sigmoid function. Information regarding the magnitude of activation values is lost when regarding nodes as switches. We hypothesize, however, that this is an appropriate method of analysis to investigate network sparsity and node specialization. This hypothesis is revisited in Chapter 6.

## 5.3 Node activity

In this section, we show how the node activity of nodes in hidden layers are similar and dissimilar for networks trained with ReLU and sigmoid activation functions. The term "node activity" refers to the percentage of active samples (number of positive theta values as defined in Section 5.2) for a specific class at a specific node. For example, if any given node is activated (has a theta value of 1) for 500 samples of a class that has 5 000 total samples, the node activity is reported as 10% for the specific class at this node.

### 5.3.1 Activity of hidden layers

Node activity is compared for two selected architectures, one relatively shallow network of four hidden layers, and one relatively deep network of eight hidden layers to investigate the activity of nodes in deeper layers. Both architectures have a constant width of 200 nodes. All figures shown are generated from the models trained in Section 3.4.1 on the MNIST dataset unless specifically stated otherwise. These networks achieved a training

accuracy of 100.00% and a test accuracy of 98.55% for ReLU and 98.18% for sigmoid.

Figure 5.1 shows the node activity of the 4x200 ReLU network. The top left figure is generated on the training data (using the trained model), top right on the evaluation data and the bottom figure is generated with the untrained model on the training data. The gray lines are used to distinguish between hidden layers in the network, where nodes 0-199 are from layer 1, 200-399 from layer 2, etc. For the trained model it is observed that earlier layers have fewer nodes that have high activity, while the later layers (third and fourth hidden layer here) either have very high (close to 100%) or very low (close to 0%) activity with fewer mid-range counts. The activity for the evaluation set looks similar to the train set, with the exception that fewer nodes in deeper layers have 100% activation counts. The node activity of the untrained model is unstructured since model weights are randomly initialized from a Gaussian distribution.

These observations reinforce the observations made in Chapter 4 where it was seen that earlier layers of the trained ReLU networks are more agnostic towards classes by having overlapping activation values relative to other classes. Here it is seen that in earlier layers, there are more nodes that have mid-range activity (i.e. not close to 100% or 0% activity) compared to nodes in later layers. This supports our hypothesis that later layers become more specialized by either activating for all samples of a class or none of the samples of a class.

Figure 5.2 shows the node activity of the 4x200 sigmoidal network. Although the activity of nodes looks similar to that of the ReLU network in Figure 5.1, a difference we observe is that the 1st hidden layer has higher average activity, where there are more nodes that activate for most samples of the dataset. The 3rd and 4th hidden layers again have more nodes with activity closer to 100% and 0% with fewer mid-range activation counts than earlier layers. A clear distinction made between the sigmoid and ReLU network is the activity of the untrained model. At deeper layers, nodes of the untrained sigmoid model are either fully dead or fully alive for all samples.

Nodes in deeper layers are also less specialized, with nodes switching on for many samples

of several classes. This is expected due to the way that activation distributions overlap each other near the saturation points when the network is trained, as seen in Figure 4.2.

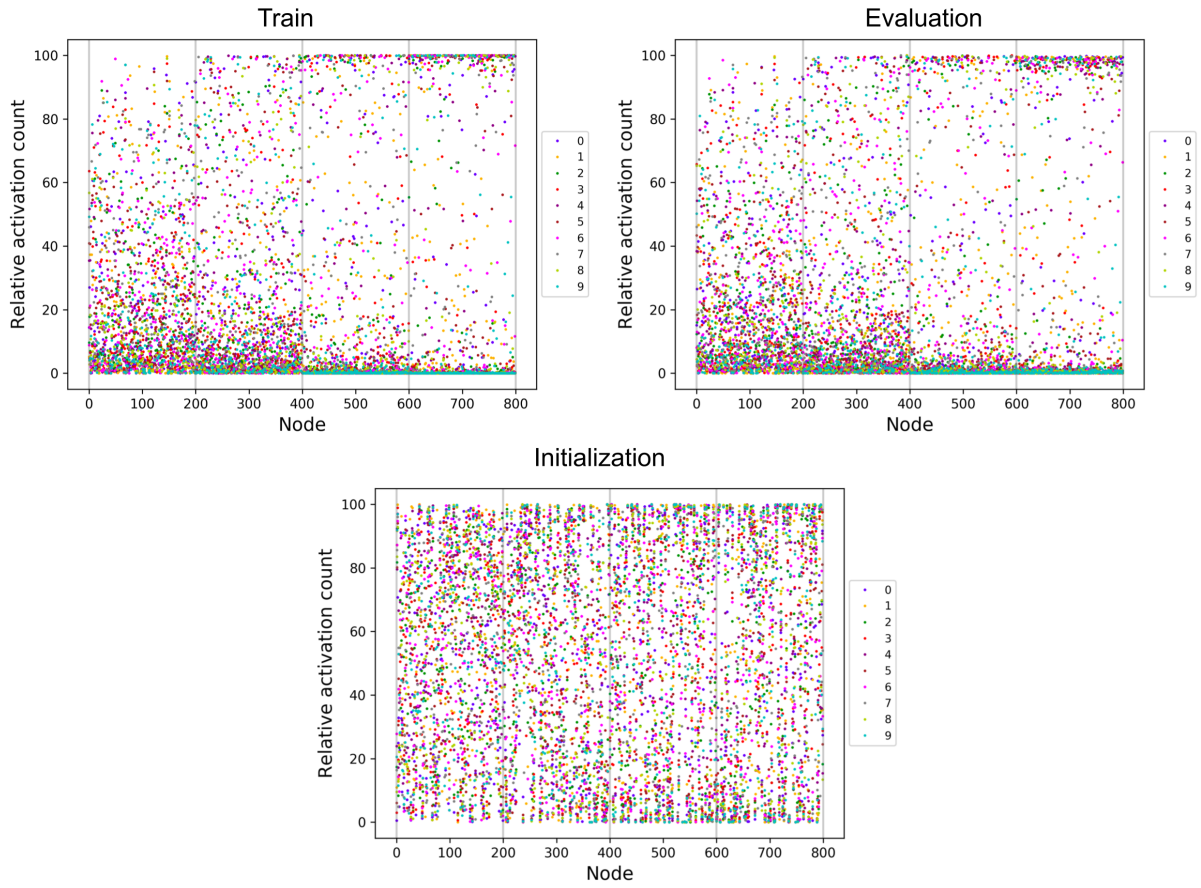


Figure 5.1: Activity per node for a 4x200 ReLU network trained on MNIST.

From Figure 5.3 and 5.4, similar behavior is observed for the deeper 8x200 networks: the tendency towards nodes in later layers being either on or off for *all* samples of a class being even more pronounced.

### 5.3.2 Batch normalization

In Section 3.4 we saw that batch normalization generally improves performance, especially for ReLU networks. How does batch normalization affect node activity? We only investigate the 8-layer networks trained with batch normalization since it best shows trends also observed in shallower architectures.

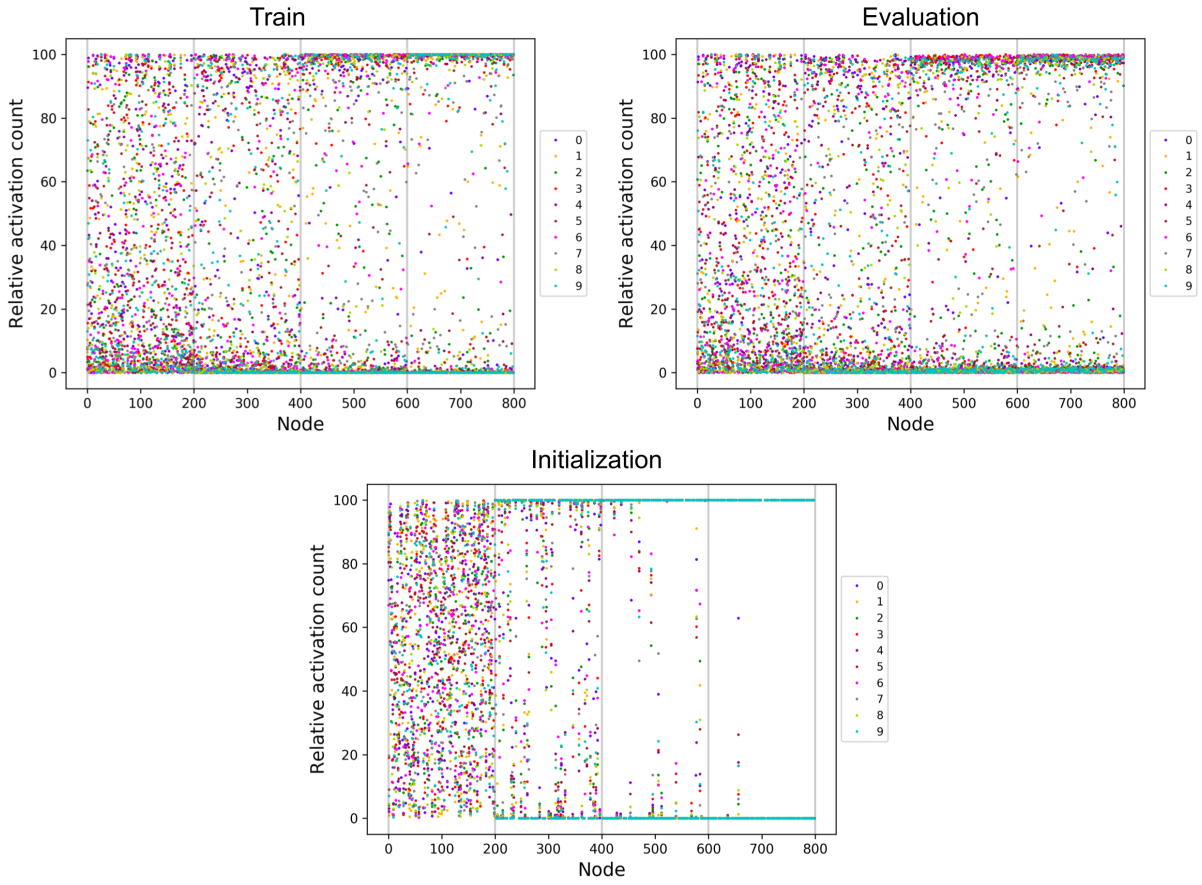


Figure 5.2: Activity per node for a 4x200 sigmoid network trained on MNIST.

It is widely accepted [19], [20] that the application of batch normalization to a network has a regularizing effect on the training process and introduces useful noise that tends to stabilize training. In most cases, this results in better generalization performance, as seen in Chapter 3. From Figure 5.5 we see that when we train networks using batch normalization, the node activity in hidden layers is more moderate for ReLU and sigmoidal networks. We observe that the transition towards more polarized activity occurs at a deeper layer than nodes trained without batch normalization. For the ReLU network this shift towards more specialized nodes is still clear from layer 5 (nodes 800-1000) to deeper layers, compared to the 2nd and 3rd layer for the network trained without batch normalization. The sigmoidal network trained with batch normalization has fewer nodes overall that have either 0% or 100% node activity.

In summary then, it is observed that when nodes are regarded as discrete switches that

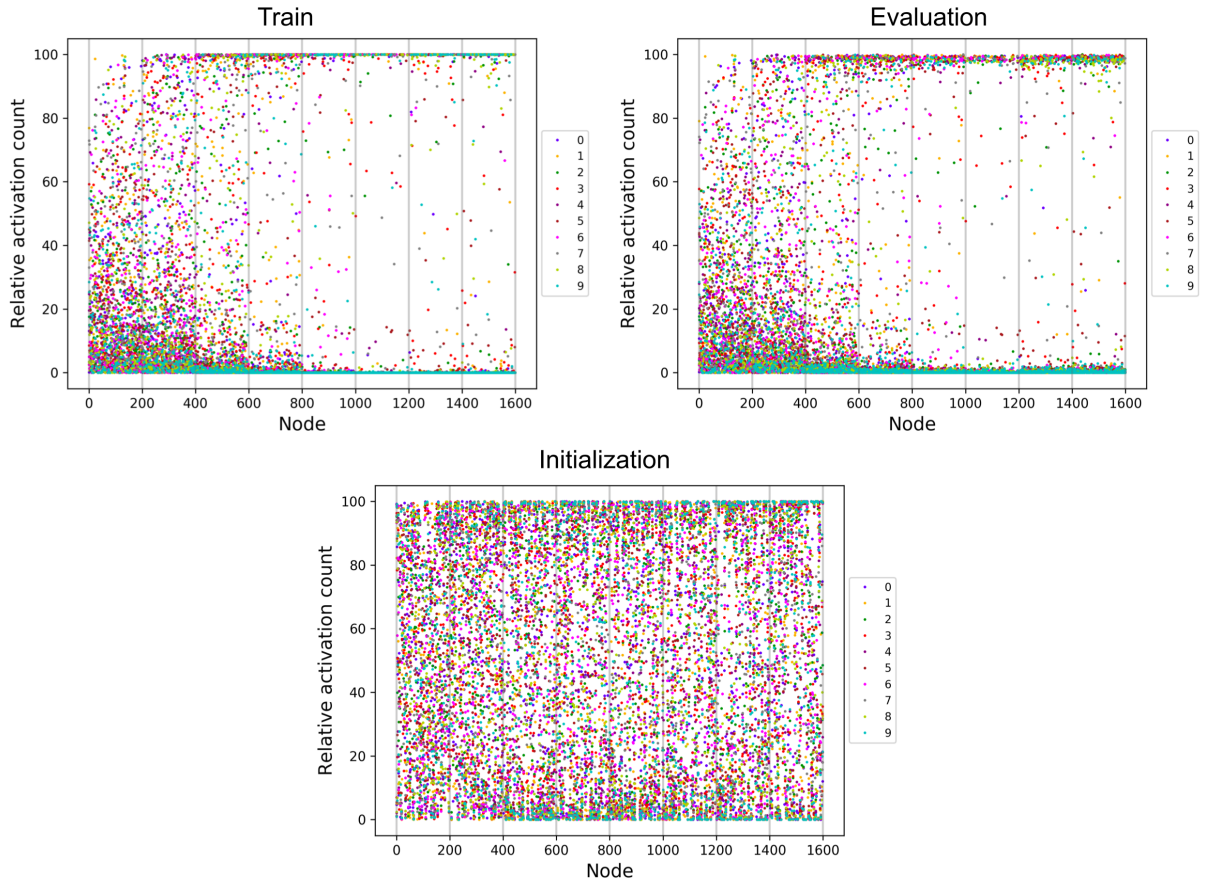


Figure 5.3: Activity per node for a 8x200 ReLU network trained on MNIST.

either switch "on" or "of" for any given sample, the activity of nodes in sigmoid and ReLU trained networks (trained without batch normalization) are surprisingly similar, while the activity of the untrained models vary. This is seen more clearly in networks with many hidden layers. It was also observed that when trained with batch normalization, ReLU networks still have a clear transition to more polarized activity (where nodes become more class-specific) compared to sigmoidal networks that have fewer nodes that exhibit this behavior.

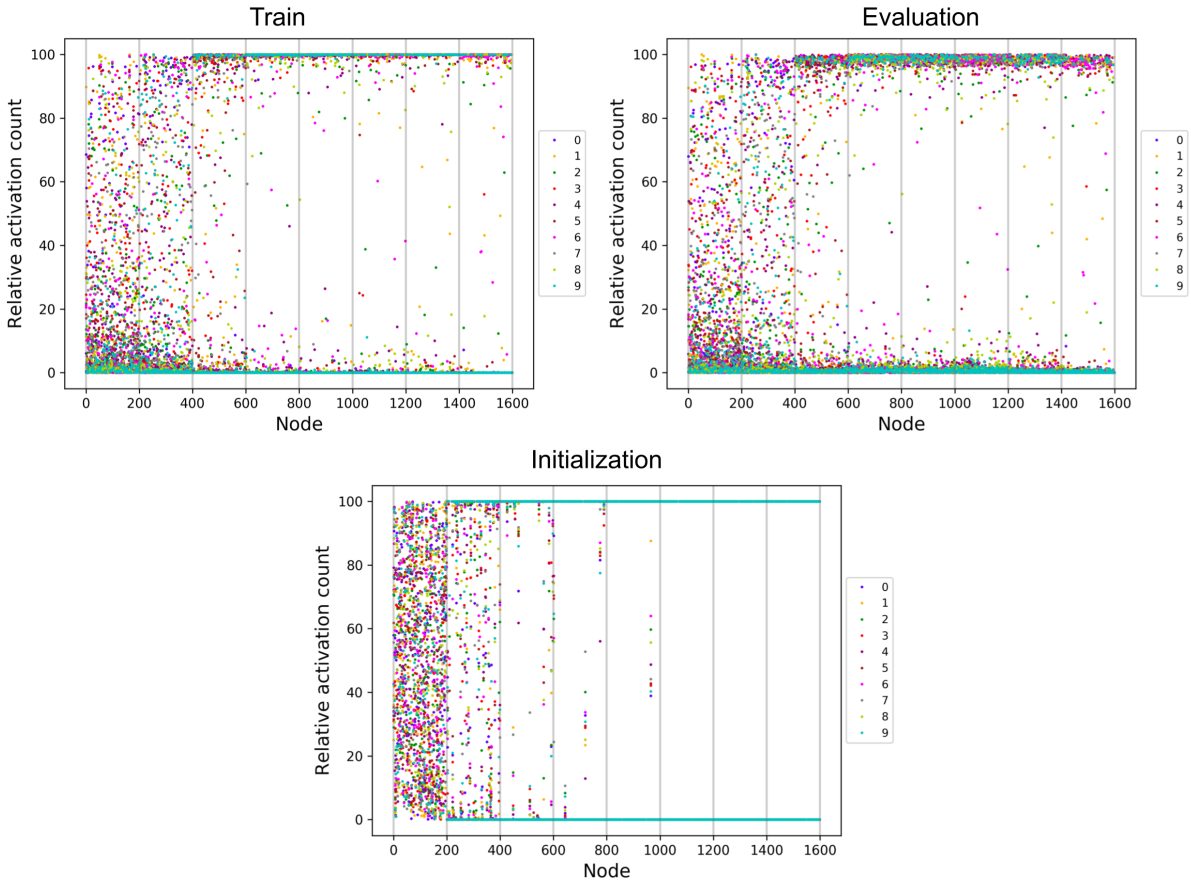


Figure 5.4: Activity per node for a 8x200 sigmoid network trained on MNIST.

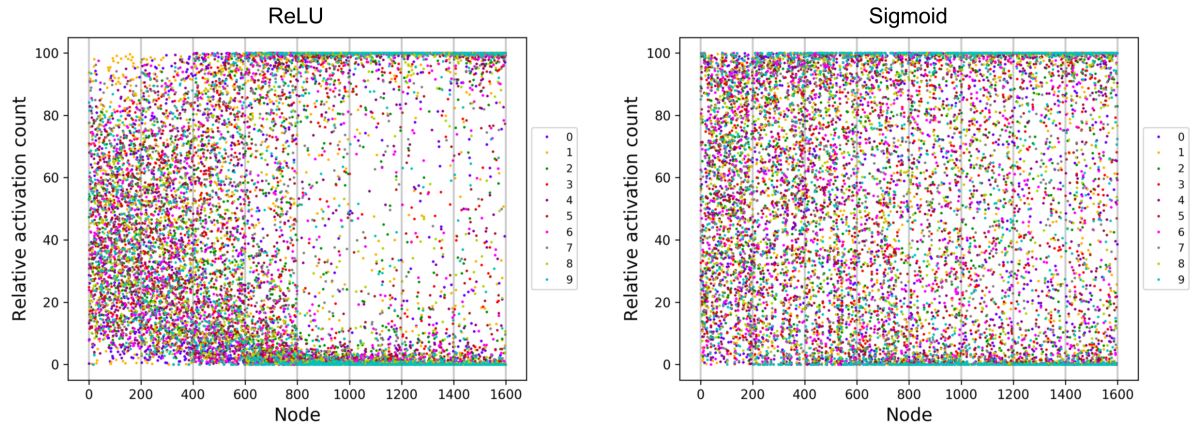


Figure 5.5: Activity per node for a 8x200 network trained with batch normalization. Trained using ReLU (left) and sigmoid (right) activation functions. (MNIST, training set)

## 5.4 Sparsity and dead nodes

Since the beneficial effects of sparsity in deep rectifier models have been proposed as a mechanism for their performance [6], we calculate the sparsity of several of our model architectures trained with and without batch normalization. We use the theta values as described in Section 5.2. Sparsity here refers to the definition given in Section 2.2.4: the characteristic of a representation having several zero or near zero values. The representation we refer to is the output vector of each layer after the activation function has been applied to it.

To better observe how nodes become more specialized in deeper layers and how earlier layers are less class aware, we plot heatmaps that show the class-specific activation counts per node. We also indicate the presence of dead nodes using a red line. Dead nodes do not activate for any sample regardless of class; they are completely inactive and do not contribute to the final classification at the output layer. These nodes arise naturally during back-propagation when using ReLU activation functions. Dead nodes are typically mostly observed in deeper layers and especially in the penultimate layer (last hidden layer). Possible reasons for this observation are discussed in Section 5.5. It is important to note that while dead nodes do not add anything to the classification process, they are able to become alive again during the training process, if the weights in the layers preceding them are updated in such a manner. Dead nodes in the first layer are not able to become alive again.

The activity of nodes for each specific class is observed from the activation patterns in Figure 5.6 to Figure 5.9. The x-axis indexes the nodes in hidden layers while the y-axis represents labeled classes. The intensity of the color represents the amount of samples that activate for the specific class at the specific node. Red lines indicate a dead node that has 0% activity for all classes.

From Figures 5.6 and 5.7, we can see that for ReLU trained networks, nodes in earlier layers have moderate activity for all classes while nodes in deeper layers have high activity for some classes while having low activity for others. This can be seen by the blue-

green hues of the heatmap in earlier layers, while later layers have contrasting dark and light patterns. The 4-layer network has a few dead nodes while the 8-layer network has numerous dead nodes in deep layers. The number of dead nodes in each network is reported in Table 5.1. For the 4-layer and 8-layer networks, it is observed that nodes in deeper layers are overall more inactive for samples of a given class. We report the sparsity of layers in Table 5.2.

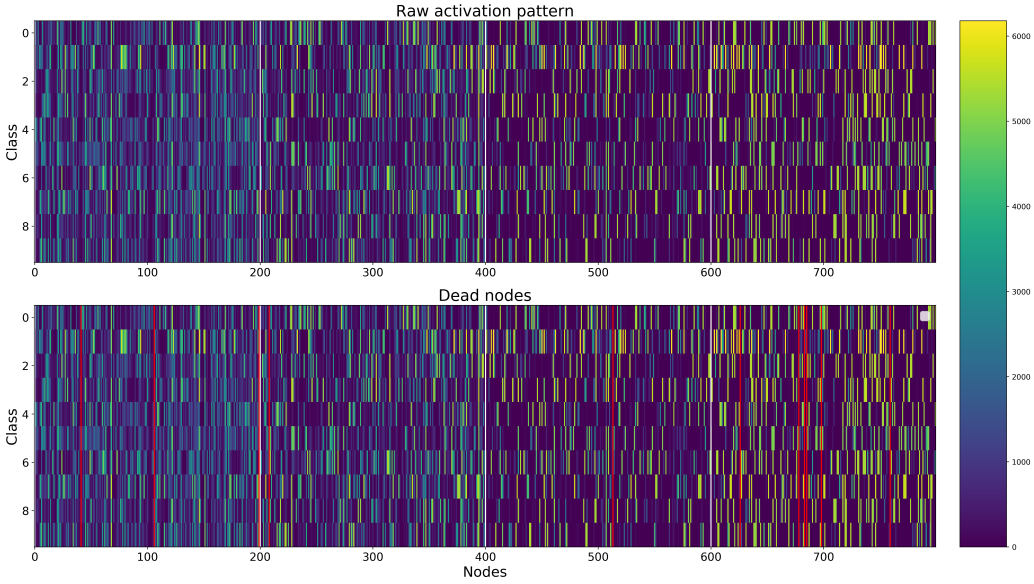


Figure 5.6: Class-specific activity per node for a 4x200 network trained with ReLU activation functions. Brighter colors indicate higher activation counts while red lines indicate dead nodes. (MNIST)

It is observed from Figure 5.8 and Figure 5.9 that sigmoid trained networks have similar activation patterns to ReLU networks, with some distinctive differences. From these heatmaps, we see the same activity as in Section 5.3 where nodes in earlier layers have lower overall activity and nodes in deeper layers have contrasting high and low activity. These figures however also show that the networks have sparse representations throughout hidden layers. The 4-layer sigmoid network has higher overall activity with nodes switching on for more samples than the ReLU network, meaning that the representations are less sparse — see Table 5.2. Interestingly, the 8-layer network has high activity in the 4th to 6th hidden layer with nodes in the 7th to 8th layer having activation patterns closer to that of ReLU networks. The nodes in layers closer to the output layer also have more dead nodes compared to the 4-layer network, but less than the 8-layer ReLU network.

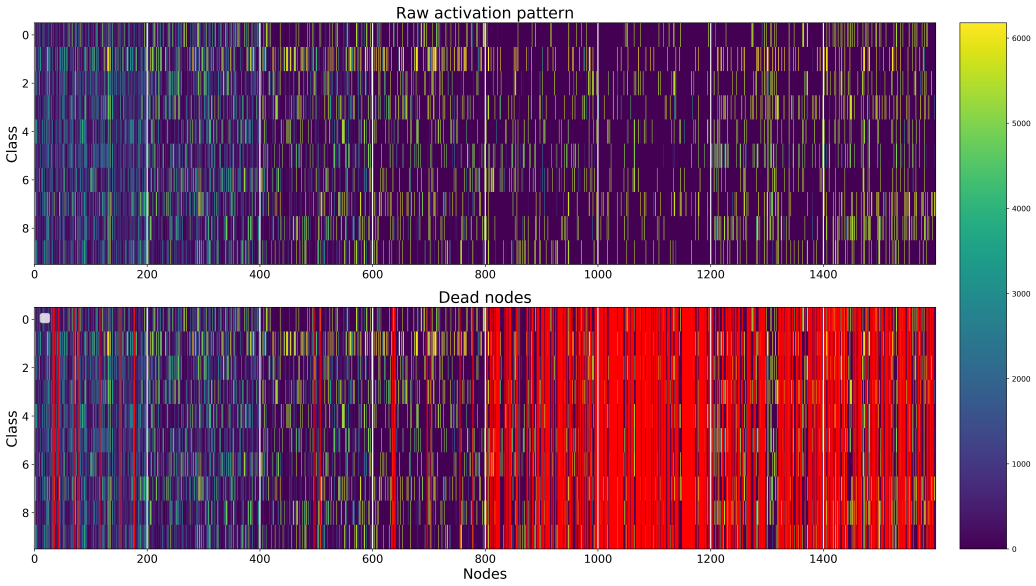


Figure 5.7: Class-specific activity per node for a 8x200 network trained with ReLU activation functions. Brighter colors indicate higher activation counts while red lines indicate dead nodes. (MNIST)

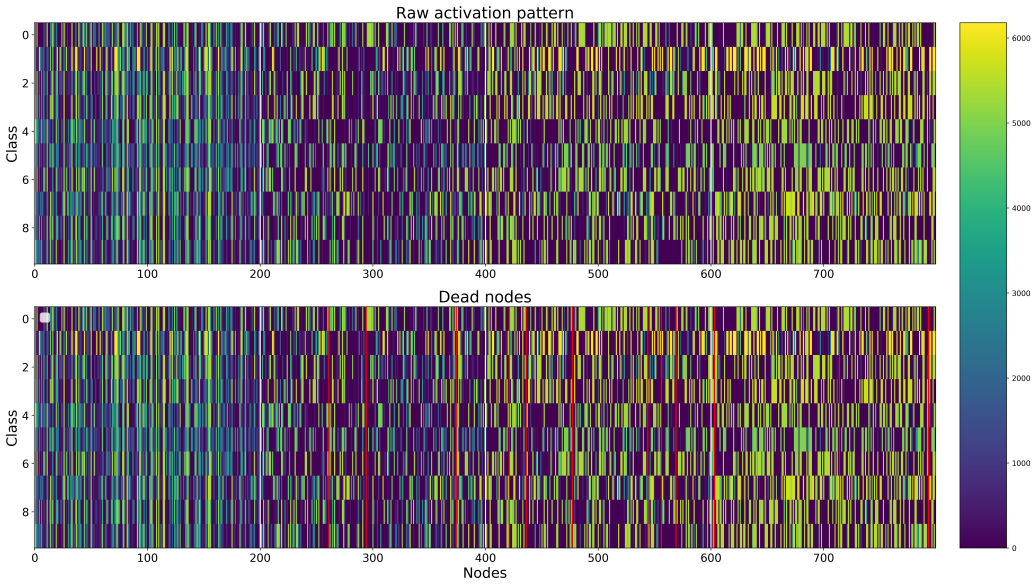


Figure 5.8: Class-specific activity per node for a 4x200 network trained with sigmoidal activation functions. Brighter colors indicate higher activation counts while red lines indicate dead nodes. (MNIST)

We define a metric of sparsity for each hidden layer in a network. This metric is calculated by counting the number of samples that are inactive for each node and then averaging over nodes in a layer to determine the sparsity of the layer. We exclude dead nodes from this calculation. Table 5.2 shows the difference in sparsity for each network shown, as

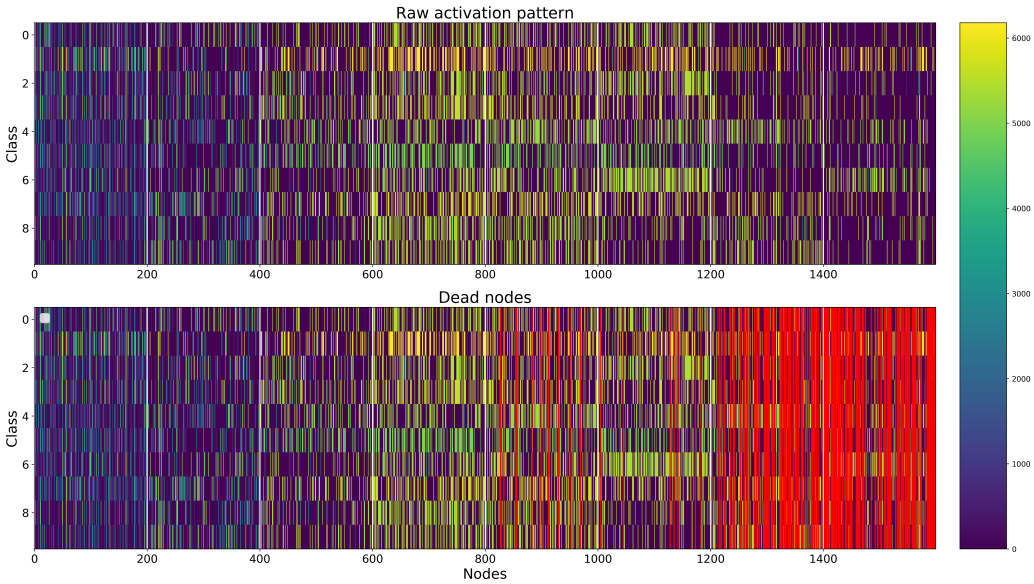


Figure 5.9: Class-specific activity per node for a 8x200 network trained with sigmoidal activation functions. Brighter colors indicate higher activation counts while red lines indicate dead nodes. (MNIST)

Table 5.1: Percentage of dead nodes for networks trained with ReLU and sigmoid activations without and with batch normalization. All layers have a width of 200 nodes.

| | **ReLU** | | **Sigmoid** | |
|---|---|---|---|---|
| **Layers** | **No BN** | **With BN** | **No BN** | **With BN** |
| MNIST (%) | | | | |
| 4 | 1.38 % | 2.38 % | 1.5 % | 2.87 % |
| 8 | 20.12 % | 3.93 % | 10.5 % | 8.81 % |

well as the sparsity of networks trained using batch normalization.

The sparsity of initialized layers in the untrained models are all close to 50%, similar to the results reported by Glorot et al. [6]. It can be observed that the ReLU networks have more sparse interactions than the discrete sigmoid networks. Networks that are trained with batch normalization have fewer sparse interactions per layer, meaning that nodes are (on average) active for more samples of a class. From Chapter 3, we know that the use of batch normalization causes the generalization performance of models to increase, especially for the ReLU networks. A direct comparison of the sparsity of ReLU networks trained with and without batch normalization leads us to believe that sparsity, although useful as shown by Glorot et al. [6], does not entirely describe the impressive generalization abilities of ReLU networks.

Table 5.2: Layer sparsity for networks trained with ReLU and sigmoid activations without and with batch normalization. Sparsity here refers to the average percentage of samples per node that are inactive for a hidden layer (training set). We show the mean sparsity in a layer with standard error over three random training seeds.

| | **ReLU** | | **Sigmoid** | |
|---|---|---|---|---|
| **Layer** | **No BN** | **With BN** | **No BN** | **With BN** |
| 4-layer (%) | | | | |
| 1 | $75.65 \pm 0.88$ | $62.86 \pm 0.53$ | $55.41 \pm 0.78$ | $33.25 \pm 0.94$ |
| 2 | $75.52 \pm 0.65$ | $53.38 \pm 0.18$ | $60.70 \pm 0.52$ | $33.23 \pm 0.71$ |
| 3 | $77.11 \pm 0.67$ | $65.71 \pm 0.44$ | $56.41 \pm 1.22$ | $34.39 \pm 1.14$ |
| 4 | $78.91 \pm 0.66$ | $65.80 \pm 0.18$ | $52.17 \pm 0.45$ | $33.69 \pm 1.18$ |
| 8-layer (%) | | | | |
| 1 | $61.40 \pm 4.51$ | $61.40 \pm 0.82$ | $36.10 \pm 2.94$ | $40.34 \pm 0.34$ |
| 2 | $61.89 \pm 0.78$ | $61.89 \pm 1.10$ | $49.57 \pm 1.88$ | $40.96 \pm 0.64$ |
| 3 | $67.98 \pm 3.26$ | $67.98 \pm 0.26$ | $46.59 \pm 5.15$ | $40.82 \pm 0.61$ |
| 4 | $61.73 \pm 1.63$ | $61.73 \pm 0.21$ | $54.16 \pm 6.38$ | $40.04 \pm 0.15$ |
| 5 | $65.69 \pm 5.42$ | $56.23 \pm 0.61$ | $56.19 \pm 3.63$ | $41.48 \pm 0.49$ |
| 6 | $68.79 \pm 2.60$ | $55.23 \pm 0.08$ | $52.14 \pm 4.11$ | $40.46 \pm 0.79$ |
| 7 | $63.30 \pm 7.08$ | $55.17 \pm 0.83$ | $41.82 \pm 3.35$ | $40.97 \pm 0.31$ |
| 8 | $67.72 \pm 3.55$ | $53.35 \pm 1.25$ | $53.63 \pm 1.82$ | $41.98 \pm 0.29$ |

## 5.5 Discussion

In this section, we discuss possible reasons for the observations made in this chapter.

A reason for dead nodes manifesting in later layers could be attributed to the hypothesis that, when a network is sufficiently trained, the nodes in deeper layers only propagate class information after layers closer to the input has discriminated between class-distinctive information. Excessive nodes in deeper layers are thus not necessary to discriminate between class-distinctive information and those nodes become inactive. This hypothesis is studied in the next chapter.

While sparse interactions force information to be disentangled and linearly separable (as discussed in Section 2.2.4) the presence of dead nodes allows for a direct and overall decrease in representational capacity, allowing the model to control the effective dimensionality of the needed representation [6]. This could act as a model regularizer by lowering the dimensionality of the latent space in which class information can be separated and clustered. While constraining model parameters could effectively act as regularizer, ex-

cessive dead nodes and over-constraining the model could lead to it not appropriately fitting the task. The fact that there are fewer dead nodes when using a regularizer such as batch normalization is also an indication that the presence of dead nodes act as model regularizer. The observation that deeper 8-layer networks have more dead nodes than shallower 4-layer networks, could be attributed to the fact that the over-parameterized networks require more intense regularization so as not to overfit the training data. We also see that the 8-layer networks seem to have slightly higher sparsity in the first layer (but have mostly comparable sparsity than other layers in the network), this could be attributed to dead nodes not being able to to become active again, after initially dying, in the first layer.

From the activation patterns we observe that ReLU networks tend to be less discriminatory in earlier layers and then become more specialized towards single classes in later layers. Due to the activation distributions of sigmoid networks overlapping near the saturation points, nodes in later layers are less specialized towards single classes. The 8-layer sigmoidal network shows that in very deep layers, node activity is more polarized, meaning that activation distributions are saturated more towards 0.0 than 1.0 with significant overlap between classes. In Chapter 6, we investigate the effect that this drop in activity has on the classification ability of deep sigmoidal networks.

## 5.6 Conclusion

In this chapter it was seen that when we regard nodes as discrete switches having either an "on" or "off" state (using a definition appropriate for each activation function), the node activity of networks trained with ReLU and sigmoid activation functions are similar for shallower networks, with nodes of sigmoidal networks being more active overall. The activity of nodes in very deep layers decreases suddenly for sigmoidal networks trained without batch normalization, while the same trends are observed for the activity of shallow and deeper ReLU networks trained without batch normalization. It was seen that, for ReLU networks, nodes in earlier layers are less class-specific while nodes in deeper layers

become specialized and more class-specific.

The sparsity of several architectures were calculated and compared. It was seen that networks trained with ReLU activations are, on average, more sparse than networks trained with sigmoid activation functions. We also showed that when training a network with batch normalization, nodes are more active compared to networks trained without batch normalization, meaning that there are less sparse representations. Since the regularized, "less sparse", networks trained with batch normalization have better generalization performance, we suggest that sparsity is less pertinent than the way in which class-distinctive information is propagated throughout networks, and how the activity of nodes differ for sigmoid and ReLU trained networks.

When regarding nodes as discrete switches, it can be inferred that ReLU networks have the ability to still effectively separate class information in the positive domain—while separating information between the positive and negative domain—whereas sigmoid networks struggle to do this due to overlap of activation distributions at saturation points. The analysis of nodes as discrete switches is useful to compare the activity, sparsity and specialization of nodes in DNNs. In Chapter 6, we show how we can use the continuous analysis from Chapter 4 as well as the discrete analysis from Chapter 5 to determine the role that the continuous and discrete behavior play in the classification ability of nodes and hidden layers.

# Chapter 6

# Nodes as Classifiers

---

*In this chapter we introduce the concept of both nodes and layers as individual classifiers. We use the continuous node behavior in Chapter 4 and the discrete node behavior in Chapter 5 to determine how each layer can use the information available to classify samples when trained with either activation function.*

---

## 6.1 Introduction

Chapter 4 gave an indication of the information available at each node and how class-distinctive information, in the form of continuous activation distributions, is separated and propagated throughout ReLU and sigmoid trained networks. Chapter 5 gave an indication of the activity of nodes in hidden layers, where it was observed that, if each node is regarded as a discrete switch, both ReLU and sigmoid trained networks have similar behavior with some principal differences.

In this chapter, we introduce a model of nodes as individual classifiers, and demonstrate how these individual node-specific classifiers can use the available information of two

collaborating systems, one discrete and one continuous, to determine the probability estimate for a layer and in so doing, effectively create a layer classifier. We do this analysis for networks trained with ReLU and sigmoid activation functions individually, in order to investigate and compare the training and generalization ability of layers trained with these activation functions. We show that there are clear similarities in the way that the ReLU and sigmoid activations are able to effectively use these systems.

## 6.2 DNNs as layers of cooperating classifiers

In a recent paper [1] titled *"DNNs as Layers of Cooperating Classifiers"* [9], we theorized that the continuous and discrete behavior of nodes gives rise to two collaborative systems which are essentially responsible for the classification task.

In the paper [9], we demonstrate intriguing regularities in the activation patterns of hidden nodes in fully-connected feedforward networks. We trace the origin of these patterns and show how networks can be viewed as the combination of two information processing systems: one continuous and one discrete. We describe how these systems arise from the gradient-based optimization process, and demonstrate the classification ability of the two systems , individually and in collaboration. This perspective on classification offers a novel way to think about generalization, in which different subsets of the training data are used to train distinct classifiers. These classifiers are combined to perform the classification task, and their consistency is crucial for accurate classification [9].

In the rest of this section, we first provide more detail on the framework proposed in [9] before applying it to analyze the difference between ReLU and sigmoid networks.

[1] I was a co-author of this paper, and produced the empirical results when applying the framework developed by my co-authors. I also extended the framework to be applicable to sigmoid-activated networks. Some of my dissertation text is therefore repeated in the paper.

### 6.2.1 Theoretical hypothesis

This section summarizes the relevant concepts from [9]; additional detail is provided in the paper.

In [9] it was theorized that two systems arise naturally from gradient descent, one continuous and one discrete, with each having the following characteristics:

1. The discrete system associates an "on/off" value with every sample-node pair in the dataset, depending on whether the node is active or not. This system is fully specified by the $\Theta$ values of Equation 5.1 and 5.2.
2. The continuous system associates a continuous value with each sample-node pair, which is simply the pre-activation value of the sample at the given node. See Figure 4.1 and Figure 4.6.

The discrete system is viewed as a mask during the training process, solely responsible for identifying which samples are taken into account when updating a weight. The continuous system is then responsible for determining the magnitude of the weight update. During training, both systems are utilized to optimize the network. The collaborative effort of the systems has two important effects during the forward and backward pass:

1. During a forward pass (when a sample is fed into a network), the current weights determine whether any sample should be included in a set of similar samples at a specific node.
2. During the backward pass (when the gradient is used to update the weights), the set of similar samples at a node is used to update the relative weighing of the input features. This change in parameterization creates a new feature exclusively attuned to the set of similar samples.

The role of the continuous and discrete system can be better understood by looking at the classification accuracy of the systems given the available information to each system.

For the discrete system this information is the "on" or "off" status of each node and the pre-activation value for the continuous system. We use this postulation then to calculate the training and generalization performance of each system at each node. By taking the joint probability estimate of nodes in a layer, the layer itself is also seen as a classifier.

### 6.2.2 Probabilities based on distributions

To interpret each node as an individual classifier the probability estimate is calculated as $P(z|y_c)$, where $z$ is the continuous or discrete variable (depending on the applicable system) and $y_c$ a specific class. A discrete, continuous and combined estimate is calculated for each variable at each node. At each node we calculate the estimate as follows:

- **discrete:** $P(z|y_c)$ is estimated as the ratio of positive activation values of class $c$ with regard to all samples of class $c$; 1 minus this value otherwise. As in Chapter 5 we regard a value as positive if $z > 0$ for ReLU and $z > 0.5$ for sigmoid.
- **continuous:** the estimate $P(z|y_c)$ provided by a Kernal Density Estimator (KDE) trained using all pre-activation values of class $c$ observed at this node.
- **combined:** applicable only to ReLU trained networks; using the discrete estimate if $z \leq 0$, and the continuous estimate otherwise.

Figure 6.1 shows an example of the kernel density function applied to the pre-activation distributions of two generic class-node pairs similar to those in Figure 4.1 and Figure 4.6. A bandwidth of 0.2 was chosen and fits the original histograms well.

To determine the probability of a class given a specific pre-activation value, we use Bayes' theorem. For all three systems, the estimation is combined with the prior probability $P(y_c)$ of any given class being observed to estimate the classification probability $P(y_c|z)$ given a specific continuous or discrete observation:

$$P(y_c|z) = \frac{P(z|y_c)P(y_c)}{\sum_m P(z|y_m)P(y_m)} \tag{6.1}$$

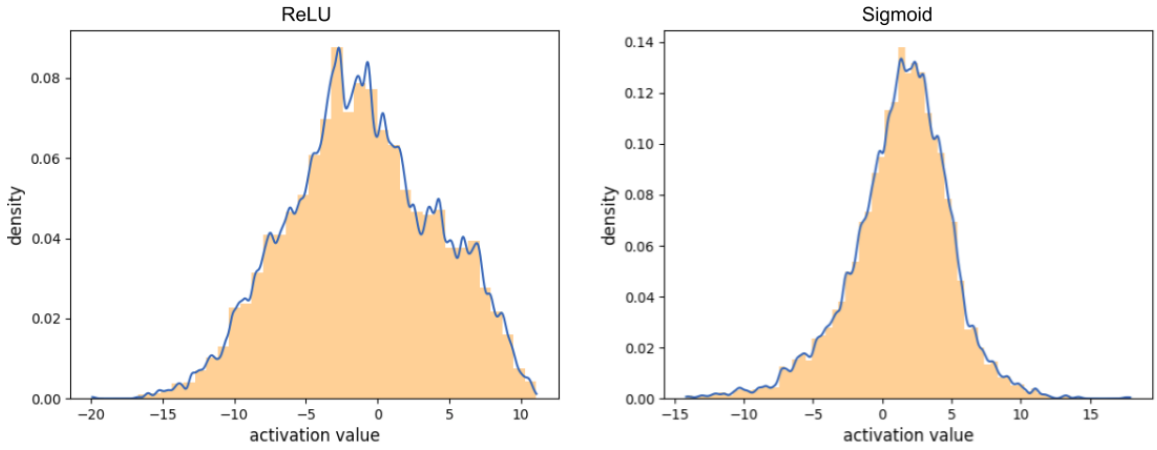


Figure 6.1: Example of applied kernel density estimation to pre-activation distributions of ReLU (left) and sigmoidal (right) trained nodes.

with $m$ ranging over all classes. Due to the class-balanced datasets that are used in this study, the prior probability is near identical for all classes.

As nodes in a single layer are treated as individual classifiers, the probability estimates of nodes for a specific class are simply multiplied over all nodes in a layer to obtain a layer-specific probability estimate. This is achieved in practice by summing the log probabilities of all nodes in a layer. While nodes are not in fact independent, any dependence is ignored when calculating the estimate. The assumption of independence assumes the worst case scenario for classification of the data, however, results are still exceptional given this wrong assumption; and for the purpose of this study, shows that ReLU and sigmoidal trained networks both have the ability to effectively use this continuous and discrete information to perform classification.

While neither the nodes nor the layers in a DNN use these probabilities directly, they provide insight into the information available locally at each point in the network. By evaluating layer-specific classification ability, we can better demonstrate the interaction between the discrete and continuous systems and compare this interaction for ReLU and sigmoid trained networks to determine if one has a distinctive advantage over the other.

## 6.3 Probabilistic comparison of ReLU and sigmoid

Before comparing the training and generalization performance of layer estimators, we discuss the differences in calculating these estimators for ReLU and sigmoidal networks.

For the ReLU trained networks we calculate the probability estimation of the *combined* system in a way that approximates the actual behavior of the network. Activation values below the 0.0 threshold are rectified and information regarding the magnitude of negative activation values are lost while positive activation values are passed and information regarding their magnitude is retained. For the sigmoidal networks there is no combined system that approximates the actual behavior of the network and no explicit discrete system exists due to the continuous nature of the sigmoid function. A discrete classifier can still be constructed using sigmoid activations, by selecting a threshold activation value where the node is deemed to switch from inactive (input values below this threshold) to active (values above). As in Chapter 5, we simply utilize a post-activation value of 0.5 as this threshold, as it is the point at which the sigmoid function's domain is divided in half and where it obtains maximum gradient. Therefore, input signals are expected to diverge from this threshold, over iterations of optimization. In the next section we investigate whether a discrete system also arises from the sigmoid trained networks, and to what extent the sigmoid and ReLU trained networks are similar.

### 6.3.1 Layer accuracy

We calculate the layer accuracy for several of the best performing architectures from Chapter 3 according to the techniques described above. We limit our attention to fully-connected networks trained without explicit regularization or batch normalization. The main focus is to determine how each of the systems plays a role in the training and generalization performance and how they compare for ReLU and sigmoidal networks. We mainly focus on the MNIST and FMNIST datasets.

Figure 6.2 shows the accuracy of each system per layer. The red dotted line indicates

the final train and test accuracy of the actual network. The orange curve shows the accuracy of the continuous system, the blue curve of the discrete and the green curve of the combined system. From Figure 6.2 it is observed that in the first layer, the combined system has higher initial accuracy than the continuous and discrete systems. While it is to be expected that the combined system would outperform the other two (since its probability estimates have access to information pertaining to both the continuous and discrete subsystems) this is not always the case: at later layers, the other two systems are able to perform at levels comparable to the system subsuming them. The difference in accuracy between the systems is diminished at deeper layers and in the last hidden layer, all three system have comparable classification accuracy. It is observed that for all depth configurations, except the 2-layer network, the accuracy of the subsystems are very similar: equal to or very slightly below that of the network. For the 8-layer network, it is observed that the continuous system has a less gradual increase per layer compared to the other systems.

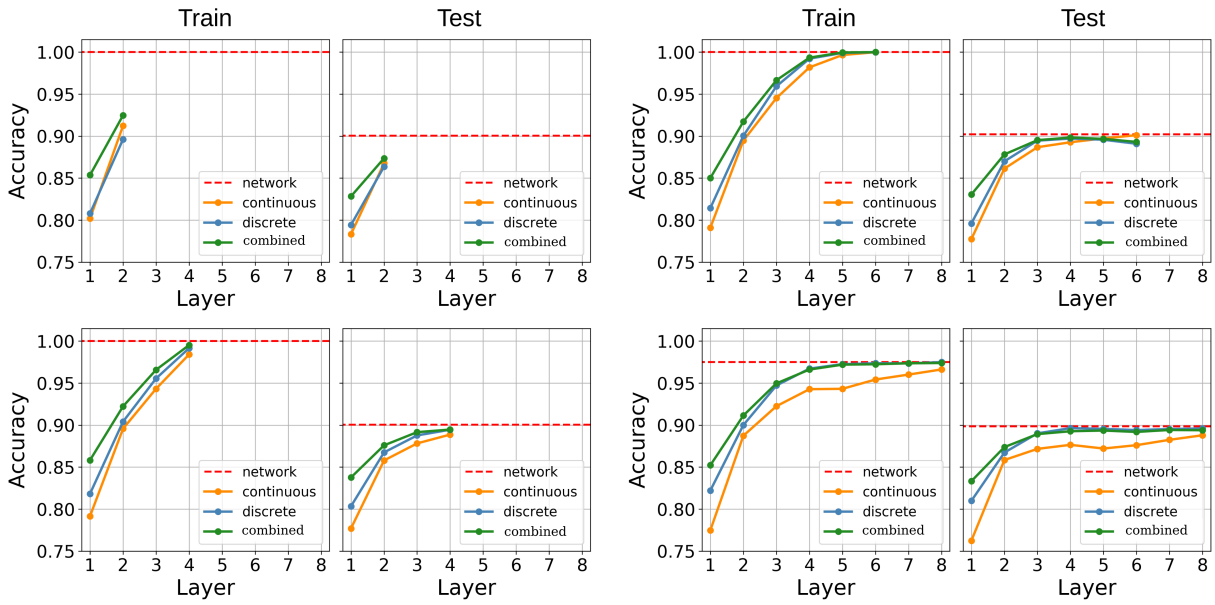


Figure 6.2: Discrete, continuous and combined system train and test accuracies per layer for ReLU networks with varied depth (2-8) and width of 200 nodes. (FMNIST)

These observations made with regard to rectifier networks were surprising at first. We are not aware of related work that has shown that a discrete and continuous subsystem can be used individually to solve the classification task (or parts thereof). It is notable

that these systems can be used individually or in combination to approximate the actual network behavior, reaching the same classification accuracy as the network. What is even more surprising is that similar continuous and discrete behavior can be seen in networks trained with the continuous sigmoidal function.

Figure 6.3 shows the accuracy per layer for sigmoid trained networks on FMNIST. Note that this time no combined system is shown. Overall, similar behavior is observed to that of the ReLU trained networks. The discrete system has higher initial accuracy, but the classification ability of the two systems converge in deeper layers to have the same accuracy as the network.

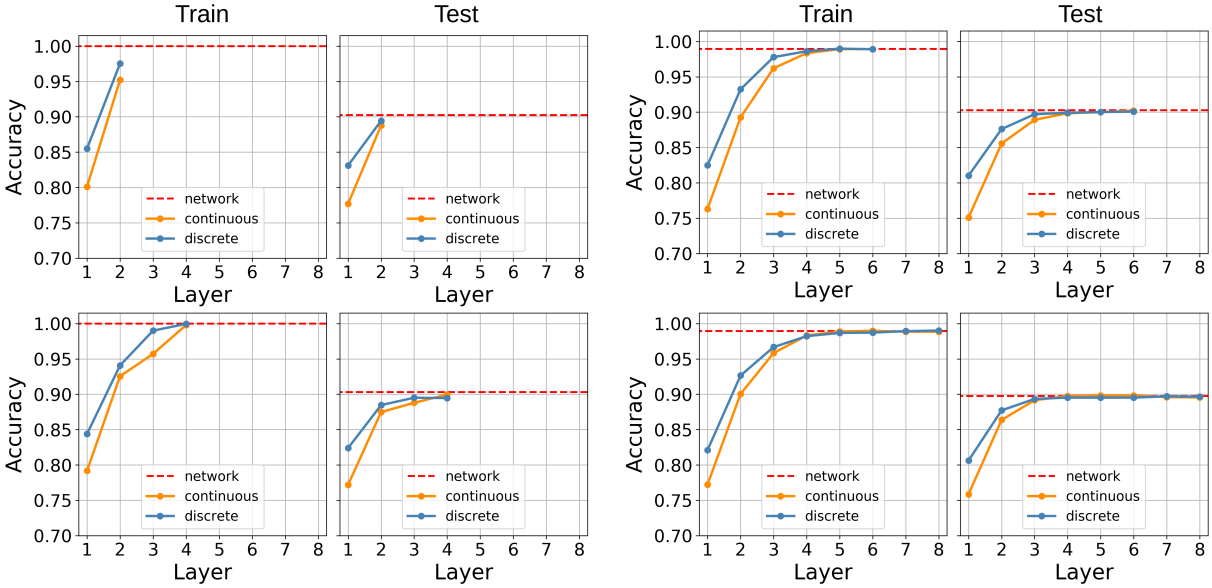


Figure 6.3: Discrete and continuous system train and test accuracies per layer for sigmoidal networks with varied depth (2-8) and width of 200 nodes. (FMNIST)

For ReLU networks trained on the MNIST dataset, similar behavior is observed (as for the FMNIST dataset) with some noteworthy differences: (1) the depth at which the three systems converge is earlier for the easier task; (2) higher accuracies are observed overall compared to the networks trained on the FMNIST dataset; and (3) compared to the ReLU networks trained on FMNIST, it seems that there clearly exists a layer where maximum test accuracy is achieved. After this point a slight decrease in test accuracy per layer for the discrete and combined system specifically is observed. Since the combined system shares information with the discrete system, the parallel drop in accuracy is not

surprising. Note however that the higher accuracy of the continuous system benefits the combined system and prevents it from decreasing to the same accuracy as that of the discrete system. Possible reasons for this drop in accuracy are discussed in Section 6.3.3.

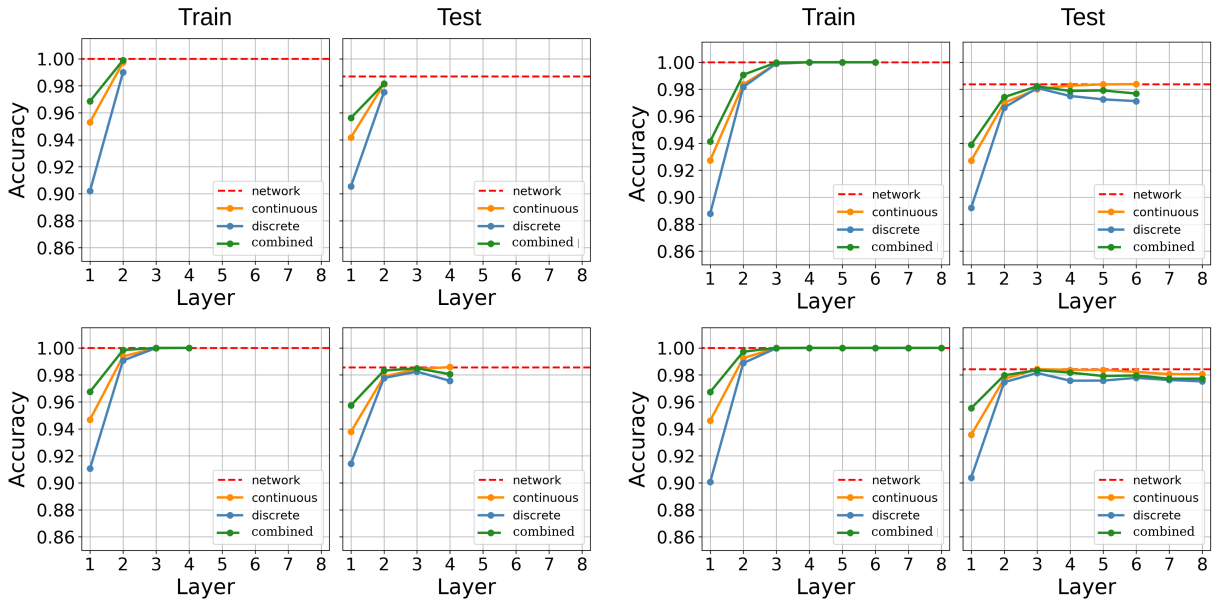


Figure 6.4: Discrete, continuous and combined system train and test accuracies per layer for ReLU networks with varied depth (2-8) and width of 200 nodes. (MNIST)

The behavior of the sigmoidal networks trained on MNIST (Figure 6.5) is comparable with that of the sigmoidal networks trained on FMNIST (Figure 6.3). As with the ReLU networks, the MNIST systems converge at an earlier depth and a slight drop in test accuracy is seen for the discrete system (especially for the 4x200 network).

When the same analysis is repeated for wider networks, similar trends are observed for both ReLU and sigmoid activation functions, except that narrower networks show more stable performance per layer. In Appendix A.2 the accuracy per layer of each system is shown for networks that are 800 nodes wide and trained on FMNIST. From Figure A.5 in Appendix A.2 it was observed that when changing the width of hidden layers, the accuracy of the discrete system initially underperforms compared to the other two systems when networks are not wide enough. When networks are wider, as in the case of the 200 and 800 width networks that are trained in this study, the continuous system initially under-performs but converges to a comparable accuracy in later layers. From Figure A.6 in Appendix A.2, it was seen that for narrower (100 nodes) sigmoidal networks the

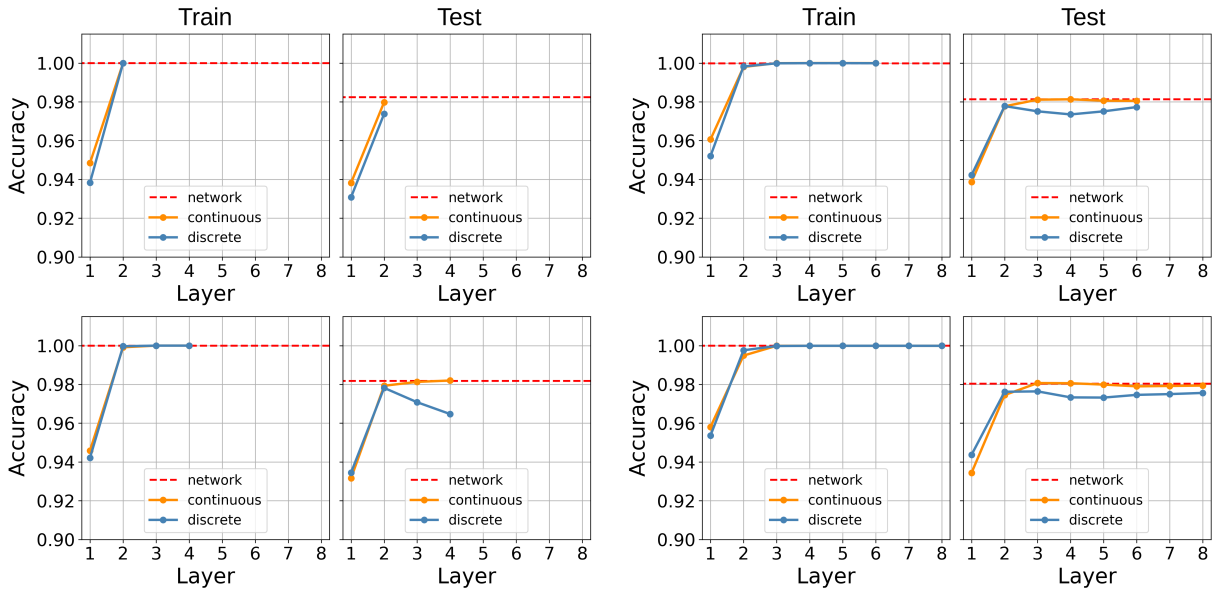


Figure 6.5: Discrete and continuous system train and test accuracies per layer for sigmoidal networks with varied depth (2-8) and width of 200 nodes. (MNIST)

accuracy of the discrete system is comparable (although lower) to that of the continuous system until sufficient depth is reached and the accuracy slightly drops off.

Overall it is observed that for ReLU trained networks, the accuracy of the three systems differ in earlier layers, but achieve equal accuracy in later layers, albeit with some difference as to how close the final result, depending on the architecture and task. The continuous and discrete systems of sigmoidal trained networks tend to have relatively comparable performance for all layers, while the test accuracy of the discrete system slightly underperforms when a certain depth is reached on the MNIST task. This drop in discrete accuracy is slightly more compared to ReLU networks. In deeper layers, the discrete accuracy rises again to just below the network accuracy.

It seems then that the creation of sets of similar samples is something that occurs in both ReLU and sigmoidal networks, the only difference being that similar sample sets have true, discrete participation (or lack thereof) in ReLU networks while having a weighted similarity in sigmoidal networks. Nodes in both ReLU and sigmoidal networks are able to use the continuous information available at the node to accurately distinguish between class-specific features. In Section 6.3.2 we investigate how these observations hold up during the training phase.

### 6.3.2 Learning process

We investigate the behavior of the systems during several epochs of the training phase. Figure 6.6 shows the performance of a ReLU trained network with 6 hidden layers of 100 nodes each, trained on the FMNIST dataset. We choose to show this architecture as it has stable performance and the behavior of this model during training expresses the overall tendencies for all the analyzed ReLU models very well. This network achieved a final training accuracy of 95.56% and a test accuracy of 89.24%.

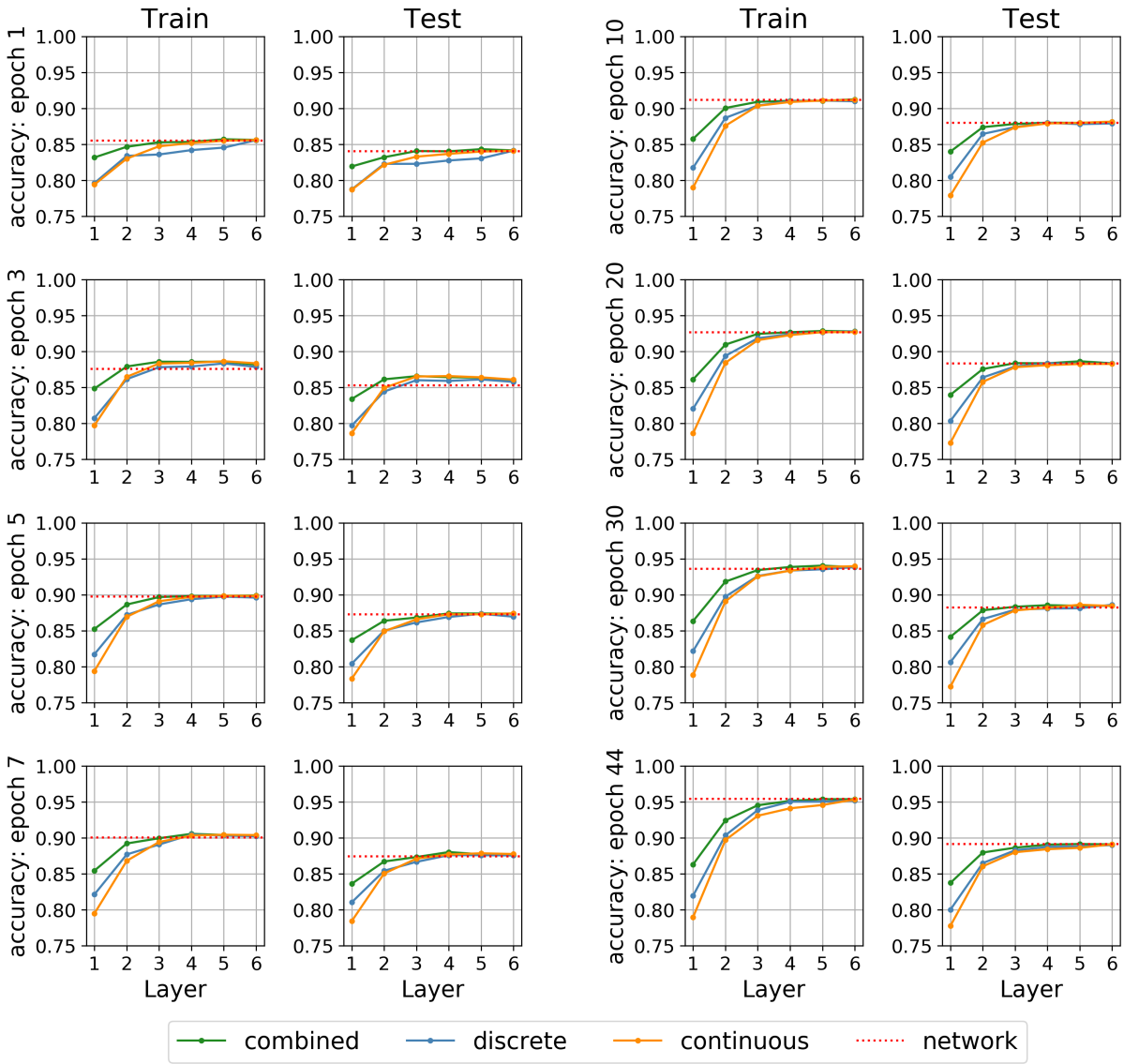


Figure 6.6: ReLU: Train and test accuracies of the discrete, continuous and combined systems as measured on an FMNIST 6x100 DNN. System performance is shown after specific epochs.

In addition to the observations made from Figure 6.2, it can be seen from Figure 6.6 that the accuracies in later layers improve visibly over iterations of learning while the performance of the earlier layers (specifically layer 1) improves less. From Figure 6.6 we observe the very slight increase in accuracy of systems in layer 1, while there is a substantial increase in the accuracy of deeper layers. This reinforces the idea that the function of earlier layers is not to classify samples into the classes involved in the global classification problem, but instead act as general sample differentiators (that is, earlier layers attempt to group and solve subsets of the main task, which may not necessarily be class-specific); later layers use these elements to more efficiently perform the classification task and become more specialized towards single classes. During training the overall accuracy of each system in later layers increases on the train and on the test set until it reaches similar accuracy as the network itself.

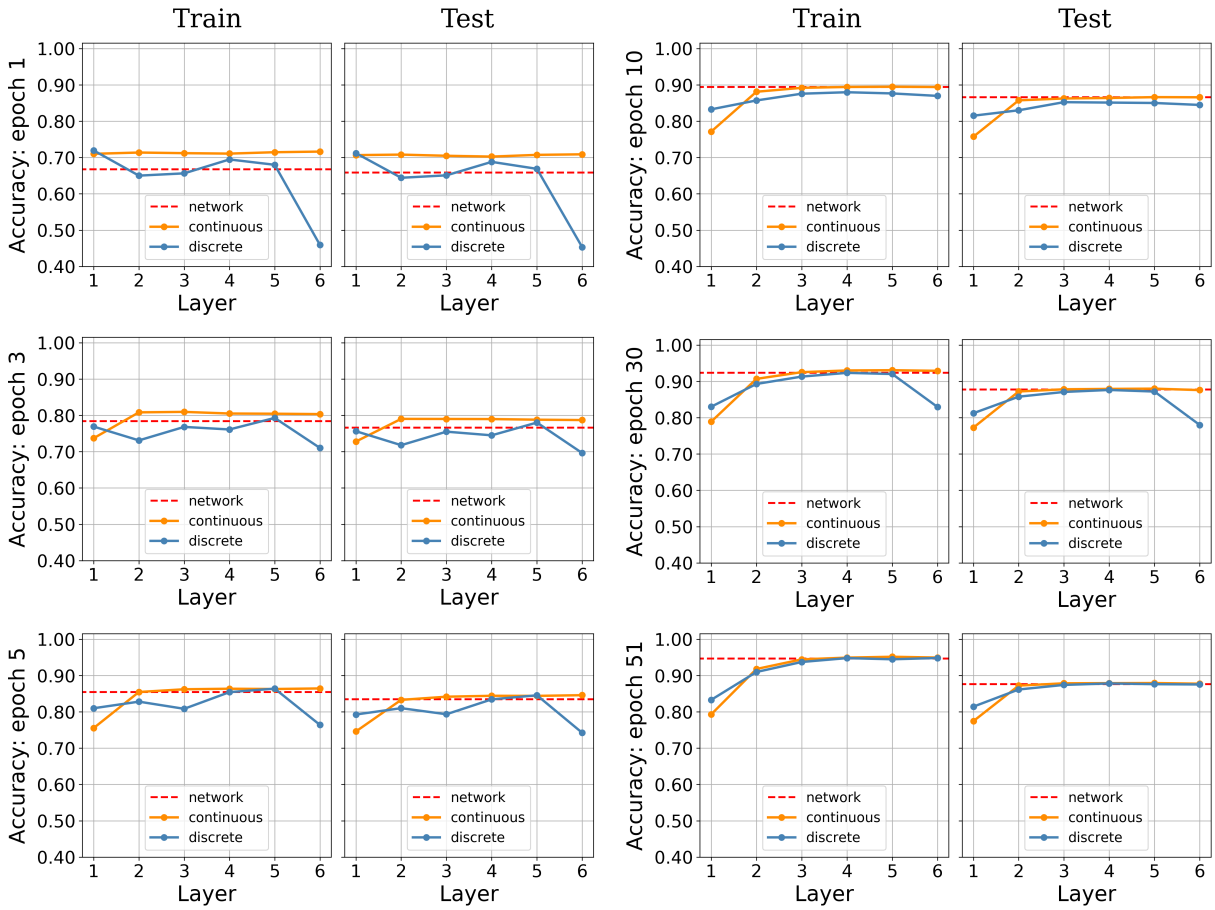


Figure 6.7: Sigmoid: Train and test accuracies of the discrete and continuous systems as measured on an FMNIST 6x100 DNN. System performance is shown after specific epochs.

We now compare the learning process of a 6x100 sigmoidal network (Figure 6.7) with the learning process of the earlier ReLU network (Figure 6.6). This sigmoidal network achieved a final train accuracy of 96.0% and test accuracy of 89.6%. We observe similar trends for the continuous system but a different trend for the discrete systems. The continuous system still shows a gradual increase in accuracy per layer (note that the scales on the two figures are different), whereas the discrete system has an irregular curve (as seen from the oscillating better and worse performance from layers) over the first couple iterations of optimization. From the 10th epoch, the performance of the discrete system is more similar to the discrete performance of the final trained model. Interestingly we see a drop in performance for the discrete system at the last hidden layer, after the 30th epoch; this difference in accuracy then disappears in the final optimized model. Again it is observed that the accuracy of the systems in the first hidden layer does not drastically change over iterations of optimization, although there is more variation than in the 6-layer ReLU network.

From the observations made in this section, it could be inferred that ReLU and sigmoidal networks both have the ability to effectively use the continuous and discrete information available to them at hidden nodes. The discrepancies in test and training accuracy, for the discrete system, are visible in ReLU and sigmoidal networks when trained on the MNIST task. The ReLU network shows more stability for each system during the training phase compared to the sigmoidal network. In section 6.3.3 we discuss what these observations suggest in terms of learning and generalization.

### 6.3.3 Discussion

We now consider the implications of the observations made in Section 6.3.1. Although the analysis of nodes as discrete classifiers is more intuitive in the case of ReLU networks, due to their piecewise linear structure, we clearly see similar behavior in sigmoid trained networks with an appropriate definition of what "on" or "off" means. It is striking to note that in all networks of sufficient size (whether ReLU or sigmoid), there is a discrete layer that achieves the same classification accuracy as the overall network. That is, the

layer encoding of a sample at that layer is fully class-discriminative. This is seen from Figure 6.6 and Figure 6.7, where the curves that show accuracy per layer joins the red curve that shows the accuracy of the network as a whole.

It is further noticeable that system accuracy follows different trends if a network is too small, of the appropriate size for the task, or larger than necessary. For the networks analyzed here, see for example:

- too small: the 2x200 (Figure 6.2, 6.3, 6.4 and 6.5) networks are unable to reach optimal classification accuracy, once fully trained. (Systems do not converge to network accuracy.)
- appropriate: the 4x200 (Figure 6.2, 6.3 and 6.4) networks are fairly compact but still achieve optimal accuracy. (Systems converge with network result.)
- additional parameters: the 8x200 (Figure 6.2, 6.3) and 6x800, 8x800 (Figure A.3, A.4) networks have additional and/or wider layers, but do not achieve significantly higher accuracy. For sigmoid, a drop in discrete accuracy is seen on the MNIST task. For ReLU, convergence is less gradual (on FMNIST) or a slight drop in every system accuracy is visible (on MNIST).

After a specific depth, a clear drop in the test accuracy of the discrete and combined systems is observed for the ReLU trained networks in Figure 6.4 and the discrete accuracy of the sigmoidal networks in Figure 6.5. The training accuracy of all systems is identical to that of the network. We also see this slight drop in test accuracy from the 6-layer and 8-layer networks in Figure A.3. This behavior is still to be fully understood, but a possible reason for this drop in test accuracy could be that the network is slightly over-fitting the training data. Deeper nodes create strong relationships between features of training samples so that the discrete system is not as effective when attempting to group related test samples that are not very similar to those in the training set. This could mean that later layers are more specialized towards similar samples in the training set than that of the test set (since earlier layers are general class differentiators that learn to discriminate between low-level class features).

Overall we see that apart from the slight drop in discrete/combined test accuracy of some networks (discussed above), the subsystems of these networks reach similar performance to that of the actual network well before the last hidden layer. This leads us to believe that the analysis of nodes and layers as individual classifiers could give insight into the necessary network depth for good training and generalization performance.

## 6.4 Conclusion

In this chapter we introduced the theoretical view of nodes as individual and collaborating classifiers. We described how these classifiers use two collaborating systems, one continuous and one discrete, to perform the classification task. It was shown how these systems could use the information available at each node to create probability estimators and then combine these estimators to create a layer probability estimator. Theoretical differences were discussed between the ReLU and sigmoidal networks and how the probability estimates differ for networks trained with these activation functions.

It was observed that:

- For ReLU networks, the accuracy of the three systems differs in earlier layers and then converge to have similar accuracy to each other and that of the actual network.
- The discrete systems of sigmoidal trained networks have higher accuracy in the first layer, but the continuous system has similar performance to the discrete system in deeper layers.
- On the MNIST task, and for very wide layers; ReLU networks have a slight drop in discrete and combined system test accuracy for deep layers. Sigmoid networks have a similar drop in discrete test accuracy on MNIST.
- Overall, increasing the width of networks produced system accuracies that are less comparable, with the continuous system having lower accuracy with a less gradual increase per layer. In very small networks the discrete system underperformed.

- When training a ReLU network, it is clear that the relative accuracy of all three systems in earlier layers stays more-or-less the same, while the accuracy of deeper layers increases as the network train and test accuracy increased. It can be said that, "for ReLU networks, nodes in earlier layers are more agnostic towards classes and attempt to be general class differentiators, while nodes in later layers are responsible for class-specific discrimination".

Regarding the generalization ability and discriminatory prowess of ReLU vs. sigmoid trained networks, several conclusions can be made. It is clear that both the ReLU and sigmoidal trained networks can effectively use the information available at each node to perform the classification task. If networks are not explicitly regularized, the sigmoidal networks have very similar behavior and performance to ReLU networks. In this study, the inferior performance of deep sigmoidal networks is not attributed to the networks not effectively training due to vanishing gradients, but rather to the way in which class information is separated and propagated. The analysis of layers as discrete and continuous classifiers shows that both functions are able to do this separation in a learned manner. Although not inherent to the sigmoid function, there is a clear discrete behavior present in hidden layers. We, however, still hold the theory that the true discrete nature of ReLU networks and all the indirect effects as a result of this discrete behavior is beneficial when training very deep feedforward networks. It can be said that the ReLU function's inherently discrete ability to create sets of similar samples and be completely inactive for other samples is preferred over the weighted similarity of samples in sigmoid networks—especially considering that activation distributions overlap near saturation points for sigmoidal nodes. The hybrid behavior of ReLU networks, seen from the combined system, allows nodes to not only separate class information between the positive and negative domain, but also between values in the positive domain.

# Chapter 7

# Conclusion

---

*In this chapter we summarize key findings and discuss the implications of these findings. We also reflect on future work.*

---

## 7.1 Introduction

As initially discussed in Chapter 1, the aim of this dissertation was to better understand the effect of two very different activation functions on the learning and generalization performance of feedforward DNNs. In this chapter we review the extent to which the initial objectives of the study were met. We summarize key findings, discuss their implications and propose future work.

## 7.2 Key findings and implications

After training and optimizing several models on different datasets and doing several analyses, we observed key findings. Reasons for these findings were discussed and empirical

results were shown. Key findings include:

- The performance of ReLU and sigmoidal networks were compared across various architectures [8]. It was found that: deep ReLU networks tend to generalize better than deep sigmoidal networks, while both functions perform more comparable in shallow and wider networks; ReLU networks trained with batch normalization showed overall best performance.

- The effect of vanishing gradients seem less significant than what is reported in literature [6], [7], [23]. This can be said due to the fact that while there are discrepancies in the generalization performance of ReLU and sigmoidal trained networks, both are able to train to 100% accuracy.

- Nodes in ReLU and sigmoidal networks are able to effectively separate class-distinctive information [8], however the application of the ReLU function is significantly advantageous in this regard. For ReLU networks, nodes in earlier layers have overlapping activation distributions and nodes tend to be more agnostic towards class discrimination; nodes in deeper layers exclusively activate for most samples of a single class, making nodes more specialized towards single classes. Sigmoidal networks separate class information by saturating activation values towards 0.0 and 1.0; this causes overlap of activation distributions and the discriminatory ability of nodes are restricted.

- There is a clear discrete behavior in networks trained with ReLU activations. This discrete behavior was used to measure the sparsity of networks and it was found that while networks trained with batch-normalization are less sparse, they had superior generalization performance. This leads us to believe that sparsity is not the dominating factor regarding the impressive generalization performance of ReLU networks. Using a relevant threshold for sigmoidal networks we see similar discrete behavior at nodes, with some discrepancies in deeper layers.

- It was theorized that there exists two subsystems in a feedforward network, one discrete and one continuous, that is essentially responsible for the classification

task [9]. Using the continuous and discrete behavior of nodes, we are able to create a postulation of nodes as individual classifiers and combine the probability estimates of these node-classifiers to create a layer-classifier. The accuracy of the subsystems per layer gives an indication of the information available at the layer and the role of nodes in the layer. For ReLU networks, it was seen that nodes in earlier layers act as general sample-differentiators, while later layers have accuracy comparable to that of the actual network for the continuous, discrete and combined systems. For sigmoidal networks we surprisingly see similar behavior, except for the discrete accuracy of earlier layers vs. deeper layers while training. This leads us to believe that the ReLU networks create more stable subsets of similar samples.

- It was concluded that the inherent discrete nature of the ReLU function has clear advantages when separating class information. Its ability to simultaneously separate positive and negative values, while also separating values in the positive domain alone, is (according to this study) regarded as its most prominent supremacy.

While we were able to answer some questions and give a perspective regarding the effect of activation functions on the generalization performance of specific network architectures, we do not expect to answer the whole problem of generalization.

Implications of this study include:

- More clarity on design choices (especially regarding activation functions and some perspective on required depth) for specific network architectures and datasets.
- A new perspective that does not fully rely on network sparsity or vanishing gradients to explain the superior performance of deep rectifier networks [8].
- A better understanding of the effect of activation functions on node behavior and classification ability, in terms of learning and generalization.
- The postulation of layers as classifiers gives an indication of necessary depth for a network. This is an aspect of DNNs that is still not well understood.

- The discrete behavior of ReLU networks forms the basis for additional research performed within the larger MuST research group, and ties in with other interesting findings from related studies [9], [35].

## 7.3 Contributions

By completing this study, we were able to deliver an informed statement on the comparative performance of rectified-linear models and sigmoidal models, and provided an analysis of some of the factors that contribute to this performance.

Aspects of this work has since been accepted for publication, specifically:

- A performance comparison of sigmoid and ReLU networks across various MLP architectures [8], developed fully as part of this study.
- A novel perspective on DNNs as layers of collaborating classifiers [9], developed partially as part of this study.

The objectives listed in Chapter 1 were met. The performance and effects of different activation functions were investigated and compared. Key findings were described and their implications were discussed. A perspective on DNN training and generalization, focusing on the effect of activation functions, was given.

## 7.4 Future work

The study complements and supplements research topics regarding generalization in deep neural networks such as sparsity and effective model capacity. Specific topics that we would like to investigate in the future include:

- A concrete metric and understanding surrounding the formation of similar sample sets and their effect on the classification task.

- The effect of explicit regularization techniques such as dropout and norm penalties on node behavior.
- A more precise understanding of the role of nodes as individual classifiers, rather than viewing the layer as a single entity.
- To investigate other activation functions and determine if variations of the ReLU function and the sigmoidal/TanH functions have significantly different characteristics and generalization performance.
- How do the findings and methods in this study translate to different architecture types such as convolutional neural networks.

## 7.5 Conclusion

The impressive generalization performance of DNNs is a topic of great importance and while a plethora of research has been done in this field, a good theoretical understanding is still lacking. Much of the research regarding the effect of activation functions on generalization focuses on vanishing gradients and the advantages of sparsity. This study both questioned these assumptions and provided an additional perspective on the difference in performance of ReLU and sigmoidal trained networks, focusing on node behavior and the inherent discrete and continuous structures of these activation functions. We hope that the study contributes towards a better understanding of DNN generalization.

# Bibliography


[1] V. N. Vapnik, "An overview of statistical learning theory," *Transactions on Neural Networks*, vol. 10, no. 5, pp. 988–999, Sep. 1999, https://doi.org/10.1109/72.788640, ISSN: 1045-9227. DOI: 10.1109/72.788640.

[2] D. Jakubovitz, G. Giryes, and M. Rodrigues, "Generalization error in deep learning," *CoRR*, pp. 2–22, 2018. DOI: 1808.01174.

[3] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, pp. 139, 145, 151–152, 167–170, 192–194, 224–226, 251, 254, http://www.deeplearningbook.org.

[4] K. Jarrett, K. Kavukcuoglu, Y. LeCun, and M. Ranzato, "What is the best multi-stage architecture for object recognition?" *ICCV'09*, pp. 16, 24, 27, 173, 192, 226, 363, 364, 525, 2009, http://yann.lecun.com/exdb/publis/pdf/jarrett-iccv-09.pdf. DOI: 10.1109/ICCV.2009.5459469.

[5] V. Nair and G. Hinton, "Rectified linear units improve Restricted Boltzmann machines," in *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ser. ICML'10, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.165.6419&rep=rep1&type=pdf, Haifa, Israel, 2010, pp. 807–814.

[6] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the fourteenth International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, http://proceedings.

mlr.press/v15/glorot11a/glorot11a.pdf, vol. 15, Fort Lauderdale, FL, USA: PMLR, Nov. 2011, pp. 315–323.

[7] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, vol. 9, Chia Laguna Resort, Sardinia, Italy: PMLR, 13–15 May 2010, pp. 249–256.

[8] A. Pretorius, M. Davel, and E. Barnard, "ReLU and sigmoidal activation functions," in *FAIR 2020*, 2019. [Online]. Available: http://ceur-ws.org/Vol-2540/FAIR2019_paper_52.pdf.

[9] M. H. Davel, M. W. Theunissen, A. M. Pretorius, and E. Barnard, *DNNs as layers of cooperating classifiers*, 2020. arXiv: 2001.06178 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2001.06178.

[10] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals, "Understanding deep learning requires rethinking generalization," *ICLR 2017*, vol. arXiv:1611.03530, 2016.

[11] M. Davel, *Generalization in deep learning*, In-house notes and hypotheses on generalization in DNNs.

[12] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, p. 301, http://www.deeplearningbook.org.

[13] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," *CoRR*, vol. abs/1502.01852, 2015. arXiv: 1502.01852. [Online]. Available: http://arxiv.org/abs/1502.01852.

[14] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, pp. 151–152, http://www.deeplearningbook.org.

[15] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *International Conference on Learning Representations*, Dec. 2014, arXiv preprint arXiv:1412.6980.

[16] C. E. Shannon, "A mathematical theory of communication," *Bell system technical journal*, vol. 27, no. 3, pp. 379–423, 1948.

[17] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, p. 240, http://www.deeplearningbook.org.

[18] D. E. Rumelhart, G. E. Hinton, R. J. Williams, *et al.*, "Learning representations by back-propagating errors," *Cognitive modeling*, vol. 5, no. 3, p. 1, 1988.

[19] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *CoRR*, vol. abs/1502.03167, 2015. arXiv: 1502.03167. [Online]. Available: http://arxiv.org/abs/1502.03167.

[20] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, pp. 317–321, http://www.deeplearningbook.org.

[21] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (elus)," *arXiv preprint arXiv:1511.07289*, 2015.

[22] B. Xu, N. Wang, T. Chen, and M. Li, "Empirical evaluation of rectified activations in convolutional network," *CoRR*, vol. abs/1505.00853, 2015. [Online]. Available: http://arxiv.org/abs/1505.00853.

[23] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *Proc. ICML*, vol. 30, 2013, p. 3.

[24] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, "Activation functions: Comparison of trends in practice and research for deep learning," *arXiv preprint arXiv:1811.03378*, 2018.

[25] Y. LeCun, C. Cortes, and C. Burges, *MNIST database of handwritten digits*, http://yann.lecun.com/exdb/mnist/.

[26] K. Rasul, H. Xiao, and R. Vollgraf, "Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms," *CoRR*, vol. abs/1708.07747, 2017, https://arxiv.org/pdf/1708.07747.pdf.

[27] A. Krizhevsky, V. Nair, and G. Hinton, *CIFAR10 dataset*, https://www.cs.toronto.edu/~kriz/cifar.html.

[28] 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.

[29] P. Y. Simard, D. Steinkraus, and J. C. Platt, "Best practices for convolutional neural networks applied to visual document analysis," in *International Conference on Document Analysis and Recognition (ICDAR)*, vol. 02, Aug. 2003, p. 958.

[30] A. F. Agarap, "Deep learning using Rectified Linear Units (ReLU)," *arXiv preprint*, vol. arXiv:1803.08375, 2018.

[31] 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 (ICLR)*, 2018. [Online]. Available: `https://openreview.net/forum?id=HJC2SzZCW`.

[32] Z. Lin, R. Memisevic, and K. R. Konda, "How far can we go without convolution: Improving fully-connected networks," *CoRR*, vol. abs/1511.02580, 2015. arXiv: `1511.02580`. [Online]. Available: `http://arxiv.org/abs/1511.02580`.

[33] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in PyTorch," in *NIPS Autodiff Workshop*, 2017.

[34] PyTorch, *Torchvision packaged datasets.* `https://pytorch.org/docs/stable/torchvision/datasets.html`.

[35] T. Theunissen, M. Davel, and E. Barnard, "Insights regarding overfitting on noise in deeplearning," in *FAIR 2020*, 2019. [Online]. Available: `http://ceur-ws.org/Vol-2540/FAIR2019_paper_46.pdf`.

# Appendix A

# Supplemental Figures

This appendix contains supplemental figures, referred to in the main text.

## A.1 Appendix: Chapter 4

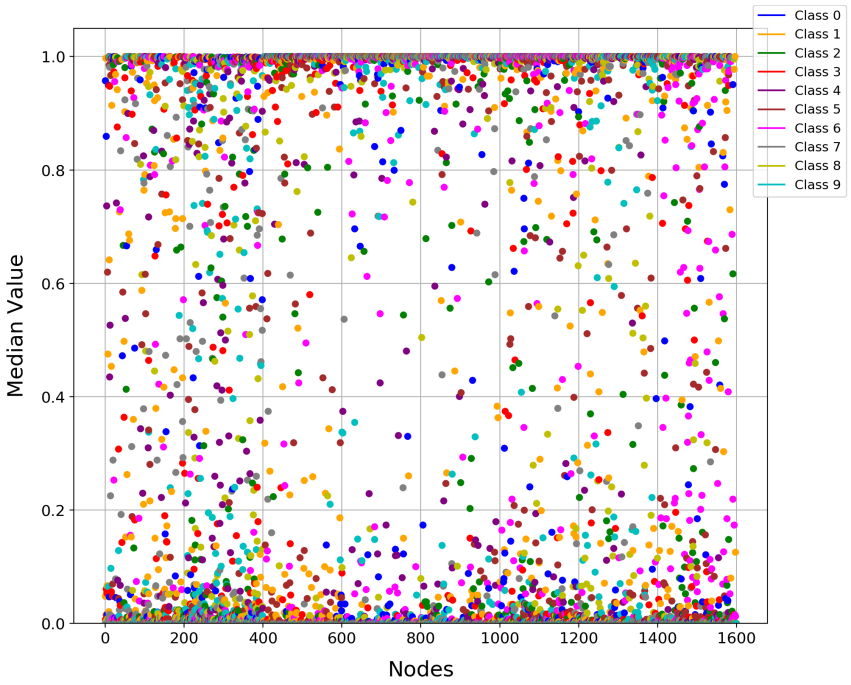


Figure A.1: Medians of activation distributions for each class at every node in a layer. Generated for a 8x200 sigmoidal trained network. (MNIST)

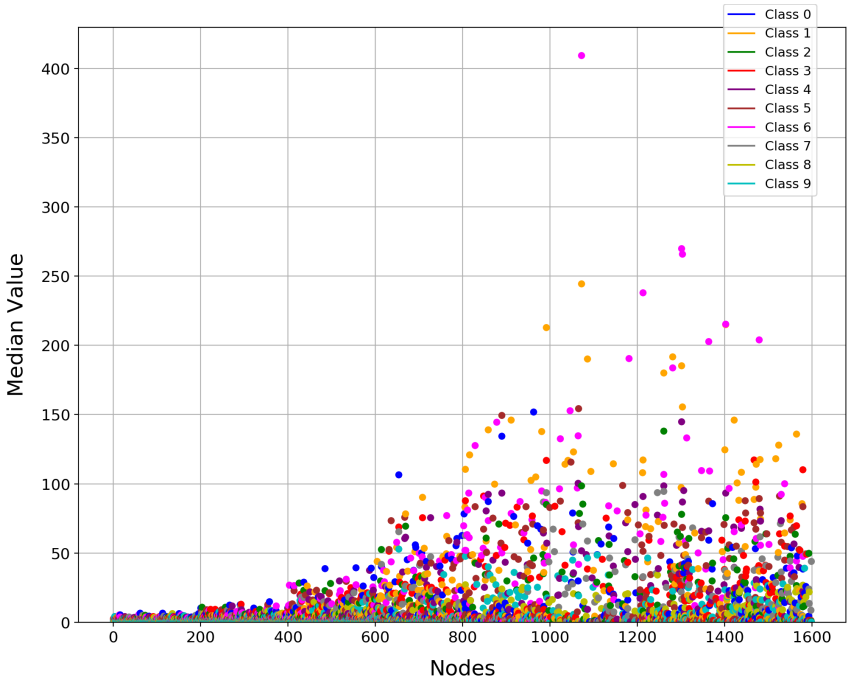


Figure A.2: Medians of activation distributions for each class at every node in a layer. Generated for a 8x200 ReLU trained network. (MNIST)

## A.2 Appendix: Chapter 6

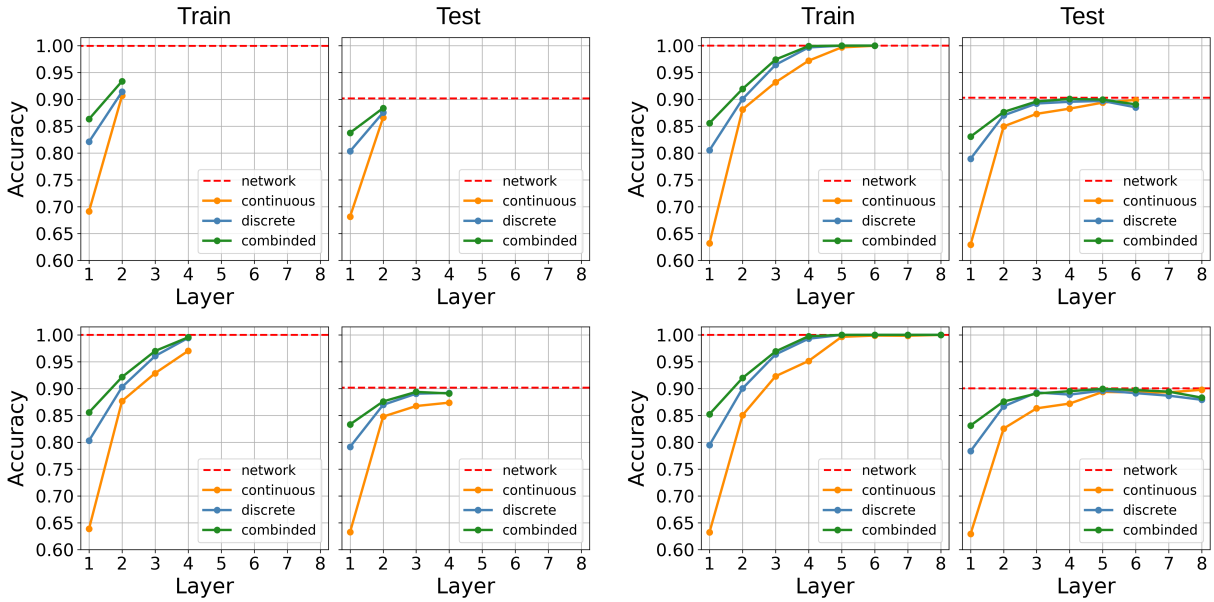


Figure A.3: Discrete, continuous and combined system train and test accuracy's per layer for ReLU networks with varied depth (2-8) and width of 800 nodes. (FMNIST)

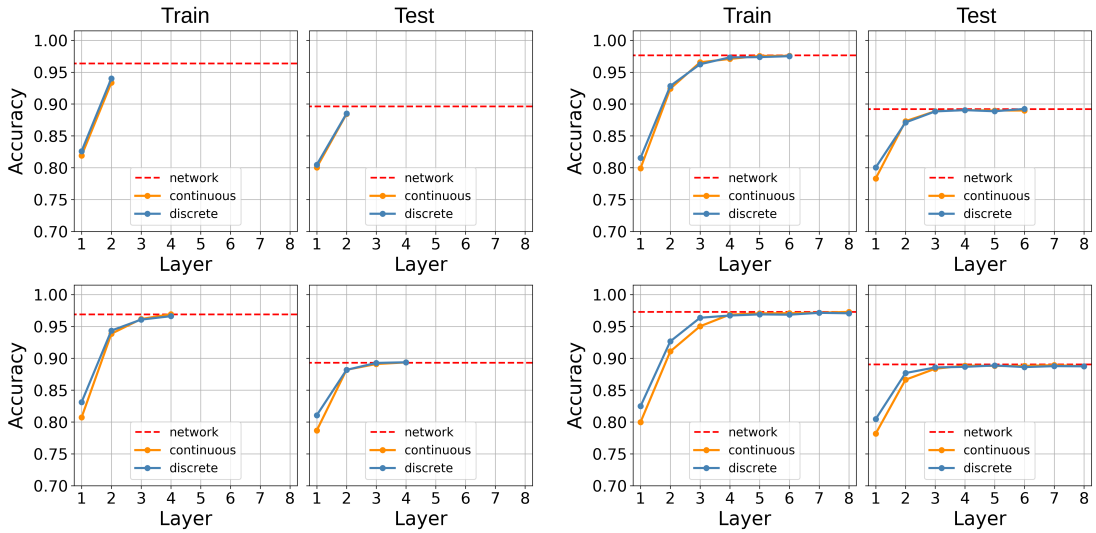


Figure A.4: Discrete and continuous system train and test accuracy's per layer for sigmoidal networks with varied depth (2-8) and width of 800 nodes. (FMNIST)

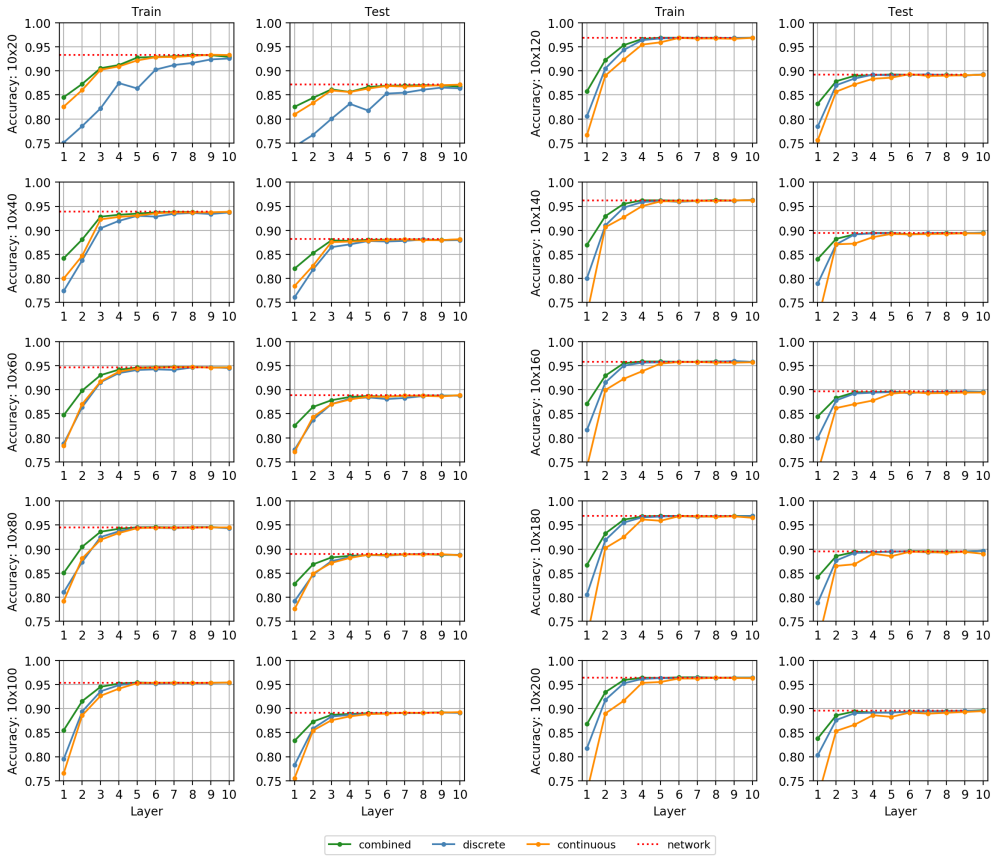


Figure A.5: Discrete, continuous and combined system train and test accuracy's for ReLU networks with varied width (20-200) and a constant depth of 10 hidden layers. (FMNIST)

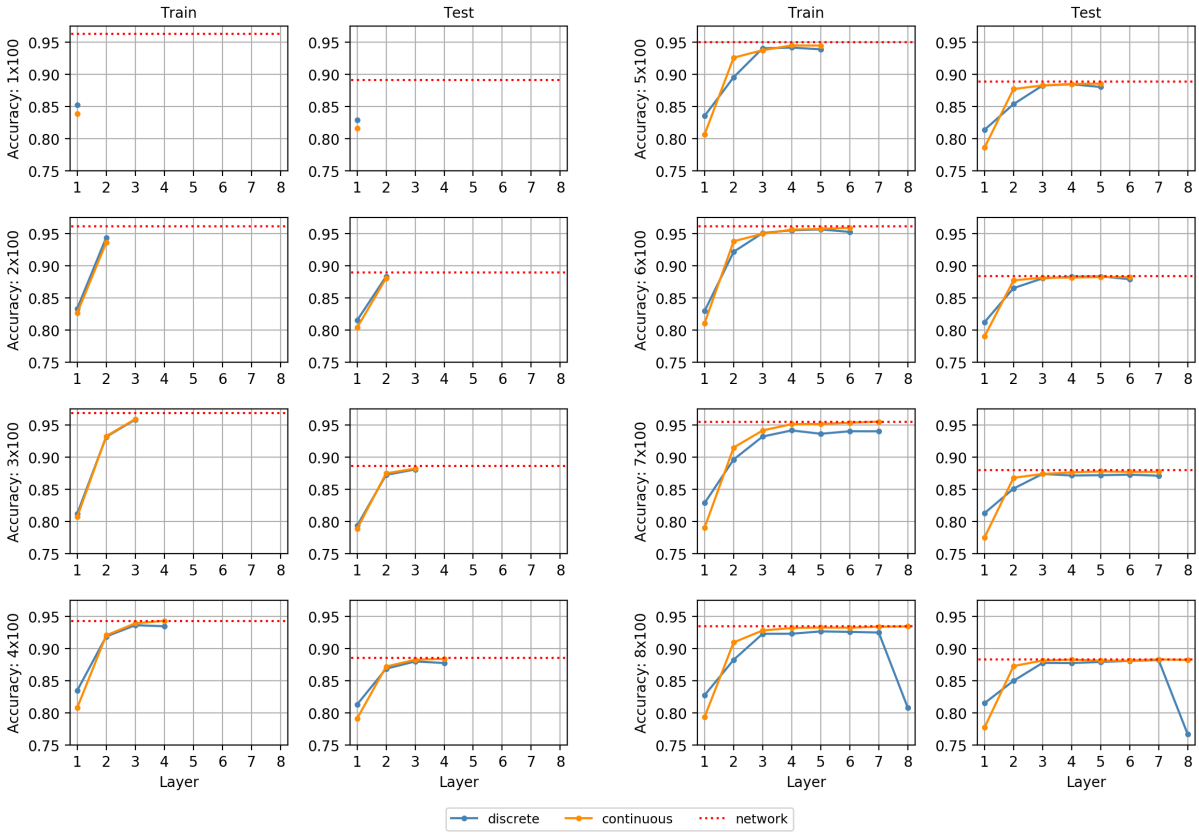


Figure A.6: Discrete and continuous system train and test accuracy's for sigmoidal networks with varied depth (1-8) and a constant width of 100 nodes. (FMNIST)