Distribution Agreement

In presenting this thesis as a partial fulfillment of the requirements for a degree from Emory University, I hereby grant to Emory University and its agents the non-exclusive license to archive, make accessible, and display my thesis in whole or in part in all forms of media, now or hereafter now, including display on the World Wide Web. I understand that I may select some access restrictions as part of the online submission of this thesis. I retain all ownership rights to the copyright of the thesis. I also retain the right to use in future works (such as articles or books) all or part of this thesis.

Evan Scope Crafts April 9, 2018

Multiresolution Methods for Convolutional Neural Networks

by

Evan Scope Crafts

Dr. Lars Ruthotto
Adviser

Department of Mathematics

Dr. Lars Ruthotto

Adviser

Dr. Avani Wildani

Committee Member

Dr. Manuela Manetta

Committee Member

2019

Multiresolution Methods for Convolutional Neural Networks

By

Evan Scope Crafts

Dr. Lars Ruthotto

Adviser

An abstract of
a thesis submitted to the Faculty of Emory College of Arts and Sciences
of Emory University in partial fulfillment
of the requirements of the degree of
Bachelor of Sciences with Honors

Department of Mathematics

2019

Abstract

Multiresolution Methods for Convolutional Neural Networks
By Evan Scope Crafts

Convolutional neural networks (CNNs) are widely used for speech, image, and video recognition due to their state of the art performance. However, little theory exists for designing CNNs and CNNs typically depend explicitly on the resolution of the input data. The CNN is a nested function comprised of a series of non-linear transformations parameterized by initially randomized convolution operators that are optimized to interpolate and extrapolate from data. A new interpretation of the CNN relates the convolution operators acting on image data to a linear combination of differential operators which yields a continuous understanding of CNNs. Multigrid methods are used to efficiently solve partial differential equations (PDEs), which are equations that relate multiple variables and their partial derivatives, using a family of fine and coarse grids. The continuous understanding of CNNs provides a way to implement multigrid methods on the convolution operators of a CNN. This can be used to efficiently handle image data of different resolutions and to train on computationally cheaper lower resolutions. The effectiveness of multigrid methods on a residual neural network architecture (ResNet), a neural network with added stability in the component functions, has been demonstrated previously. This thesis analyzes the effectiveness of multigrid methods on variations of a classical CNN. The experiments here show that on a classical CNN multigrid methods can suffer from overfitting without careful implementation due to the difficult nature of the optimization problem.

Multiresolution Methods for Convolutional Neural Networks

By

Evan Scope Crafts

Dr. Lars Ruthotto

Adviser

A thesis submitted to the Faculty of Emory College of Arts and Sciences
of Emory University in partial fulfillment
of the requirements of the degree of
Bachelor of Sciences with Honors

Department of Mathematics

2019

Acknowledgements

First and foremost, I would like to thank my advisor, Dr. Lars Ruthotto, for introducing me to neural networks, showing me what makes them so interesting, and being patient with me throughout the research process. I would also like to thank the other two committee members, Dr. Avani Wildani and Dr. Manuela Manetta, as well as many of the professors I have had at Emory, for engaging me and helping me on my path towards the computational sciences. This project would not be possible without generous support from Emory's SIRE, SURE, and honors thesis programs, as well as the National Science Foundation award DMS 1751636. Last but not least, I would like to thank my family for supporting my studies at Emory and encouraging me to discover what I am passionate about.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Over the last ten to fifteen years, a machine learning technique that learns relationships from data called deep neural networks (DNNs) has risen to prominence in the field of artificial intelligence. The origins of the neural network date back to the 1950's and 1960's when computer scientists first attempted to create computational models based off the human brain. However, for a long period of time lack of access to data and limited computational power meant that neural networks were largely ignored in the computer science community. This began to change in 1990 with the publication of "Handwritten Digit Recognition with a Back-Propagation Network", which introduced the concept of the convolutional neural network (CNN) for the first time [10]. A CNN is a special type of DNN that utilizes convolution operators to reduce computational cost and recognize spacial structure within input data. As computational power and access to large data sets continued to increase into the 2000s, the CNN rose to prominence due to its state of the art performance on canonical image recognition tasks and soon became a widely used method for related tasks such as text and speech recognition. However, learning these relationships is a very difficult optimization problem and the success of the CNN has belied the difficulty of constructing effective networks that both optimize and generalize well.

When a CNN is created it is generally designed for input data of a specific resolution. While this methodology is suitable for testing on canonical data sets, most real-world data can have varying resolutions. A possible solution to this problem comes from a connection made in recent years between the structure of a common type of deep neural networks, residual neural networks

(ResNets), and the structure of a discretized differential equation, which has created a novel way to analyze the stability of neural networks [6]. In particular, this methodology establishes a connection between the CNN and the partial differential equation (PDE), an equation that relates multiple variables and their partial derivatives [15].

Multigrid methods are a family of techniques for numerically analyzing PDEs on a family of fine and coarse grids, which connect well with the varying resolutions of real-world input data [5]. While the methods described here have applicability to a wide variety of input data with spatial structure, this thesis focuses on image data. Image data is a grid discretization of an image, which is a continuous function $f : \mathbb{R}^2 \to \mathbb{R}$. In [7] multigrid methods are implemented on a ResNet CNN trained on image data and its effectiveness is shown in that context. This thesis analyzes the performance of multigrid methods on variations of a basic (non Res-Net) CNN architecture and examines many of the factors that can effect multigrid performance.

1.1 Contributions and Outline

In this paper I show that various network architectures and optimization choices can effect the performance of multigrid methods. In chapter two I provide a mathematical formulation of the neural network framework and the PDE interpretation of the neural network. In chapter three I discuss the methodology used. In chapter four I discuss the numerical results from these experiments. These include:

- Application of multigrid methods to a basic CNN framework using the MNIST data set (section 4.2)
- Analysis of the effect of pooling on multigrid performance (section 4.3)
- How optimization methods effect performance (section 4.4)

Finally, in chapter five I discuss the importance of these results to the machine learning community.

Chapter 2

Background

To understand the connection between neural networks and differential equations that this thesis is based on it is imperative to have a mathematical description of the deep neural network and the connected optimization problem. In this chapter I first present the generalized neural network optimization problem and expand on this to describe the specific structure of the CNN and ResNet. I then show how this problem statement leads to a connection between neural networks and differential equations and how this connection can be used to address stability issues in neural networks. Finally, I provide the motivation for the introduction of multigrid methods and give a theoretical background on their relevance to the goal of functional multiresolution neural networks.

2.1 Neural Networks: An Introduction

The abstract goal of a neural network is to learn a function from an input space $X \subset \mathbb{R}^k$ to an output space $Y \subset \mathbb{R}^l$. Given an $x \in X$, we would like to learn a function f such that there is a high probability that the output $f(x)$ is equal to the desired output y. To prevent route overfitting to the data, we restrict f to be part of a class of functions H. In short, we seek a function f that is a solution to

$$\min_{f \in H} J(f), \quad \text{where} \quad J(f) = \min_{f \in H} E(\mathbb{1}(f(x) \neq y)), \tag{2.1}$$

where E is the expected value and $\mathbb{1}$ is the indicator function [4]. In practice, however, it is impossible to know the desired output y for every $x \in X$.

Instead of minimizing over the entire domain space, a technique called supervised learning is often used, where the related problem of finding the minimizing function over a set of examples $\{(x_i, y_i)\}_{i=1}^n$ called the training data is solved. This related problem is formulated as minimizing the empirical risk

$$\min_{f \in H} J_n(f), \quad \text{where} \quad J_n(f) = \min_{f \in H} \sum_{i=1}^{n} \mathbb{1}(f(x_i) \neq y_i). \tag{2.2}$$

The fact that we are solving a related empirical optimization problem and not the actual problem creates difficulties when choosing the family of functions H. We would like H to be a family that is easy to optimize over, i.e. it is easy to find the function f that solves the problem given in (2.2). In this context easy means that we can choose an optimization method that will reach a local minimum of the function with high probability, and that the computational complexity of doing this is low. We would also like to choose H so that we arrive at a low value of $J_n(f)$, and so that the value of $J(f) - J_n(f)$ is low, although in practice this last value cannot be known exactly. However, the value of $J(f)$ can be approximated by using a different set of examples $\{(x_i, y_i)\}_{i=1}^m$ called the validation data. By ensuring that the value of $J_n(f)$ over the validation data is low as well we can know with high probability that the true value $J(f)$ is also low.

The defining feature of the neural network is the choice of H. Instead of choosing a large class of functions which might be extremely difficult or impossible to optimize over, H is restricted to a series of nested functions with further restrictions on the nested functions themselves [4]. The nested functions in a neural network are called layers and take the form of affine with a point-wise linearity subsequently applied. The linear transformations are parameterized by initially randomized weights, θ, which are optimized over. The composition of the network layers is the function from the input space to another euclidean space known as a feature space, $g : X \to \mathbb{R}^{n_f}$.

In the applications considered here the output of the function, $g(x, \theta)$, must undergo further transformations to be mapped into the Y space and become a classification prediction. To accomplish this the output undergoes a final affine transformation , $\mathbb{R}^{n_f} \to \mathbb{R}^l$, and then is subject to a final activation function $\sigma_s : \mathbb{R}^l \to Y$. A common choice for σ_s is the softmax

function. These structural decisions reformulate the objective function from (2.2) to

$$\min_{\theta,w} J(\theta, w), \quad \text{where} \quad J(\theta, w) = \frac{1}{n}\sum_{i=1}^{n} l(\sigma_s(g(x_i, \theta), w), y_i), \tag{2.3}$$

and $l : \mathbb{R}^l \times \mathbb{R}^l \to \mathbb{R}$ measures the error in classification and w is the classification weights, the weights of the final affine transformation. Often a regularizing term $R(\theta, w)$ is added to prevent overfitting to the empirical data and ensure $J_n(f) - J(f)$ remains small. The structure of the neural network is determined by the choice of nested transformations that comprise f, as well as the choice of activation and objective function. In (2.2) the indicator function was used for l. However, the indicator function is not differentiable, which makes the optimization problem much harder. To remedy this, l is chosen from the class of differentiable distance metrics. A common choice is the cross-entropy function.

In practice, almost all of the optimization methods used to train neural networks rely on a technique called backpropagation which was conceived in [14]. Backpropagation uses the chain rule to differentiate the objective function with respect to theta and w. The prototypical and most commonly used optimization method is the stochastic gradient method (SGD). SGD is an iterative method that utilizes random samples from the training data to update both theta and w in the negative of the direction of the negative gradient from those samples,

$$\theta_{k+1} = \theta_k - \alpha_k \nabla l_{i_k}(\theta_k) \tag{2.4}$$

and

$$w_{k+1} = w_k - \alpha_k \nabla l_{i_k}(w_k), \tag{2.5}$$

where α_k is a scalar sequence known as the learning rate. In practice, it has been shown that for many classification problems SGD will converge to a local minima, although the reasoning behind its performance is not well understood.

2.2 Network Structure

The choice and composite structure of the layers that comprise g fundamentally impact the stability and accuracy of the the model. Two major

developments in this area, the CNN and the ResNet, are in large part responsible for the current dominance of neural networks at tasks such as image and speech recognition. Each nested function in the neural network is called a layer. Broadly, we can define the neural network nonlinear operator (the layer) using notation from [6] and [8] as being of the form

$$F(\theta, x) = \mathbf{K}_2(\theta^2)\sigma(\mathbf{K}_1(\theta^1)x + \mathbf{B}(\theta^3)) \tag{2.6}$$

where x is the input to the layer. The $\sigma : \mathbb{R}^{n_f} \to \mathbb{R}^{n_f}$ here is a point-wise non-linearity referred to as the activation function. Common choices for the function include the tanh function and the rectified linear unit (ReLU), which is defined as $\sigma(x) = \max(x, 0)$. The $\mathbf{B}(\theta^3)$ term is a point-wise bias. Finally, the $\mathbf{K}_1(\theta^1)$ and the $\mathbf{K}_2(\theta^2)$ terms are the all-important linear transformations.

Before the advent of the CNN many networks were fully connected. This meant that any element in the input of a layer can communicate with any element in the output. In practice this often took the form of operators where $\mathbf{K}_2(\theta^2)$ is the identity mapping and $\mathbf{K}_1(\theta^1)$ is a dense linear transformation where every element of the output is a linear combination of all elements in the input. However, the obvious advantage of fast communication in a fully connected network was coupled with the disadvantage of ignoring any spatial structure found in the input data. Many common classification problems, such as image, speech, and text recognition, have inherent spacial structure. Further, in networks with many layers this can result in millions and millions of weights, making the optimization process more difficult and costly. The authors of [10] were the first to propose a solution to this with their introduction of the CNN. In a CNN, each $\mathbf{K}_1(\theta^1)$ is a convolution operator parameterized by a small matrix called the kernel of the transformation.

To go from individual layers to a full neural network, a description of how the layers connect to each other is needed. Before the advent of the residual neural network, or ResNet, most networks were comprised of a series of layers in the form of

$$x_{i,j+1} = F(\theta_j, x_{i,j}) \quad \text{for} \quad j = 0, 1, 2, \ldots, N-1, \tag{2.7}$$

where $x_{i,j}$ denotes the ith example propagated to layer j and the initial layer $x_{i,0}$ is the input data. Information propagates forward through the network until it reaches the last layer and undergoes the final linear transformation and subsequent classification. As the number of layers in neural networks

began to increase over time, however, problems in the formulation given in (2.7) began to arise. Adding more layers to a neural network did not always result in increased classification accuracy; in fact, in practice adding additional layers to the network could actually decrease the the effectiveness of the network. The writers of [8] and [9] were the first to realize that a simple reformulation of (2.7) would result in increased stability; they defined the forward propagation as

$$x_{i,j+1} = x_j + F(\theta_j, x_{i,j}) \quad \text{for} \quad j = 0, 1, 2, \ldots, N-1. \tag{2.8}$$

The theoretical motivation for this was straightforward. The $x_{i,j}$ term tries to ensure that new additional layers will only increase the performance of the the network because at worst the $F(\theta_j, x_{i,j})$ term can just be driven to zero. For simplicity sake it is assumed that all of the network layers lie in the same feature space, $\mathbb{R}^{n_f}$. The input data $x \in X$ is mapped into the feature space by a linear transformation, $x_0 = \mathbf{L}x$, where $\mathbf{L}$ can be either fixed or learned.

By combining this formulation with convolutional layers, the creators of the ResNet were able to build Deep Residual CNNs that achieved state of the art performance on one of the canonical data sets in image recognition, ImageNet. Since then Deep Residual CNNs have become the workhouse behind the best performances on speech, image, and text recognition, among other challenges. However, this dominance belies how little is actually known about the why and how of neural network construction.

2.3 A New Interpretation: The Kernel as a Differential Operator

Despite the success of the ResNet and the CNN, major questions still remain about the structure and stability of the neural network. Given a classification problem, it is often unclear which neural network structure and coupled optimization method will lead to stable forward propagation and high classification accuracy. Numerous results have helped to illustrate different aspects of this problem, including the development of adversarial examples and examinations of the generally non-convex loss landscape [16, 3, 12]. Even given a suitable structure, often many hyper-parameters must still be manually tuned. In addition, neural networks lack adaptability - they are designed

for input data of a specific resolution. While this might not be an issue on canonical data sets, real world data is of varying resolutions and would require interpolation to be used on one network. This interpolation can remove valuable information from the input data, such as smoothness properties. These factors combine to show the utility and necessity of a new perspective on the dynamics of a neural network.

The authors of [7] were the first, to the author's knowledge, to propose a connection between the forward propagation in a ResNet and the Euler discretization of an ordinary differential equation (ODE). The forward propagation equation for the ResNet ((2.8)) can be easily generalized to

$$x_{i,j+1} = x_{i,j} + hF(\theta_j, x_{i,j}) \text{ for } j = 0, 1, 2, \ldots, N-1 \tag{2.9}$$

where h is a scalar with $h = 1$ in the original formulation of the ResNet. This equation is equivalent to

$$\frac{x_{i,j+1} - x_{i,j}}{h} = F(\theta_j, x_{i,j}) \text{ for } j = 0, 1, 2, \ldots, N-1.$$

The h can be interpreted here as a step size in a finite difference approximation of the change x with respect to an artificial time variable t. If x is changing sufficiently slowly then the ResNet step size of $h = 1$ will work well, but if x is changing too rapidly this will create instability. The weights, θ_j, at each layer can also be understood as a discretization of a continuous function $\theta(t)$. Taking the limit as $h \to 0$ gives the related continuous problem

$$\frac{dx}{dt}(t) = F(\theta(t), x(t)), x_0 = \mathbf{L}x$$

for $t \in [0, T]$, where T corresponds to the output layer.

The ODE interpretation of the ResNet gives new flexibility and insight into the design of neural network layers and their connections. One common problem encountered in neural networks is the phenomenon of exploding and vanishing gradients where some weights increase exponentially, creating instability in both the forward and backward propagation [1]. The ODE formulation provides new tools for solving this problem. First, the Euler discretization is only a first-order method and is not very stable, so replacing it with more stable methods such as ones in the Runge-Kutta family might

add stability. Second, the step size h can be decreased as needed. Finally, restrictions can be put on the $\mathbf{K}_1$ and $\mathbf{K}_2$ operators to increase stability.

With an ordinary ResNet the way the input data changes in the network can be understood as the discretization of an ODE with an artificial time variable t. The ResNet CNN imposes additional restrictions on the changes in the input data through the network because in each layer the transformation is spatially limited by the size of the kernel. This leads to an understanding of the ResNet CNN as a discretization of a PDE with a time variable t and m spatial variables, where m equals the dimension of the input data [15]. This is best illustrated using a one-dimensional example where the input, like image data, is the discretization of a function on on a grid. Consider a function $g(u) : \mathbb{R} \to \mathbb{R}$ that is discretized on a grid [0,1] with k cell-centers and mesh size $h = 1/k$. This results in a vector $\mathbf{g} = [g(u_1), \ldots, g(u_k)]^T$ where $u_i = (i - \frac{1}{2})h$ for $i = 1, 2, \ldots, k$. Now let $\mathbf{K}_1(\theta^1) \in \mathbb{R}^{k \times k}$ be a convolution operator on g that is parameterized by a kernel in $\mathbb{R}^3$. By taking the action of the kernel on the discretized input data and applying a basis change, we arrive at a finite difference operator on the g,

$$\begin{aligned}\mathbf{K}_1(\theta^1)\mathbf{g} &= \begin{bmatrix}\theta_1 & \theta_2 & \theta_3\end{bmatrix} * \mathbf{g} \\ &= (\frac{\beta_1}{4}\begin{bmatrix}1 & 2 & 1\end{bmatrix} + \frac{\beta_2}{2h}\begin{bmatrix}-1 & 0 & 1\end{bmatrix} + \frac{\beta_3}{h^2}\begin{bmatrix}-1 & 2 & 1\end{bmatrix}) * \mathbf{g},\end{aligned}$$

where $\beta_1, \beta_2, \beta_3$ are given by

$$\begin{bmatrix}\frac{1}{4} & \frac{-1}{2h} & \frac{-1}{h^2} \\ \frac{1}{2} & 0 & 2 \\ \frac{1}{4} & \frac{1}{2h} & \frac{1}{h^2}\end{bmatrix}\begin{bmatrix}\beta_1 \\ \beta_2 \\ \beta_3\end{bmatrix} = \begin{bmatrix}\theta_1 \\ \theta_2 \\ \theta_3\end{bmatrix}.$$

Taking the limit as $h \to 0$ gives

$$\mathbf{K}_1(\theta^1) = \beta_1(\theta^1) + \beta_2(\theta^2)\partial_u + \beta_3(\theta^3)\partial_u^2. \tag{2.10}$$

These terms correspond to reaction convection and diffusion, respectively. The finite difference approximations used to illustrate this are not unique. Choosing any linearly independent finite difference operators for the corresponding terms in (2.10) will yield the same result. Importantly, this result can be extended to higher dimensions with relative ease. If our function is now $g(u) : \mathbb{R}^2 \to \mathbb{R}^2$, as is the case with the image data being considered

in this thesis, and the kernel θ^1 is now in $\mathbb{R}^9$, the equation corresponding to (2.10) will be

$$\begin{aligned}\mathbf{K}_1(\theta) =&\beta_1(\theta) + \beta_2(\theta)\partial_u + \beta_3(\theta)\partial_v \\ &+\beta_4(\theta)\partial_u\partial_v + \beta_5(\theta)\partial_u^2 + \beta_6(\theta)\partial_v^2 \\ &+\beta_7(\theta)\partial_u^2\partial_v + \beta_8(\theta)\partial_u\partial_v^2 + \beta_9(\theta)\partial_u^2\partial_v^2.\end{aligned}$$

Thus, ResNet CNNs can be interpreted as discretized PDEs where the input data changes slowly through "time" as defined by the artificial variable t. This is illustrated in figure 2.1 which shows an image datum, the handwritten digit two, being propagated through a CNN. The first layer applies three separate convolutions to expand the width, or number of channels, of the network to three, and then there is a pooling layer to reduce image resolution and a final convolution layer. Importantly, the datum changes slowly over time. The next section discusses a new technique for neural networks that is made possible by this connection.

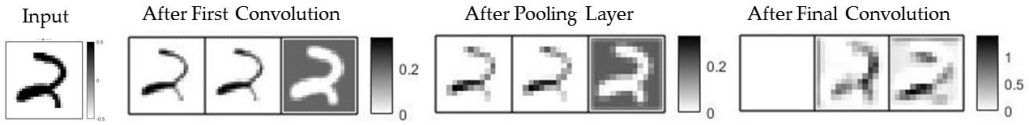


Figure 2.1: The handwritten digit changes slowly through "time" as it is propagated

2.4 Introducing Multigrid Methods

The connection between the ResNet CNN and PDEs means that the methods from PDE analysis can be applied to ResNet CNNs to make them adaptable to data of different resolutions as well as work on increased accuracy, stability and speed of training. To illustrate this consider a PDE discretization on a grid $\Omega^h \in \mathbb{R}^{k\times l}$ resulting in the system of linear equations

$$\mathbf{A}u = f. \tag{2.11}$$

Suppose this system has a unique solution and let v be the computed approximation. The error is a vector defined as

$$e = u - v. \tag{2.12}$$

Common iterative relaxation schemes, such as Gauss-Seidel and Jacobi, quickly eliminate high-frequency modes of the error but are powerless against the low-frequency modes that stubbornly remain. To remove the low-frequency modes, a family of methods called multigrid methods are often used. Multigrid is a very broad field but the focus here is on some basic details and methods as shown in [5]. Multigrid methods are based on the observation that error components that appear to be smooth on a fine grid will become oscillatory on a coarser grid. They provide algorithms for interpolating solutions between the fine grid, Ω^h, and a coarser grid, Ω^{2h}, so that both the smooth and oscillatory components of the error can be removed.

The multigrid algorithms used to transfer solutions between fine and coarse grids can be used to transfer the weights in a ResNet CNN to match the resolution of the given input data. The benefits of this are not limited to just increased adaptability to data of different resolutions. Removing the the low-frequency modes of the error could result in increased accuracy and stability. In addition, the Ω^{2h} grid can be used to obtain a better initial guess for the iterative relaxation scheme on the Ω^h grid. Since the matrix operations involved in both the forward and back propagation through a neural network are roughly four times cheaper when the input resolution is cut in half, this has obvious computational advantages as well. This thesis investigates the effectiveness of multigrid methods at these goals on classical CNNs. In the next chapter, the experimental setting for the experiments is set and the multigrid implementation details are explicated.

Chapter 3

Proposed Method

This chapter details the implementation of the experiments conducted to investigate the performance of multigrid methods on neural networks. Instead of analyzing performance on a variety of data sets, the experiments here are all based on a single canonical data set known as MNIST [11]. MNIST consists of over $60,000$ examples of image data in $\mathbb{R}^{28x28}$ that have already been normalized and centered. While other data sets such as CIFAR-10 and ImageNet represent greater classification challenges, the decision to just use MNIST was made because this allows greater focus on the multigrid methods themselves instead of factors like preprocessing time and computational cost. This chapter first discusses the issue at the core of this thesis, the implementation of multigrid methods on CNNs. Then, the construction of the CNN used for testing is detailed, as well as decisions made regarding the training process and other factors.

3.1 Multigrid Implementation

In this section a procedure for the transfer of network weights between a CNN designed to handle input data of resolution $\mathbb{R}^{28\times28}$ and a CNN designed to handle data of a different resolution. This procedure has two components. The first defines a procedure for the transfer of the network convolution weights θ. In other words, we consider the output of a linear transformation in the first network

$$a^h = \mathbf{K}_1^h(\theta^h)x^h, \tag{3.1}$$

where $x^h \in \mathbb{R}^{n_f}$ is the input to a layer in a feature space of a given resolution and $\mathbf{K}_1^h(\theta^h)$ is a linear transformation from this feature space parameterized by a convolution stencil θ^h. We then consider the output of the linear transformation

$$a^{2h} = \mathbf{K}_1^{2h}(\theta^{2h})x^{2h} \tag{3.2}$$

where x^{2h} is the input to a layer in a feature space of a different resolution $2h$ and $\mathbf{K}_1^{2h}(\theta^{2h})$ is a linear transformation in this feature space parameterized by a different convolution stencil θ^{2h}. We seek to to define two operators, a restriction operator $r : \theta^h \rightarrow \theta^{2h}$ and a corresponding prolongation operator $p : \theta^{2h} \rightarrow \theta^h$. These operators are referred to collectively as the transfer operators and we wish to define them so that the output of the two linear transformations will be the "same" in a sense that will be defined later. The second component defines a procedure for the transfer of the linear classifier weights w. As is standard in multigrid methodology, to be able to do this we first have to define the input resolutions being considered and the corresponding CNNs with linear transformations in different euclidean spaces. We then provide transfer methods for the weights of the two networks.

3.1.1 Defining the Fine and Coarse Grids

A fundamental component of any multigrid method is a family fine and coarse grids. For simplicity sake the family of fine and coarse grids is restricted to only two grids. The first grid, Ω^h, corresponds to weights in the CNN designed for input data from MNIST with the provided resolution $\mathbb{R}^{28\times 28}$. While the image data in MNIST is all of the same resolution, input data of a different resolution can be obtained by interpolating the image data to a different resolution. Interpolating the image data to a resolution finer than the one provided would not provide any more information and would increase computational cost, so the natural choice is to interpolate the image data to a coarser grid. The interpolation to the coarser grid is done using a restriction matrix $\mathbf{R}$. We define a corresponding interpolation to the finer image data by a prolongation matrix $\mathbf{P}$ such that the relationships

$$x^{2h} = \mathbf{R}x^h \quad \text{and} \quad x^h = \mathbf{P}x^{2h} \tag{3.3}$$

hold. Thus, the grid Ω^h is defined as the fine grid and the coarse grid, Ω^{2h}, corresponds to the weights in a CNN designed for input data of $\mathbb{R}^{14\times 14}$. The number of layers and structure in the two corresponding CNNs will be the

same, but the linear transformations in the CNN corresponding to the coarse input data will all take place in a feature space with half the resolution of the corresponding linear transformation in the CNN used for the fine input data. Using a coarse grid corresponding to input data with twice the spacing between the discretization points is a near universal practice because it makes the interpolation more straightforward and there is no advantage to using grids with spacing ratios other than two [5].

3.1.2 Interpolation Of Network Weights

There are two major families of multigrid methods, geometric (or standard) multigrid, and Algebraic Multigrid (AMG). In the geometric case the input data x corresponds to points at known spatial locations, and the operators $\mathbf{R}$ and $\mathbf{P}$ on the weights parameterizing the linear transformation would be based on the geometric relationships between the points. With AMG, the input data does not have a strict geometric interpretation. Instead, the operators are determined by the structure of the linear transformation itself. Geometric multigrid would seem like the more natural choice for this problem because image data does have known spatial structure. However, the transfer operators used in geometric multigrid are most effective as the spacing between the interpolation points goes to zero. Given the low resolution of MNIST data geometric multigrid may perform poorly so this thesis uses transfer operators from AMG.

Consider the outputs of a linear transformations in the CNN for the fine image data and the output of a linear transformations in the CNN for the coarse image data, along with the prolongation and restriction matrices P and R between the two feature spaces. We define the transfer operators so that

$$\mathbf{K}_1^{2h}(\theta^{2h})x^{2h} = \mathbf{R}\mathbf{K}_1^h(\theta^h)\mathbf{P}x^{2h}. \tag{3.4}$$

In other words, the transfer operators are defined so that the linear transformation applied to the coarse image data is equal to the restriction of the linear transformation applied to the prolongated fine image data. This result implies that

$$\mathbf{K}_1^{2h}(\theta^{2h}) = \mathbf{R}\mathbf{K}_1^h(\theta^h)\mathbf{P}, \tag{3.5}$$

which is the Galerkin condition for the construction of the coarse-grid operator [5]. Here, of course, θ^h is the fine scale convolution stencil of size $n_k \times n_k$. Since $\mathbf{R}$ and $\mathbf{P}$ are linear transformations and the unknown is θ^{2h}, a classic result multigrid theory says that coarse mesh stencil will also be of size $n_k \times n_k$, so the coarse stencil can be determined with n_k^2 equations [5, 7]. The same methodology can also be used to define the transfer operators from θ^{2h} to θ^h. There is no multigrid interpretation for the bias terms so these are not changed in the methodology.

3.1.3 Interpolation Of Classifier Weights

The interpolation of the classifier weights can no longer use the Galerkin condition. This is because the classifier weights are not parameterized by convolution stencils, and, more importantly, and are not mapping between feature spaces of different resolutions. Instead, the classifier weights of the coarse and fine networks are mappings from different feature spaces to the same discreet classification space so multigrid conditions do not have meaning in this setting. We can get around this dilemma by simply applying the prolongation or restriction matrix to go from coarse-to-fine classifier weights or fine to coarse, respectively. However, one adjustment does need to be made. Consider the mapping $\mathbb{R}^{n\times n} \to \mathbb{R}$ from the output of one of the kth network channel, $g(x_i, \theta)_k$, to the jth element of the discreet classification space by the classifier weights. This is a discreet version of of the continuous operation

$$y_{i,j} = \int_\Omega g(x_i, \theta)_k \cdot w_{k,j}.$$

When this operation is discretized using the coarse network output it becomes the Riemann sum

$$y_{i,j}^{2h} = \frac{1}{n^2} \sum_{i=1}^{n} \sum_{i=1}^{n} g(x_i, \theta)_k \cdot w_{k,j}$$

but the corresponding Riemann sum from the fine network output is

$$y_{i,j}^{h} = \frac{1}{4n^2} \sum_{i=1}^{2n} \sum_{i=1}^{2n} g(x_i, \theta)_k \cdot w_{k,j}$$

so we multiply the classifier weights four when going from coarse-to-fine and divide by four when going from fine to coarse.

Importantly, the mapping from the coarse classifier weight space to the fine classifier weight space, or vice versa, requires the classifier weights to be piece-wise smooth. More specifically, each mapping from a network channel output to an element of the discreet classification space must be smooth or the interpolation will not be useful.

3.1.4 Choice of Prolongation and Restriction Matrices

There are many ways to choose the interpolation operators $\mathbf{R}$ and $\mathbf{P}$ as the most important component of the multigrid methods is the Galerkin condition. Here, we revert back to the geometric interpretation of the image data. First, the restriction operator $\mathbf{R}$ takes two by two blocks from an input in the fine space and simply averages them to return the corresponding element of the output in the coarse space. The prolongation operator acts slightly differently by performing a weighted averaging of elements from the coarse input to return the elements of the fine output.

3.2 Network Frameworks and Optimization Methods

Variations of one CNN framework are used for the experiments in this thesis. The construction and training of this network as well as the multigrid implementation is all done in MATLAB using the Meganet package from XtractOpen [2]. This framework is a basic CNN consisting of two to three network layers. The first layer is a convolutional layer that takes the MNIST data as its input and expands the network width from one to thirty-two. Although this terminology was introduced in chapter two, network width is defined more rigorously here as the number of outputs of the layer. Using the layer notation as previously defined, this means that the layer

$$F(\theta, x) = \mathbf{K}_2(\theta^2)\sigma(\mathbf{K}_1(\theta^1)x + \mathbf{B}(\theta^3)) \tag{3.6}$$

has only one x input (corresponding to one MNIST image datum) but obtains thirty-two outputs by applying thirty-two distinct convolutions to the input. Here, and in all the convolution layers, the $\mathbf{K}_2(\theta^2)$ term is the identity and the $\mathbf{K}_1(\theta^1)$ term is a sparse circulant matrix parameterized by the

three-by-three convolution kernel θ^1. The bias term is zero.

In some of the variations of the network architecture the next layer is a special type of network layer called a pooling layer. A pooling layer has no trainable weights and does not apply a linear transformation to the data. Instead, it just "pools", or interpolates, the elements of the inputs to the layer to reduce their resolution. The pooling layer here utilizes a two-by-two filter with stride two. Stride refers to the space between each sample, so a two-by-two filter with stride two samples separate four pixel blocks of the input and returns one output for each block. As a result, the output of the pooling layer will have half the resolution of the input. The rationale behind the use of this pooling layer is that it will reduce the number of trainable parameters later in the network, which reduces computational complexity and helps to prevent over-fitting. There are several types of pooling layers but the one utilized here averages the pixels under consideration to return the output.

The final layer of the basic CNN is another convolutional layer similar to the first layer, but its input has width thirty-two and its output has width sixty-four.

3.2.1 Regularization and Optimization

To train the CNN, the network and classifier weights are first initialized randomly using a normal distribution. Then, the network and classifier weights are optimized. There are two optimization methods that are used in the thesis to solve the optimization problem, SGD and the CG-Steihaug inexact newton method.

The first optimization method, SGD, is a procedure defined in (2.4) and (2.5). The mini-batch, defined as the number of random samples used to compute the gradient in each iteration, is set at 32. The learning rate used here is the scalar sequence

$$\alpha_k = \frac{.003}{\sqrt{.5 \times e}}$$

where here e is the current epoch and one epoch is defined as one pass through all of the training data sampled without replacement.

CG-Steihaug is also an iterative optimization method. In CG-Steihaug, each iteration k finds an approximate solution to

$$\min_{p\in\mathbb{R}^n} m_k(p) = l_k + (\nabla l_k)^T p + \frac{1}{2}p^T(\nabla^2 l_k)p \quad \text{subject to} \quad \|p\| \leq \Delta_k, \quad (3.7)$$

where l is the objective function, p is the solution and the step taken, and Δ_k is a scalar that defines the ball known as the trust region [13]. The use of a trust region tries to ensure that the step taken is in a region where the gradient and hessian still provide good approximations of the true nature of the objective function.

The CNN also utilizes regularization in the optimization methods, i.e. the objective function (2.3) has a regularization term $R(\theta, w)$ added. The regularizing term can be split into two parts; $R(\theta, w) = R(\theta) + R(w)$. The regularization method used for both terms is Tikonov regularization. The Tikonov regularization used here for the theta weights is defined as

$$R(\theta) = .5\alpha\|\mathbf{B} \times \theta\|^2,$$

where $\alpha = 2^{-3}$ is a scalar and $\mathbf{B}$ is the identity operator. This implementation penalizes large network weight values which can create instability and exploding or vanishing gradients.

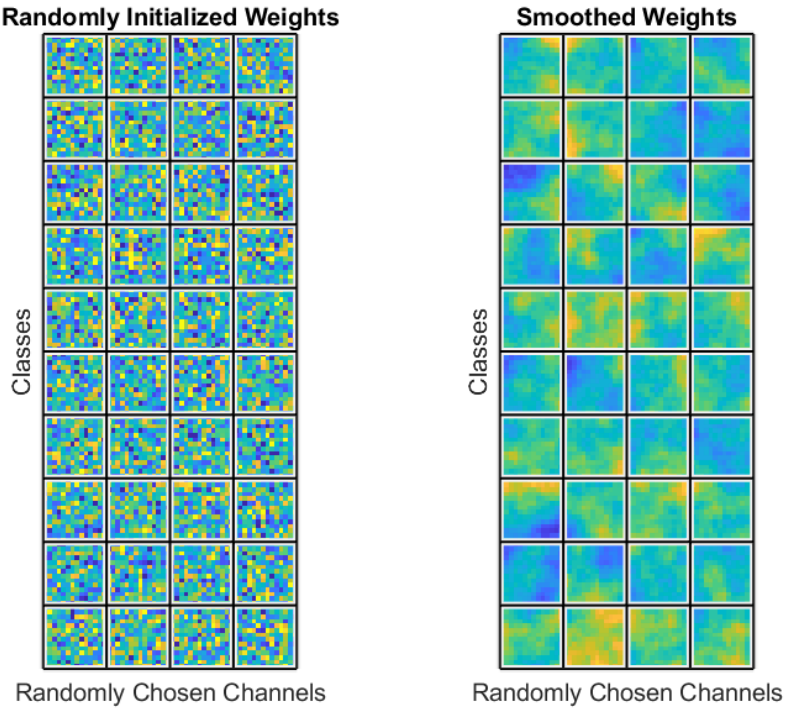
Without regularization the trained weights are not smooth and interpolation will be unsuccessful. The $R(w)$ term is designed to penalize non-smoothness in the classifier weights to meet the condition given in section 3.1.3. Thus we define

$$R(w) = .5\alpha\|\mathbf{B} \times w\|^2$$

where $\alpha = 2^{-5}$ for the network for the fine MNIST data and $\alpha = 4 \cdot 2^{-5}$ for the network for the coarse MNIST data because of the Riemann sum coefficients discussed in 3.1.3. Instead of the identity operator $\mathbf{B}$ now computes a fine difference approximation of the gradient. This will penalize non-smooth classifier weights. However, for this regularization to be effective the classifier weights must be initialized smoothly as well. To implement this, the initially randomly generated classifier weights are restricted to the domain $[-.2, .2]$ and then subject to a discrete cosine transform. In the discrete cosine space the variance of the classifier weights is reduced and the weights are then transformed back to euclidean space. Figure 3.1 shows twenty (10

x 2) random coarse network weight blocks. They are highly unsmooth when initialized randomly but become smooth after the variance has been reduced.

Figure 3.1: Smoothing of Randomly Initialized Classifier Weights



Chapter 4

Numerical Experiments

In this chapter several numerical experiments covering the performance of multigrid methods on several variations of the CNN are explicated and the results are analyzed. In addition to numerical results, data visualizations are used to help interpret the performance. In section 4.1 the general structure of the numerical experiments conducted is discussed. Section 4.2 covers the baseline performance of multigrid on the network. Sections 4.3 covers the effects of pooling on multigrid performance. Finally, section 4.4 looks at how different optimization methods effects performance and convergence.

4.1 Experiments Run

The general structure of the numerical experiments here is as follows. First, the coarse network is trained on 40,000 examples of coarse MNIST data. Then, these weight values are stored and multigrid methods are applied to both the network and classifier weights. This process creates two networks for classifying the fine resolution data with the same structure but different weight values. The first network has the network weights with the multigrid method applied and with the prolongated classifier weights and is referred to as the fine network with multigrid. The second network has the network weights without the multigrid method applied and is referred to as the fine network without multigrid. It also uses the prolongated classifier weights because the size of the classifier weight matrix is dependent on the size of the network output. For comparison, the fine network is also trained on newly

initialized weights, a process referred to as new initialization.

Next, the three networks are all subsequently retrained on 10,000 examples of fine MNIST data. The objective function values on the fine data for both networks are computed. The networks are subsequently retrained on the fine data and objective function values, validation loss, runtimes, and other parameters are analyzed. This process is referred to as coarse-to-fine training and forms the core of this thesis.

4.2 Baseline Results

To understand how factors like pooling, choice of activation function, and optimization methods effect multigrid performance, a baseline result is needed to compare with. For this baseline a pooling layer is not used and the ReLU function as the activation function. Twenty epochs of SGD are used to train on the coarse image data and then SGD is used again to retrain on the fine data. The performance of multigrid on the fine data is compared with transferring the weights and training from randomly initialized weights.

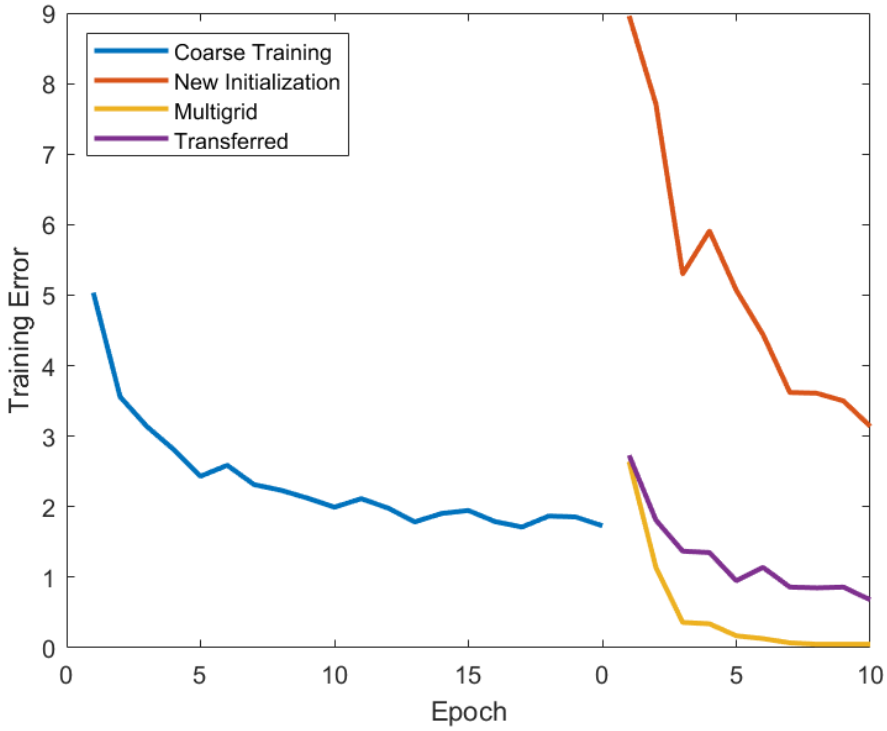
Twenty trials of this coarse-to-fine training process were run and the summary statistics are displayed in table 4.1. The "Initial" column show classification accuracy on both the fine training data and fine validation data after the first epoch of retraining, while the "ReTrain" column shows these statistics after ten epochs of SGD. The first row shows results when retraining starts from initially randomized weights, the second row shows results when the weights were transferred, and the third row has the performance of multigrid.

Table 4.1: Baseline Performance of Coarse-to-Fine Training

	Initial		ReTrain	
Training Type	Training	Validation	Training	Validation
New	88.4	85.0	96.4	91.9
Transfer	97.9	93.5	99.4	94.6
Multigrid	97.5	62.9	99.9	63.6

Both transferring the weights and just using multigrid show significant advantages over new training from an optimization prospective. Both methodologies perform very well on the training data after just one epoch of SGD and converge to better local minimum after the ten epochs, with multigrid performing the best, as shown in the convergence plot in figure 4.1.

Figure 4.1: Convergence of Multigrid

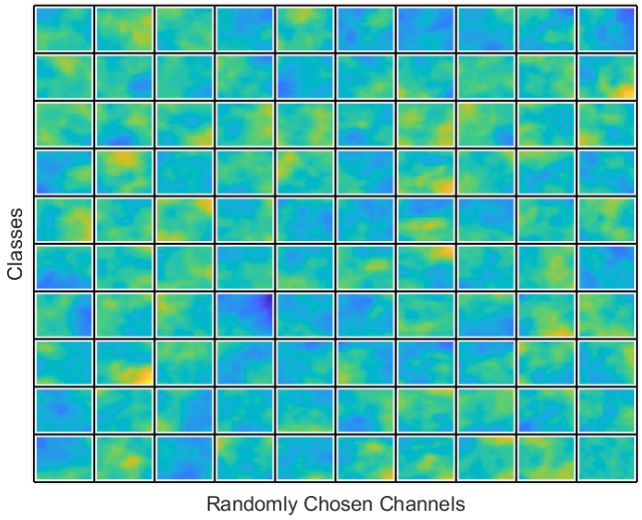


This plot uses one of the twenty trials and demonstrates multigrid's performance advantage. Training error for the twenty epochs of coarse training is shown as well as training error for the ten epochs of retraining. However, multigrid methods overfit the data and converge to a local minimum that generalizes very poorly. To gain insight into why this is true the smoothness of the weight values that the original coarse network has trained, the loss landscapes of the fine networks, and visualize image data propagated

through the fine networks are analyzed.

One possible reason for the poor generalization performance of multigrid could be related to the smoothness of the classifier weights after the coarse training. If regularization is not effective, than the classifier weight interpolation will not be effective. However, in figure 4.2, which shows 100 random blocks of the classifier weights after coarse training, the classifier weights are piece-wise smooth.

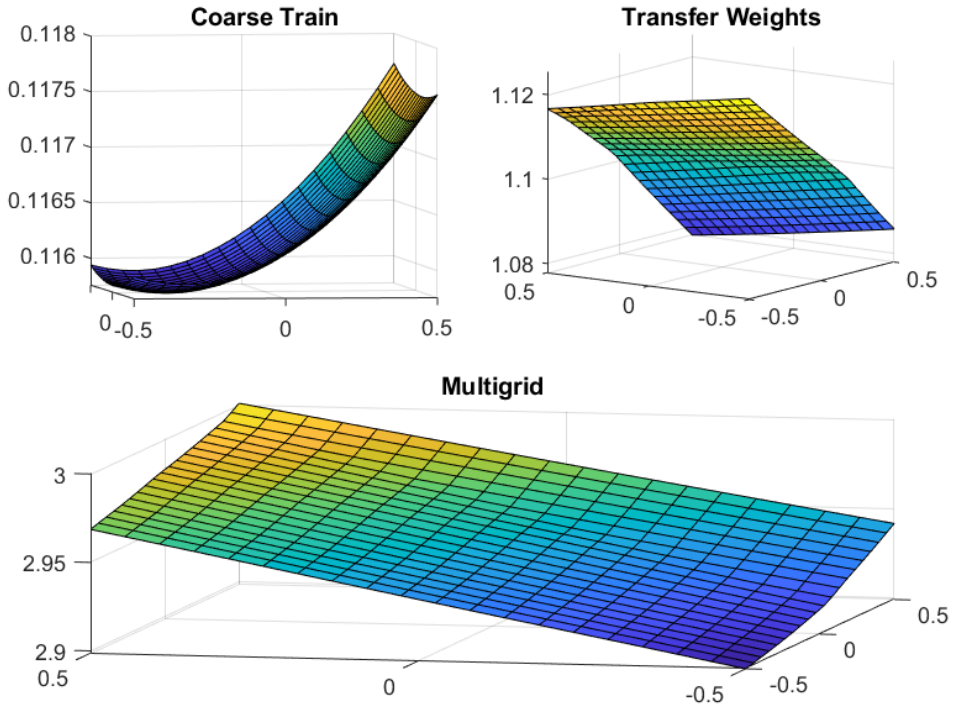
Figure 4.2: Classifier Weights after Training



Another possible reason for multigrid's poor generalization could be related to the position of the weight values in the loss landscape. The loss landscape can show how the objective function changes as the network and classifier weights are perturbed. However, since there are tens of thousands of weight values in even the relatively simple networks used here, the high dimensionality of the loss landscape makes straight-forward visualization impossible. To get around this issue, methodology explicated in [12] that visualizes the loss landscape by plotting the objective function values in two random normalized directions is utilized.

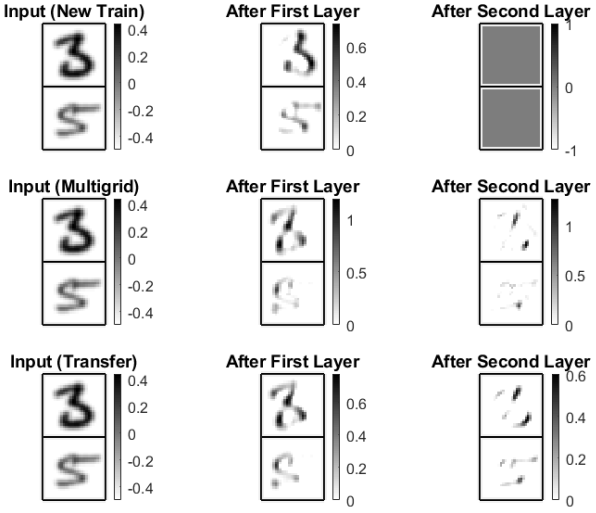
The results of this visualization for a single instance of the coarse-to-fine training are displayed in figure 4.3. The top left quadrant shows the objective function values where the point $(0, 0)$ is the minimum that the original coarse training reach reached. The top right shows the same plot using the point that the network with transferred weights starts retraining from, and the bottom plot uses the weights after multigrid is applied as the central point. Importantly, the loss landscape near the minimum that the coarse training reached is not highly convex. Unsurprisingly, transferring or applying multigrid to the weights results in a loss landscape that is non-convex. Interestingly, however, the loss landscape for the transferred weights is much less sensitive to perturbations in the weight values than the multigrid network is and, gives a clear descent direction as a linear combination of the two random directions. The multigrid network is highly sensitive to perturbations and has less clear of a descent direction.

Figure 4.3: Perturbations in the Loss Landscape



To gain addition insight into the generalization gap and difference in performance, image data propagated through the network frameworks is visualized. Figure 4.4 visualizes image data propagated through a random channel of the fine network with three weight configurations. The first row shows results using weights trained from new initialization, the second row row uses weights from the fine network with multigrid after retraining, and the third row uses weights from the fine network without multigrid after retraining. Training from new initialization does not generate recognizable image data. However, the propagated image data in the third row is slightly more clear and recognizable than the difficult to recognize propagated data in the second row. Transferring the weights and retraining generates clearer propagated data than multigrid and retraining.

Figure 4.4: Propagation of Two Random Inputs using Coarse-to-Fine training



4.3 Pooling

This section details how the use of pooling can affect multigrid performance. In the multigrid context considered here, multigrid methods prove somewhat effective at preventing overfitting. Whereas in the previous section the

multigrid generalization gap was extremely large, pooling succeeds in partially closing this gap.

Table 4.2 shows the summary statistics for twenty trials of coarse-to-fine training using the same methodology and format as table 4.1. The only difference here is the network utilizes a pooling layer between the first and second convolution layers. Crucially, the generalization gap for multigrid here, while still extant, is much smaller than it was in table 4.1. Again, multigrid performs the best from an optimization perspective.

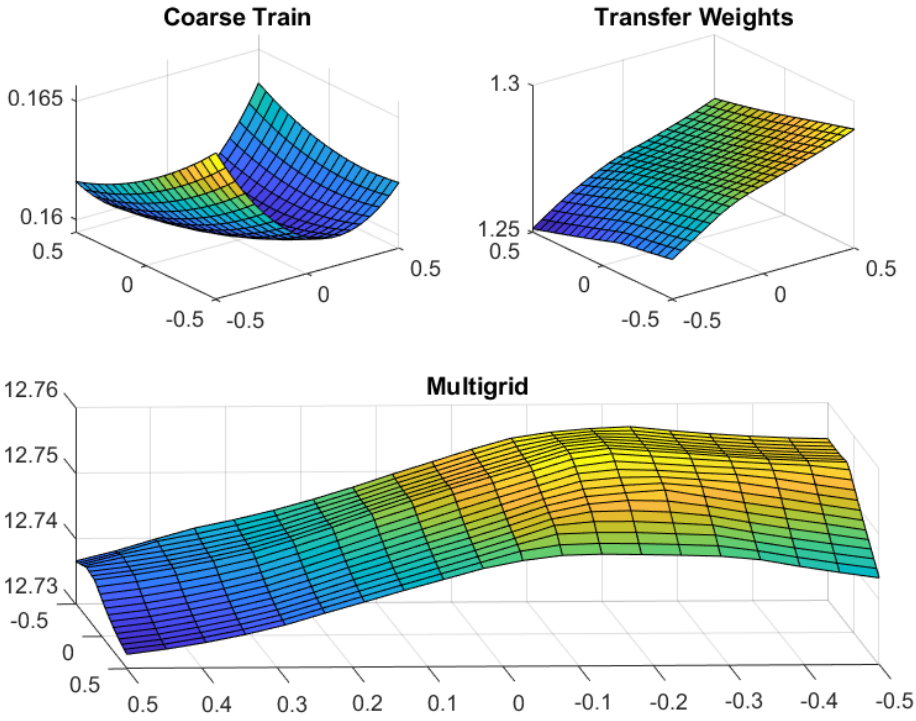
Table 4.2: The Effect of Pooling on Multigrid Performance

	Initial		ReTrain	
Training Type	Training	Validation	Training	Validation
New	91.1	89.0	97.0	93.2
Transfer	96.7	91.5	98.3	92.8
Multigrid	96.8	84.1	99.8	88.0

The loss landscape of the network with pooling has similar properties to the loss landscape of the network without pooling. Figure 4.5 shows how perturbations affect the objective function in the same format as figure 4.3. Results here are similiar in that the transferred weights are much more resilient to perturbations than the multigrid weights. One possibly crucial difference is that the loss landscape appears to be more convex around the minimum reached by the coarse training.

The propagation of random inputs displayed in figure 4.6 also lends some insight as to why pooling causes better performance. The propagated image data in the last column are all more clear than the propagated image data from the network without pooling. The propagated data from the network with multigrid is recognizable as an eight and a three and contains many of the crucial curves that define the number three.

Figure 4.5: The Pooling Loss Landscape



4.4 Optimization Methods

In this section the effect of changing the optimization method for the retraining process is analyzed. The network with multigrid weights in the last two sections has weight values that do not generalize well before retraining. Retraining with SGD is only partially effective at closing this generalization gap, so here the CG-Steihaug method described in (3.7) is used to see if it is effective at closing this gap.

The results show that CG-Steihaug actually performs worse than just retraining with SGD. Table 4.3 shows results when the retraining process first trains the classifier for five epochs of CG-Steihaug while holding the network weights fixed. Then, both the classifier and network weights are trained for ten epochs of SGD. Training the classifier beforehand results in overfitting on the multigrid weights.

Figure 4.6: Propagation and Pooling

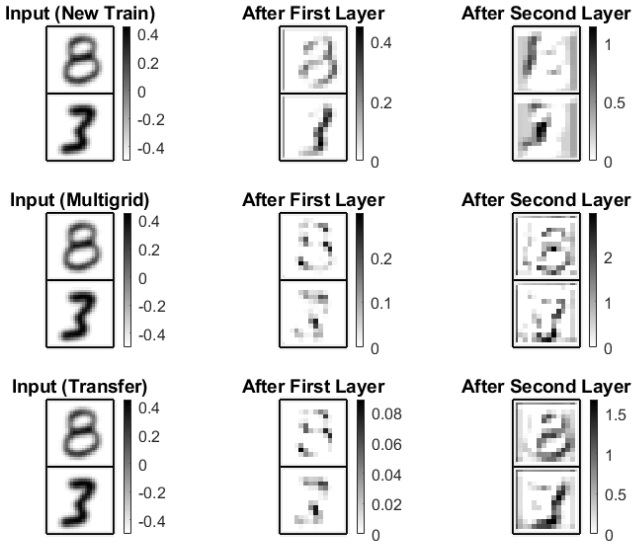


Table 4.3: Training the classifier using CG-Steihaug

	Classifier Training			Network Training		
Training Type	Initial		After	—Initial		After
	Training	Val.	Val.	Training	Val.	Val.
New	32.5	34.1	75.3	91.1	89.0	92.2
Transfer	94.4	82.1	92.9	99.1	94.0	93.8
Multigrid	85.2	34.1	38.3	97.6	60.3	62.4

In Table 4.4 the classifier weights and then both the network and classifier weights are trained using CG-Steihaug. The performance in both cases is very poor; in particular training just using CG-Steihaug is very ineffective and results in a generalization gap even when starting from new initialization. These results emphasize the difficulty of the learning problem and the difficulty of converging to a local minimum that generalizes well.

Table 4.4: Training the Classifier and Network with CG-Steihaug

	Classifier Training			Network Training		
Training Type	Initial		After	—Initial		After
	Training	Val.	Val.	Training	Val.	Val.
New	51.8	50.3	76.6	91.9	77.1	79.5
Transfer	92.8	81.5	91.4	99.7	91.7	92.4
Multigrid	85.2	32.6	35.9	97.2	42.4	50.8

Chapter 5

Summary and Conclusion

The results of the experiments run in this manuscript illustrate important properties the neural network learning problem and of the MNIST data set, as well as some of the difficulties to consider when implementing multigrid methods in the neural network context. The goal of the learning process, to converge to a good local minimum with low training error and strong generalization, is extremely difficult given the non-convex loss landscape and the problems with generalization. The experiments here show how convergence to a good local minimum is far from guaranteed and is heavily dependent on network architecture decisions and optimization methods. In addition, MNIST data is low-resolution and piece-wise constant. These properties makes the application of multigrid methods difficult, as discontinuities can slow down the convergence rate of algebraic multigrid or result in complete failure to converge [5]. The neural network setting here, and the addition of complications like point-wise biases terms that cannot be easily addressed with multigrid methods, further hurts the performance.

In the numerical experiments here the benefits of pooling and other measures that prevent overfitting when applying multigrid methods can be seen. This is because in the coarse-to-fine setting considered here the properties of the minimum reached by the coarse training are of crucial importance to multigrid performance. In addition, the effectiveness of retraining after multigrid is heavily dependent on the choice of optimization method.

Future directions of research in this field build off the limitations of this work. Many of these limitations are related to the very limited context of

these experiments. These future directions include the following:

- testing the performance of multigrid methods on other data sets, including video data
- using a wider variety of network architectures and optimization methods
- testing fine-to-coarse training and joint training (training on both the coarse and fine data at the same time)
- Using geometric instead of algebraic multigrid (particularly on higher-resolution data sets)

This thesis allowed me to incorporate different applied math fields including optimization methods, PDEs, and multigrid. In addition, I gained experience with computational aspects such as MATLAB coding and GPU computation. The results show the importance of careful consideration of optimization methods and network architectures when implementing multigrid methods. It also shows that in certain situations simply transferring the weights from a coarse trained network can result in increased performance on the desired resolution. More experimentation is needed to determine the specific context and architectures where multigrid is most effective and where it fails.

Appendix A

Main Notation

$\mathbb{R}^n$	n-dimensional Euclidean space
$\Omega \subset \mathbb{R}^d$	domain
Ω^h	domain of the fine grid
Ω^{2h}	domain of the coarse grid
J_n	objective function
R	regularizing term
X	input space, e.g. input to a layer
x_i	input data
$x_{i,j}$	data propogated to layer j
H	a class of functions
f, g	functions
l	loss function
σ	activation function
$\mathbb{1}$	the indicator function
θ	network weights
w	classifier weights
n_f	dimension of feature space
α, h	scalars
t	time
$\mathbf{L}, \mathbf{B}$	linear transformations
$\mathbf{K}_n(\theta^n)$	a linear transformation parameterized by convolution stencil
F	network layer
$\mathbf{P}$	prolongation matrix
$\mathbf{R}$	restriction matrix
r, p	transfer operators

Appendix B

Abbreviations

DNN	Deep Neural Network
CNN	Convolutional Neural Network
ODE	Ordinary Differential Equation
PDE	Partial Differential Equation
ResNet	Residual Neural Network
SGD	Stochastic Gradient Descent

Bibliography

[1] Gradient Flow in Recurrent Nets: The Difficulty of Learning LongTerm Dependencies. In *A Field Guide to Dynamical Recurrent Networks*. IEEE, 2009. 8

[2] A fresh approach to deep learning written in MATLAB: XtractOpen/Meganet.m, Feb. 2019. original-date: 2018-01-07T23:56:09Z. 16

[3] B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Srndic, P. Laskov, G. Giacinto, and F. Roli. Evasion attacks against machine learning at test time. *CoRR*, abs/1708.06131, 2017. 7

[4] L. Bottou, F. E. Curtis, and J. Nocedal. Optimization Methods for Large-Scale Machine Learning. *arXiv:1606.04838 [cs, math, stat]*, June 2016. arXiv: 1606.04838. 3, 4

[5] W. L. Briggs, V. E. Henson, and S. F. McCormick. *A Multigrid Tutorial: Second Edition*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000. 2, 11, 14, 15, 30

[6] E. Haber and L. Ruthotto. Stable architectures for deep neural networks. *CoRR*, abs/1705.03341, 2017. 2, 6

[7] E. Haber, L. Ruthotto, E. Holtham, and S. H. Jun. Learning Across Scales - Multiscale Methods for Convolution Neural Networks. page 7. 2, 8, 15

[8] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. 6, 7

[9] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. *CoRR*, abs/1603.05027, 2016. 7

[10] Y. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, and L. D. Jackel. Handwritten digit recognition with a back-propagation network. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pages 396–404. Morgan-Kaufmann, 1990. 1, 6

[11] Y. LeCun and C. Cortes. MNIST handwritten digit database. 2010. 12

[12] H. Li, Z. Xu, G. Taylor, and T. Goldstein. Visualizing the loss landscape of neural nets. *CoRR*, abs/1712.09913, 2017. 7, 23

[13] J. Nocedal and S. Wright. *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. Springer Science & Business Media, New York, Dec. 2006. 18

[14] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, Oct. 1986. 5

[15] L. Ruthotto and E. Haber. Deep neural networks motivated by partial differential equations. *CoRR*, abs/1804.04272, 2018. 2, 9

[16] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. J. Goodfellow, and R. Fergus. Intriguing properties of neural networks. *CoRR*, abs/1312.6199, 2013. 7