THESIS

LATE RESIDUAL NEURAL NETWORKS: AN APPROACH TO COMBAT THE DEAD RELU PROBLEM

Submitted by

Matthew Frederick Ernst

Department of Computer Science

In partial fulfillment of the requirements

For the Degree of Master of Science

Colorado State University

Fort Collins, Colorado

Fall 2021

ABSTRACT

LATE RESIDUAL NEURAL NETWORKS: AN APPROACH TO COMBAT THE DEAD RELU PROBLEM

The rectified linear unit (ReLU) activation function has been a staple tool in deep learning to increase the performance of deep neural network architectures. However, the ReLU activation function has trade-offs with its performance, specifically the dead ReLU problem caused by vanishing gradients. In this thesis we introduce "late residual connections" a type of residual neural network with connections from each hidden layer connected directly to the output layer of a network. These residual connections improve convergence for neural networks by allowing more gradient flow to the hidden layers of a network.

ACKNOWLEDGEMENTS

I would like to acknowledge the many people that helped me in pursuing my thesis. First and foremost, I would like to thank Dr. Darrell Whitley for his infinite support, expertise and knowledge in this field. I have learned and understood more from Dr. Whitley than any other professor. I would also like to thank Dr. Chuck Anderson, Dr. Nathaniel Blanchard, and Tim Whitaker for support on technical aspect of this thesis. Lastly, I would like to thank my family and friends for supporting me through my time at Colorado State University.

DEDICATION

I would like to dedicate this thesis to my mother Christine Perich, who encouraged me to pursue higher education and supported me throughout my studies at Colorado State University.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

1.1 Overview

Deep learning utilizes machine learning algorithms such as neural networks to solve complex problems. Some of the uses for deep learning includes image processing with convolutional neural networks (CNN) for classification problems, text to speech applications with natural language processing (NLP), and countless others. The deep learning community started to takeoff around 2015 when graphic processing units (GPUs) were used to solve issues such as training speeds and to handle significant data volumes. Increased availability to GPUs allowed researchers to start developing complex and high-performance algorithms specific to neural networks, such as AlexNet [Alom et al., 2018]. Along with hardware, AlexNet also utilized neural network training advances such as the new activation functions designed to improve the learning of deep neural network architectures [Bengio, 2009].

The most common activation function used to improve deep learning is the rectified linear unit activation function (ReLU). Hahnloser introduced ReLU in 2000 during his research on digital circuits [Hahnloser et al., 2000]. Soon deep learning communities started to find uses for this function in the training of neural networks. The appeal of the ReLU activation function is its ability to preserve the gradient in deep architecture settings. The ReLU is also fast due to its simplicity. The ReLU is expressed as $f(x) = max(0, x)$, where x is the summation of the weights from a previous layer of a network into a given neuron. x can be referred to as the raw NET. The value of the raw NET is then passed through a ReLU and its value remains the same or is changed to a zero if the raw NET is negative.

The ReLU has a much simpler derivative than more traditional activation functions such as Sigmoid and TanH. For ReLU, the derivative is either zero or one, again reducing complexity and overall computation cost [Agarap, 2019]. The ReLU activation function and its derivative can be seen in Figure 1.1. However, with the ReLUs simplicity comes problems, specifically the dead ReLU problem caused by vanishing gradients when the raw sum of inputs is consistently less than zero.

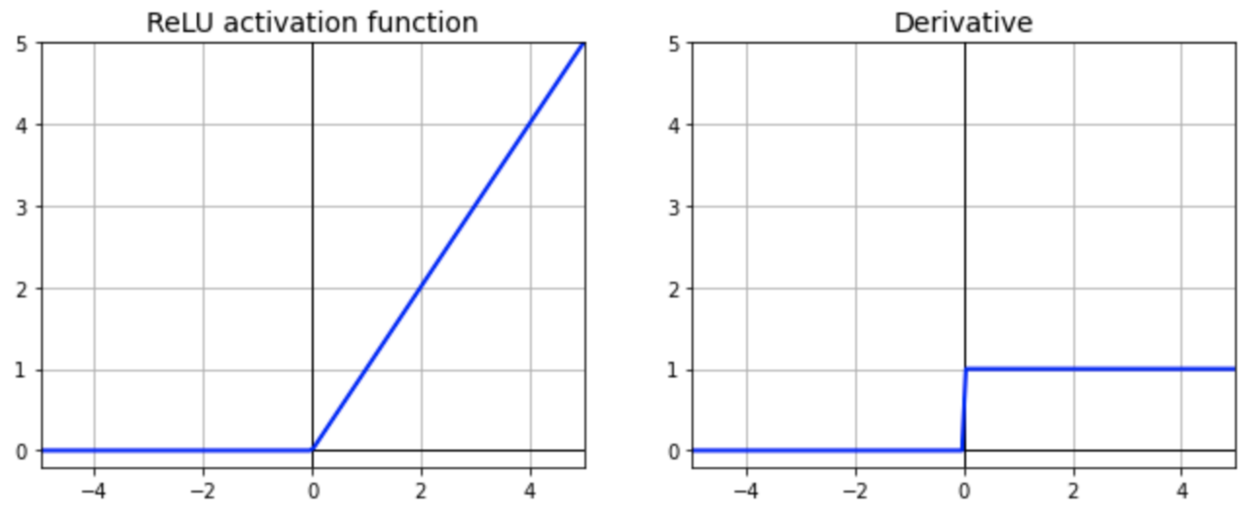


Figure 1.1: The ReLU activation and the derivative of it. The ReLU activation function is expressed as $f(x) = max(0, x)$ where x is the raw NET. The raw NET is the summation of the weights from a previous layer. [Google, 2017]

1.2 Contributions

The contributions of this thesis are as follows:

- We demonstrate the dead ReLU problem in a sample problem, showing the shortcomings of neural networks that encounter the issue. Moreover, we document these shortcomings on various learning rates, deep neural network architectures, and optimizers.

- We develop a new methodology that combats the dead ReLU problem by adding residual connections, named "late residual connections", to standard neural networks. These late residual connections are tested against non-residual connections and we illustrate that this new technique shows improved convergence and learning capabilities for neural networks.

Chapter 2

Review of Literature

2.1 The Dead ReLU Problem

The ReLU activation function is used extensively in deep learning for its gradient properties and speed. However, a neuron using ReLU as an activation function can fall victim to vanishing gradients. When this problem affects a neuron, the ReLU can collapse and become a constant function; this means the neuron no longer contributes to the network [Arnekvist et al., 2020].

An example of the vanishing gradient problem can be outlined in ReLUs derivative and its role in backpropagation. When a derivative of a ReLU is zero, this zero is multiplied during backpropagation causing the gradient to vanish [Hanin, 2018]. If a neuron is affected by vanishing gradients, this causes the neuron to potentially die. This thesis explores a possible solution to this dilemma.

A ReLU or neuron can be classified as dead when the neuron outputs a signal of zero for every input passed into the network. Research done by deep learning community members Isac Arnekvist and Lu Lu explains that the underlying issue of the dead ReLU problem is not fully understood. However, there have been varying methods used to alleviate it [Arnekvist et al., 2020] [Lu et al., 2020].

2.2 Previous Methods For Dead ReLUs

There are several methods used to combat the dead ReLU problem. The traditional solution uses other forms of the ReLU activation function, such as LeakyReLU or the Exponential Linear Unit (ELU). These approaches add a slight invariant to the ReLU to try and mediate vanishing gradients. For example, in the LeakyReLU, instead of allowing a signal of zero to be outputted by a neuron when the raw NET is negative, the LeakyReLU has a small sloped linear function. Specifically, the LeakyReLU function can be represented as $f(x) = max(\alpha x, x)$ where α is a small constant, usually 0.01. The LeakyReLU method has been shown to help, and in some research, small twists to the LeakyReLU have been used, such as the PReLU, which is parameterized on a neuron-specific basis. A visual representation of these activation functions can be seen in Figure 2.1. However, the results when using these activation functions can be inconsistent. [He et al., 2015b]. Another activation function used is ELU, shown in Figure 2.2. Clevert, one of the researchers that introduced ELU, describes this activation function to have improved learning characteristics compared to activation functions like LeakyReLU [Clevert et al., 2016]. These improvements are due to ELU allowing negative values, which pushes the mean unit activation closer to zero [Clevert et al., 2016]. However, this method can have some disadvantages, specifically, it can saturate for larger negative values [Clevert et al., 2016].

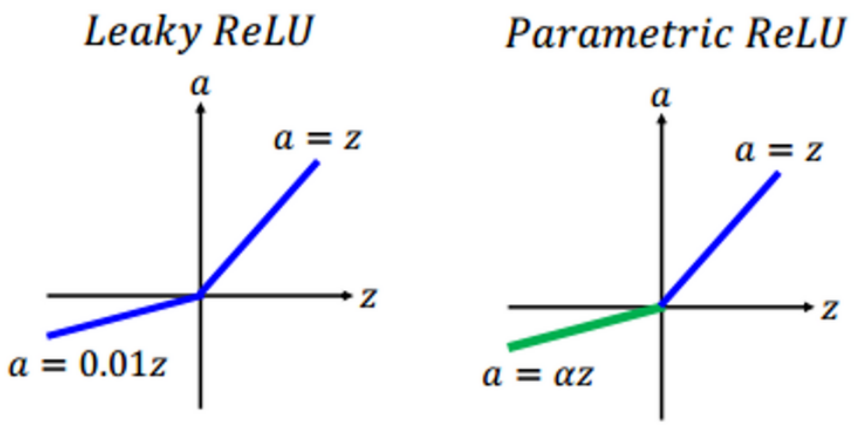


Figure 2.1: The Leaky ReLU activation function and PReLU activation function. [Google, 2017]

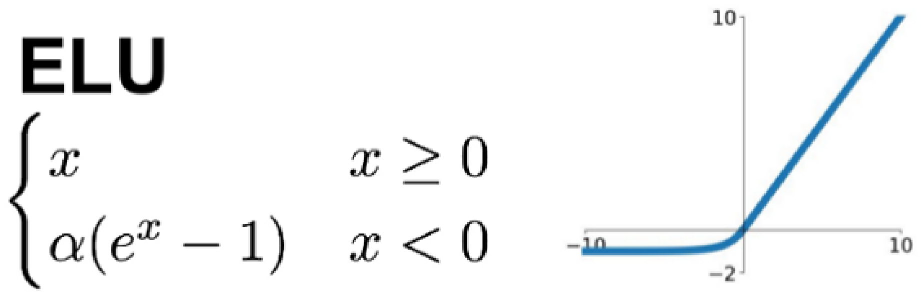


Figure 2.2: The ELU activation function. [Google, 2017]

Another possible solution explored is target normalization. Research in this area considered looking at the effect of target normalization and momentum on dying ReLUs [Arnekvist et al., 2020]. Moreover, this research was able to show empirically that the dead ReLU problem can be aggravated for deeper models and that ReLUs die when the target variance approaches zero [Arnekvist et al., 2020]. These solutions take away from the core of what makes the ReLU activation function so powerful, because it changes the function entirely or how it operates. This thesis explores a solution that does not invalidate the ReLUs advantages while maintaining simplicity.

2.3 Residual Connections in Deep Learning

Residual connections in neural networks have been utilized in many areas. The most notable being ResNet, a deep residual learning neural network developed to improve image recognition on complex datasets such as ImageNet and COCO while also reducing complexity [He et al., 2015a]. ResNet when evaluated on these complex datasets improved recognition drastically; for COCO a 28% relative improvement was achieved [He et al., 2015a]. This improvement was shown to be the result of adding residual connections inside of a deep neural network (152 layers). The creators of ResNet explain that the residual connections allows the learning of residual functions rather than unreferenced functions [He et al., 2015a].

Figure 2.3 shows an abstracted view of a residual connection at work. In the Figure, x is a given input to a layer. x is then both copied and passed through the regular layers. At the end of the layer, the original x and the layer processed x are pointwise added together and continued on to the rest of the network.

Residual connections are used in deep learning applications like ResNet for its advantage in gradient flow [Santhanam and Davis, 2020]. Specifically, ResNet creators explain that these connections improve gradient flow throughout the network by adding paths devoid of nonlinearities [He et al., 2015a]. These paths improved recognition results of known complex datasets and improved convergence of neural networks better than previous methods.

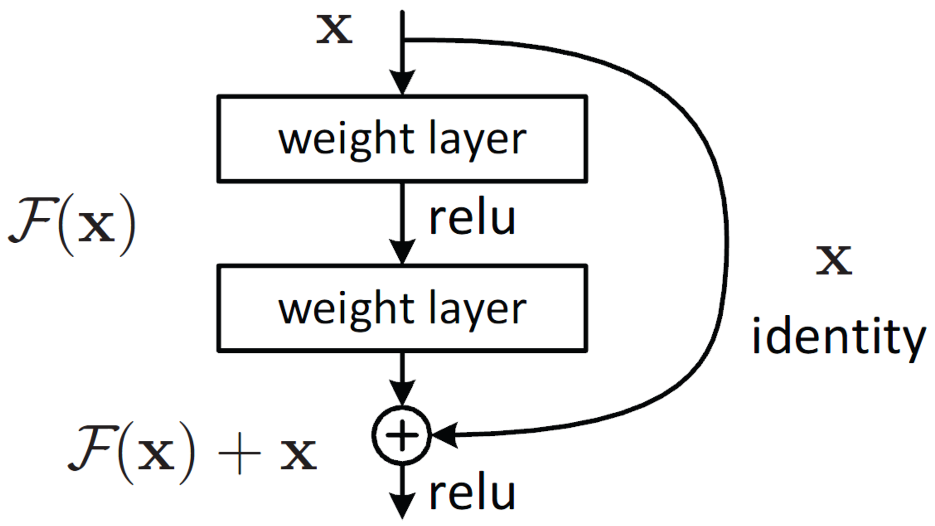


Figure 2.3: An abstracted view of a residual connection from one layer to another, inside the ResNet residual neural network. [He et al., 2015a]

By adding in residual connections, it can be empirically shown these connections can be used to improve certain deep learning architectures. Moreover, improvement of gradient flow into a network utilizing an activation function that is known to have problems with vanishing gradients could result in an improved network.

Chapter 3

Methodology

Our approach required building special neural network architectures by hand. This approach lead to the use of the PyTorch framework [PyTorch, 2021]. This approach allowed us to see into the heart of a network and observe dead neurons caused by vanishing gradients. It ensured we had control over every aspect of the network, and when we made a subtle change, we knew what was affected.

We developed two classes in the PyTorch framework, one to construct neural networks with non-residual connections, and one with residual connections.

3.1 Constructing the Dataset

To further remove abstraction, a dataset was needed that could be easily understood. A paper by Lu, presents a dataset based on the representation of the absolute value function ($f(x) = |x|$). When used with a ReLU activation function architecture, the resulting network probabilistically approaches a constant function [Lu et al., 2020].

The dataset for these experiments was constructed using NumPy, a library consisting of multi-dimensional array objects and a collection of routines for processing those arrays [NumPy, 2021]. Specifically, a random uniform distribution of values between -1 and 1 was chosen with a sample size of 20. The data can be seen in Table 3.1.

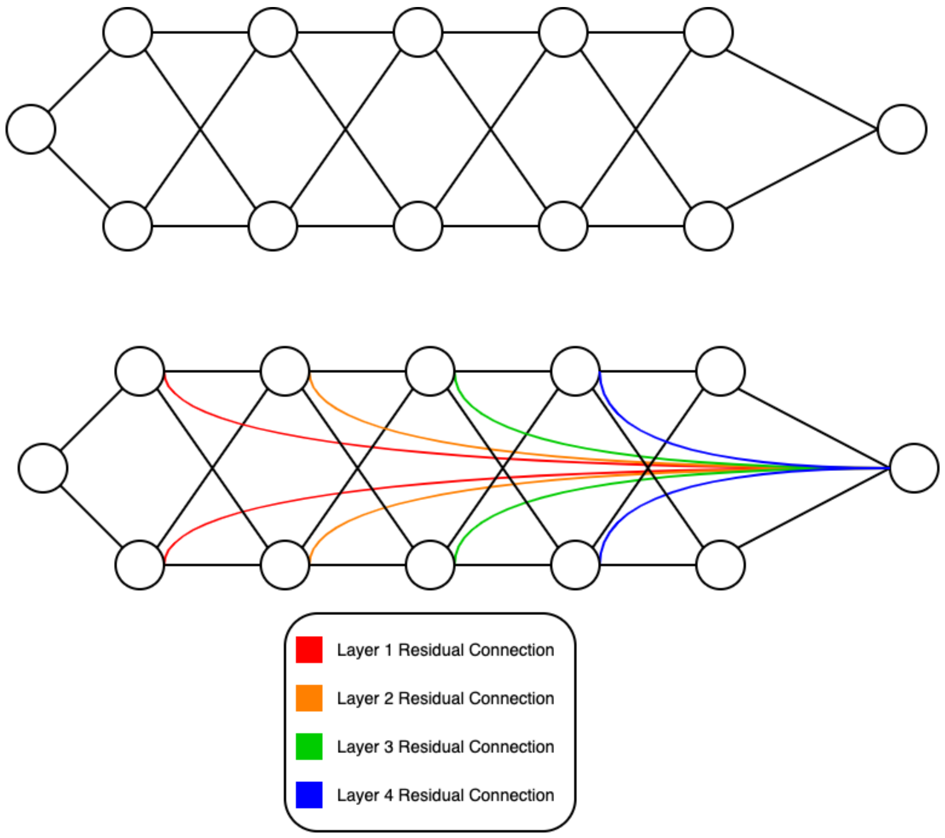
Approach Justification As stated earlier, the justification for this dataset was its simplicity. We wanted the attention of the experiments to be based on the architectures of the neural networks, not the dataset.

Table 3.1: The sample data used for the neural networks. Where X is the input and T is the targeted output.

X	T
-0.2509	0.2509
0.9014	0.9014
0.4639	0.4639
0.1973	0.1973
-0.6879	0.6879
-0.6880	0.6880
-0.8838	0.8838
0.7323	0.7323
0.2022	0.2022
0.4161	0.4161
-0.9588	0.9588
0.9398	0.9398
0.6648	0.6648
-0.5753	0.5753
-0.6363	0.6363
-0.6331	0.6331
-0.3915	0.3915
0.0495	0.0495
-0.1361	0.1361
-0.417	0.4175

3.2 Creating Neural Networks With PyTorch

With the dataset constructed, the architectures were created. One architecture would build a neural network with standard connections, while another would have residual connections. However, instead of residual connections that connect individual hidden layers together such as ResNet, the residual connections would be connected from each hidden layer directly to the output of the network. We coin the term "late residual connections" as it is a modification to traditional residual connections due to the connection directly to the output of a network. A visualization of this can be seen in Figure 3.1.



🟥 Layer 1 Residual Connection
🟧 Layer 2 Residual Connection
🟩 Layer 3 Residual Connection
🟦 Layer 4 Residual Connection

Figure 3.1: A visual representation of a non-residual connections versus late residual connections. Each color represents a hidden layers residual connection directly to the output

The non-residual class was programmed in classic PyTorch convention. Inheriting from PyTorch's neural network module, it constructs neural networks of varying depths made up of linear layers and ReLU activation function layers. This architecture could make any two wide depth network based on the size of a list passed to it containing the number of hidden neurons at each hidden layer. Additionally, the user could specify either Adam or SGD to optimize the network. Lastly, the architectures had a unique forward pass to capture the outputs of each layer of the networks. Other than this capture method, the forward pass was standard for iterative and batch training.

The late residual connection class varied from the non-residual architecture in how the neural networks were created and the forward pass of the networks. The construction of the late residual connection neural networks varied slightly in that for every hidden layer in the network being constructed, a PyTorch Identity would also be constructed. In other words, for every hidden layer in the network, there was a twin PyTorch Identity. This ensures the standard connection kept flowing through the heart of the network and that the PyTorch Identity is able to copy and save the outputs from these hidden layers. Next, the output layer of the network was changed slightly. Rather than the standard two inputs into this layer like the non-residual neural network, it now had $2n$ connection feeding into it where n is the depth of the network. Again, Figure 3.1 highlights this difference visually. The last change was to the forward pass. Here, instead of traditionally flowing through each layer of the network, for every signal coming out of a hidden layer, that signal would be saved by the PyTorch Identity and feed forward to the next layer in the neural network. Every signal that the PyTorch Identity saved, was stored in a list. Once the forward pass reached the output layer, the standard input at the end of the network was concatenated with all of the PyTorch Identity outputs, thus creating a late residual connection from the output layer to every hidden layer into the network. Along with these connections, PyTorch's dynamic graph in the background could infer the connections and construct a graph to train more efficiently.

Approach Justification Frameworks such as Tensorflow were not used here, as we wanted to see everything happening, in other words, pull the curtain back and remove as much abstraction as possible. By doing this, we could see what changes would happen throughout the network, even if a subtle change occurred. Specifically, we wanted to see the weights, bias, and output signal of the neurons.

3.3 Detecting Dead Neurons

The last tool needed for the experiments was a way to detect dead neurons in a network. As mentioned in the Review of Literature section, a neuron can be classified as dead when, for any given input to a network, the output of a neuron is always zero [Lu et al., 2020]. With this definition of a dead neuron, the neurons for each network could be evaluated. After training a network, the outputs of the neurons would be looked at for all of the input data. If a neuron was outputting only zero for all of these inputs, it was flagged for further evaluation and considered dead if it met the requirements.

The experiments had seven different network configurations. These networks all had one input, one output, and hidden layers of width two; the only variation was their depth in hidden layers. These network architectures can be read like a list of hidden layers. For example, the first network architecture is comprised of two hidden layers with two neurons in each layer, for short $[2, 2]$ or $[2\ for\_\ in\ range(2)]$ in terms of Python. All the architectures for this experiment had two wide hidden layers but varying depths, with a total of seven depths at 2, 5, 10, 15, 20, 25, and 30. A subset of these architectures can be seen visually in Figure 3.2.

Approach Justification These network architectures were selected based on the initial architecture described in Lu's paper. They were expanded to thirty layers deep to see the effect of the dying ReLU as a network becomes deeper.

3.4 Hyperparameters

Another factor that can cause gradients to vanish with the ReLU activation function is when a learning rate is set too high [Leung, 2021]. A high learning rate can kill a neuron because the ReLU activation function does not squash the raw NET like the Sigmoid or TanH activation functions do, thus causing the weight update during backpropagation to become highly negative in some cases and highly positive in other cases. With high learning rates in mind, three learning rates were considered 0.01, 0.001, and 0.1. As for epochs, 3000 was chosen to train this network. Initial experiments with this dataset with both the non-residual and late residual connection networks showed that the search space may have a local optimum or valley. These results lend further creditability to Lu's implication that a local optimum or valley exists for this dataset [Lu et al., 2020]. The existence of this local optimum indicates that regardless of the number of epochs, a network would likely not fully converge if, during training, the local optimum was encountered. Alternatively, the results may indicate a valley in the search space in which an optimizer may oscillate in-between and not reach an optima. To circumvent these issues, ten iterations were performed for each architecture at each learning rate to see the total number of successful convergence.

Approach Justification The learning rates chosen were to highlight the difference between of the non-residual connection and the late residual connections. The number of epochs was chosen to give the network time to converge. Furthermore, it was also chosen to highlight the changes to the loss on a learning rate basis.

3.5 Optimizers

In addition to multiple learning rates two optimizers were used for training. For these networks the optimizers were Adam and stochastic gradient descent (SGD).

Approach Justification These optimizer were chosen for a few specific reasons. Adam was chosen as it is a widely used optimizer in machine learning and deep learning applications. Adam is a common choice due to its adaptive learning rate behavior and ability to converge faster than most optimizers [Kingma and Ba, 2017]. However, due to this behavior, the amount of dead neurons could rise if the learning rate becomes too high. As for SGD, it was chosen for its simplicity. Having SGD move at the same constant learning rate could provide real insight into the correlation between all of these moving parts [Bottou, 2012].

3.6 Training Styles

Along with higher learning rates and optimizers, we also wanted to investigate the effects of iterative and batch training.

Approach Justification Iterative and batch training were chosen as they directly impact the gradients. For iterative training, a neural network is fed one input at a time. For this dataset, the values randomly switch between negative and positive. Due to this, the loss for a network utilizing iterative training could spike. For example, if the networks kept switching between negative and positive, during backpropagation the overall loss at a given epoch could spike from the previous epoch. This spike could cause a considerable shift in the weight for a neuron, thus killing it. In contrast, the batch training style is fed multiple inputs at a time, thus having a single gradient for the entire training set. Furthermore, batch training causes a less drastic change to the weights, potentially lessening the number of dead neurons.

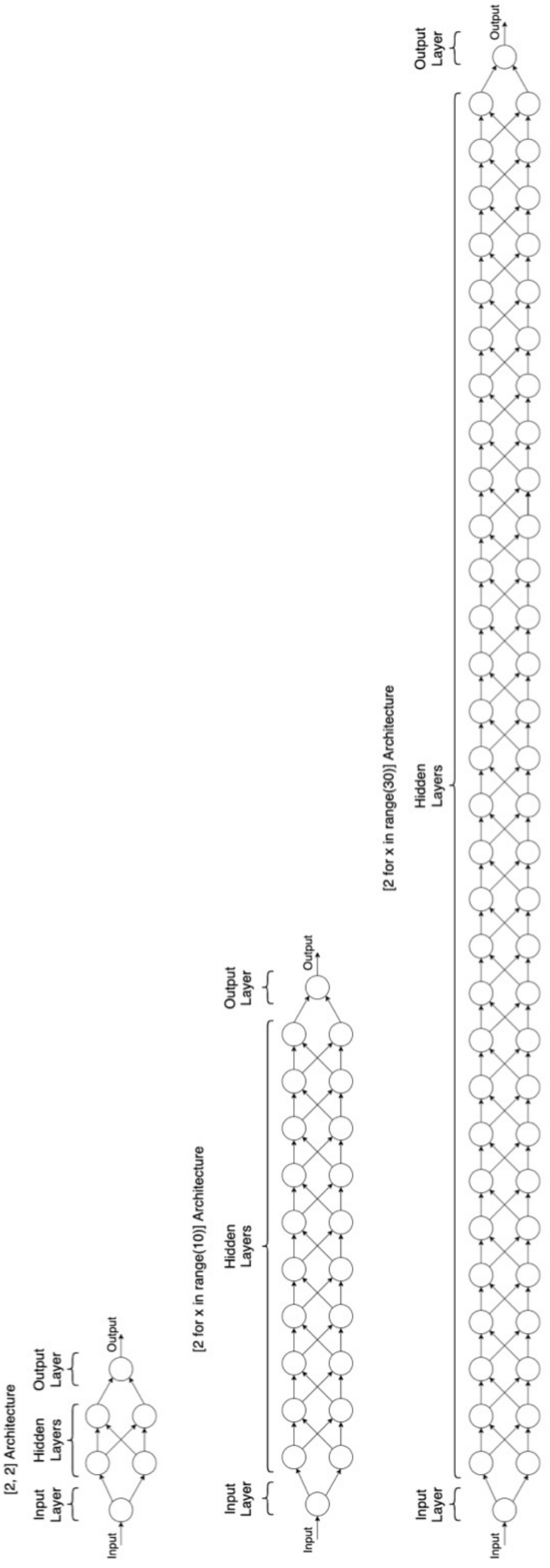


Figure 3.2: A subset of the neural network architectures used for experiments.

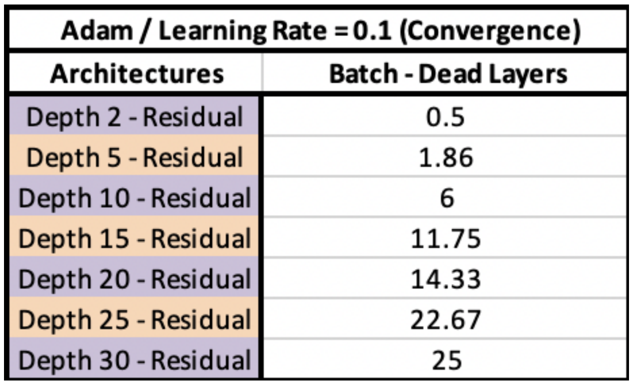
Chapter 4

Findings and Results

4.1 Dead Layers - Pruning

The experiments produced, an interesting result. For the networks using late residual connections when converged, occasionally the number of dead neuron found was over half of the total number of neurons in a network. For example, a network of depth 30 could have 35 out of 60 dead neurons. This also can be thought of as dead layers rather than dead neurons. For this example, 35 dead neurons are found or roughly 17.5 dead layers. Figure 4.1 shows the results of the most dead layers out of all the experiments at each learning rate. Figure 4.1 shows that as the depth increases so does the number of dead layers, however, these networks are still converging.

Initially, these results seemed incorrect as a network cannot learn if a layer is dead. However, a closer look at the architecture of these networks provided a clear reason behind this phenomena.



Adam / Learning Rate = 0.1 (Convergence)	
Architectures	Batch - Dead Layers
Depth 2 - Residual	0.5
Depth 5 - Residual	1.86
Depth 10 - Residual	6
Depth 15 - Residual	11.75
Depth 20 - Residual	14.33
Depth 25 - Residual	22.67
Depth 30 - Residual	25

Figure 4.1: A table showing a subset of dead layers for learning rate of 0.1. This table is taken out a larger table in Figure 4.15

Figure 4.2 shows the reason behind the convergence. Here, a subset of the networks are shown from Figure 4.1 with non-residual connections and with late residual connections. The red X's in the late residual networks are dead neurons. If two red X's are in the same layer, then that layer is dead.

By adding in the late residual connections, the computation of the network goes directly to the output. By doing so, the network is "short circuited". This short circuit can be thought of as the network being pruned and thus creates a network with fewer layers. These smaller networks with varying connections is what allows convergence to occur with the dead layers present.

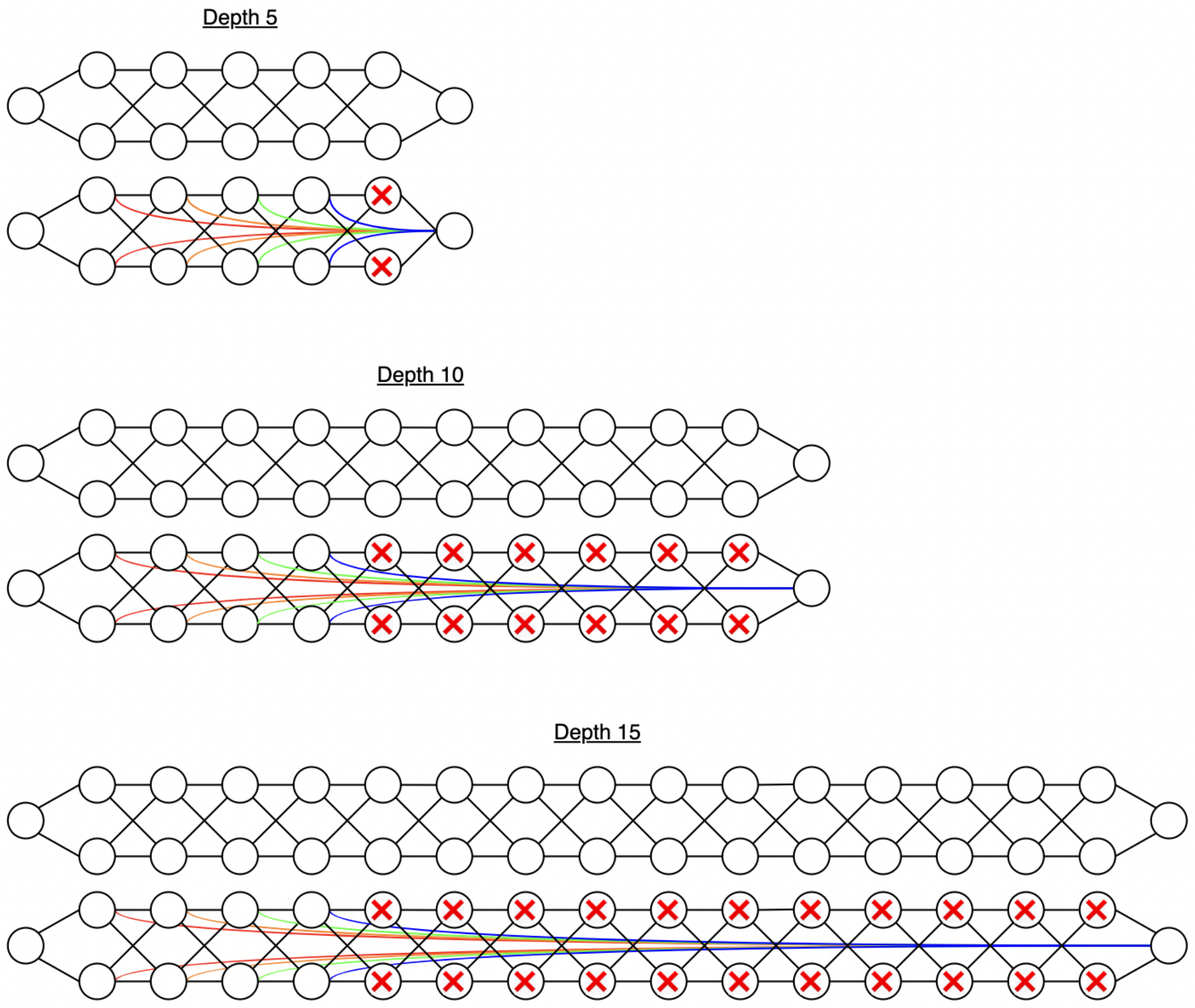


Figure 4.2: A subset of Figure 4.1 networks with dead layers along side the traditional non-residual connection networks. Each neuron with a red X indicates a dead neuron. If a layer has two red X's, those residual connection are not shown as it is not longer contributing to the network. It should be noted that this Figure depicts the worst case scenario for dead layers in these networks and that the networks all having four alive layers is a coincidence.

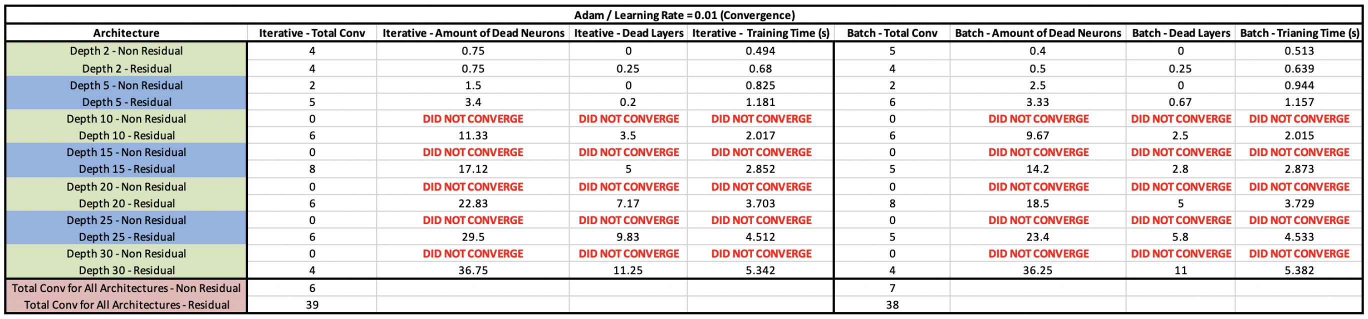
4.2 Convergence

As stated in the Methodology section, initial experiments and evidence by Lu's paper suggest a local optimum or valley exists in this dataset's search space. This local optimum or valley's correlation with dead neurons are investigated specifically at varying learning rates (0.01, 0.001, and 0.1). For each learning rate, both the Adam and SGD optimizers are analyzed. For simplicity, only a subset of the results are looked at for each learning rate. The outcomes examined here provide critical insights into the late residual connections influence.

4.2.1 Learning Rate 0.01

Adam

Figure 4.3 shows the full results for the Adam optimizer at a learning rate of 0.01 for iterative and batch training. The Figure can be broken up into different subsections. First, the left inner table shows the iterative results, and the inner right shows the batch results. On the far left, the architectures are listed from shallowest to deepest. Each green and blue color, group together a specific depth architecture for both the non-residual and the late residual connections. This description is done to easily compare the connections side by side for a given architecture. At the bottom of the table, in red, is the sum of convergence iterations over all architectures for both iterative and batch training.



Adam / Learning Rate = 0.01 (Convergence)								
Architecture	Iterative - Total Conv	Iterative - Amount of Dead Neurons	Iteative - Dead Layers	Iterative - Training Time (s)	Batch - Total Conv	Batch - Amount of Dead Neurons	Batch - Dead Layers	Batch - Trianing Time (s)
Depth 2 - Non Residual	4	0.75	0	0.494	5	0.4	0	0.513
Depth 2 - Residual	4	0.75	0.25	0.68	4	0.5	0.25	0.639
Depth 5 - Non Residual	2	1.5	0	0.825	2	2.5	0	0.944
Depth 5 - Residual	5	3.4	0.2	1.181	6	3.33	0.67	1.157
Depth 10 - Non Residual	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE
Depth 10 - Residual	6	11.33	3.5	2.017	6	9.67	2.5	2.015
Depth 15 - Non Residual	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE
Depth 15 - Residual	8	17.12	5	2.852	5	14.2	2.8	2.873
Depth 20 - Non Residual	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE
Depth 20 - Residual	6	22.83	7.17	3.703	8	18.5	5	3.729
Depth 25 - Non Residual	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE
Depth 25 - Residual	6	29.5	9.83	4.512	5	23.4	5.8	4.533
Depth 30 - Non Residual	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE
Depth 30 - Residual	4	36.75	11.25	5.342	4	36.25	11	5.382
Total Conv for All Architectures - Non Residual	6				7			
Total Conv for All Architectures - Residual	39				38			

Figure 4.3: Results of convergence with the Adam optimizer with a learning rate of 0.01. Where each green and blue architecture grouping shows a certain depth for both non-residual and late residual connections. The inner part of the table is broken up by iterative and batch training style. The bottom red grouping is the total number of convergence for all iterations of each network architecture. If a network did not converge for all ten iterations, each cell is labeled DID NOT CONVERGE.

A pattern in this Figure can be seen right away with the iterative training. As the depth of the architecture increases, the neural networks with non-residual connections do not converge. This is denoted with the red font title "DID NOT CONVERGE" in Figure 4.3. When an architecture does not converge, that network fails to converge at any given random weight initialization for all ten iterations. This can be seen more clearly in Figure 4.4. On the left side of the figure, we see spiking in the loss function graph for the first 100 epochs for iterative training. As mentioned in the Methodology section, this spiking is caused by the dataset inputs switching between negative and positive values. On the right side of the figure, we can see the green graph showing the target values. Notice that these values are not sorted, which again explains the spiking on the iterative side. In Figure 4.5, we have the exact same architecture as Figure 4.4, however, with late residual connections.

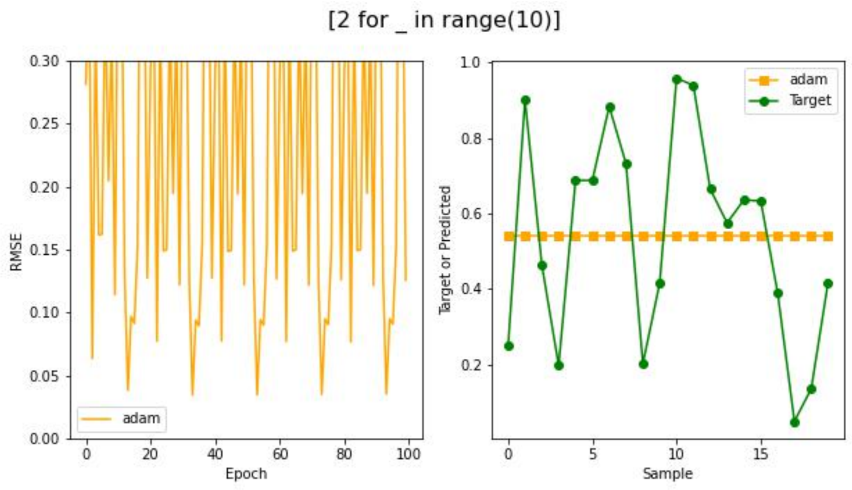


Figure 4.4: Plot of loss and prediction for non-residual network of depth 10 for iterative training at a learning rate of 0.01. The left side of the figure shows the loss for each epoch. In this case, the iterative training causes spiking of the loss. On the right, the networks predictions indicate the network became a constant function.

With the late residual connections added, the network could converge—specifically, six out of the ten iterations. These results imply that the network is now able to converge because of these late residual connections. Specifically, by directly connecting to the output, the gradient has more flow to a neuron and can update the weights accordingly. This effect causes the neurons not to die and ultimately allows for better learning. This pattern can be seen in Figure 4.3. The number of dead neurons per number of converging iterations is lower than non-residual connections in both the iterative and batch training. Lastly, at the bottom of Figure 4.3 in red, we can see the total number of iterations that converged for all architectures for both batch and iterative. Here, we again see that the addition of late residual connections drastically improves the convergence rate for these neural networks.

It can also be noted that the number of dead neurons for these networks typical are surpassing over half the amount of total neurons of the full architecture. As stated in the Dead Layer section, pruning is occurring with these layers.

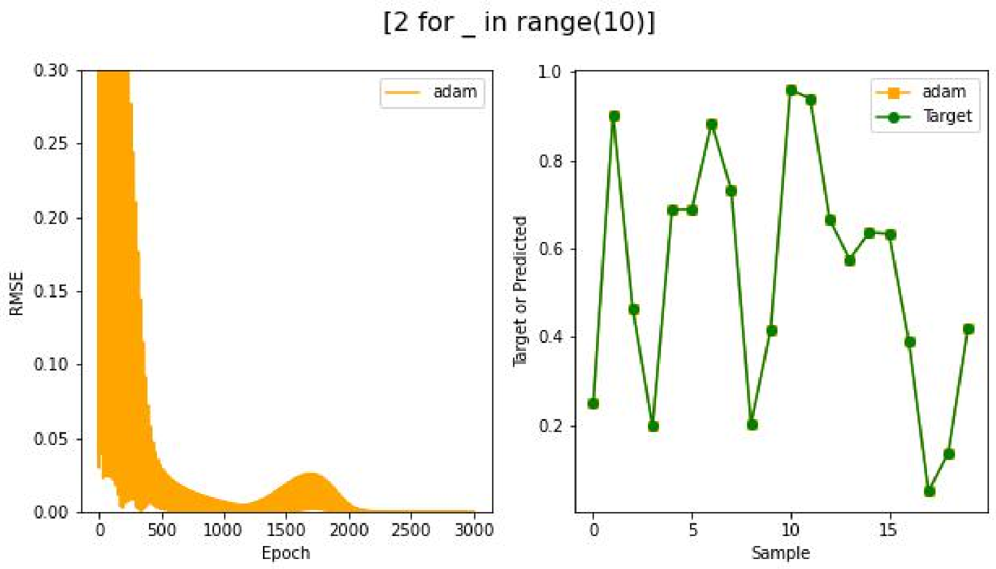
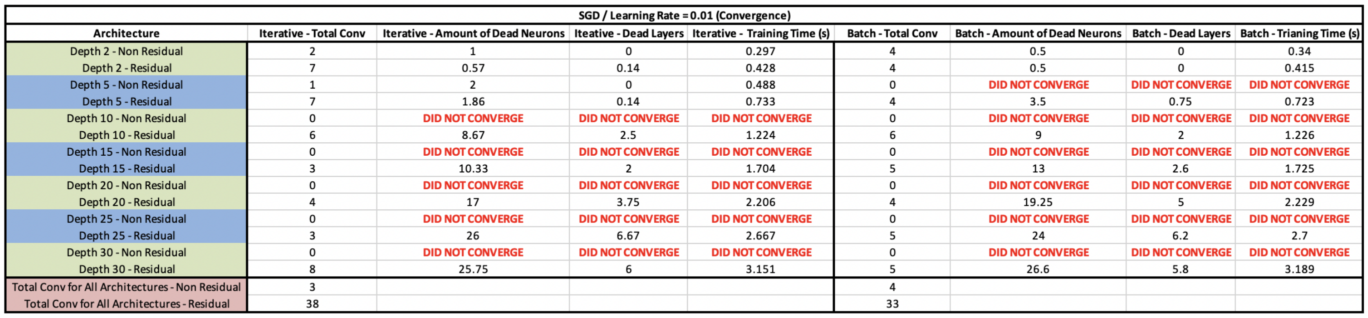


Figure 4.5: Plot of loss and prediction for late residual network of depth 10 for iterative training at a learning rate of 0.01. The left side of the figure shows the loss for iterative training style. The right side of the figure shows the prediction for the network. In this case, the network converges and its predicted values in orange are overlapping with the target values in green.

SGD

Next, we look at the SGD optimizer with the same learning rate and architectures. Likewise in Figure 4.3, the SGD Figure 4.6 follows the same pattern. Here, both iterative and batch training are split inside the table, with the depth of the architectures increasing on the far left.



SGD / Learning Rate = 0.01 (Convergence)								
Architecture	Iterative - Total Conv	Iterative - Amount of Dead Neurons	Iteative - Dead Layers	Iterative - Training Time (s)	Batch - Total Conv	Batch - Amount of Dead Neurons	Batch - Dead Layers	Batch - Trianing Time (s)
Depth 2 - Non Residual	2	1	0	0.297	4	0.5	0	0.34
Depth 2 - Residual	7	0.57	0.14	0.428	4	0.5	0	0.415
Depth 5 - Non Residual	1	2	0	0.488	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE
Depth 5 - Residual	7	1.86	0.14	0.733	4	3.5	0.75	0.723
Depth 10 - Non Residual	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE
Depth 10 - Residual	6	8.67	2.5	1.224	6	9	2	1.226
Depth 15 - Non Residual	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE
Depth 15 - Residual	3	10.33	2	1.704	5	13	2.6	1.725
Depth 20 - Non Residual	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE
Depth 20 - Residual	4	17	3.75	2.206	4	19.25	5	2.229
Depth 25 - Non Residual	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE
Depth 25 - Residual	3	26	6.67	2.667	5	24	6.2	2.7
Depth 30 - Non Residual	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE
Depth 30 - Residual	8	25.75	6	3.151	5	26.6	5.8	3.189
Total Conv for All Architectures - Non Residual	3				4			
Total Conv for All Architectures - Residual	38				33			

Figure 4.6: Results of convergence with the SGD optimizer with a learning rate of 0.01. Where each green and blue architecture grouping shows a certain depth for both non-residual and late residual connections. The inner part of the table is broken up by iterative and batch training style. The bottom red grouping is the total number of convergence for all iterations of each network architecture. If a network did not converge for all ten iterations, each cell is labeled DID NOT CONVERGE.

The iterative training in Adam had caused considerable spikes in the loss function. With the use of the SGD optimizer, the learning rate is constant during training. In addition to this, looking at the batch training could also yield different results. In Figure 4.7, we have a batch training style with a network architecture of depth five. As shown on the left, the network was not able to converge.

On the right side of the Figure 4.7, the network effectively became a constant function. This result could be due to the local optimum or valley in this search space and SGD getting stuck. However, investigating the late residual connection at this depth could give more insight.

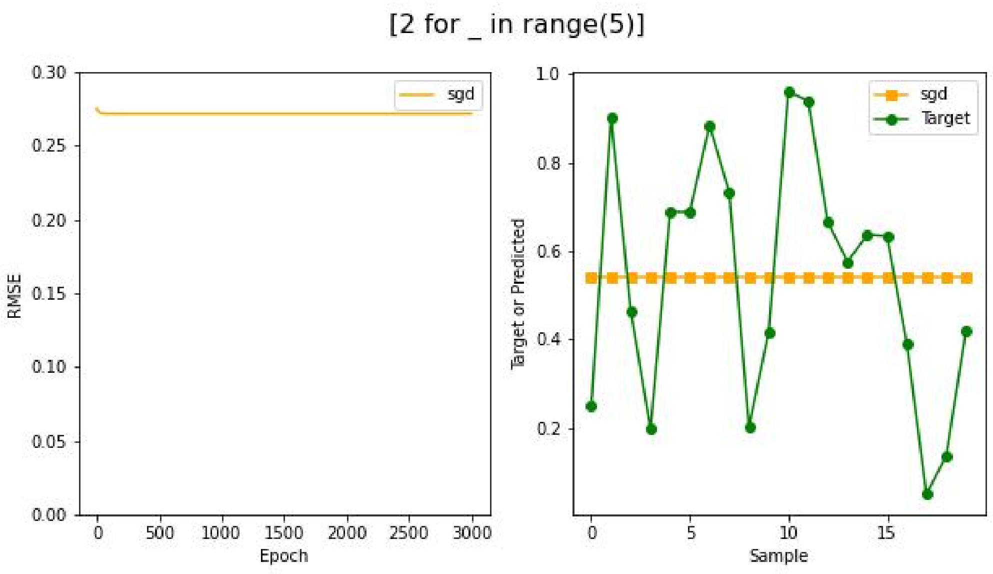


Figure 4.7: Plot of loss and prediciton for non-residual network of depth 10 for batch training at a learning rate of 0.01. The left side of the figure shows the loss for batch training style with SGD. Here, the networks loss never changed. The right side of the figure shows the result of no change in the loss, with no convergence for this network.

Figure 4.8 shows the same network as Figure 4.7, but with late residual connections. Just as in the Adam optimizer network with the addition of late residual connections, we have more convergence than non-residual networks. Using batch training, this network could converge after roughly 1000 epochs and does so gradually and with less spiking compared to iterative training. Part of the reason there is less spiking in this experiment can be explained by batch training and its averaging of gradients. In Figure 4.6, every architecture with late residual connection neural networks converges more often. Figure 4.6 highlights this for the architecture of depth thirty with iterative training. Here, none of the non-residual connections were able to converge. Meanwhile, eight out of the ten iterations were able to converge for the late residual connections. Again, like the Adam optimizer, more than likely adding the late residual connections allowed for more neurons to be to contribute to learning of the search space. In addition, pruning could be effecting the landscape as well.

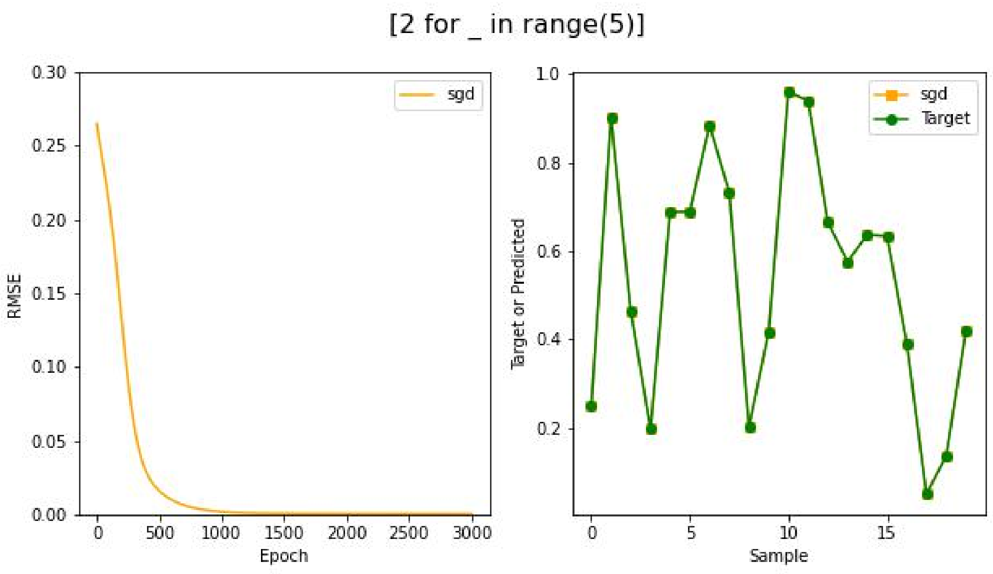


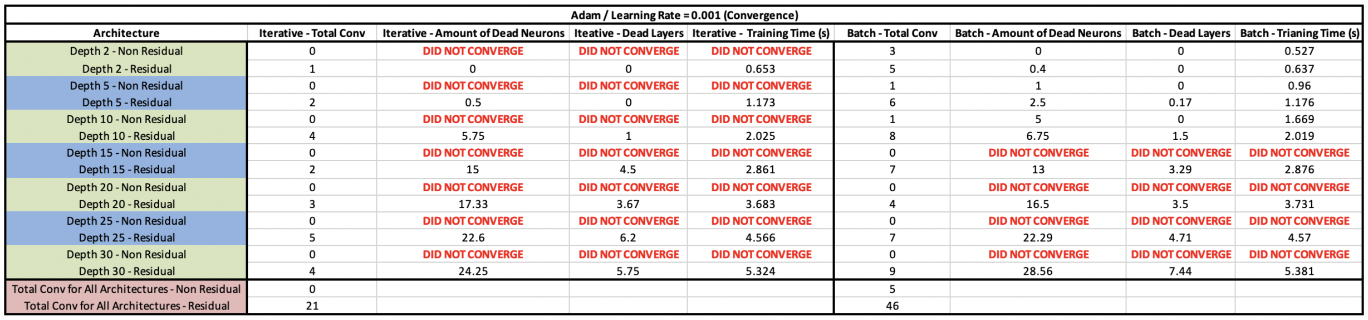
Figure 4.8: Plot of loss and prediction for late residual network of depth 10 for batch training at a learning rate of 0.01. The left side of the figure shows networks loss minimizing with the batch training style and late residual connections. The right side of the figure shows the networks predictions values in orange overlapping with the target values in green.

4.2.2 Learning Rate 0.001

We wanted to investigate a lower learning rate after seeing the correlation with the late residual connections at the 0.01 learning rate. A learning rate of 0.001 was chosen to further explore the local optimum or valley in the search space. Can the late residual connections keep neurons alive and allow the network to converge? Again, as in the previous subsection, the results are broken down by optimizers Adam and SGD.

Adam

Adam's results at the 0.001 learning rate can be shown in Figure 4.9. At first glance, there are interesting results, especially with the training styles. None of the non-residual connection networks were able to converge, while the late residual connections had at minimum one convergence out of ten iterations. The local optimum or valley in the search space may be the cause of no convergence for non-residual connections and low convergence for late residual connections. Moreover, another factor could be the iterative training causing the loss to not be able to minimize



Adam / Learning Rate = 0.001 (Convergence)								
Architecture	Iterative - Total Conv	Iterative - Amount of Dead Neurons	Iteative - Dead Layers	Iterative - Training Time (s)	Batch - Total Conv	Batch - Amount of Dead Neurons	Batch - Dead Layers	Batch - Trianing Time (s)
Depth 2 - Non Residual	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE	3	0	0	0.527
Depth 2 - Residual	1	0	0	0.653	5	0.4	0	0.637
Depth 5 - Non Residual	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE	1	1	0	0.96
Depth 5 - Residual	2	0.5	0	1.173	6	2.5	0.17	1.176
Depth 10 - Non Residual	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE	1	5	0	1.669
Depth 10 - Residual	4	5.75	1	2.025	8	6.75	1.5	2.019
Depth 15 - Non Residual	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE
Depth 15 - Residual	2	15	4.5	2.861	7	13	3.29	2.876
Depth 20 - Non Residual	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE
Depth 20 - Residual	3	17.33	3.67	3.683	4	16.5	3.5	3.731
Depth 25 - Non Residual	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE
Depth 25 - Residual	5	22.6	6.2	4.566	7	22.29	4.71	4.57
Depth 30 - Non Residual	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE
Depth 30 - Residual	4	24.25	5.75	5.324	9	28.56	7.44	5.381
Total Conv for All Architectures - Non Residual	0				5			
Total Conv for All Architectures - Residual	21				46			

Figure 4.9: Results of convergence with the Adam optimizer with a learning rate of 0.001. Where each green and blue architecture grouping shows a certain depth for both non-residual and late residual connections. The inner part of the table is broken up by iterative and batch training style. The bottom red grouping is the total number of convergence for all iterations of each network architecture. If a network did not converge for all ten iterations, each cell is labeled DID NOT CONVERGE.

due to the gradients flipping back and forth. With the side effect of iterative training, the batch training was investigated.

Figure 4.9 highlights a part of the story. Even at the deepest architecture of thirty layers and combined with the batch training, non-residual connections are not enough to converge. Typically, adding more neurons allows more degrees of freedom and thus enables the potential for more learning. However, even with the addition of more neurons convergence was not possible. The network's loss barely minimizes and consequently causes the network to become a constant function.

The other half of the story is shown in Figure 4.11. Here, the late residual connections are applied, and convergence is attained. Furthermore, we can see the loss function flows in textbook Adam fashion. The network was able to minimize the loss quickly and attain convergence.

Moving back to Figure 4.9, the whole picture starts to become clear. The neural networks are more than likely to converge with the combination of late residual connections, the Adam optimizer, and batch training. However, the question of the local optima or valley in the search space affecting the learning and convergence remains. Adam's learning rate can adaptively change during training, which could help the networks get out of the local optimum. Furthermore, Adam's accountability of variance could help move the network correctly out of the valley. To investigate

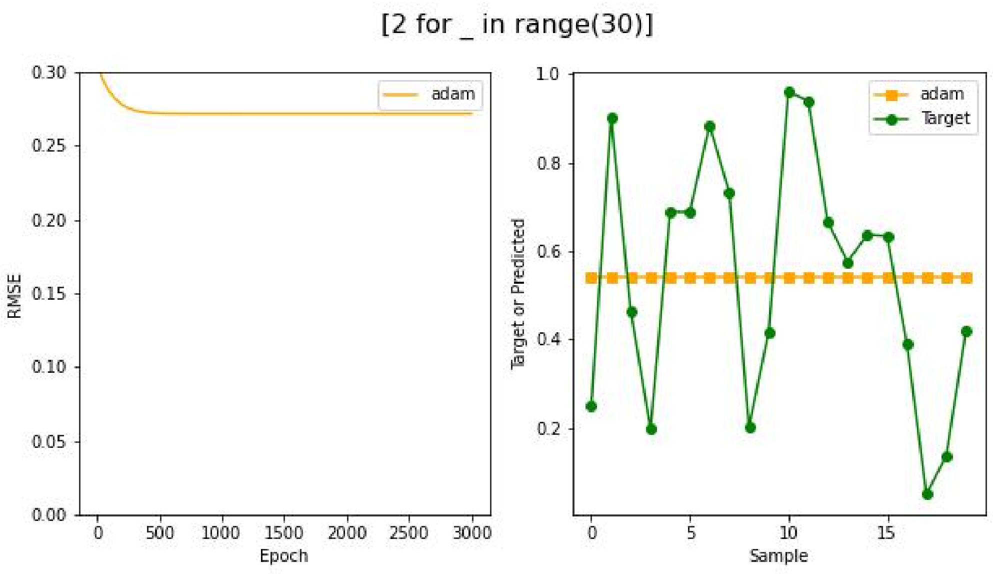


Figure 4.10: Plot of loss and prediction for non-residual network of depth 30 for batch training at a learning rate of 0.001. The left side of the figure shows the loss not minimizing for this network. The right side of the figure shows the result on the network not being able to minimize its loss with the network not converging.

this idea further, SGD was examined. Since SGD has a constant learning rate, one would suspect that the neural networks would not converge as well as the Adam networks.

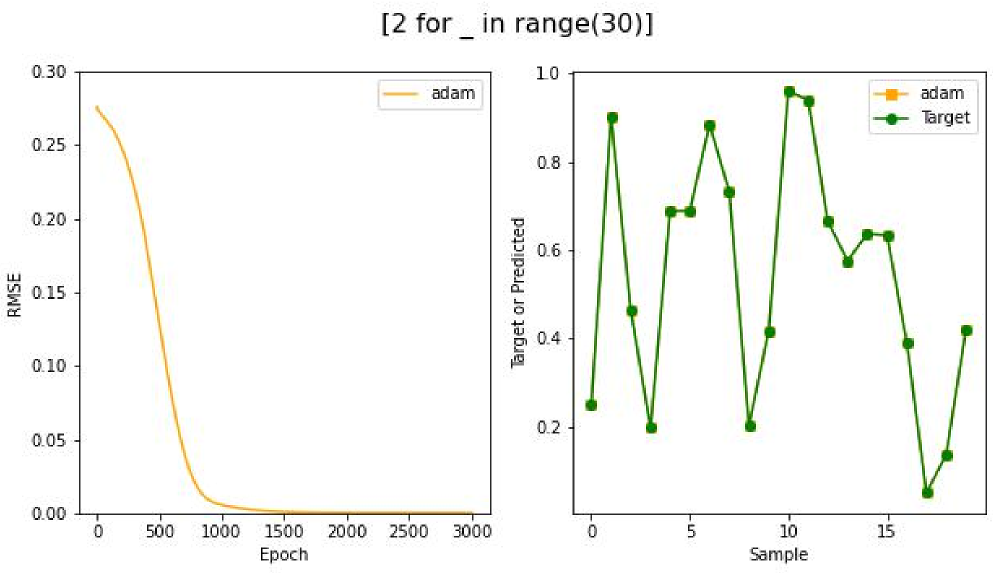
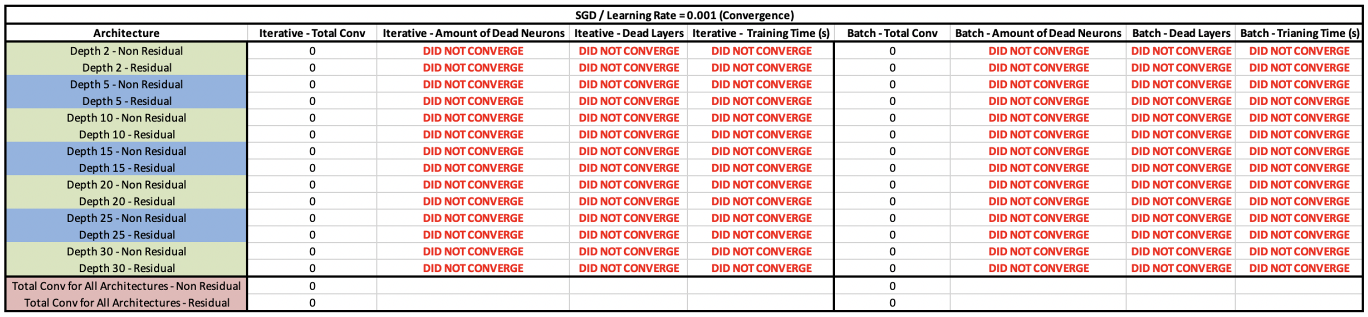


Figure 4.11: Plot of loss and prediction for late residual network of depth 30 for batch training at a learning rate of 0.001. The left side of the figure show the loss of the network minimizing with batch training and late residual connections. The right side of the figure shows the networks predicted values in orange overlapping with target values in green.

SGD

As shown in Figure 4.12, this suspicion appears to be true. Regardless of late residual connections, iterative versus batch training, or architecture depth, none of the neural networks could converge. A local optimum or valley is more than likely the cause of failed convergence. With SGD moving along at the same lower learning rate in the search space, it may not be able to get out of the optimum. Alternatively, the search space could also have a valley in which SGD is oscillating in-between. The only method that seemed to help mediate this was using the Adam optimizer. This mediation can be understandable as Adam can tune its learning rate carefully enough to get out of the local optimum and also accounts for variance to move out of a valley. The figures below show the SGD optimizer's failure to converge regardless of non-residual or late residual connections.



SGD / Learning Rate = 0.001 (Convergence)								
Architecture	Iterative - Total Conv	Iterative - Amount of Dead Neurons	Ieative - Dead Layers	Iterative - Training Time (s)	Batch - Total Conv	Batch - Amount of Dead Neurons	Batch - Dead Layers	Batch - Trianing Time (s)
Depth 2 - Non Residual	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE
Depth 2 - Residual	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE
Depth 5 - Non Residual	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE
Depth 5 - Residual	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE
Depth 10 - Non Residual	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE
Depth 10 - Residual	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE
Depth 15 - Non Residual	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE
Depth 15 - Residual	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE
Depth 20 - Non Residual	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE
Depth 20 - Residual	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE
Depth 25 - Non Residual	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE
Depth 25 - Residual	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE
Depth 30 - Non Residual	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE
Depth 30 - Residual	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE
Total Conv for All Architectures - Non Residual	0				0			
Total Conv for All Architectures - Residual	0				0			

Figure 4.12: Results of convergence with the SGD optimizer with a learning rate of 0.001. Where each green and blue architecture grouping shows a certain depth for both non-residual and late residual connections. The inner part of the table is broken up by iterative and batch training style. The bottom red grouping is the total number of convergence for all iterations of each network architecture. If a network did not converge for all ten iterations, each cell is labeled DID NOT CONVERGE.

Starting with the non residual connections in Figure 4.13, the potential of the local optimum or valley problem can be seen. Again, here the network cannot minimize its loss function and ultimately causes the network to become a constant function.

Even with the added late residual connections, the networks are still not able to fully converge, as shown in Figure 4.14. However, in Figure 4.14, it appears that the loss might have been fully minimized if given more time, since the loss is still decreasing during the last epoch. Moreover, this can be seen on the right of the Figure with the predictions of the network. The accuracy is not

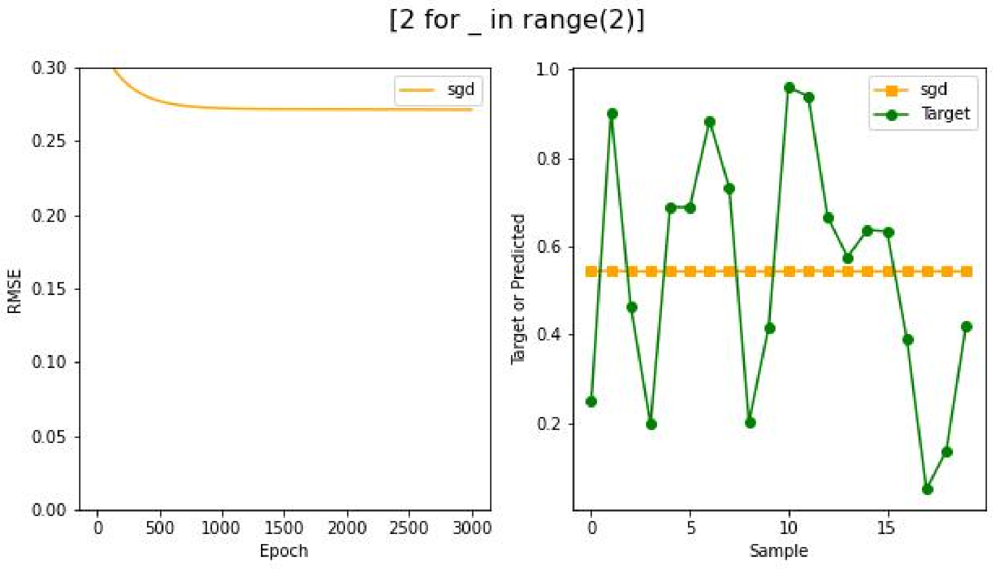


Figure 4.13: Plot of loss for non-residual network of depth 2 for batch training at a learning rate of 0.001. The left side of the figure shows the networks loss not minimizing. The right side of the figure shows the network becoming a constant function from not minimizing the loss.

the best but is still better than non-residual networks. Nonetheless, the late residual connections could still not converge given the same hyperparameters as the other neural networks.

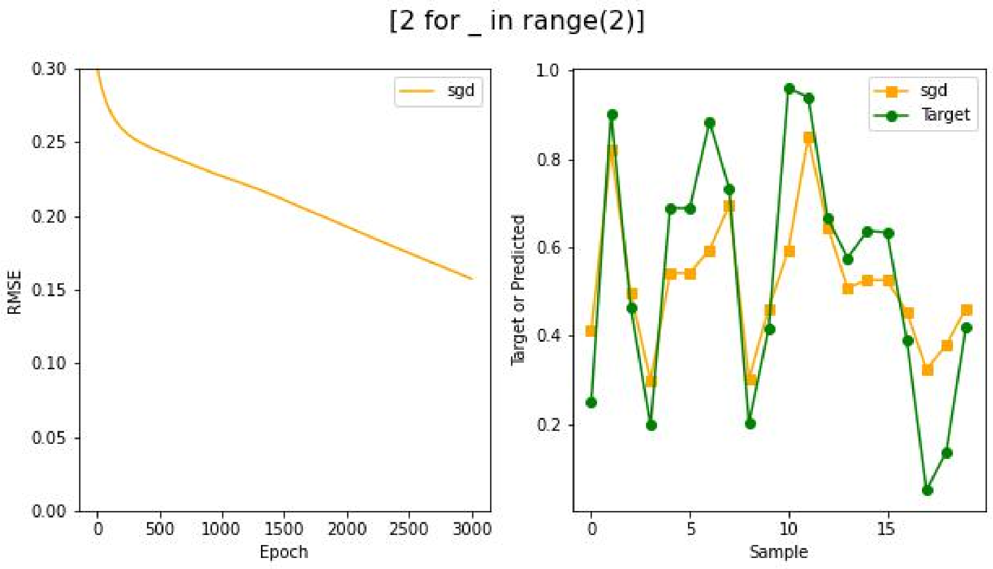


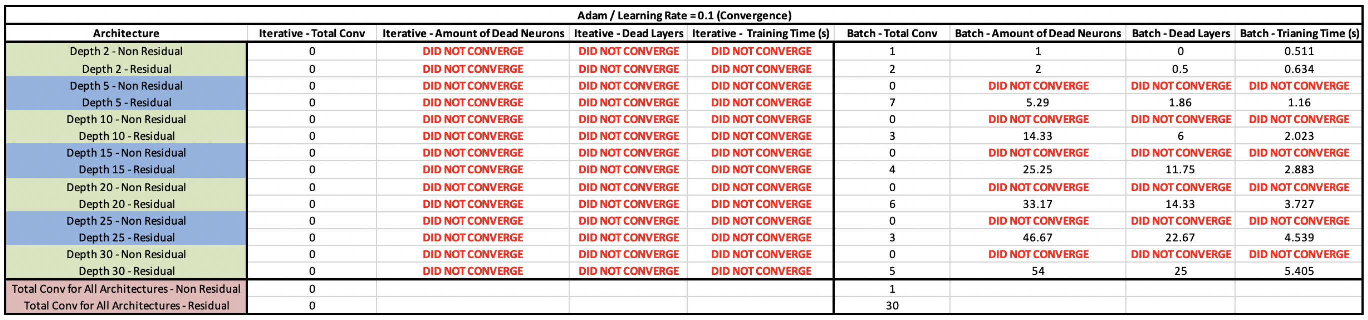
Figure 4.14: Plot of loss and prediction for late residual network of depth 2 for batch training at a learning rate of 0.001. The left side of the figure shows the networks loss slowly starting minimize but fails to fully do so in the number of epochs allotted. The right side of figure shows the networks predicted values in which it has not full converged but is close to doing so.

4.2.3 Learning Rate 0.1

Due to the results of the previous learning rate implying the issue with a potential local optimum or valley in the search space, an experiment at a higher learning rate was performed specifically at 0.1. As stated earlier in the Methodology section, it is known that a higher learning rate can cause more dead neurons with the ReLU activation function. However, it is also known that a higher learning rate could help a network move out of a local optimum but also are susceptible to oscillations in the loss. Could training with a higher learning rate help convergence despite the side effects?

Adam

The Adam results at this learning rate in are shown in Figure 4.15. Not surprisingly, we see that all networks, regardless of non-residual or late residual connections, fail to converge for iterative training. This result could be explained because the learning rate is now exceptionally high. The spiking pattern shown with this training could cause the networks not to converge due to too much flipping of the gradient for this dataset. However, this is not the case for the batch training.



Adam / Learning Rate = 0.1 (Convergence)								
Architecture	Iterative - Total Conv	Iterative - Amount of Dead Neurons	Iteative - Dead Layers	Iterative - Training Time (s)	Batch - Total Conv	Batch - Amount of Dead Neurons	Batch - Dead Layers	Batch - Trianing Time (s)
Depth 2 - Non Residual	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE	1	1	0	0.511
Depth 2 - Residual	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE	2	2	0.5	0.634
Depth 5 - Non Residual	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE
Depth 5 - Residual	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE	7	5.29	1.86	1.16
Depth 10 - Non Residual	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE
Depth 10 - Residual	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE	3	14.33	6	2.023
Depth 15 - Non Residual	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE
Depth 15 - Residual	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE	4	25.25	11.75	2.883
Depth 20 - Non Residual	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE
Depth 20 - Residual	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE	6	33.17	14.33	3.727
Depth 25 - Non Residual	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE
Depth 25 - Residual	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE	3	46.67	22.67	4.539
Depth 30 - Non Residual	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE
Depth 30 - Residual	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE	5	54	25	5.405
Total Conv for All Architectures - Non Residual	0				1			
Total Conv for All Architectures - Residual	0				30			

Figure 4.15: Results of convergence with the Adam optimizer with a learning rate of 0.1. Where each green and blue architecture grouping shows a certain depth for both non-residual and late residual connections. The inner part of the table is broken up by iterative and batch training style. The bottom red grouping is the total number of convergence for all iterations of each network architecture. If a network did not converge for all ten iterations, each cell is labeled DID NOT CONVERGE.

Figure 4.16 shows the training results of the Adam optimizer with non-residual connections and a depth of 25. Once more, we see a similar story. The loss was not able to be minimized and caused the network to become a constant function. This constant function could be due to the learning rate affecting the neurons. To answer this question, we can examine Figure 4.17 results.

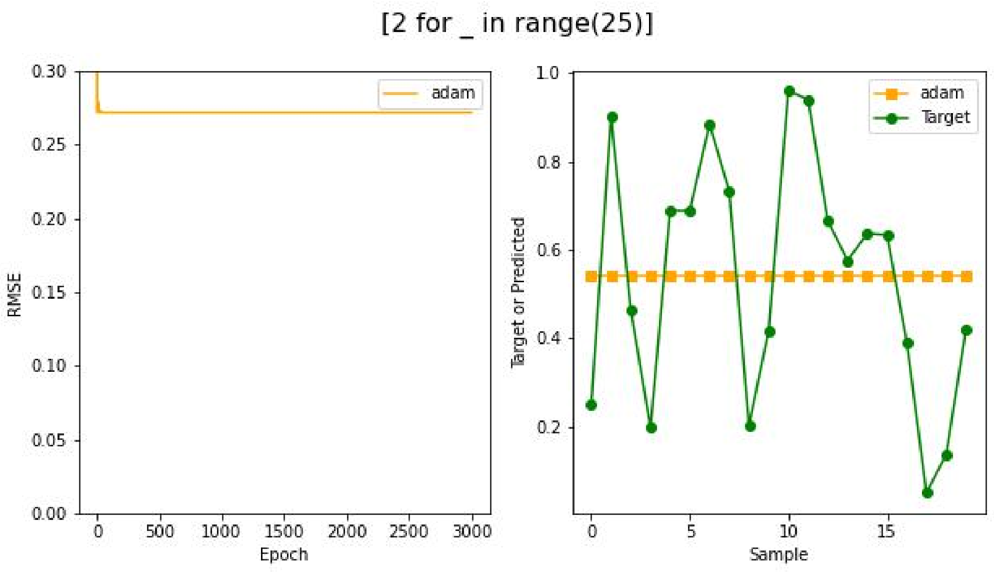


Figure 4.16: Plot of loss and prediction for non-residual network of depth 25 for batch training at a learning rate of 0.1. The left side of the figure shows the loss of the network not minimizing. The right side of the figure shows the network becoming a constant function.

Figure 4.17 shows the results of the added late residual connections for the same architecture. Even with a very high learning rate, the late residual connection neural network was able to converge. Moreover, it was able to converge quickly after roughly 100 epochs. The speed at which this network converged could be explained by the increased learning rate. However, to fully show these results, SGD needed to be investigated.

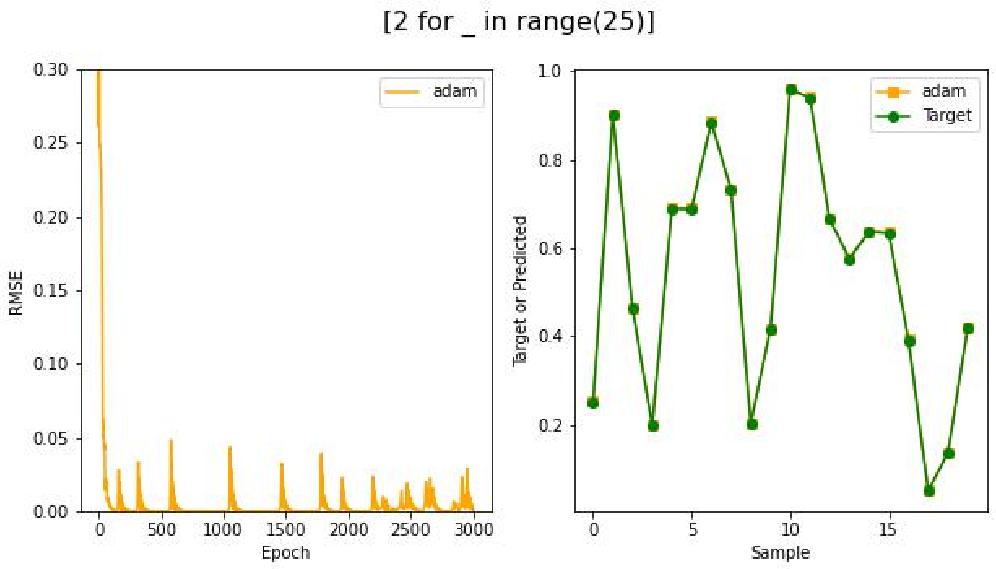
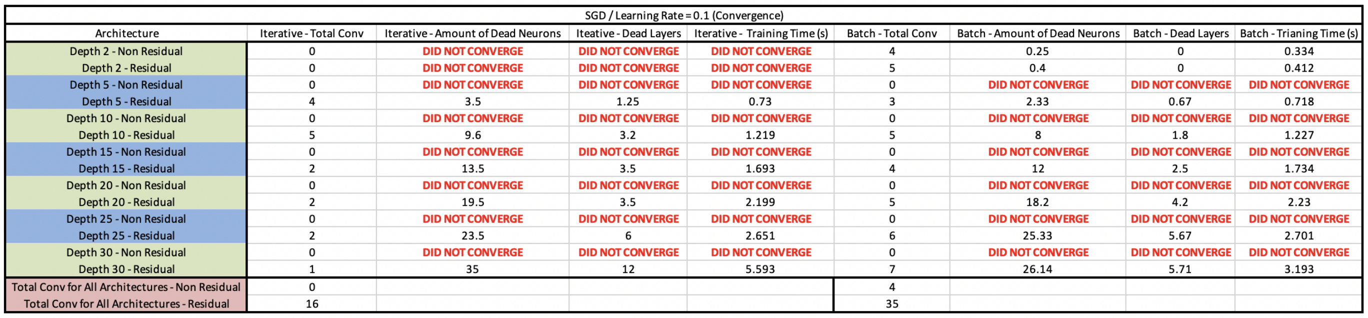


Figure 4.17: Plot of loss and prediciton for late residual network of depth 25 for batch training at a learning rate of 0.1. The left side of the figure shows the loss of the network minimizing. The right side of the figure shows the networks prediction values in orange overlapping with the target values in green.

SGD

As shown in the 0.001 learning rate section, all of the non-residual networks failed to converge while the late residual connection network had a low number of convergence. However, the learning rate was increased to 0.1 to clarify the connection between the potential local optimum or valley and dead neurons. Figure 4.18 shows the results. Here, in iterative and batch training, the late residual connections succeed. Moreover, they can converge at least once at all depths of the network. The figures below hammers this point home.



SGD / Learning Rate = 0.1 (Convergence)

Architecture	Iterative - Total Conv	Iterative - Amount of Dead Neurons	Iteative - Dead Layers	Iterative - Training Time (s)	Batch - Total Conv	Batch - Amount of Dead Neurons	Batch - Dead Layers	Batch - Trianing Time (s)
Depth 2 - Non Residual	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE	4	0.25	0	0.334
Depth 2 - Residual	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE	5	0.4	0	0.412
Depth 5 - Non Residual	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE
Depth 5 - Residual	4	3.5	1.25	0.73	3	2.33	0.67	0.718
Depth 10 - Non Residual	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE
Depth 10 - Residual	5	9.6	3.2	1.219	5	8	1.8	1.227
Depth 15 - Non Residual	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE
Depth 15 - Residual	2	13.5	3.5	1.693	4	12	2.5	1.734
Depth 20 - Non Residual	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE
Depth 20 - Residual	2	19.5	3.5	2.199	5	18.2	4.2	2.23
Depth 25 - Non Residual	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE
Depth 25 - Residual	2	23.5	6	2.651	6	25.33	5.67	2.701
Depth 30 - Non Residual	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE	0	DID NOT CONVERGE	DID NOT CONVERGE	DID NOT CONVERGE
Depth 30 - Residual	1	35	12	5.593	7	26.14	5.71	3.193
Total Conv for All Architectures - Non Residual	0				4			
Total Conv for All Architectures - Residual	16				35			

Figure 4.18: Results of convergence with the SGD optimizer with a learning rate of 0.1. Where each green and blue architecture grouping shows a certain depth for both non-residual and late residual connections. The inner part of the table is broken up by iterative and batch training style. The bottom red grouping is the total number of convergence for all iterations of each network architecture. If a network did not converge for all ten iterations, each cell is labeled DID NOT CONVERGE.

Figure 4.19 shows the SGD results for the non-residual connection at a depth of 25. As shown in all of the previous non-residual connection networks, the network is not able to converge. Again, this is a could be a side effect of the dead neurons not allowing the network to learn.

Finally, Figure 4.20 shows a network with depth 25. With the learning rate at 0.1, both the Adam optimizer and the SGD optimizer converged with the addition of late residual connections. This higher learning rate could be the reason behind why the networks break out of the potential local optimum. The lack of oscillation in the loss also supports this claim. Moreover, the late residual connections allowed the possibility to keep more neurons alive to contribute to the network's learning. These results ultimately show the key insight of improved convergence and potential reduced dead neurons by utilizing late residual connections and both optimizers.

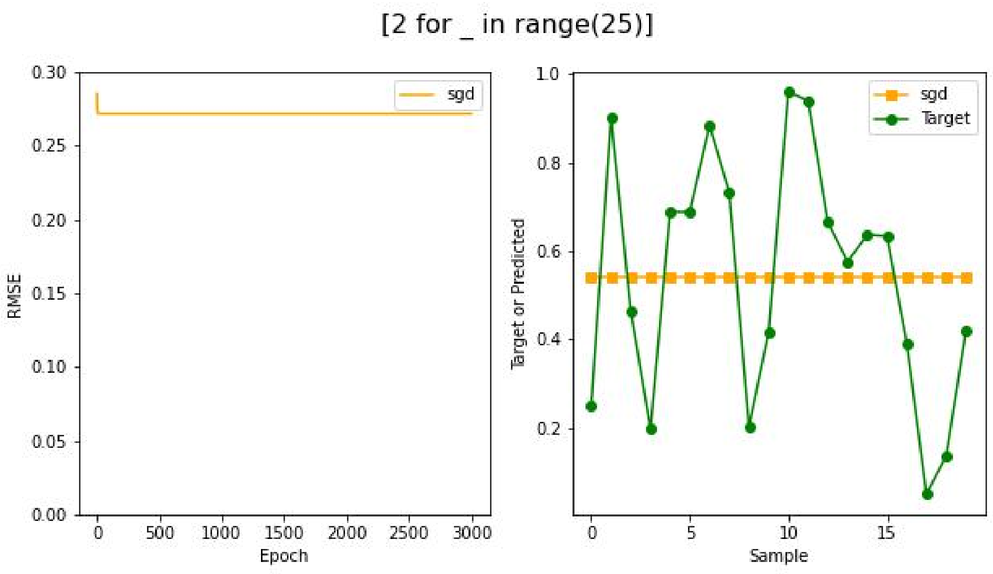


Figure 4.19: Plot of loss and prediction for non-residual network of depth 25 for batch training at a learning rate of 0.1. The left side of the figure shows the loss of the network not minimizing. The right side of the figure shows the network becoming a constant function from not being able to minimize its loss.

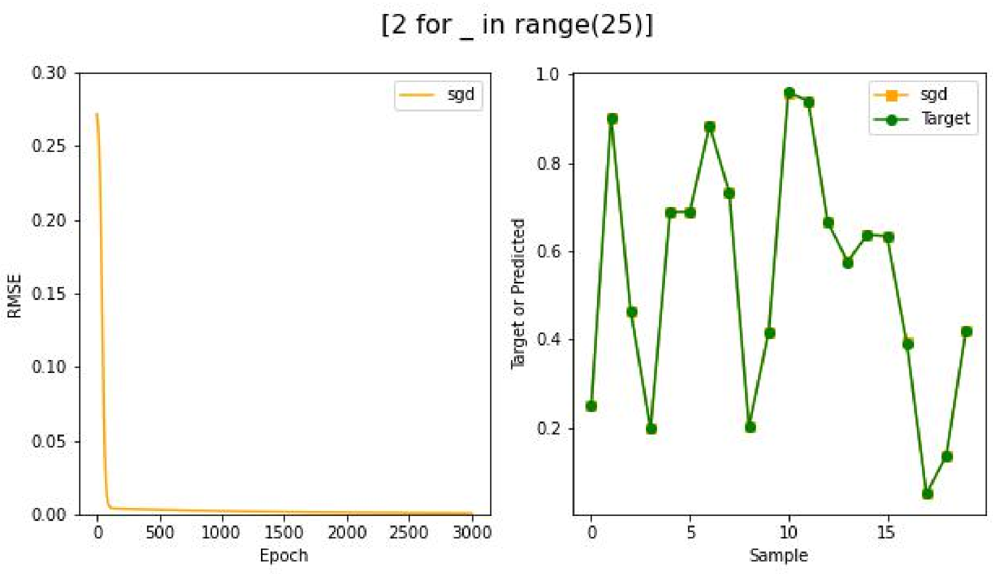


Figure 4.20: Plot of loss and prediction for late residual network of depth 25 for batch training at a learning rate of 0.1. The left side of the figure shows the loss of the network minimizing quickly due to the higher learning rate. The right side of the figure shows the networks predicted values overlapping with the target values in green.

4.3 Decline of Dead Neurons

In the previous sections it was shown that the addition of late residual connections improved convergence. However, the question of whether this improved convergence was the result of a decrease in amount of dead neurons, the pruning affect occurring, or both, still needed to be investigated. This section of results expands upon the previous experiments initially conducted. Here, the architectures are tested again but with each layer becoming wider from two to four. Figure 4.21 shows an example of these architectures.

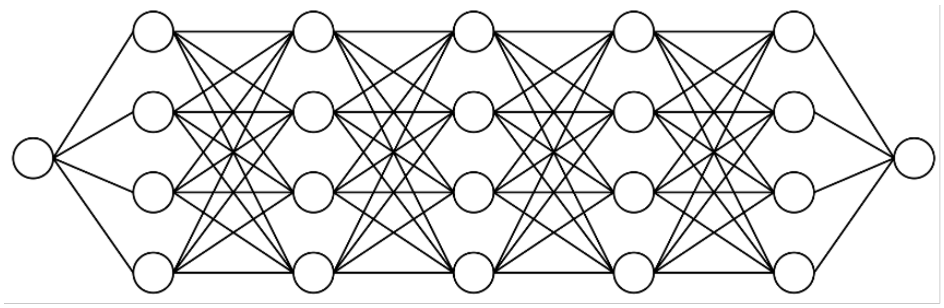
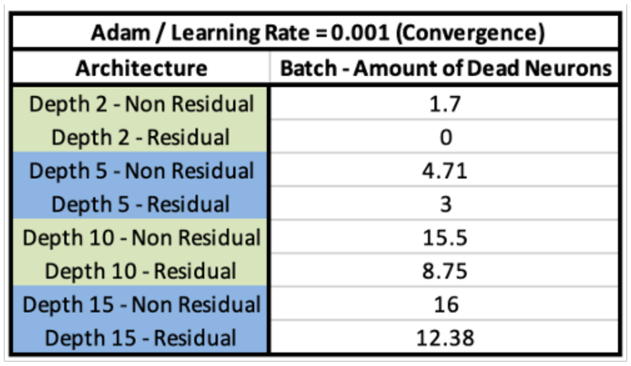


Figure 4.21: An example of the wider non-residual connection architecture. Each layer now has four neurons versus two in the previous experiments.

The architectures were expanded in width to allow more paths for the signal to move through on the forward pass. Specifically, this was done so that for a given layer if the two middle neurons died and the upper and lower neurons had not, then the next layer would have more probability to stay alive. Furthermore, in these experiments each architecture at each depth was run until three convergences were attained for both non-residual and late residual connections. This was done to allow more even balance to compare the decline in neurons. A subset of the results are shown in Figure 4.22 and Figure 4.23. Only a subset is shown for simplicity. However, where both connections converged late residual connections had a decrease in dead neurons.

As shown in both the Figure 4.22 and Figure 4.23 there is a decline in dead neurons with the late residual connections added. Furthermore, these results also took in account dead layers. Each architecture shown did not have dead layers. These results show that the late residual connections have the ability to short circuit around dead layers and decrease the number of dead neurons.



Adam / Learning Rate = 0.001 (Convergence)	
Architecture	Batch - Amount of Dead Neurons
Depth 2 - Non Residual	1.7
Depth 2 - Residual	0
Depth 5 - Non Residual	4.71
Depth 5 - Residual	3
Depth 10 - Non Residual	15.5
Depth 10 - Residual	8.75
Depth 15 - Non Residual	16
Depth 15 - Residual	12.38

Figure 4.22: A table showing the decline in dead neurons for depths 2, 5, 10, and 15 for both non-residual and late residual connections. These depths are only shown as after a depth of 15, none of the non-residual connections converged.

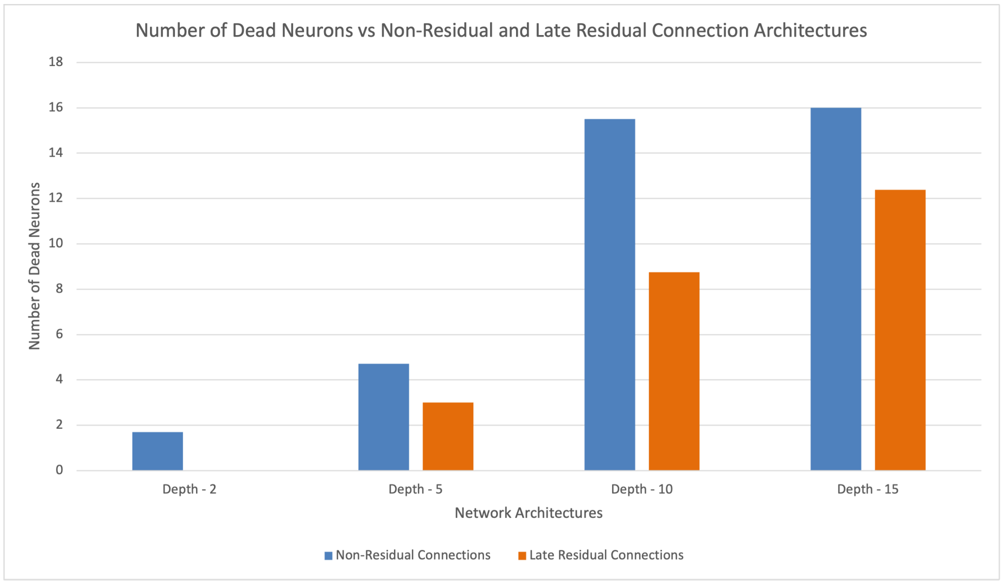


Figure 4.23: A graph showing the decline in dead neurons for depths 2, 5, 10, and 15 for both non-residual and late residual connections. These depths are only shown as after a depth of 15, none of the non-residual connections converged.

Chapter 5

Discussion and Conclusion

5.1 Summary

The experiments conducted in this thesis aimed to find a newly proposed solution for the dead ReLU problem and learn more about it. After investigating the problem, a hypothesis was formed that if neurons that use the ReLU activation function were to have more gradient flow, it could prevent killing of neurons.

The proposed solution involved adding residual connections from each hidden layer directly to the output layer. These connections were named "late residual connections" as they behave as a later connection to the output. The late residual connections were tested on known cases where the dead ReLU problem exists. Such cases include varying learning rates, deep architectures, and a known dataset that induced dead neurons.

Furthermore, these architectures were tested with two different optimizers, Adam and SGD, and two training styles, batch and iterative. The optimizers and training styles were chosen to fully depict the late residual connections' effects when added to a neural network. These added connections showed an improvement in convergence in a neural network using the ReLU activation function.

5.2 Future Work

This thesis lays the ground work in research of late residual connection to combat the dead ReLU problem. Future research added to this work could increase the validity of the results. Specifically, researching the generalizability of models used in deep learning with residual connections is an important step to show the robustness of the networks. Furthermore, an analysis of the space and time complexity of the late residual connections could give insight on the trade offs of using them.

In terms of architectures, the architectures in this thesis's experiments have a constant width but increasing depth. It would add to the validity of this research if there were more experiments performed on wider architectures. Furthermore, experiments run on more advanced networks that use the ReLU activation function, such as convolutional neural networks (CNN), could also show the validity of these connections. Moreover, if these connections can keep more neurons in a neural network alive, looking into the reduction of neurons needed to generalize a problem could take this research down a different path, such as pruning networks to only have the necessary neurons needed to generalize a problem. Moreover, it was shown that a pruning affect was occurring with these connections and could be investigated. Future work involving this phenomena could be explored as well.

Bibliography

[Agarap, 2019] Agarap, A. F. (2019). Deep Learning using Rectified Linear Units (ReLU). *arXiv:1803.08375 [cs, stat]*. arXiv: 1803.08375.

[Alom et al., 2018] Alom, M. Z., Taha, T. M., Yakopcic, C., Westberg, S., Sidike, P., Nasrin, M. S., Van Esesn, B. C., Awwal, A. A. S., and Asari, V. K. (2018). The History Began from AlexNet: A Comprehensive Survey on Deep Learning Approaches. *arXiv:1803.01164 [cs]*. arXiv: 1803.01164.

[Arnekvist et al., 2020] Arnekvist, I., Carvalho, J. F., Kragic, D., and Stork, J. A. (2020). The effect of Target Normalization and Momentum on Dying ReLU. *arXiv:2005.06195 [cs, stat]*. arXiv: 2005.06195.

[Bengio, 2009] Bengio, Y. (2009). Learning Deep Architectures for AI. page 56.

[Bottou, 2012] Bottou, L. (2012). Stochastic Gradient Descent Tricks. In Montavon, G., Orr, G. B., and Müller, K.-R., editors, *Neural Networks: Tricks of the Trade*, volume 7700, pages 421–436. Springer Berlin Heidelberg, Berlin, Heidelberg. Series Title: Lecture Notes in Computer Science.

[Clevert et al., 2016] Clevert, D.-A., Unterthiner, T., and Hochreiter, S. (2016). Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs). *arXiv:1511.07289 [cs]*. arXiv: 1511.07289 version: 5.

[Hahnloser et al., 2000] Hahnloser, R. H. R., Sarpeshkar, R., Mahowald, M. A., Douglas, R. J., and Seung, H. S. (2000). Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature*, 405(6789):947–951. Bandiera_abtest: a Cg_type: Nature Research Journals Number: 6789 Primary_atype: Research Publisher: Nature Publishing Group.

[Hanin, 2018] Hanin, B. (2018). Which Neural Net Architectures Give Rise To Exploding and Vanishing Gradients? *arXiv:1801.03744 [cs, math, stat]*. arXiv: 1801.03744.

[He et al., 2015a] He, K., Zhang, X., Ren, S., and Sun, J. (2015a). Deep Residual Learning for Image Recognition. *arXiv:1512.03385 [cs]*. arXiv: 1512.03385.

[He et al., 2015b] He, K., Zhang, X., Ren, S., and Sun, J. (2015b). Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. *arXiv:1502.01852 [cs]*. arXiv: 1502.01852.

[Kingma and Ba, 2017] Kingma, D. P. and Ba, J. (2017). Adam: A Method for Stochastic Optimization. *arXiv:1412.6980 [cs]*. arXiv: 1412.6980.

[Leung, 2021] Leung, K. (2021). The Dying ReLU Problem, Clearly Explained.

[Lu et al., 2020] Lu, L., Shin, Y., Su, Y., and Karniadakis, G. E. (2020). Dying ReLU and Initialization: Theory and Numerical Examples. *Communications in Computational Physics*, 28(5):1671–1706. arXiv: 1903.06733.

[NumPy, 2021] NumPy (2021). NumPy.

[PyTorch, 2021] PyTorch (2021). PyTorch.

[Santhanam and Davis, 2020] Santhanam, V. and Davis, L. S. (2020). A Generic Improvement to Deep Residual Networks Based on Gradient Flow. *IEEE Transactions on Neural Networks and Learning Systems*, 31(7):2490–2499. Conference Name: IEEE Transactions on Neural Networks and Learning Systems.

Appendix A

Link to Code Implementation

https://github.com/matthewfernst/LateResidualConnections