Effects of Learning Rate and Mini-Batch Size on Neural Network Performance in the Classification of Handwritten Digits

Nicholas Fabrizio
nicholas.fabrizio@colostate.edu
Colorado State University
Fort Collins, Colorado, USA

ABSTRACT

This paper evaluates the effects of the learning rate and mini-batch size hyper-parameters on neural network performance for several activation functions. The effects are measured during network classification of handwritten digits taken from a popular database. Several types of neural networks are used in order to compare and assess any differences in the performance when adjusting the hyper-parameters for each network. The networks are compared in a controlled environment to ensure that the effects on performance are caused by tuning the learning rate and mini-batch size rather than external factors.

CCS CONCEPTS

• **Networks → Network performance modeling**; **Network experimentation**; **Network performance analysis**.

KEYWORDS

neural networks, neural network performance, learning rate, mini-batch size, activation function, sigmoid, tanh, ReLU, handwritten digit recognition, MNIST data set, Scikit Learn, TensorFlow, Keras

ACM Reference Format:
Nicholas Fabrizio. 2021. Effects of Learning Rate and Mini-Batch Size on Neural Network Performance in the Classification of Handwritten Digits. In *Proceedings of CS-440 Introduction to Artificial Intelligence (CS-440).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Neural networks are a popular tool in use in artificial intelligence today; however, they can be quite complicated. There are several popular libraries that are available to make building and running neural networks simple and easy. These libraries offer the option of customizing the neural network with multiple configuration hyper-parameters. To the untrained practitioner, it can be challenging to determine what these hyper-parameters do, how they interact with each other and how they affect the performance of the neural network.

Three of the most conspicuous configuration hyper-parameters in neural networks are the activation function, the learning rate and the mini-batch size. Since these hyper-parameters are fairly well-known, they were selected as the subject of this study. The objective in using these hyper-parameters is to determine their effect on network performance as well as their effect on each other.

To thoroughly test these concepts, a challenging and complex task is needed. Classification of handwritten digits was chosen as an appropriate one. The data set used is the Modified National Institute of Standards and Technology (MNIST) data set of handwritten digits, and the measurement chosen is the accuracy of the network in classifying the digits.

This paper presents two sets of experiments that test the effects of changing these hyper-parameters on network performance and any impact the hyper-parameter values have on the other hyper-parameters. In section 2 of this paper, an explanation of the motivation for this study along with the reasoning behind what was done is provided. Some background and descriptions of similar studies in the field are given in section 3. Section 4 describes in detail the experimental design and the MNIST data set. This is followed by a discussion of the results in section 5, and final thoughts and conclusions in section 6.

2 MOTIVATION

The motivation for this set of experiments was to become more familiar with neural networks and some of their tuning hyper-parameters. These experiments compare three types of neural networks, a custom network, a multi-layer perceptron network using the Scikit Learn library and a TensorFlow with Keras network. The reason for comparing these three types of networks was to gain familiarity with some popular tool sets like Scikit Learn and TensorFlow with Keras and to have the ability to inspect how neural networks are coded to gain a better understanding of how they work by using a custom network for which the source code is fully available to study and update. The custom network was designed as a feedforward network that uses backpropagation to continuously update the weights and biases in the network using stochastic gradient descent. The custom network uses mini-batch updates as part of its backpropagation. The code for the custom network was taken from the online textbook Neural Networks and Deep Learning by Michael Nielsen [8]. The Scikit Learn network used the Scikit Learn MLPClassifier multi-layer perceptron classifier. The reason for using this classifier is that it is the simplest classifier available through Scikit Learn, and it was thought that it would be the most similar to the custom network. The code for the multi-layer perceptron network from Scikit Learn was taken from the example code on the Scikit Learn website [6]. The reason for including a TensorFlow Keras network in this study, other than gaining a familiarity with

CS-440, Fall 2021, Fort Collins, Colorado
© 2021 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

a popular library, is that it is an advanced network with many optimizations built in, and it was thought that this could provide a baseline of what is being used in neural networks today to compare the simpler custom and Scikit Learn networks against. The code for the TensorFlow Keras network was taken from the example code on the TensorFlow website [11].

Each of these networks was built to analyze and classify the MNIST data set of handwritten digits. Each network was set up with the same architecture, an input layer with 784 neurons, a hidden layer with 30 neurons and an output layer with 10 neurons. The number of neurons in the input layer is based on the shape of the MNIST data set. Each handwritten digit image in the data set is 28 pixels by 28 pixels square creating a total of 784 pixels, or inputs. The output layer is 10 neurons to allow for classifications for each of the digits zero through nine. The hidden layer is 30 neurons based on the recommendations of Michael Nielsen for optimal custom network performance [8].

Each network was also set up to use the stochastic gradient descent optimization algorithm. This optimization algorithm was selected because the custom network was written using this algorithm and selecting a different algorithm would require significant effort to be able to update the custom network code. In each of the experiments, the networks were set up to iterate over 30 epochs. This value was also chosen based on the recommendations of Michael Nielsen for optimal performance of the custom network [8].

While there are many hyper-parameters available with neural networks, this set of experiments focuses on the activation function and its interaction with the learning rate and mini-batch size. These hyper-parameters were chosen mainly due to the ease of which they can be manipulated without needing to make serious code modifications to the custom network code. Each network was also tested using the sigmoid, tanh and rectified linear unit (ReLU) activation functions. These activation functions were chosen due to their popularity in use for neural networks.

For each network, the performance accuracy was measured and compared for a series of values for both learning rate and mini-batch size for each of the three activation functions, and additionally, the execution time was measured and compared for each of these combinations. The range of hyper-parameter values for the learning rate were selected based on several factors, the recommendations of Michael Nielsen for optimal performance of the custom network [8], the values used in the example code for the Scikit Learn and TensorFlow Keras networks and the values at which the accuracy of the custom network started decreasing. The range of hyper-parameter values for mini-batch size were selected based on the values used in the example code for each network [6, 8, 11]. The value of the mini-batch size was kept constant in the learning rate experiments, and the value of the learning rate was kept constant in the mini-batch size experiments. This was done to maintain experimental control over the variables in order to ensure that the variables being tested were responsible for any changes in classification accuracy. The mini-batch size constant value was selected based on the recommendations of Michael Nielsen for optimal performance of the custom network [8]. The learning rate constant value was selected based on the results from the learning rate experiments. The learning rate with overall optimal average accuracy was used.

3 BACKGROUND

Handwritten digit classification and the MNIST data set in particular have been the subjects of many experiments and studies using neural networks. Early studies focused mainly on comparing different neural network architectures, and more recent studies have focused on adjusting the number of hidden layers and epochs. A summary of selected studies is provided below.

In 1988, a study was done on handwritten digit recognition to compare different neural network architectures. The authors used their own set of handwritten digits normalized to 16x16 pixel maps, and they compared K-Nearest Neighbors and Parzen windows to networks using either single or double adaptive layers. As part of their experiment, they trained several small networks to handle subsets of the data in a pair-wise fashion and then combined the results. K-Nearest Neighbors outperformed Parzen windows; however, the networks with the adaptive layers performed the best and were smaller in size. The overall best performing architecture was the multi-layer network of adaptive units trained with a backpropagation algorithm [3].

In 1994, a similar study was performed to classify handwritten digits from the US National Institute of Standards and Technology (NIST). The study compared different neural network architectures which used K-Nearest Neighbors and a linear classifier as a baseline. The baseline performance was compared to several other architectures including tangent distance classifier, optimal margin classifier, LeNet 1, LeNet 4 and a boosted version of LeNet 4. In this study, boosted LeNet 4 outperformed all of the other classifiers in terms of raw error rate, but it also had the longest training time. LeNet 1 had the lowest training time, and LeNet 1, LeNet 4 and boosted LeNet 4 had the lowest memory requirements [1].

A more recent study in 2019 classified handwritten digits from the MNIST data set using K-Nearest Neighbors, decision tree, support vector machines and a deep neural network. The authors compared each of these architectures using accuracy, sensibility, specificity, precision, Jaccard, Dice and Matthews's Correlation Coefficient (MCC) evaluation criteria. They found that the deep neural network performed the best out of the architectures studied for all of the evaluation criteria used [4].

In 2018, a study similar to the one conducted for this paper was done using TensorFlow and the MNIST data set. The authors of that study focused on changing the numbers of hidden layers in the neural network with small adjustments to the mini-batch sizes as well as changing the number of epochs, and they compared the accuracies alone to determine the best combination. The authors initialized the weights and biases randomly and used gradient descent to adjust them with each mini-batch. They found the highest accuracy between two and five hidden layers with a mini-batch size of 50 [10].

A similar study was done in 2019 with TensorFlow and the MNIST data set using accuracies to compare the performance of the architectures. That study used stochastic gradient descent as well, and they found overall accuracy to be optimal when using 4 hidden layers and a mini-batch size of 100 [9].

While all of these studies present interesting aspects of neural networks and use handwritten digit classification as the experimental task, they are largely focused on network design and resource allocation. Network design and resource allocation are important factors to consider and examine when using neural networks as they impact needed computer resources and the cost to run the network. However, they still leave the question of how adjusting hyper-parameters such as learning rate and mini-batch size will impact the performance of networks during handwritten digit classification.

4 EXPERIMENTS AND DATA

This section provides a detailed description of the experiments and how they were run as well as detailed information about the data set used. The purpose of providing a high level of detail about the experiments is twofold. The first reason is to aid in the reader's understanding of what was done in the experiments, and the second reason is to provide enough information to allow the reader to replicate the experiments on her/his own.

4.1 Experiments

For this study, there were two sets of experiments, both of which were focused on classification of handwritten digits. One set to test the effects of the learning rate on network classification accuracy and one set to test the effects of mini-batch size on network classification accuracy. Both sets of experiments had a similar design.

Each experiment utilized three different neural networks, a custom network, a Scikit Learn MLPClassifier network and a TensorFlow Keras network. For each network, an architecture of three layers was used, an input layer, one hidden layer and an output layer. The input layer was composed of 784 neurons, the hidden layer was composed of 30 neurons and the output layer was composed of 10 neurons. Each network used stochastic gradient descent as the optimization algorithm and ran for 30 epochs. For each set of experiments, each network was run against the data set using each of the following activation functions, sigmoid, tanh and ReLU.

For the effects of the learning rate on network classification, the range of learning rates used was 0.001, 0.01, 0.1, 1.0, 2.0, 3.0, 4.0 and 5.0. For the effects of mini-batch size on network classification, the range of mini-batch sizes used was 25, 50, 75, 100, 125, 150, 175 and 200. In the experiment to test the effects of the learning rate on network classification accuracy, the mini-batch size was kept constant at 10 for each network and activation function. For the experiment that tested the effect of mini-batch size on network classification accuracy, the learning rate was kept constant at 0.1 for each network and activation function. Each network was run once using the configuration for each of the values in the learning rate and mini-batch size ranges.

All the experiments were done on a machine running macOS with a 2.2 GHz Quad-Core i7, 16 GB memory, a 1600 MHz DDR3 with 1 processor and 4 cores. The network runs for each experiment were done one at a time to avoid any resource constraints with the machine that was used. At the end of each network run, the data for the epoch with the highest accuracy on the validation data was recorded along with the execution time for that run.

The execution time was calculated using the Python time module. The execution start time for each network was initiated just before the network architecture was initialized, and the execution end time was measured at the end of each network run. This means that the time for the loading of the data set is not included in the execution time.

One difference between the three networks was the method of weight and bias initialization. For the custom network, weights and biases were initialized randomly and then adjusted using backpropagation. For the Scikit Learn network, the Glorot uniform method was used for weight and bias initialization [2]. The TensorFlow Keras network used the Glorot uniform method for weight initialization and initialized the biases to zeros [5]. In addition to this, the Scikit Learn classifier had 23 hyper-parameters, and 19 of these had default values set when none was provided [6]. The TensorFlow Keras network had 10 hyper-parameters, and three of these havd default values when none was provided [5].

Another difference between the networks was how the data set was imported. The custom network loaded a local pickle file containing all the data. The Scikit Learn network fetched the data set from OpenML, and the TensorFlow Keras network loaded the data from the TensorFlow data sets library. All three networks were set up in such a way that for each epoch both the training data and the validation data were used in conjunction with the weights and biases for that epoch. This allowed the networks to provide a measure of accuracy for the validation data with each epoch.

4.2 Data Set

The data set that was used for these experiments was the MNIST database of handwritten digits. This data set was composed of a training set and a test set with sizes of 60,000 and 10,000 elements respectively. The images in this data set were size normalized and centered within a 28-pixel by 28-pixel image. Each image contained a single, handwritten digit from zero to nine that was mostly black with some grey on a white background. The images were 28 pixels by 28 pixels with each pixel having a value from zero to 255, zero being white and 255 being black. The files in the data set were stored in the IDX file format using the most significant byte (MSB) first format, and the data sets used for this project had already been labeled with integer values from zero to nine [7].

5 RESULTS

Figures 1 through 6 show the result data from the two sets of experiments. Figures 1 through 3 show the results for the effect of learning rate on network performance, and figures 4 through 6 show the results for the effect of mini-batch size on network performance. The results for each set of experiments are grouped by activation function for easier comparison. Also, for ease of comparison, the colors used to represent each network type are consistent in all figures.

Figure 1 shows the accuracy for each network for each learning rate in the given range when using the sigmoid activation function. The TensorFlow Keras network achieved an accuracy of 96.92%, the highest accuracy of all the networks, when a learning rate of 0.1 was used. The Scikit Learn network achieved its highest accuracy of 96.5% when using a learning rate of 0.01, and the custom network achieved its highest accuracy of 95.26% when using a learning rate of 4.0. The custom network was least accurate at lower learning rates and then remained relatively high. The Scikit Learn network was most accurate with lower learning rates, and its accuracy dropped off at higher learning rates. The TensorFlow Keras network had relatively high accuracies for all learning rates.

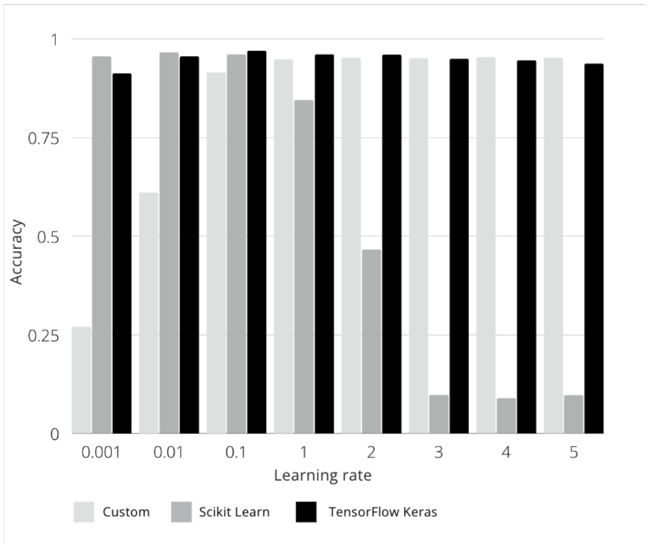

Figure 1: Accuracy by learning rate for sigmoid activation function.

The data shown in Figure 2 is the accuracy of the networks for learning rate using the tanh activation function. The Scikit Learn network had the highest overall accuracy of 96.93% at a learning rate of 0.001 followed by the TensorFlow Keras network with an accuracy of 96.51% at a learning rate of 0.01. The custom network had its highest accuracy of 86.63% at a learning rate of 0.1. When using the tanh activation function, there is a much more abrupt drop in accuracy for both the custom and Scikit Learn networks at higher learning rates, and the TensorFlow Keras network also has a drop in accuracy at higher learning rates although it is not as drastic.

Figure 3 shows the accuracy for each network by learning rate while using the ReLU activation function. The TensorFlow Keras network had the highest overall accuracy of 96.96% at a learning rate of 0.01, and the Scikit Learn network had its highest accuracy of 95.88% at a learning rate of 0.01. The custom network had an extremely low accuracy for all learning rates while using ReLU, so the data from that network has been omitted from Figure 3. Using ReLU resulted in a sharp drop in accuracies at higher learning rates for both the TensorFlow Keras and the Scikit Learn networks.

In Figure 4, the data for the effects of mini-batch size on accuracy using the sigmoid activation function is shown. The TensorFlow Keras network had the highest overall accuracy of 96.92% when using a mini-batch size of 25. The Scikit Learn network had its highest accuracy of 96.78% with a mini-batch size of 100, and the custom network had its highest accuracy of 88.81% with a mini-batch size of 25. Both the TensorFlow Keras and Scikit Learn networks maintained

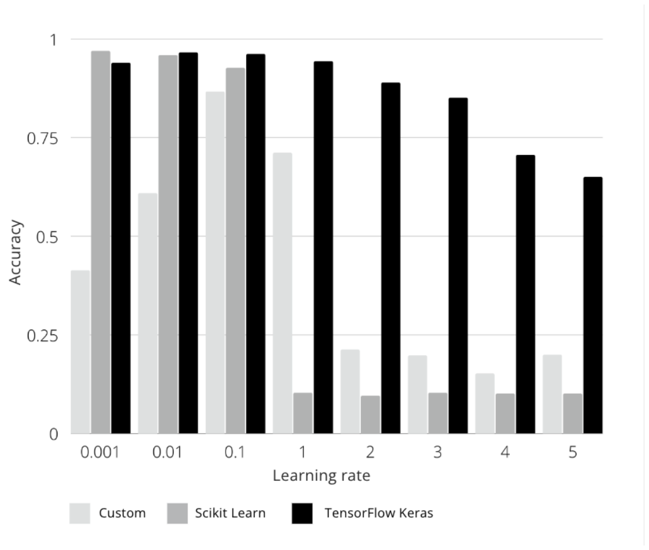

Figure 2: Accuracy by learning rate for tanh activation function.

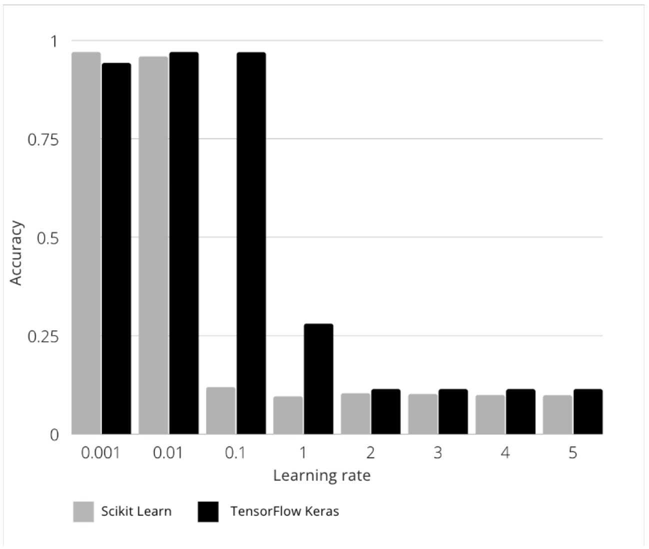

Figure 3: Accuracy by learning rate for ReLU activation function.

relatively high accuracies with larger mini-batch sizes; however, the custom network had significantly lower accuracy with larger mini-batch sizes.

Figure 5 shows the accuracy of the networks by mini-batch size using the tanh activation function. With this configuration, the TensorFlow Keras network achieved an accuracy of 96.71% with a mini-batch size of 125, the Scikit Learn network achieved 96.26% with a mini-batch size of 175 and 200 and the custom network achieved 79.41% with a mini-batch size of 50. Again, the TensorFlow Keras and Scikit Learn networks were able to maintain relatively high

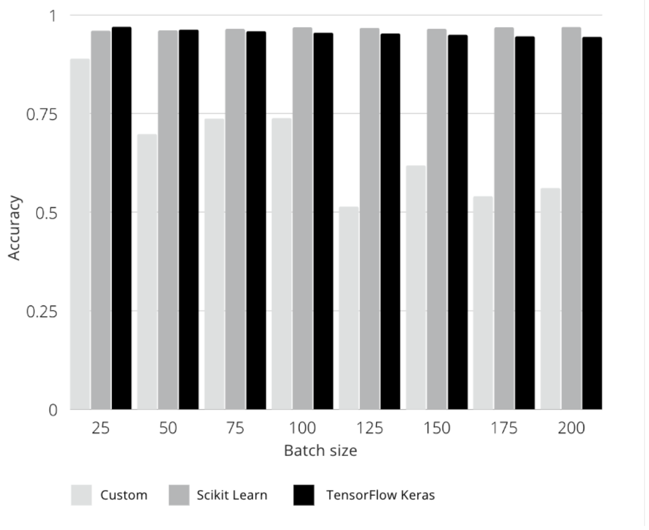

Figure 4: Accuracy by mini-batch size for sigmoid activation function.

accuracies regardless of mini-batch size, but the custom network saw a drop in accuracy with larger mini-batch sizes.

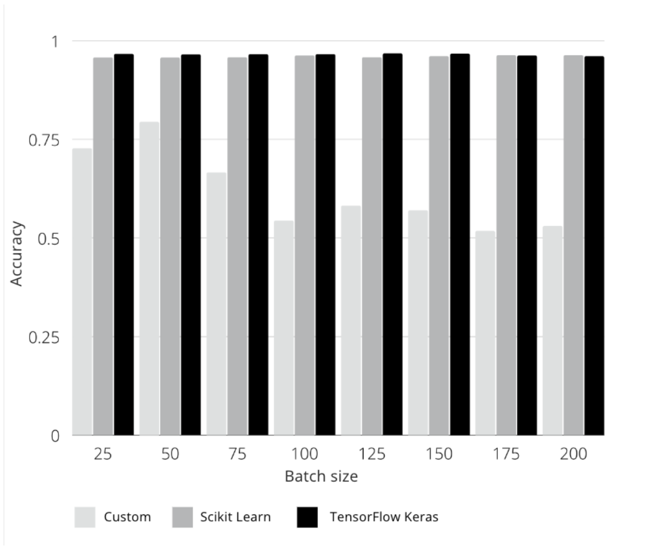

Figure 5: Accuracy by mini-batch size for tanh activation function.

In Figure 6, the data for mini-batch size versus accuracy using ReLU is shown. The TensorFlow Keras network had an accuracy of 97.19% with a mini-batch size of 50, and the Scikit Learn network had an accuracy of 96.83% with a mini-batch size of 175. Similar to the learning rate experiment, the custom network had a constant, low accuracy for all epochs and mini-batch sizes, so the data for that network has been omitted from Figure 6. As with the other activation functions, the TensorFlow Keras and Scikit Learn networks both maintained relatively high accuracies regardless of mini-batch size.

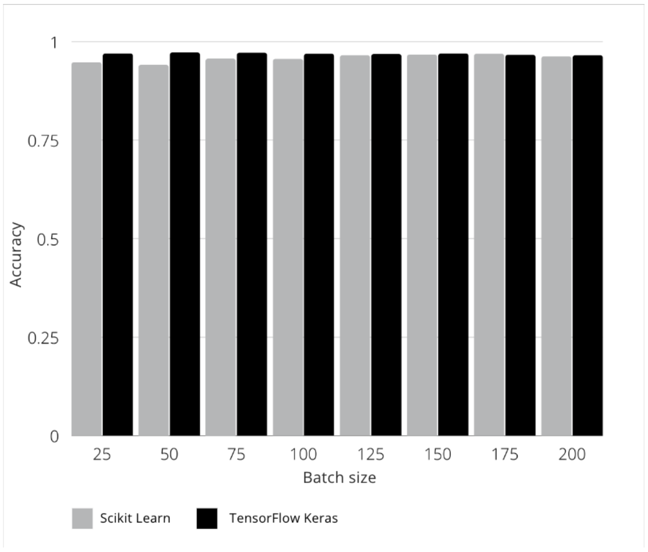

Figure 6: Accuracy by mini-batch size for ReLU activation function.

In analyzing the data, the learning rate had little effect on accuracy for the TensorFlow Keras and custom networks when using the sigmoid activation function, and higher learning rates caused a decrease in accuracy for all network types tested when using tanh or ReLU activation functions. Higher learning rates decreased accuracy regardless of the activation function when using the Scikit Learn network. With the popularity of the ReLU activation function, it is interesting that it had the lowest tolerance for changes in learning rate while the sigmoid activation function showed good tolerance for changes in learning rate for the TensorFlow Keras network. Higher mini-batch sizes resulted in lower accuracy for the custom network, but mini-batch size had little effect on the accuracy of the TensorFlow Keras and Scikit Learn networks.

A surprising aspect of the data displayed in Figure 1 and Figure 2 is the ability of the TensorFlow Keras network to maintain a high level of accuracy with very high learning rates. There are several potential explanations for this behavior. Overfitting of the model might be one explanation if this behavior were only exhibited on the training data; however, the data displayed in Figure 1 and Figure 2 is from the validation set, so this explanation seems unlikely. Another possible explanation is an error in the implementation of the network code. While this may initially seem to be the most likely, the network code used was taken from the TensorFlow website example code, so this explanation is probably also unlikely. It is also possible that the TensorFlow Keras network is somehow caching and reusing the results or that there is a fault in the Keras codebase. Of these options, the fault in the Keras codebase seems most likely, but more research is necessary to determine the cause of this behavior.

6 CONCLUSIONS

In this study, the effects of changing the learning rate and mini-batch size on the performance of different types of neural networks while classifying handwritten digits were analyzed. A majority of the internal and external variables were controlled to yield the most accurate results. Three network types were tested using the same architectural configuration, optimization algorithm, number of epochs and activation functions. The results showed that higher learning rates had strong detrimental effects on network performance for a majority of the activation functions tested, but that the sigmoid activation function was resistant to these effects. The results also showed that mini-batch size had little effect on the performance of widely used neural network types regardless of which activation function was used.

While it was originally thought that using a small mini-batch size during the learning rate experiments might have a negative effect on the accuracies measured for the TensorFlow and Scikit Learn networks, the mini-batch size experiment proved that this belief was unfounded. When using popular neural networks, it is important to keep the learning rate low; however, it is also important to consider the type of network being used when selecting the learning rate since some networks may be designed to perform better with higher learning rates. Further research should be done to determine the reason for the apparent lack of impact that very high learning rates had on the accuracy of the TensorFlow Keras network. When selecting a mini-batch size, it is important to remember that, in general, mini-batch size has little effect on network classification accuracy, but it can be adjusted to make small improvements to network performance.

While many of the variables were controlled in these experiments, the weight and bias initialization is one that could account for the difference in performance levels between the TensorFlow Keras and Scikit Learn networks and the custom network. These experiments could be repeated with the initialization variable controlled to see if the performance levels would be closer between all networks since there was a large disparity between them in the experiments performed for this study. Additionally, the cause of the low performance of the custom network when using the ReLU activation function should be investigated, so that data for the custom network can be collected and used in comparing the network types for the ReLU activation function.

The main motivation for this study was to become more familiar with neural networks and their tuning hyper-parameters, and that objective was certainly accomplished. However, this study investigated a small number of the tuning hyper-parameters available. Future work will investigate more of the tuning hyper-parameters and their effect on network performance both individually and combined.

REFERENCES

[1] L. Bottou, C. Cortes, J.S. Denker, H. Drucker, I. Guyon, L.D. Jackel, Y. LeCun, U.A. Muller, E. Sackinger, P. Simard, and V. Vapnik. 1994. Comparison of Classifier Methods: A Case Study in Handwritten Digit Recognition. In *Proceedings of the 12th IAPR International Conference on Pattern Recognition, Vol. 3 - Conference C: Signal Processing (Cat. No.94CH3440-5)*, Vol. 2. 77–82 vol.2. https://doi.org/10.1109/ICPR.1994.576879

[2] Github. 2017. *Scikit Learn Multilayer Perceptron Python File.* Retrieved November 27, 2021 from https://github.com/scikit-learn/scikit-learn/blob/ef5cb84a/sklearn/neural_network/multilayer_perceptron.py#L300

[3] I. Guyon, I. Poujaud, L. Personnaz, G. Dreyfus, J. Denker, and Y. LeCun. 1989. Comparing Different Neural Network Architectures for Classifying Handwritten Digits. In *International 1989 Joint Conference on Neural Networks.* 127–132 vol.2. https://doi.org/10.1109/IJCNN.1989.118570

[4] Soufiane Hamida, Bouchaib Cherradi, Abdelhadi Raihani, and Hassan Ouajji. 2019. Performance Evaluation of Machine Learning Algorithms in Handwritten Digits Recognition. In *2019 1st International Conference on Smart Systems and Data Science (ICSSD).* 1–6. https://doi.org/10.1109/ICSSD47982.2019.9003052

[5] Keras. [n.d.]. *Dense Layer.* Retrieved November 27, 2021 from https://keras.io/api/layers/core_layers/dense

[6] Scikit Learn. 2021. *Visualization of MLP Weights on MNIST.* Retrieved November 24, 2021 from https://scikit-learn.org/stable/auto_examples/neural_networks/plot_mnist_filters.html

[7] Yann LeCun, Corinna Cortes, and Christopher J.C. Burges. [n.d.]. *The MNIST Database of Handwritten Digits.* Retrieved October 3, 2021 from http://yann.lecun.com/exdb/mnist

[8] Michael Nielsen. 2019. *Neural Networks and Deep Learning.* Retrieved November 24, 2021 from http://neuralnetworksanddeeplearning.com/chap1.html#implementing_our_network_to_classify_digits

[9] Fathma Siddique, Shadman Sakib, and Md. Abu Bakr Siddique. 2019. Recognition of Handwritten Digit Using Convolutional Neural Network in Python with Tensorflow and Comparison of Performance for Various Hidden Layers. In *2019 5th International Conference on Advances in Electrical Engineering (ICAEE).* 541–546. https://doi.org/10.1109/ICAEE48663.2019.8975496

[10] Md. Abu Bakr Siddique, Mohammad Mahmudur Rahman Khan, Rezoana Bente Arif, and Zahidun Ashrafi. 2018. Study and Observation of the Variations of Accuracies for Handwritten Digits Recognition with Various Hidden Layers and Epochs using Neural Network Algorithm. In *2018 4th International Conference on Electrical Engineering and Information Communication Technology (iCEEiCT).* 118–123. https://doi.org/10.1109/CEEICT.2018.8628144

[11] TensorFlow. 2021. *TensorFlow 2 Quickstart for Beginners.* Retrieved November 24, 2021 from https://www.tensorflow.org/tutorials/quickstart/beginner