CS EE World
https://cseeworld.wixsite.com/home
May 2022
28/34
A

Submitter info:
Email: andrewp0809 [at] gmail [dot] com

Investigating the performance of neural networks given mislabelled training data

To what extent is the performance of a neural network dependent on the batch size and number of epochs at varying rates of training data mislabelling?

Subject: Computer Science

Word Count: 3714

Table of Contents

1. Introduction **3**

2. Theoretical Background **5**

2.1 Feedforward Neural Networks 5

2.2 Training a Neural Network 7

2.3 Batch Sizes 7

2.4 Number of Epochs 8

3. Experimental Methodology **8**

3.1 The Dataset 9

3.2 The Structure of the Neural Network 10

3.3 The Activation Function 11

3.4 The Accuracy 12

3.5 The Loss Function 12

3.6 The Experimental Procedure 14

4. Experimental Results **14**

4.1 Tabular Data Presentation 14

4.2 Graphical Data Presentation 18

4.3 Data Analysis 19

5. Limitations to Investigation **21**

6. Conclusion **22**

7. Works Cited **24**

8. Appendix **27**

8.1 Code for Neural Network 27

1. Introduction

The development of technology allowing computers to learn, specifically technology that is able to learn from current data and apply to new data, can lend an invaluable hand to many fields once thought to be dominated by manual labour. For instance, it has been shown that this technology, known as machine learning, is able to accurately diagnose many common diseases from medical data, providing an invaluable aid to medical professionals and further automating many data-driven classification problems (Zeng et al. 227).

Supervised learning is a subcategory of machine learning where labelled datasets are used to train algorithms to classify data or project outcomes, and one such commonly used method efficient at image recognition and classification is a neural network (IBM Cloud Education, *What is supervised learning?*; Schmidhuber 24). As labelled data is fundamental in the function of a neural network, an ample quantity of quality labelled training data must be sourced for a neural network to "learn" effectively.

Autonomous vehicles are one such field making use of neural networks to classify objects on the road, and thus require massive amounts of data to be labelled for the purpose of training. However, this job is a task often outsourced to cheaper labour, leading to generally lower quality training data, usually presented as mislabelled data (Elliott). Cleaning such training data can be an incredibly tedious process, and it may be infeasible to find and correct all examples of mislabelled training data (Vincent). In the context of autonomous vehicles where a computer wrongly classifying an object may lead to a fatal accident, a classification algorithm must make as few errors as possible. Hence, the ability of machine learning models to remain effective despite mislabelled data is extremely important in the further implementation of classification algorithms to a wider scope.

This essay aims to investigate how two parameters often controlled in the training of a neural network—the batch size and number of epochs—affect the effectiveness of a neural network in environments with varying levels of training data mislabelling. Hence, my research question is: "To what extent is the **performance** of a neural network dependent on the **batch size** and **number of epochs** at varying rates of training data **mislabelling**?". Based on secondary research, the expected result is that the performance of the neural network would increase as the batch size decreases, the number of epochs increases, and the rate of training data mislabelling decreases.

To do so, a theoretical background of a generic feedforward neural network and the related training process are first provided, before the procedure and results of experiments conducted to determine the efficacy of several neural network training parameters at varying levels of training data mislabelling are presented. Specifically, accuracy and loss will be used to quantify the performance of the neural networks. These results would be analysed in order to explain any apparent outcomes, and come to a conclusion on the investigation.

2. Theoretical Background

2.1 Feedforward Neural Networks

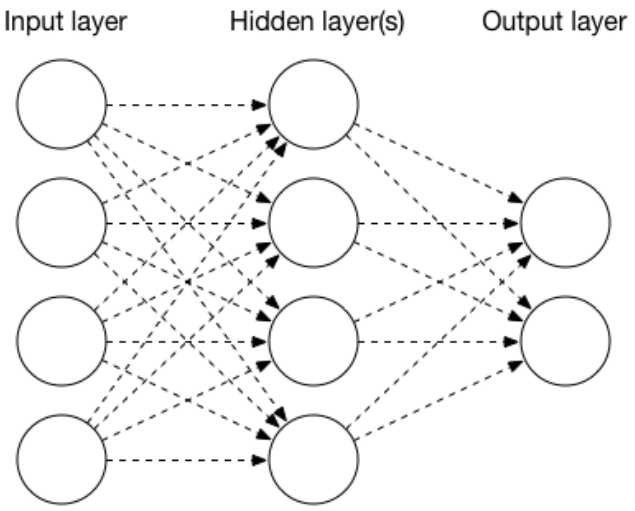


Figure 1: A simple feedforward neural network (Kerimbaev)

Neural networks are computing systems loosely based on the biological neural networks that constitute animal brains. Neural networks consist of nodes (also known as artificial neurons or perceptrons), which have a bias, and store a specific normalised value between 0 and 1 (IBM, *What Are Neural Networks?*). In a feedforward neural network, these nodes form two or more layers, where all the nodes in each layer are connected to all the nodes in the next layer, with each connection having its own weight. The data in each layer propagates forwards, with each node in the next layer being calculated through a series of matrix operations on the previous layer (IBM, *What Are Neural Networks?*). Through the feedforward process, complicated logical "decisions" can arise from these simple calculations, for example in how two nodes can act as logic gates. In the rest of the essay, it can be presumed that all neural networks mentioned are feedforward neural networks.

$$\begin{bmatrix} a_1^{(k+1)} \\ a_2^{(k+1)} \\ \cdots \\ a_i^{(k+1)} \end{bmatrix} = \sigma\left(\begin{bmatrix} a_1^{(k)} \\ a_2^{(k)} \\ \cdots \\ a_j^{(k)} \end{bmatrix} \cdot \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,j} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,j} \\ \cdots & \cdots & \cdots & \cdots \\ w_{i,1} & w_{i,2} & \cdots & w_{i,j} \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ \cdots \\ b_i \end{bmatrix} \right)$$

Equation 1: The forward propagation matrix operation

The above equation represents the matrix operation performed on each layer in order to calculate the activation of the next layer. In the equation, a_1 to a_j represent the activations of the first neuron all the way to the *j*th neuron of the specific layer *k*. The weights of a connection between the *j*th neuron in a specific layer to the *i*th neuron in the next layer is represented by a $w_{i,j}$, and the bias of a neuron a_i in the next layer is given as b_i. For example, to calculate the new value of a node a_i, the nodes in the previous layer from a_1 to a_j are multiplied by their respective weights from $w_{i,1}$ to $w_{i,j}$. These values are then added together along with the bias b_i, to get the new weighted sum. The weighted sum is passed through some form of non-linear activation function σ, the reason for which will be explained below, thus giving the new normalised value for the node a_j in the next layer. This is performed for every node in the next layer, and for every layer following that, to get the final activations of the nodes in the output layer.

The output layer of a neural network is made up of a number of nodes equaling the number of possible categories the neural network is intended to classify (IBM, *What Are Neural Networks?*). Theoretically, a perfect neural network should take an input and output only a large positive number on the node corresponding to the desired output and a large negative

number on all other nodes (Riedmiller). However, in an untrained or insufficiently trained neural network, a random set of nodes may fire weakly, or the wrong node may fire strongly, which would result in a misclassification. Thus, the biases and weights of all the nodes and connections in a neural network need to be tweaked, to reduce the likelihood of such events occurring (Riedmiller). This is known as the training of a neural network.

2.2 Training a Neural Network

The process of training a multi-layered neural network generally consists of two stages. The first stage is to simply calculate the current output of the neural network for a given training sample, and does not seek to change anything. In the next stage, the output is compared to the desired response of the neural network, and the deviations between the desired response and those produced by the neural network for each output layer node are calculated (Riedmiller). Following this, the weight of the connections between the output layer and the previous layer, as well as the biases of the nodes in the previous layer, are updated depending on the magnitude and direction of the deviations, with the aim of reducing the total error of the output layer (Riedmiller). This process is then repeated for each previous layer consecutively. This step, as seen from how the process steps backwards through the neural network, is called backward propagation.

2.3 Batch Sizes

The backwards propagation step in the training of a neural network is a resource intensive step due to the number and complexity of calculations needed to be performed. Hence, the concept of a *batch* is often implemented in the training of neural networks, by performing the backpropagation step on the cumulative desired change of the output nodes across numerous data samples (Sharma). This cuts down on the number of times the computationally

expensive backpropagation calculation needs to be performed, speeding up the training of the neural network.

2.4 Number of Epochs

In order to achieve the highest accuracy possible off of a limited dataset, a neural network may be configured to train from a dataset multiple times, with each pass through the dataset referred to as an *epoch* (Sharma). With each epoch, the weights and biases of the neural network should be updated to more accurately fit the training data. Despite this, a greater number of epochs may not have a fully positive impact on the ability of the neural network to classify generalised testing data, due to issues such as overfitting—a process where the neural network is trained to recognise features too specific to its training data, thus becoming less applicable to more generalised testing data (IBM, *What Is Overfitting?*).

3. Experimental Methodology

To answer the research question, a neural network was trained on a public dataset at differing combinations of batch size, number of epochs, and rates of mislabelling in the training portion of the dataset. The accuracy and loss of the neural network in each case were recorded. Due to the random nature of the error in the labels of the training data and the setting of the original weights and biases, the performance of the neural network was not consistent and prone to random fluctuations. Hence, each experimental combination was performed 3 times and averaged to obtain more consistent results.

The independent variables refer to what is being changed to produce a result in the experiment, and are the **batch size**, **number of epochs**, **and rates of mislabelling** in the training portion of the dataset. The result produced by the altering of the independent

variables would lead to a change in the dependent variables, which are the **accuracy** and **loss** of the neural network.

The specific details of each component of this experiment, as well as the procedure by which this experiment was carried out, are described in this section. The code used in this experiment is listed in the appendix.

3.1 The Dataset

The MNIST (Modified National Institute of Standards and Technology) dataset is a public dataset of thousands of 28x28 labelled images of handwritten digits ranging from 0-9 by LeCun et al. The 60,000 training images and 10,000 testing images were collected from a mix of Census Bureau employees and high school students.

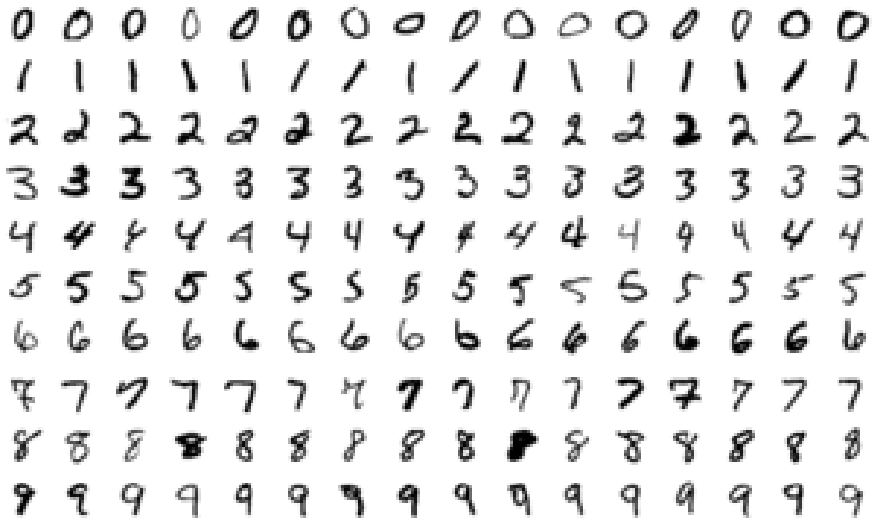


Figure 2: A few samples from the MNIST test dataset (Steppan)

The MNIST dataset is a dataset often used in the testing of neural networks due to the small image sizes and number of classifications; each image is composed of only 784 pixels values

between 0 and 255, and has 10 possible categories. When compared to other datasets, for example those with numerous colour channels, the MNIST dataset is significantly simpler. In the case of this experiment, this simple dataset enables the performance of the neural network to be tested without long training times, or the complexity of dealing with multiple colour channels and many more first layer nodes.

In order to achieve the aims of this experiment, derivations of the MNIST dataset at varying rates of mislabelling in the training data portion were produced. They were generated at run time of the program, by altering a specified number of random training data labels to another random classification before training the neural network.

3.2 The Structure of the Neural Network

The neural network was constructed with a singular hidden layer, along with the input and output layers. This structure was chosen to keep the neural network simple, while still providing enough complexity for the neural network to potentially fit the training data well. The input layer had 784 nodes, corresponding to every pixel in each 28x28 image of the MNIST dataset, and the output had 10 nodes, corresponding to the 10 possible classifications (from 0 to 9) of each image. The hidden layer contained 16 nodes, an arbitrary number chosen only because it was small enough to not complicate the neural network. The structure of the neural network was rather basic due to the minimal requirements of this experiment. In addition, this simpler neural network made analysing the effect that mislabelled training data had on the performance of the neural network easier.

3.3 The Activation Function

The activation function in a neural network is a function that normalises the weights of the nodes in the neural network. Without it, the neural network would remain a linear function of its inputs, taking away the network's ability to "learn" or perform complex functions (Brownlee). The activation function chosen for the hidden layer of the neural network is the rectified linear activation function, implemented in nodes known as rectified linear activation units (ReLU).

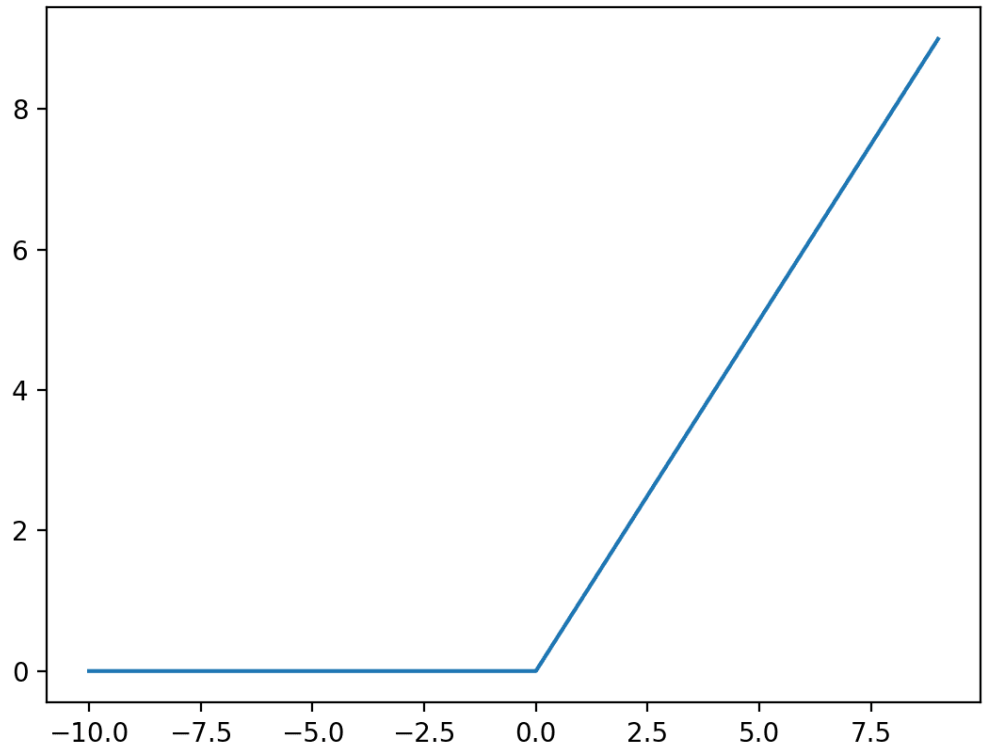


Figure 3: Line Plot of Rectified Linear Activation for Negative and Positive Inputs (Brownlee)

The rectified linear activation function can be described by the simple equation, $f(x) = \max(0, x)$, meaning the output is the same as the input if the input is positive, otherwise the output is

0. As the calculations in a ReLU are computationally simple, the neural network can be trained quicker; Many other common activation functions, such as the logistic (sigmoid) and hyperbolic (tanh) functions, require significantly more computation to calculate their outputs (Sycorax). In addition, due to the nature of these functions having outputs within smaller ranges, they suffer from the vanishing gradient problem. This makes the backwards propagation step much slower, especially in neural networks with many layers (Sycorax). This is not the case for the rectified linear activation function, and is another reason why it was chosen for this experiment.

3.4 The Accuracy

The accuracy recorded is the proportion of testing images the neural network correctly classified after being trained. Considering that the role of a neural network is simply to categorise data, accuracy is the most relevant metric in determining the performance of the neural network, and can be used to compare the effect of different parameters on the real world applicability of the neural network.

3.5 The Loss Function

The loss function is a function used to map the outputs of a neural network to some sort of "cost" associated with the prediction, known as the loss. Generally, the lower the loss, the better the prediction made by the neural network (Koech). The sparse categorical cross-entropy loss function, used in this experiment, is one example of a cross-entropy loss function, defined by the equation below (where L_{CE} refers to the cross-entropy loss):

$$L_{\mathrm{CE}} = -\sum_{i=1}^{n} t_i \log(p_i), \text{ for n classes,}$$

where t_i is the truth label and p_i is the Softmax probability for the i^{th} class.

Equation 2: Mathematical definition of Cross-Entropy. Note the log is calculated to base 2. (Koech)

In the context of this experiment, the value of p_i is calculated by passing the raw values of the output layer nodes through the softmax function (a function that converts the values of nodes in a layer to probabilities) (Koech). However, considering that each data sample in the MNIST dataset has only one label, all but one of the terms in equation 2 would equate to 0, and a simplified equation for the loss specific to this experiment can be constructed, as shown below:

$Loss = -log_2(p)$, where p is the softmax probability for the correct class.

Equation 3: Simplified Loss function used in this experiment.

From this equation, the resulting loss is clearly independent of the accuracy in this experiment, and only describes how sure the neural network is of the correct classification. Hence, it was chosen as one of the metrics in determining the performance of the neural network as it may provide more insight into how the neural network is performing, even when the accuracy remains constant.

3.6 The Experimental Procedure

The neural network was trained three times at each combination of the parameters listed below. After each time being trained, the neural network was evaluated and the accuracy and loss were recorded. In total, this means the neural network was trained at 27 combinations of parameters 3 times, for a total of 81 times.

Training Data Mislabelling Rate: 0%, 25%, 50%

Batch Size: 16, 32, 64

Number of Epochs: 1, 5, 10

4. Experimental Results

4.1 Tabular Data Presentation

The tables below show the Accuracy and Loss for all 3 trials, and the average of all 3 trials, for every combination of number of epochs and batch size. Each table shows the results at different training data mislabelling rates.

0% Training Data mislabelling		Accuracy (out of 1)				Loss			
Epoch Count	Batch Size	Trial 1	Trial 2	Trial 3	Average	Trial 1	Trial 2	Trial 3	Average
1	16	0.9614	0.9625	0.9649	0.9629	0.123	0.1224	0.1192	0.1215
	32	0.9614	0.9587	0.9567	0.9589	0.1354	0.1396	0.1416	0.1389
	64	0.9506	0.9511	0.9552	0.9523	0.172	0.1655	0.1554	0.1643
5	16	0.9773	0.9758	0.9781	0.9771	0.0779	0.0758	0.0745	0.0761
	32	0.9772	0.9779	0.975	0.9767	0.0756	0.0759	0.079	0.0768
	64	0.9763	0.9744	0.9742	0.9750	0.0813	0.0871	0.0826	0.0837
10	16	0.9775	0.9748	0.9779	0.9767	0.0909	0.1054	0.1003	0.0988
	32	0.9776	0.9787	0.977	0.9778	0.0845	0.0811	0.0843	0.0833
	64	0.9777	0.9779	0.9775	0.9777	0.0766	0.0763	0.0778	0.0769

Table 1: 0% Training Data mislabelling Rate Results

25% Training Data mislabelling		Accuracy (out of 1)				Loss			
Epoch Count	Batch Size	Trial 1	Trial 2	Trial 3	Average	Trial 1	Trial 2	Trial 3	Average
1	16	0.9465	0.9461	0.9508	0.9478	0.5449	0.4654	0.4735	0.4946
	32	0.9478	0.9476	0.948	0.9478	0.5251	0.5105	0.5242	0.5199
	64	0.9424	0.945	0.9488	0.9454	0.5303	0.4984	0.4973	0.5086
5	16	0.9615	0.9651	0.9664	0.9643	0.416	0.4343	0.4311	0.4271
	32	0.9683	0.9643	0.9609	0.9645	0.4626	0.4152	0.4405	0.4395
	64	0.9685	0.9608	0.9673	0.9655	0.4179	0.4768	0.3955	0.4301
10	16	0.9581	0.9576	0.9571	0.9576	0.4084	0.4516	0.4046	0.4215
	32	0.9617	0.9612	0.9584	0.9604	0.4014	0.3914	0.4128	0.4019
	64	0.9651	0.9618	0.9613	0.9627	0.4355	0.4293	0.4415	0.4354

Table 2: 25% Training Data mislabelling Rate Results

50% Training Data mislabelling		Accuracy (out of 1)				Loss			
Epoch Count	Batch Size	Trial 1	Trial 2	Trial 3	Average	Trial 1	Trial 2	Trial 3	Average
1	16	0.9341	0.921	0.926	0.927	0.9053	0.9777	0.9553	0.9461
	32	0.9166	0.922	0.9347	0.9244	1.017	0.9593	0.9094	0.9619
	64	0.9284	0.9237	0.9311	0.9277	1.0124	0.9423	1.016	0.9902
5	16	0.9449	0.943	0.9395	0.9425	0.9122	0.905	0.8871	0.9014
	32	0.9491	0.9372	0.9458	0.944	0.838	0.9069	0.8716	0.8722
	64	0.9468	0.947	0.9537	0.9492	0.8405	0.8727	0.8516	0.8549
10	16	0.9342	0.9275	0.9321	0.9313	0.8209	0.8606	0.8674	0.8496
	32	0.9366	0.925	0.9328	0.9315	0.8223	0.8991	0.8682	0.8632
	64	0.9394	0.9403	0.9382	0.9393	0.8731	0.8028	0.8143	0.8301

Table 3: 50% Training Data mislabelling Rate Results

4.2 Graphical Data Presentation

The experimental data below is presented graphically so as to visually illustrate the trends in the data, and more easily come to conclusions on how the independent variables affect the dependent variables of this experiment.

In each bar chart, 9 bars representing the 9 possible combinations of batch size and epoch size are presented, with the height of the bar chart either signifying the accuracy or the loss of the neural network with each combination of parameters. The 9 bars are separated into 3 clusters with different numbers of epochs, as labelled underneath the bars. In addition, the 3 bars in each cluster correspond to different batch sizes, as detailed by the legend above the bar chart. The data in the graphs are the averaged values presented in the tables above.

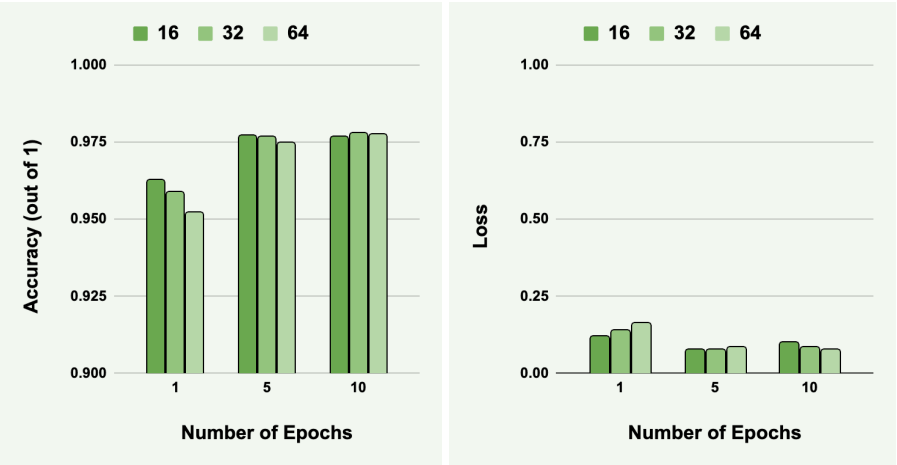
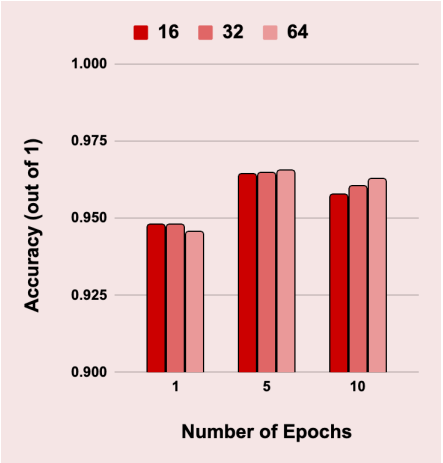


Figure 4 (Left): Accuracy of neural network given 0% mislabelled training data

Figure 5 (Right): Loss of neural network given 0% mislabelled training data



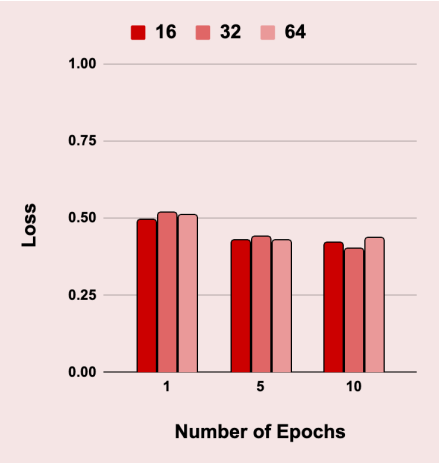
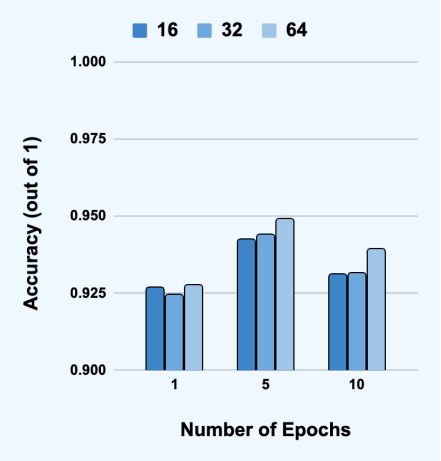


Figure 6 (Left): Accuracy of neural network given 25% mislabelled training data

Figure 7 (Right): Loss of neural network given 25% mislabelled training data



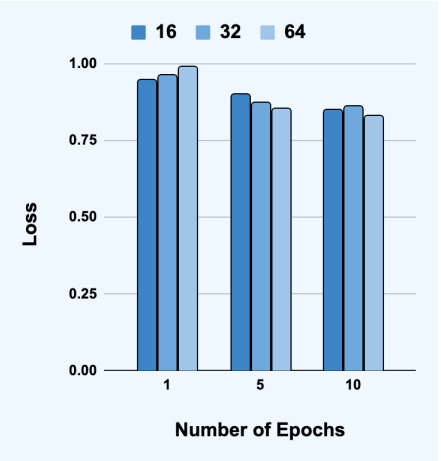


Figure 8 (Left): Accuracy of neural network given 50% mislabelled training data

Figure 9 (Right): Loss of neural network given 50% mislabelled training data

4.3 Data Analysis

Firstly, the results clearly indicate that the mislabelling rate of the training data provided to the neural network has the largest impact on the accuracy and loss of the neural network. The neural network always performs better at lower rates of training data mislabelling, meaning

the accuracy is always higher and the loss is always lower in the neural network trained with lower rates of data mislabelling when all other parameters are kept constant.

As explained in the theoretical background, training a neural network for a greater number of epochs generally increases accuracy and decreases loss. However, not only would each pass through the training data bring diminishing returns to the accuracy and loss of the neural network, training the neural network for too many epochs may cause the neural network to overfit to the training data and become less effective at classifying unseen testing data. Although this is present when the neural network is trained on 0% mislabelled training data, it is most clearly demonstrated when the neural network is trained on 25% and 50% mislabelled training data. This is because the neural network memorises the label noise present in the training data, thus fitting the noisy training data but becoming unable to classify the unseen cleaner testing data as effectively. Basically, the neural network is more prone to overfitting when trained with training data containing information irrelevant or inapplicable to the general testing data, such as in mislabelled training data, as it is able to learn the noise present in the training data but missing in the testing data (Richter).

Generally, as the accuracy of a neural network decreases, it can be expected that the loss would increase. Although this is supported on a rough scale by the data, the above statement is simply a generalisation rather than an actual rule. Thus, counterintuitively, the loss of the neural network continues to decrease as the neural network overfits to the training data. This is likely due to the neural network becoming more confident in the correct answer even while occasionally being even more confident in other answers, meaning that although the neural network is wrongly classifying the testing data in these instances, it still seems to be correlating the testing data with the correct answer.

When the training data mislabelling rates and number of epochs are low during the training of a neural network, increasing the batch size appears to decrease the accuracy and increase the loss of the neural network. However, this relationship appears to swap as the training data mislabelling rate and number of epochs increases. For example, when the neural network was trained for 1 epoch and with training data mislabelled at 0%, the accuracy fell and loss rose as the batch size increased. However, when the neural network was trained for 10 epochs and with training data mislabelled at 50%, the accuracy fell and loss rose as the batch size decreased instead.

For a neural network trained with training data that is 0% mislabelled, having smaller batch sizes would allow the neural network to train more thoroughly on each piece of data, and thus would be more accurate and have less loss. However, under different parameters where the neural network is prone to being overfit to the training data, having larger batch sizes can actually benefit the neural network. When the batch sizes are larger, the incorrect loss of wrongly labelled testing data would be averaged out by the loss of correctly labelled data samples, and thus the impact that each wrongly labelled testing data would have on the backpropagation step would be minimised. Thus, having larger batch sizes can actually allow the neural network to be more resistant to mislabelled training data.

5. Limitations to Investigation

The limitations to my investigation include the following:

1. Due to limited time and computational resources, only 3 values for each independent variable could be included in my experiment. As only 3 data points is not enough to reveal a full relationship between these variables, this limited the analysis that could

be made on the impact that the 3 independent variables had on the 2 dependent variables.

2. A component with a likely bigger impact on the performance of a neural network at varying levels of training data misclassification is the structure of the neural network itself, and it would be interesting to investigate how the structure of a neural network could be reorganised to produce the best results in a situation where training data available is mislabelled at a high rate. This would reveal more about neural networks that can be designed to combat mislabelled training data and would benefit research into implementing neural networks in environments where available data is mislabelled at a high rate.

6. Conclusion

Through the experimental procedure in this paper, the performance of neural networks trained with training data at varying levels of mislabelling were analysed. Specifically, the effect that the batch size and number of epochs had on the effectiveness of neural networks under such conditions were ascertained. Overall, the performance of a neural network is shown to be highly dependent on the batch size and number of epochs at all rates of training data mislabelling. The results of the experimental procedure were used to back explanations to this statement.

The experimental results show that the effect both the epoch count and batch size have in the training of a neural network depends on the mislabelling rate of the training data, and that these two parameters can be adjusted according to the prominence of mislabelled data in a set of training data so as to ensure the neural network functions as effectively as realistically attainable given the structure of the neural network.

Firstly, it was shown that neural networks trained with data at higher rates of mislabelling are more prone to overfitting. Thus, the issue of overfitting should be considered more carefully when available data for training a neural network is mislabelled at a high rate, such as by reducing the number of epochs a neural network is trained for.

In addition, the results show that the effect batch sizes have on the performance of a neural network depends on the mislabelling rate of the training data. For training data labelled accurately, decreasing the batch size would likely bring a positive benefit to a neural network's performance. However, for training data mislabelled at higher rates, decreasing the batch size would likely decrease the performance of the neural network instead.

Hopefully the insight from this paper can help guide the design choices of neural networks intended to be used when available data has a high rate of mislabelling, enabling neural networks to be implemented into a wider array of fields.

7. Works Cited

Brownlee, Jason. "A Gentle Introduction to the Rectified Linear Unit (ReLU)." *Machine Learning Mastery*, 20 Aug. 2020, machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/.

Elliott, Vittoria. "Training Self-driving Cars for $1 an Hour." *Rest of World*, 3 Aug. 2021, restofworld.org/2021/self-driving-cars-outsourcing/.

IBM Cloud Education. "What Are Neural Networks?" *IBM - United States*, 17 Aug. 2020, www.ibm.com/cloud/learn/neural-networks.

---. "What is Overfitting?" *IBM - United States*, 3 Mar. 2021, www.ibm.com/cloud/learn/overfitting.

---. "What is Supervised Learning?" *IBM - United States*, 19 Aug. 2020, www.ibm.com/cloud/learn/supervised-learning.

Kerimbaev, Bolot. "Neural Network Diagram." Figure. *https://bignerdranch.com/blog/neural-networks-in-ios-10-and-macos/*, 28 June 2016, 4bj5ozxzwtu3i1od01ulxmj1-wpengine.netdna-ssl.com/assets/img/blog/2016/06/neural_network_diagram.png.

Koech, Kiprono E. "Cross-Entropy Loss Function." *Medium*, 3 Oct. 2020, towardsdatascience.com/cross-entropy-loss-function-f38c4ec8643e.

LeCun, Yann, et al. "MNIST Handwritten Digit Database." *Yann LeCun's Home Page*, yann.lecun.com/exdb/mnist/.

Richter, Till. "Data Noise and Label Noise in Machine Learning." *Medium*, 1 July 2021, towardsdatascience.com/data-noise-and-label-noise-in-machine-learning-98c8a3c8322e.

Riedmiller, Martin. "Advanced supervised learning in multi-layer perceptrons — From backpropagation to adaptive learning algorithms." *Computer Standards & Interfaces*, vol. 16, no. 3, 1994, pp. 265-278.

Schmidhuber, Jürgen. "Deep learning in neural networks: An overview." *Neural Networks*, vol. 61, 2015, pp. 85-117.

Sharma, Sagar. "Epoch Vs Batch Size Vs Iterations." *Medium*, 23 Sept. 2017, towardsdatascience.com/epoch-vs-iterations-vs-batch-size-4dfb9c7ce9c9.

Steppan, Josef. "MnistExamples." Graph. *Wikipedia*, 14 Dec. 2017, commons.wikimedia.org/wiki/File:MnistExamples.png.

Sycorax. "Why Do We Use ReLU in Neural Networks and How Do We Use It?" *Stack Exchange*, 2 Aug. 2016, stats.stackexchange.com/questions/226923/why-do-we-use-relu-in-neural-networks-and-how-do-we-use-it. Accessed 5 Nov. 2021.

Tensorflow. "Basic Classification: Classify Images of Clothing." *TensorFlow*, www.tensorflow.org/tutorials/keras/classification. Accessed 7 Nov. 2021.

Vincent, James. "The Biggest Headache in Machine Learning? Cleaning Dirty Data off the Spreadsheets." *The Verge*, 1 Nov. 2017,

www.theverge.com/2017/11/1/16589246/machine-learning-data-science-dirty-data-kaggle-survey-2017.

Zeng, Min, et al. "Effective prediction of three common diseases by combining SMOTE with Tomek links technique for imbalanced medical data." *2016 IEEE International Conference of Online Analysis and Computing Science (ICOACS)*, 2016.

8. Appendix

8.1 Code for Neural Network

The following code was used to generate, train, and evaluate the neural network in my experiment, as well as randomise a portion of the MNIST dataset training labels. It makes use of the tensorflow library, a free and open-source software library for machine learning and artificial intelligence. This particular code was adapted from code available on the tensor flow website (Tensorflow).

```
#imports the required modules
import tensorflow as tf
import random

#imports the dataset and normalises pixel values to between 0 and 1.
mnist = tf.keras.datasets.mnist
(training_images, training_labels), (testing_images, testing_labels) =
mnist.load_data()
training_images, testing_images = training_images / 255.0, testing_images /
255.0

#the independent variables changed throughout the experiment
epochs = 5
batch_size = 32
misclassified = 0.5

#randomly alters some of the training labels according to the misclassified
rate
mislabelled_labels = training_labels.copy()
```

```
randomlist = random.sample(range(60000), int(misclassified * 60000))
for x in randomlist:
    new_label = random.randint(0, 9)
    while new_label == mislabelled_labels[x]:
        new_label = random.randint(0, 9)
    mislabelled_labels[x] = new_label

#creates the neural network structure
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(10)
])

#initialises the training parameters the neural network will use
model.compile(optimizer='adam',
loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
            metrics=['accuracy'])

#trains the neural network on the training data
model.fit(training_images, mislabelled_labels, epochs=epochs,
batch_size=batch_size, verbose=0)

#evaluates the accuracy and loss of the neural network
loss, accuracy = model.evaluate(testing_images,  testing_labels, verbose=0)

#outputs the values measured from the experiment
print("Accuracy: " + str(accuracy))
print("Loss: " + str(loss))
```