REDUCTION OF NEURAL NETWORK COMPLEXITY THROUGH STOCHASTIC NUMBERS

James Ridey

Bachelor of Engineering
Software Engineering

School of Engineering
Macquarie University

June 7th, 2019

Supervisor: Associate Professor Tara Hamilton

ACKNOWLEDGMENTS

I would like to acknowledge and thank, Tara Hamilton and Alan Kan for their support and advice during my thesis.

STATEMENT OF CANDIDATE

I, James Ridey, declare that this report, submitted as part of the requirement for the award of Bachelor of Engineering in the School of Engineering, Macquarie University, is entirely my own work unless otherwise referenced or acknowledged. This document has not been submitted for qualification or assessment at any academic institution.

Student's Name: James Ridey

Student's Signature: James Ridey

Date: 7/6/2019

ABSTRACT

Neural networks considered a core foundation of machine learning, has recently ballooned into a large framework that requires vast amounts of computing resources and time. Here, we examine how to improve these networks, hoping to achieve networks that can calculate the same result with less training time and/or resources using ideas such as stochastic numbers and probabilistic results.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Neural networks: introduction

Neural networks are the core structure of what many believe to be the next step in computing, machine learning. We are starting to see machine learning used to solve problems that originally were only able to be solved by people. However, all this machine learning comes at a cost, machine learning needs vast amounts of data to learn and thus large amounts of computing resources, electricity, time and even the best machine learning models work within a narrow, controlled set of inputs, anything outside this ”range” are misclassified, in other words, machine learnings models are sensitive to noise and not general enough.

My thesis details, my efforts to try and find a way to reduce machine learning complexity or finding a way to make machine learning models more tolerant to noise. The preliminary work done in this thesis relates to trying to use stochastic numbers with the Tensorflow machine learning framework on the MNIST dataset to try and find ways to reduce complexity.

Chapter 2

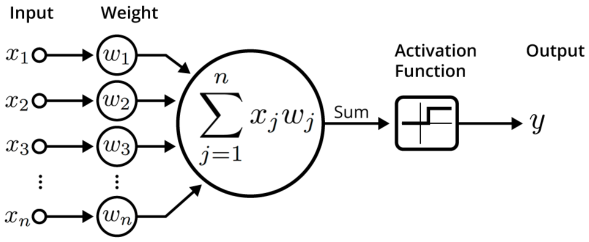
Background and Related Work

2.1 Neural networks: explanation

Neural networks are a very rough simulation of how the neurons in a human brain actually work, they have been researched since the 1960s [1] but at the time we did not have the computing power necessary to run these large neural networks and thus progress was halted until around such time as 1970-1980's when backpropagation was discovered making neural network training much more efficient and parallel computing system was also starting to kick off.

The most common form of a neural network in the feedforward model where the inputs are passed to the neural network and it outputs an answer. The feedforward model consists of an input layer, hidden layers and an output layer each of these layers are composed of varying amounts of neurons depending on the complexity of the data that is trying to be learned. More models of neural networks exist which aim to memorise or improve the feedforward model, however, the feedforward model is incredibly ubiquitous and is the main driving force behind machine learning in the industry.

To understand what each layer in a neural network is doing, it is first necessary to understand what a neuron is doing. A neuron is comprised of inputs, a set of weights, one output and an activation function, as shown below.



An illustration of an artificial neuron. Source: Becoming Human.

Figure 2.1: Neuron model.

Neurons are fed a set of data through its inputs which are multiplied by a specific weight, all of these numbers are then added up and fed to an activation function. An activation function is a function that maps the input to a value between 0 and 1, a commonly used activation function is the sigmoid function, numbers in the positive infinity direction become increasingly closer to 1, while numbers in the negative infinity direction become increasing closer to 0.

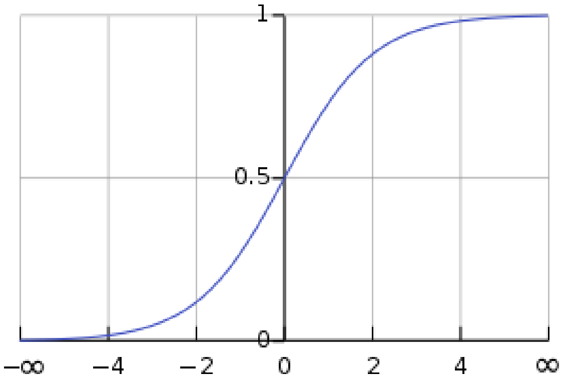


Figure 2.2: Sigmoid graph.

Next is neural network layers varying based on the task you want to try an accomplish, some examples include convolutional layers which attempt to use neural networks on squares of pixels, dropout layers which randomly pick weights to drop to try and reduce overfitting and the most common layer used to most machine learning models, is the dense layer, in this layer every neuron connects to every other neuron in the previous layer.

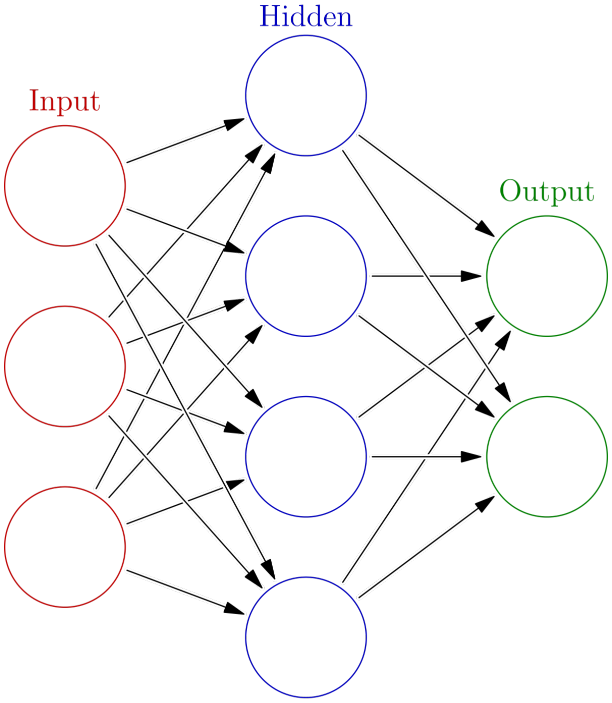


Figure 2.3: Neural network model.

Each circle in this figure represents a neuron and each line in this figure represents a connection between 2 neurons. By manipulating the weights in the neural network, we change what each neuron calculates and outputs as part of its activation function. In doing so, we now have a framework of which for any given input, the neural network can generate any output (given the right weights), regardless of the complexity of the output [2].

2.1.1 Training

The next question about neural networks is then how are the weights calculated for a given input. This is where training a neural network comes into play. The basic summary for how a neural network is trained is they are fed the input data, the error amount is calculated from the neural network output and then the weights are slowly adjusted to minimize the error rate. How this is all done is where the complexity of neural networks lie and why they take so long to train. One of the simplest algorithms that implements this is the stochastic gradient descent algorithm, which is a core component of many other machine learning algorithms like RMSprop, ADAM and AdaGrad (RMSprop being a commonly used algorithm and the one that was used in the majority of the MNIST tests).

The gradient descent or GD works by measuring the gradient and trying to find a point at which the error is the lowest. The gradient being a measure of ”how much the output of a function changes if you change the inputs a little bit" [3]. The best way of explaining GD is with a hill climbing example, a blinded person is trying to find the highest point of a hill and so he starts by taking large steps to estimate where the steepest part of the hill is and then smaller and smaller steps until he finds the top of the hill. Applying this to machine learning, the error of the output of the machine learning model is compared with the actual values that are to be expected and this is the error amount, then the weights are changed are relatively "large" amount and the error is measured again and compared to the previous error, this is the gradient. This gradient is then used to estimate how big of a "step"/how much to adjust the weights by on the next iteration.

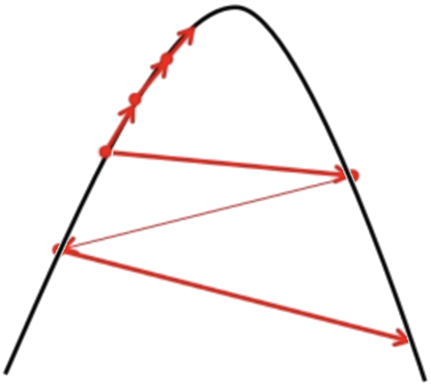


Figure 2.4: Stochastic gradient example. Y-Axis represents accuracy rate

However in realities, there are much more dimensions to a machine learning model and secondly, there is not just one point that is the highest but multiple hills of varying heights, this is one of the many reasons why machine learning is hard. You can create simple machine learning algorithms that find a maximum point but not the global maximum point or the global maxima, you can also try and adjust how much you step by each amount but the best way, which requires all the computation power is a small stepping amount which tries all points on the "hills".

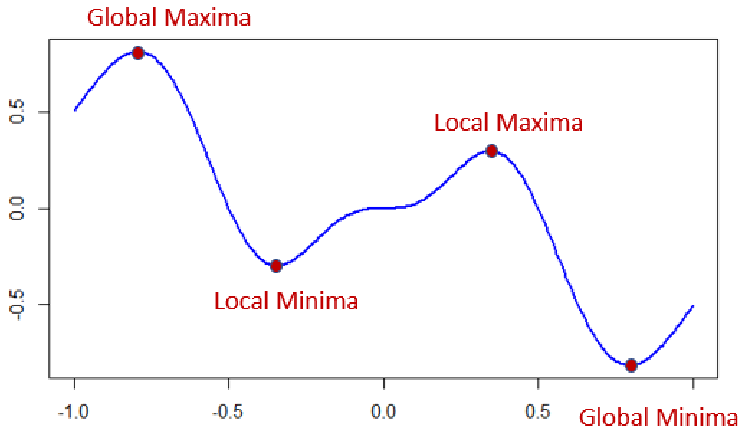


Figure 2.5: Global maxima example. Y-Axis represents accuracy rate

2.2 Stochastic computing

Stochastic computing is a branch of computing that relies on probabilistic bit streams to calculate numbers, often relying on simple bit-wise operations. Every number in a stochastic computer is arranged as a probability of the ones and zeros in a continuous binary number stream.

2.2.1 Advantages of stochastic numbers

There are 2 major advantages with this technique, one is that the system is extremely resistant to noise, flipping multiple bits in the stream will have very little effect or will only change the number minorly. The other major advantage is that certain operations have a simpler solution, for example in terms of multiplication on a conventional computer this would be an n^2 operation, however, on a stochastic computer, this is accomplished with an AND gate at a cost of n.

2.2.2 Disadvantages of stochastic numbers

However, the reason for stochastic computing not being proliferated is the inherent weakness which is the randomness of stochastic numbers. Stochastic numbers can never truly represent the number they are trying to calculate and thus need a large amount of sampling to ”zone in” on the number being calculated.

Secondly is that while stochastic numbers have a simple analog for multiplication, the core of stochastic numbers like the PRNG (Pseudo-random number generator) for generating the random bit stream and decoding the bit stream by sampling do not have simple gate-level analogues and this is where stochastic numbers lose their advantage.

2.3 Tensorflow

Tensorflow [4] is a machine learning framework primarily written in Python. It was released by Google under the Apache License 2.0 on November 9th, 2015. The Tensorflow network employs the idea of creating a neural network model which is then compiled before being handed off to the appropriate processing unit be it CPU, GPU or TPU, this is in contrast to other machine learning frameworks like PyTorch that generate their models dynamically on the fly which results in a small performance hit.

Tensorflow continues to grow in support with new frameworks like Kera that aim to simplify common operations and overall streamline the process and show no signs of stopping with a stable release only 2 months ago.

2.4 MNIST dataset

The MNIST dataset [5] is a commonly used dataset for machine learning that consists of 70,000 images of handwritten digits (60,000 training images and 10,000 test images) and the associated labels for each of these images. The dataset is commonly chosen due to its simplicity and many frameworks support it out of the box.

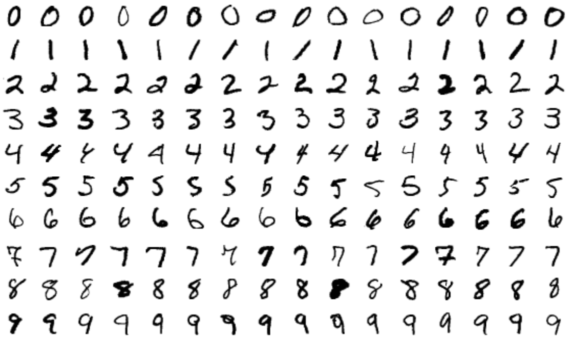


Figure 2.6: Samples of the MNIST dataset.

Another common reason that MNIST is chosen is to demonstrate the accuracy of neural networks over more traditional statistical analysis such as linear classifiers and K-nearest neighbours, both of which score around an error rate of 12% and 5% respectively [6] as compared to a simple neural network of 3 layers which can score around 3.05% [6].

However this dataset is not without its faults, there are quite a number of images that are dubiously labelled.

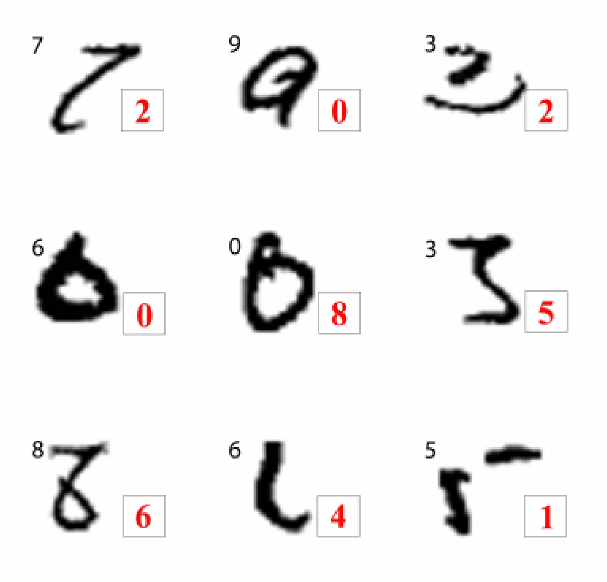


Figure 2.7: Samples of ambiguous digits.

The dataset is also often constructed as too small and too simple, consisting of only 70,000 images of greyscale images. Compared to datasets such as CIFAR10 which has the same amount of images but 32x32 colour images or another dataset like ImageNet consisting of 1,331,167 images, MNIST is certainly behind. However, MNIST has been used and is continued to be used as a way to quickly validate a machine learning algorithm before moving on to larger more complicated datasets that take much more time to compute.

Chapter 3

Preliminary results

3.1 Stochastic numbers

3.1.1 Stochastic numbers: Abstract

These preliminary results are based upon the idea of if we could use stochastic numbers as a way of improving neural networks, whether that is through the reduction or datasets, noise tolerance, computation complexity or anything that would give it some edge over a more traditional network. The idea mainly comes from spiking neural networks and more theories about how the human brain works on probabilistic methods rather than hardcoded logic.

All of these results were trained with the MNIST dataset, 20 repetitions using a normal model with the following structure.

1. Dense input layer
2. Dropout layer (0.2 probability)
3. Dense layer (RELU activation)
4. Dropout layer (0.2 probability)
5. Dense layer (Softmax activation)

The stochastic model was an extension of the normal model but the first layer was the custom Stochastic layer.

1. Custom Stochastic layer
2. Dense input layer
3. Dense layer (RELU activation)
4. Dense layer (Softmax activation)

TODO, talk about results of stochastic numbers compared to the normal model

3.1.2 Bit size

The following graph show the accuracy results when the bit size of the stochastic layer was modified.

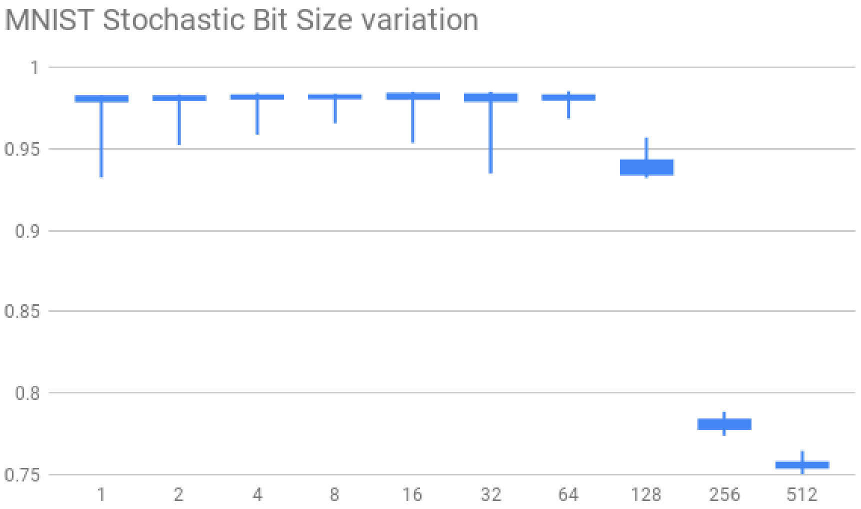


Figure 3.1: Bit size results.

From this test we can see that there is not much difference when I start increasing the number of bits, so for the rest of these tests, they were done with 8 bits.

However interestingly to note that above 128 bits we start seeing a large drop in accuracy, I'm unsure of the reason behind this.

3.1.3 Noise tests

The following graph shows the accuracy results when a normal model and a stochastic model are trained with the MNIST dataset and then fed the testing dataset from the Noisy MNIST dataset.

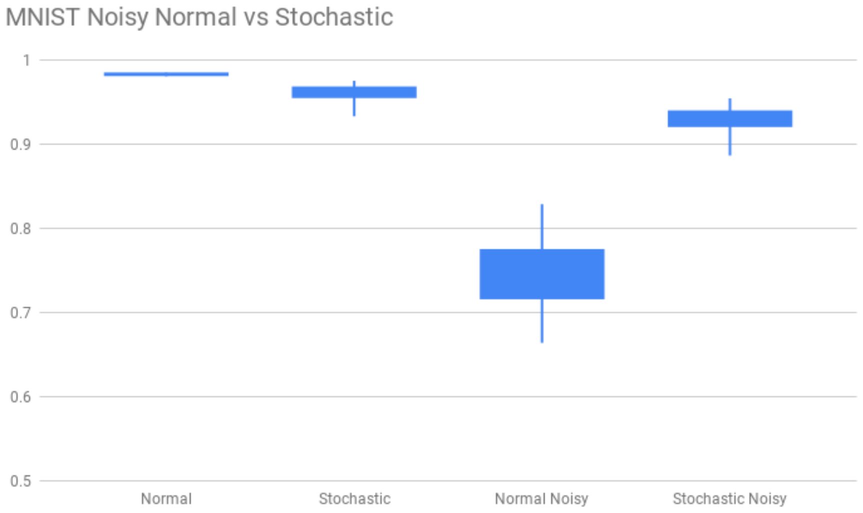


Figure 3.2: Noise test results.

In this test, we can see compared to the normal, the stochastic model loses 0.002% in accuracy. However when both of these pre-trained models are fed the noisy MNIST dataset, the stochastic model fairs considerably better.

However, upon further consideration, this seems to be an artifact of the dataset, after some investigation, there was a slight mistake in the custom stochastic layer which meant that all the values in the training and testing datasets for the stochastic layer were inverted. This should not have affected the results, comparing the images below shows an image that has been passed through the stochastic layer and both the mistake and fixed version shows minor differences other than the inversion.

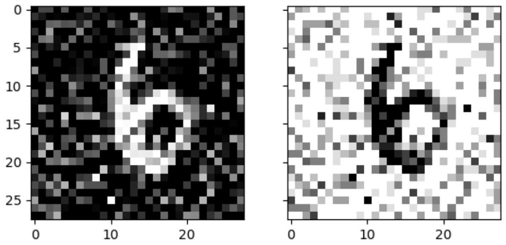


Figure 3.3: Left is original, right is inverted stochastic layer result.

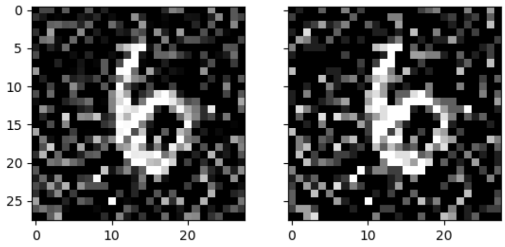


Figure 3.4: Left is original, right is corrected stochastic layer result.

After the correction, the advantage disappears.

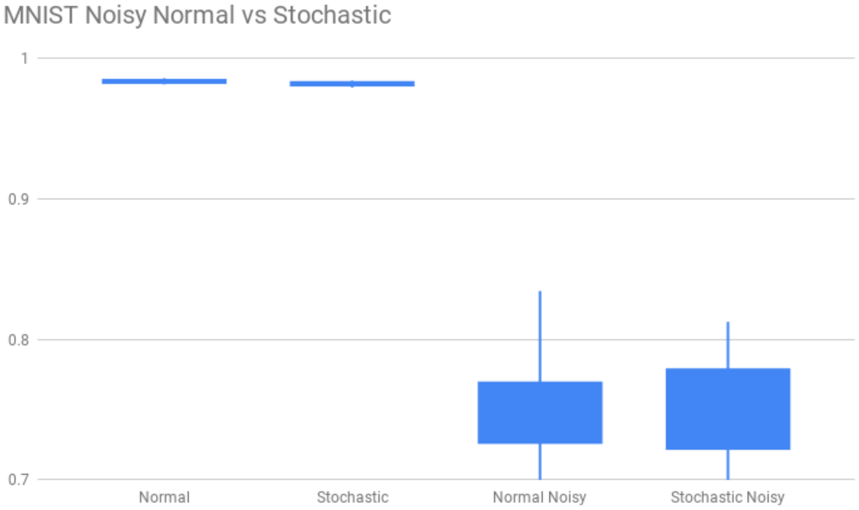


Figure 3.5: Corrected noise test results.

This is further shown by inverting the test and training data for the normal model, this now shows that the results have flipped and that this result is an artefact of the dataset rather than the stochastic layer.

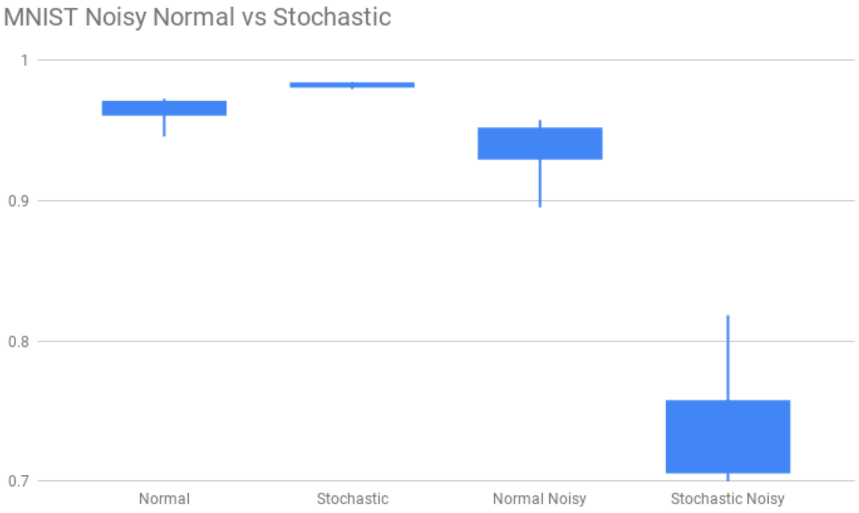


Figure 3.6: Inverted noise test results.

3.1.4 Noise shuffle

The following graph shows the accuracy results when we take both the normal dataset and the noisy dataset and shuffle them together, producing a new dataset that contains normal and noisy MNIST images.

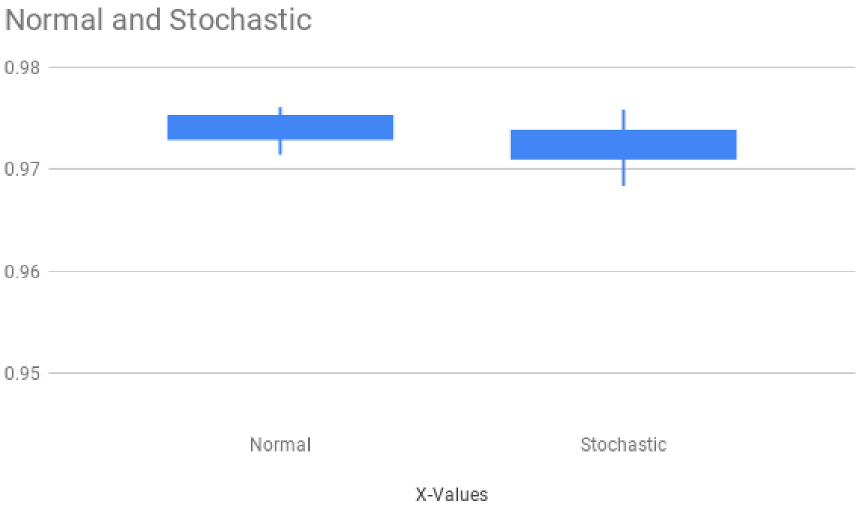


Figure 3.7: Noise shuffle results.

The stochastic model performs similarly to the normal model, nothing much to note here.

3.1.5 Reduced dataset

The following graph shows the accuracy of both the normal model and stochastic model as the size of the training datset is reduced.

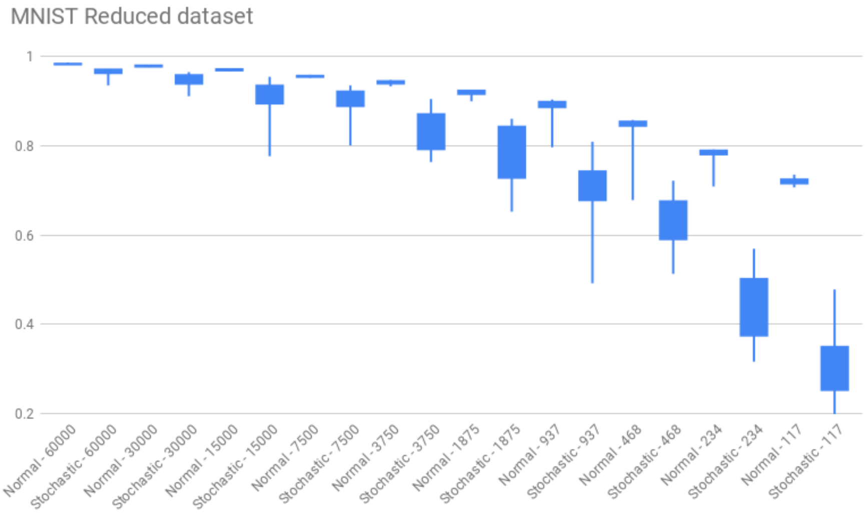


Figure 3.8: Bit size results.

The stochastic model has significantly worse performance as the dataset size decreases.

3.1.6 Reduced Epochs

The following graph shows the accuracy of the test dataset as training epochs are increased.

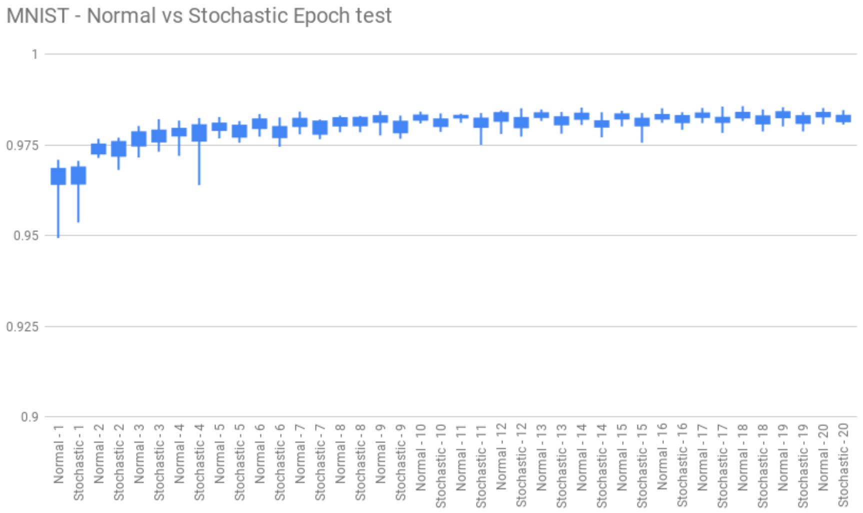


Figure 3.9: Epoch results.

The stochastic model performs similarly to the normal model, nothing much to note here.

3.1.7 Convolution2D

For this test, the model was changed to use a convolution model of the following form

1. Conv2D(32 neurons, (3x3 kernel), RELU activation)
2. MaxPooling2D((2x2 kernel))
3. Conv2D(64 neurons, (3x3 kernel), RELU activation)
4. MaxPooling2D((2x2 kernel))
5. Conv2D(64 neurons, (3x3 kernel), RELU activation)
6. Flatten layer
7. Dense layer (64 neurons, RELU activation)
8. Dense layer (10 neurons)

With the stochastic model being

1. StochasticLayer(728 neurons)
2. Reshape
3. Conv2D(32 neurons, (3x3 kernel), RELU activation)

4. MaxPooling2D((2x2 kernel))
5. Conv2D(64 neurons, (3x3 kernel), RELU activation)
6. MaxPooling2D((2x2 kernel))
7. Conv2D(64 neurons, (3x3 kernel), RELU activation)
8. Flatten layer
9. Dense layer (64 neurons, RELU activation)
10. Dense layer (10 neurons)

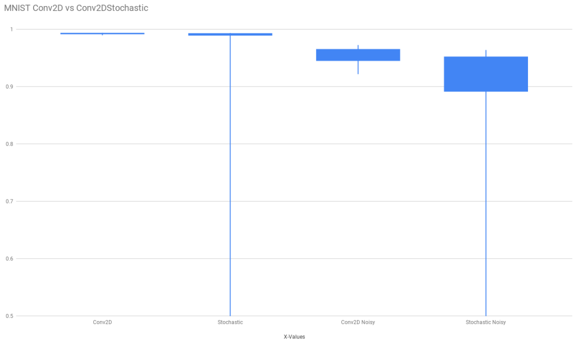


Figure 3.10: Conv2D results.

The stochastic model performs similarly to the normal model, nothing much to note here.

3.1.8 Dense layer size

The following graph shows the accuracy of the test dataset as the size of the hidden dense layers neurons are increased.

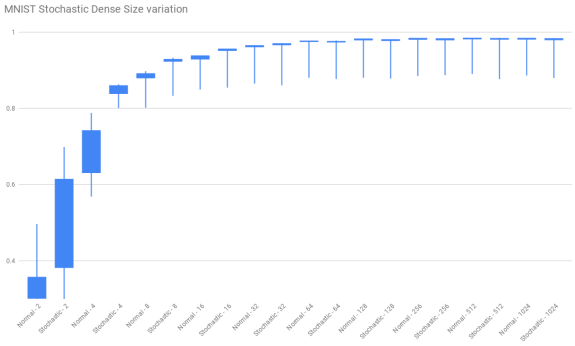


Figure 3.11: Dense layer results.

This is by far the most interesting results so far and thus the results above are 30 samples instead of the usual 20 samples. The results show that for hidden dense layer sizes of 32 and below the stochastic layer fairs much better, however, I'm still unsure if this is because of the stochastic layer or another artefact of the dataset, I will continue to research.

Chapter 4

Conclusions and Future Work

4.1 Conclusions

In conclusion, it seems that the stochastic layer has a minimum impact on the performance of neural networks in all the tests I have done and in some case worse performance.

4.2 Future work

My next steps for this project is to look into alternative ways of trying to use the stochastic network or another representation that might yield better results. It is notable that MNIST may not provide the complexity that I'm looking for, especially after the inverted artefact. My next steps for this is to look into more complicated datasets like CIFAR100 to see if I can find performance in that area.

Chapter 5

Abbreviations

AWGN	Additive White Gaussian Noise
SGD	Stochastic Gradient Descent
SC	Stochastic numbers
NN	Neural network

.1 Codebase and results

https://github.com/AeroX2/stochastic-demo

Bibliography

[1] 2014. [Online]. Available: https://cs.stanford.edu/people/eroberts/courses/soco/projects/neural-networks/History/history1.html

[2] M. A. Nielsen, "Neural networks and deep learning," 2018. [Online]. Available: http://neuralnetworksanddeeplearning.com/chap4.html

[3] N. Donges, "Gradient descent in a nutshell," 2018. [Online]. Available: https://towardsdatascience.com/gradient-descent-in-a-nutshell-eaf8c18212f0

[4] 2015. [Online]. Available: https://www.tensorflow.org/

[5] Y. Lecun, "Mnist handwritten digit database, yann lecun, corinna cortes and chris burges," 2019. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[6] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, November 1998.