MATRIX MULTIPLICATION 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 are considered a core foundation of machine learning and has recently ballooned into a large framework that requires vast amounts of computing resources and time. Here, we examine how to improve the efficiency of these networks, by utilizing ideas such as stochastic numbers and probabilistic results to reduce the overall training time and/or the consumption of computing resources. As such a core part of machine learning networks is a matrix multiplication and implementing this operation in stochastic numbers would be a first step to trying to use stochastic numbers within the domain of machine learning.

Contents

List of Figures

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 [3]. 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.

In my preliminary thesis, I detailed how machine learning models react to noise and shared some of my initial tests, in this final thesis we try to expand and implement a matrix multiplication, a core component of neural networks through the use of stochastic numbers.

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 [4], they have been researched since the 1960s [24] 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 [10], as shown below.

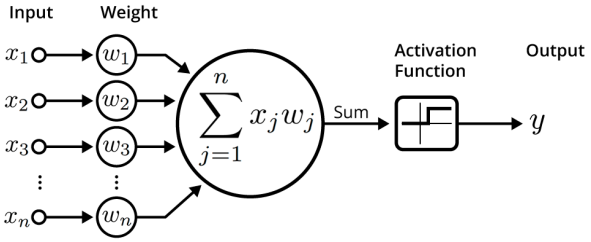

Figure 2.1: Neuron model [20].

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.

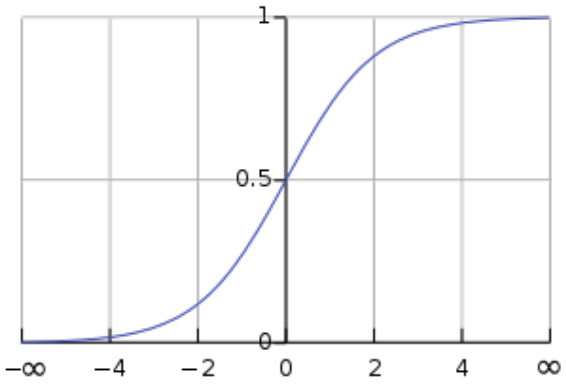

Figure 2.2: Sigmoid graph [21].

The next concept is neural network layers which vary based on the task you want to try and 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.

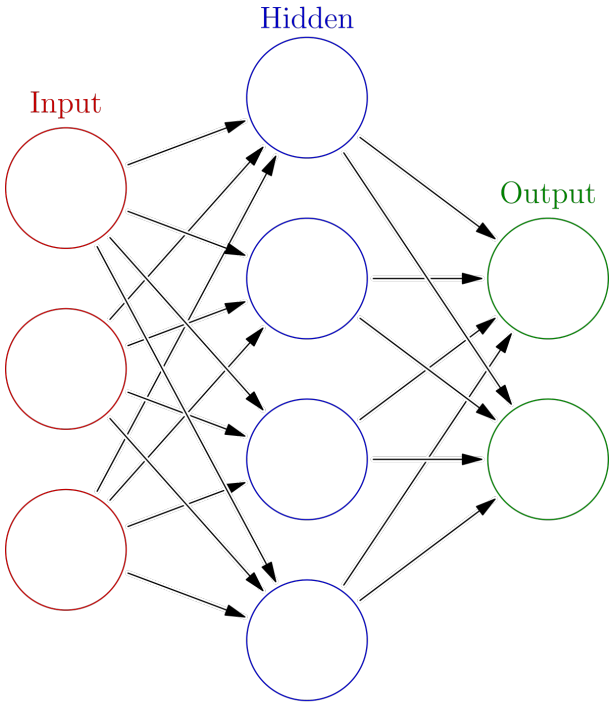

Figure 2.3: Dense layer neural network model [22].

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 [18].

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 difference between the neural network output and the expected output. 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" [6]. The best way of explaining GD is with a hill-climbing example, a blind 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.

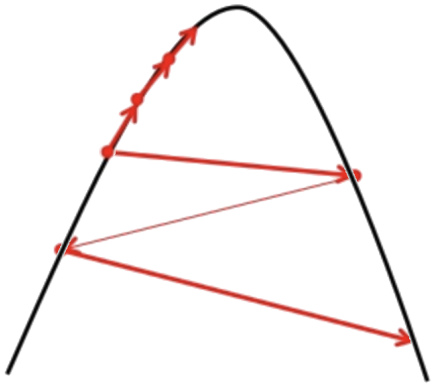

Figure 2.4: Stochastic gradient example. Y-Axis represents accuracy rate

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".

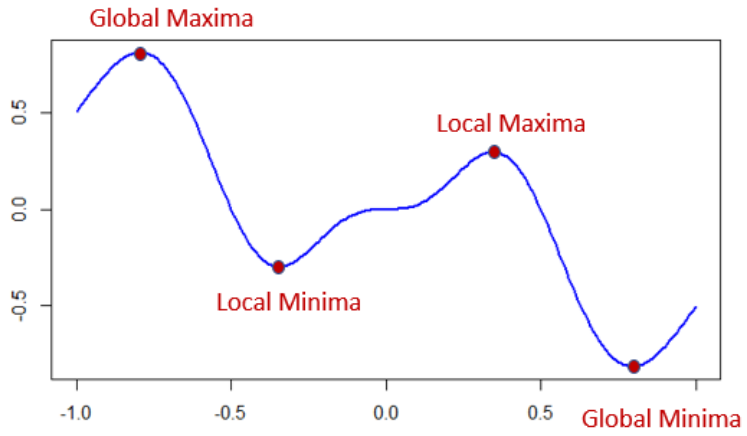

Figure 2.5: Global maxima example. Y-Axis represents accuracy rate

2.2 Complexities of neural networks

To help familiarize readers with neural networks and why a stochastic numbers implementations of a neural network may be beneficial, I've included a brief section about neural network complexities.

2.2.1 Overfitting and underfitting

Neural networks are a statistics tool with the main purpose of being able to generalize from one set of training data and then predict a value based on a new set of input data.

To train a neural network, we divide the training into 2 phases, the learning/training stage and the prediction state. The learning/training stage is where the weights of the neural network are adjusted using backpropagation or a similar algorithm and the neural network is trying to fit the data as well as it can by correctly predicting the correct value for each given input. Then the prediction stage is where the neural network is evaluated against a subset of the data that it has not seen before, this stage is purely to combat overfitting [13].

Overfitting is when the neural network begins to fit so well with the training data, that is unable to generalise/achieve good accuracy in the prediction stage. In other words, it fails as a classification or prediction model because it hasn't really learnt the underlying structure of the data but rather is just "cheating" in the training stage.

Conversely, underfitting occurs when the neural network fails to achieve any accuracy in the training or prediction stage, this is quite rare to see in practice but can occur with bad datasets or a too simplistic neural network model [28].

2.2.2 Complexities

Given overfitting and underfitting, it becomes a challenge to train neural networks because having a model that is too simplistic will cause underfitting and having a model that is too large not only increases computation time with all the various amounts of weights that each layer incurs but also leads to overfitting [11].

Not to mention, the training stage usually requires multiple iterations and the backpropagation algorithm has a high time complexity [9]. All in all, this means we end up with a large neural network, with thousands to millions of weights and due to how neural networks work, most of the operations needed are matrix multiplications which is what GPU's tend to be good at, considering computer graphics are just matrix manipulations.

2.3 Research into neural network complexity

There has been a lot of research in neural networks, while much of it is focused upon learning more and more complex datasets [19] [8] [7], there has also been a lot of research into trying to prune neural networks and a large number of parameters (weights) that are needed for these large scale neural networks. A recent example is "Importance Estimation for Neural Network Pruning" [17], where they cite a 40% reduction in complexity by trimming off 30% of the parameters.

In the field, which I'm planning to research there have been a few papers on stochastic numbers and the benefits regarding using them on neural networks, for example, one such paper states a 47% area and 60% power reduction while maintaining a high accuracy rate [16].

Another paper [5] goes on to show how its stochastic neural network allows for building reliable systems due to the high level of noise-immunity that the network model shows.

While both of these papers are impressive research and they show that their networks are comparable with a regular dense layer neural network or MLP (Multi-Layer Perceptron) for short, they fail to address advantages and disadvantages that a stochastic neural network might bring to the table, an example of such an advantage is how a stochastic neural network compares with a regular MLP network when exposed to a noisy dataset. The previous paper regarding noise immunity is mainly focused around random noise injected alongside the input data, not noise inherent within the input data. One of my research avenues is to explore how stochastic neural networks, react to noise during the training and validation stages.

2.4 Matrices in neural networks

One of the important things to understand about a neural network is how they are implemented, this will help when discussing how stochastic computing could potentially help

with neural networks. The core component of almost all machine learning frameworks and neural networks is the matrix multiplication operation, this is used during the feed-forward operation of the network, computing the output for each of the inputs and the weights of each layer.

Since each layer is composed of neurons which take all the inputs and multiplies it with all of the weights, it is functionally identical to a matrix multiplication where the first matrix is the inputs to the neural network layer, the second matrix is the weights of that layer and the output matrix is the output of the current layer.

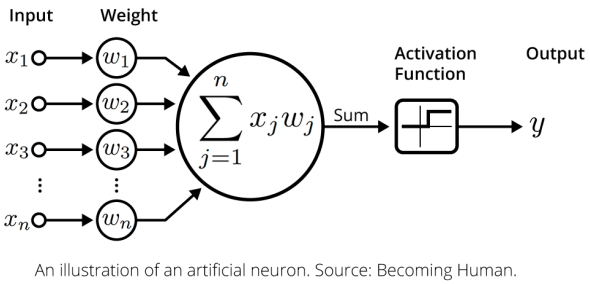

Figure 2.6: Neuron model [20].

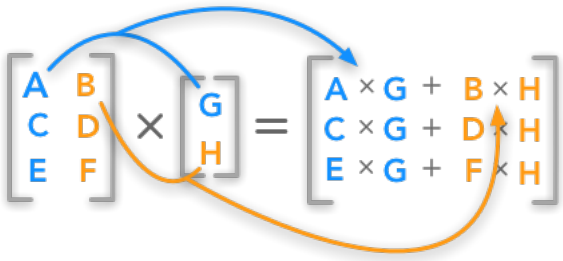

Figure 2.7: Matrix multiplication [29].

What advantage can we get out of neural networks being represented this way? The major advantage is that the component responsible for rendering graphics on the computer, the GPU is extremely efficient at matrix multiplications partially because of the architecture is arranged for bandwidth instead of latency, partially because GPU's have multiple cores that run in parallel and mostly because 3D graphics are all about matrix multiplications to map 3D points to 2D through a camera matrix.

2.5 Stochastic computing

Stochastic computing is a branch of computing that relies on probabilistic bitstreams 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 [1].

2.5.1 Advantages of stochastic numbers

There are 2 major advantages with this technique, one is that the system is extremely resistant to noise [2], 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 a n^2 operation, however, on a stochastic computer, this is accomplished with an AND gate at a cost of $O(n)$.

2.5.2 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 produce a more accurate result, they simply try to "emulate" a floating-point number by using the ratio of 1's and 0's as a probability.

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 bitstream and decoding the bitstream by sampling do not have simple gate-level analogues and this is where stochastic numbers lose their advantage.

Lastly stochastic numbers do not take advantage of all of the states of a binary number. To represent 8 different numbers you need at least 8 bits (n $bits$), as opposed to a normal binary number where you only need 3 bits for 8 numbers or $log_2(n)$ $bits$ of information.

2.6 The Idea

To help understand this concept, it is important to explain the origin of this idea of using stochastic numbers for machine learning. One of the underlying issues with neural networks is that they are slow and are sensitive to noise in the dataset. An idea to speed them up and to make them more noise resistant mainly comes from spiking neural networks. Spiking neural networks are built on the idea of artificial spiking neurons which are meant to be much more analogous to a human neuron, instead of matrix multiplications and activation functions, a spiking neuron is a neuron that takes in a stream of spikes of activity and after a certain threshold of spikes outputs its own spike, creating a so-called spike train. Stochastic numbers act in a similar way, you have spikes (1's) in a bitstream of information and process them in some way to get more or fewer spikes. Stochastic numbers are known to have properties that would make them beneficial in neural networks such properties include noise resistance and faster floating-point operations. So the idea was to see how stochastic numbers would work in a machine learning domain and the first

step to this would be to replicate what a regular neural network model does which is a series of matrix multiplications.

2.7 Tensorflow

Tensorflow [30] 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.

2.8 PyTorch

PyTorch [31] is another machine learning framework that is also written in Python, it is primarily developed by Facebook under a modified BSD license, first released in October 2016. PyTorch offers similar benefits to Tensorflow with easy GPU and CPU optimization as well as premade Dense, RNN and CNN networks, furthermore, as the thesis progressed, it became apparent that Tensorflow was not a suitable framework to do low-level operations such as trying to implement the matrix multiplication with stochastic numbers. PyTorch fixes this issue by working at a much lower level than Tensorflow that allows direct access to its feed-forward and backpropagation components, it also allows users to directly define their networks and how they should be computed.

2.9 Problems implementing stochastic computing in existing machine learning frameworks

Both Tensorflow and PyTorch were originally used to emulated stochastic computing inside of a machine learning system, however, due to complications that arose when trying to implement the stochastic operations this was ultimately scrapped and the results found in this thesis were made using Python 3. More information regarding these complications will be detailed in the following section.

Chapter 3

Implementation

3.1 Introduction

The primary idea behind this thesis was to create a system where we could reliably compare 2 machine learning systems with one using stochastic computing and the other using conventional computing. While attempts have been made in building up entirely new systems from the ground up using stochastic computing to accomplish a machine learning task [12]. As far as the research shows there have been little to no systems that use stochastic computing inside of existing machine learning framework such that both systems are on an even playing field and you aren't comparing oranges and apples. This was the goal we set out to accomplish, creating a system within an existing machine learning framework that could be applied to multiple datasets.

3.2 The design

Considering that the goal of this thesis was to compare stochastic computing and conventional computing in the field of machine learning, the first step would be to implement the core part of a machine learning framework which is the matrix multiplication, this consists of three parts, the conversion from a binary floating-point number to a stochastic number, the multiplication between the rows and columns of the matrix and finally the addition of each of the rows of the matrix. While most of these operations would be easy outside of a machine learning framework, implementing them within a machine learning framework requires a different way of thinking as each operation needs to be parallelizable and as such you are often restricted with what you are "allowed" to do, one such example of this is the heavy discouragement of the use of for loops [27] as used incorrectly this requires the data being processed to be passed between the GPU and CPU and will likely cause massive slowdowns.

Each task had multiple ways of implementing the solution and each came with their

benefits and drawbacks, which is expanded in more detail below.

3.3 The conversion

One of the secondary goals of this project was to create essentially a drop-in machine learning layer that would use stochastic computing, this would mean that the drop-in would have to convert the floating points it had received from the previous layer, convert it to a stochastic number for processing and then convert it back after the computation.

The general operation for how a binary number is converted to a stochastic number is

1. Create a boolean array the size of the stochastic number you want
2. For each boolean in the array, set it to the condition (value $\geq$ random_float(0,1))

This is further shown in the Python code below

```
stochastic_num = [1 if random.random() <= prob1 else 0 for n in range(length)]
```

However in terms of a machine learning framework the following is done instead,

1. Take the matrix of floating points, and add an extra dimension to it
2. Scale the size of the newly created dimensional such that it repeats each value for the size of the stochastic number
3. Apply (value $\geq$ random_float(0,1)) over the entire matrix

This covers the general approach to converting binary numbers to stochastic numbers, converting them back means simply taking the mean of the entire array/matrix. However, while this covers the general approach some nuances need to be discussed. The first is the use of the random number generator, stochastic numbers are very sensitive to correlation [14] and so each random number must be generated separately from the other, while a general pseudorandom number generator (PRNG) will certainly do the trick, there have been studies showing better results can be achieved with a customised PRNG [15], we decided against going down this route as it would be complicated to integrate and ultimately detract from the goal of this thesis.

Another nuance of this algorithm is the size of the stochastic number, having more bits in your stochastic number will ultimately improve its performance but it comes at the cost of memory (which can balloon very quickly with large matrixes) and computational time. Plus after further investigation, we found that increasing the number of bits has diminishing effects.

As shown in the graph above, as the bits increase, the variance decreases but it reaches a point where the number of bits accounts for such a small change in the variance that it is ultimately not worth the memory or time. We did some research to try and find a way to have progressive accuracy with stochastic computing but we unable to find

any instances of such a case. And finally the last nuance is the range of the stochastic numbers as the conditional (value $\geq$ random_float(0,1)) will give a range on the stochastic numbers between 0 and 1, there exists an alternative form for negative numbers where the conditional becomes (value+1/2 $\geq$ random_float(0,1) essentially the -1 to 1 range of value is mapped between 0 and 1.

However there are a few caveats with this, one is that now each bit of the stochastic number represents a larger range and thus you need a large number of bits to accurately represent a number with a large number of digits after the decimal point, the second problem is that numbers can become small enough that when represented in a stochastic format they essentially become zero and no longer have an impact on the final value when they might have had a small but meaningful impact on the final value if they were not being computed stochastically.

3.4 Format

For stochastic numbers, there are 2 main formats in which they can be represented with each format changing how results are calculated. The first most common format is the unipolar format in which each bit in the stochastic bitstream has a probability of $\frac{1}{number\ being\ represented}$ this in turn only allows the stochastic bitstream to have a range from 0 to 1, with 0 being where all the bits are 0's and 1 being where the entire bitstream is composed of 1's. As a quick example $X = SN1000111010111001$, this stochastic bitstream would represent the number $X = 0.5625$ as there are 9 one's with a length of 16 ($\frac{9}{16} = 0.5625$)

The other format is the bipolar format in which each bit in the bitstream is represented by the probability of $\frac{number\ being\ represented-1}{2}$, the advantage with this format is the range is expanded out to (-1,1). While in a unipolar format each 1 and 0 represent a positive range, zero's represent -1 and one's represent 1, this means that a unipolar stochastic bitstream that is equal to 0 is composed of half 1's and half 0's. As example of bipolar bitstream, $X = BSN1100110100000100$, this stochastic bitstream would represent the number $X = -0.25$ as there are 6 one's and 10 zero's with the length being 16 ($\frac{(+6)+(-10)}{16} = -0.25$)

While each format can easily be multiplied to a larger range once converted back to a floating-point number, what matters is how certain operations like multiplication differ between unipolar and bipolar. In multiplication for unipolar stochastic numbers, we use the AND gate and for bipolar stochastic numbers, the XNOR gate is used.

3.5 Multiplication

The multiplication of stochastic numbers is probably the easiest operation to implement, the most reliable and what attracted us to look into stochastic computing itself. Multiplication of stochastic numbers is implemented in one of two ways, using an AND gate or

the XNOR gate depending on the format.

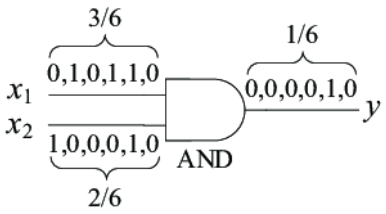

Figure 3.1: Stochastic unipolar multiplication [25].

The above example shows a stochastic unipolar multiplication, the AND gate produces a 1 if both inputs contain a 1 else the output is 0. As shown in the example above in the 5th position both bitstream (x_1) and (x_2) contain a 1 so the output for the 5th position is also a 1.

The next example is a multiplication with the bipolar format using a XNOR gate which produces a 1 if both the inputs are the same ie (0 and 0) or (1 and 1) otherwise it produces a 0.

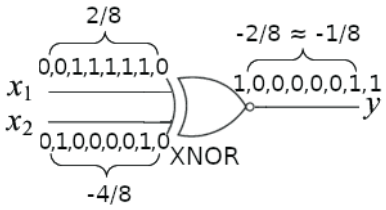

Figure 3.2: Stochastic bipolar multiplication.

3.6 Addition

The last step of matrix multiplication is the addition, but this is the part where things didn't work out. So the first thing about stochastic addition is you cannot represent a number larger than what your stochastic range is, for example this means that you cannot represent a number larger than 1 if you are in a bipolar/unipolar format this means you have two forms of addition, saturating and non-saturating, in saturating addition it works similarly to normal addition with the exception that you cannot go over or under the limit (0/1 for unipolar, -1/1 for bipolar). In non-saturating addition, the resulting addition is divided by 2, this ensures that the stochastic number always stays within range.

However, the 2 caveats of stochastic addition lead to problems when trying to do matrix multiplication, starting the most common addition formation, non-saturating where a MUX gate is used as shown in the figure below.

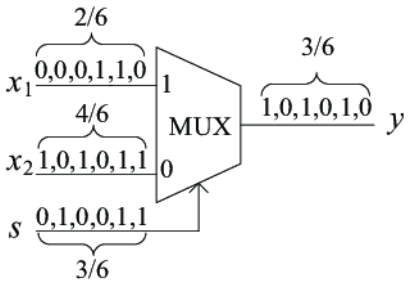

Figure 3.3: Stochastic unipolar non-saturating addition [25].

The MUX selects between the two incoming bitstreams with a probability of 50% which translates to a result where the number is added together but scaled to always stay within the range of 0 to 1, ie $\frac{a+b}{2}$.

However, because of this division by 2, consecutive additions result in the number getting smaller and smaller with no way of recovering the original value. If a,b,c and d are stochastic numbers then the non-saturating addition gets represented as,

$$\frac{\frac{\frac{a+b}{2}+c}{2}+d}{2} \neq a+b+c+d$$

An idea to get around this would be to premultiply each number beforehand such that the equation becomes

$$\frac{\frac{\frac{2a+2b}{2}+4c}{2}+8d}{2} = a+b+c+d$$

But the numbers get progressively larger by 2^x which would easily balloon out of hand for example if you have a 512 size array.

There is also another way of adding stochastic bitstreams and that is with the saturating addition which acts very much like a normal non-saturating addition. The saturating addition is simply an OR gate between the bitstreams however the problem with this to do with the scale of the numbers used within a neural network, the floating-point numbers coming in from the previous layer in the neural network are normalized between 0 and 1 but these numbers can easily reach values that are smaller than 0.01 and representing really small numbers stochastic is not really doable without a high number of bits, even then the amount of bit required would easily exceed any systems memory and computation power. The other problem is the weights, inside of a neural network are not bound, they have infinite range and can be positive or negative. This makes things especially difficult when working with the limited range of stochastic numbers, one could try to restrict the range of the weights but this would require intimate knowledge of how the weights are calculated, which is not the goal of this thesis.

3.7 Experiments

In order to judge the viability of using stochasitc computing operations to implement matrix multiplication, I've setup 3 experiments that measure the accuracy of stochastic computing using different formats and bitlength.

3.7.1 Accuracy of stochastic multiplication as a function of bitstream length

The first experiment is regarding the accuracy of the multiplication of stochastic numbers, this is the bread and butter of stochastic numbers and the most commonly flouted operation in terms of speed and accuracy. The experiment measures the accuracy of computing $0.5 \cdot 0.25$ with different bitlengths, these numbers were picked purely as a middle of the road estimate and are meant to give a rough estimate of the accuracy. The second experiment expands upon this and using a sampling of 20 different random numbers, averaging the difference between the intended result and the received result.

3.7.2 Accuracy of stochastic addition as a function of bitstream length

This set of experiments is very similar to the first, with example graphs generated with $0.5 \cdot 0.25$ and the absolute difference graph being generated in the same way. However to highlight the oddities of saturating addition at higher values, a number of graphs highlighting low values (< 0.5), high values (> 0.5) and a step graph were generated.

3.7.3 Division of bitstream to improve stochastic addition

The last experiment consisted of a new idea proposed to try and get around issues regarding stochastic numbers and their limited range. The idea consisted of spliting the bitstream into 2 bitstreams, one containing the integer component and the other containing the floating-point component. The split stochastic number still has a range which has to be predefined beforehand, it is essentially a multiplier for the integer component.

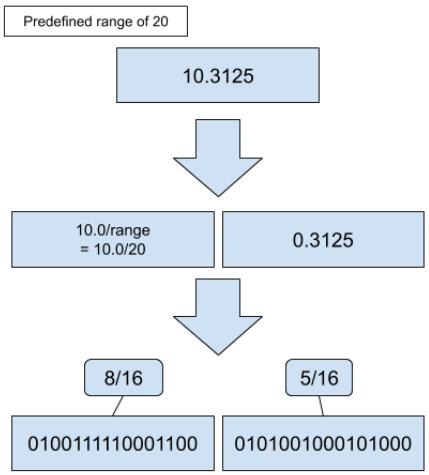

Figure 3.4: Diagram of split stochastic number.

The graphs generated compare both a normal unipolar stochastic multiplication with the custom 2 bitstream idea and the same things but with multiplication. Both tests used a length of 1000 for the bitlength and 10000 iterations of random values between 0 and 10 for the range.

Chapter 4

Results

4.1 Introduction

In this section, I share some of the results and graphs I've made and talk about what this means in the context of my thesis.

4.2 Multiplication results

4.2.1 Unipolar multiplication accuracy

This graph shows the multiplication of 2 stochastic bitstreams using a unipolar format with a varying bit-length (up to 10000 bits), the result of the multiplication should be $0.5 * 0.25 = 0.125$ as represented by the orange line.

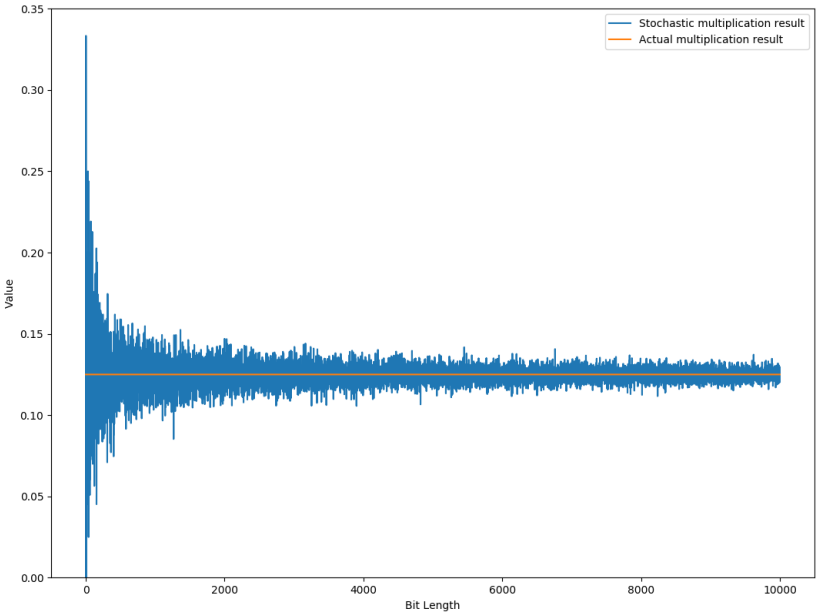

Figure 4.1: Stochastic multiplication. $(0.5 \cdot 0.25)$

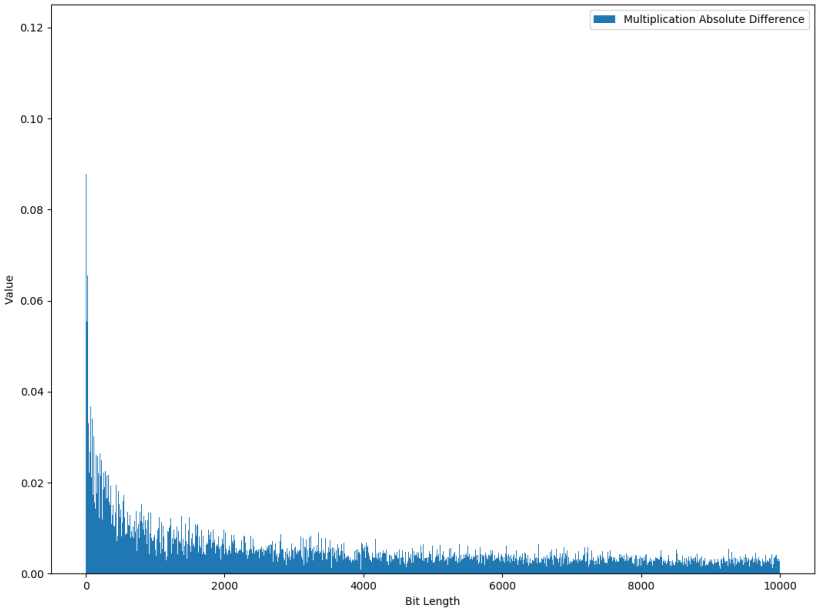

Figure 4.2: Stochastic multiplication absolute difference.

In the graphs above, we can see that as the length of the bitstream is increased the variance with the accuracy goes down but the gains are not linear and dropoff over time. The amount of accuracy needed to represent the ranges that a neural network uses would exceed 10000 bits.

4.2.2 Bipolar multiplication accuracy

This graph shows the multiplication of 2 stochastic bitstreams using a bipolar format with a varying bit-length (up to 10000 bits), the result of the multiplication should be $0.5 * 0.25 = 0.125$ as represented by the orange line.

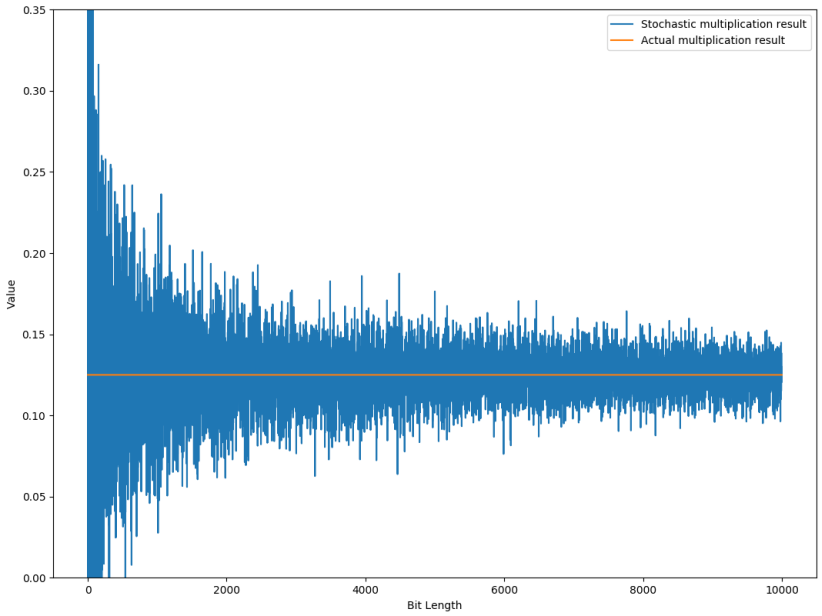

Figure 4.3: Stochastic bipolar multiplication. $(0.5 \cdot 0.25)$

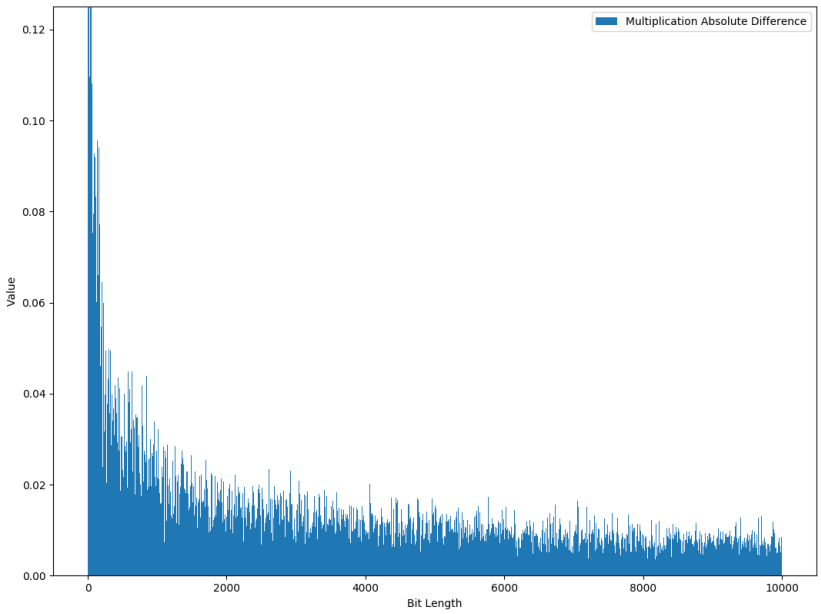

Figure 4.4: Stochastic bipolar multiplication absolute difference.

When using a bipolar format the range is increased by a factor of 2 since instead of the range (0 to 1) we are now covering (-1 to 1) and when matched up at the same scale as the unipolar graph, we can see that the accuracy range has also increased, it becomes even more obvious when we look at the absolute values which have much larger spikes than the unipolar format.

4.3 Addition results

4.3.1 Non saturating addition accuracy

This graph shows the addition of 2 stochastic bitstreams using the non-saturating addition with a varying bit-length (up to 10000 bits), the result of the multiplication should be $0.5 * (0.5 + 0.25) = 0.375$ as represented by the orange line.

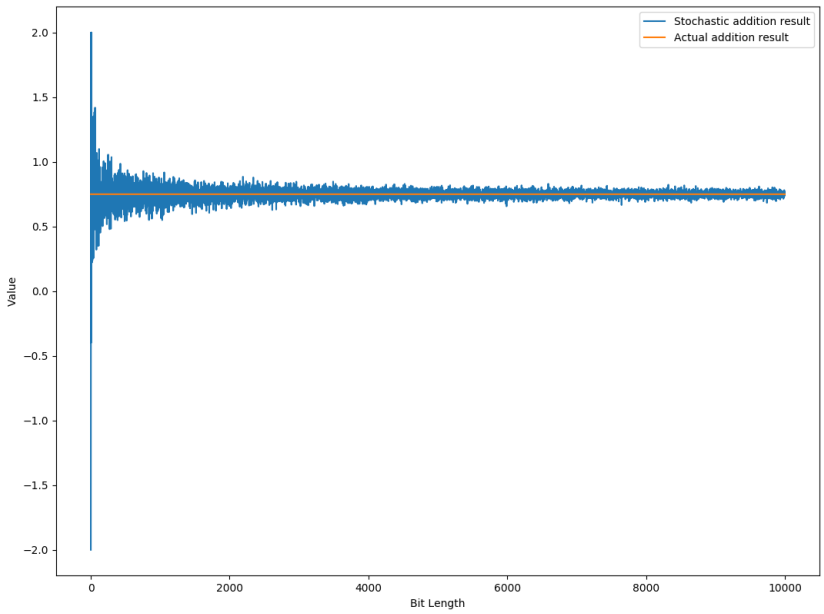

Figure 4.5: Non saturating stochastic addition.

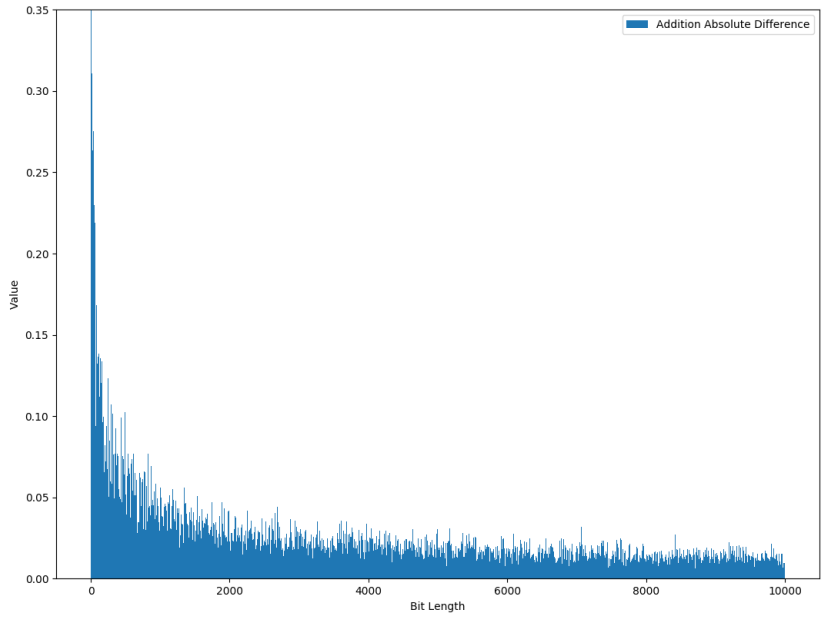

Figure 4.6: Non saturating stochastic addition absolute difference.

Similar to multiplication, non-saturating additions accuracy variance goes down with the number of bits but it also has the same issue with the gains in accuracy dropping off over time. Fortunately, the addition seems to have slightly better accuracy than multiplication but not by much.

4.3.2 Saturating addition accuracy

This graph shows the addition of 2 stochastic bitstreams using saturating addition with a varying bit-length (up to 10000 bits), the first graph should show the addition of $0.5 + 0.25 = 0.75$

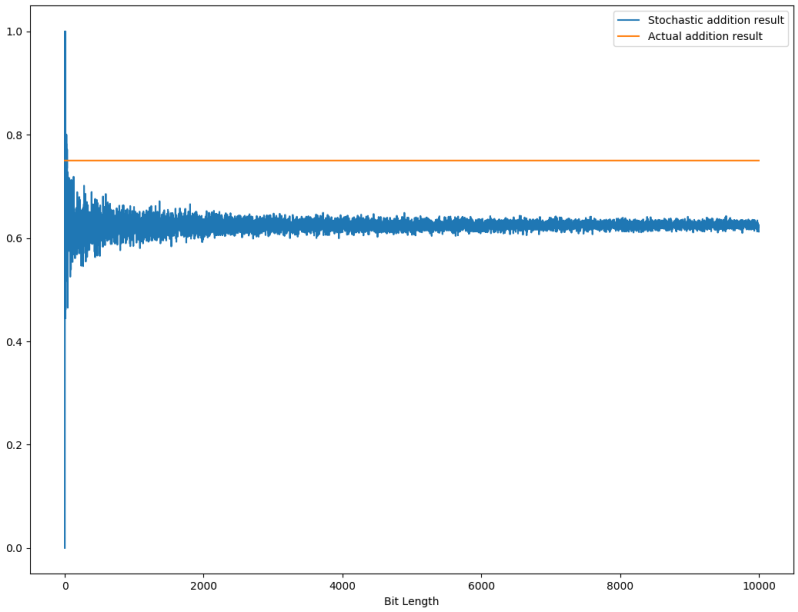

Figure 4.7: Saturating stochastic addition.

Saturating addition has the advantage of not trying to restrict the range of the addition which is useful in repeated additions for matrix multiplications however what was found was that saturating additions seems to have an offset from the intended target which increases as the result is closer to 1. The graphs below should demonstrate my point,

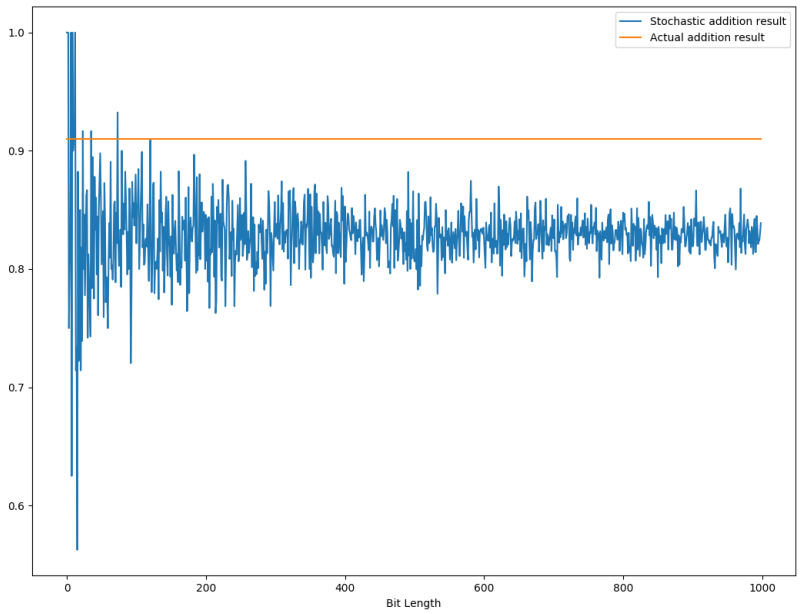

Figure 4.8: Saturating stochastic addition high values.

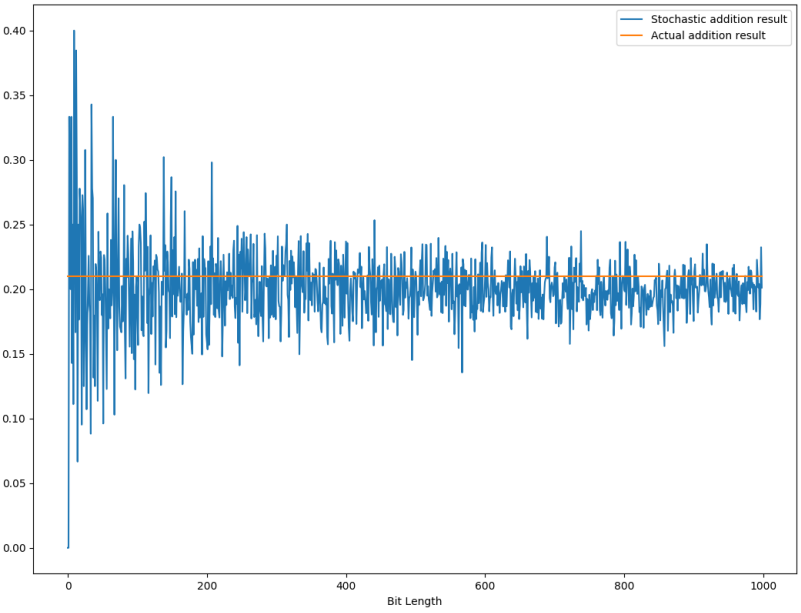

Figure 4.9: Saturating stochastic addition low values.

In the first graph, the resulting addition should be $0.81 + 0.1 = 0.91$ but the stochastic addition is offset by around 0.8 but in the second graph there is a much lower offset of around 0.02. To demonstrate my point further, I've fixed the bitlength to a 1000 bits, changing only the result of the addition.

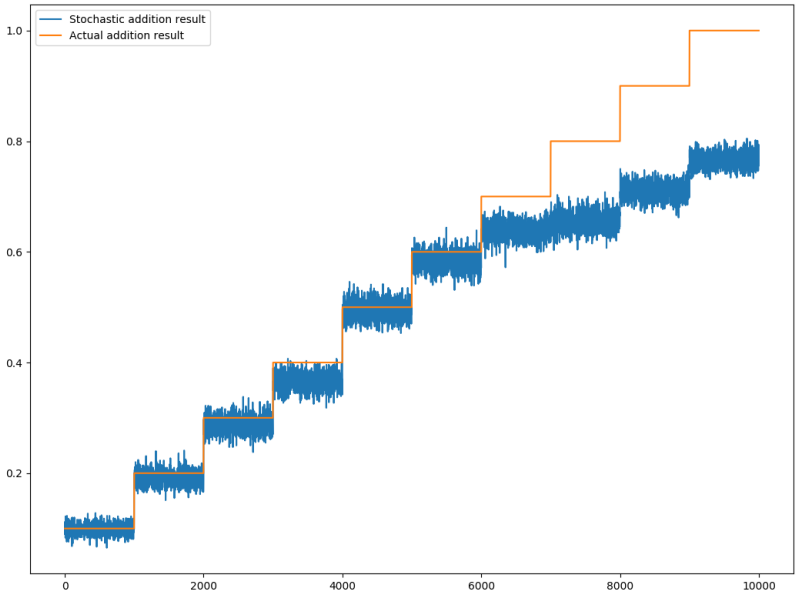

Figure 4.10: Stochastic stochastic addition multiple values.

As a result of the addition approaches 1, the addition becomes offset from the target by an ever-increasing amount. This seems to match up closely with an algorithmic anaylsis of stochastic computing which states that a saturating addition computes to $X = a+b-a \cdot b$ [23], such that when a and b becomes larger so does $a \cdot b$.

4.4 Spliting the bitstream

One of my proposals to get around the limitations the accuracy of stochastic bitstreams was to split the bitstream into two parts, one containing the decimal part and the other containing the integer component. This is a new and novel idea with the hope that splitting the bitstream into two parts would keep the high accuracy for the decimal part and because the integer component is only composed of whole numbers we could use a smaller bitstream length to represent more integers and gain more accuracy, this is especially relevant when considering the diminishing returns on increasing the bitstream for more accuracy.

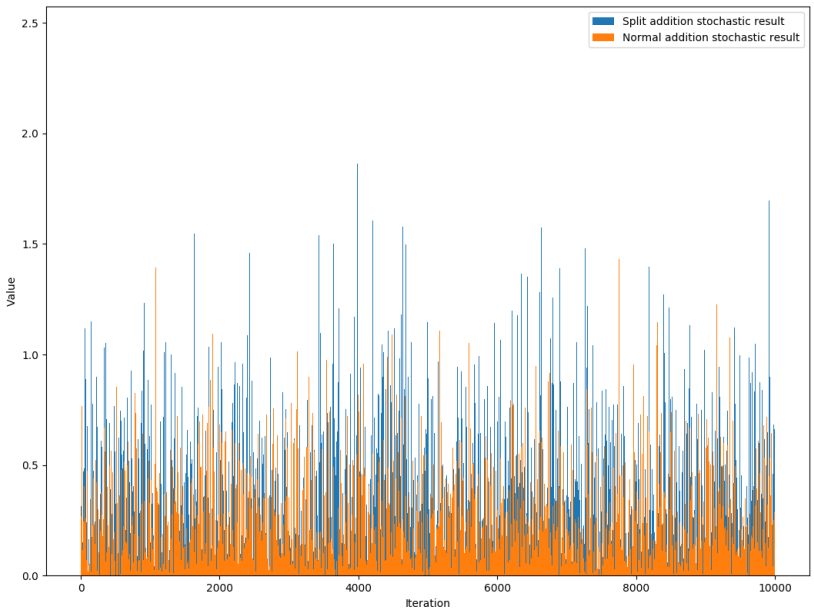

Figure 4.11: Spliting the bitstream (addition).

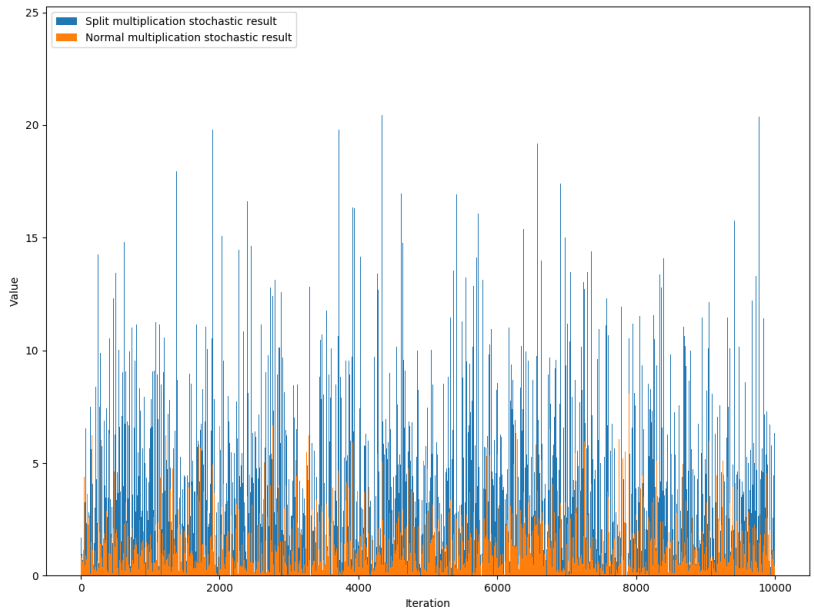

Figure 4.12: Spliting the bitstream (multiplication).

The first figure above shows a stochastic addition with the absolute difference between the result and the intended result, the method where the bitstream is split into 2 parts shows a higher absolute difference and thus is less accurate than the normal non-splitting method. This is the same with multiplication as shown in the second figure above, both addition and multiplication are less accurate and don't seem to offer any other tangible benefits.

Chapter 5

Conclusions

5.1 Conclusions

In conclusion, trying to apply stochastic computing to the field of machine learning through matrix operations alone is unlikely to succeed. Stochastic computing needs a lot more groundwork, regarding the inaccuracy and range limitation to be used outside its field and for stochastic computing to be able to be fully integrated with machine learning there will need to be a large paradigm shift in how they are used and applied.

Chapter 6

Future Work

The current design I have proposed has certain limitations and bounds that make it suitable for application regarding machine learning, as such this section will describe, future work to try and accomplish this goal as well as describing a theoretical device that would be able to work with neural networks.

6.0.1 Limitation 1

The first limitation of my design is the memory limits, as I'm emulating a stochastic bit-stream, each number in the matrix gets expanded out and this can easily exceed memory limits. The solution to this a way of a restructuring of the machine learning framework to accept a stream of data, this way only a few bits are being processed at a time. The difficulty here is that machine learning frameworks are not based around the idea of streaming data, batches of data are loaded onto the GPU and processed but this is at a very low level and it is very likely you would need to heavily restructure the machine learning framework, which is where a secondary problem comes in which is by modifying the machine learning framework to work with streams it becomes very hard to compare 2 results (other than raw accuracy) with each other as the calculations have changed and are no longer 1 to 1 comparable.

6.0.2 Limitation 2

The second limitation of my design is which has made it impossible to get results, is the acceptable range of the stochastic matrix. Due to the nature of stochastic numbers as to how they are simply a probability, each stochastic number has a range from 0 to 1, this can be mapped to other ranges after converting it back to a floating-point but when it is within a stochastic number the range is limited. Due to this limitation, my implementation of matrix multiplication can only do matrix multiplication where the final range is between 0 and 1 or where the final range of the matrix multiplication is known and can be mapped afterwards.

6.0.3 Theoretical device

Regarding the future work of this thesis, the neural network has to be built from the ground up using a stream-based idea. The multiplication and addition will need to be separate units which are then chained together to create the final solution. In addition to this, the addition problem regarding the range of the stochastic numbers will need an alternative solution with fewer drawbacks. In reality the stochastic numbers cannot really be thought of as just floating-point numbers of integer binary point numbers, in reality, they are more arbitrary point numbers and the larger amount of bits processed the surer you are of the answer, maybe a concept to look into is quantum computers as they also work on the idea of probabilities.

Chapter 7

Abbreviations

AWGN	Additive White Gaussian Noise
SGD	Stochastic Gradient Descent
SC	Stochastic numbers
NN	Neural network
MLP	Multi Layer Perceptron
CNN	Convolutional neural network
RNN	Recurrent neural network
RELU	Rectified linear unit
MNIST	Modified National Institute of Standards and Technology
CIFAR	Canadian Institute For Advanced Research

.1 Codebase and results

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

Bibliography

[1] A. Alaghi and J. P. Hayes, "Fast and accurate computation using stochastic circuits," in *2014 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2014, pp. 1–4.

[2] A. Alaghi, W. Qian, and J. P. Hayes, "The promise and challenge of stochastic computing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 8, pp. 1515–1531, Aug 2018.

[3] M. Bojarski, D. D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba, "End to end learning for self-driving cars," 2016.

[4] M. Buckland. Using neural nets to recognize handwritten digits. [Online]. Available: http://www.ai-junkie.com/ann/evolved/nnt1.html

[5] V. Canals, A. Morro, A. Oliver, M. L. Alomar, and J. L. Rosselló, "A new stochastic computing methodology for efficient neural network implementation," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 27, no. 3, pp. 551–564, March 2016.

[6] N. Donges. (2018) Gradient descent in a nutshell. [Online]. Available: https://towardsdatascience.com/gradient-descent-in-a-nutshell-eaf8c18212f0

[7] K. Feng, X. Pi, H. Liu, and K. Sun, "Myocardial infarction classification based on convolutional neural network and recurrent neural network," *Applied Sciences*, vol. 9, no. 9, 2019. [Online]. Available: https://www.mdpi.com/2076-3417/9/9/1879

[8] R. Ferduła, T. Walczak, and S. Cofta, "The application of artificial neural network in diagnosis of sleep apnea syndrome," in *Advances in Manufacturing II*, J. Trojanowska, O. Ciszak, J. M. Machado, and I. Pavlenko, Eds. Cham: Springer International Publishing, 2019, pp. 432–443.

[9] K. Fredenslund. Computational complexity of neural networks. [Online]. Available: https://kasperfred.com/series/introduction-to-neural-networks/computational-complexity-of-neural-networks

[10] K. Gurney, *An Introduction to Neural Networks*. Bristol, PA, USA: Taylor & Francis, Inc., 1997.

[11] J. Heaton. The number of hidden layers. [Online]. Available: https://www.heatonresearch.com/2017/06/01/hidden-layers.html

[12] T. Hirtzlin, B. Penkovsky, M. Bocquet, J.-O. Klein, J.-M. Portal, and D. Querlioz, "Stochastic computing for hardware implementation of binarized neural networks," 2019.

[13] C. W. Kim. Overfitting (what they are & train, validation, test & regularization). [Online]. Available: https://medium.com/machine-learning-intuition/overfitting-what-they-are-regularization-e950c2d66d50

[14] V. T. Lee, A. Alaghi, and L. Ceze, "Correlation manipulating circuits for stochastic computing," in *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2018, pp. 1417–1422.

[15] V. T. Lee, S. A. Elliot, A. Alaghi, and L. Ceze, "Synthesizing number generators for stochastic computing using mixed integer programming," 2019.

[16] B. Li, Y. Qin, B. Yuan, and D. J. Lilja, "Neural network classifiers using stochastic computing with a hardware-oriented approximate activation function," in *2017 IEEE International Conference on Computer Design (ICCD)*, Nov 2017, pp. 97–104.

[17] P. Molchanov, A. Mallya, S. Tyree, I. Frosio, and J. Kautz, "Importance estimation for neural network pruning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019.

[18] M. A. Nielsen. (2018) Neural networks and deep learning. [Online]. Available: http://neuralnetworksanddeeplearning.com/chap4.html

[19] R. M. Sadek, S. A. Mohammed, A. R. K. Abunbehan, A. K. H. A. Ghattas, M. R. Badawi, M. N. Mortaja, B. S. Abu-Nasser, and S. S. Abu-Naser, "Parkinson disease prediction using artificial neural network," *International Journal of Academic Health and Medical Research (IJAHMR)*, vol. 3, no. 1, pp. 1–8, 2019.

[20] Neuron model figure. [Online]. Available: https://github.com/trekhleb/machine-learning-octave/blob/master/neural-network/README.md

[21] Sigmoid figure. [Online]. Available: https://upload.wikimedia.org/wikipedia/commons/thumb/8/88/Logistic-curve.svg/320px-Logistic-curve.svg.png

[22] Neural network model figure. [Online]. Available: https://upload.wikimedia.org/wikipedia/commons/thumb/4/46/Colored_neural_network.svg/1200px-Colored_neural_network.svg.png

[23] Stochastic logic gate figure. [Online]. Available: https://player.slideplayer.com/84/13697725/slides/slide_6.jpg

[24] (2014) Neural network history. [Online]. Available: https://cs.stanford.edu/people/eroberts/courses/soco/projects/neural-networks/History/history1.html

[25] Stochastic multiplication and addition figure. [Online]. Available: https://www.researchgate.net/publication/334459474_Performing_Stochastic_Computation_Deterministically/figures?lo=1

[26] Stochastic logic gate figure. [Online]. Available: https://medium.com/bitgrit-data-science-publication/spiking-neural-networks-a-more-brain-like-ai-6f7ad86b7e7e

[27] Broadcasting in pytorch. [Online]. Available: https://towardsdatascience.com/speed-up-your-python-code-with-broadcasting-and-pytorch-64fbd31b359

[28] Overfitting in machine learning: What it is and how to prevent it. [Online]. Available: https://elitedatascience.com/overfitting-in-machine-learning#overfitting-vs-underfitting

[29] Matrix multiplication figure. [Online]. Available: https://hadrienj.github.io/assets/images/2.2/dot-product.png

[30] (2015) Tensorflow. [Online]. Available: https://www.tensorflow.org/

[31] (2016) Pytorch. [Online]. Available: https://pytorch.org/