## 1. INTRODUCTION

In the modern world, machine learning has created a plethora of tasks simpler and far less time-consuming. In recent years, drawing inspiration from the human nervous system, a Perceptron can be thought of as the machine learning equivalent of a neuron. Neural networks are one type of deep learning process that have become useful for operations such as classification, pattern recognition and prediction in a variety of fields [1]. Neural networks are of great importance and continue to be prove as a proficient and simplifying way of processing data [2]. The application of neural networks can be evaluated by factors such as, accuracy, processing speed, latency, performance, fault tolerance, volume, scalability, and convergence [3]. Deep learning methods have been ascertained for studies with large vats of data to yield phenomenal success in applications of speech recognition, pattern assessment and subsequent recognition, processing of language etc [4]. The aim of this study is using comparison techniques to depict the effectivity of various optimization methods to a multilevel neural network to one without them. This neural network will be used to categorise datapoints classified across different categories. Further information of the optimization methods is given in the methodology section.

## 2. METHODOLOGY

### 2.1. RECTIFIED LINEAR UNITS ACTIVATION

Rectified linear unit (ReLU) is an activation function with significant biological and mathematical underpinning [5]. Introduced by Hahnloser, R. H. R., et al. [5], this activation functions segregates values at the output thresholded at 0. It will output 0 if the input is less than 0 and a will express a linear function when the input is greater than 0. Figure 2.1 depicts a graphical representation of how this function outputs a value based on the corresponding input.

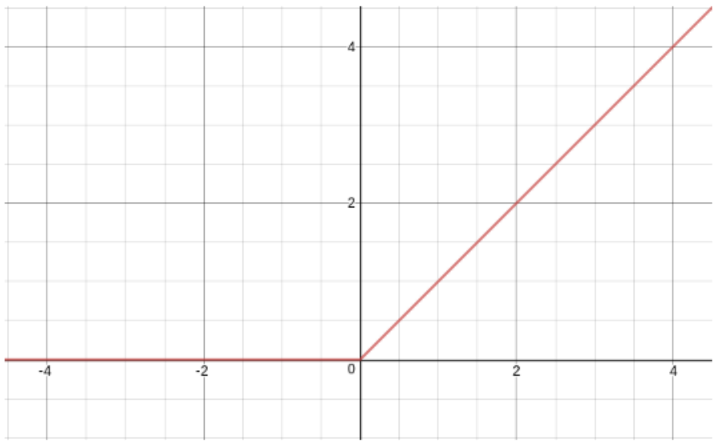


Figure 2.1 – The above graph shows the production of 0 as an output when the input is less than 0 and the expression of a linear gradient when the input is more than 0 [6].

## 2.2. SOFTMAX

This is a deep learning solution which utilizes the SoftMax function as a means of classification in the last layer. This function denotes the probability distribution for a number of specified classes, K and is represented as $\sum_{k=1}^{K} \mathrm{Pk}$ [6].

## 2.3. CROSS ENTROPY

Entropy is defined as the measure of the disorder of a system, in the case of a neural network it relates to the difference between two probability distributions. The cross entropy between two probabilities can be expressed as $H(P, Q)$, the probabilities of the two events can then be depicted as $H(P, Q) = -\sum_{x \epsilon X} P(x) * \log\big(Q(x)\big)$, where the sum runs across all events in X, P(x) and Q(x) are the true probabilities of those events between two probability distributions.

## 2.4. WEIGHT DECAY

Weight decay regularises to prevent overfitting by adding a sort of penalty to the activation/loss function. This penalty is limiting factor which essentially penalises large weight by decaying the value of the weight to zero. The formulaic representation of this is as follows: $L' = L + \frac{\lambda}{2}\left||W|\right|^2$, where L is the original loss, W are the weights, $\lambda$ is the weight decay coefficient and $\left||W|\right|^2$ is the squared L2 norm of the weights [7].

## 2.5. MOMENTUM IN STOCHASTIC GRADIENT DESCENT

The contribution of momentum in the stochastic gradient descent helps to fasten the SGD in the correct direction and decreases oscillations allowing for quicker convergence. Formulaically, it can be denoted as $v_{t+1} = \beta * v_t + (1 - \beta) * g_t$, where $v_t$ is the velocity (momentum term) at a time t and $g_t$ is the gradient. This method of gradient descent updates the weights dependent on the fractions of previous weight updates. Formulaically, the updating can be represented as follows: $W_{t+1} = W_t - \alpha * v_{t+1}$, where $\beta$ is the momentum coefficient and $\alpha$ is the learning rate [8].

## 2.6. BATCH NORMALISATION

The aim behind batch normalisation is to create neural networks that are faster and steadier through normalising the inputs into the layers. This is achieved by taking a batch of a dataset and then computing the batch's mean and variance. The formulation of this is as shown in figure 2.2. The batch is then normalised using the variance and mean and is subsequently scaled and shifted. This causes two new parameters to culminate symbolically denoted by $\gamma$ & $\beta$ [9].

$$\mu_B = \frac{1}{m}\sum_{i=1}^{m} X_i \ (Batch\ Mean)$$

$$\sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m} (X_i - u_B)^2 \ (Batch\ Variance)$$

$$X_{norm} = \frac{X_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \ (Normalisation\ of\ Batch)$$

$$Z_i = \gamma * X_{norm} + \beta \ (Scaling\ and\ Shifting$$

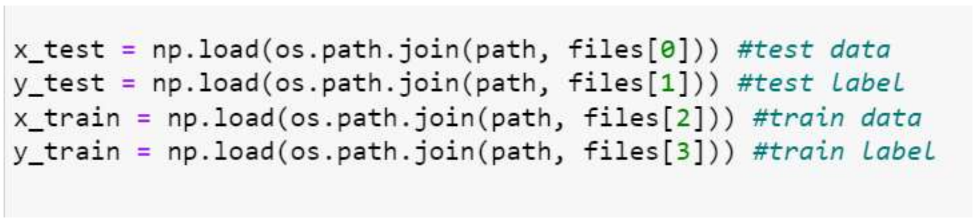
Figure 2.2 – the above formulae depict the mathematical process of batch normalisation. First the mean and variance of the batch is calculated and then used to normalise the data. The normalised data is then used to correct the shifting and scaling.

## 2.7. MINI-BATCH TRAINING

A slightly different method of normalisation to batch normalisation, where a fragment of the dataset is taken (mini-batch) to approximate the gradient. This method proves to be more memory-efficient than other gradient descent methods. The size of the mini-batch is a hyperparameter that can be manipulated and changed as the memory constraints of the system [10].

## 2.8. HOW THE CODE WORKS

*Refer to section 1 of code* – A path was initially created to the folder containing the numpy data arrays and then each file was saved into a variable (see figure 2.2). The training and testing sets were designated x and y to distinguish between their input and label components. The input sets for both the testing and training sets were normalised using the minimum and maximum in the following manner $\frac{data-\min}{max-min}$. The input data and labels were cropped to adapt to smaller memory of the system they were run on.



```
x_test = np.load(os.path.join(path, files[0])) #test data
y_test = np.load(os.path.join(path, files[1])) #test label
x_train = np.load(os.path.join(path, files[2])) #train data
y_train = np.load(os.path.join(path, files[3])) #train label
```

Figure 2.2 – The various data files are saved into their respective variables.

*Refer to section 2 of code* – the neural network scaffolding is split into two sections, namely “NeuralInput” and “NeuralLayers”. The class neural input contains the information to input for the number of neurons and number of inputs per neuron. The function in this class initialises the weights with random values which are subsequently updated with each iteration. The class entitled neural layers structures the whole neural network and contains the function for initialisation of the neural network layers. First the neural layers are defined in initialisation function. The ReLU function is then defined along with the associated gradient function. The ReLU activation function and gradient is applied to the first three layers of the neural network (input layer, hidden layer 1 and hidden layer 2).

The activation function applied to the output layer of the neural network is the softmax-cross-entropy function. This function processes the input data signified by “raw data” by clipping them within a certain range before calculating the exponential of the difference between the raw input and its max. The labels are then processed and converted to a numpy array and squeezed to remove any single dimensional entries. The probabilities are then calculated by exponential value taken before and dividing it by the sum of all the exponential values. The log of the probabilities and labels is calculated to compute the final loss. The SCE gradient function computes the gradient for loss correction in the last layer of the neural network.

The defined train function has the inputs, training_set (input data), training output (labels), iterations (number of times the train function will loop) a fixed learning rate and a gradient clip which also remains constant. The first step accounts for back propagation where each corresponding layers delta and error are calculated using the respective activation function and gradient. These error and delta computations are then used to calculate the weight update parameters used later to update the actual weight values by multiplying by the learning rate. The self-contained contemplate functions is what is used to prompt the neural network to “think” and spit out its predictions hence the values it returns. The accuracy is the metric used to measure the performance of the neural network.

*Refer to section 3 of code* – The neural network is introduced various optimization methods in an attempt to see it performs better. The first one included is batch normalisation, eponymously defined, it computes the mean and variance and then the normalisation using those two calculations. A weight decay constant of 0.01 and momentum constant of 0.9 are also incorporated into calculation using the gradient to update weights.

## 3. RESULTS/ANALYSIS

The structure of the neural network consisted of one input layer, two hidden layers and one final output layer. The ReLU function was used to activate inputs for every layer except the output layer, for which the softmax cross-entropy function was used. In this component of the study, a simple neural network for two hidden layers and one input and one output layer was curated. Initially, the learning rate was kept constant and $1 * 10^{-5}$ and the accuracy of the neural network was measured against the number of iterations, as observed in figure 3.2. The accuracy measurement was calculated by taking only taking the actual outputs that intersected with the predicted labels. The mean average was then computed for these values and denoted the accuracy. As the number of iterations decreased no stark effect to the accuracy was observed in fact it remained the same. The accuracy increased, however when higher number of iterations were tested. One notable observation was that at a higher number of iterations the output values became unreadable with the output indicating NaN values. Figure 3.2 shows that the accuracy did increase as the number of iterations increased, however this could not properly be confirmed as NaN values began to dominate the neural network. Through adjusting the learning rate, variations in the accuracy could be observed as seen in figure 3.1. As the learning rate decreased, the accuracy seemingly increased as observed in figure 3.1

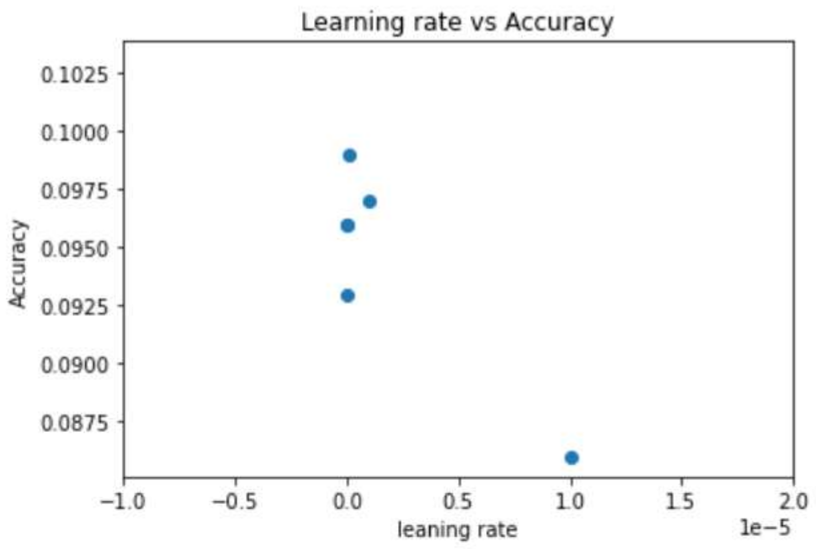
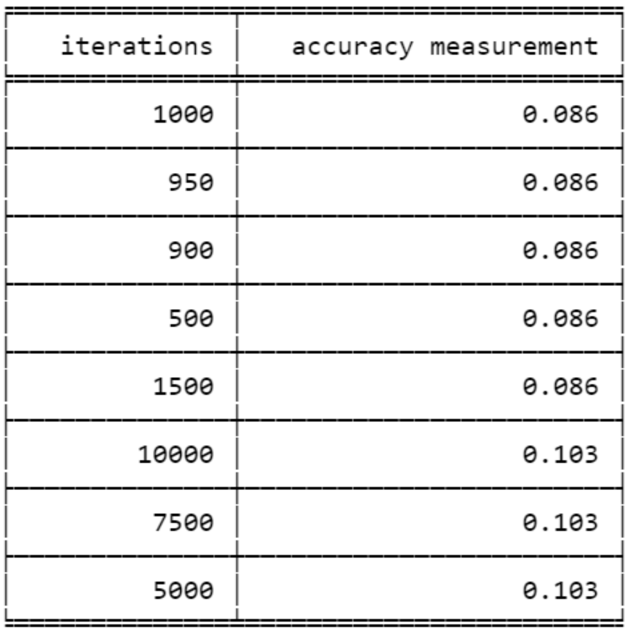


Figure 3.1 – The above graph shows the variation in the accuracy of the neural network with decreasing learning rate. The number of iterations was kept constant at 1000 and accuracy values were averaged over three epochs for each learning rate.



| iterations | accuracy measurement |
|---|---|
| 1000 | 0.086 |
| 950 | 0.086 |
| 900 | 0.086 |
| 500 | 0.086 |
| 1500 | 0.086 |
| 10000 | 0.103 |
| 7500 | 0.103 |
| 5000 | 0.103 |

Figure 3.2 – The above table shows the number of iterations are their corresponding accuracy measurements. The learning rate was kept constant at $1 * 10^{-5}$.

Through the addition of a weight decay constant and momentum value to the gradient descent and finally batch normalisation, it was observed that the general accuracy of the neural network increased. This represents a key indicator that through various optimization tools, a neural network can perform with greater proficiency. These advantageous methodologies allow for the use of neural networks. Figure 3.3 shows a similar relationship trend as shown without these additions in the previous neural network in section 2 of the code. Upon comparing the two tables from figure 3.2 and 3.4, it can also be concluded that additional optimization methods improves the general accuracy of the system. Despite showing a decreasing accuracy with decreasing number of iterations, the general accuracy system proved to be grater than the accuracy without the optimization methodologies.

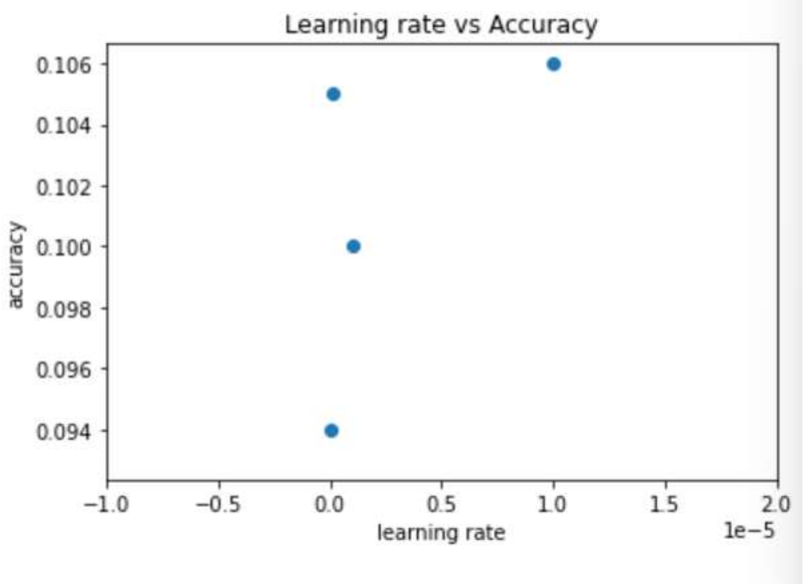
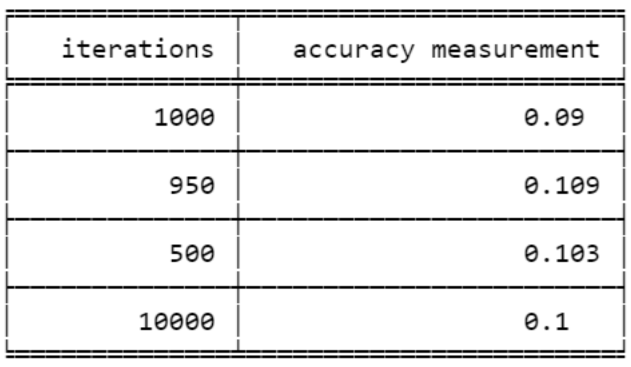


Figure 3.3 – The above graph shows the learning rate against the accuracy of the neural network of section 3 of the code with the additional optimization methodology.



| iterations | accuracy measurement |
|---|---|
| 1000 | 0.09 |
| 950 | 0.109 |
| 500 | 0.103 |
| 10000 | 0.1 |

Figure 3.4 – the above table shows the variation in number of iterations and the affects on the accuracy of the neural network in section 3 of the code.

## 4. CONCLUSION

Neural networks form a novel classification tool that have shaped the modern world leading to wonderous tools such as AI and deep learning. Through this simple study, in which a simple neural network was compared to one with various optimization methods applied to it. It was clear from a simplistic accuracy measurement that though neural networks are a useful tool in various regards, requires certain optimizations to be applicable proficiently. In essence, it was clear that the accuracy of a system was improved with additions such as weight decay, momentum to the gradient descent and batch normalisation.

## 5. REFERENCES

1) Abioduna, O. I., et al. (2018). "State-of-the-art in artificial neural network applications: A survey." Heliyon **4**.
2) Izeboudjen, N., et al. (2012). "A new classification approach for neural networks hardware: from standards chips to embedded systems on chip." Artificial Intelligence Review **41**.
3) Mozaffari, A., et al. (2019). "A comprehensive investigation into the performance, robustness, scalability and convergence of chaos-enhanced evolutionary algorithms with boundary constraints." Artificial Intelligence Review **52**.
4) a, W. L., et al. (2017). "A survey of deep neural network architectures and their applications." Neurocomputing **234**.
5) Hahnloser, R. H. R., et al. (2000). "Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit." Nature **405**.
6) Agarap, A. F. (2019). "Deep Learning using Rectified Linear Units (ReLU)."
7) Krogh, A. S. and J. A. Hertz (1991). "A simple weight decay can improve generalization." NIPS'91: Proceedings of the 4th International Conference on Neural Information Processing Systems.
8) Sutskever, I., et al. (2013). "On the importance of initialization and momentum in deep learning." Proceedings of the 30th International Conference on Machine Learning **28**.
9) Ioffe, S. and C. Szegedy (2015). "Batch Normalization: Accelerating Deep Network Training byReducing Internal Covariate Shift."
10) Goodfellow, I., et al. (2016). Deep Learning, MIT Press.