Determining the Classification Accuracy for Image Recognition of Feedforward Neural Networks with Different Activation Functions

Aaron Manning

Contents

Abstract

The design of multi-layer perceptron neural networks is primarily based on the optimisation of the network's cost function, a measure of performance for each input. With this goal in mind, a variety of different activation functions have been proposed. While the sigmoid function has historically been the most common, many alternatives have been proposed in recent years. The aim of this study was to determine which of six common activation functions provided the greatest proportion of correctly identified images, called classification accuracy, from the 10000 images in the MNIST testing set of handwritten digits. It was determined that the Swish function resulted in the greatest classification accuracy, followed by Softplus and Leaky ReLU, followed by Hyperbolic Tangent and Inverse Tangent, followed by Sigmoid.

Literature Review

Since the publication 'A Logical Calculus of the Ideas Immanent in Nervous Activity', the research into Multi-Layer Perceptrons (MLPs) and subsequently more complex forms of Artificial Neural Networks (ANNs) has expanded with the primary areas of focus being the effects of changing the structure of the networks, different activation functions, and alternative hyperparameters and their appropriate values. The goal of this is to minimise the cost function of a network on testing data to improve the classification accuracy and applications in solving more complex problems.

Much early research on MLPs suggest that the vast majority of local minima of the cost function have cost very similar to that of the global minimum (LeCun et al. 1998), and those that do not have a very high probability of being saddle points rather than local minima. Therefore small changes to the design of the network and the results of weight initialisation cause a lower cost value to be found (Dauphin et al. 2014). However, this provides a basis for how points can be found within the cost function that are not a local minima but have a tangent plane with a gradient of 0.

'The Loss Surfaces of Multilayer Networks' (Choromanska et al. 2015) provides an explanation for

these observations by showing that the number of local minima close to the global minima of the cost function increases as a function of the size of the network. Furthermore, attempting to find the global minima by going up to a saddle point and retraining the network on the training data often leads to over-fitting and worse results on unseen data than would be achieved with many local minima that lead to greater error on the training set. As part of the results of this paper, the correlation between training and testing cost is given for networks with different numbers of hidden layers, in which a clear trend is shown where the greater the number of hidden neurons, the lower this correlation. This provides an indication that the minima of the cost function lie within a region of a size that is negatively inversely proportional to the size of the network. While the size of the network is also inversely proportional to the variance of the cost statistic over a large number of networks.

The initialisation of weights within a neural network can be computed according to a variety of different methods. 'Understanding the difficulty of training deep feedforward neural networks' (Glorot & Bengio 2010) suggests initialisation according to a normal distribution with a variance of $\frac{2}{n_{\text{in}}+n_{\text{out}}}$, since otherwise weights are known to 'explode' (increase to infinity) or 'die' (approach 0). Just as certain weight initialisation methods can introduce problems, certain activation functions can be more susceptible to the features of these methods. While ReLU is known to provide a faster rate of learning than many previously used options it introduces the 'Dying ReLU' problem where because negative results have a rate of change of 0, certain data sets cause the network to lose information and halt learning. Leaky ReLU is designed to overcome this problem by using a minimal coefficient, often 0.1, to the first argument of the maximum function and therefore providing a non-zero derivative within that domain. It does however have the disadvantage of being unbounded below 0 (Ramachandran, Zoph & Le, 2017).

Activation functions can be bounded or unbounded, where bounded functions have a maximum and/or minimum value and 'squish' the inputs within some range (LeCun et al. 1998). Functions such as Sigmoid and Inverse Tangent are bounded by a maximum and minimum, whereas function such as ReLU and Swish are bounded only at a minimum. A function such as Swish is known to perform better than ReLU in almost all circumstances due to its unbounded maximum and non-zero gradients just below 0 (Ramachandran, Zoph & Le, 2017). Therefore, there is less of a need to do research into this specific comparison between Swish and ReLU. Additionally, since ReLU

must be initialised differently to the method outlined above, direct comparisons of the nature in this research, with the inclusion of ReLU, would be invalid.

In recent years Multilayer Perceptrons have mostly been neglected and no thorough study which clearly outlines the differences in performance of a wide variety of activation functions exists. Some studies which provide direct comparisons between functions ignore the Hyperbolic Tangent and Inverse Tangent functions, or focus on the benefits of a specific function. This is due to their limitations on complicated problems, with only localised random filters and local feature learning such as those found in convolutional neural networks, where classification correctness of over 98% on common sets such as MNIST and CIFAR10 were observed, have been biologically plausible (Illing, Gerstner & Brea 2019). However, in general, MLPs provide a testing ground for general features of machine learning algorithms and therefore can be used as a research tool for Convolutional and Long short-term memory (LSTM) ANNs, and thus are still crucial in many modern applications of machine learning.

Scientific Research Question

Which activation function provides the most consistently correct classification on the MNIST set of handwritten numbers when applied to a simple multi-layer perceptron neural network?

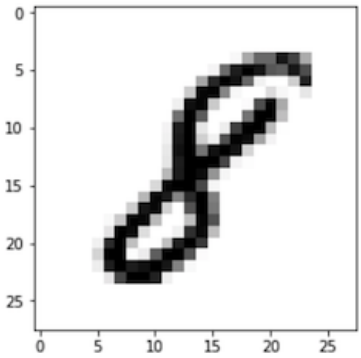
Scientific Hypothesis

When a Swish activation function is used within a multi-layer neural network, the classification accuracy will be higher than when using Sigmoid, Softplus, Leaky ReLU, Inverse Tangent and Hyperbolic Tangent.

Methodology

The neural network used in this experiment was designed for, and trained and tested on the MNIST database of handwritten numbers, there are 784 neurons in the first layer, each corresponding to a pixel of the original image, with their activation determined by the grey-scale colour value of each pixel on a graduated scale, normalised between 0 and 1. Figure 1 below shows an example image from this data set.

Figure 1: Sample Image



For an image such as this, the expected output in the case of the network is performing optimally, is an activation of 1 in the output neuron labelled 8 and an activation of 0 for all others.

For the purposes of this experiment, six different activation functions have been directly compared. Each of these functions is plotted below with their corresponding derivatives in blue. Notice the different properties of each activation function that are noteworthy as outlined in the literature review, including the existence or absence of upper and lower bounds, monotonicity and the presence or absence of possible negative outputs.

Figure 2: Sigmoid Function

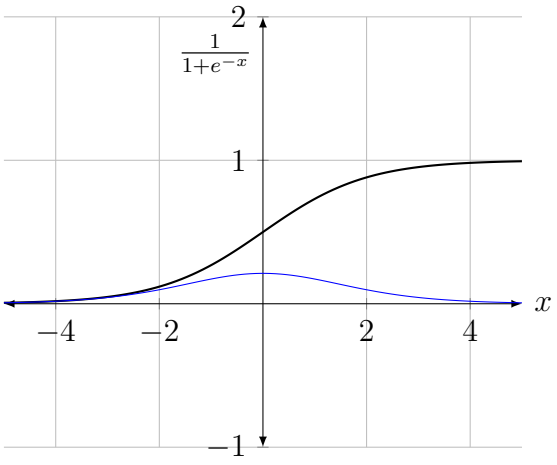
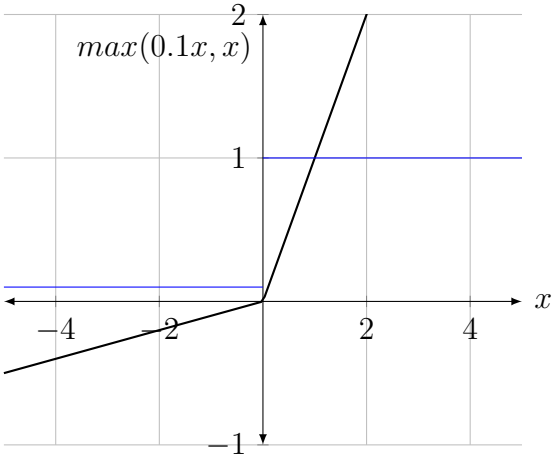


Figure 3: Leaky Rectified Linear Unit Function



Due to the lack of smoothness at $x = 0$, the Leaky ReLU function shown in Figure 5 can only be differentiated as a piecewise function with $f'(0)$ defined as 0.1.

Figure 4: Swish Function

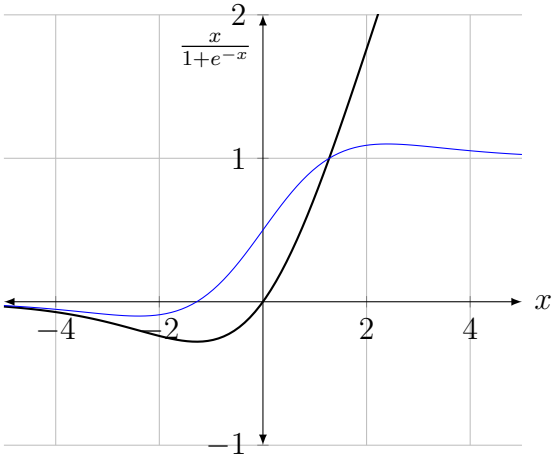


Figure 5: Softplus Function

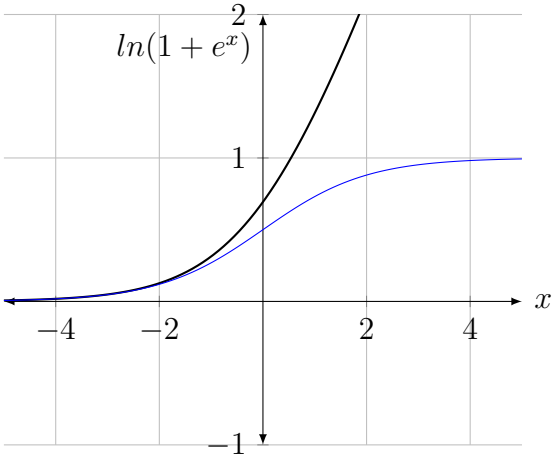


Figure 6: Hyperbolic Tangent Function

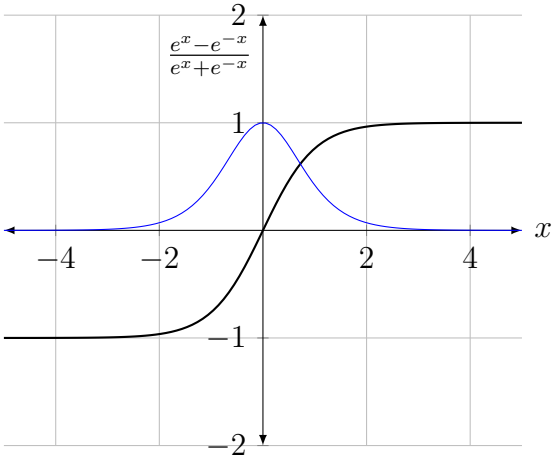
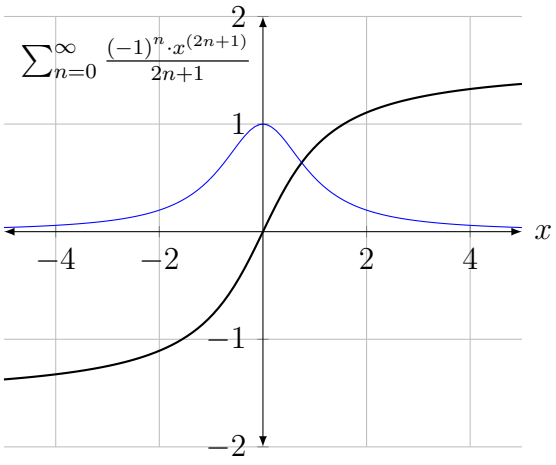


Figure 7: Inverse Tangent Function



See Appendix A for a full explanation of the mathematical systems involved in feeding forward a neural network and the properties of activation functions.

Within this experiment 6 different formats for neural networks were created, one for each activation function. These were identical to each other with the exception of the activation function, and the inbuilt randomness. All other hyperparameters were determined according to the values listed below.

- Weights Learning Rate: 0.01
- Biases Learning Rate: 0.1
- Network Structure (layer by layer): [784, 20, 20, 10] - one in input layer for each pixel and one in each output layer for each possible digit
- Weight Initialisation: Xavier Initialisation $N(0, \sqrt{\frac{2}{n_{\text{in}}+n_{\text{out}}}})$
- Biases Initial Value: 0.01
- Gradient Descent Method: Stochastic
- Epochs: 3
- Size of Training Set: 60,000

The complete computer code used to create, train and test each neural network can be found in Appendix C.

For each trial within each group, a new neural network with identical parameters was initialised with newly randomised weights. The network was then fully trained on a training set of 60,000 images and tested on 10,000 images distinct from the training set. Hence, the collected measurements were the number of correctly identified images for each network out of 10,000. 100 trials were completed for each group and therefore 600 total measurements were taken. See Appendix B for full details on the process of training the network.

Results

The complete set of results collected within the experiment can be found in Appendix D while density curves for each group can be found in Appendix E. Summary statistics for this data are tabulated below.

Statistic	Sigmoid	Leaky ReLU	Swish	Softplus	Hyperbolic Tangent	Inverse Tangent
Mean	9172	9433	9472	9446	9380	9358
Variance[1]	856.8	1397	705.8	561.9	1265	1005
Standard Deviation[2]	29.27	37.38	26.57	23.71	35.57	31.70
Skewness[3]	-0.39	-0.051	0.16	0.096	-0.20	-0.11
Excess Kurtosis[4]	-0.18	-0.27	-0.053	-0.56	-0.67	-0.54

[1] Sample variance with Bessel's correction applied was used. [2] Sample standard deviation with Bessel's correction applied was used. [3] Ranges of skewness between -1 and 1 generally allow the distribution to be considered symmetrical enough to be normal. [4] Ranges of Excess Kurtosis between -2 and 2 generally allow the distribution to be considered normal in terms of tailedness.

As can be seen from the above data, the Swish function had the highest average classification accuracy with the second smallest variance. Further tests for the significance of this result are detailed in the discussion.

Discussion

Statistical Analysis

To identify if the differences between each group's mean are statistically significant, the multiple comparisons technique was used.

In determining the appropriate test to use for these comparisons, some underlying assumptions were tested.

Testing the Assumption of Normality

To verify the normality of each group, the skewness and kurtosis were calculated, along with the results from the Shapiro-Wilk normality test.

With respect to the summary statistics, skewness closer to 0 denotes a more symmetrical distribution where a conservative range of -1 to 1 is often used to determine if a distributions symmetricality is close enough to be deemed having met that criteria of normality, of which, all the groups meet this criteria. In terms of kurtosis, a range of -2 to 2 of excess kurtosis (or 1 to 5 for normal kurtosis) is often accepted for a distribution to be considered normal, a criterion that all groups meet with clear room for error.

Additionally the Shapiro-Wilk test for normality was conducted. This test was used as it has the greatest statistical power in detecting a difference assuming one exists. Given that the null hypothesis for this test is that the distribution being tested is normal, rejecting the null hypothesis indicates that a distribution is not normal. While failure to reject the null hypothesis, does not provide verification that it is true, due to the inherently high statistical power of the test, the great number of samples and high statistical significance level of 0.15 used for this test, it does provide an indication of its normality.

The results for this test across each sample are shown below:

Group	Sigmoid	Leaky ReLU	Swish	Softplus	Hyperbolic Tangent	Inverse Tangent
Test Statistic W	0.9819	0.9933	0.9893	0.9879	0.9810	0.9885
p-Value	0.19	0.91	0.61	0.50	0.16	0.55

Each p-value in this table can be interpreted as the probability of seeing a distribution as far from a normal distribution as was examined assuming it was normal. Hence, the high values indicate that each distribution is roughly normal.

Testing the Assumption of Homoscedasticity

To test for homescedasticity (the variance of each group being the same), Bartlett's test was used. This test relies on the assumption that each group is normally distributed, a property verified above. For this test, the null hypothesis is that the variances of each group are equal. A standard alpha significance level of 0.05 was used for this test, as there is little harm in selecting a non-parametric test when a parametric test is available, only lowering overall statistical power.

Performing this test on the collected data, a p-value of $2.45 \cdot 10^{-5}$ was calculated, allowing a conclusion to be drawn that the variances of the different groups differ too significantly to use a parametric test that assumes equal variance.

Welch's t-test

Given the verification that each sample is normally distributed with differing variances, the test used for each pair of comparisons was Welch's t-test, as many alternative tests such as the students t-test require equal variance between groups. For these comparisons the following hypotheses were used:

- H_0: There is no difference between the means of the two groups being compared.
- H_1: There is a difference between the means of the two groups being compared.

The corresponding test statistic for Welch's t-test was compared against a t-distribution using a two-tailed test, due to the initially unknown ranks of the groups, and the desire to not bias the experiment by using a test statistic with a symmetrical sampling distribution. For these tests a significance level of 0.001 was used. This was in the interest of keeping the total probability of a type I error across all comparisons as low as possible. Given that a total of $\binom{6}{2} = 15$ comparisons will be made, the probability of a type I error will be:

$$1 - (1 - 0.001)^{15} \approx 0.149 = 1.49\%$$

This choice of this p value resulted in a low statistical power, counteracted only be the significant number of trials in each group. However, due to the importance of minimising the chance of error, this trade off was made.

Multiple Comparisons

Here is a table of the resulting p-values from the comparisons between groups.

Function	**Sigmoid**	**Leaky ReLU**	**Swish**	**Softplus**	**Hyperbolic Tangent**
Inverse Tangent	$1.64 \cdot 10^{-102}$	$6.28 \cdot 10^{-35}$	$2.15 \cdot 10^{-68}$	$2.61 \cdot 10^{-53}$	$1.12 \cdot 10^{-2}$
Hyperbolic Tangent	$1.74 \cdot 10^{-100}$	$1.34 \cdot 10^{-25}$	$1.92 \cdot 10^{-55}$	$4.12 \cdot 10^{-40}$	
Softplus	$3.33 \cdot 10^{-12}$	$5.72 \cdot 10^{-3}$	$3.33 \cdot 10^{-12}$		
Swish	$1.96 \cdot 10^{-147}$	$8.79 \cdot 10^{-15}$			
Leaky ReLU	$1.29 \cdot 10^{-117}$				

Given the α level of 0.001 a statistically significant difference was present in 13 of the 15 comparisons. Those that did not meet this threshold were the Leaky ReLU - Softplus comparison and the Inverse Tangent - Hyperbolic Tangent comparison, with p-values of $5.72 \cdot 10^{-3}$ and $1.12 \cdot 10^{-2}$ respectively.

Ranking the networks in performance according to these results:

1	Swish
2	Softplus Leaky ReLU
3	Hyperbolic Tangent Inverse Tangent
4	Sigmoid

Some of the functions have been grouped due to the lack of statistically significant difference between them. However, they have been ordered within their group according to the indication that the results gave towards the better performing network.

Analysis of Experimental Design

Due to the deterministic nature of the data and calculations, as well as the careful control of all parameters of each network, the claims made within this research are valid and sound. The only controlled variable that had an element of randomness was the initial value of the weights of the network. It was crucial that this be random due to the seemingly random nature of the cost function and thus the requirement to initialise the different weights differently to reach a different local minimum. To prevent the consequences of this required randomness from invalidating the results, the same randomisation function was used for all networks across all groups and the number of trials in each group minimised the probability of this randomness having a significant effect on the final results.

To improve the reliability 100 trials per group were performed for each activation function and hence these results should be easily reproducible. Additionally, with the final results of the multiple comparisons providing p-values less than 0.01 in all cases, any individual comparison should also be easily reproducible with a statistical significance level sufficient for that individual experiment.

As the data was collected as discrete counts, there is no inherent measurement error within these measurements. However, it is important to note the limitation of the programming language and computer used, with the smallest possible unit able to be stored being $2.47 \cdot 10^{-324}$. Additionally, in the calculations made, there is an inherent error as a consequence of the protocals the computer uses when calculating floating point arithmetic, requiring numbers be stored in binary and thus decimal numbers not being represented for their exact value. For example, the value calculated by the computer when asked to calculate $0.1 + 0.2$ is $3 + 4 \cdot 10^{-18}$. The consequences of these errors were minimal enough given the size of the numbers that were used within the calculations and as output data. However, the measurement accuracy of the measurements as proportions are limited by an upper boundary of the total number of images in the testing set, hence 4 significant figures are known for each measurement. If a greater training set were available, a greater number of significant figures could have been calculated.

Potential Future Research Opportunities

There are many other characteristics and aspects of MLPs, including the learning rates, initialisation algorithm, update frequency, number of epochs, normalisation method for input data and initial value for the biases, which, while being controlled in this experiment could also be changed independently to determine the best parameters for a neural network. This extends to more complicated hyperparameters not considered within the scope of this research including momentum and weight decay.

Additionally, due to the weight initialisation method chosen for this network, the ReLU function was omitted due to how it would be unfairly disadvantaged. As such, further research opportunities exist for comparing ReLU directly to other networks where a more appropriate initialisation method is used.

Conclusion

The purpose of this experiment was to identify the activation function that provided the greatest classification accuracy on the MNIST database of handwritten numbers when used within an MLP ANN. Testing for differences was done by comparing groups of 100 measurements where each data point represents a count of the number of correct images identified out of 10,000. Concluding each individual hypothesis: given the required alpha level of 0.001, the null hypothesis that there is no difference between the two groups being compared was rejected in 13 cases. Given the above criteria for assessing differences between groups, statistically significant differences were not found in the comparisons between Softplus and Leaky ReLU, and Hyperbolic Tangent and Inverse Tangent. Hence, in answer to the scientific research question, the Swish function had the highest performance of the six activation functions being examined when tested on classification accuracy with the MNIST data base of handwritten numbers and hence that hypothesis has been accepted. Swish was then followed by Softplus and Leaky ReLU in second place, Hyperbolic Tangent and Inverse Tangent in third place and Sigmoid in fourth place.

References

Choromanska, A, Henaff, M, Mathieu, M, Arous, G, LeCun, Y 2015, "The Loss Surfaces of Multilayer Networks": JMLR W&CP, Vol. 38, accessed February 4, 2019, <https://arxiv.org/abs/1412.0233>.

Dauphin, Y, Pascanu, R, Gulcehre, C, Cho, K, Ganguli, S, Bengio, Y 2014, "Identifying and attacking the saddle point problem in high-dimensional non-convex optimization": Preceedings of the 27th International Conference on Neural Network Information Processing Systems, Vol. 2, pp. 2933-2941, accessed February 9, 2019, <https://arxiv.org/abs/1406.2572>.

Glorot, X, Bengio, Y 2010, "Understanding the difficulty of training deep feedforward neural networks": Journal of Machine Learning Research, Vol. 9, pp. 249-256, accessed February 3, 2019, <http://proceedings.mlr.press/v9/glorot10a.html>.

Illing, B, Gerstner, W, Brea, J 2019, "Biologically plausible deep learning - But how far can we go with shallow networks?": Neural Networks, Vol. 118, pp. 90-101, accessed February 3, 2019, <https://arxiv.org/abs/1905.04101>.

LeCun, Y, Botton, L, Bengio, Y, Patrick, H 1998, "Gradient-Based Learning Applied to Document Recognition": Proceedings of the IEEE, Vol. 86, No. 11, pp. 2278-2324, accessed February 10, 2019, <https://ieeexplore.ieee.org/document/726791>.

McCulloch, W, Pitts, W 1990, "A Logical Calculus of the Ideas Immanent in Nervous Activity": Bulletin of Mathematical Biology, Vol. 52, No. 1/2, pp. 99-115, accessed January 20, 2019, <https://link.springer.com/article/10.1007/BF02478259>.

Ramachandran, P, Zoph, B, Le, Q 2017, "Searching for Activation Functions": ICLR Conference Blind Submission, 2018, accessed February 5, 2019, <https://arxiv.org/abs/1710.05941>.

Acknowledgements

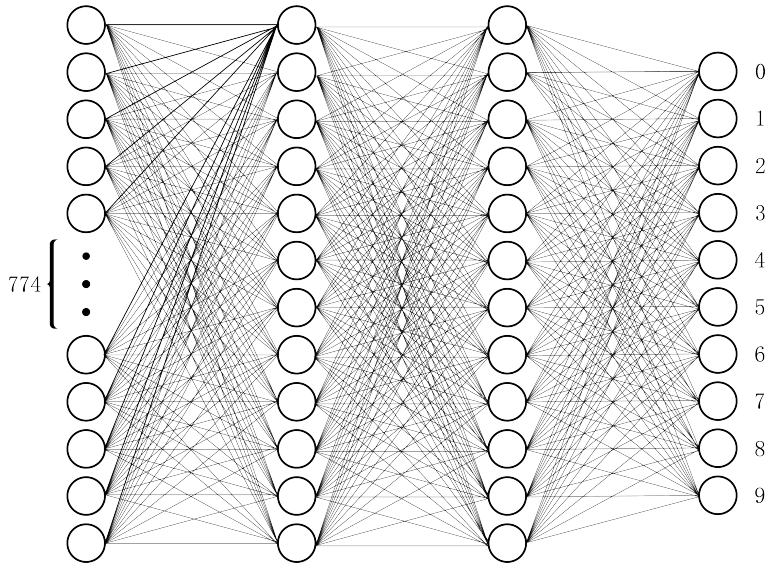
As the author of this paper, I wish to thank the following people for their contributions to its completion:

- Carolyn Imre, for assistance throughout the proposal, data collection and analysis as my teacher and feedback on the report nearing completion.
- Matthew Hill, for feedback on data analysis and report write-up.
- Floris Van Ogtrop, for assistance in choosing and conducting appropriate analysis of data.
- Rocco Ianni, for feedback on the computer code used in the data collection.

Appendix

Appendix A: Network Structure and Feeding Forward

A Multi-Layer Perceptron (MLP), is a Neural Network type categorised by its structure. They are comprised of layers of neurons each with a numerical activation associated with it. Any given neuron in the network is fully connected to all neurons in both the previous and following layer with weights, but is not connected to any other neurons within its own layer. The first layer is called the input layer, since the activations of each neuron in this layer are given as inputs, the internal layers are called hidden layers and the final layer is called the output layer. With the exception of the input layer, each layer's neuron activations are calculated using the previous layer, the weights connecting it and a set of biases.



The process of calculating the activations of each neuron in the network is called feeding forward. The activation of any given neuron is calculated as the weighted sum of the neurons in the previous layer plus a bias, with an activation function then applied to this result.

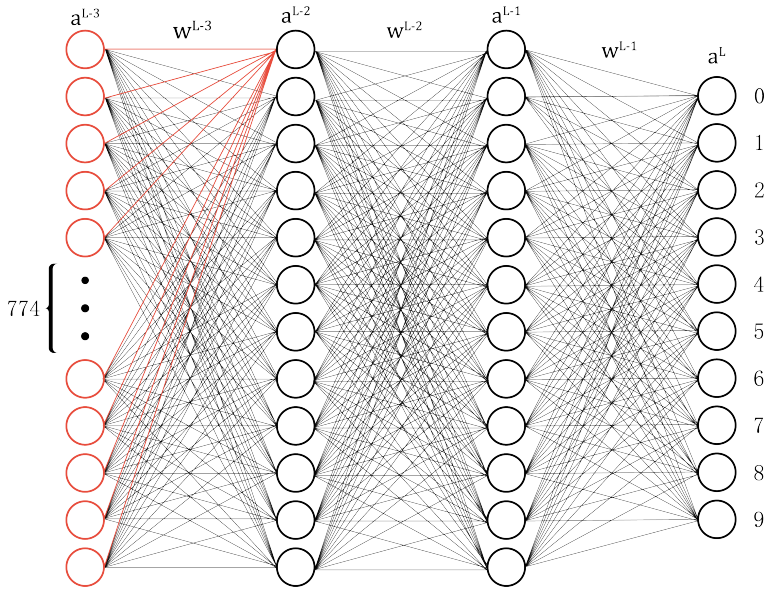
When referencing and indexing each element of the neural network, the output layer is referenced as layer L while others are given in terms of it.

Consider the calculation for the activation of neuron 0 in layer $L-2$.

$$a_0^{L-2} = \sigma(z^{L-3}) = \sigma(b_0^{L-3} + \sum_{j=0}^{783} a_j^{L-3} w_{0,j}^{L-3})$$

In this calculation take additional note that the term z, equal to the argument of the sigmoid function, is indexed to match the parameters that calculated it. Therefore, the last set of weights is indexed as $L-1$ and the biases to calculate layer L are indexed as $L-1$. The activation function used in this case is called the sigmoid function.

The subscript of a (the activation) denotes that neuron's index within its layer (starting at 0) and the superscript denotes the layer. In the case of the weights, the subscript assumes the form: *neuron connection in second layer, neuron connection in first layer.* This diagram shows the network with the elements used in this weighted sum coloured in red.



The relevant calculations for neuron activations are done in matrix vector form as it is less computationally expensive and easier to organise the data. Here is the calculation for the activations of neurons in layer $L-2$ in this form.

$$\sigma\left(\begin{bmatrix} w_{0,0}^{L-3} & w_{0,1}^{L-3} & w_{0,2}^{L-3} & ... & w_{0,n}^{L-3} \\ w_{1,0}^{L-3} & w_{1,1}^{L-3} & w_{1,2}^{L-3} & ... & w_{1,n}^{L-3} \\ w_{2,0}^{L-3} & w_{2,1}^{L-3} & w_{2,2}^{L-3} & ... & w_{2,n}^{L-3} \\ ... & ... & ... & ... & ... \\ w_{k,0}^{L-3} & w_{k,1}^{L-3} & w_{k,2}^{L-3} & ... & w_{k,n}^{L-3} \end{bmatrix} \begin{bmatrix} a_0^{L-3} \\ a_1^{L-3} \\ a_2^{L-3} \\ ... \\ a_n^{L-3} \end{bmatrix} + \begin{bmatrix} b_0^{L-3} \\ b_1^{L-3} \\ b_2^{L-3} \\ ... \\ b_k^{L-3} \end{bmatrix}\right) = \begin{bmatrix} a_0^{L-2} \\ a_1^{L-2} \\ a_2^{L-2} \\ ... \\ a_k^{L-2} \end{bmatrix}$$

Where n is the number of neurons in the known layer and k is the number of neurons in the layer being calculated.

Note that the sigmoid function is applied individually to each element of the vector that it takes as an input once it has been computed.

Using this calculation method for the second layer of the network from the inputs, and then each subsequent layer from the previous, the output layer can be calculated.

The purpose of the activation function is to provide the non-linearity necessary in allowing the neural network to approximate complex functions. Any network with a linear activation function is reducible to a simple mathematical function. Therefore, to satisfy the requirements of a valid activation function $f(x)$, a given function must be able to be differentiated and $f'(x)$ must not be defined as a constant, meaning $f(x)$ is non-linear. This derivative criterion is for the purposes of training.

Appendix B: Backpropagation and Training Methods

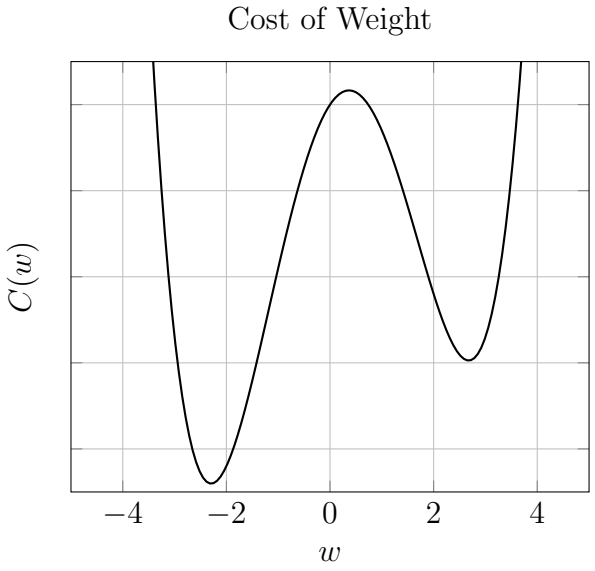
When attempting to train the network, a calculation called cost is used to determine how effective it is on any given training example. The cost is a calculation of the sum of squared difference between the expected output of a network and the actual output across each neuron in the output layer.

$$C(\mathcal{N}) = \sum_{j=0}^{n-1} (a_j^L - E_j)^2$$

Where $\mathcal{N}$ is the network being examined, n is the number of neurons in the final layer and E is the expected output for the neuron specified in the subscript.

When attempting to train a neural network, a technique called backpropagation is used which attempts to find the minimum value of the cost function by calculating the rate of change in the value of the cost with respect to each specific connection's associated weight $\frac{dC}{dw}$ or bias $\frac{dC}{db}$.

Given the number of factors that influence the cost, the following graphical representation of this function is simplified by showing it only with respect to one of it's parameters, a single generic weight.



Even with respect to one variable, the mathematical definition of this function is unknown. However, the function's value and gradient at any specific point can be calculated, assuming a version of the network with the weight at the desired value exists. As such the process of minimising this function is iterative. Each specific weight is updated by subtracting a value proportional to the rate of change of the function at the existing value of the weight.

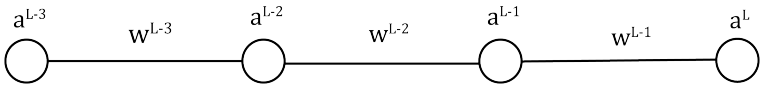
$$w_{\text{new}} = w_{\text{original}} - \eta \cdot \frac{dC}{dw}$$

This multiplicative constant η is called the learning rate and assumes a value between 0 and 1.

By repeatedly taking steps in the positive direction when the gradient is negative, and in the negative direction when the gradient is positive, the cost approaches smaller values. The use of this learning rate is to allow the function to approach a minimum turning point without making steps too significant that it will not converge. As a result of this method of minimising the cost, only a local minimum is reached and possible lower values of the cost may exist. As an example, for an initial weight of 1 in the above example, initial changes will be made to increase the weight's value despite a lower value of cost existing for a less positive weight.

The methods involved in applying the modifications to the biases of the network are similar, with their own corresponding learning rate.

To simplify the calculus involved within the backpropagation algorithm, consider the following basic neural network with only one neuron per layer.



Calculating the relevant rates of change, once again assuming an activation function of sigmoid:

$$\text{If } C(\mathcal{N}) = (a_0^L - E_0)^2, \text{ then } \frac{dC}{da_0^L} = 2(a_0^L - E_0)$$

$$\text{If } a_0^L = \sigma(z_0^{L-1}) = \frac{1}{1+e^{-z_0^{L-1}}}, \text{ then } \frac{da_0^L}{dz_0^{L-1}} = \sigma'(z_0^{L-1}) = \frac{e^{-z_0^{L-1}}}{(1+e^{-z_0^{L-1}})^2}$$

$$\text{If } z_0^{L-1} = a_0^{L-1}w_0^{L-1} + b_0^{L-1}, \text{ then } \frac{dz_0^{L-1}}{dw_0^{L-1}} = a_0^{L-1}$$

$$\text{By relating the rates of change, } \frac{dC}{dw_0^{L-1}} = \frac{dz_0^{L-1}}{dw_0^{L-1}} \times \frac{da_0^L}{dz_0^{L-1}} \times \frac{dC}{da_0^L}$$

In order to find the derivative with respect to the weights in earlier layers, differentiating with respect to a_0^{L-1} rather than w_0^{L-1} in the third set of equations above will give a result of w_0^{L-1}, which can then be used to calculate $\frac{da_0^{L-1}}{dz_0^{L-2}}$ and $\frac{dz_0^{L-2}}{dw_0^{L-2}}$. Relating the corresponding rates of change:

$$\frac{dC}{dw_0^{L-2}} = \frac{dz_0^{L-2}}{dw_0^{L-2}} \times \frac{da_0^{L-1}}{dz_0^{L-2}} \times \frac{dz_0^{L-1}}{da_0^{L-1}} \times \frac{da_0^{L}}{dz_0^{L-1}} \times \frac{dC}{da_0^{L}}$$

For the change in cost in terms of a given bias, the derivative of z_0^{L-1} with respect to b_0^{L-1} is 1. Hence the rate of change with respect to a set of weights is equal to the rate of change with respect to the input of the activation function z with the same layer superscript.

Computational Implementation of Backpropagation

When implementing the backpropagation algorithm over a full sized network using vectorised operations, a recursive function is often used to compute the rate of change between the cost and the input to each layer.

This rate of change to the input of the activation function for the last layer can be calculated as follows:

$$\frac{dC}{dz^{L-1}} = 2(a^L - E) \circ f'(w_{L-1}a_{L-1} + b_{L-1})$$

While the other rates of change can be calculated in terms of the next as follows.

$$\frac{dC}{dz^{L-2}} = (w_{L-1})^T \frac{dC}{dz_{L-1}} \circ f'(w_{L-2}a_{L-2} + b_{L-2})$$

Where $\circ$ is the element wise product and A^T represents the transpose of a generic matrix A.

The rate of change to any set of biases is equal to this term calculated since the next relevant derivative is 1. The rate of change to any set of weights can be calculated as:

$$\frac{dC}{dw^{L-1}} = \frac{dC}{dz^{L-1}}(a^{L-1})^T$$

Batch, Mini-Batch and Stochastic Gradient Descent

When updating the weights and biases of a neural network, changes are calculated according to each training example. Depending on the design of the network these changes may be applied at different frequencies. These frequencies of changes are categorised as follows:

- Batch: Where the changes are accumulated and updated at the end of all training examples.
- Mini-Batch: Where the changes are accumulated according to a smaller batch size and updated at the end of each mini-batch.
- Stochastic: Where the changes are not accumulated, but rather applied after every training example.

These different methods have benefits and limitations in terms of rate of training and computational time as well as implications in terms of the optimal learning rate to use.

When training the network, the entire data set is iterated over a number of times, called the number of epochs.

Weight and Bias Initialisation

When setting up a neural network before training it, the weights and biases of the network need to be assigned initial values.

Any given neural network is extremely sensitive to small changes in the initialisation function. Weights within any given network are randomised according to some function as the randomness creates some low-value cost cases. For the biases of the network, it is common to set their initial values to a small positive number to guarantee that all second layer neurons will fire, regardless of the input.

Appendix C: Code of Neural Network

The following is code designed specifically for this experiment, written in R, to create, train and test each neural network. The below example is for the sigmoid function however a minor modification are made in the 'NetworkGenerator' function call to change this.

```
Normalise <- function(x)
{
        return((x - min(x)) / (max(x) - min(x)))
}

DataSetReader <- function(first.input.column = 1, last.input.column,
   file.path, normalise = FALSE)
{
        data.set <- unname(as.matrix(read.csv(file.path, header = FALSE)
           ))[,first.input.column:last.input.column]

        if (NCOL(data.set) == 1)
        {
                data.set <- t(data.set)
        }

        if (normalise)
        {
                data.set <- Normalise(data.set)
        }

        return(data.set)
}

sigmoid <- function(x) { return(1 / (1 + exp(-x))) }

sigmoid.derivative <- function(x) { return(exp(-x)/(1 + exp(-2 *x) + 2 *
    exp(-x))) }
```

```
leakyrelu <- function(x) { return(ifelse(x > 0, x, 0.1 * x)) }

leakyrelu.derivative <- function(x) { return(ifelse(x > 0, 1, 0.1)) }

tanh <- function(x) { return((2 / (1 + exp(-2 * x))) - 1) }

tanh.derivative <- function(x) { return((4 * exp(-2 * x)) / ((exp(-2 * x
   ) + 1)^2)) }

arctan <- function(x) { return(atan(x)) }

arctan.derivative <- function(x) { return(1 / (x^2 + 1)) }

swish <- function(x) { return(x / (1 + exp(-x))) }

swish.derivative <- function(x) { return((1 + exp(-x) + x * exp(-x)) /
   (1 + 2 * exp(-x) + exp(-2 * x))) }

softplus <- function(x) { return(log(1 + exp(x))) }

softplus.derivative <- function(x) { return(1 / (1 + exp(-x))) }

BiasesGenerator <- function(structure, initial.value = 0.01)
{
        biases <- as.list(rep(NA, times = length(structure)))

        for (i in 2:length(structure))
        {
                biases[[i]] <- rep(initial.value, structure[i])
        }

        return(biases)
```

```
}

RandomWeights <- function(left.layer.length, right.layer.length)
{
        return(matrix(
                rnorm(left.layer.length * right.layer.length, mean = 0,
                   sd = sqrt(2/(left.layer.length + right.layer.length))
                   ),
                nrow = right.layer.length,
                ncol = left.layer.length
        ))
}

WeightsGenerator <- function(structure)
{
        weights <- as.list(rep(NA, times = (length(structure) - 1)))

        for (i in 1:(length(structure) - 1))
        {
                weights[[i]] <- RandomWeights(structure[i], structure[i
                   + 1])
        }

        return(weights)
}

FeedForward <- function(weights, biases, activation.function, input,
   structure)
{
        activations <- as.list(rep(NA, times = length(structure)))

        activations[[1]] <- input
```

```
        for (i in 2:length(structure))
        {
                activations[[i]] <- activation.function(weights[[i - 1]]
                    %*% activations[[i - 1]] + c(biases[[i]]))
        }

        return(activations)
}

DeltaBackpropagation <- function(n, activations, expected.output,
   weights, biases, derivative.function, structure)
{
        Delta <- function(n)
        {
                if (n == length(structure) - 1)
                {
                        return((2 * (activations[[n + 1]] - expected.
                           output)) * derivative.function(weights[[n]]
                           %*% activations[[n]] + biases[[n + 1]]))
                }
                else
                {
                        return((t(weights[[n + 1]]) %*% Delta(n + 1) *
                           derivative.function(weights[[n]] %*%
                           activations[[n]] + biases[[n + 1]])))
                }
        }

        delta <- as.list(rep(NA, times = (length(structure) - 1)))

        for (i in 1:(length(structure) - 1))
        {
                delta[[i]] <- Delta(i)
```

```
        }

        return(delta)
}

WeightsBackpropagate <- function(delta, structure, activations)
{
        delta.weights <- as.list(rep(NA, times = (length(structure) - 1)
           ))

        for (i in 1:(length(structure) - 1))
        {
                delta.weights[[i]] <- delta[[i]] %*% t(activations[[i]])
        }

        return(delta.weights)
}

BiasesBackpropagate <- function(delta, structure)
{
        delta.biases <- as.list(rep(NA, times = length(structure)))

        for (i in 2:(length(structure)))
        {
                delta.biases[[i]] <- delta[[i - 1]]
        }

        return(delta.biases)
}

Train <- function(network, input, output, epochs = 1, batch.size = 1,
   weights.lr = 0.01, biases.lr = 0.1)
{
```

```
activation.function <- network[["activation.function"]]
derivative.function <- network[["derivative.function"]]
weights <- network[["weights"]]
biases <- network[["biases"]]
structure <- network[["structure"]]

reset.cummulative.changes <- TRUE

for (epoch.iterator in 1:epochs)
{
        for (data.set.iterator in 1:ncol(input))
        {
                activations <- FeedForward(weights, biases,
                   activation.function, input[,data.set.iterator
                   ], structure)

                delta <- DeltaBackpropagation(n, activations,
                   output[,data.set.iterator], weights, biases,
                   derivative.function, structure)
                weights.changes <- WeightsBackpropagate(delta,
                   structure, activations)
                biases.changes <- BiasesBackpropagate(delta,
                   structure)

                if (reset.cummulative.changes)
                {
                        weights.changes.cummulative <- weights.
                           changes
                        biases.changes.cummulative <- biases.
                           changes
                        reset.cummulative.changes = FALSE
                }
                else
```

```
                {
                        for (element in 1:(length(structure) -
                           1))
                        {
                                weights.changes.cummulative[[
                                   element]] <- weights.changes.
                                   cummulative[[element]] +
                                   weights.changes[[element]]
                                biases.changes.cummulative[[
                                   element + 1]] <- biases.
                                   changes.cummulative[[element
                                   + 1]] + biases.changes[[
                                   element + 1]]
                        }
                }

                if (data.set.iterator %% batch.size == 0)
                {
                        for (i in 1:(length(structure) - 1))
                        {
                                weights[[i]] <- weights[[i]] -
                                   weights.lr * weights.changes.
                                   cummulative[[i]]
                                biases[[i + 1]] <- biases[[i +
                                   1]] - biases.lr * biases.
                                   changes.cummulative[[i + 1]]
                        }

                        reset.cummulative.changes <- TRUE
                }
        }
}
```

```
        return(list(weights = weights, biases = biases, activation.
           function = activation.function, structure = structure))
}

Test <- function(network, input)
{
        activation.function <- network[["activation.function"]]
        weights <- network[["weights"]]
        biases <- network[["biases"]]
        structure <- network[["structure"]]

        activations <- as.list(rep(NA, times = length(structure)))

        activations[[1]] <- input

        for (i in 2:length(structure))
        {
                activations[[i]] <- activation.function(weights[[i - 1]]
                    %*% activations[[i - 1]] + c(biases[[i]]))
        }

        return(activations[[length(structure)]])
}

ReformatByMax <- function(results)
{
        results.reformatted <- matrix(0, ncol = ncol(results), nrow =
           nrow(results))
        for (i in 1:ncol(results))
        {
                results.reformatted[which(results[,i] == max(results[,i
                   ])),i] <- 1
        }
```

```
        return(results.reformatted)
}

IdenticalColumnCheck <- function(output, results.reformatted)
{
        correct.counter <- c()

        for (i in 1:ncol(results.reformatted))
        {
                correct.counter <- c(correct.counter, all(results.
                   reformatted[,i] == output[,i]))
        }

        return(correct.counter)
}

NetworkGenerator <- function(structure, activation.function, derivative.
   function, biases.initial.value = 0.1)
{
        return(list(
                weights = WeightsGenerator(structure),
                biases = BiasesGenerator(structure, biases.initial.value
                   ),
                activation.function = activation.function,
                derivative.function = derivative.function,
                structure = structure
        ))
}

input.data.set <- Normalise(DataSetReader(1, 60000, "numberstraininput.
   csv"))
```

```
expected.output.data.set <- DataSetReader(1, 60000, "numberstrainoutput.
   csv")
input.testing <- Normalise(DataSetReader(1, 10000, "numberstestinput.csv
   "))
expected.output.testing <- DataSetReader(1, 10000, "numberstestoutput.
   csv")

for (i in 1:100)
{
        network <- NetworkGenerator(c(784,20,20,10), activation.function
            = leakyrelu, derivative.function = leakyrelu.derivative)

        network <- Train(network, input.data.set, expected.output.data.
           set, epochs = 3, batch.size = 1)

        results <- Test(network, input.testing)
        results.reformatted <- ReformatByMax(results)
        correct.counter <- IdenticalColumnCheck(expected.output.testing,
            results.reformatted)

        print(as.data.frame(table(correct.counter)))

        write.table(as.numeric(table(correct.counter)[2]), file = "
           leakyrelu.csv", append = TRUE, col.names = FALSE, row.names =
            FALSE)
}
```

Appendix D: Raw Measurements

The data listed in the table below is organised per trial (or measurement) and per group. Each number represents the number of images correctly identified out of the testing set of 10,000 images. Therefore each data point is the results of a fully trained network with the activation function of

the corresponding column label.

Trial	Sigmoid	Leaky ReLU	Swish	Softplus	Hyperbolic Tangent	Inverse Tangent
1	9176	9473	9471	9463	9343	9400
2	9117	9407	9465	9444	9389	9416
3	9193	9421	9441	9448	9401	9370
4	9179	9459	9468	9487	9401	9365
5	9178	9401	9448	9436	9312	9351
6	9162	9513	9466	9491	9342	9393
7	9152	9431	9499	9461	9373	9367
8	9142	9408	9530	9431	9394	9393
9	9151	9511	9434	9437	9411	9355
10	9176	9407	9485	9422	9391	9351
11	9167	9441	9516	9411	9405	9334
12	9158	9512	9514	9446	9374	9321
13	9179	9443	9466	9409	9388	9393
14	9187	9429	9455	9401	9397	9299
15	9200	9492	9453	9484	9349	9353
16	9143	9426	9481	9447	9409	9376
17	9165	9422	9457	9408	9347	9371
18	9175	9428	9527	9453	9350	9367
19	9201	9398	9459	9443	9382	9383
20	9146	9461	9457	9465	9362	9402
21	9209	9436	9497	9463	9346	9408
22	9156	9422	9494	9470	9421	9342
23	9216	9385	9507	9466	9410	9360
24	9175	9429	9496	9464	9347	9350
25	9220	9458	9541	9418	9408	9384
26	9179	9424	9462	9428	9381	9277
27	9192	9393	9408	9462	9330	9321
28	9199	9395	9445	9411	9369	9379

29	9120	9420	9464	9457	9376	9407
30	9168	9381	9457	9425	9361	9375
31	9166	9421	9428	9463	9356	9328
32	9186	9402	9478	9463	9380	9349
33	9131	9487	9479	9458	9322	9325
34	9182	9488	9487	9490	9357	9386
35	9235	9449	9423	9452	9425	9399
36	9216	9471	9486	9472	9407	9339
37	9148	9455	9481	9421	9381	9375
38	9152	9494	9439	9445	9317	9355
39	9191	9501	9477	9428	9362	9343
40	9205	9468	9464	9458	9407	9354
41	9190	9420	9461	9441	9390	9333
42	9192	9468	9481	9449	9413	9353
43	9195	9437	9411	9483	9412	9340
44	9156	9450	9487	9395	9348	9410
45	9178	9443	9463	9413	9416	9380
46	9226	9415	9460	9455	9318	9394
47	9113	9415	9490	9437	9333	9333
48	9175	9467	9518	9456	9395	9342
49	9198	9448	9475	9449	9395	9412
50	9132	9450	9485	9470	9378	9344
51	9143	9447	9496	9417	9388	9379
52	9126	9476	9483	9432	9314	9419
53	9174	9476	9462	9463	9396	9385
54	9182	9398	9450	9422	9328	9375
55	9178	9466	9441	9453	9338	9381
56	9177	9359	9429	9449	9359	9342
57	9168	9394	9474	9459	9383	9402
58	9098	9431	9468	9495	9390	9317
59	9155	9495	9470	9465	9316	9418

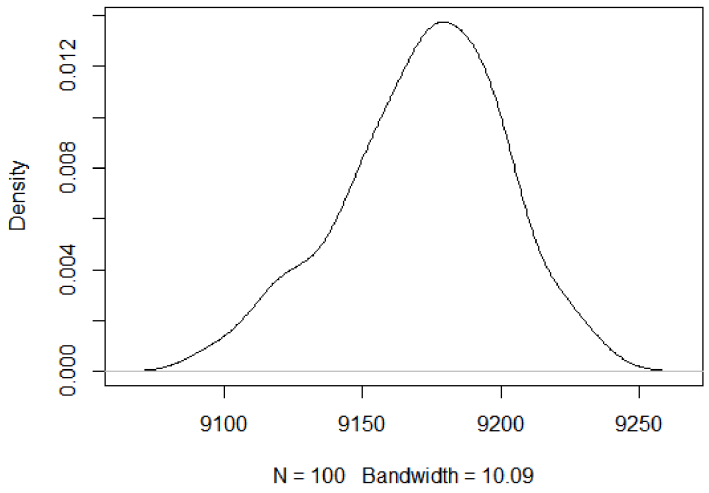
60	9190	9379	9478	9412	9432	9419
61	9203	9409	9439	9455	9393	9321
62	9120	9412	9478	9474	9366	9330
63	9174	9424	9445	9414	9378	9311
64	9113	9376	9490	9433	9331	9344
65	9157	9447	9460	9433	9315	9320
66	9203	9431	9475	9492	9336	9372
67	9187	9440	9477	9445	9394	9396
68	9168	9448	9494	9435	9321	9379
69	9222	9465	9448	9436	9378	9371
70	9161	9387	9438	9436	9350	9364
71	9200	9498	9485	9465	9402	9399
72	9171	9408	9478	9450	9386	9306
73	9163	9436	9450	9447	9365	9343
74	9189	9400	9463	9425	9391	9320
75	9190	9435	9462	9478	9402	9373
76	9118	9362	9490	9434	9402	9367
77	9157	9360	9459	9422	9282	9357
78	9152	9333	9456	9435	9338	9343
79	9171	9457	9459	9402	9388	9325
80	9197	9482	9510	9448	9431	9385
81	9197	9413	9493	9502	9311	9341
82	9142	9434	9424	9439	9308	9335
83	9143	9479	9515	9466	9458	9390
84	9095	9438	9514	9482	9372	9288
85	9201	9444	9517	9448	9296	9370
86	9203	9456	9508	9405	9336	9324
87	9128	9426	9464	9417	9370	9313
88	9149	9363	9477	9423	9346	9344
89	9199	9367	9462	9447	9353	9352
90	9173	9416	9437	9414	9403	9350

91	9191	9456	9475	9439	9410	9347
92	9226	9421	9528	9448	9353	9365
93	9161	9409	9467	9449	9411	9304
94	9162	9405	9470	9416	9347	9361
95	9193	9425	9486	9416	9376	9327
96	9131	9384	9465	9456	9327	9297
97	9178	9413	9431	9437	9362	9351
98	9188	9458	9461	9477	9428	9346
99	9182	9442	9477	9454	9408	9374
100	9172	9459	9484	9443	9318	9348

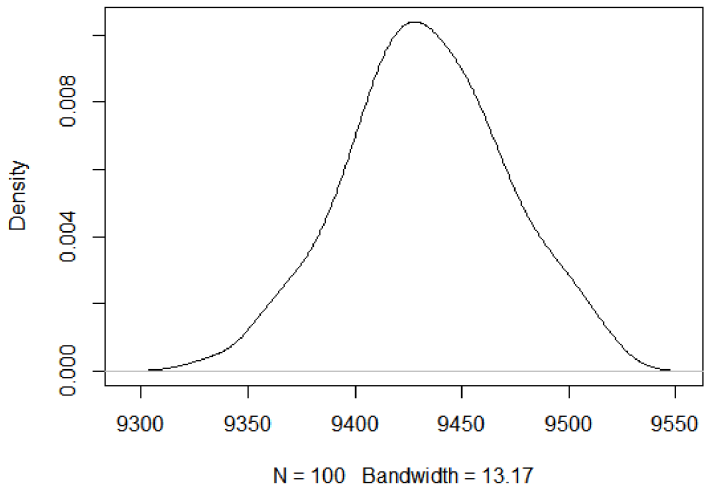
Appendix E: Density Curves for Data

The following density curves were generated using kernel density estimation, and are used to demonstrate the shape of the distribution of each set of results and highlight any skew in the distribution.

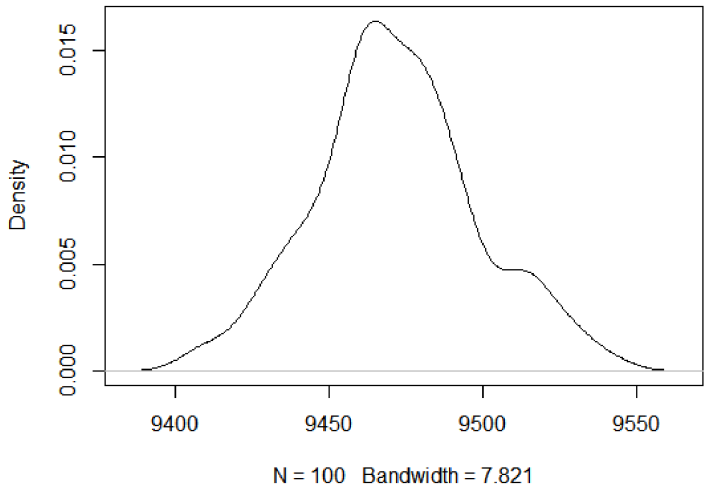
Sigmoid



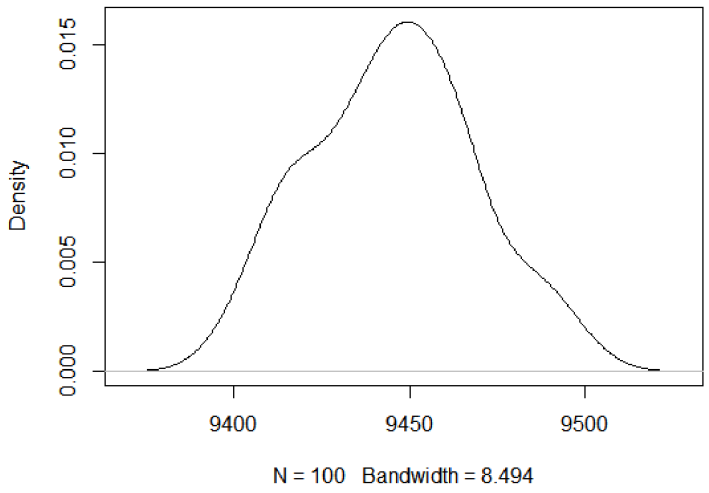
Leaky ReLU



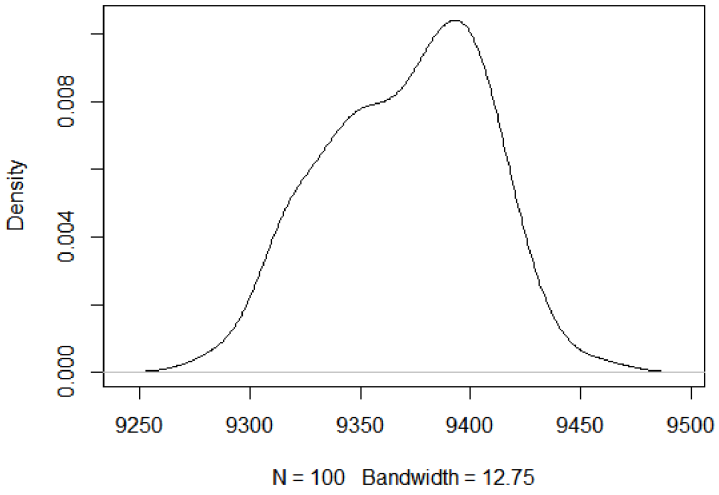
Swish



Softplus



Hyperbolic Tangent



Inverse Tangent