Biologically Inspired Computation: *Multi-Class Classifier Neural Network*

JACK ROME[1]

[*] Student Number: H00219766, email: jar7@hw.ac.uk

1. INTRODUCTION

The network is defined as an object, using a tuple to represent the layer structure, and is the 'propagate' method is called to train the network. A subset of the training data is taken due to time constraints. All formulae will denote 'L' as 'layer' and 'n' as 'neuron'. In this report I have detailed all math required for completeness, regardless of simplicity. Such math relates heavily to the calculation of the gradient descent model.

A. Learning Rate

An exponential decaying learning rate is set up so that the adjustments are less rapid as the network iterates. This allows for less meaningful adjustments as the network converges.

$$\alpha = \alpha * e^{-kt}$$

Where α is the learning rate, k is a very small integer (0 <= k <= 0.001) and t is the number of iterations. From my last report, it was noted that a small value of α led to better performance.

2. GRADIENT

^{Ref1,2} To find the adjustments to the weights and biases, the chain rule was used to find the derivative of the cost with respect to the weights and biases. Following the chain rule, the derivative can be found to be the product of the derivative of the activation function, the predicted output and the cost:

$$\frac{dC^o}{dw^L} = \frac{dz^L}{dw^L}\frac{da^L}{dz^L}\frac{dC^o}{da^L}$$

$$\frac{dC^o}{db^L} = \frac{dz^L}{db^L}\frac{da^L}{dz^L}\frac{dC^o}{da^L}$$

A. Prediction

The output of a neuron is calculated as the dot product between the weights the input predictions from the image or previous layer of neurons. This prediction is denoted as 'z'.

$$z_n^L = \begin{bmatrix} w_{n,0}^L \\ ... \\ w_{n,N}^L \end{bmatrix} \cdot \begin{bmatrix} a_0^{L-1} ... a_N^{L-1} \end{bmatrix} + b_n^L$$

The partial derivatives can be calculated as:

$$\frac{dz_n^L}{dwL_n} = a^{L-1}, \frac{dz_n^L}{dwL_n} = 1$$

B. Loss

^{Ref3} The loss is calculated using 'square loss' method.

$$Loss = \sum_{L=0}^{N} (Y^L - a^L)^2$$

This gives the differential for the loss with respect to 'a':

$$\frac{dC^o}{da^L} = -2 * (Y_n^L - a_n^L)$$

C. Activation

^{Ref4} The activation of 'z' is denoted as 'a' and the derivation of the activation, $\frac{da^L}{dz^L}$. Experiments have been set up to test the effectiveness of two functions using the same neural network topology. The two activation functions used are Sigmoid(z) and Tanh(z)

C.1. Sigmoid

The sigmoid function and its derivative is calculated as follows:

$$g(z) = \frac{1}{1+e^{-z}}$$

$$g'(z) = g(z)(1-g(z))$$

C.2. Tanh

The tanh function and its derivative is calculated as follows:

$$g(z) = \frac{2}{1+e^{-2z}} - 1$$

$$g'(z) = 1 - g(z)^2$$

3. EXPERIMENTS

The following networks have been set up for testing with differing learning rate models and activation functions:

- L = (4,3,10). This is established in the hope of testing the algorithm with a predicted under fitting topology.
- N = (16,16,10). This is established as a standard topology. The expectation is for this to perform well.
- M = (18,16,12,10). This is to test an over-fitting topology.
- O = (30,10). This is to test if the number of layers is a factor in the algorithm

4. RESULTS

The cost, refereed to as 'avg loss' in the graphs, are the average of losses in all of the images during one iteration. ie. The sum of all losses in one propagation divided by the number of images used. In this case, only 5000.

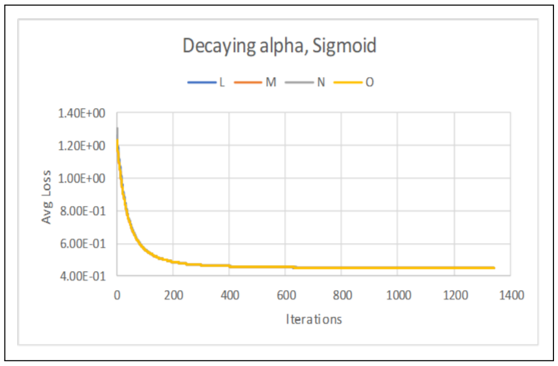

Fig. 1. Plotted graph of cost vs time. Networks tested with Sigmoid as the activation function. Learning rate started at 0.01 and decayed exponentially.

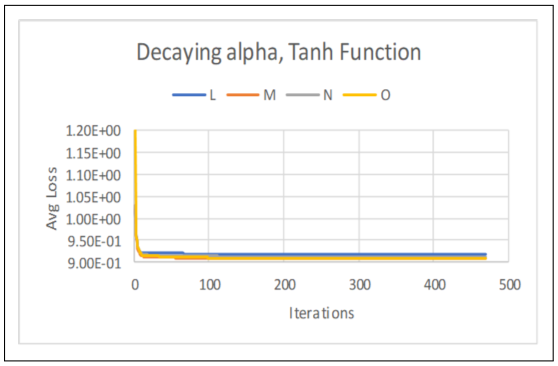

Fig. 2. Plotted graph of cost vs time. Networks tested with Tanh as the activation. Learning rate started at 0.01 and decayed exponentially.

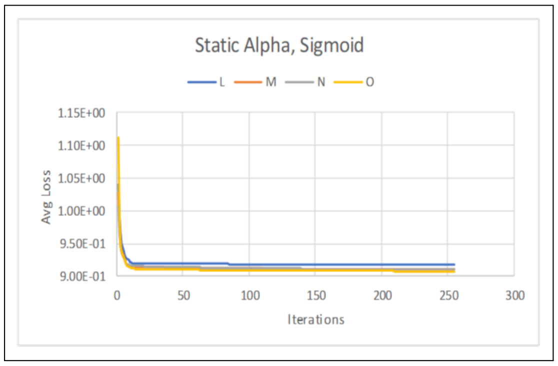

Fig. 3. Plotted graph of cost vs time. Networks tested with Sigmoid as the activation function. Learning rate started at 0.01 remained static.

5. CONCLUSIONS

With more time, I would improve my networks in the following Ways:

- The network would use all 60000 entries of the 'fasionMNist' data-set. This would hopefully lead to a better set of tests after convergence.
- I would have implemented batch learning into the propagation of the networks. This method has been utilized to achieve more successful networks.
- A supervised learning method would be added to routinely compare the test set to the network so that it can avoid falling into a local minima.

6. CITATIONS

- 1. Chapter 3 of Supervised Sequence Labelling with Recurrent Neural Networks - Alex Gravos
- 2. https://en.wikipedia.org/wiki/Backpropagation. Back Propagation and derivative calcualtion
- 3.https://en.wikipedia.org/wiki/Loss_functions_for_classification. Square loss function
- 4. Lecture Slides: Lecture 3 - Multi-layer Perception

7. APPENDIX

Code written in Python 3.7. Algorithm is easily executed. Network objects need instantiated and propagated. Notes on Code:

- Cross-Entropy Loss Function and Softmax Functions are implemented yet unused. I had implemented these functions but could not find the math to use these in gradient descent. Advice was taken from feedback on the previous assignment and so the cross-entropy loss function is able to handle bad cases such as a=0, a=1. Note, I understand that Sigmoid and Tanh are not to be used for multiclass problems, however, I did not have softmax to be operating correctly and so I had to use these due to lack of time.
- Output values for train and test set is converted to matrix. I converted the integer arrays into matrices of zeros and ones. The one denotes which node should be lit up.
- Learning rate decay: A second gradient descent algorithm was tried and implemented. However, due to timing, I do not have tangible results using this and so the two methods cannot be compared at this moment. It would have been beneficial to also test and compare a static learning rate.