COMP5329 Deep Learning - Semester 1 2021: Assignment 1

Group members:
Jesse Serina Narvasa(500525438),
Patrick McLennan (500448953),
Sushmita Shrikrishna Jadhav(500393860)

April 25, 2021

1 Introduction

1.1 Aim of study

The report aims to build a multi-layer classification model on a huge dataset containing 10 classes and data samples of 128 features. Multi-layer Perceptron is supervised learning. The report emphasis the implementation of modules like Relu Activation function, optimization using Momentum method, Weight Decay, Dropout, Mini Batch training and Batch Normalisation without using external libraries like Sklearn, Pytorch or Keras.

1.2 Importance of study

Since the first Perceptron designed by Rosenblatt in 1958, Artificial Intelligence and Machine learning has progressed significantly to a point where it is implemented in the majority of the technology today. This study sets out to expand on the Perceptron by building a Multi-Layer Perceptron (MLP) using mostly native python and basic data structure/manipulation packages (Numpy) and fit this to the provided dataset.

Despite the breakthrough with Rosenblatt's single neuron perceptron, this simple design had limitations when it came to complex/non-linear problems and decision making. With the state of Deep Learning in modern society, it is important to gain an understanding of the logic and implementation of a Neural Network by creating one manually as current programming packages are developed to the stage where minimal understanding of the underlying process is required for use in a basic sense. In general, the study of neural networks is very important in the field of data science due to the wide range of designs and uses when combined with current (and future) technology.

1.3 Dataset

The dataset used for this study consists of 128 features, with the training set containing 50,000 observations and the test set containing 10,000 observations. These are equally split up into 10 different classes of values 0 to 9, which allows our classification model learn from a balanced dataset.

2 Methods

2.1 Preprocessing

Pre-processing methods were added to ensure that raw data is modified to achieve better efficiency and lower loss of the multi-layer perceptron model.

2.1.1 Normalisation

Feature scaling the raw data is very important which is carried out by Normalisation of data. The input data was normalized before passing it through the multi-layer perceptron model. The classical normalization formula was applied to convert the input features in the range of [0,1] (Brownlee, 2021):

$$X = \frac{x - min(x)}{max(x) - min(x)} \tag{1}$$

2.1.2 standardisation / Whitening

Another technique to re-scale the dataset is standardisation, also known as Whitening within the Neural Networks community, which is very similar to normalization. In this technique, the data is centered around the mean with a unit standard deviation (Brownlee, 2021).

$$X_{stand} = \frac{x - mean_x}{StandardDeviation_x} \tag{2}$$

2.1.3 Data Shuffling

During the process of fitting the model, an extra step of shuffling the data for each epoch was implemented to ensure that the neural network does not pick-up on the pattern of labels which the mini-batches may provide. As an example, feeding the network data which are not shuffled per epoch will allow the model to recognise that the pattern of labels coming through and also influence the direction of how the weight matrix and bias vector are updated.

2.2 Principles of Different modules

The modules implemented within the Multi-Layer Perceptron are listed below with a brief background on their relevant theory.

2.2.1 Multiple Hidden Layers

A Perceptron is a simple model to classify binary problems, with its primary limitation being able to handle datasets with no clear boundary. As such, to handle non-linear classification problems Multi-layer Perceptron (MLP) is used. Multi-layer Perceptron (MLP) or Neural Network utilises feed-forward and back-propagation with the layers itself consisting of the input layer, output layer and most importantly the hidden layer(s) where the non-linearity within the model is introduced (Thomas, 2021). The neurons (units) in each hidden layer extract the data from input layer and pass it to other set of neurons present in the following hidden layer. The more the number of hidden layers the more the features of the input data are captured on a deeper level, thus aiding in better classification results as various relationships between the inputs are discovered in the process(Thomas, 2021). So, including more hidden layers helps the MLP to better handle complex or huge dataset input for classification.

2.2.2 Relu

ReLU was added as an activation function along with softmax, tanh, logistic activation functions. The advantage of Relu over other method as it overcomes the vanishing gradient problem which activation functions such as logistic suffers from, due to the very small gradient value in the lower and upper ends of the activation function. Another advantage is the simplicity in calculation which also allows the model to run faster with higher performance ("Xu, C (2021). Lecture 4: Regularizations for Deep Models, Lecture Slides, COMP5329: Deep Learning, The University of Sydney, delivered 26 March 2021.", 2021).

Formula for Relu Activation function is ("Xu, C (2021). Lecture 4: Regularizations for Deep Models, Lecture Slides, COMP5329: Deep Learning, The University of Sydney, delivered 26 March 2021.", 2021):

$$R(x) = \begin{cases} x, & \text{if } x > 0. \\ 0, & \text{if } x <= 0. \end{cases} \tag{3}$$

The resulting derivative of Relu Activation fuction is("Xu, C (2021). Lecture 4: Regularizations for Deep Models, Lecture Slides, COMP5329: Deep Learning, The University of Sydney, delivered 26 March 2021.", 2021):

$$\frac{dR(x)}{dx} = \begin{cases} 1, & \text{if } x > 0. \\ 0, & \text{if } x <= 0. \end{cases} \tag{4}$$

2.2.3 Momentum using SGD

Momentum is one of the optimization methods used to accelerate Stochastic Gradient Descent(SGD) in the relevant direction that reduces the cost function ("Xu, C (2021). Lecture 4: Regularizations for Deep Models, Lecture Slides, COMP5329: Deep Learning, The University of Sydney, delivered 26 March 2021.", 2021). It achieves this by adding an additional momentum parameter during the weight updates, which can be characterised by the addition of momentum experienced by a ball rolling down a steep slop on top of the gradient at the existing point. This allows for faster convergence, but also dampening any oscillations that is generally experienced with SGD, as each single observation triggers a weight update that may deviate from the global minimum of the dataset as a whole ("Xu, C (2021). Lecture 4: Regularizations for Deep Models, Lecture Slides, COMP5329: Deep Learning, The University of Sydney, delivered 26 March 2021.", 2021).

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta) \tag{5}$$

$$\theta_t = \theta_{t-1} - v_t \tag{6}$$

Where:
v_t is the 'trajectory' or 'velocity' parameter
η is the learning rates
∇ is the gradient of the loss function with respect to the weight or bias parameter (θ)
θ is the parameter update
$\theta - 1$ is the parameter value from the previous update (or initialisation)

2.2.4 Dropout

Dropout is one of the regularization techniques which helps to overcome the over-fitting of model. This method random nodes in every pass is dropped out, this makes the following layer to not be exactly dependent on the nodes of the previous layer and thus, overcomes the overfitting problem (Kristiadi, 2021). To implement Dropout, during the training phase, the hidden layers are assigned the probability of any given neuron being dropped out from the network (p), considering there are n neurons in a hidden layer the expectation of number of neuron becomes n*p at each Dropout(Kristiadi, 2021).

For example if one hidden layer we have 560 neurons and the probability of dropout is set to p=0.5. Then after the result of drop out will be 280 neurons as half of the total neurons will be dropped. This dropping of neurons is done randomly by multiplying the hidden layer array with a binomial array of size n (number of neurons). Multiplication of the hidden layer with the randomly generated binomial array makes half of the neurons drop out in a hidden layer. The following formula is used:

$$HiddenLayer = HiddenLayer * BinomialArray \tag{7}$$

During the backprop, it is necessary to ensure we remove the dropped neurons from the network so that the gradient does not flow through them. To remove the dropped neuron is done easily by just considering the multiples array of hidden layer with the random binomial array.

2.2.5 Weight Decay

Another method to control overfitting is regularization by using the Weight Decay method. In this method, the growth of the weights in the network is limited by adding a penalty to the original loss function. This decreases the value of each weight on each training iteration ("Xu, C (2021). Lecture 4: Regularizations for Deep Models, Lecture Slides, COMP5329: Deep Learning, The University of Sydney, delivered 26 March 2021.", 2021).

$$\theta = \theta - (0.02 * \theta) \tag{8}$$

or

$$\theta = \theta - 0.98 \tag{9}$$

2.2.6 Softmax Function

Softmax function is used within multi-class classification problems, akin to how logistic function is used within binary classification problems for assigning the predicted class of the model. This is achieved by calculating the natural exponential of the input for a given neuron, which will be treated as the input for that specific class, and divided against the sum of exponential functions of the other classes. This denoted by the following formula ("Xu, C (2021). Lecture 4: Regularizations for Deep Models, Lecture Slides, COMP5329: Deep Learning, The University of Sydney, delivered 26 March 2021.", 2021):

$$Z_k = \frac{e^{net_k}}{\sum_{n=1}^{K} e^{net_i}} \tag{10}$$

Derivative formula used during backpropagation is (Roelants, 2021) :

$$\frac{dLR(x)}{dx} = y\_hat - y \tag{11}$$

Where:

net is the input vector of K-dimensions, and K is the number of classes within the dataset;
Z_k is a K-dimensional vector which has the probability values ranging from 0 and 1 for each class.
y_hat are the probabilities of the observation belonging to the different class;
y is the one-hot encoded vector containing 0 and a value of 1 in the index of the correct class.

As such, this vector of length K is the output of the neural network, with softmax used to provide the probability of an observation belonging to a particular class.

2.2.7 Cross Entropy Loss

Since the output of our model is a probability distribution, cross-entropy is used as our loss function. This is calculated by getting the sum of the element-wise multiplication between y and natural log of y_hat (Bushaev, 2021). Since mini-batch training is implemented within the model, the loss is further divided by the size of the batch to get the average. Moreover, the loss is also further decreased through the multiplication of the weight decay parameter.

$$\frac{\sum_{k=1}^{k} y_k * log(y\_hat_k)}{batch\_size} \tag{12}$$

2.2.8 Xavier's Initialisation

Initializing all weights as 0.00 leads to neurons computing the same output and also same gradient computation is observed ("Xu, C (2021). Lecture 4: Regularizations for Deep Models, Lecture Slides, COMP5329: Deep Learning, The University of Sydney, delivered 26 March 2021.", 2021). If very small values are assigned to the weights very slow learning is observed and problem of vanishing gradient is observed during forward pass. On the other hand, if large values are assigned to weights, the problem of exploding gradients is observed ("Xu, C (2021). Lecture 4: Regularizations for Deep Models, Lecture Slides, COMP5329: Deep Learning, The University of Sydney, delivered 26 March 2021.", 2021). Using Xavier's Initialization helps to keep the variance same across every layer. The weights are random numbers generated by calculating a uniform probability distribution between the range $-(1/\sqrt{n})$ and $(1/\sqrt{n})$ (Dellinger, 2021).

$$weight = U[-(1/\sqrt{n}), (1/\sqrt{n})] \tag{13}$$

Where, n is number of inputs to the node

2.2.9 Mini Batch training

Mini Batch Stochastic Gradient Descent(SGD) which works by splitting the training dataset into smaller batches which is used to calculate the model error and update model coefficients. It finds a balance between the SGD and batch gradient descent. Using, Mini batch training the model update frequency

increases noticeably and leads to better convergence as it avoid local minima("Xu, C (2021). Lecture 4: Regularizations for Deep Models, Lecture Slides, COMP5329: Deep Learning, The University of Sydney, delivered 26 March 2021.", 2021).

2.3 Design of Best model

Post through testing hyper-parameter tunning, of the best MLP model is set to run for 200 epoch with parameters mentioned in Table 2. The batch training is used in the MLP model training which improves the model training time substantially. Standardisation is used for preprocessing the datasets rather than using normalization technique. Another preprocessing technique Data shuffling is also before with the batch training approach. For the given model, 1 hidden layer gave better accuracy and loss value rather than using multiple hidden layers. Relu activation showed better results compared to Tanh and Logistic activation function as it overcomes the problem of vanishing gradient. For regularization, dropout and weight decay proved to be effective.

Sr. No.	Module	Best Parameter
1.	Number of Neurons in each layer	120
2.	Activation function	Relu
3.	Number of hidden layers	1
4.	Preprocessing	Standardisation
5.	Momentum	None
6.	Weight Decay	0.98
7.	Dropout	0.5

Table 1: Design of Best MLP model

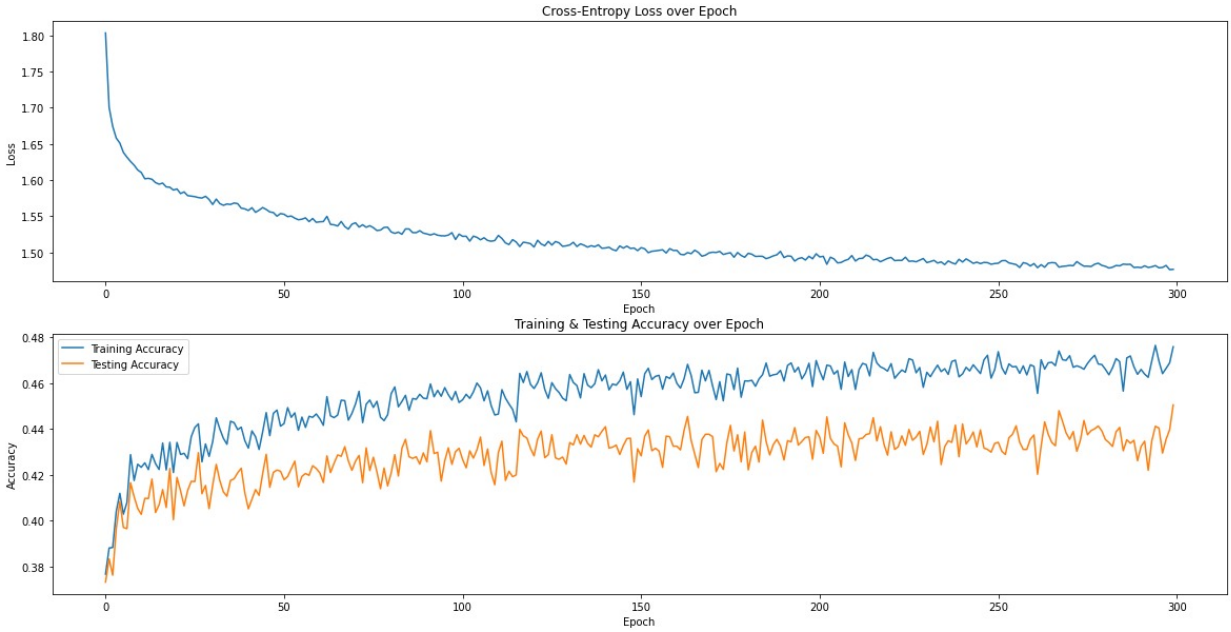

Figure 1: Loss and accuracy Curve of best model

The final Cross-Entropy Training loss of the best model was 1.4764, the Training Accuracy was 47.574%, the Testing Accuracy was 47.574% and the Average F1 Score was 0.4374.

3 Experiment Results

3.1 Module Analysis

The methodology we've chosen to undertake to determine the value-add of the different neural network techniques/modules (outlined in the second column of the table), specified within the previous section, is to perform ablation studies with our neural network model.

For this exercise, we have chosen to provide a different parameter input when assessing each technique, whilst keeping the remaining parameters as the default best-practice parameter, marked as Default Parameter within the table, ensuring that the tests conducted are consistent against each other. Doing so allows us to determine the usability of including the module within the model, but also in knowing which parameter might suit our problem best.

Techniques for Evaluation

Sr. No.	Module	Default Parameter
1.	Number of Neurons per layer	100
2.	Activation function	Relu
3.	Number of hidden layers	3
4.	Preprocessing	Standardisation
5.	Momentum	0.9
6.	Weight Decay	0.98
7.	Dropout	0.5

Table 2: Default Parameters for MLP

The following parameters are tested within the ablation study, to determine its usefulness from the base model which is outlined under "Design of Best Model".

Ablation Study Results per Technique and Parameter

Techniques	Default Parameter	Test Accuracy(%)	Training Loss
Number of Neurons (50/100/120)	100	50: 33.36 100: 40.79 120: 42.28	50: 1.7944 100: 1.6232 120: 1.5828
Activation Function (Relu/Logistic/Tanh)	Relu	Relu: 40.31 Logistic: 38.08 Tanh: 39.56	Relu: 1.6003 Logistic: 1.6830 Tanh: 1.6069
Hidden Layers (1/3/5)	3	1: 43.26 3: 40.79 5: 35.88	1: 1.5428 3: 1.6232 5: 1.7375
Preprocessing (normalization/ Standardisation/None)	Standardisation	Standard: 40.31 Normal: 9.99 None: 39.78	Standard: 1.6003 Normal: 2.2565 None: 1.6217
Momentum Optimisation (None/ 0.5/ 0.9)	None	None: 40.46 0.5: 31.8 0.9: 10.16	None: 1.588 0.5:1.843 0.9:2.25
Weight Decay (0.98/ None)	0.98	0.98: 40.46 None: 39.96	0.98: 1.588 None: 1.631
Dropout Probability (1/0.5/0.8)	0.5	1(no dropout): 0.1 0.5: 40.36 0.8: 24.93	1:0.2565 0.5: 1.596 0.8: 1.905

Table 3: Hyper Parameter tuning

3.1.1 Number of Neurons

The number of Neurons in each layer was examined to determine more neurons would increase performance, as expected, or show signs of overfitting by observing lower testing accuracy. It was found that increasing the number of Neurons in each layer did increase the performance of the MLP, with the resulting loss for 120 Neurons (the highest) at 1.5828 vs 1.6232 for 100 Neurons and 1.7944 for 50 Neurons. The testing accuracy was also the best with 120 Neurons at 42.28% vs 40.79% for 100 and 33.36% for 50 Neurons which indicated there was no signs of overfitting.

It is somewhat expected that more Neurons would increase the performance as this factor increases the complexity of the Network, and the improvements were noticeable, so 120 Neurons in each hidden layer was selected to proceed with the chosen model.

3.1.2 Activation functions

In order to ensure that the optimal activation is used within our neural network model, we've tested the effects of using different functions for introducing non-linearity. Specifically, we've tested the difference between tanh, logistic, as well as relu activation functions.

Out of the three, logistic function performed the worse with a test accuracy score of 38.08%, although it is only marginally worse compared to the other two. The loss encountered during training of the neural networks with logistic function is relatively stable with minimal zig-zagging over 200 epochs. Whilst this activation function fared the worst of the three, it's quite possible that it would further benefit from

increased number of epochs for training, although it is notable that it is the slowest to learn, based on the loss over number of epochs.

The second best activation function is tanh. As with logistic, it only performed marginally worse than relu, with the loss over 200 epochs also suggesting that it would benefit from further training. It is however, the slowest to train amongst the three.

Finally, relu provided us with the highest test accuracy and lowest training loss with 40.31% and 1.6003 respectively. Compared to the first two above, relu's main advantage is it's ability to fare better with vanishing gradients, as it retains the value of 1 for the derivative of its upper asymptote. In contrast, logistic and tanh would have an equally close to zero gradient on its lower and upper asymptotes which would then result in no further updates to the weights as gradients reach zero.

We will therefore be using relu as our activation function for the best model, due to its versatility and ability to handle vanishing gradients, particularly with deeper networks.

3.1.3 Hidden Layers

The number of hidden layers was examined across 1, 3 and 5 to determine if the models performance improved significantly with the additional complexity. It was expected that in general, more hidden layers would improve performance due to the higher degree and complexity of decision boundaries feasible by the MLP. However, the tests proved the contrary where 1 hidden layer had the lowest loss of 1.5428 compared to 1.6232 for 3 hidden layers and 1.7375 for 5 hidden layers as well as testing accuracy of 43.26% for the 1 hidden layer vs 40.79% for 3 and 35.88% for 5.

Despite the general expectations for the relationship between the number of hidden layers and performance, the results mentioned above that adding further complexity may have caused the model to overfit (reflected by the lower testing accuracy). It is also worthwhile to note that the lack of context regarding the dataset meant that the expected required complexity could not be determined, and so the type of data could have been such that the decision boundaries are not too complex.

Given the other parameters (e.g. activation functions and/or number of neurons) were held equal in this ablation study, an extension could be to incorporate the comparison of the number of hidden layers with combinations of activation functions. Given the possible combinations and runtime of the Network training, such comparison was not pursued in this study and thus 1 Hidden Layer was selected for the chosen model.

3.1.4 Pre-processing

It is well understood that pre-processing must be performed prior to feeding the data to the input layer of the model. In building our model, we have tested the use of pre-processing, in particular, the three varieties being: normalisation, standardisation/whitening, and no pre-processing at all.

Running the model with standardised inputs performs the best in our observations. It is clear that ensuring the dataset being of zero mean and unit variance allows the model to learn faster and have a higher accuracy with less epochs. This aligns with literature where standardisation is often referred to as whitening within the data science community and in the context of neural networks. With standardisation, we are able to achieve 40.31% accuracy, keeping the remaining hyperparameters set to the default parameter.

In contrary, normalisation seems to have the adverse on neural networks that's expected from pre-processing. Specifically, normalisation in this context is min-max normalisation where all the values within the dataset will be between the values 0 and 1. Performing normalisation seems to prevent our model from learning, causing the activation functions to die and inherently stop the weights from further updating as the derivative reaches 0. This then causes the loss functions to get stuck within a narrow range and consequently have poor accuracy, as the neural network has not been able to properly learn.

Interestingly, performing no pre-processing yields in better results, with a test accuracy of 39.78%, when compared to using normalisation. It is however still marginally worse than using standardisation. The conclusion for this section of ablation study is to make use of standardisation going forward.

3.1.5 Momentum Optimisation

SGD Optimization via Momentum was implemented and was compared across 0.5 and 0.9 for the Momentum parameter as well as no implementation. It was expected that applying the standard 0.9 as the Momentum parameter would improve the model fit and performance, however it was found that omitting momentum had the best results with the Loss as 1.588 vs 1.843 for 0.5 and 2.25 for 0.9 as well as Testing Accuracy of 40.46% vs 31.8% for 0.5 and 10.15% for 0.9.

While this does not follow expectation, it is possible that including momentum caused divergence in the learning and thus the yielded poor results. Therefore, omitting Momentum optimisation was selected for the chosen model.

3.1.6 Weight Decay

Weight Decay was testing by decreasing the weights by the constant value of 0.98 or not doing reducing the weights at all. According to the theory, weight decay is a regularization technique, thus should produce better loss and accuracy value. This, was tested and the results matches with the theory as there was a difference noticed in the loss and accuracy values. The loss reduces from 1.631 to 1.588 and also there accuracy boosted from 39.96% to 40.46%. This stated that dropping the weight did not hamper the accuracy and loss of the model but instead incresed it.

3.1.7 Dropout Probability

The MLP network was tested by dropping the weights by the ration of 0.5, 0.8 or not dropping them at all(hence the dropout probability is 1). In case of no dropout, there was an abnormal behaviour notices both for loss and accuracy values. The loss obtained was 0.2565 and accuracy was 0.1%. Next ration tested was dropout probability of 0.5, where the loss and accuracy were 1.596 and 40.36% respectively. With the dropout of 0.8 ration were loss=1.905 and accuracy=24.93%, This clearly stated that dropping half the number of neurons is better than dropping 4/5th of the neurons, which is a bit reduction in neurons. This clearly states that, with less number of neurons the, MLP network is not able to predict the features accurately.

3.2 Justification on your best model

To summarise the justification on the best model, the parameters were selected as per the results above. 120 (the highest) as the number of neurons resulted in the highest test accuracy and lowest training loss. ReLU as the activation function provided the best results for metrics, albeit only slightly better than tanh, however it was selected due to it's recognition as the best activation function for generalisation. 1 hidden layer proved to perform the best, with both test accuracy and loss better than the other number of hidden layers. Standardisation of the data as opposed to Normalisation or no preprocessing also yielded the best results. Omitting momentum also gave best results in terms of the metrics, as did applying Weight Decay of 0.98 vs none at all. Finally, setting the dropout probability to 0.5 highest testing accuracy but the lowest loss was obtained with no dropout however this was unusually low (0.25) and also had much lower testing accuracy (10%).

3.2.1 Runtime comparison

The time taken to run the best model was 2293.85 seconds, or 38 minutes 13.85 seconds, which was far lower than the runtime of the model with the standard parameters which took 3718 seconds, or 61 minutes and 58 seconds.

3.3 Comparison between best model and default model

Furthermore, the performance between the best model will be compared against the performance of the model obtained using the default parameters that we had at the very start of the study. In performing the

comparison, we'll be able to ascertain if the combination of parameters does have a material improvement.

As seen on the graph below, fitting the model with the default parameters performs well within the first 40 epochs before the loss starts to increase, to the point that the loss is greater than the starting loss by approximately the 270th epoch. It is also worth noting that even while the neural network was performing well, with the training loss decreasing, the loss values are still higher at each point of epoch when compared to our best model.

In unison, the accuracy graph shows the inverse relationship between loss and accuracy, with both training and testing accuracy finishing at approximately 22%. It is interesting to note that both training and testing loss are very close to each other, even though it's decreasing, which marks the sign that it is not an issue of over-fitting, but rather that the main issue is due to the learning rate being too high for this set of parameters.

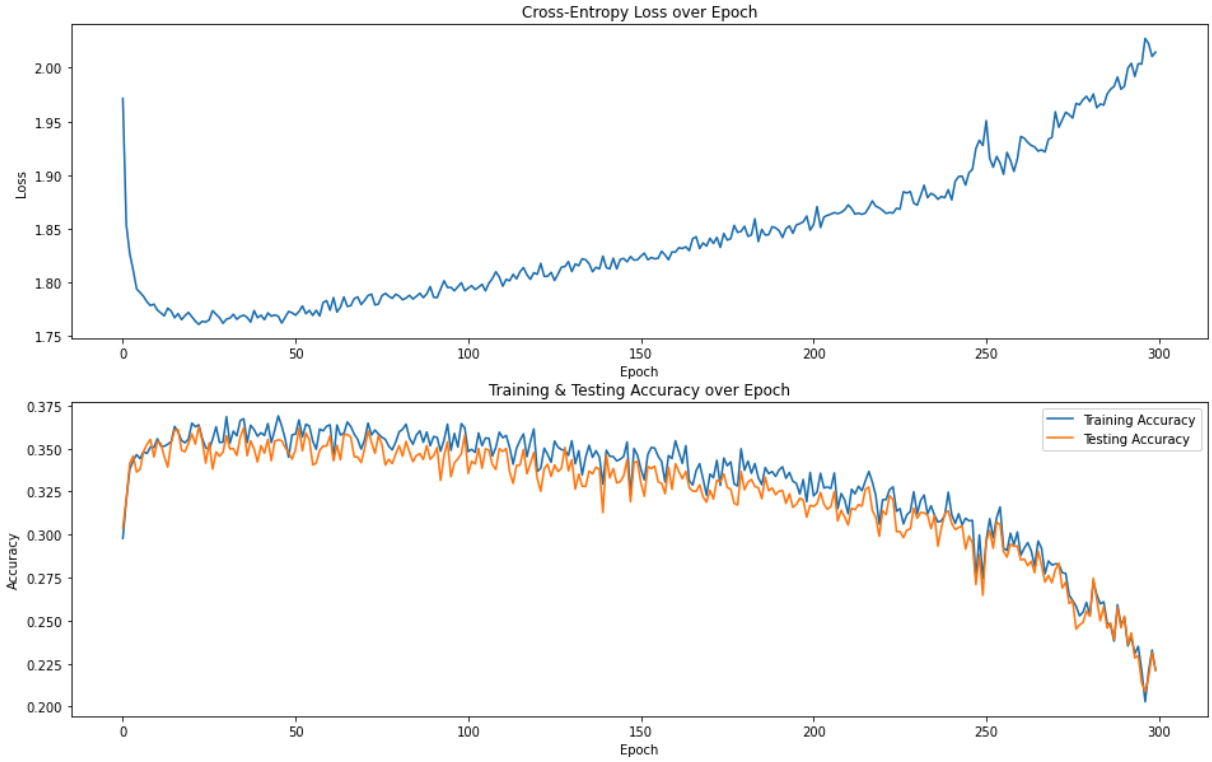

Figure 2: Cross entropy loss and training/testing accuracy graphs for model with default parameters

4 Future Scope

To improve the accuracy and loss values of the model Data Augmentation techniques can be applied and Optimization methods like Adam, Adagrad, Adadelta, RMSProp, etc. Also, implementing Leaky Relu will be a good addition to avoid loosing of neurons(Dead Relu problem). Implementing Batch Normalisation would also help to increase the accuracy scores substantially.

5 Personal Reflection

While forming a MLP network, mostly more than 1 hidden layers are used to form the network but in this case the Best Model uses 1 hidden layer with 120 neurons. This says the entire dependency of the MLP network performance is not entirely dependent on number of hidden layers but also depends on the use of activation functions, optimization, regularization methods. There was a huge difference between the loss and accuracy scores using normalization and standardisation, which says that appropriate use of preprocessing methods is highly recommended. This huge difference was notice whilst performing testing, which states the importance of aggressive testing and hyper-parameter tuning to increase model efficiency. The next observation of the execution time efficiency using batch normalization method.

6 Conclusion

The MLP model generated is tested well and the best design of the model generates a low training loss of 1.4764. The train accuracy is 47.574% and test accuracy is 44.08% which also denoted that the model is not over fitted. The Average F! score of the bets model is 0.4374. The model's efficiency is the result of implementation of standardisation of data set, Relu activation function, batch training, 0.5 dropout probability of of neurons and weight decay.

APPENDIX

STEPS TO RUN THE CODE SUCCESSFULLY

1. Upload the 500525438_500448953_500393860 - COMP5329 2021S1A1 - Code.ipynb in Googel Colab.
2. Upload the dataset files train_data.npy,test_data.npy, train_label.npy, test_label.npy to Google Colab's runtime session storage.
3. Let all files upload completely before running any code cells.
4. Once all the files are successfully uploaded, press 'cntrl+F9' to start the sequential execution of all the cells.
5. The loss graph and loss value of the neural network will be appear as the result of successful execution of the Multi-layer perceptron model. A confusion matrix and metrics per class (Precision, Recall and F1 Score) are in the following outputs, with the metrics aggregated in the final output chunk.

Specifications

The hardware and software specification used to successfully run the code are as follows:

Software
Language: Python3.7 Web-interface: Google Colab

Hardware
Windows system: Windows 10
Processor: Intel® Core ™ i7-5500U CPU @2.40GHz 2.40GHz
Installed memory (RAM): 8.00GB
System type: 64-bitOperating System, x64-based processor

References

Brownlee, J. (2021). *How to use data scaling improve deep learning model stability and performance.* Retrieved from https://machinelearningmastery.com/how-to-improve-neural-network-stability-and-modeling-performance-with-data-scaling/

Bushaev, V. (2021). *Stochastic gradient descent with momentum.* Retrieved from https://towardsdatascience.com/stochastic-gradient-descent-with-momentum-a84097641a5d

Dellinger, J. (2021). *Weight initialization in neural networks: A journey from the basics to kaiming.* Retrieved from https://towardsdatascience.com/weight-initialization-in-neural-networks-a-journey-from-the-basics-to-kaiming-954fb9b47c79

Kristiadi, A. (2021). *Implementing dropout in neural net - agustinus kristiadi's blog.* Retrieved from https://wiseodd.github.io/techblog/2016/06/25/dropout/

Roelants, P. (2021). *Softmax classification with cross-entropy.* Retrieved from https://peterroelants.github.io/posts/cross-entropy-softmax/

Thomas, M. (2021). *Neural networks: Advantages and applications | marktechpost.* Retrieved from https://www.marktechpost.com/2019/04/18/introduction-to-neural-networks-advantages-and-applications/

Xu, c (2021). lecture 4: Regularizations for deep models, lecture slides, comp5329: Deep learning, the university of sydney, delivered 26 march 2021. (2021).