AI201 PA 3 Report
Implementing the Backpropagation Algorithm

Jemima Bian T. Anila (2015-05643)

Date of Submission: May 31, 2024

1. INTRODUCTION

A perceptron is an artificial neuron. A network of perceptrons called a neural network is used in supervised learning of binary classifiers. The network classifies objects by applying activation functions into its input variables and feeding it throughout the network to produce an output. It learns by adjusting the weight of each neuron to obtain the minimum (iterating to the direction of steepest descent) of the cost function. The weight of each perceptron is saved after the cost function is minimized using training data and is used for classification of testing data.

One of the common ways to update the weight of each perceptron is through the backpropagation algorithm. Here, the errors from the output nodes back-propagate to the input nodes. Training is done in three stages - the feed-forward of input training pattern, the calculation and backpropagation of the error, and updating of the weight [1].

There are instances wherein imbalanced datasets are encountered when building a model. Imbalanced datasets can cause oversampling that can lead to overfitting and lower performance of a model. Synthetic Minority Oversampling Technique (SMOTE) is an oversampling technique where the synthetic samples are generated for the minority class to generate a more balanced dataset. It generates new instances with the help of interpolation between the positive instances that lie together [2].

This paper documents the production of a multi-layer perceptron network that uses backpropagation and training with an imbalanced dataset.

2. OBJECTIVES

The objectives of the assignment are the following:

1. Implement a Multi-layer Perceptron Neural Network
2. Train the network using the backpropagation Algorithm on a challenging dataset and different activation functions
3. Understand the effect of varying hyper-parameters such as learning rate parameter, momentum constant, and number of hidden nodes on the performance of the network

3. METHODOLOGY

3.1 Preparing the Dataset

The provided dataset is a highly-imbalanced 3486 samples with 354 features grouped into 8-classes. Figure 1 shows that there are more samples in the first class and least samples in the third class. To balance the data, SMOTE was applied by using 7 nearest neighbors. The resulting training and validation set is more balanced with around 1500 samples for each class for training and 100 samples for each class for validation.

As seen in Figure 2 and 3, the different classes seem to have a big overlap with respect to the first and second feature. This will make classification challenging for simple linear classifiers.

3.2 Multi - Layer Perceptron Algorithm with Backpropagation

A perceptron algorithm with two hidden layers was created that incorporates a momentum term to speed up computations. Training was done by feeding the features of the samples into the input nodes, running different activation functions on to these features as they move forward the network. The error was calculated at the output layer by comparing the prediction with the expected output. This error was then combined with the momentum and fed to the derivatives of the activation functions to adjust the weights and biases of the network. This entire process was done in batches of 8 samples for the training set.

Two networks were created. Network A had hyperbolic tangent (tanh) activation functions for the hidden layers and logistic function for the output layer. Network B had leaky rectified linear unit (reLU) for the hidden layers and logistic function for the output layer.

3.3 Training the Network

Before training the network on the full training set, the hyperparameters were first adjusted to find the best setting for training. Using a shorter epoch of 50 to make computation run faster, one hyperparameter was modified to show different values while other hyperparameter was kept constant. The setting that gives the most optimal combination of low validation SSE and shortest run time was chosen.

The first hyperparameter that was adjusted was the number of hidden nodes. It was tested between 100, 200, and 300. The learning rate was kept at 0.1 and the momentum term α was kept at 0.9.

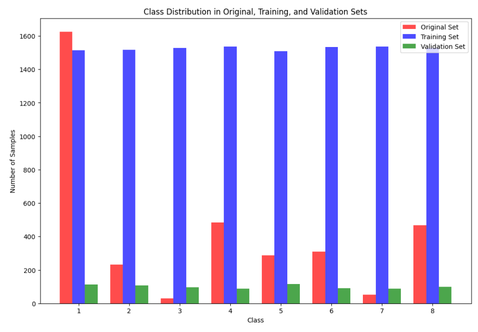

Figure 1: The original dataset (orange) is highly imbalanced, having more samples of the first class. SMOTE was applied to generate synthetic samples of the other classes to create similar sample sizes of the different classes.

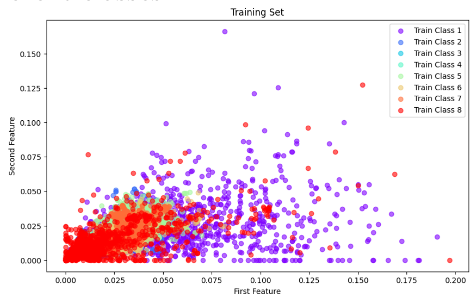

Figure 2: The plot of Feature 1 and 2 of the training set shows a lot of overlaps for the different classes. The first class varies more as it is more distributed.

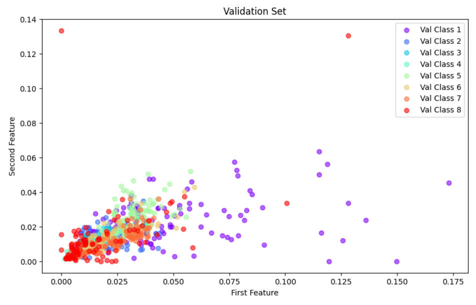

Figure 3: The validation set follows a similar pattern to the training set.

The next hyperparameter that was adjusted was the learning rate. It was adjusted to 0.3, 0.6, 0.05, and 0.01. The last hyperparameter adjusted was the momentum term α. It was adjusted to 0.01, 0.1, and 0.5.

3.4 Testing the Network

The dataset came with a test set. Unfortunately, the labels for the test set was never provided and could not be used in evaluating the performance of the model. Thus, the validation set created earlier was used as a test set to evaluate the performance of the model.

4. ANALYSIS AND DISCUSSION OF RESULTS

Table 1: Training Time, Lowest SSE and its Epoch at Varying Number of Hidden Nodes at Learning Rate = 0.1, Alpha = 0.9, and Epoch = 50

Hidden Nodes	Training Time	Epoch with Lowest SSE	Lowest SSE
100	43.74 sec	41	98.63
200	79.58 sec	29	97.89
300	194.47 sec	41	88.68

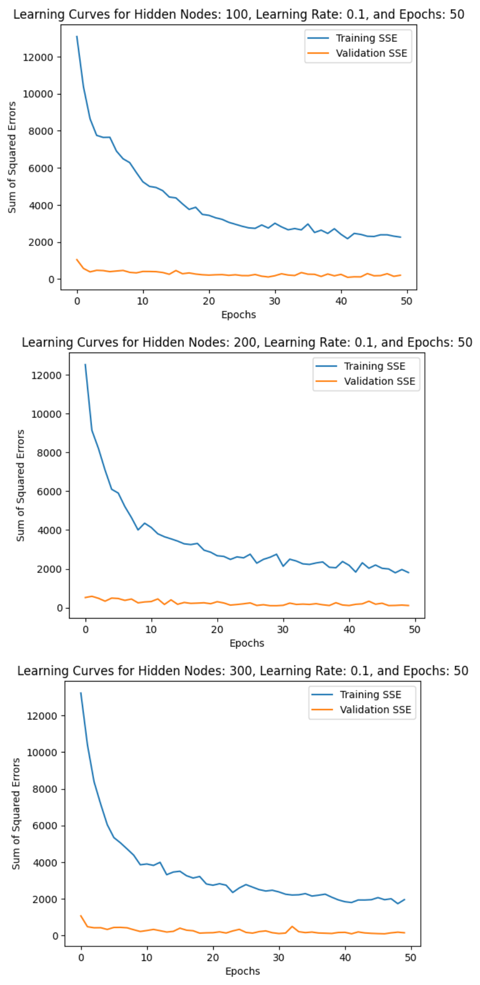

Figure 4: The network goes to a lower SSE in less epochs at higher number of hidden nodes but at the cost of a longer processing time.

Increasing the number of hidden nodes in the network yields to longer processing time but yields a lower SSE in less epochs. This is because increasing the hidden nodes introduces more minute weight adjustments to the features allowing more refined changes to the output.

Table 2: Training Time, Lowest SSE and its Epoch at Varying Learning Rate at Nodes = 300, Alpha = 0.9, and Epoch = 50

Learning Rate	Training Time (sec)	Epoch with Lowest SSE	Lowest SSE
0.01	198.47	48	15.11
0.05	206.04	46	71.76
0.1	194.47	41	88.68
0.3	192.27	47	63.47
0.6	174.65	47	59.66

Decreasing the learning rate by one magnitude helps the

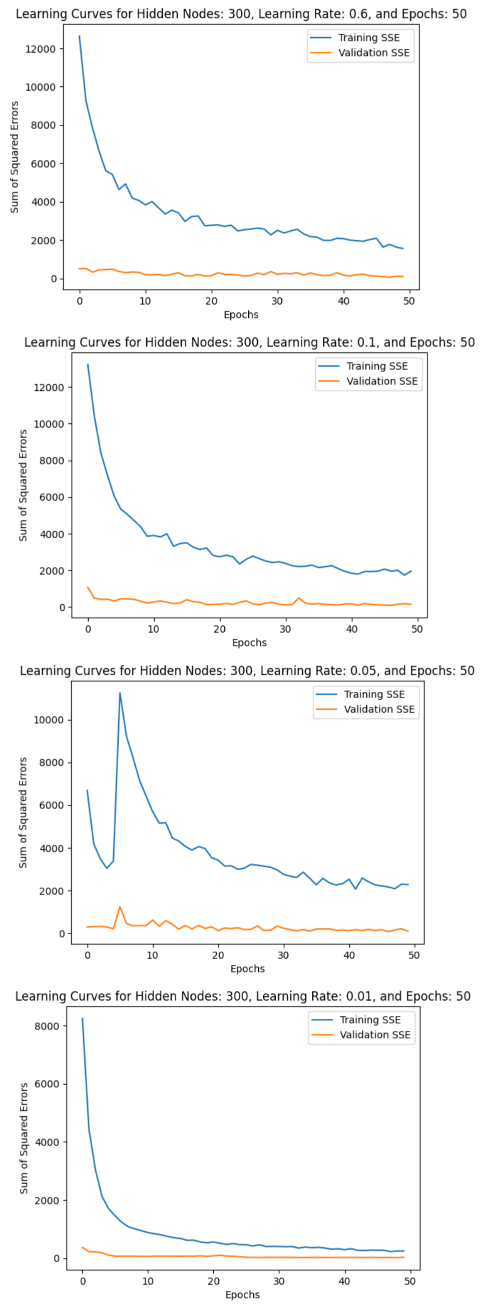

Figure 5: The learning curve becomes smoother and smaller as the learning rate decreases. However, the learning curve seems to be overshooting at 0.05 and overfitting at 0.01.

network reach a lower SSE in the same number of epochs. However, it also leads to overshooting at 0.05 and overfitting in 0.01 as seen in Figure 5. Increasing the learning rate leads to lower SSE but more epochs.

Increasing the momentum constant α resulted into lower convergence because the function overshoots and oscillates around a local minimum. However, decreasing it also resulted into a smoother learning curve suggesting overfitting.

Table 3: Training Time, Lowest SSE and its Epoch at Varying Momentum Constant Alpha at Nodes = 300, Learning Rate = 0.1, and Epoch = 50

Momentum Term Alpha	Training Time (sec)	Epoch with Lowest SSE	Lowest SSE
0.01	161.86	47	16.11
0.1	157.96	38	21.82
0.5	176.41	49	16.79
0.9	176.25	47	83.70

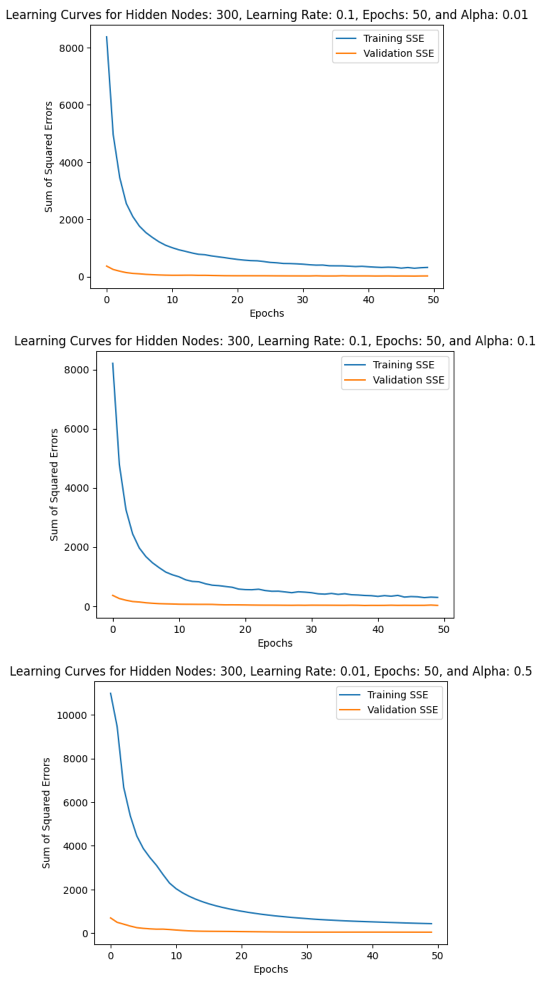

Figure 6: The learning curve becomes smoother and error decreases as alpha decreases, suggesting overfitting. Computational time takes longer and error increases as alpha increases, suggesting overshooting.

Thus, the final parameters selected to train Network A and B are: hidden nodes = 300, learning rate = 0.1, and momentum term $\alpha = 0.5$.

Network A did not perform well on the validation set, with most values being predicted into Class 3 and 8. This bias may be due to exposure to more samples in Class 3 and 8 during training suggesting that the SMOTE done to balance the dataset failed. It could also suggest incorrect hyperparameter setting leading to overfit.

As seen in the accuracy of the models, Network B correctly predicted the class of objects 99.5% of the time vs Network

A that only got correct predictions 6.38% of the time. Moreover, as seen in the precision of the models, if an object is predicted to belong to one class, it is correct 99.5% of the time with Network A but only 1.37% of the time with Network B.

The F1 score shows that Network B performed excellently while Network A has very poor performance. Moreover, the MCC shows that Network B almost always makes no mistakes while Network A is a little worse that random predictions.

It is noted that both Network A (tanh-tanh-logistic) and Network B (leaky ReLU-leaky ReLU-logistic) both overfit the data but Network B results into faster convergence at 2867.83 seconds vs 3940.51 seconds of Network A.

Table 4: Performance Metrics show that Network B performed much better than Network A.

Metric	Network A	Network B
Accurary	0.06375	0.995
Precision	0.01365	0.995
Recall	0.06375	0.995
F1 Score	0.02247	0.995
Matthews Correlation Coefficient	-0.08837	0.994

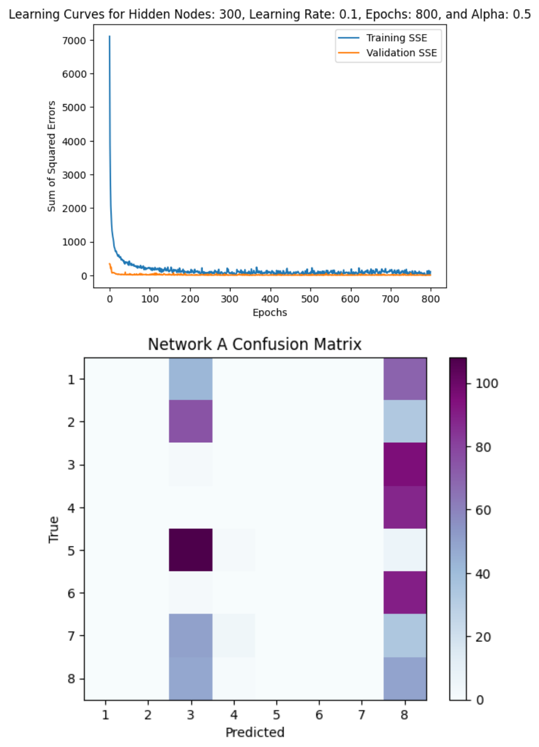

Figure 7: The learning curves show that Network A converged to a low SSE early at epoch = 347 then started oscillating to 5.63. Its confusion matrix shows its prediction is biased towards two classes.

5. CONCLUSION

The number of hidden nodes in a network is proportional to processing time and accuracy. Learning rate is inversely proportional to processing time and accuracy and can result to overfitting at really small values. Momentum constant is inversely proportional to processing time and accuracy but can also lead to overshooting and overfitting.

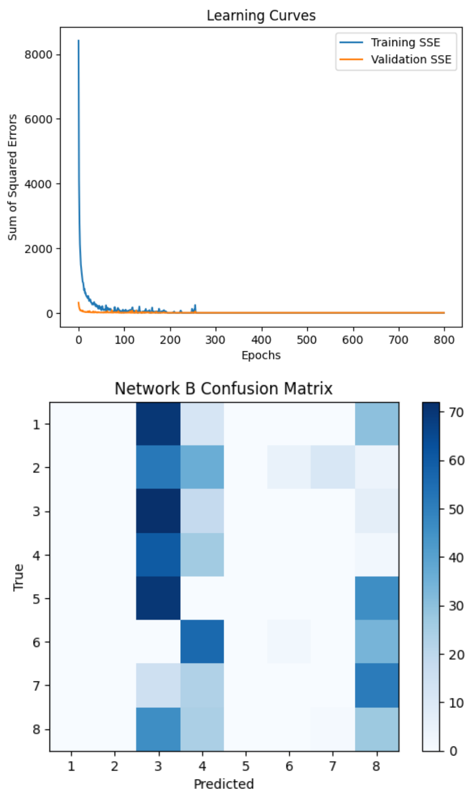

Figure 8: The learning curves show that Network B converged to a low SSE of 5.40 early at epoch = 210. Its confusion matrix shows its prediction is biased towards three classes.

The network with Leaky ReLU activation function performed better than the network with tanh activation function in terms of accuracy, precision, recall, F1 score, and MCC.

6. REFERENCES

[1] Geek for Geeks Website (2023). Retrieved from: https://www.geeksforgeeks.org/backpropagation-in-data-mining/

[2] Satpathy S (2023). Retrieved from: https://www.analyticsvidhya.com/blog/2020/10/overcoming-class-imbalance-using-smote-techniques/