2025

MCMASTER SEP 740 FINAL PROJECT

USING ANN TO PREDICT CREDIT CARD DEFAULTS NEXT MONTH

JULIUSZ GASIOR 400490100

Contents

Problem Definition

The problem being investigated is if a Taiwanese bank can predict if a customer will default (ex. fail to pay) on their next month's credit card bill based on customer characteristics and their payment history from the last 6 months.

This is a binary classification problem where the model will be predicting a 1 (yes, the customer will default next month) or 0 (no, the customer will pay next month).

Relevance

This problem is extremely relevant as it can be used to identify risky customers to the bank for further investigation on how to minimize these risks (i.e. payment plans, reduction in available credit, removing as a customer).

This methodology can be used in other aspects as well. For example, a car manufacturer's accounts receivable department may be able to use such a model to identify high risk suppliers based on their past payment obligations and the type of supplier they are.

Executive Summary

A multilayer ANN model that had gone through 13 experiments to maximize it's accuracy was able to achieve a testing accuracy of about 81.7%. The ANN model used was able to perform as well, from an accuracy point of view, when compared against a multilayer perceptron model used in a research paper using similar data. (Begüm & Ünal, 2019)

Methodology

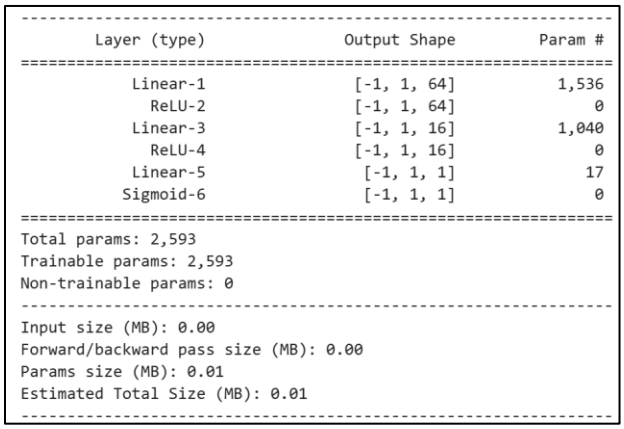
Model Selection

An artificial neural network (ANN) model was chosen due to the nature of the data. The data was not image-based or time-series-based so feed-forward network (FFN) and convolutional neural network (CNN) models were not deemed to be appropriate. The data is relatively straight forward with a clear target, thus an ANN was deemed to be sufficient.

Architecture

The architecture of the model was chosen at random to start the experiments. By experiments #8 through to #11, the model architecture was finalized. These experiments helped determine the breadth and depth of the architecture as well as the appropriate activation functions to use. The number of parameters had increased by 5.4X.

Starting Architecture

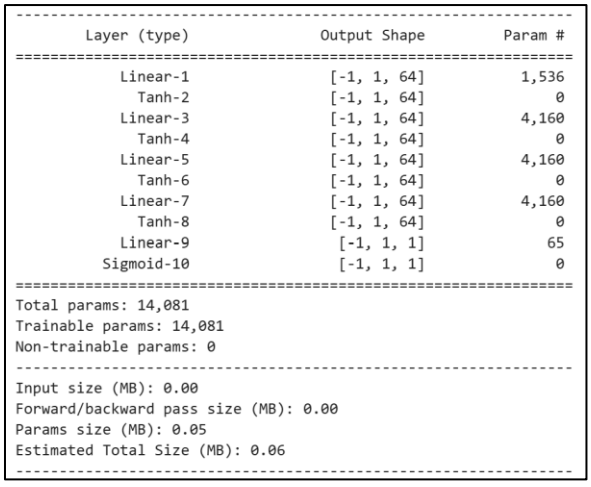


Layer (type)	Output Shape	Param #
Linear-1	[-1, 1, 64]	1,536
ReLU-2	[-1, 1, 64]	0
Linear-3	[-1, 1, 16]	1,040
ReLU-4	[-1, 1, 16]	0
Linear-5	[-1, 1, 1]	17
Sigmoid-6	[-1, 1, 1]	0

Total params: 2,593
Trainable params: 2,593
Non-trainable params: 0

Input size (MB): 0.00
Forward/backward pass size (MB): 0.00
Params size (MB): 0.01
Estimated Total Size (MB): 0.01

Ending Architecture



Layer (type)	Output Shape	Param #
Linear-1	[-1, 1, 64]	1,536
Tanh-2	[-1, 1, 64]	0
Linear-3	[-1, 1, 64]	4,160
Tanh-4	[-1, 1, 64]	0
Linear-5	[-1, 1, 64]	4,160
Tanh-6	[-1, 1, 64]	0
Linear-7	[-1, 1, 64]	4,160
Tanh-8	[-1, 1, 64]	0
Linear-9	[-1, 1, 1]	65
Sigmoid-10	[-1, 1, 1]	0

Total params: 14,081
Trainable params: 14,081
Non-trainable params: 0

Input size (MB): 0.00
Forward/backward pass size (MB): 0.00
Params size (MB): 0.05
Estimated Total Size (MB): 0.06

Techniques

Data exploration and model building were done using Python libraries and in Jupyter Notebooks.

For pulling in CSV data and exploring the data, the pandas library was used. PyTorch was used to build models.

Numpy was used to create data arrays for parameter experimentation and holding performance data like losses and accuracies.

Data Handling

Data Sources

Data on "Default of Credit Card Clients" from UC Irvine, donated by I-Cheng Yeh in 2016 was used. This data contains 30,000 instances describing the credit borrowing and payment history of customers, as well as other characteristics related to the customer.

Default of Credit Card Clients - UCI Machine Learning Repository

Variable	Description	Contents
X1	Amount of credit given	Integer
X2	Gender	1 = male; 2 = female
X3	Education	1 = graduate school; 2 = university; 3 = high school; 4 = others
X4	Marital Status	1 = married; 2 = single; 3 = others
X5	Age (year)	Integer
X6-X11	History of payment in last 6 months	-1 = duly paid; 1 = payment delay of 1 month; 2 = payment delay of 2 months; … 9 = payment delay of 9 months and so on
X12-X17	Amount of bill statement in last 6 months	Integer
X18-X23	Amount of previous payment in last 6 months	
Y	Default Payment	1 = Yes; 0 = No

Augmentations

The dataset comprises of 23 features, all pre-conditioned to be real integers and with no missing values (see Figure 1).

Variable Name	Role	Type	Demographic	Description	Units	Missing Values
ID	ID	Integer				no
X1	Feature	Integer		LIMIT_BAL		no
X2	Feature	Integer	Sex	SEX		no
X3	Feature	Integer	Education Level	EDUCATION		no
X4	Feature	Integer	Marital Status	MARRIAGE		no
X5	Feature	Integer	Age	AGE		no
X6	Feature	Integer		PAY_0		no
X7	Feature	Integer		PAY_2		no
X8	Feature	Integer		PAY_3		no
X9	Feature	Integer		PAY_4		no
X10	Feature	Integer		PAY_5		no
X11	Feature	Integer		PAY_6		no
X12	Feature	Integer		BILL_AMT1		no
X13	Feature	Integer		BILL_AMT2		no
X14	Feature	Integer		BILL_AMT3		no
X15	Feature	Integer		BILL_AMT4		no
X16	Feature	Integer		BILL_AMT5		no
X17	Feature	Integer		BILL_AMT6		no
X18	Feature	Integer		PAY_AMT1		no
X19	Feature	Integer		PAY_AMT2		no
X20	Feature	Integer		PAY_AMT3		no
X21	Feature	Integer		PAY_AMT4		no
X22	Feature	Integer		PAY_AMT5		no
X23	Feature	Integer		PAY_AMT6		no
Y	Target	Binary		default payment next month		no

Figure 1 Description of features

Exploring the data in Python, a decision was made before any experimentation started to apply a Z-score normalization to all features. Figure 2 highlights that most features are not normally distributed so a scaling operation would not have been appropriate.

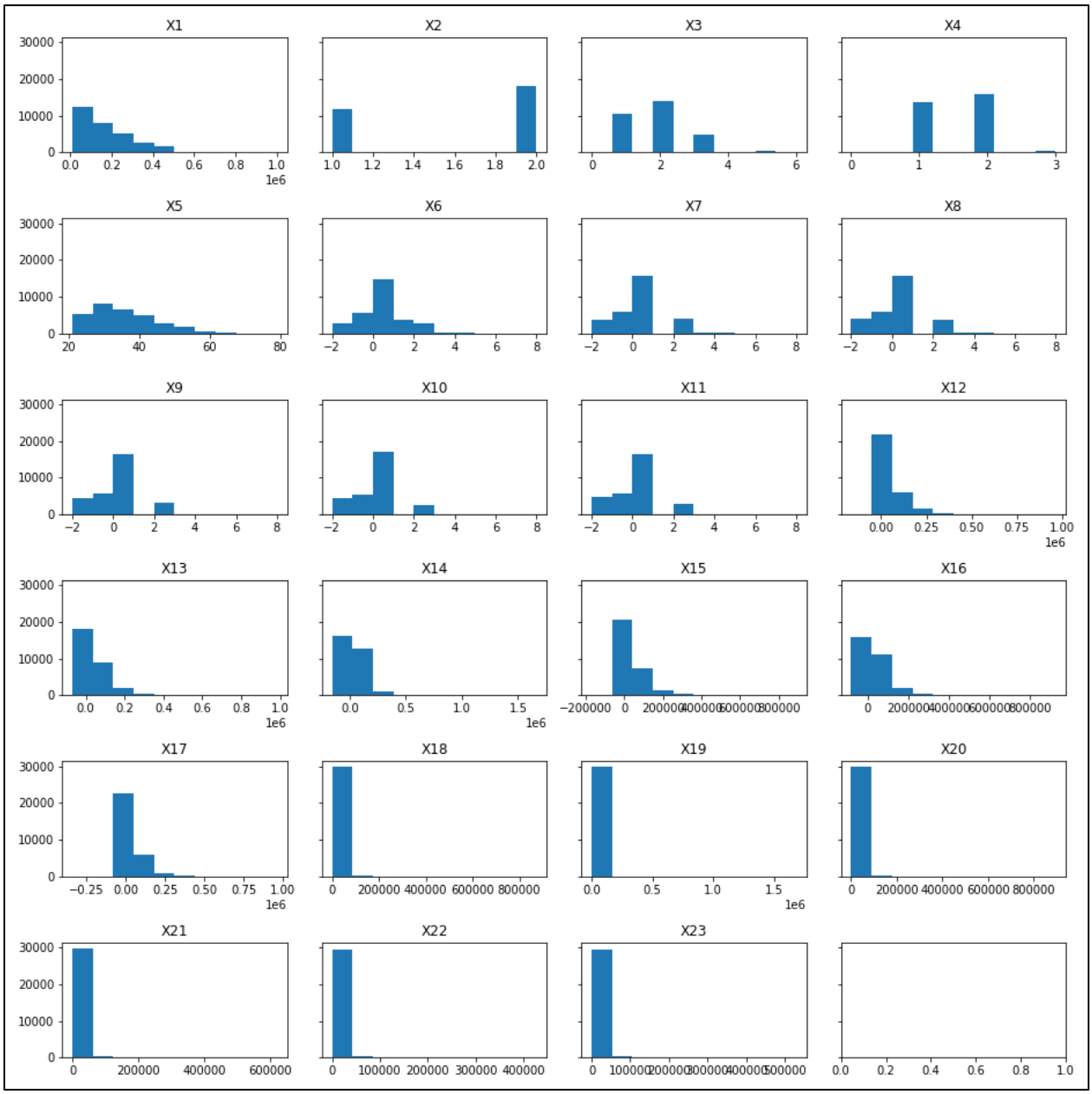


Figure 2 Distribution map of all features

Experimentation

Hyperparameters

Experiment #	Hyperparameter	Thresholds Tested	Objective
1	Learning Rate	0.001 – 0.1	Improved accuracy and loss reduction
2	Number of Epochs	10,000 – 20,000	Loss convergence
3	Data Normalization	None, 0-1 Max, Z-Score	Loss reduction
4	Train Split Size	75%, 90%	Improved accuracy and loss reduction
5	Batch Sizes	16 - 8192	Loss reduction & train/test accuracy tracking
6	Epoch Resizing		Training speed
7	Regularization	L1, L2, Dropout	Train and test accuracy
8	Breadth and Depth	Units: 8, 32, 64, 128 Layers: 1, 2, 3	Improved accuracy and loss reduction
9	Batch Normalization	With and Without	Improved accuracy and loss reduction
10	Output Activation Function	Sigmoid, ReLU	Improved accuracy and loss reduction
11	Hidden Layer Activation Functions	ReLU, ReLU6, Tanh	Improved accuracy and loss reduction
12	Optimizer	SGD, RMSprop, Adam	Improved accuracy and loss reduction
12b	Learning Rate	0.1 - 1	Improved accuracy and loss reduction
13	Momentum	0.000 – 0.999	Improved accuracy and loss reduction

Analysis

For most experiments, the models were evaluated using loss, training accuracy and testing accuracy.

The main objective was to choose hyperparameters that would reduce loss as much as possible to some convergence value, as well as have the testing accuracy match the training accuracy as much as possible.

Experiments

Experiment 1 – Learning Rates

The learning rate was varied from 0.001 through to 0.1. At this point, data was not split into train and test datasets so all 30,000 instances were used for training. No batching was done yet either. The algorithm was trained using 10,000 epochs. Based on figure 3's accuracy plot, a default learning rate of 0.1 was used for subsequent experiments.

The model is initially setup using stochastic gradient descent for its optimizer and binary cross entropy for the loss function.

Judging by the uptick at LR=0.1, there may be evidence to suggest that a higher accuracy exists with an increased learning rate. If redoing this experiment, I would have varied the learning rate from 0.001 through to 1 logarithmically to get a better sense of where the optimal learning rate is for this architecture. However, since the architecture is likely to change, I kept LR=0.1.

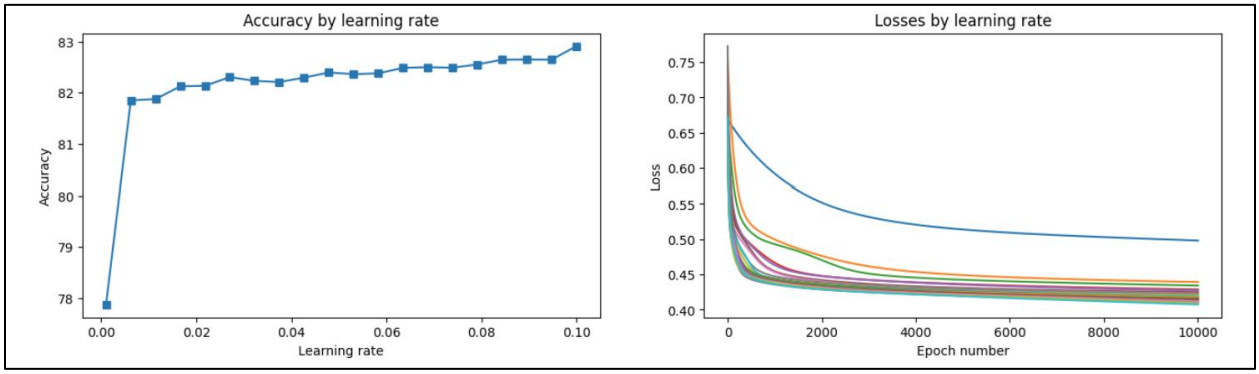


Figure 3 Accuracy and losses plots for learning rates

Experiment 2 – Number of Epochs

The number of epochs was increased to 20,000 to judge convergence. According to figure 4, it looks like there is a linear decline from 0.45 loss to 0.4 without much flattening of the curve, 20,000 epochs was chosen as the default value for subsequent experiments.

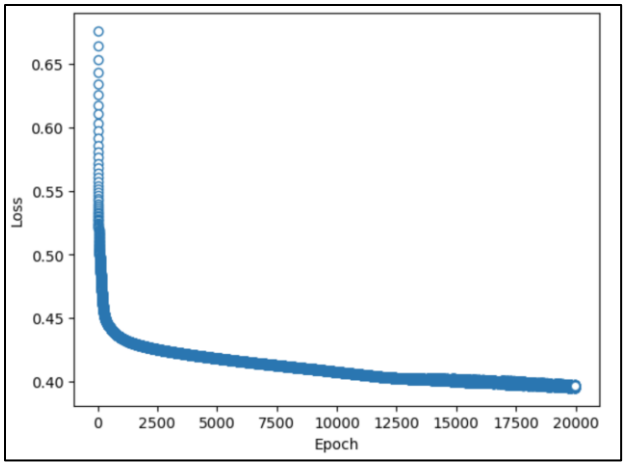


Figure 4 Loss curve of algorithm with LR=0.1

Experiment 3 – Data Normalization

Three experiments were done: no normalization; scaling to 0-1; and Z-score normalization. No normalization showed poor training and high losses so was not considered further. 0-1 scaling had acceptable results, however Z-score normalization showed a good loss curve and convergence to a lower value so was decided to sue for subsequent experiments.

To help speed up subsequent experiments, the epoch value was changed from 20,000 to 1,000. This increases the error in the model by a marginal amount but was seen as acceptable to help decrease experimentation time.

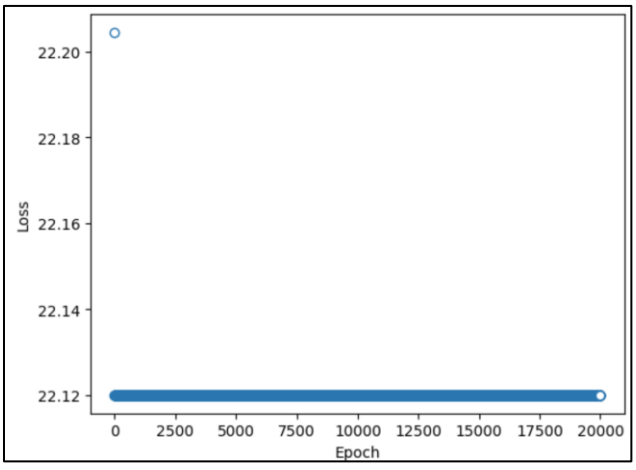


Figure 5 Loss curve where data has not been normalized

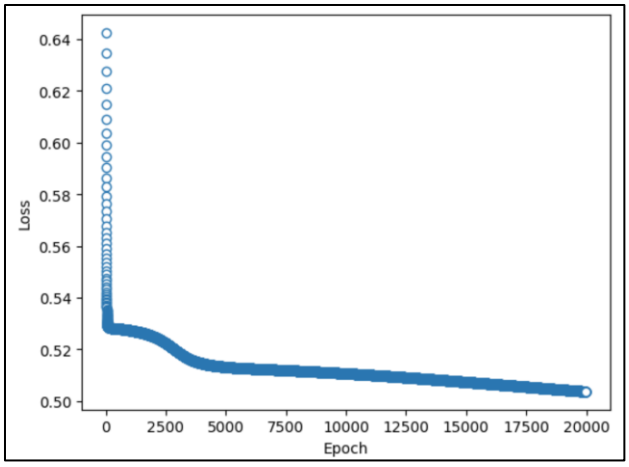


Figure 6 Loss curve where data has been divided by the maximum value (data = X/max(X))

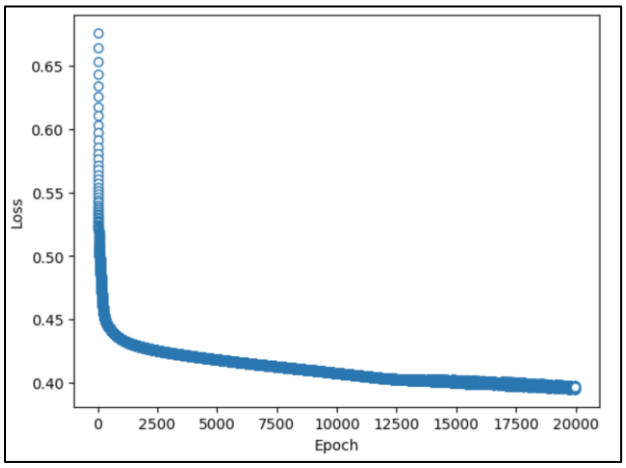


Figure 7 Loss curve of data that has been Z-score normalized

Experiment 4 – Train Split Size

Two experiments were done where the training size was set to 75% and another at 90%. Loss and accuracy were calculated to evaluate.

Both training sizes could be seen as acceptable as they perform relatively similar. A training size of 75% was used for subsequent experiments since the test accuracy tracks training accuracy slightly better.

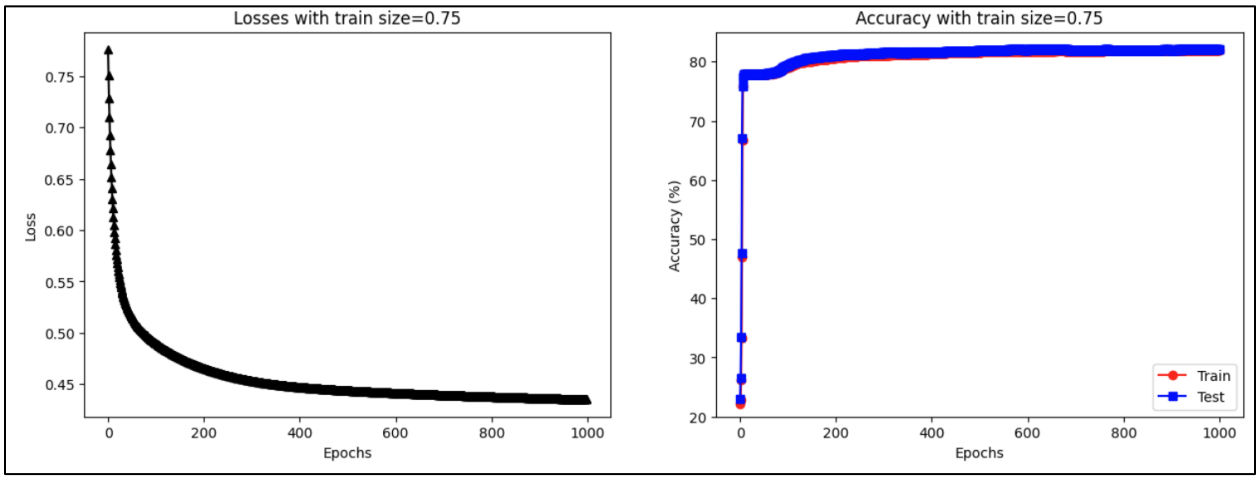


Figure 8 Model performance with training split size of 75%

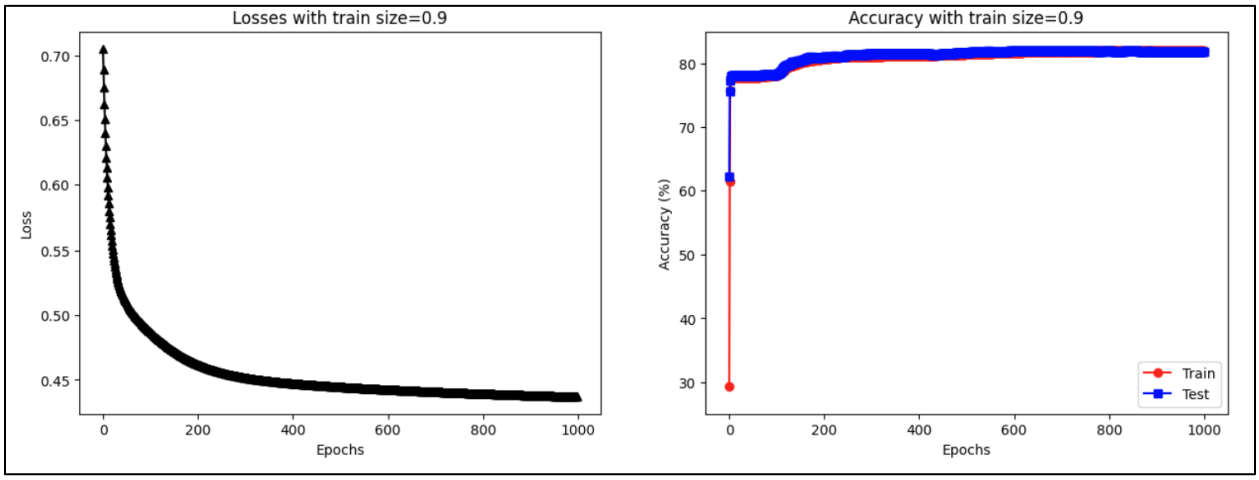


Figure 9 Model performance with training split size of 90%

Experiment 5 – Batch Sizes

Batch sizes from 2^4 to 2^{13} were explored. Figure 10 shows smaller batch sizes have increased training accuracy, but their corresponding test accuracy may not track well. This may be an indicator of fit issues. Large batch sizes have test accuracies that track the training accuracy better. To explore this more deeply, experiments were run on a handful of batch sizes to gain more insight.

Figures 11 to 13 show that, although the training loss is reduced, the testing accuracy does not track well. This may indicate that these batch sizes may not provide a generalized model.

Figures 14 and 15 show larger batch sizes, although not as accurate, will be better generalized. A batch size of 8192 was chosen for subsequent experiments since it's test accuracy does a slightly better job of tracking with training accuracy.

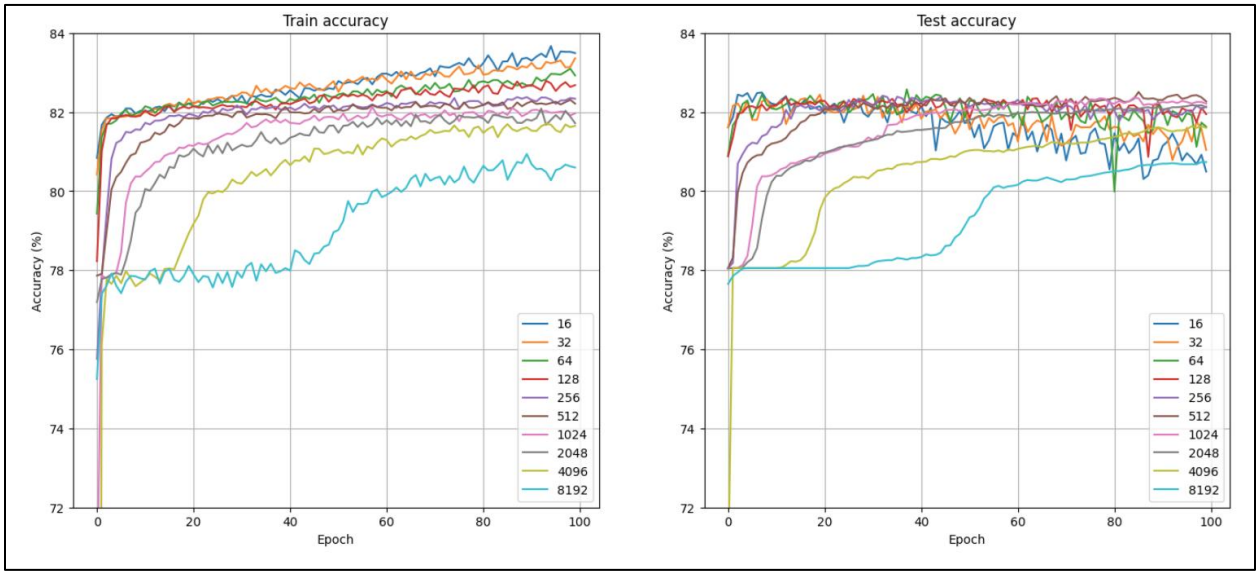


Figure 10 Train and test accuracy for various batch sizes

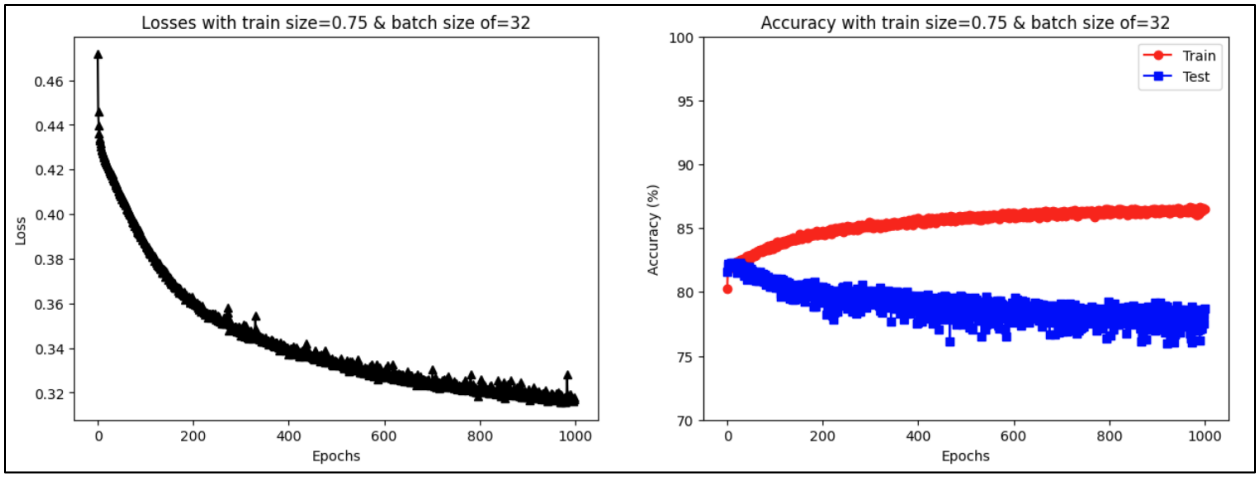


Figure 11 Model performance when batch size = 32

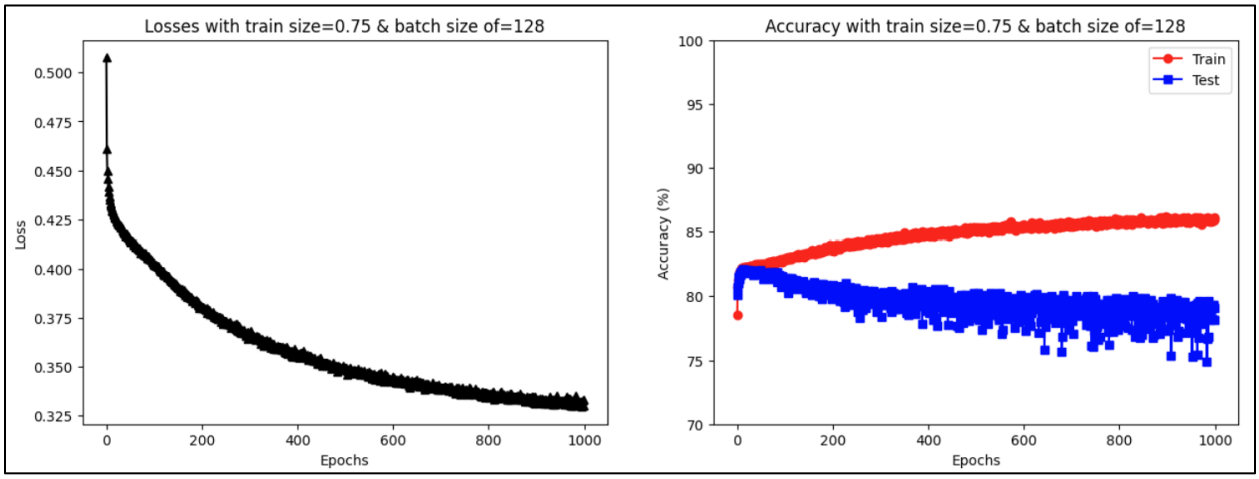


Figure 12 Model performance when batch size = 128

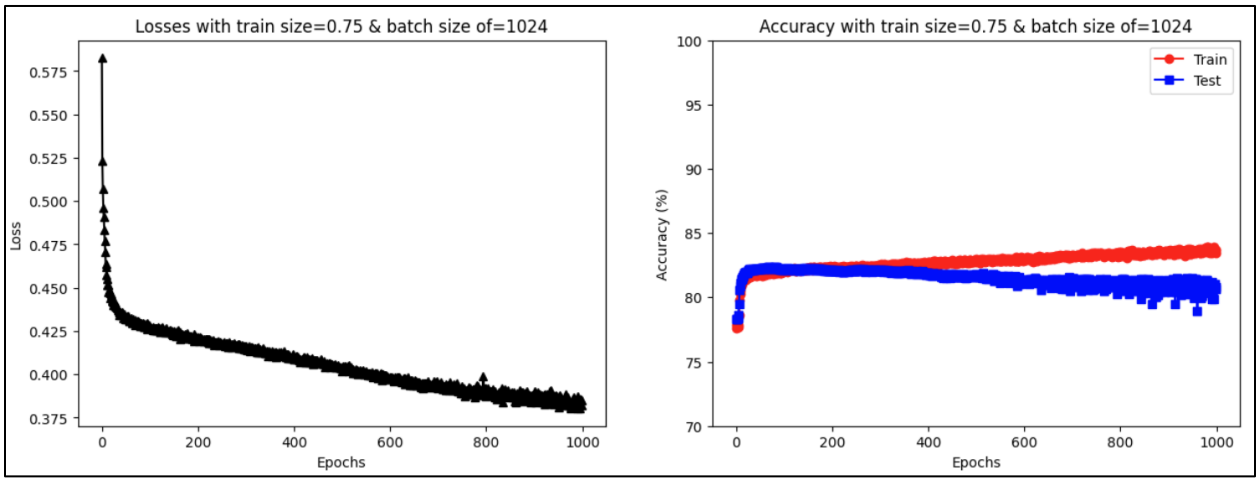


Figure 13 Model performance when batch size = 1024

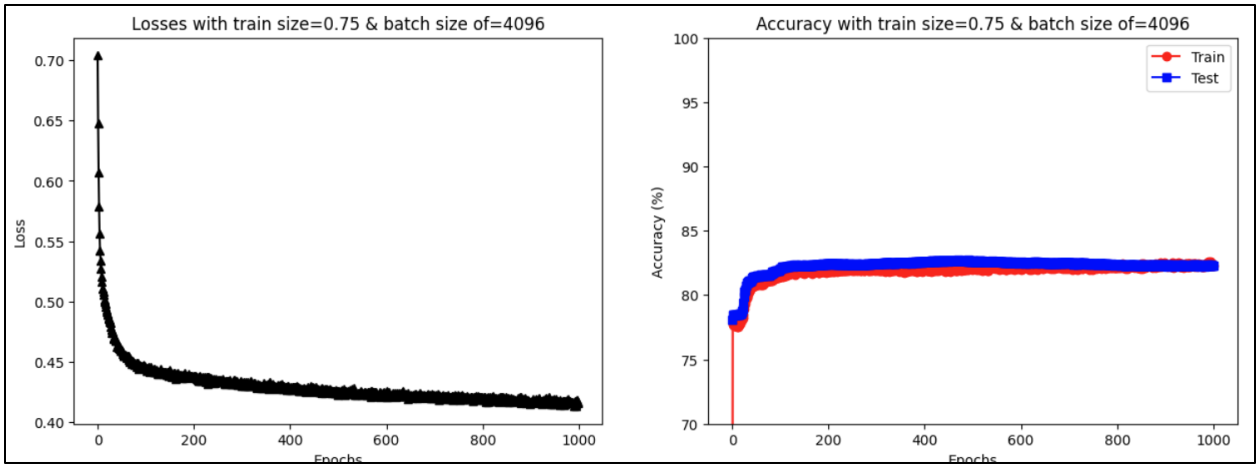


Figure 14 Model performance when batch size = 4096

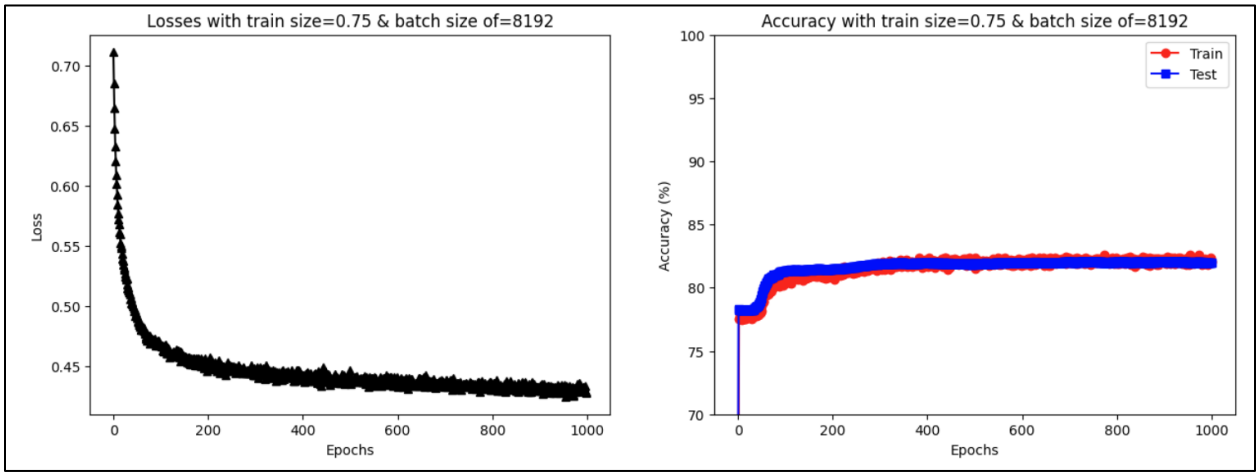


Figure 15 Model performance when batch size = 8192

Experiment 6 – Epoch Resizing

To help speed up experimentation further, especially since batching is now being used, the number of epochs was reduced to 300. From Figure 15, there is marginal improvement in model performance after epoch 300.

Experiment 7 – Regularization

Three regularization techniques were explored: L1, L2, and dropout. Loss and accuracies were used to evaluate the performance of each regularization technique.

L1 regularization was done using a range from 0 to 0.01. This technique did not show improvement in model accuracy sow as not selected.

L2 regularization was done using a range from 0 to 0.1. This technique showed a decrease in model performance sow as not selected.

Dropout was done using drop out rates from 0 to 0.9. Generally, this technique decreased model performance. Additional investigation was done using a dropout rate of 0.1 since this rate showed the closest train to test accuracy. Ultimately, no dropout was used since the model performance was either unchanged or decreased.

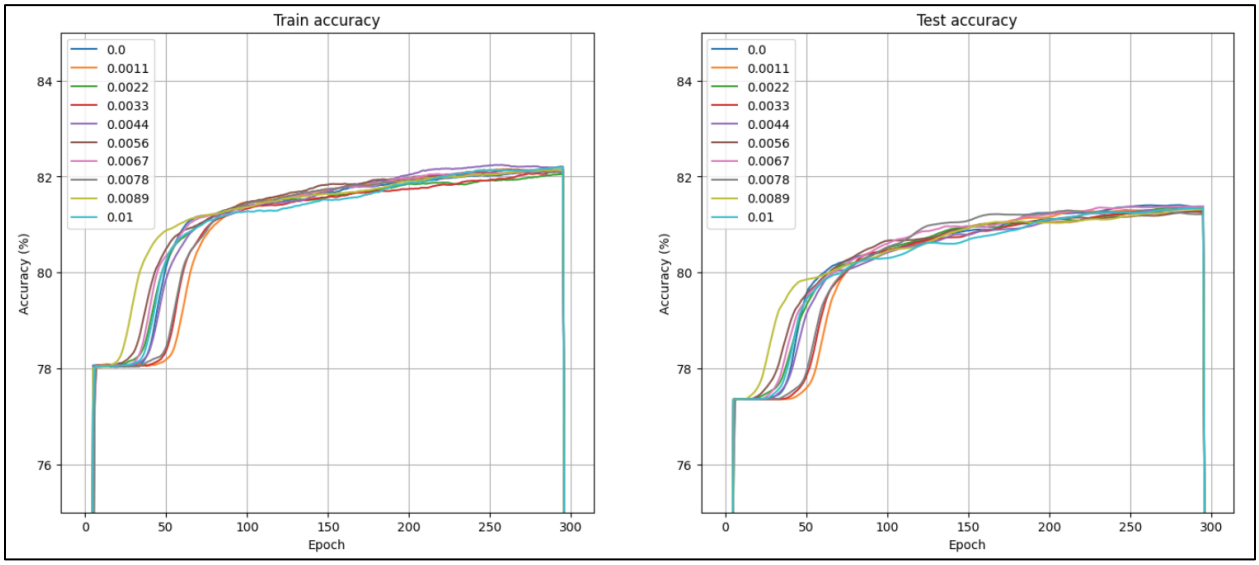


Figure 16 Model performance with L1 regularization

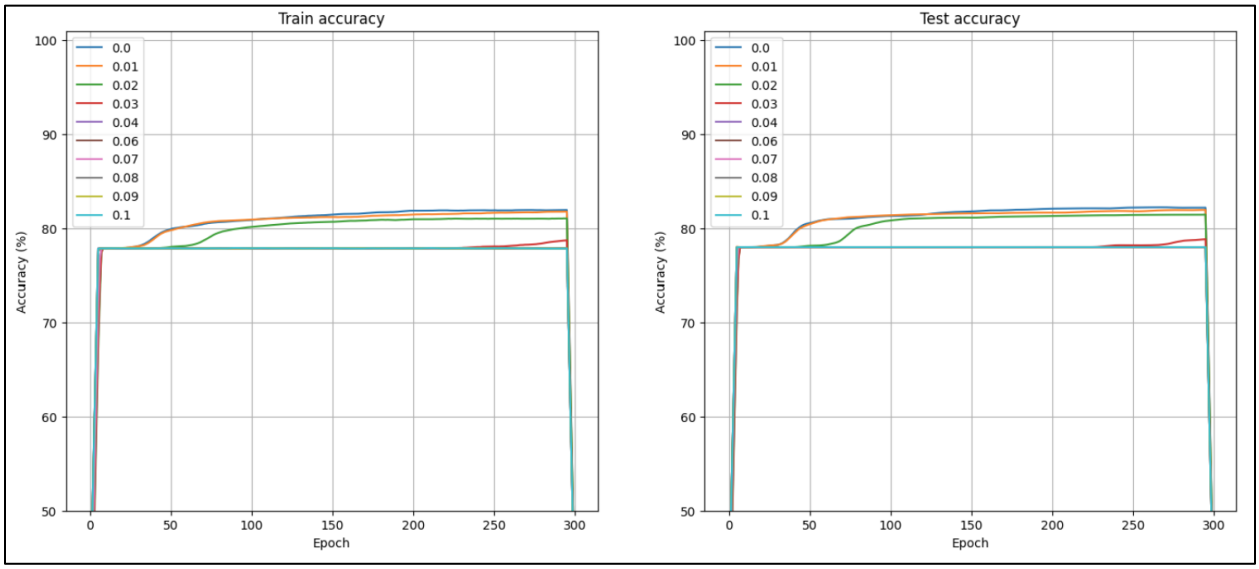


Figure 17 Model performance with L2 regularization

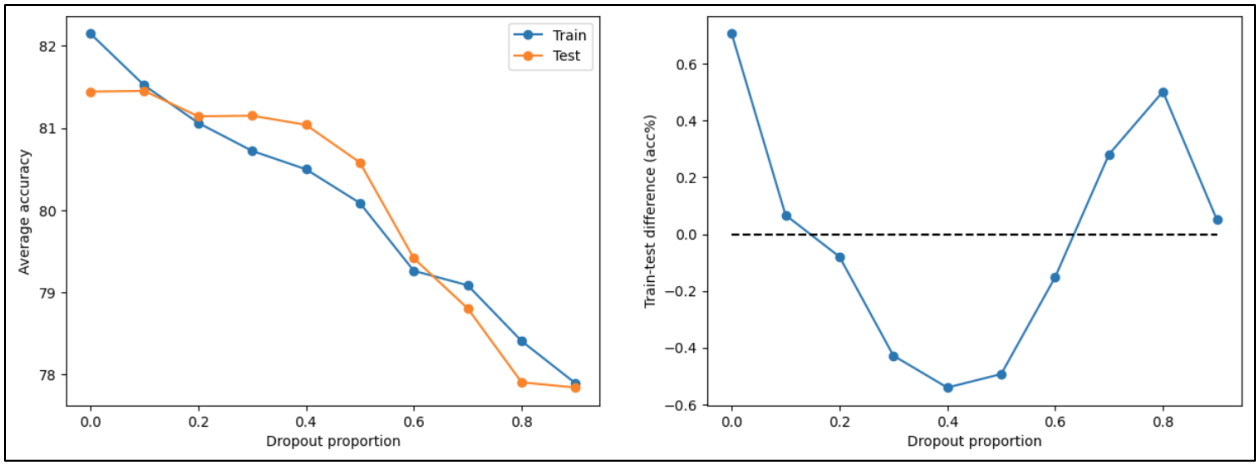


Figure 18 Model performance with dropout enabled

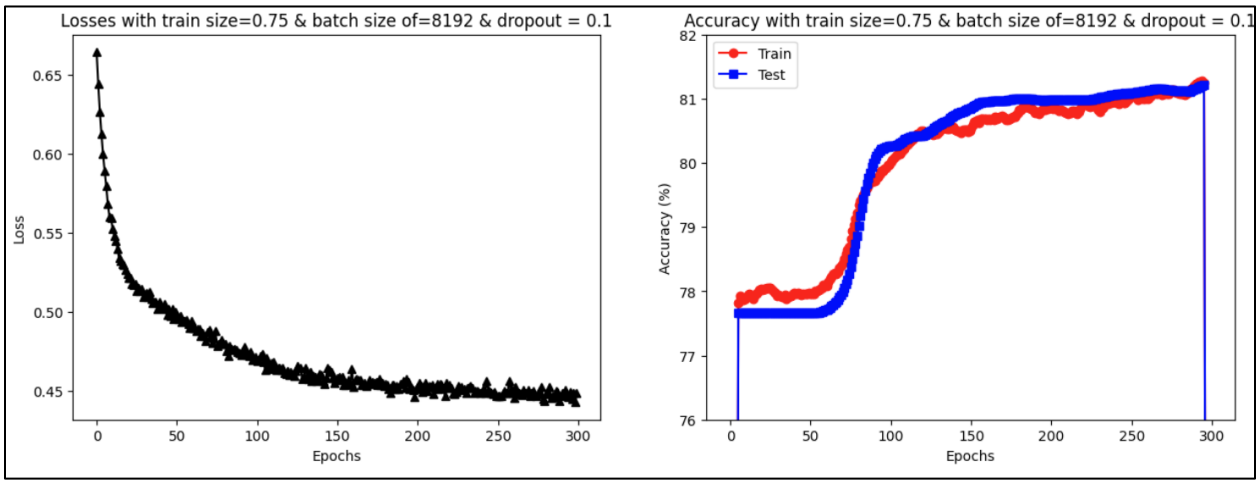


Figure 19 Model performance details when dropout rate = 0.1

Experiment 8 – Breadth and Depth

Fifteen experiments were completed where the number of units in each hidden layer were changed and the number of hidden layers was increased. I experimented with units per layer of 8, 32, 64 and 128 and the number of hidden layers from 1, 2, and 3.

An example architecture of the first experiment can be seen below.

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                [-1, 1, 8]             192
              ReLU-2                [-1, 1, 8]               0
            Linear-3                [-1, 1, 8]              72
              ReLU-4                [-1, 1, 8]               0
            Linear-5                [-1, 1, 1]               9
           Sigmoid-6                [-1, 1, 1]               0
================================================================
Total params: 273
Trainable params: 273
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.00
Params size (MB): 0.00
Estimated Total Size (MB): 0.00
----------------------------------------------------------------
```

An example architecture of the last experiment can be seen below:

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1               [-1, 1, 128]           3,072
              ReLU-2               [-1, 1, 128]               0
            Linear-3               [-1, 1, 128]          16,512
              ReLU-4               [-1, 1, 128]               0
            Linear-5               [-1, 1, 128]          16,512
              ReLU-6               [-1, 1, 128]               0
            Linear-7               [-1, 1, 128]          16,512
              ReLU-8               [-1, 1, 128]               0
            Linear-9                 [-1, 1, 1]             129
          Sigmoid-10                 [-1, 1, 1]               0
================================================================
Total params: 52,737
Trainable params: 52,737
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.01
Params size (MB): 0.20
Estimated Total Size (MB): 0.21
----------------------------------------------------------------
```

When the number of units per layer was set to 8, figure 20 generally showed higher error than normal. A layer size of 2 may have been acceptable as the loss is minimized.

When the number of units per layer was set to 32, figure 21 showed good loss and train vs test accuracy tracking up until the number of layers increased to 3.

When the number of units per layer was set to 64, the figure 22 showed good loss curves. The train vs test accuracy tracking when the number of layers was 3 showed the best generalized model. This was decided to be the model architecture for subsequent experiments.

When the number of units per layer was set to 128, the figure 23 showed poorer train vs test accuracy tracking. This indicates that the model may be too complex and does not generalize as well.

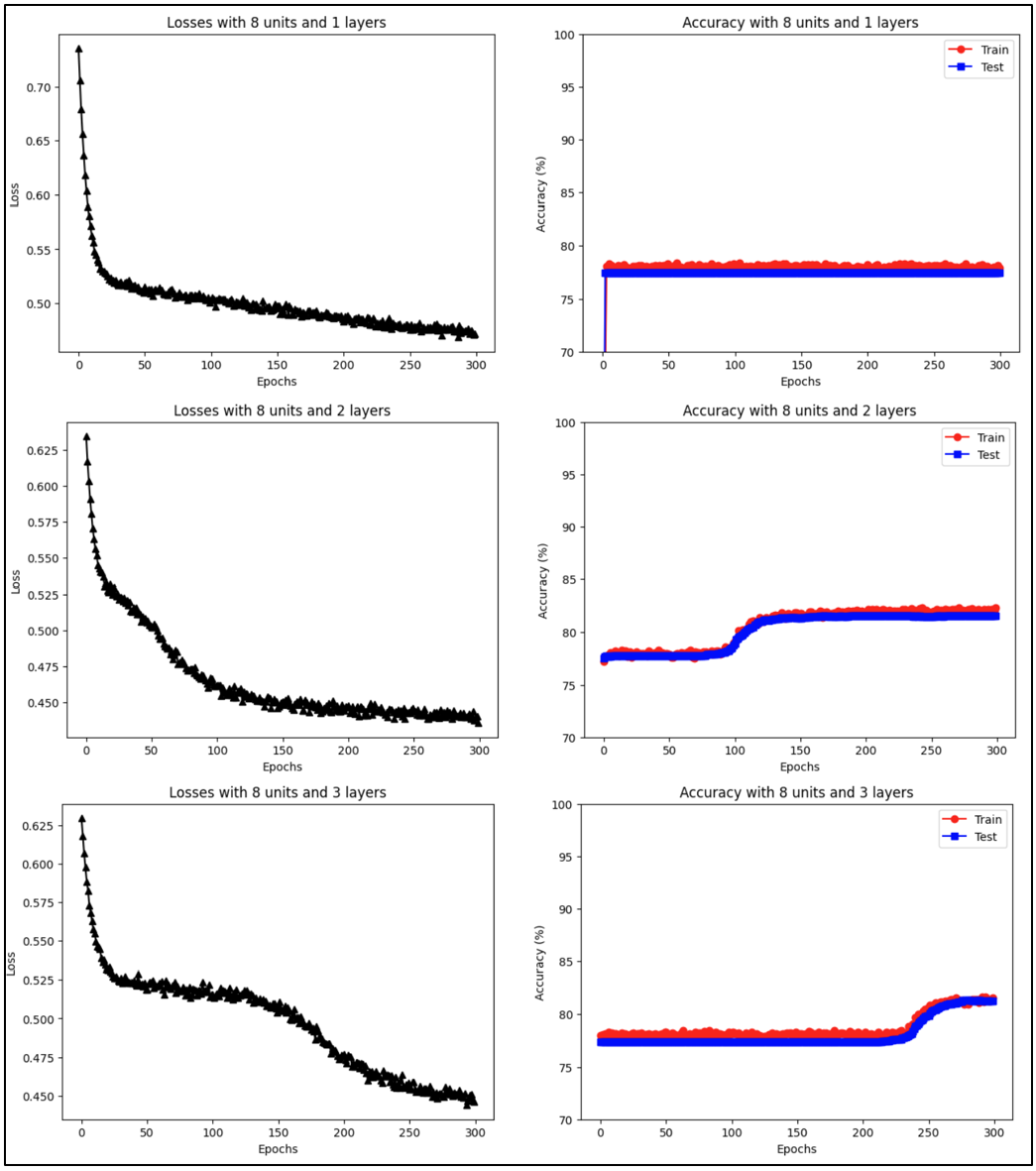


Figure 20 Experiments where number of units per layer were 8 and layers increased

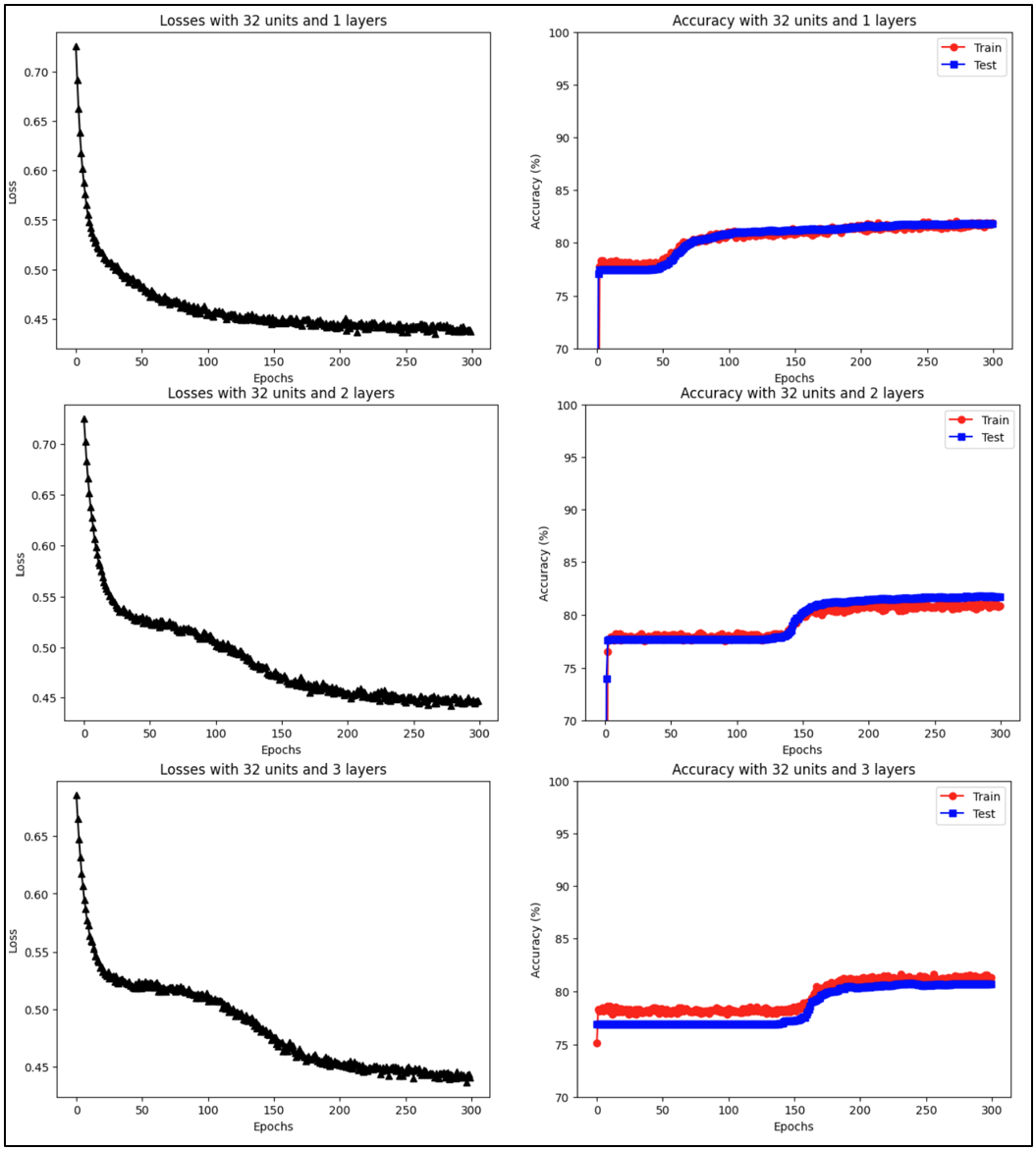


Figure 21 Experiments where number of units per layer were 32 and layers increased

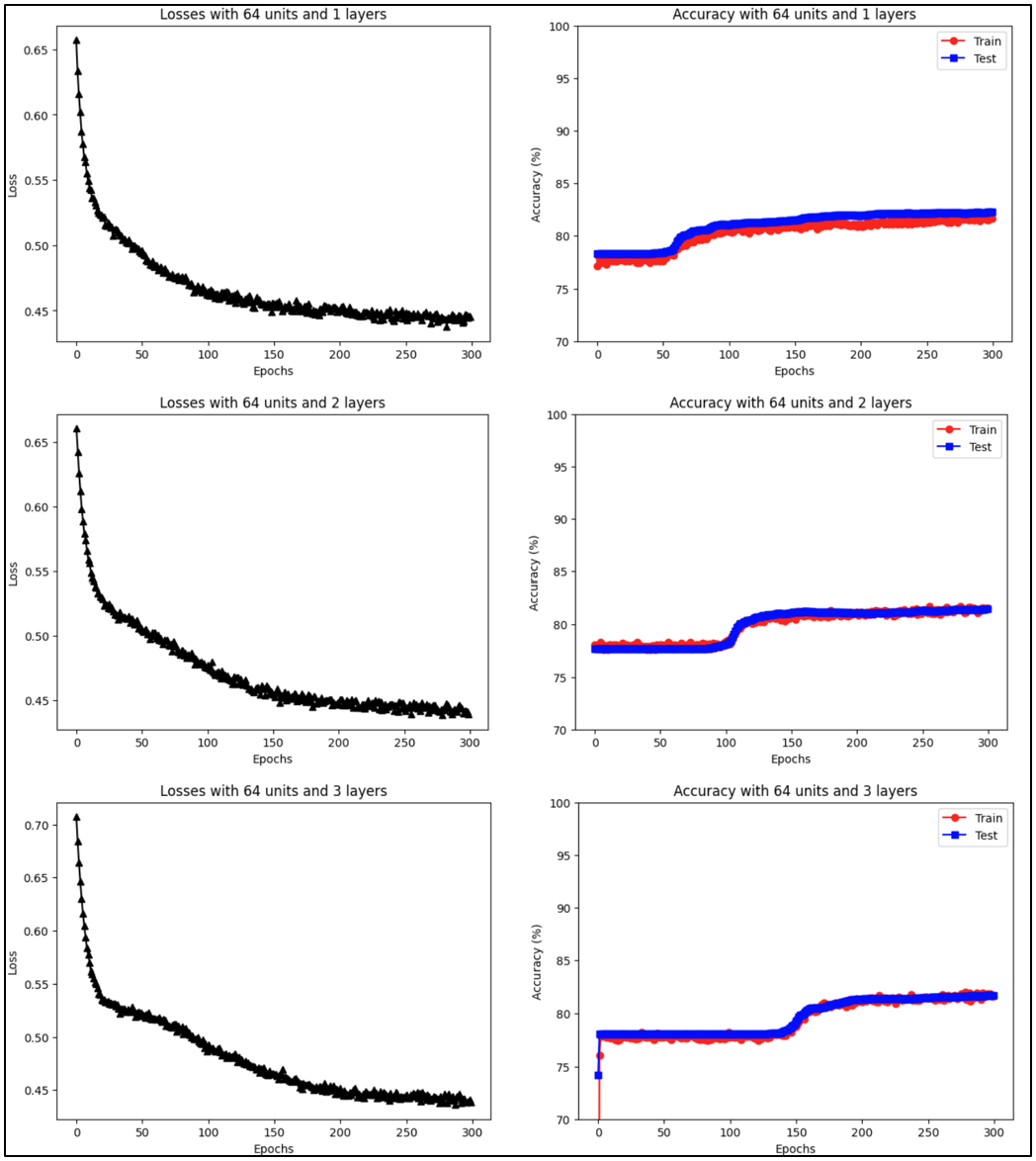


Figure 22 Experiments where number of units per layer were 64 and layers increased

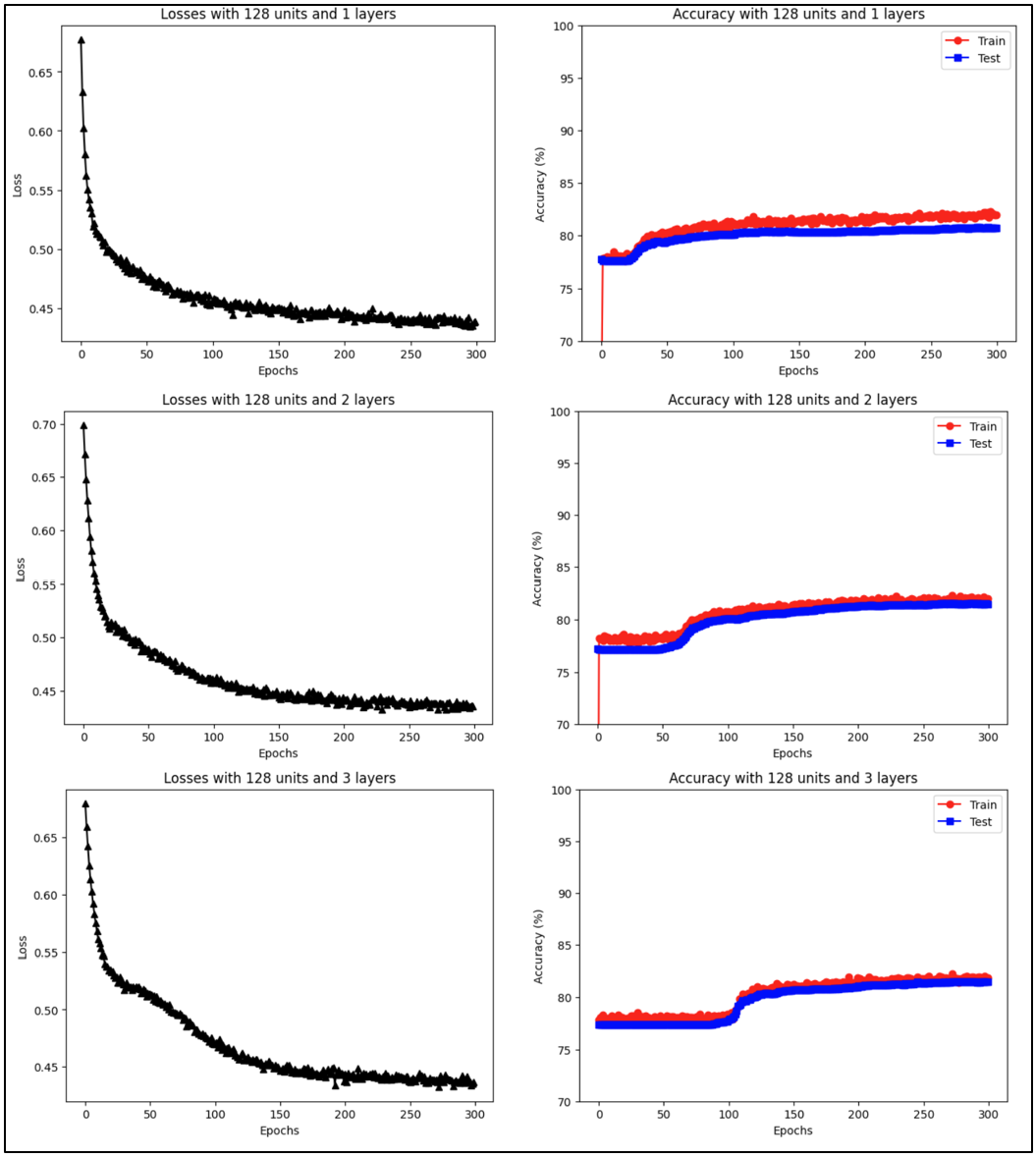


Figure 23 Experiments where number of units per layer were 128 and layers increased

Experiment 9 – Batch Normalization

The new model architecture for this experiment is shown below:

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                [-1, 1, 64]           1,536
              ReLU-2                [-1, 1, 64]               0
            Linear-3                [-1, 1, 64]           4,160
              ReLU-4                [-1, 1, 64]               0
            Linear-5                [-1, 1, 64]           4,160
              ReLU-6                [-1, 1, 64]               0
            Linear-7                [-1, 1, 64]           4,160
              ReLU-8                [-1, 1, 64]               0
            Linear-9                 [-1, 1, 1]              65
          Sigmoid-10                 [-1, 1, 1]               0
================================================================
Total params: 14,081
Trainable params: 14,081
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.00
Params size (MB): 0.05
Estimated Total Size (MB): 0.06
----------------------------------------------------------------
```

The experiment was run with and without batch normalization at all the input and hidden layers. Although the loss is decreased with batch normalization, the model begins to show degradation in generalization after epoch 100. Figure 26 shows that, with batch normalization, the model does not behave in a stable manner.

Due to this, it was decided not to run with subsequent experiments with batch normalization.

Based on figure 24, the number of epochs was further reduced to 200 from 300 to speed up experimentation. The loss curve and accuracy curves show stability and convergence after epoch 200.

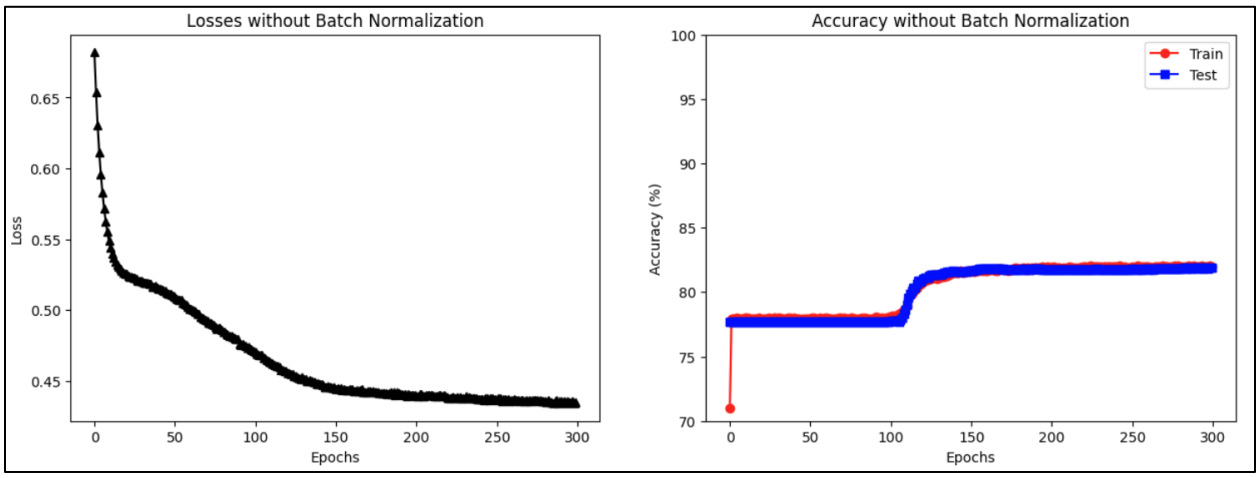


Figure 24 Model performance without batch normalization

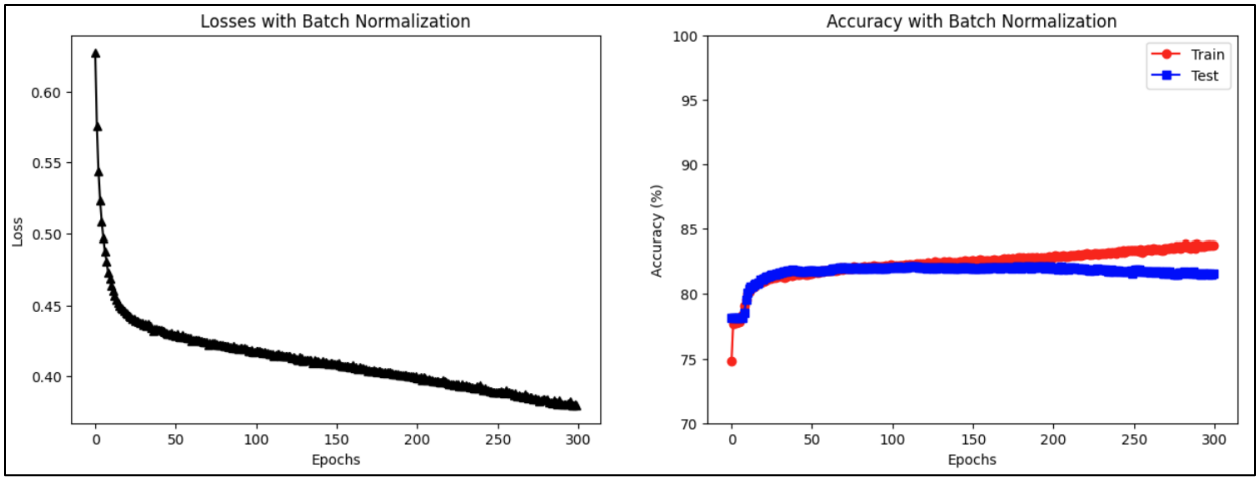


Figure 25 Model performance with batch normalization at all input and hidden layers

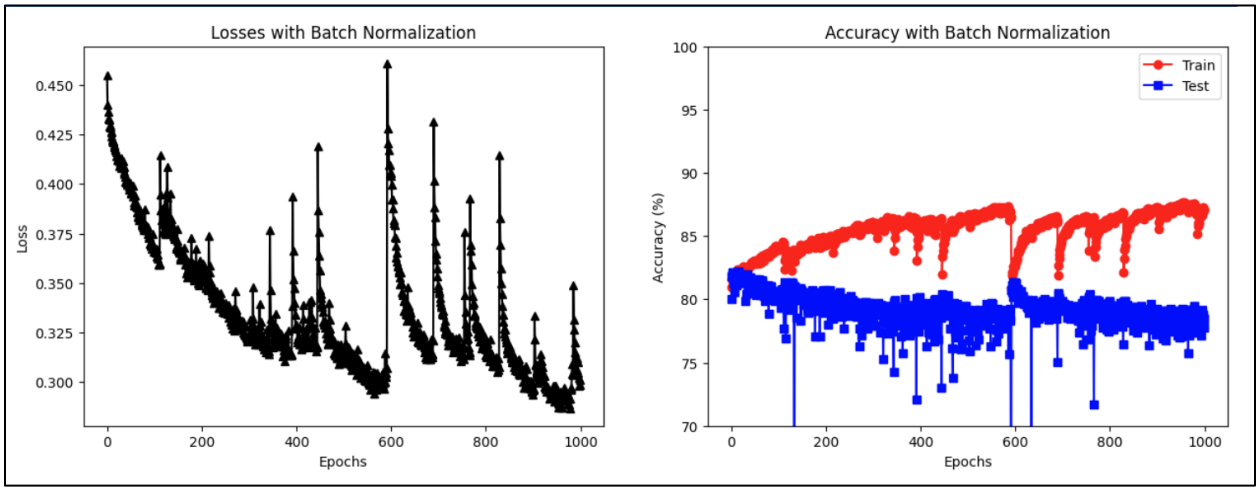


Figure 26 Model performance with batch normalization but with increased epochs

Experiment 10 – Output Activation Function

In this experiment, the output layer's activation function was experimented between Sigmoid and ReLU.

Figure 27 shows normal performance since previous experiments used Sigmoid.

Figure 28 shows unstable behavior in the loss curve so was not used for further experiments.

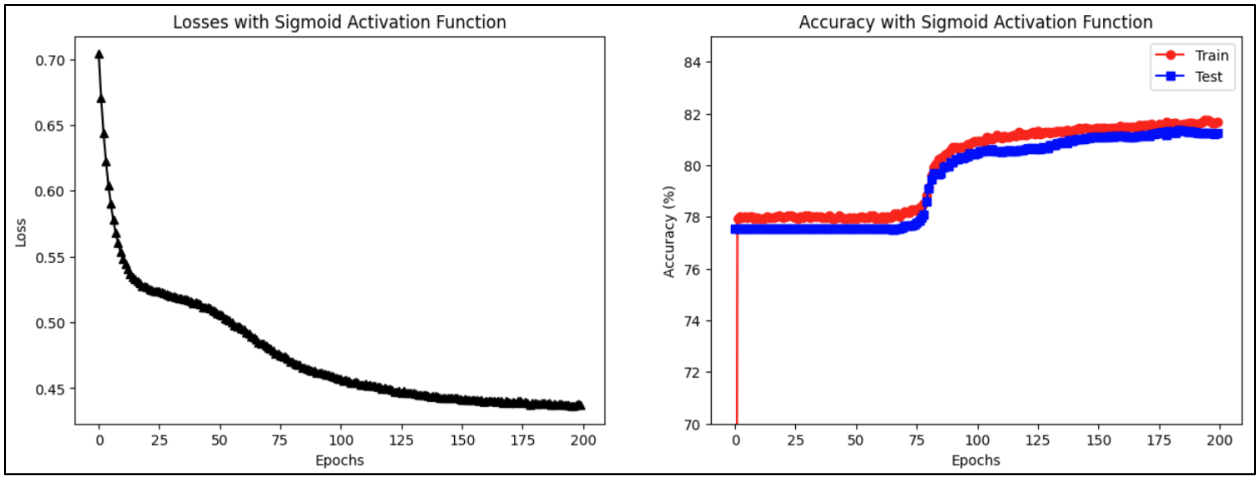


Figure 27 Model performance with Sigmoid in the output layer

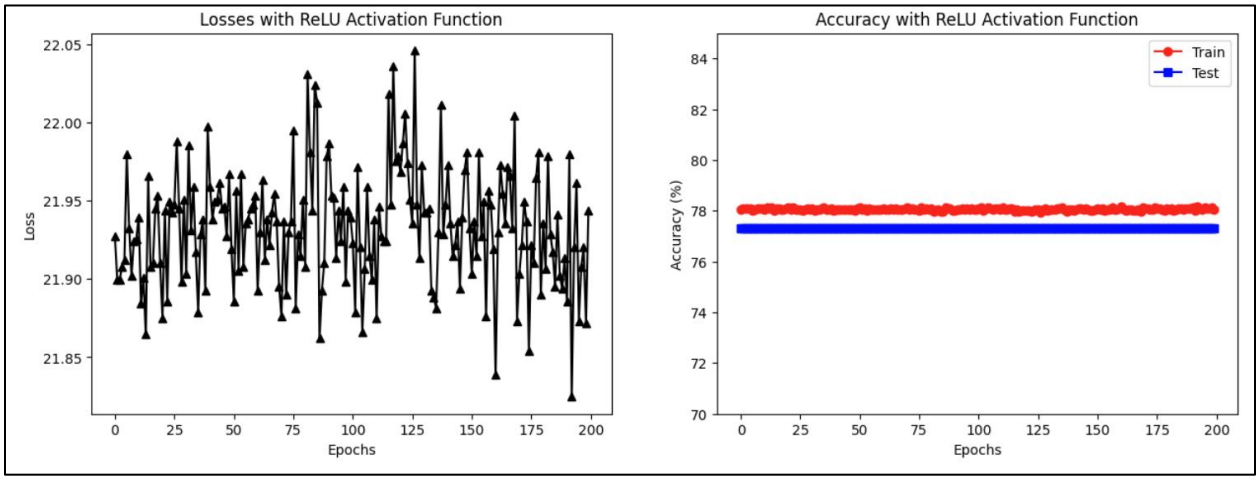


Figure 28 Model performance with ReLU in the output layer

Experiment 11 – Hidden Layer Activation Functions

In this experiment, the activation functions in all input and hidden layers were changed to be either ReLU (existing in all experiments), ReLU6, or Tanh.

ReLU has been used as the activation function up to the point. Figure 30 shows ReLU6 as creating an overfitted model since the test accuracy is more than the training accuracy. Figure 31 shows Tanh as having a good fit.

For subsequent experiments, Tanh was chosen although ReLU would have been just as acceptable. Tanh showed slightly better loss behaviour compared to ReLU.

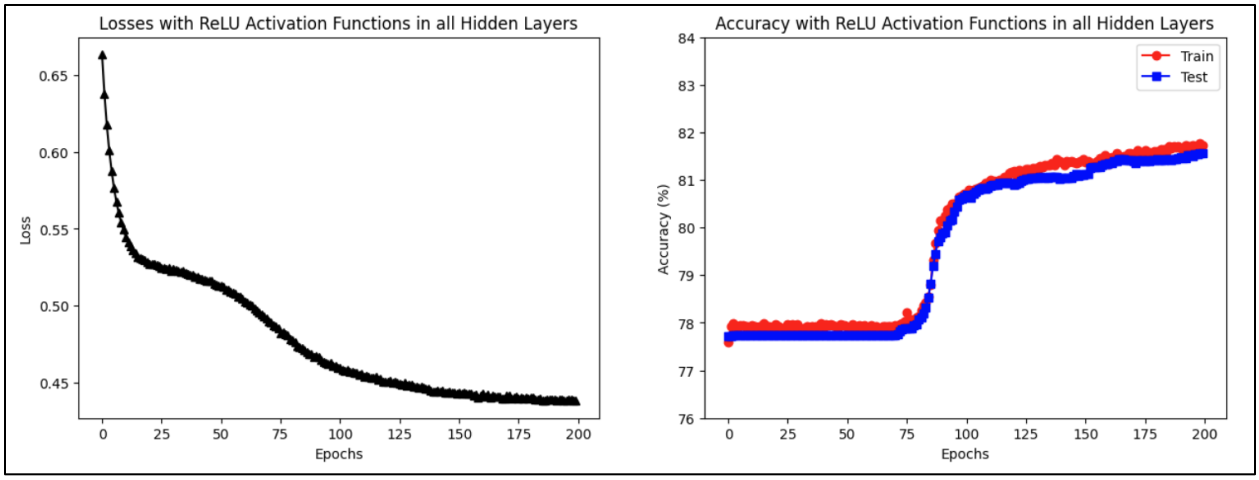


Figure 29 Model performance with ReLU in the input and hidden layers

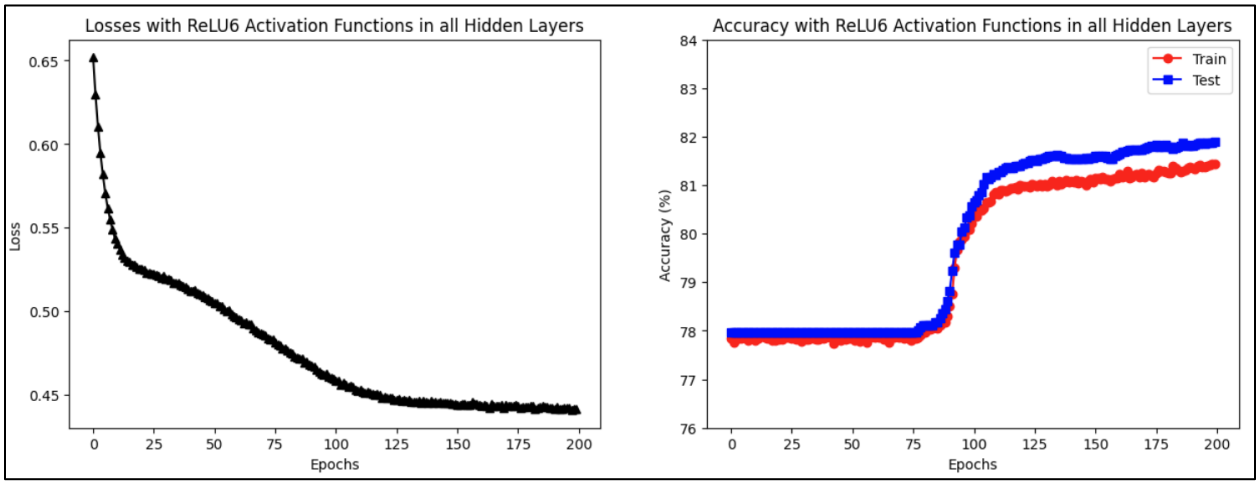


Figure 30 Model performance with ReLU6 in the input and hidden layers

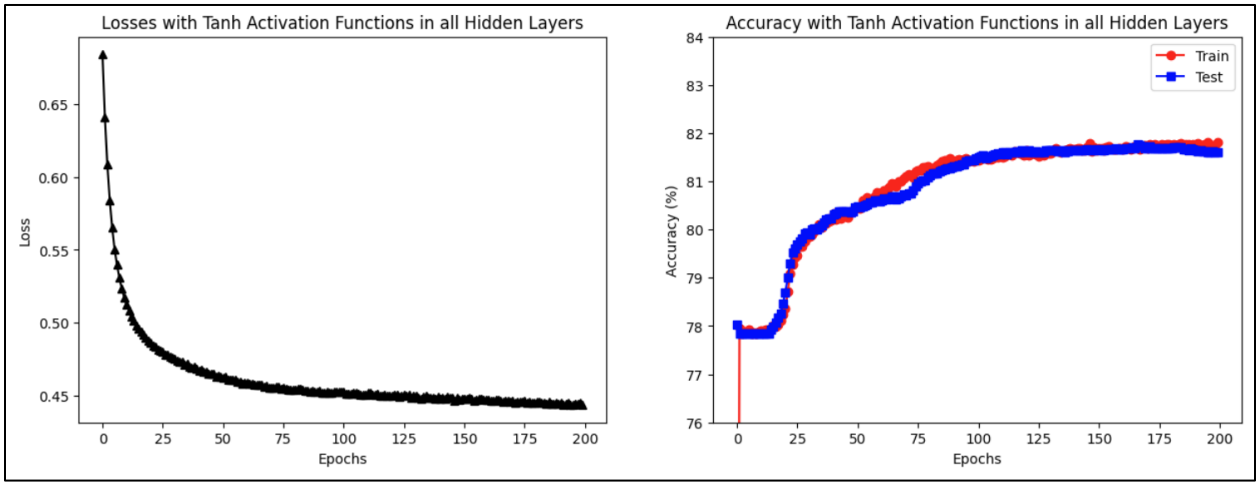


Figure 31 Model performance with Tanh in the input and hidden layers

Experiment 12 – Optimizers

In this experiment, three optimizers were experimented with: (1) stochastic gradient descent (SGD); (2) RMSprop; and (3) Adam.

Figure 32 shows that the SGD optimizer provided the most generalized model. The Adam optimizer may also be a good choice but the generalization behaviour is not as good as SGD. RMSprop does not provide a stable model so was not considered for further investigation.

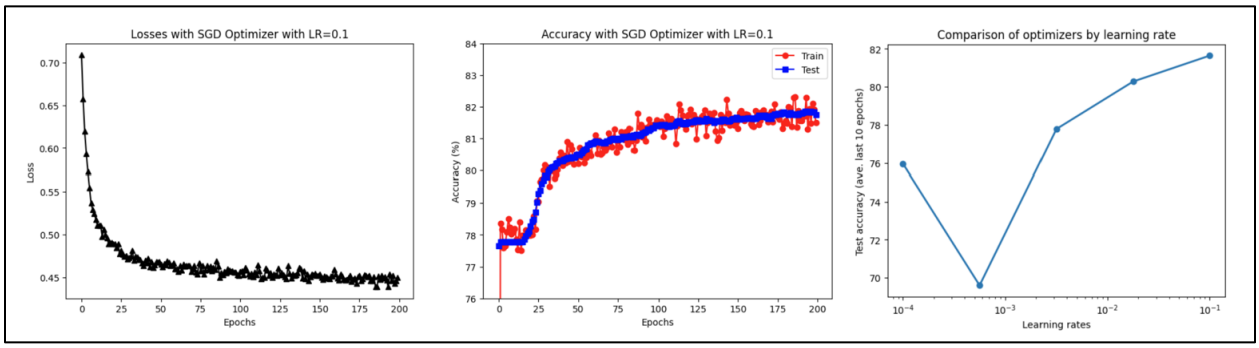


Figure 32 Model performance of SGD optimizer when LR=0.1 and test accuracies at various LRs

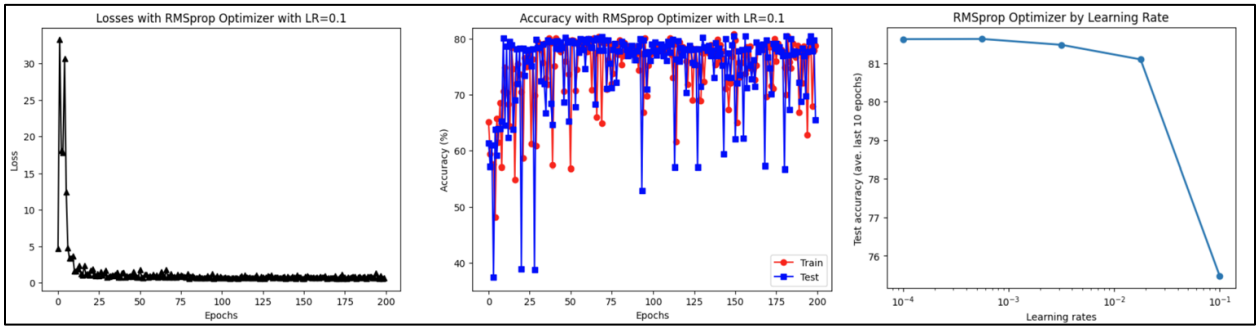


Figure 33 Model performance of RMSprop optimizer when LR=0.1 and test accuracies at various LRs

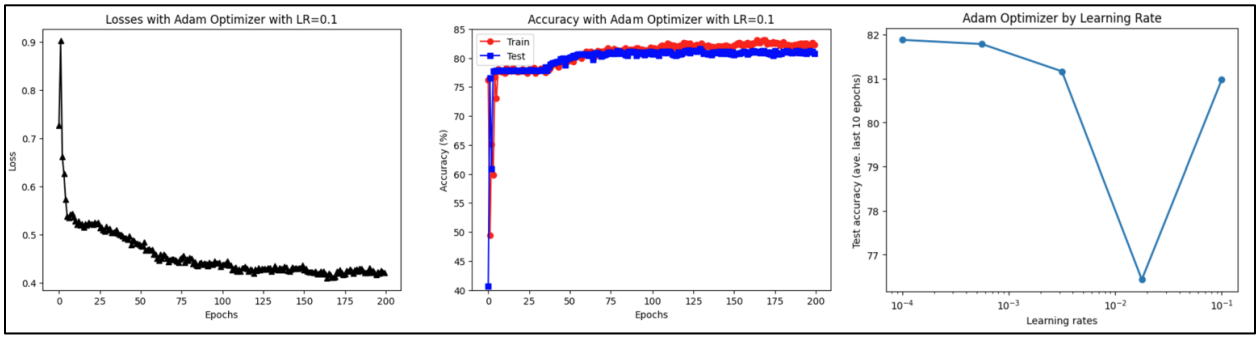


Figure 34 Model performance of Adam optimizer when LR=0.1 and test accuracies at various LRs

Experiment 13 – Learning Rate Again

In this experiment, learning rate was re-explored since SGD was shown to provide the best outcomes. Learning rates between 0.1 through to 1 were explored. Figure 35 showed that there was still opportunity to increase the learning rate. Figure 36 showed the optimal learning rate was at a value of 0.70710678. This value was set as the default for subsequent experiments.

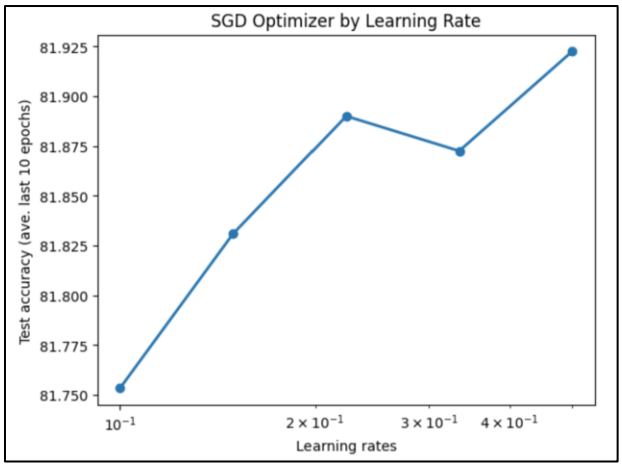


Figure 35 Test accuracies when LR ranges from 0.1 through to 0.5

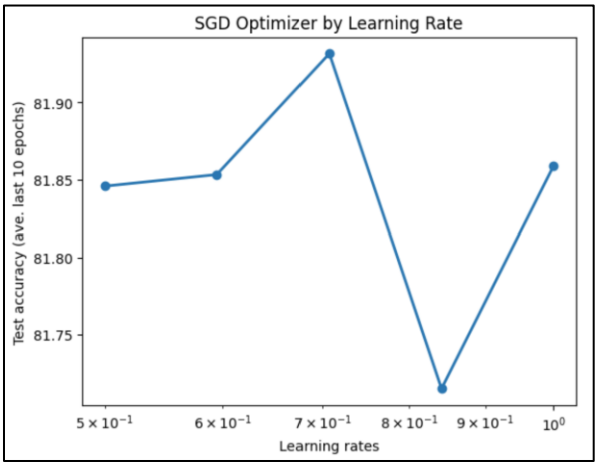


Figure 36 Test accuracies when LR ranges from 0.5 through to 1

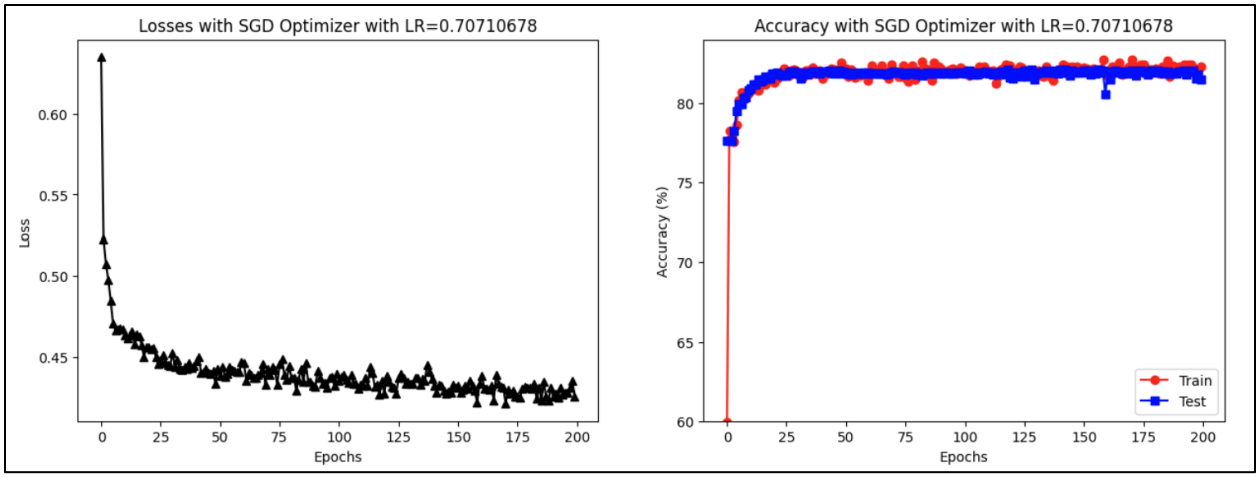


Figure 37 Model performance

Experiment 14 – Momentum

In this experiment, momentum was added to the optimizer. Momentum values from 0 through to 0.999 were explored.

Although figure 38 may show that momentum values up to 0.75 may perform similarly, at larger moment values the test accuracy starts to deviate from the training accuracy. Lower momentum values were further explored in figure 39. Comparing figure 37 to 39 shows that, although accuracy is not significantly increased, the test accuracy tracks training accuracy slightly better.

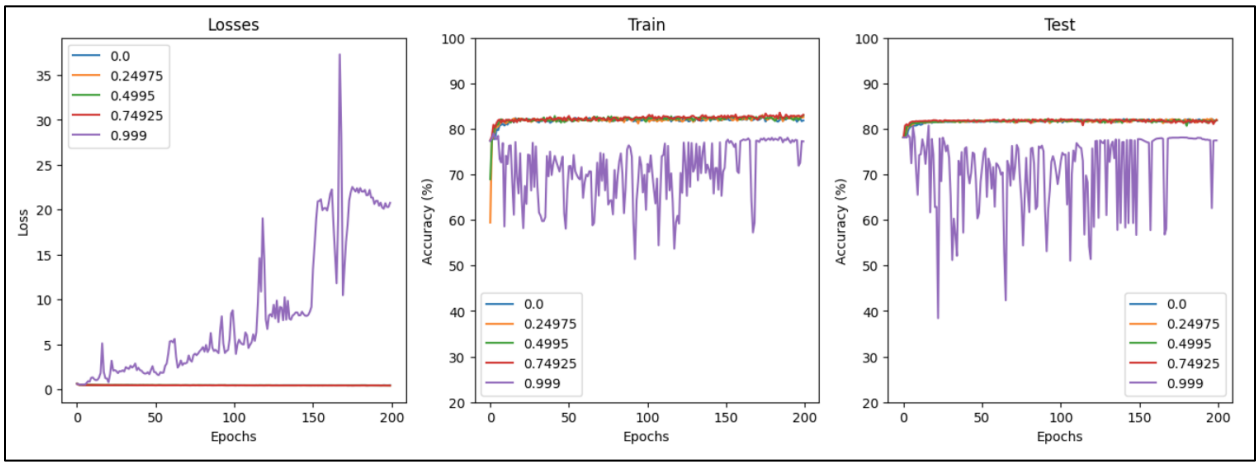


Figure 38 Model performance of various momentum values

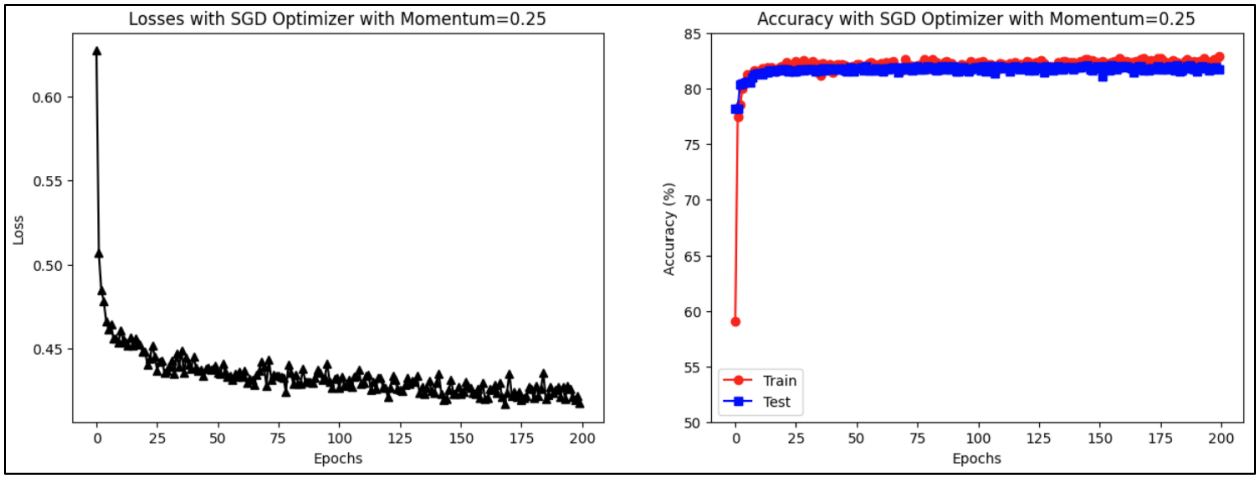


Figure 39 Model performance when momentum = 0.25

Conclusion

The experimentation created a model that was complex enough o be more generalized, faster to train, and slightly more accurate.

On the topic of accuracy, experimentation did not increase accuracy significantly as originally expected. Starting accuracies may have been around 80-83% and the model's accuracy at the end was at a stable ~82%. Most experiments provided marginal accuracy benefits.

The most beneficial experiments that helped accuracy and training time were data normalization, increasing complexity through breadth and depth, and further optimizing the learning rate.

Challenges

There were technical challenges like properly calculating accuracy and reducing the amount of time spent training so experimentation can be done quicker.

However, the biggest challenge was understanding if a model is good or not based on a loss curve and accuracy curves.

I chose to evaluate models by judging how well the test accuracy curve followed the training accuracy curve. I did this since the test dataset was especially large at 7,500 instances so there was strong probability that the model was being tested with unseen and unique data.

If a model's testing accuracy did not follow the training accuracy well I judged this as not being generalized well or by having over/underfitting issues.

Future Work & Improvements

The number of features in the dataset (23) seems high and most are unlikely to add value. I would go through a round of feature selection using principal components analysis or Chi-squared analysis to understand which features either may be summarized into a new feature (i.e. binned) or to be removed altogether.

Another improvement would be to calculate more performance metrics, like confusion matrix, ROC and AUC, precision and recall, etc. to better describe the performance of the ANN model after experiments. For example, although the accuracy may not change, perhaps the type of error you may want to minimize changes significantly.

The last part of the project would be to compare the custom ANN model created to popular machine learning algorithms. This may be time consuming as one could also go through a set of experiments to optimize ML algorithms. However, this would show if the complexity of an ANN model is appropriate for the problem at hand. Perhaps a K-nearest neighbours model would have provided a more accurate predictions.

Appendix

	count	mean	std	min	25%	50%	75%	max
X1	30000.0	167484.322667	129747.661567	10000.0	50000.00	140000.0	240000.00	1000000.0
X2	30000.0	1.603733	0.489129	1.0	1.00	2.0	2.00	2.0
X3	30000.0	1.853133	0.790349	0.0	1.00	2.0	2.00	6.0
X4	30000.0	1.551867	0.521970	0.0	1.00	2.0	2.00	3.0
X5	30000.0	35.485500	9.217904	21.0	28.00	34.0	41.00	79.0
X6	30000.0	-0.016700	1.123802	-2.0	-1.00	0.0	0.00	8.0
X7	30000.0	-0.133767	1.197186	-2.0	-1.00	0.0	0.00	8.0
X8	30000.0	-0.166200	1.196868	-2.0	-1.00	0.0	0.00	8.0
X9	30000.0	-0.220667	1.169139	-2.0	-1.00	0.0	0.00	8.0
X10	30000.0	-0.266200	1.133187	-2.0	-1.00	0.0	0.00	8.0
X11	30000.0	-0.291100	1.149988	-2.0	-1.00	0.0	0.00	8.0
X12	30000.0	51223.330900	73635.860576	-165580.0	3558.75	22381.5	67091.00	964511.0
X13	30000.0	49179.075167	71173.768783	-69777.0	2984.75	21200.0	64006.25	983931.0
X14	30000.0	47013.154800	69349.387427	-157264.0	2666.25	20088.5	60164.75	1664089.0
X15	30000.0	43262.948967	64332.856134	-170000.0	2326.75	19052.0	54506.00	891586.0
X16	30000.0	40311.400967	60797.155770	-81334.0	1763.00	18104.5	50190.50	927171.0
X17	30000.0	38871.760400	59554.107537	-339603.0	1256.00	17071.0	49198.25	961664.0
X18	30000.0	5663.580500	16563.280354	0.0	1000.00	2100.0	5006.00	873552.0
X19	30000.0	5921.163500	23040.870402	0.0	833.00	2009.0	5000.00	1684259.0
X20	30000.0	5225.681500	17606.961470	0.0	390.00	1800.0	4505.00	896040.0
X21	30000.0	4826.076867	15666.159744	0.0	296.00	1500.0	4013.25	621000.0
X22	30000.0	4799.387633	15278.305679	0.0	252.50	1500.0	4031.50	426529.0
X23	30000.0	5215.502567	17777.465775	0.0	117.75	1500.0	4000.00	528666.0

Figure 40 Statistical description of feature data

Bibliography

Begüm , Ç., & Ünal, D. (2019). Comparison of Data Mining Classification Algorithms Determining the Default Risk. *Scientific Programming*, 5.