Coursework A

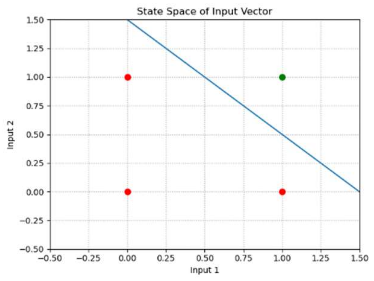
EXERCISE ONE

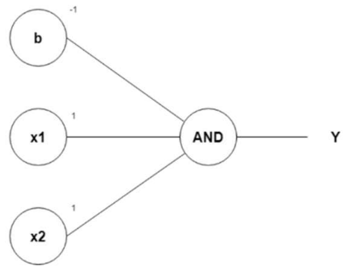
Each perceptron has three inputs: weight 1, weight 2, and a bias.

AND Gate

	Weight 1	Weight 2	Bias
Perceptron 1	2	2	-3

Input x1	Input x2	Output
0	0	0
0	1	0
1	0	0
1	1	1

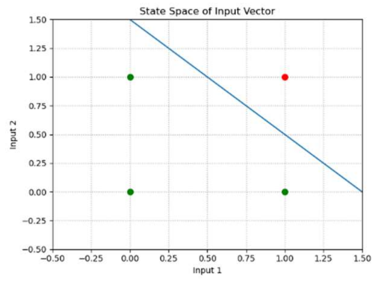


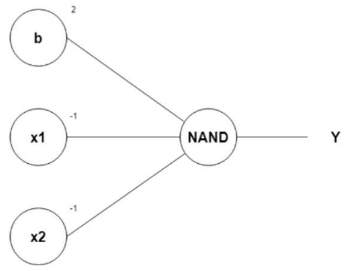


NAND Gate

	Weight 1	Weight 2	Bias
Perceptron 1	-2	-2	3

Input x1	Input x2	Output
0	0	1
0	1	1
1	0	1
1	1	0

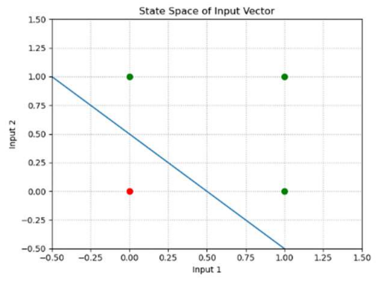


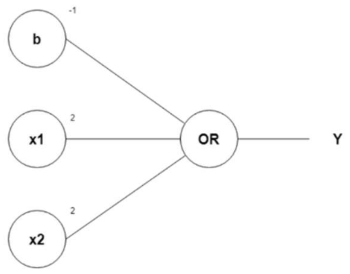


OR Gate

	Weight 1	Weight 2	Bias
Perceptron 1	2	2	1

Input x1	Input x2	Output
0	0	0
0	1	1
1	0	1
1	1	1

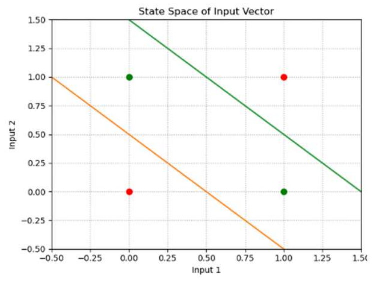


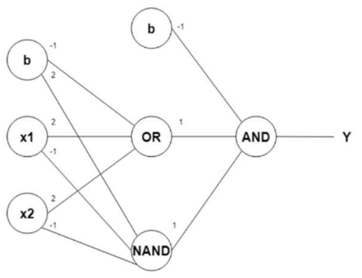


XOR Gate

	Weight 1	Weight 2	Bias
Perceptron 1	-2	-2	3
Perceptron 2	2	2	1
Perceptron 3	2	2	-3

Input x1	Input x2	Output
0	0	0
0	1	1
1	0	1
1	1	0





EXERCISE 2

A custom neural network class was built with support for variable number of inputs, outputs, learning rate, number of hidden layers, and number of perceptrons for each hidden layer. The neural network was built with support for biases automatically adding a bias perceptron to the input and hidden layers.

Variation in Learning Rate and Number of Perceptrons in Hidden Layer

The first step for finding the optimum weights and biases for a neural network is to very the number of perceptrons within the hidden layer and vary the learning rate. A range of perceptrons starting at 50 going to 950 in steps of 50 were tested with a range of learning rates starting at 0.01 to 0.9 increasing logarithmically.

The heatmap seen in Figure 1 shows that at high learning rates (>0.5), the performance of the neural network is less than 90%. This shows how the gradient descent algorithm has converged to a local minima resulting in a suboptimal solution. In the extremes, the large learning rates lead to an unstable learning process oscillating over the epochs.

On the other hand, if the learning rate becomes too small (<0.03), the performance of the neural network is again poor (<92%). This is because the gradient descent algorithm was unable to converge to a minima as there were not enough epochs to accommodate the small learning rate.

Figure 1 also shows that for a small number of perceptrons in the hidden layer also leads to poor performance (<92%). However, as the number increases, little amount of performance gain can be seen. This hints that there is a wide range of the amount of hidden perceptrons that can be used before a noticeable drop in performance can occur.

From this data, the best performance of 96.34% was with 400 hidden nodes and a learning rate of 0.1.

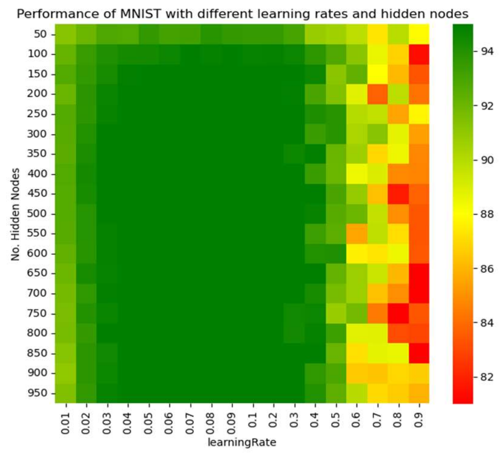


Figure 1 Performance of MNIST with different learning rates and hidden nodes.

Adding Another Hidden Layer

All the test conducted above were conducted with one hidden layer. The tests seen in Figure 2 vary the number of perceptrons in a new second layer from 50 to 950 in steps of 50 using a learning rate of 0.1 and 400 perceptrons in the first hidden layer.

The maximum performance is 94.31% with 100 nodes.

Adding another layer did not increase the performance from 96.34% when using only one hidden layer. This is because the second layer causes the training data to be overfit leading to poor generalisation of the test data.

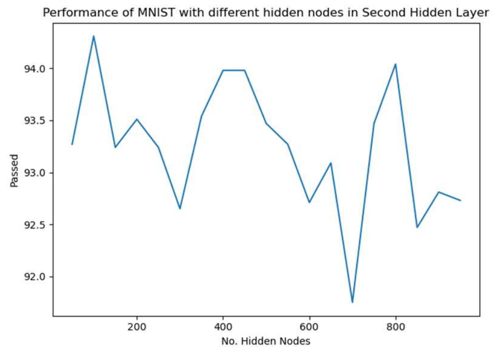


Figure 2 Performance of MNIST with different hidden nodes in second hidden layer.

Increasing Epochs

All the tests conducted so far only used one epoch. A new set of tests was conducted ranging the epoch from 1 to 100 using a neural network with one hidden layer with 400 perceptrons and a learning rate of 0.1. Figure 3 shows the results.

As the number of epochs increases, the performance also increases until it reaches a performance of 97.81% at 11 epochs and then gradually decreases. This is due to the

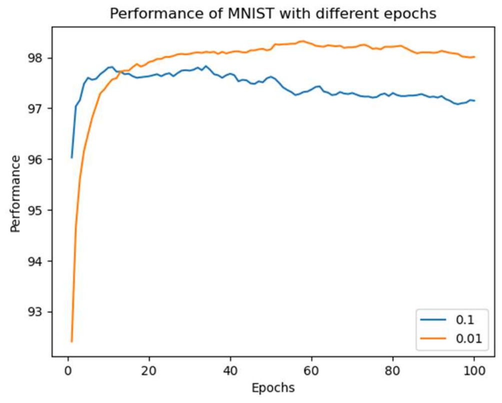


Figure 3 Performance of MNIST with different epochs at 400 perceptrons in hidden layer.

overshoot when searching for the global (or local minima). The gradient descent will continue to try and find the minima with each epoch but as the learning rate is too high, it will never be reached. Therefore, the tests we reconducted with a learning rate of 0.01. Despite having a performance of 92.42% with 1 epoch, as the number of epochs increases, the performance also increased reaching a maximum of 98.32% at 58 epochs. However, once reaching this performance, the same problem occurs the gradient descent algorithm overshoots. This again can be fixed by increasing the number of epochs whilst also decreasing the learning rate however doing so increasing the computational runtime to train the neural network.

The best performance reached with the MNIST dataset is 98.32% with one hidden layer containing 400 nodes, a learning rate of 0.01, and 58 epochs.

Comparison to the Fashion MNIST Dataset

The same tests discussed above were conducted with the fashion MNIST dataset achieving a maximum performance of 88.73%. This was achieved with 150 hidden nodes, a learning rate of 0.01, and 47 epochs.

The reasoning for the poorer performance when compared o the MNIST neural network can be due to the larger number of similarities between each class in the fashion MNIST causing confusion when classifying unknown samples.

A possible solution would be to decrease the learning rate whilst also increasing the number of epochs. However, though doing so would increase the performance of the neural network, it would also increase the runtime to train the neural network.

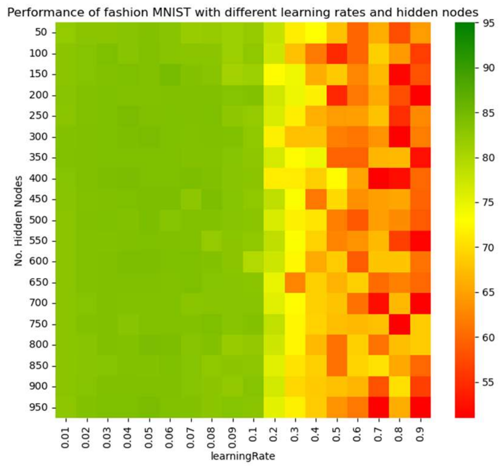


Figure 4 Performance of fashion MNIST with different learning rates and hidden nodes.

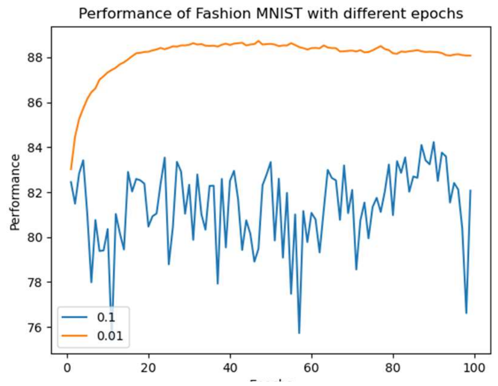


Figure 5 Performance of fashion MNIST with different epochs at 150 perceptrons in hidden layer.

Possible Neural Network Improvements

Despite varying the number of hidden layer, perceptrons within the hidden layers, learning rate, and epochs, the performance could not increase past 99%. There are different machine learning algorithms that could be used to increase performance.

A convolutional neural network uses kernels (also called filters) of a defined size and convolves across the input image and sum the results which is then passed to an activation function. These filters compute new features by only combining features that are near each other in the image. By randomly initializing our filters, we ensure that different filters learn different types of information, like vertical versus horizontal edge detectors. These patterns could be used to classify the dataset more accurately. However, the convolution could cause an increased runtime as more weights need to be updated per perceptron.

Another machine learning algorithm is K-Nearest Neighbours which plots each training image to n-dimensional space (in this case 784 dimensions). The algorithm then plots the unknown image to the same dimension and classifies it by using the most common class out of the k nearest training samples. Another model is the decision tree model which splits the training data based on its features into smaller groups. The feature to split the data is chosen based on that features Gini impurity score. The closer the Gini impurity score is to 0.5, the group is split by 50% resulting in a better division. The process repeats recursively, building the decision tree, until all groups in the leaf nodes contain only one class. The tree, and the decisions are each node, are used to classify the testing data.

To conclude, the best performance with the MNIST sataset was 98.32% an 88.73% with the fashion MNIST dataset. Improvements in the performance could be achieved if using a convolutional neural network; a neural network designed for image processing.