CS 529 - Introduction to Machine Learning

Program 2 – Neural Networks

Submitted By:

Vasman Kaur

Submitted To:

Prof. Lydia Tapia

Part 1: Provide a hypothesis

In terms of accuracy for MINST dataset the gradient descent will work better because it goes through or uses the whole training dataset so, the chances of errors are less. Stochastic gradient descent because it works on same basic idea of gradient descent will also give good accuracy. The main difference their working will have is on time because of the difference in how they iterate. The accuracy overall there will not be major difference for both the methods. For Stochastic gradient descent it can be a little lower.

Default Parameters

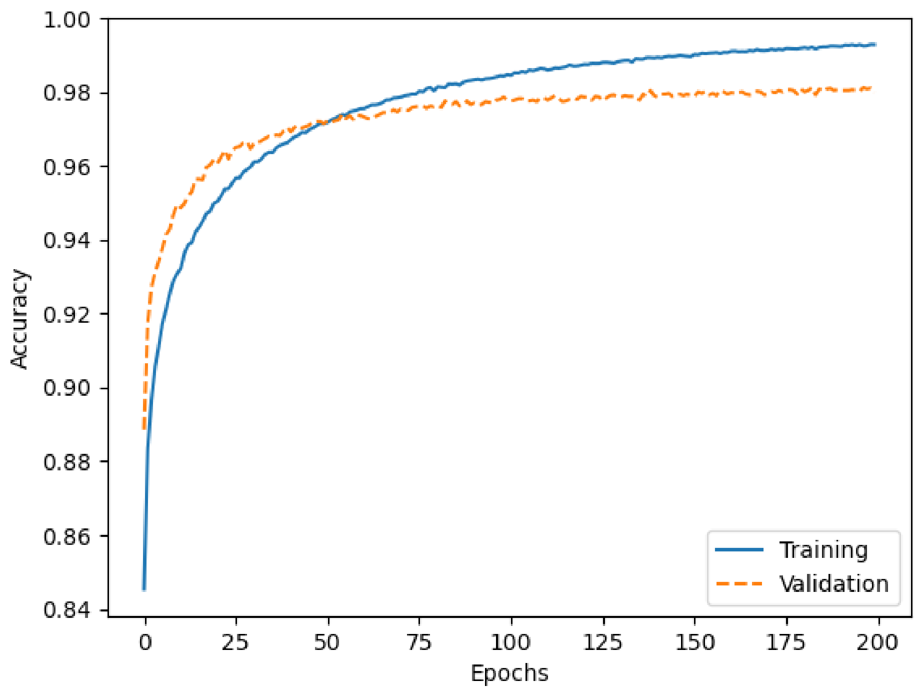
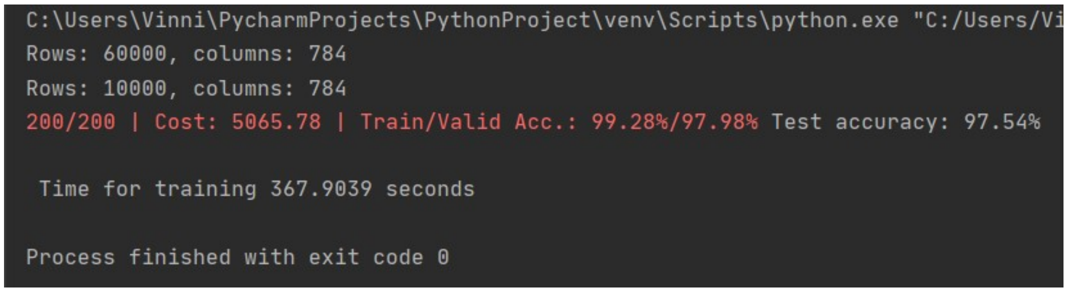


Figure 1.1 (a) minibatch value default



```
C:\Users\Vinni\PycharmProjects\PythonProject\venv\Scripts\python.exe "C:/Users/Vi
Rows: 60000, columns: 784
Rows: 10000, columns: 784
200/200 | Cost: 5065.78 | Train/Valid Acc.: 99.28%/97.98% Test accuracy: 97.54%

 Time for training 367.9039 seconds

Process finished with exit code 0
```

Figure 1.1 (b) minibatch value default

Minibatch value 500

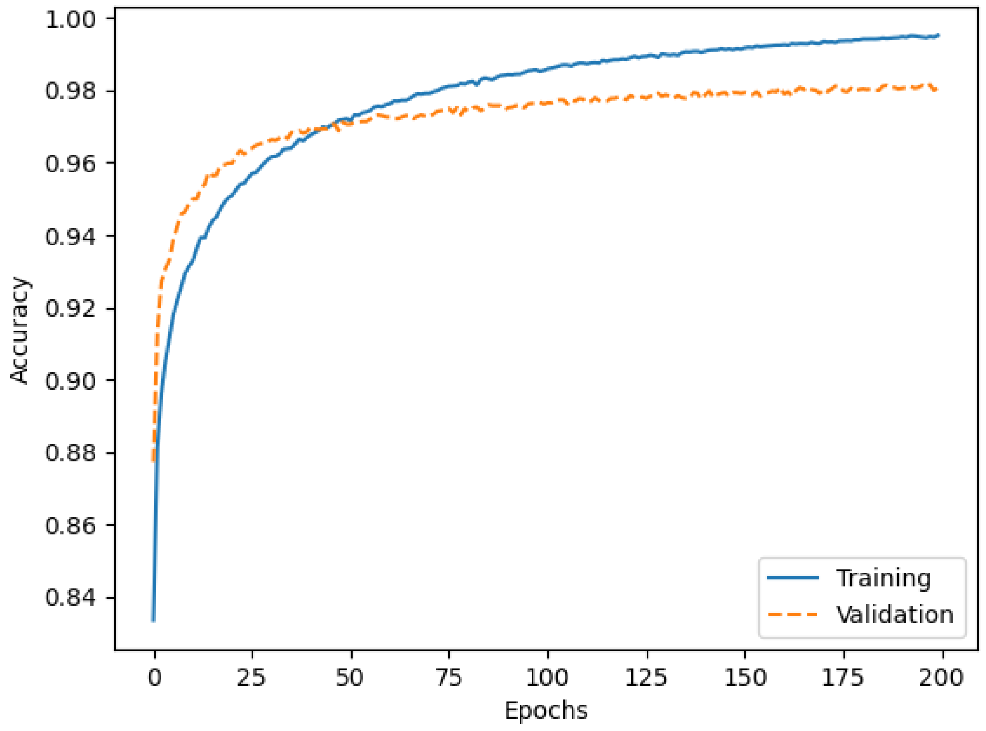
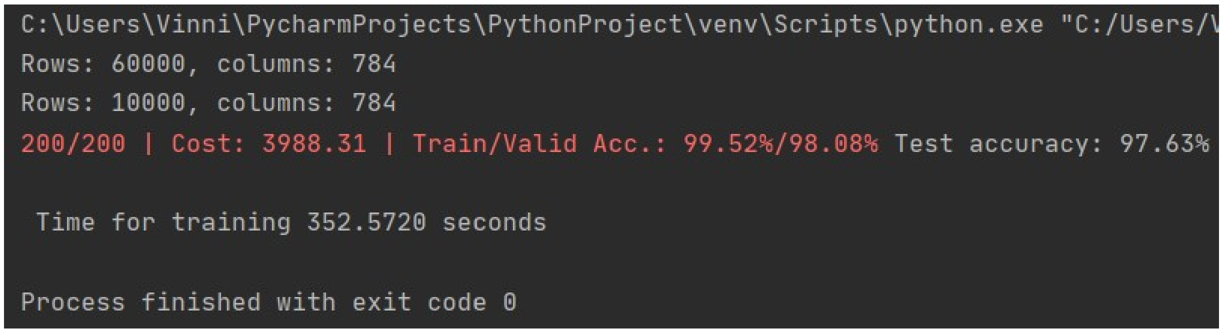


Figure 1.2 (a) minibatch value 500



```
C:\Users\Vinni\PycharmProjects\PythonProject\venv\Scripts\python.exe "C:/Users/V
Rows: 60000, columns: 784
Rows: 10000, columns: 784
200/200 | Cost: 3988.31 | Train/Valid Acc.: 99.52%/98.08% Test accuracy: 97.63%

 Time for training 352.5720 seconds

Process finished with exit code 0
```

Figure 1.2 (b) minibatch value 500

Minibatch value 1000

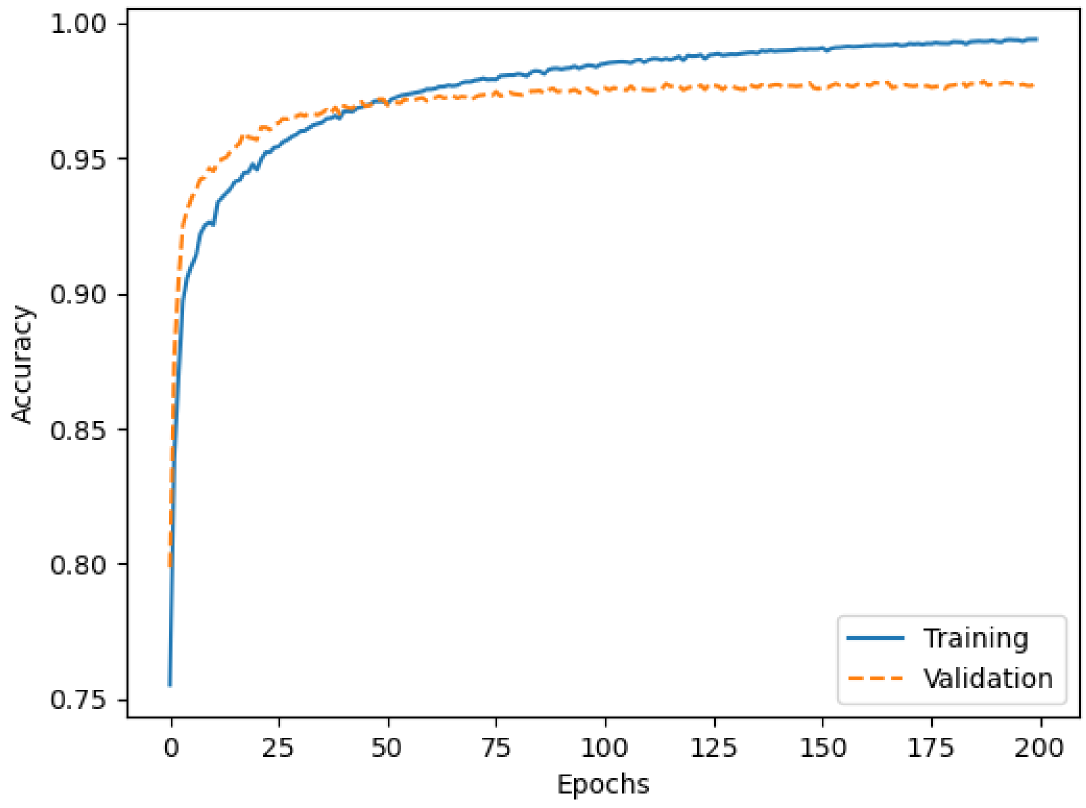
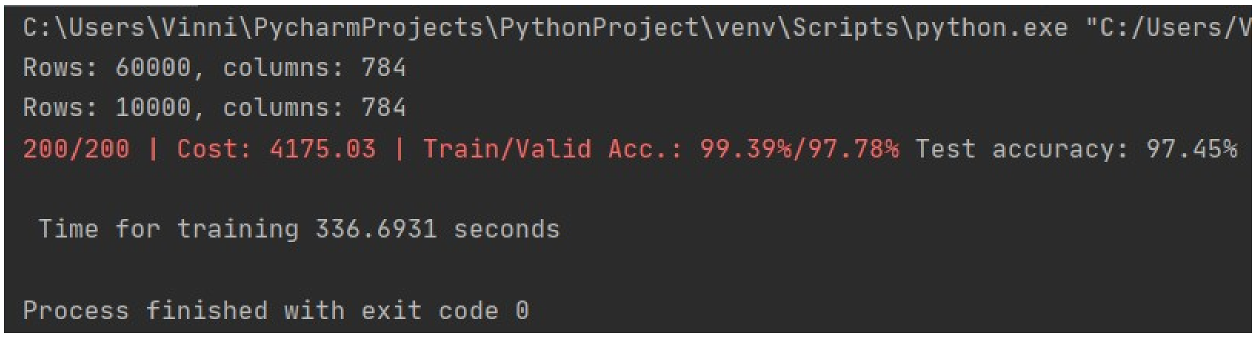


Figure 1.3 (a) minibatch value 1000



```
C:\Users\Vinni\PycharmProjects\PythonProject\venv\Scripts\python.exe "C:/Users/V
Rows: 60000, columns: 784
Rows: 10000, columns: 784
200/200 | Cost: 4175.03 | Train/Valid Acc.: 99.39%/97.78% Test accuracy: 97.45%

 Time for training 336.6931 seconds

Process finished with exit code 0
```

Figure 1.3 (b) minibatch value 1000

Minibatch 1500

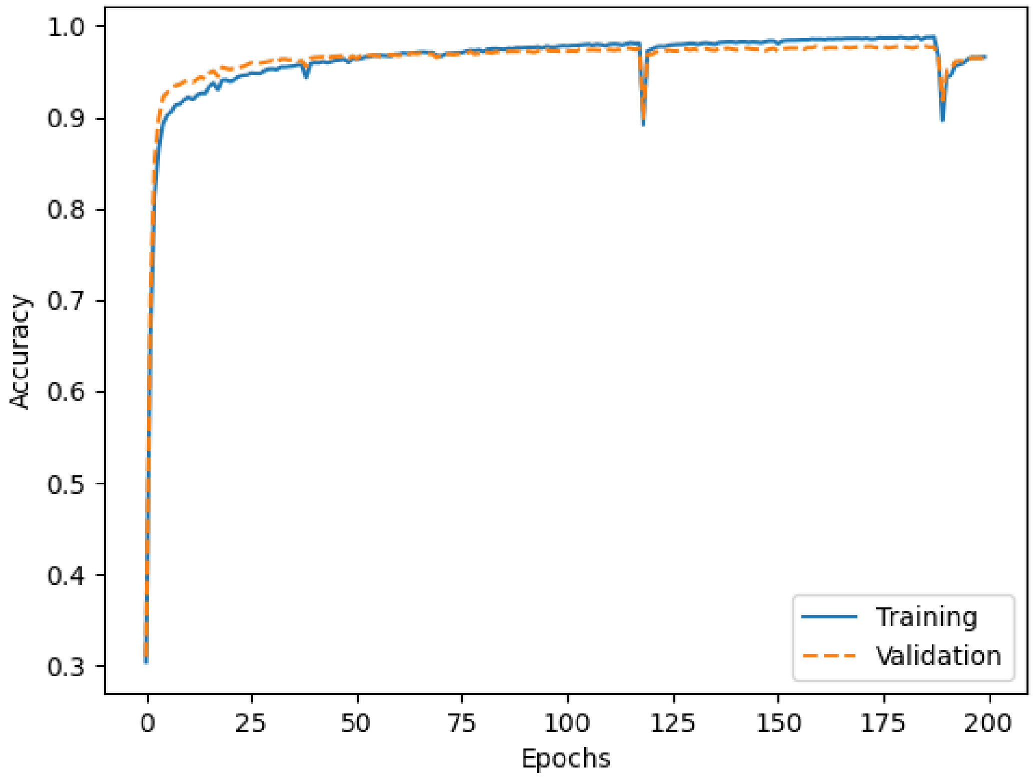
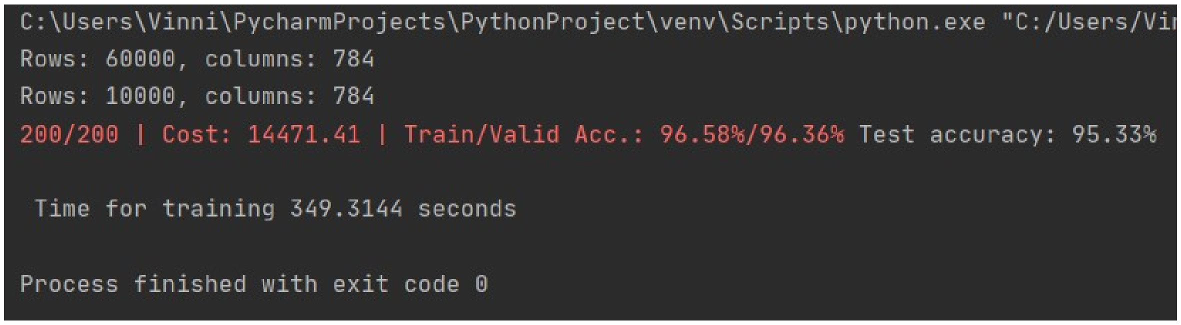


Figure 1.4 (a) minibatch value 1500



```
C:\Users\Vinni\PycharmProjects\PythonProject\venv\Scripts\python.exe "C:/Users/Vi
Rows: 60000, columns: 784
Rows: 10000, columns: 784
200/200 | Cost: 14471.41 | Train/Valid Acc.: 96.58%/96.36% Test accuracy: 95.33%

 Time for training 349.3144 seconds

Process finished with exit code 0
```

Figure 1.4 (b) minibatch value 1500

From the above plots for different minibatch values (default, 500,100,1500) it can be seen that as the minibatch value increases the accuracy decreases even though the accuracy decrease was slight, the cost and time for training was seen to have higher difference. There was a decreases in their values but to an extent and then the values became worse (for minibatch value 1500). So, having minibatch value not too high is seen to be better. My hypothesis is seen to be true. There is not extreme difference in the accuracy output for stochastic gradient descent and gradient descent.

Part 2: Hyper parameter tuning for MINST dataset.

Below are the plots for the variations made for the parameters

2.1 Hidden Layer 90

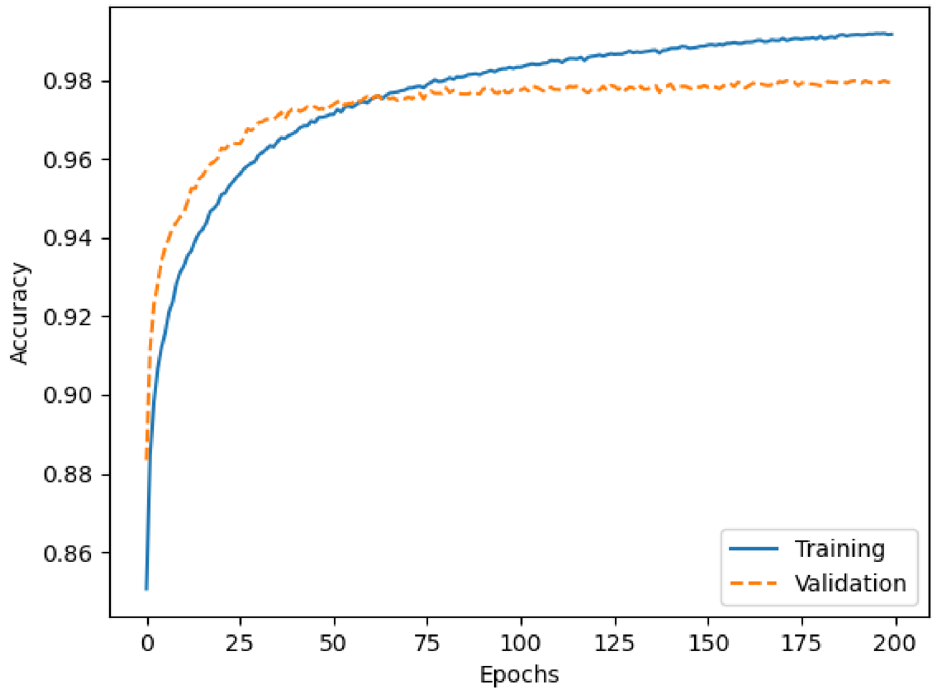
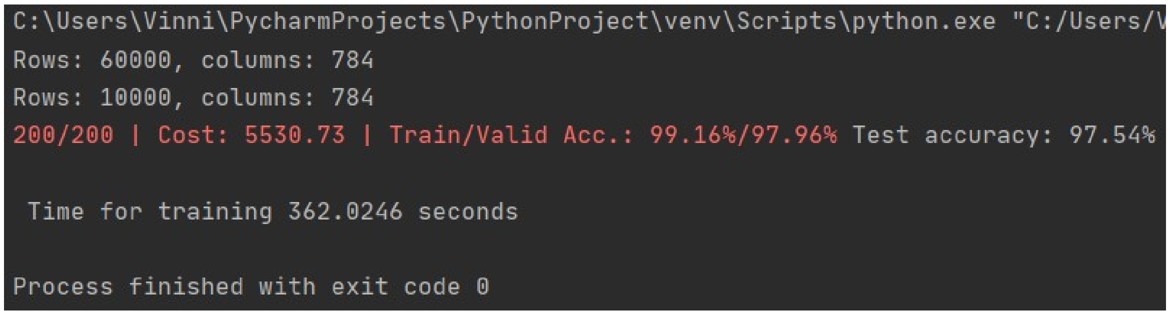


Figure 2.1 (a) Hidden layer size 90



```
C:\Users\Vinni\PycharmProjects\PythonProject\venv\Scripts\python.exe "C:/Users/V
Rows: 60000, columns: 784
Rows: 10000, columns: 784
200/200 | Cost: 5530.73 | Train/Valid Acc.: 99.16%/97.96% Test accuracy: 97.54%

 Time for training 362.0246 seconds

Process finished with exit code 0
```

Figure 2.1 (b) Hidden layer size 90

2.2 Hidden Layer 50

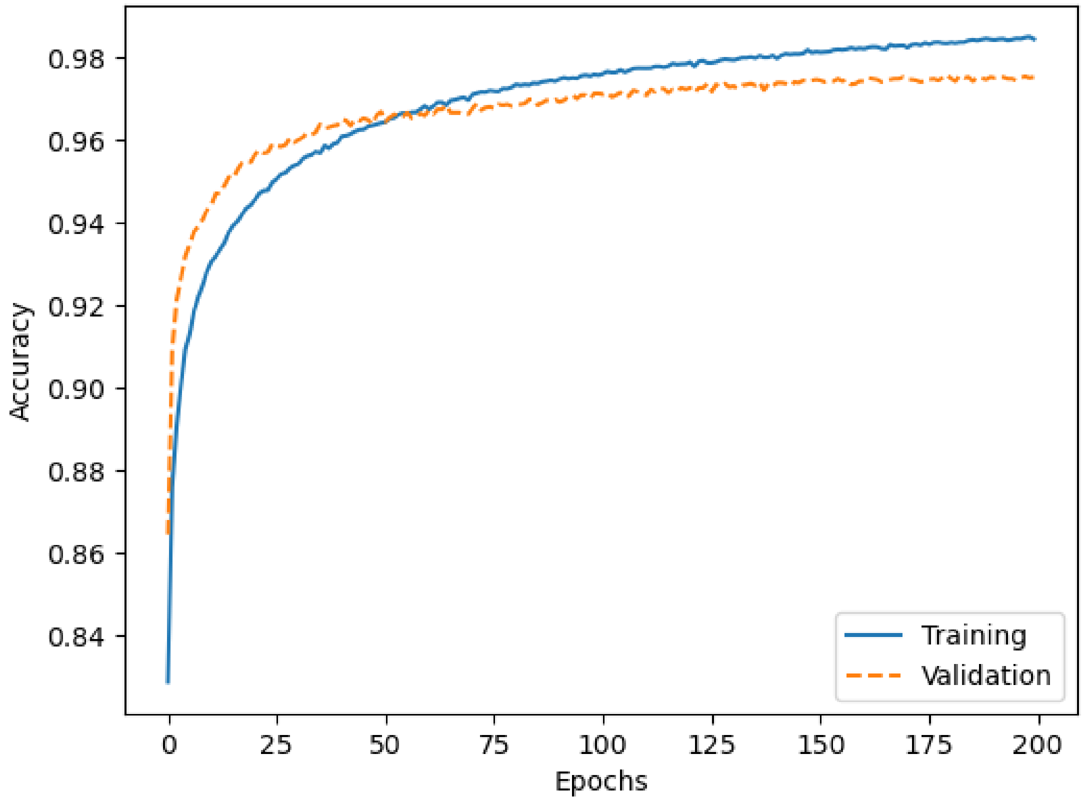
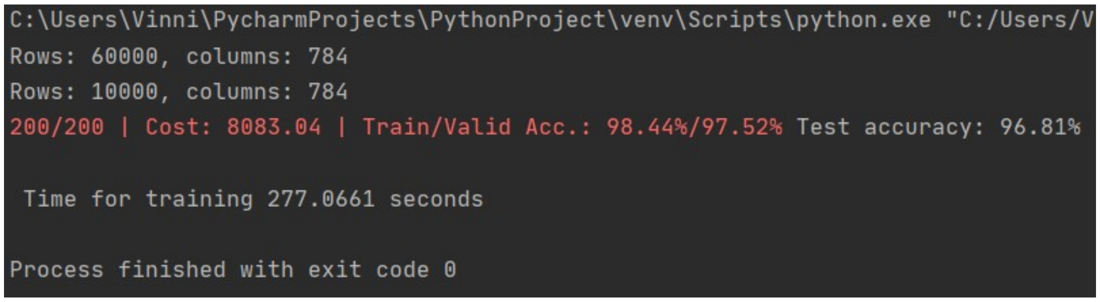


Figure 2.2 (a) Hidden layer size 50


```
C:\Users\Vinni\PycharmProjects\PythonProject\venv\Scripts\python.exe "C:/Users/V
Rows: 60000, columns: 784
Rows: 10000, columns: 784
200/200 | Cost: 8083.04 | Train/Valid Acc.: 98.44%/97.52% Test accuracy: 96.81%

 Time for training 277.0661 seconds

Process finished with exit code 0
```

Figure 2.2 (b) Hidden layer size 50

2.3 Epoch 800, minibatch 500

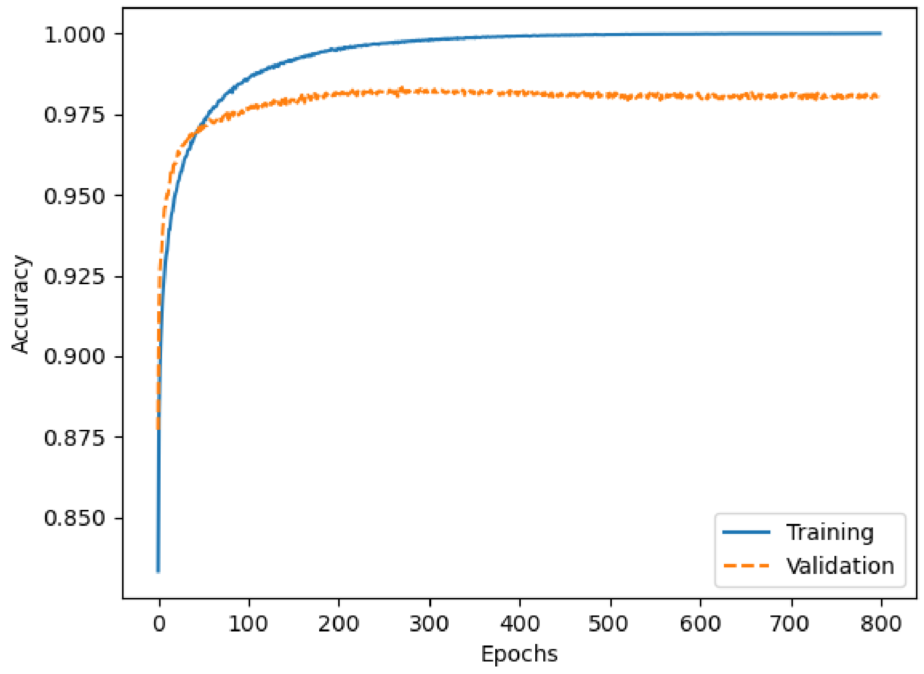
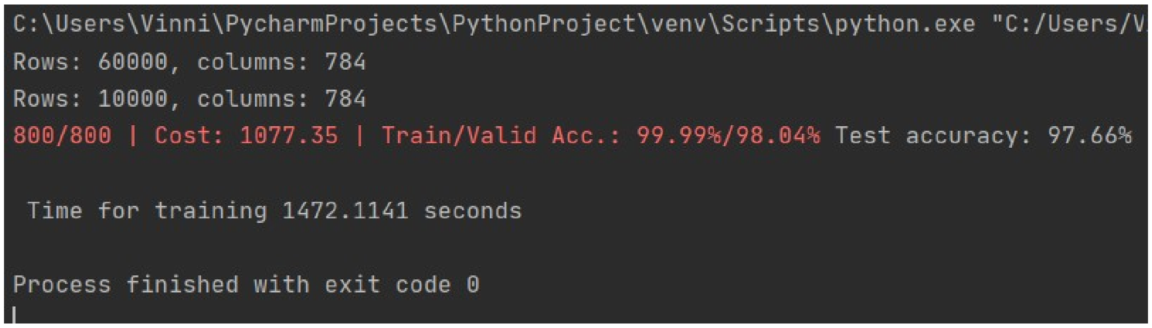


Figure 2.3 (a) Epoch 800, minibatch 500


```
C:\Users\Vinni\PycharmProjects\PythonProject\venv\Scripts\python.exe "C:/Users/V
Rows: 60000, columns: 784
Rows: 10000, columns: 784
800/800 | Cost: 1077.35 | Train/Valid Acc.: 99.99%/98.04% Test accuracy: 97.66%

 Time for training 1472.1141 seconds

Process finished with exit code 0
```

Figure 2.3 (b) Epoch 800, minibatch 500

2.4 L2 0.3, minibatch 500

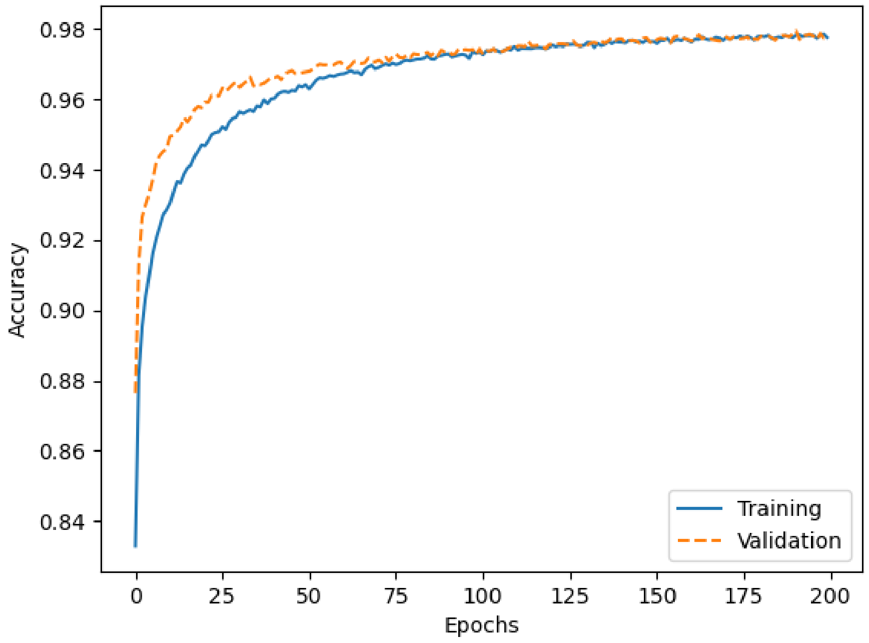
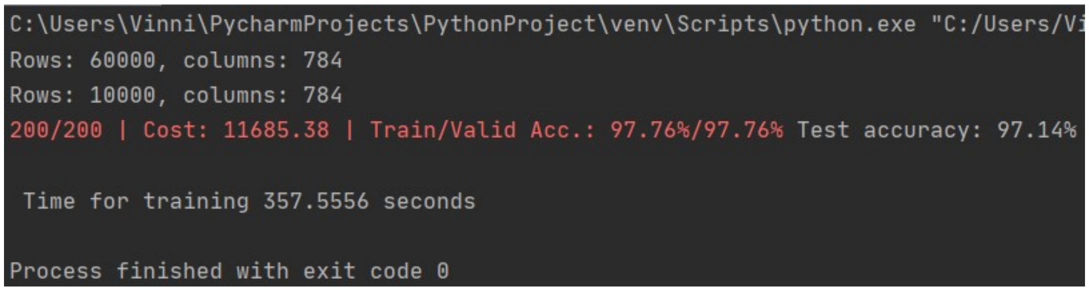


Figure 2.4 (a) L2 0.3, minibatch 500



```
C:\Users\Vinni\PycharmProjects\PythonProject\venv\Scripts\python.exe "C:/Users/Vi
Rows: 60000, columns: 784
Rows: 10000, columns: 784
200/200 | Cost: 11685.38 | Train/Valid Acc.: 97.76%/97.76% Test accuracy: 97.14%

 Time for training 357.5556 seconds

Process finished with exit code 0
```

Figure 2.4 (b) L2 0.3, minibatch 500

The parameters changed to see the impact include hidden layer, minibatch, L2 and Epoch. Changing these parameters did not show high impact on the test accuracy. But again, showed difference in the values for cost and training time. The accuracy for the changed parameters changes the accuracy for training and validation data, but the test accuracy remains almost the same with minor difference. This in fact can be seen to be closer to the test accuracy with default parameters which can be seen in Figure 1.1(a). There is no overfitting because the gap between training/testing accuracy is small.

The parameters for Kaggle submission which gave the accuracy of 0.97630 are as follows:

n_hidden=100, l2=0.01, epochs=200, eta=0.0005, minibatch_size=500, shuffle=True, seed=1

Part 3: Implement Momentum

Momentum 0.0

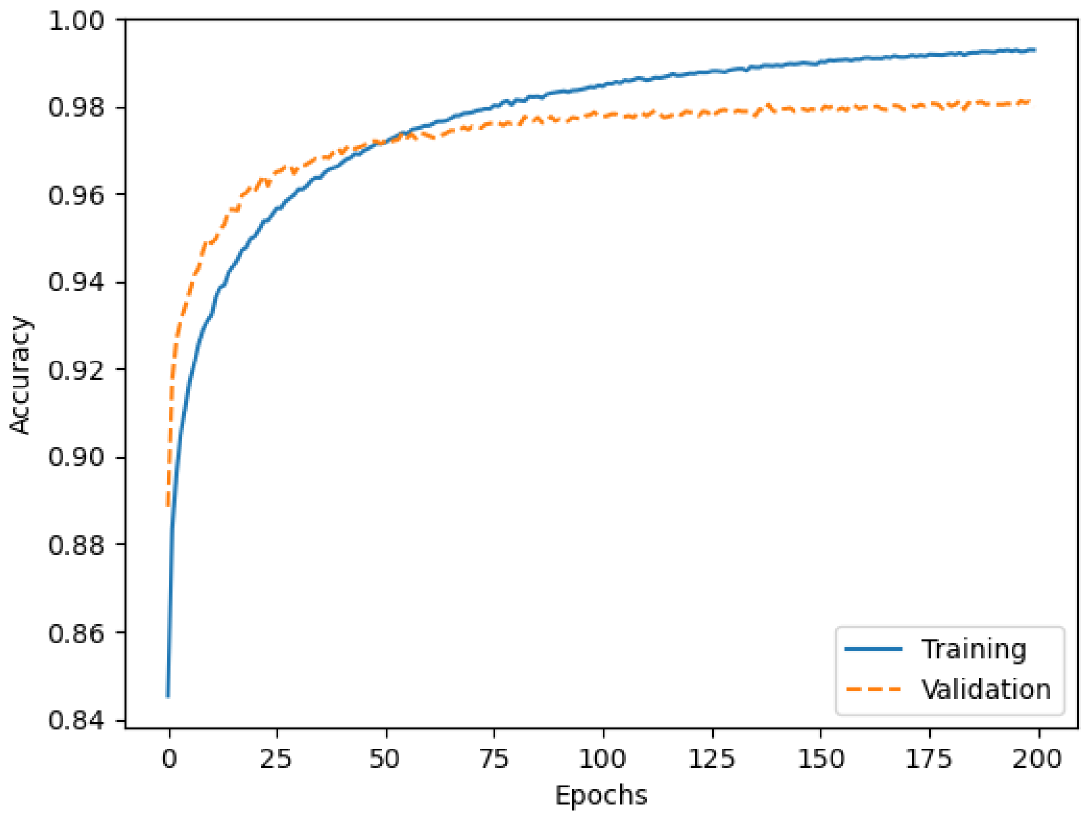
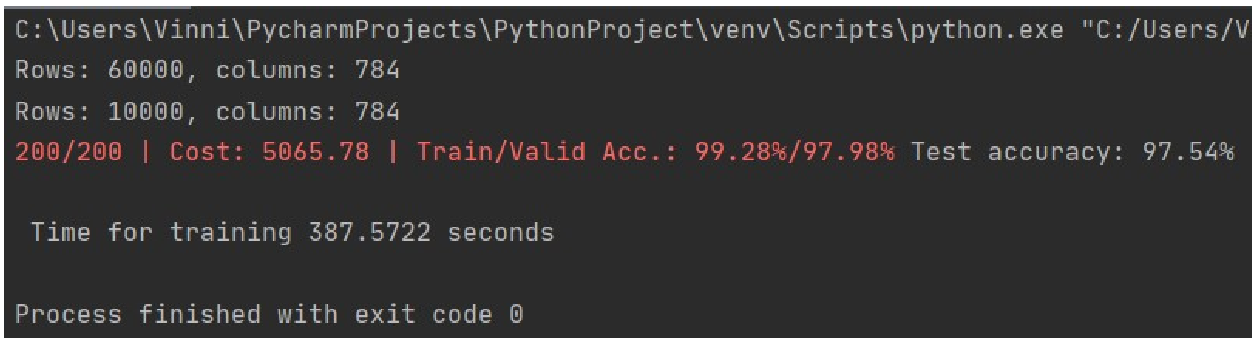


Figure 3.1 (a) Momentum 0.0



```
C:\Users\Vinni\PycharmProjects\PythonProject\venv\Scripts\python.exe "C:/Users/V
Rows: 60000, columns: 784
Rows: 10000, columns: 784
200/200 | Cost: 5065.78 | Train/Valid Acc.: 99.28%/97.98% Test accuracy: 97.54%

 Time for training 387.5722 seconds

Process finished with exit code 0
```

Figure 3.1 (b) Momentum 0.0

Momentum 0.1

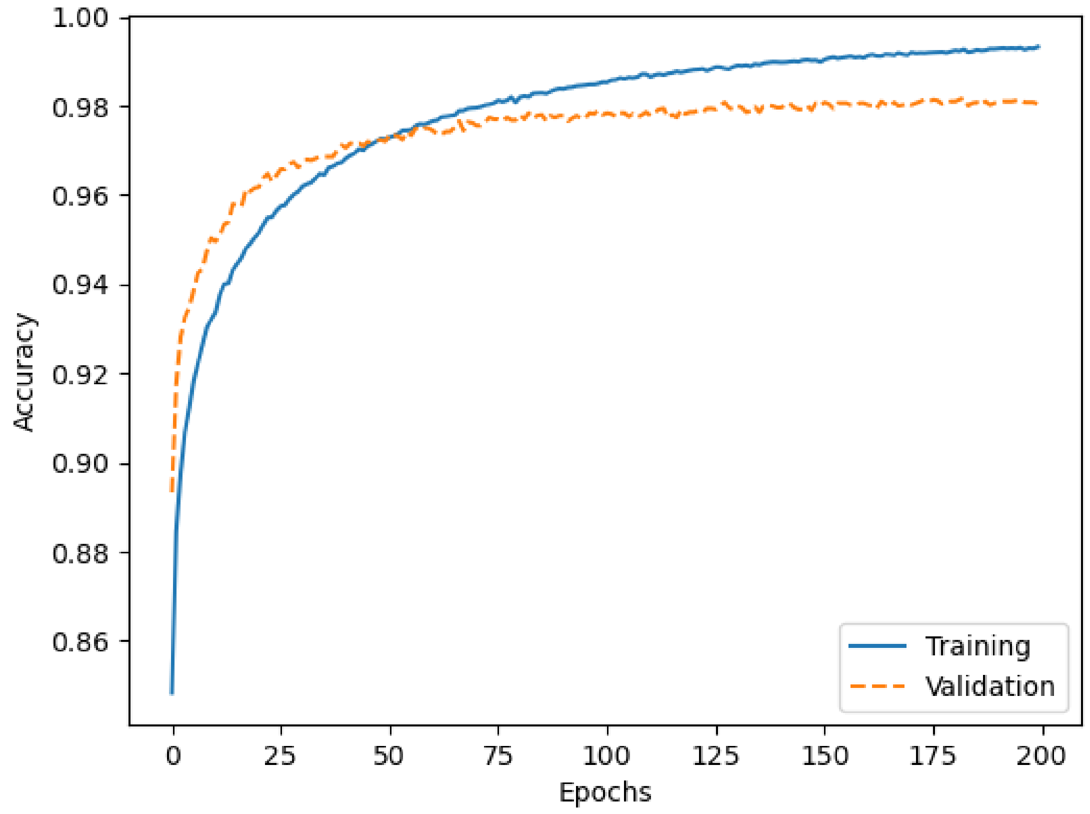
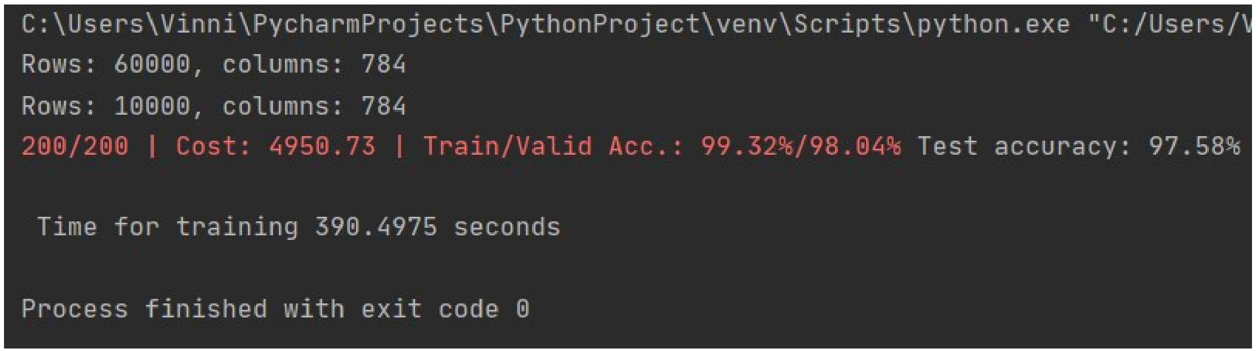


Figure 3.2 (a) Momentum 0.1



```
C:\Users\Vinni\PycharmProjects\PythonProject\venv\Scripts\python.exe "C:/Users/V
Rows: 60000, columns: 784
Rows: 10000, columns: 784
200/200 | Cost: 4950.73 | Train/Valid Acc.: 99.32%/98.04% Test accuracy: 97.58%

 Time for training 390.4975 seconds

Process finished with exit code 0
```

Figure 3.2 (b) Momentum 0.1

Momentum 0.5

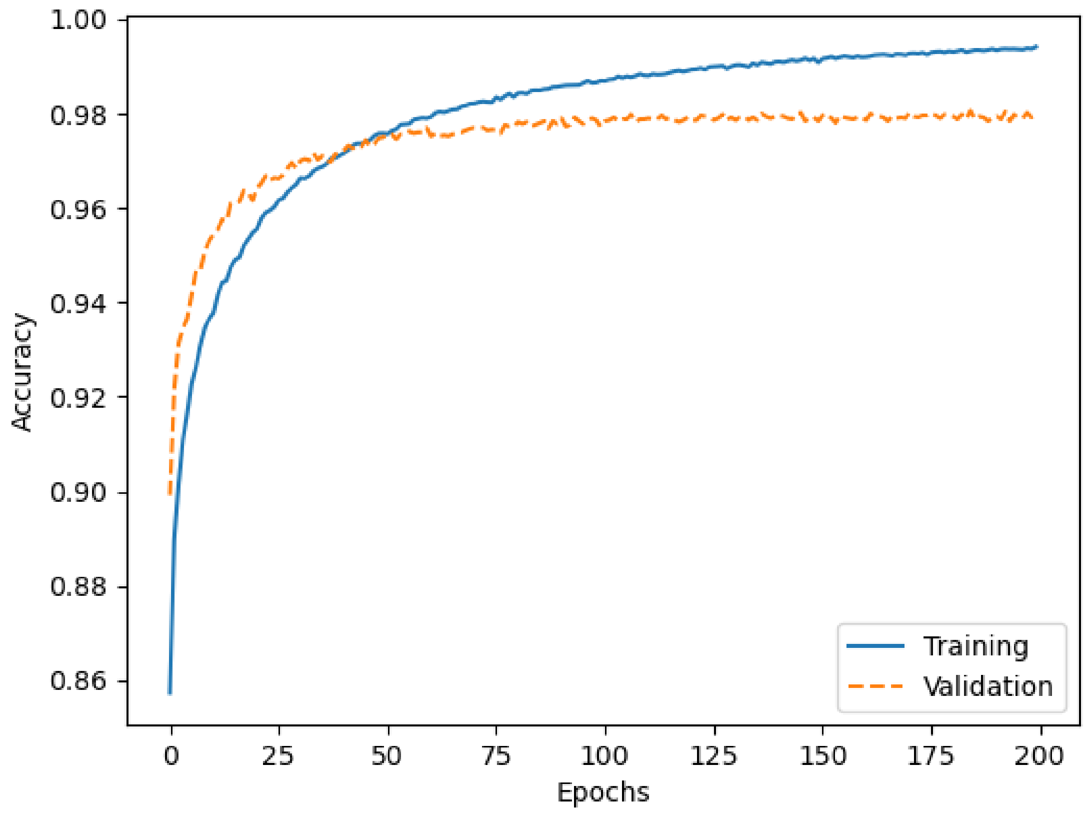
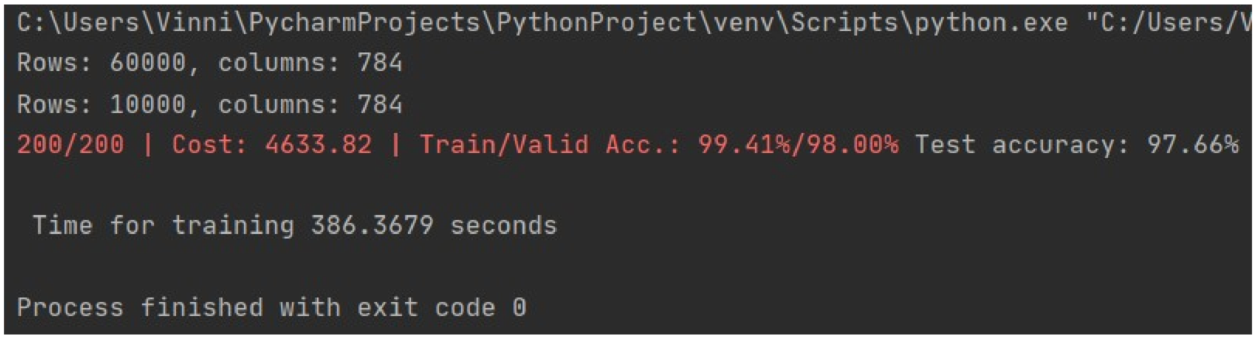


Figure 3.3 (a) Momentum 0.5



```
C:\Users\Vinni\PycharmProjects\PythonProject\venv\Scripts\python.exe "C:/Users/V
Rows: 60000, columns: 784
Rows: 10000, columns: 784
200/200 | Cost: 4633.82 | Train/Valid Acc.: 99.41%/98.00% Test accuracy: 97.66%

 Time for training 386.3679 seconds

Process finished with exit code 0
```

Figure 3.3 (b) Momentum 0.5

Momentum 0.9

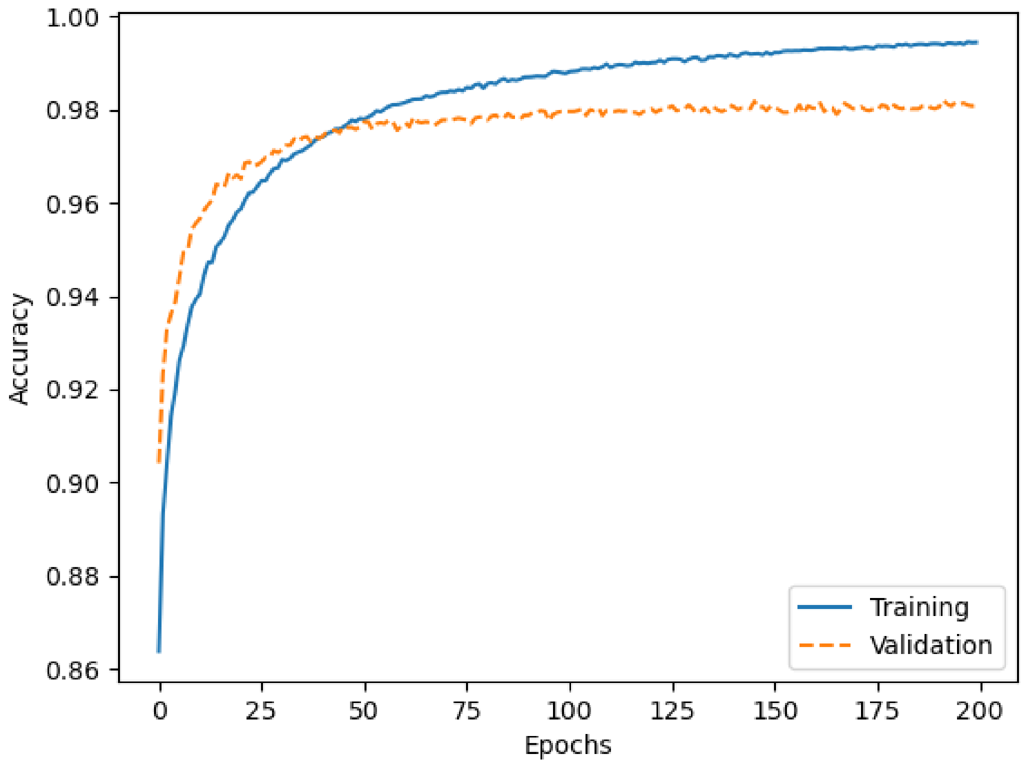
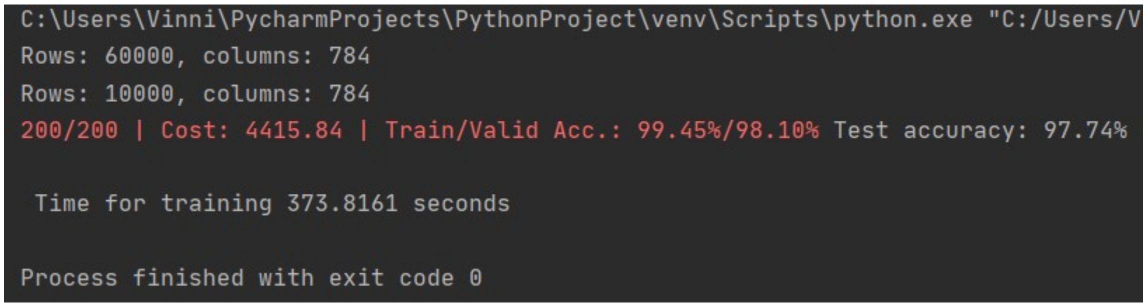


Figure 3.4 (a) Momentum 0.9



```
C:\Users\Vinni\PycharmProjects\PythonProject\venv\Scripts\python.exe "C:/Users/V
Rows: 60000, columns: 784
Rows: 10000, columns: 784
200/200 | Cost: 4415.84 | Train/Valid Acc.: 99.45%/98.10% Test accuracy: 97.74%

 Time for training 373.8161 seconds

Process finished with exit code 0
```

Figure 3.4 (b) Momentum 0.9

From the above plots for momentum value0.0, 0.1, 0.5, 0.9. The output with momentum value 0.9 is good in all terms i.e. cost, test accuracy, and time for training. The accuracy values for training and validation data are also better for momentum 0.9. Because momentum uses the previous weight values to update the weights it is better to use for output as it can give better results.
This means that momentum has high effect on convergence as the values are better. Momentum is connected to speeding things as it works with previous weight values so it will lead to easy convergence.

Part 4: Working with wine dataset

Accuracy on Kaggle: 0.44200

Parameters: n_hidden=10, l2=0.01, epochs=200, eta=0.0005, minibatch_size=500, shuffle=True, seed=1

For the wine dataset, the default parameters were giving good test accuracy. From the part one minibatch size 500 was seen to give better results as discussed in that section so, I chose that value and ran the program. Changing value for l2 was making things a little on the bad side so I kept it as it is. Changing values for eta was also not much fruitful. But changing n_hidden, decreasing it made the accuracy better. I started decreasing the n_hidden value because at 100 it gave the accuracy of 42% which is not good. This accuracy is actually not good as compared to the accuracy using decision trees.

Part 5: Multi multi layer perceptron

Accuracy on Kaggle: 0.42800

Parameters: l2=0.01, epochs=2, eta=0.0005, minibatch_size=50, shuffle=True, seed=1, n_hidden_layer = [10,5]

The parameters were chosen as described in part 4.

Part 6: Other Problems

Yeast Dataset: The network design for this is 8 inputs as there are 8 attributes and 10 output nodes because there are 10 classes and sigmoid threshold because of linearly inseparable data. The number of hidden layer will be one with lesser nodes, 1 or 2 because there are some values that not useful.

So the parameters will be : l2=0.01, epochs=2, eta=0.0005, minibatch_size=50, shuffle=True, seed=1, n_hidden= 2

Modifies Wine: The network design for this is 11 inputs as there are 11 attributes and 10 output nodes because there are 10 outputs (quality which ranges between 1 to 10). The number of hidden layer will be one and hidden nodes will be less than 11 so 5.

So the parameters will be : l2=0.01, epochs=2, eta=0.0005, minibatch_size=50, shuffle=True, seed=1, n_hidden= 5

Smart Thermostat: The network design will have seven inputs because 7 rooms and one output which is the time it takes for the whole house to set to that temperature. With 4 hidden layer nodes and a single hidden layer and sigmoid function. The network will be fully connected in all three cases.

So the parameters will be : l2=0.01, epochs=2, eta=0.0005, minibatch_size=50, shuffle=True, seed=1, n_hidden= 4

In all three cases lower value for l2 and eta will be suitable and these wont require more than one hidden layer.