Delft University of Technology

Computational Intelligence

CSE 2530

Assignment 1:

Multilayer Perceptron

Authors:

Akdemir, Rauf
Farooq, Shah Muhammad
Khan, Zeeshan
Sahin, Tamer

Exercise 1

The single perceptron is able to learn for the OR and AND gate functions quite well since the outputs are linearly separable, this means that there exists a single decision boundary separating the two types of classes. This is represented by the fact that both the OR and AND perceptron are able to reach an error of 0 and not deviate, as seen in figure 1 and 2. Contrarily, the XOR function is not linearly separable therefore the single perceptron is unable to learn and predict correct values leading to a high mean squared error, as seen in figure 3. Multiple random initialization of weights and bias were run and similar results were returned.

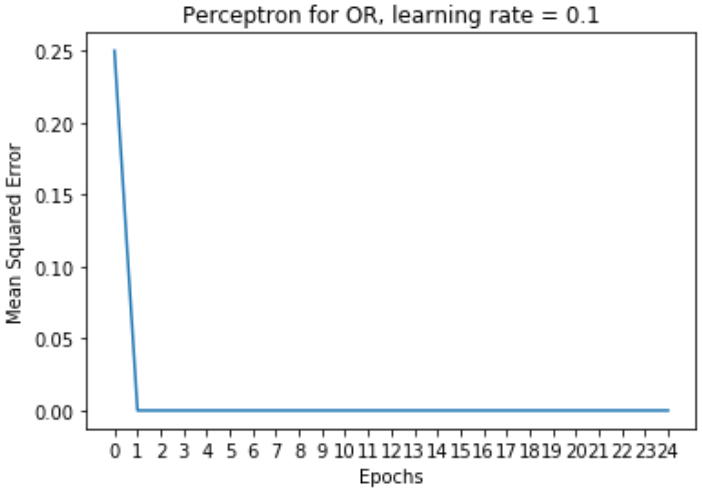


Figure 1

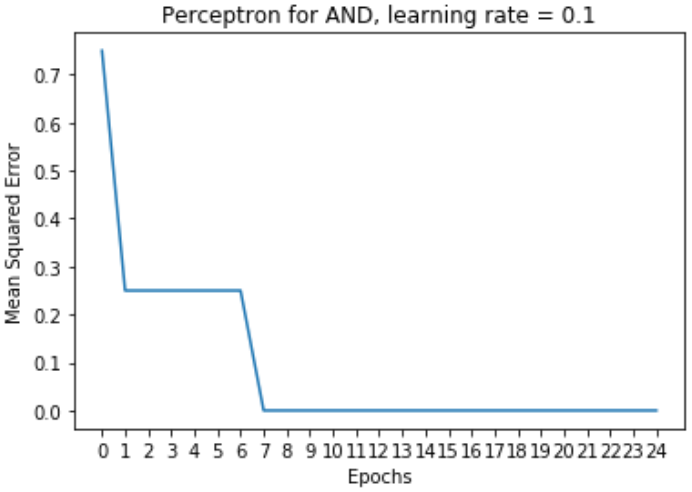


Figure 2

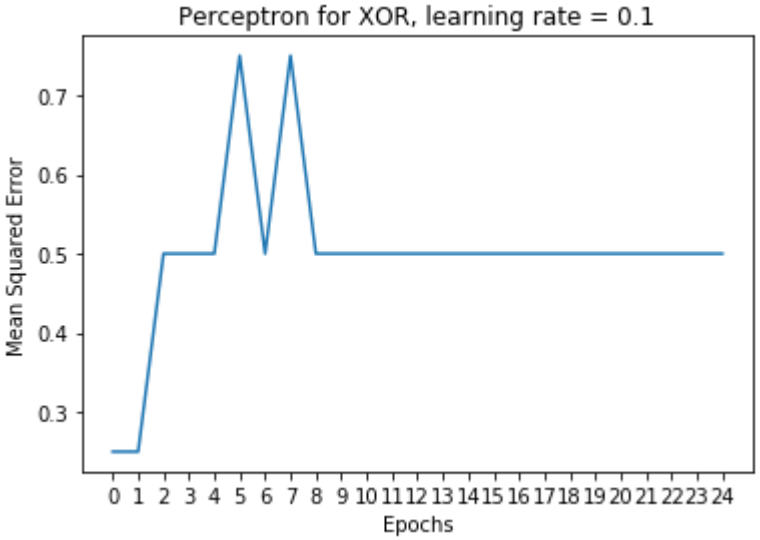


Figure 3

Exercise 2

One neuron per feature is required since the features set shows that there are **10** input **features** therefore **10** input **neurons** are required.

Exercise 3

Since there are 7 different target classes, we need **7 output neurons**

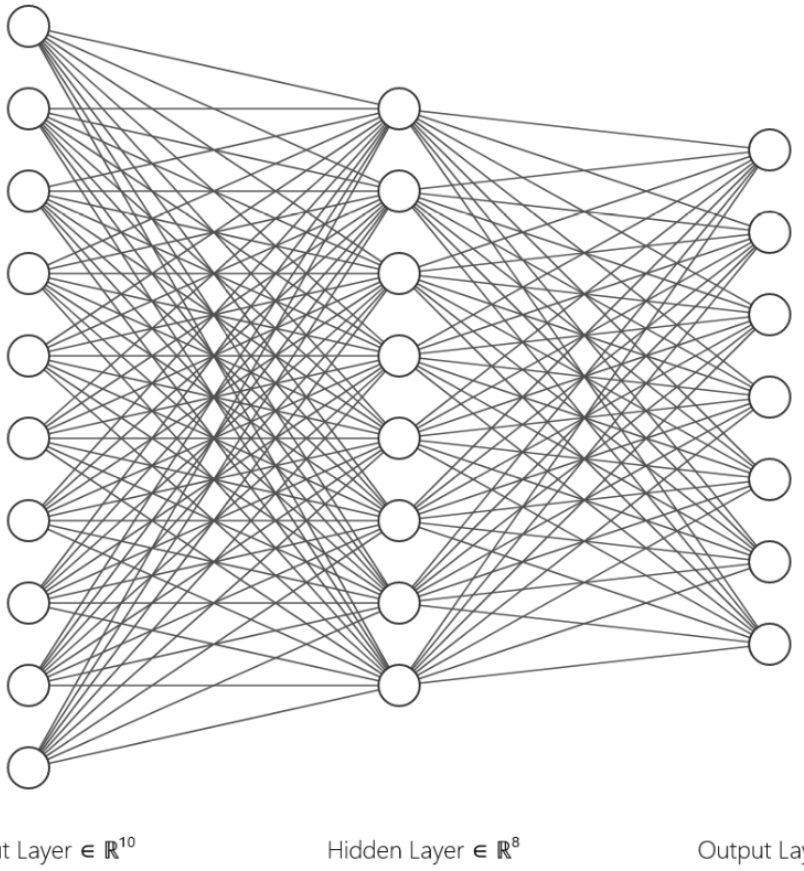
Exercise 4

We estimated having at least **8 hidden neurons** and **1 hidden layer**. For this dataset, this estimation should be a good starting point to achieve good results to improve upon since we have more than the minimum required neurons and one hidden layer should be enough to map the features to their respective classes.

Exercise 5

We plan to use several activation functions such as **ReLU** and **sigmoid**. The problem at hand is a classification problem, therefore the sigmoid function would provide a more continuous output combined with the softmax function which normalizes the result of the activation function into a probability distribution consisting of k probabilities proportional to the exponentials of the inputs.

Exercise 6



Exercise 7

The idea of dividing the data in training, validation and tests sets is to prevent overfitting and predict new data as accurately as possible; it is therefore divided into ratios of **80:10:10** respectively. 80% of the data is used in the **training** set to fit the parameters of the model. The **validation** set provides an **unbiased** evaluation of a model fit on the **training** dataset while tuning the model's **hyperparameters** to show signs of possible **overfitting**. The **test** set is used to provide an unbiased evaluation of the **tuned model** fit against **unseen** data.

Exercise 8

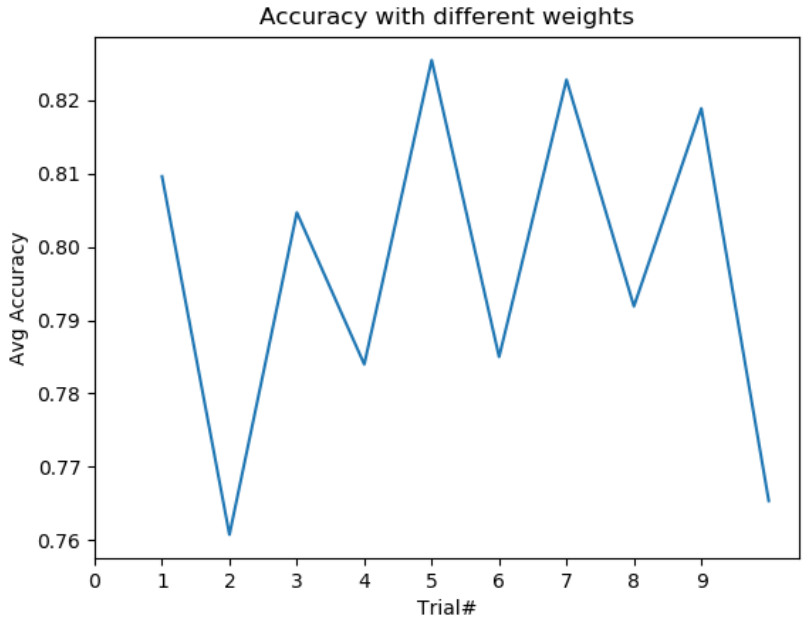
We evaluate the performance of this network based on **accuracy** as a metric on the validation set. This accuracy is the fraction of the correct predictions over the total number of predictions made on the given dataset. It is also important to see if the model does not bias towards the most popular class. Therefore we found the target distributions as follows: {1: 1119, 2: 1117, 3: 1131, 4: 1111, 5: 1127, 6: 1126, 7: 1123} which shows an almost balanced set of data with the average number of instances for a classes being 1122. Hence, the use of accuracy as a performance metric

Exercise 9

We stop the training once the number of epochs have been reached or through the early stopping method when the cross entropy loss is less than 0.1 when running our stochastic gradient descent with momentum model. However, if the number of epochs is very large the model there is a high possibility that the model might overfit and comes at the expense of higher generalization error.

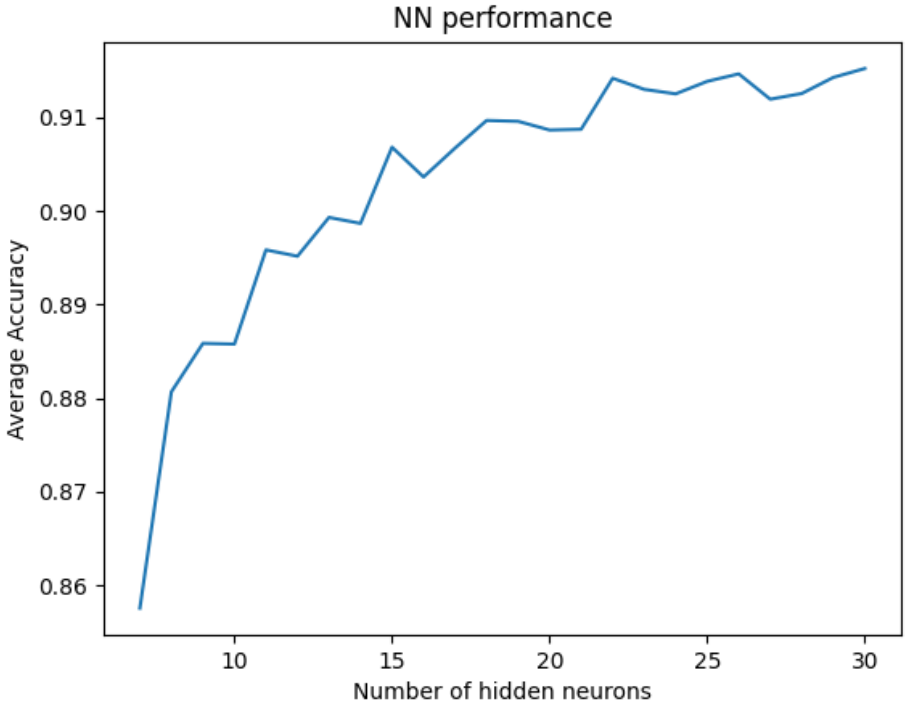
Exercise 10

The weights are initialized at the start by a random number generator in a uniform distribution from the numpy library. These weights affect the performance of the NN since the backpropagation algorithm needs to be able to adjust these weights with the goal to return predictions with higher accuracies. If we were to select weights we might fall into the trap of accepting a local minima instead of reaching for the global minimum to improve the model. As seen in the below figure, there is high variance in the accuracy.



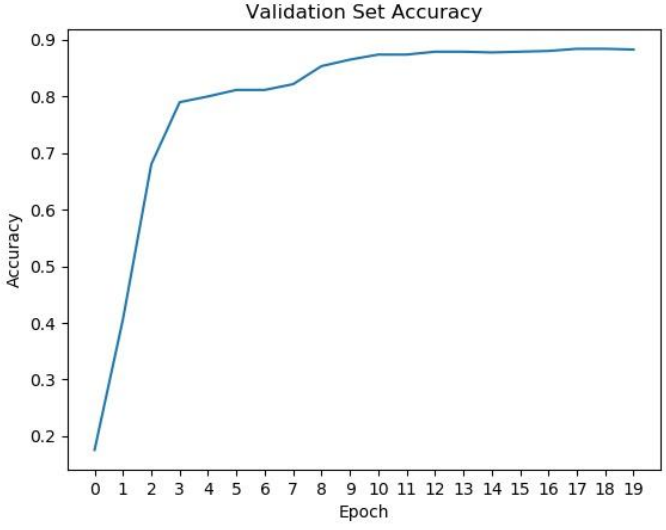
Exercise 11

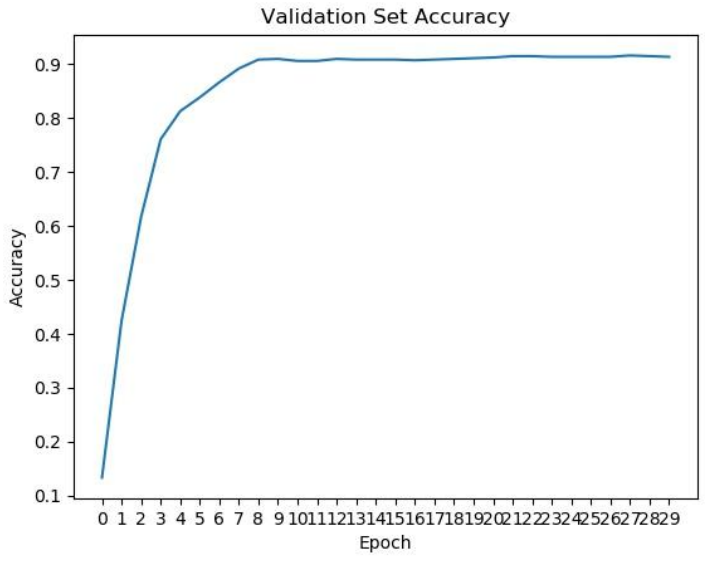
We make use of the popular k-fold cross validation method to find out the best number of neurons that would be suited for the model and that are tested on the validation set returning the average accuracy. Due to the size of the dataset, we estimate k = 10 would be sufficient, this means 10 folds are made over the model and their final accuracies are averaged. The other hyperparameters remain constant to ensure fairness in results. The model is tested in the range of 7-30 hidden neurons. The figure below shows the highest average accuracy of ~92% was reached at 30 neurons, it is interesting to note that 22-25 neurons performed similarly well. The cause?

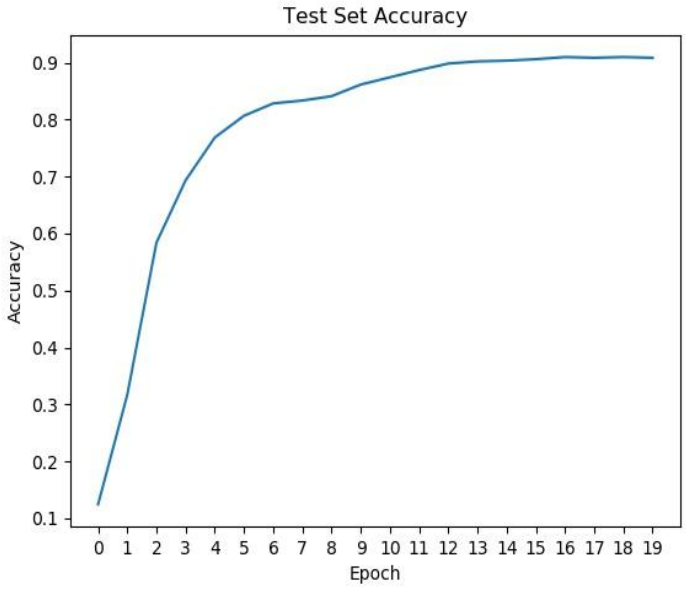


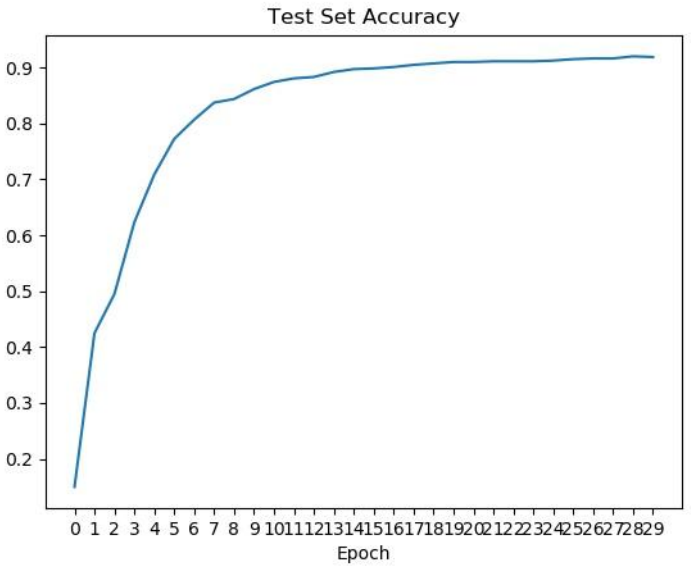
Exercise 12

As seen from the k-fold cross validation, we select the following parameters to be suitable: 22 neurons, 0.01 learning rate. A test is made for comparison for the number of epochs between 20 and 30 to have a clearer idea for the best results.



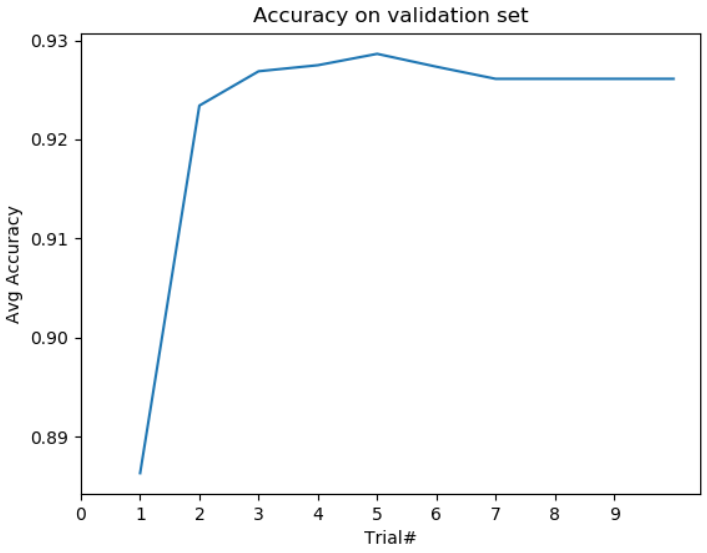


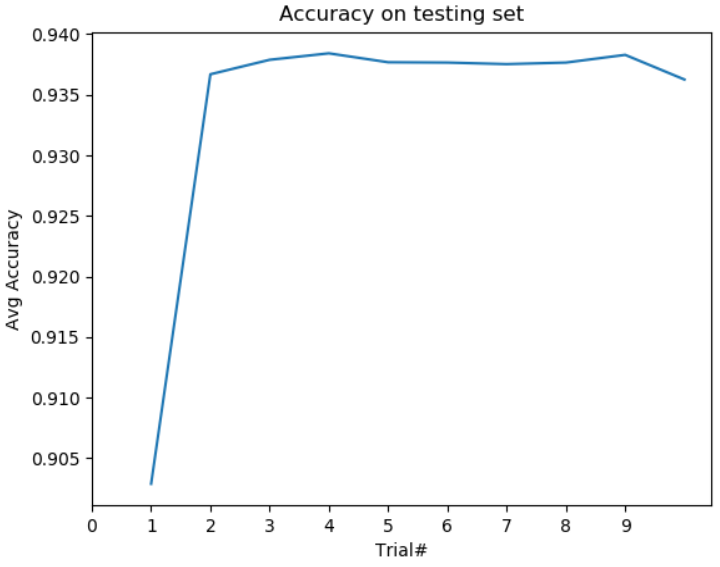




It is observed that there is a subtle difference between the different epochs. The validation accuracy on the 30 epochs model reaches 90% accuracy after only 8 epochs however the 20 epoch model terminates before reaching the 90%. On the other hand, The test set with the lower epochs seems to have a very steep increase in accuracy in the beginning however both models show little to little to no difference in shape after the 7th epoch.

Exercise 13





To identify the difference in the success rate for the different data sets, we run a repeated testing of the model 10 times on both sets and plot their final accuracies. It is observed that the success rate on the testing set seems slightly higher with mean accuracy of 93.4% and validation set not falling far behind at 92.2%

Exercise 14

To show the accuracy of the class predictions of the trained network, we produce a confusion matrix on the test set.

Predicted Labels

	1	2	3	4	5	6	7
1	**0,95**	0,00	0,03	0,00	0,00	0,01	0,01
2	0,01	**0,97**	0,01	0,00	0,00	0,00	0,01
3	0,02	0,02	**0,91**	0,00	0,01	0,03	0,02
4	0,01	0,00	0,01	**0,95**	0,00	0,01	0,02
5	0,01	0,02	0,02	0,02	**0,94**	0,00	0,00
6	0,00	0,01	0,00	0,01	0,03	**0,94**	0,01
7	0,03	0,00	0,03	0,05	0,00	0,02	**0,88**

The numbers in the matrix are normalized to the points that were classified as that class. The column of the confusion matrix is the predicted class, while the row is the actual class (true label). Therefore, if we take the second column and the third row, this shows the percentage of times that our network predicted class 3 while it should have predicted class 2. Therefore, every percentage which is not in the diagonal is the error of the predicted class.

The sum of a row has to be equal to 1. In most of the cases the confusion matrix is not 100% accurate, because of rounding errors. In this run of our network, we see that class 7 is classified the worst, with an accuracy of 88%, and class 2 is classified the best with an accuracy of 97%. Therefore, we aim for a high percentage at the diagonal.

Exercise 15

The method save_predictions outputs the prediction on the unknown data on a csv file named Group_61_classes.txt attached in the zip file

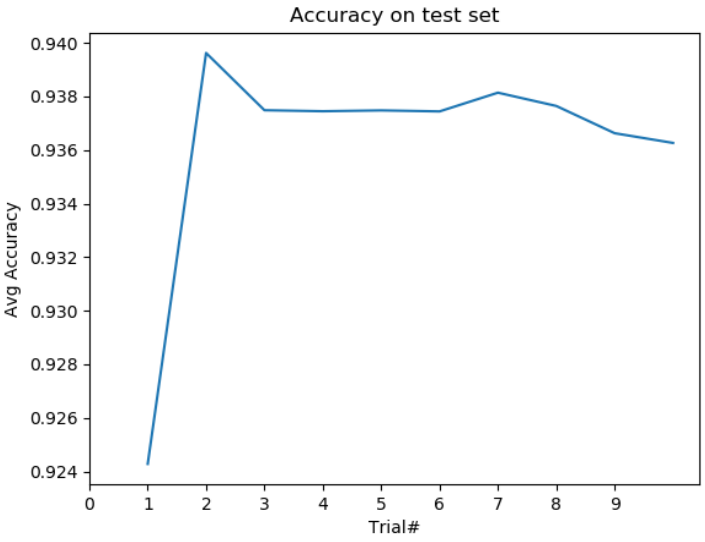
Exercise 16

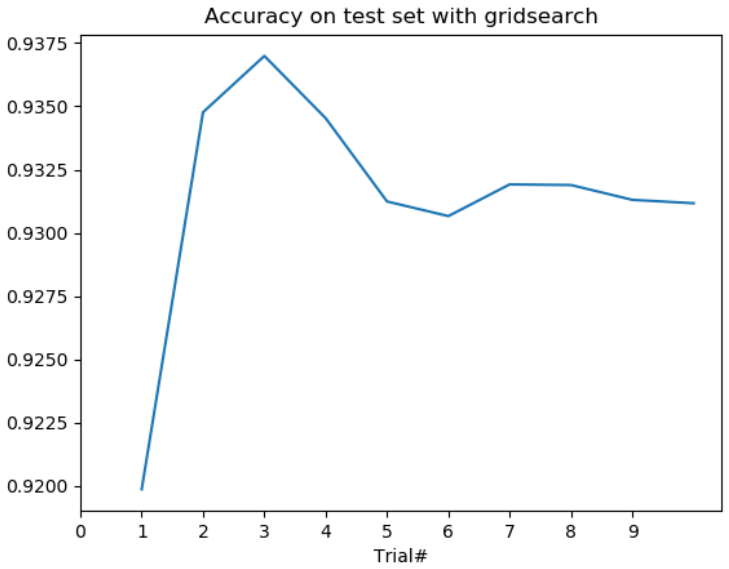
```
Grid search on grid:
-hidden_layer_sizes: [[10], [15], [17], [20], [25], [30]]
-activation: ['logistic']
-alpha: [0, 0.01]
-solver: ['adam']
-learning_rate_init: [0.005, 0.01]
-max_iter: [200]
-batch_size: ['auto']
-validation_fraction: [0.09000000000000001]
-early_stopping: [True]
-n_iter_no_change: [10]
-tol: [0.0001]

Fitting 5 folds for each of 24 candidates, totalling 120 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done  26 tasks      | elapsed:    6.1s
[Parallel(n_jobs=-1)]: Done 120 out of 120 | elapsed:   22.3s finished

Best parameters found by grid search:
-activation: logistic
-alpha: 0.01
-batch_size: auto
-early_stopping: True
-hidden_layer_sizes: [30]
-learning_rate_init: 0.01
-max_iter: 200
-n_iter_no_change: 10
-solver: adam
-tol: 0.0001
-validation_fraction: 0.09000000000000001
```

```
Resulting Training Cross-Entropy loss: 0.2123340552258078
Resulting Test Cross-Entropy loss: 0.22848740223148517

Resulting Training accuracy: 0.9316638370118846
Resulting Validation accuracy: 0.9167974882260597
Resulting Test accuracy: 0.9351145038167938

Resulting Training MSE: 0.016056884696160286
Resulting Test MSE: 0.015409745075525458
```

It is interesting to see that a lot of the parameters are similarly used as our own designed neural network. On multiple runs of the best parameters found were usually between 25-30 neurons, this is quite close to our own selected parameter of 22 neurons. The output accuracy was also similar as our own results.

Exercise 17





10 trials were run with both the models and were tested on the test set with both their final accuracies averaged.

A comparison between the two figures shows that our network which uses the stochastic gradient with momentum model had slightly less variance and maintained a higher mean accuracy of 93.6%.

The figure on the right shows that the network performs slightly worse under the optimal parameters from the grid search function which suggested the use of 30 neurons and learning rate of 0.01, this resulted in a mean accuracy of 93%. The slight difference in the expected accuracy of 94% could be due to the way we implemented the network.