NANYANG TECHNOLOGICAL UNIVERSITY

CZ4042 NEURAL NETWORKS & DEEP LEARNING

ASSIGNMENT 1

#YE HTET MYAT# U1621116B
YEHT0005@E.NTU.EDU.SG

Table of Contents

Part A: Classification Problem

Introduction

The dataset given is the Landsat satellite data which consists of multi spectral values of pixels in 3x3 neighborhoods in satellite image and the classification associated with the central pixel in each neighborhood. The objective of the problem is to be able to predict the classification with given multi spectral values.

The multi-spectral attributes are numerical in the range 0 to 255 and the classes for central pixel are from 1 to 7 where 1 represents red soil, 2 for cotton crop, 3 for grey soil, 4 for damp grey soil, 5 for soil with vegetation stubble,6 for mixture class and 7 for very damp grey soil respectively.

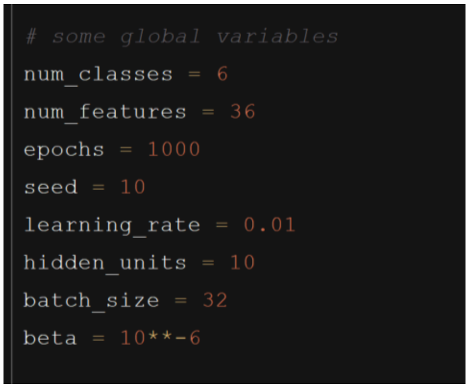
There is a total of 6,435 data entries of which 4,435 entries are used in training and the remaining 2000 are used in testing and evaluating the performance of the network designed. Since this is a classification problem involving identification of multiple classes, a softmax output layer is implemented for all the solutions to this problem. Test errors, test accuracies, training accuracies, training errors and time for training are evaluated to choose the optimal parameters such as batch size, decay rate and hidden layer number of neurons.

1. Design a feedforward neural network

Specification: One hidden perceptron layer of 10 neurons and output softmax layer, learning rate = 0.01, L2 regularization with weight decay parameter = 10^-6, scaling of input features

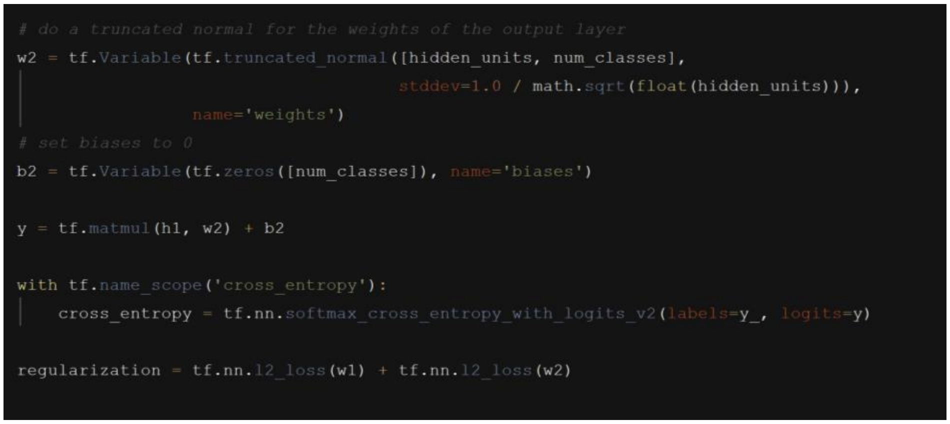
1.1 Methods

Hidden layer units =10, beta (decay rate) = 10^-6, Learning rate = 0.01



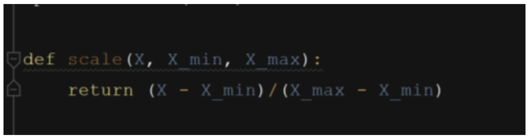
```
# some global variables
num_classes = 6
num_features = 36
epochs = 1000
seed = 10
learning_rate = 0.01
hidden_units = 10
batch_size = 32
beta = 10**-6
```

3 Layer Feedforward Neural Network

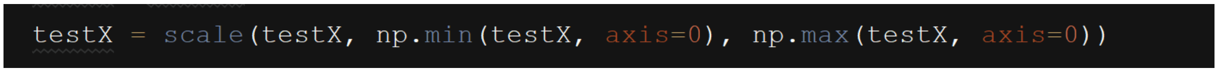


```
# do a truncated normal for the weights of the output layer
w2 = tf.Variable(tf.truncated_normal([hidden_units, num_classes],
                                     stddev=1.0 / math.sqrt(float(hidden_units))),
                 name='weights')
# set biases to 0
b2 = tf.Variable(tf.zeros([num_classes]), name='biases')

y = tf.matmul(h1, w2) + b2

with tf.name_scope('cross_entropy'):
    cross_entropy = tf.nn.softmax_cross_entropy_with_logits_v2(labels=y_, logits=y)

regularization = tf.nn.l2_loss(w1) + tf.nn.l2_loss(w2)
```

Scaling of input Features

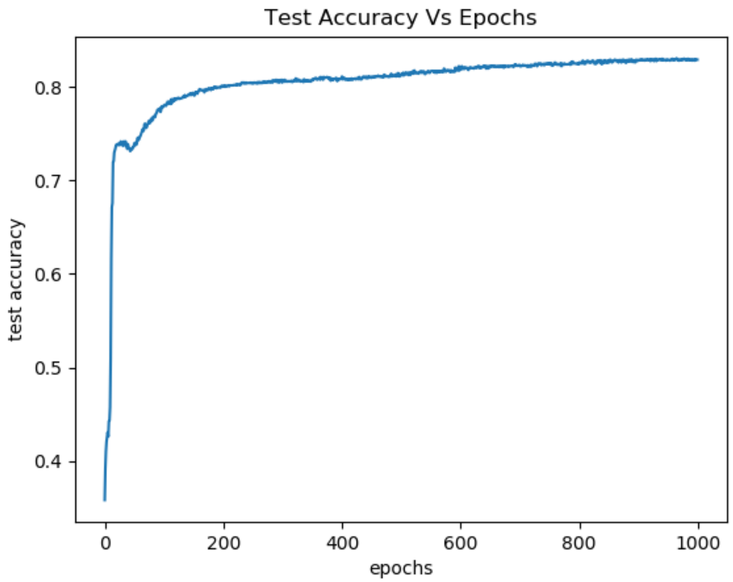


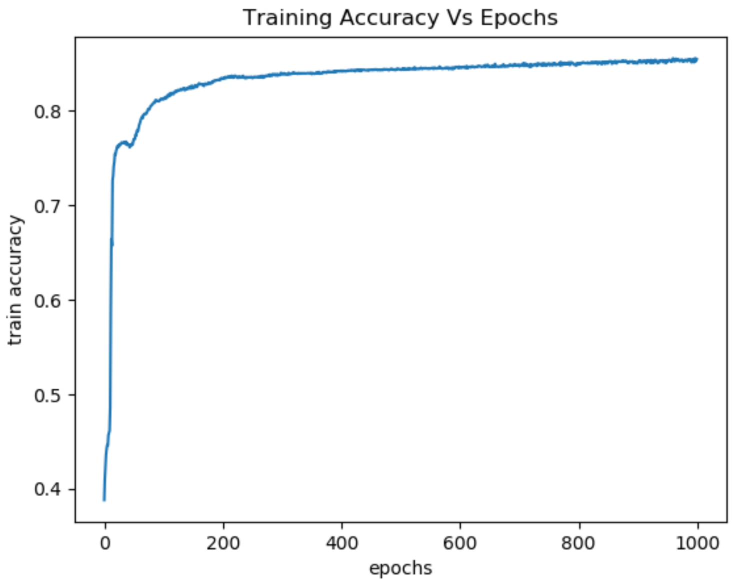
```
def scale(X, X_min, X_max):
    return (X - X_min)/(X_max - X_min)
```

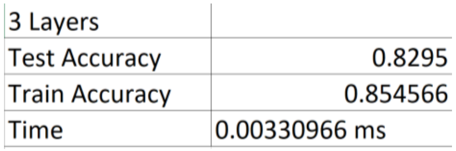


```
testX = scale(testX, np.min(testX, axis=0), np.max(testX, axis=0))
```

1.2 Experiments and Results







3 Layers	
Test Accuracy	0.8295
Train Accuracy	0.854566
Time	0.00330966 ms

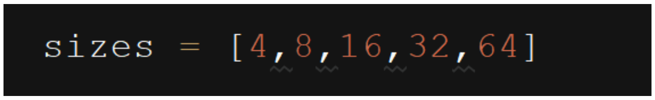
1.3 Conclusion

At convergence, the test accuracy is at 82.95% while the training accuracy is at 85.46%. The total time for training (i.e for weight and bias updating is 0.00330966 ms.

2. Find the optimal batch size

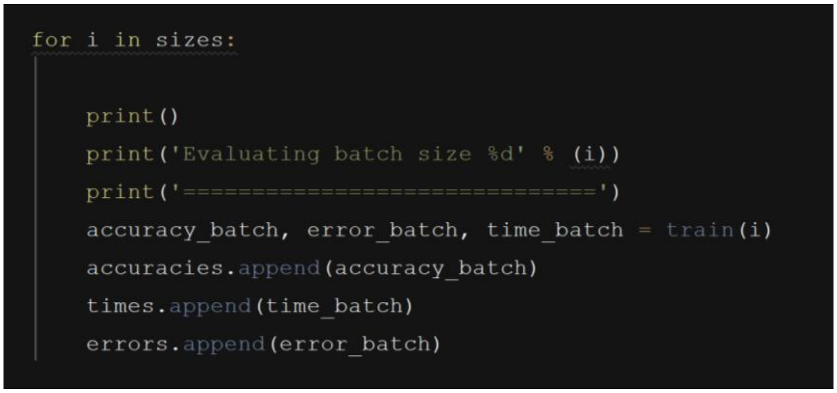
2.1 Methods

Sample space batch sizes = {4, 8, 16, 32 ,64}



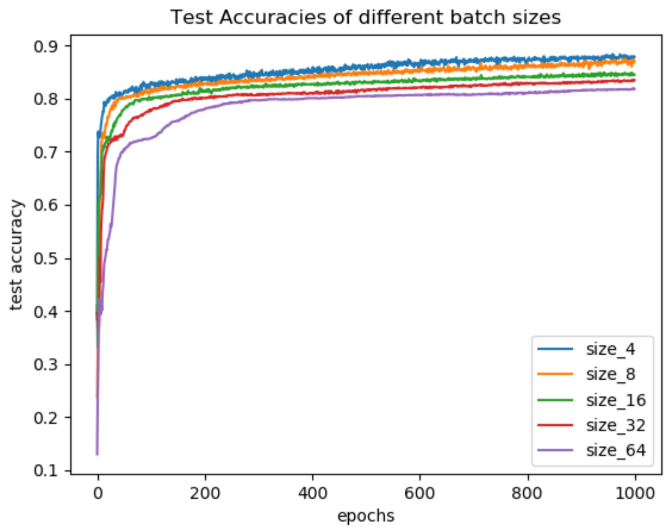
```
sizes = [4,8,16,32,64]
```

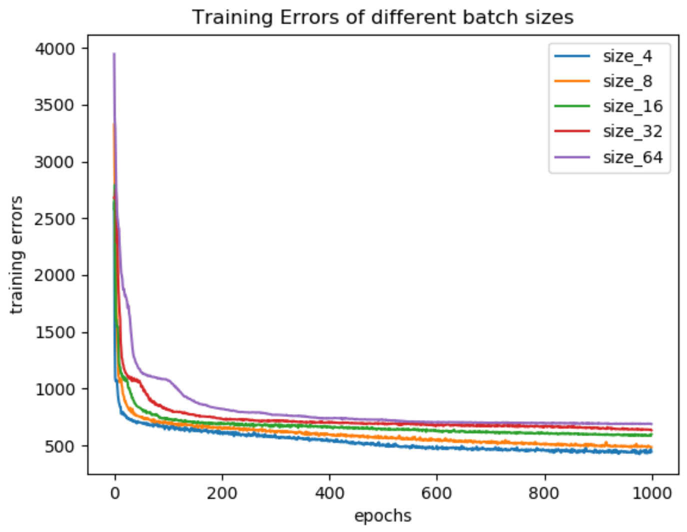
Train and Evaluate for each batch sizes in the sample space

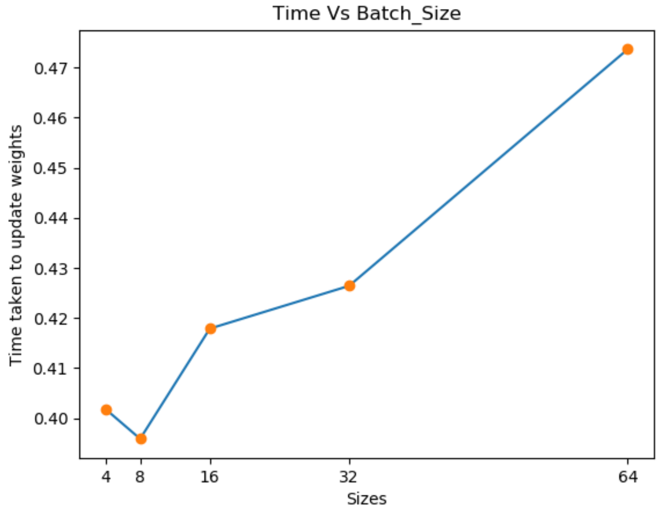


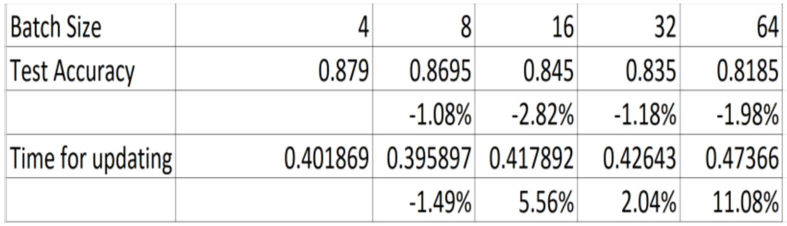
```
for i in sizes:

    print()
    print('Evaluating batch size %d' % (i))
    print('==============================')
    accuracy_batch, error_batch, time_batch = train(i)
    accuracies.append(accuracy_batch)
    times.append(time_batch)
    errors.append(error_batch)
```

2.1 Experiments and Results









Batch Size	4	8	16	32	64
Test Accuracy	0.879	0.8695	0.845	0.835	0.8185
		-1.08%	-2.82%	-1.18%	-1.98%
Time for updating	0.401869	0.395897	0.417892	0.42643	0.47366
		-1.49%	5.56%	2.04%	11.08%

2.3 Conclusion

Generally, for all the different batches, the training errors decreases over each epoch to reach to a minimal at the convergence. Likewise, the test accuracies also increase over the epochs.

The table on the 4th quadrant shows the analysis of the test accuracies at convergence for different batch sizes and time for updating each epoch.

The percentage shown is the percentage increment or decrement from the previous batch size.

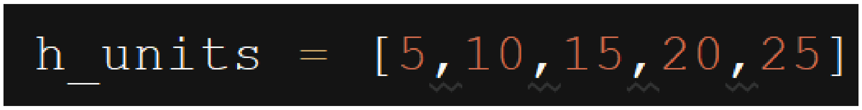
Generally, as the batch size increases, test accuracies and time to update weights and biases also increases with the exception from batch size 4 to 8. This can be attributed to the fact that as batch size increases, the computations also increase and hence an increase in time to update the parameters for each epoch.

From the results, batch size 8 is chosen as the optimal batch size since at the cost of -1.08% for test accuracy from size of batch 4 we get the best ‘time for update’. This gives us the best time for training per epoch with a close second best test accuracy.

3. Find the optimal number of hidden neurons

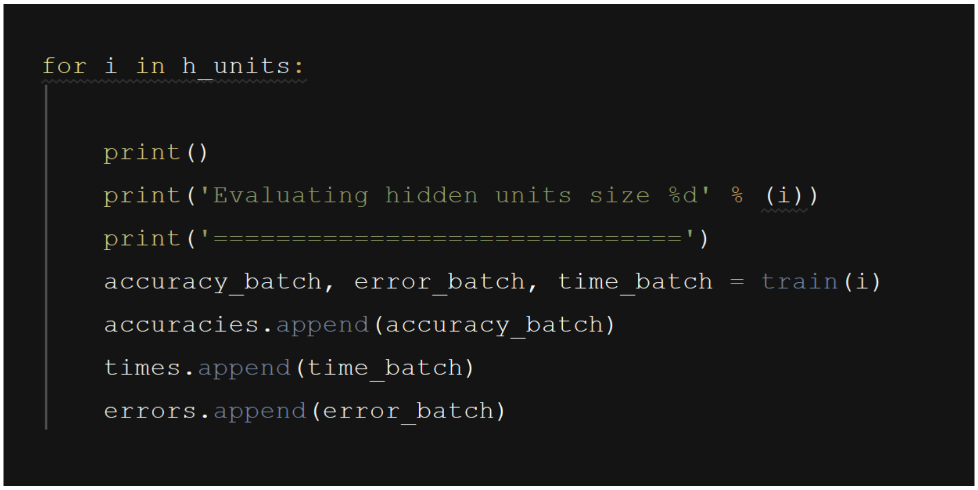
3.1 Methods

Sample space of hidden layer neurons = {5, 10, 15 ,20 , 25 }



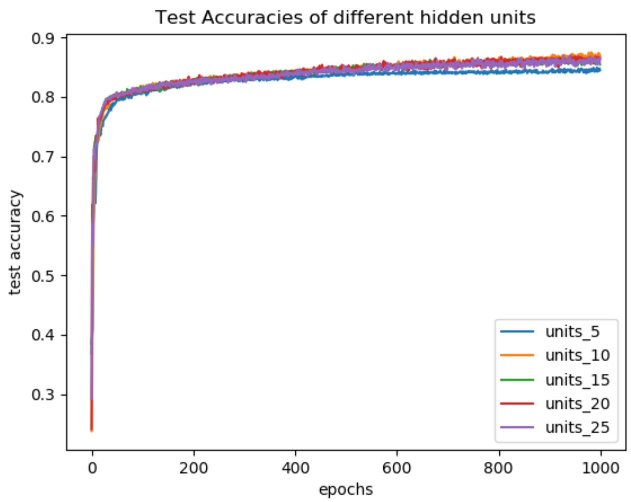
```
h_units = [5,10,15,20,25]
```

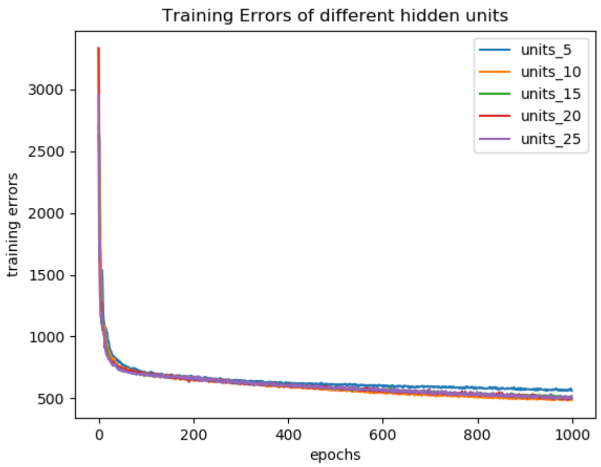
Train and evaluate for each number of hidden layer number of units

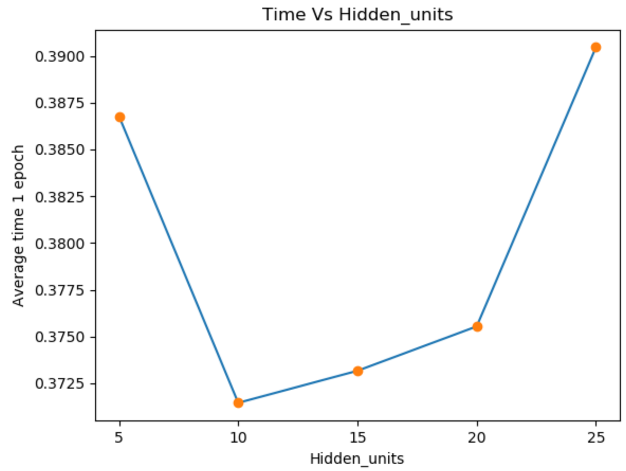


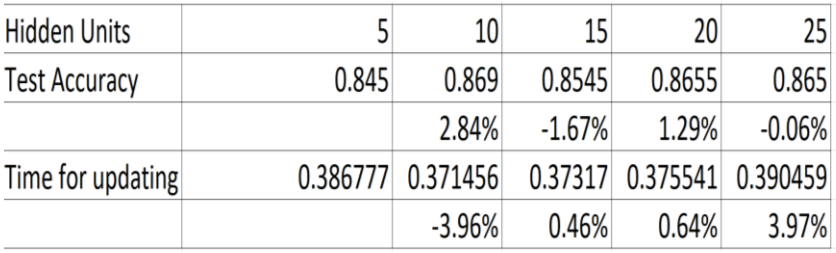
```
for i in h_units:

    print()
    print('Evaluating hidden units size %d' % (i))
    print('==============================')
    accuracy_batch, error_batch, time_batch = train(i)
    accuracies.append(accuracy_batch)
    times.append(time_batch)
    errors.append(error_batch)
```

3.2 Experiments and Results









Hidden Units	5	10	15	20	25
Test Accuracy	0.845	0.869	0.8545	0.8655	0.865
		2.84%	-1.67%	1.29%	-0.06%
Time for updating	0.386777	0.371456	0.37317	0.375541	0.390459
		-3.96%	0.46%	0.64%	3.97%

3.3 Conclusion

The experiment was conducted with the optimal batch size of 8 which was found in section 2. The general trend for the test accuracies for different hidden units is increasing with each epoch as the network learns and their convergence test accuracies are fairly close as observed from figure in the first quadrant.

Likewise, for the training errors, we see a general decrement over the epochs as the network trains and learns with close convergence training errors.

From the statistics in the table from quadrant 4 there is an increment in test accuracy from 5 hidden units to 10 which tops the rest of the hidden layer neuron sizes in terms of test accuracy. This could be attributed to the fact that as the number of hidden layer neurons increases, the number of parameters in the network increases while the network tries to remember the training patterns which eventually leads to overfitting. Overfitting in turn will result in lower test accuracies.

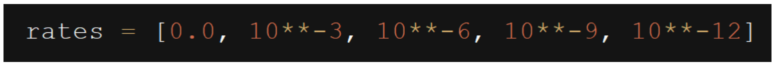
There is also a decrement in time to update parameters from 5 to 10 hidden layer neurons after which the time increments again. As number of hidden layer neurons increases, the weight parameters also increases resulting in more computation required to update them hence leading to increasing time for update.

Although 10 hidden layer neurons have similar test accuracies with 15, 20 and 25, it gives the best time for update per epoch and hence it is chosen as the optimal number of neurons for the hidden layer. It is also observed that the optimal number of hidden layer neurons is between the size of the input layer and the output layer size.

4. Find the optimal decay parameter

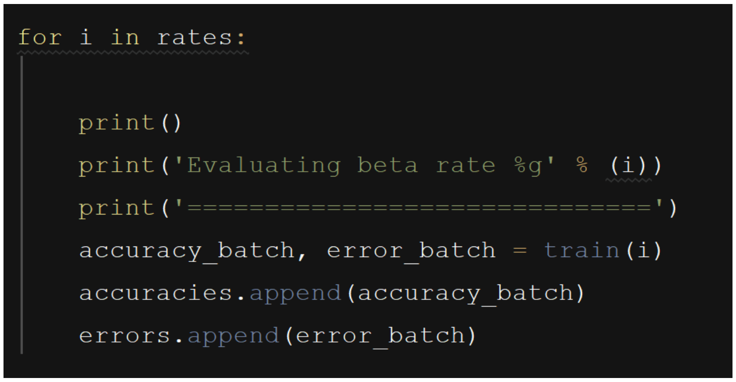
4.1 Methods

Sample space of decay parameters = {0.0, 10**-3, 10**-6, 10**-9, 10**-12}



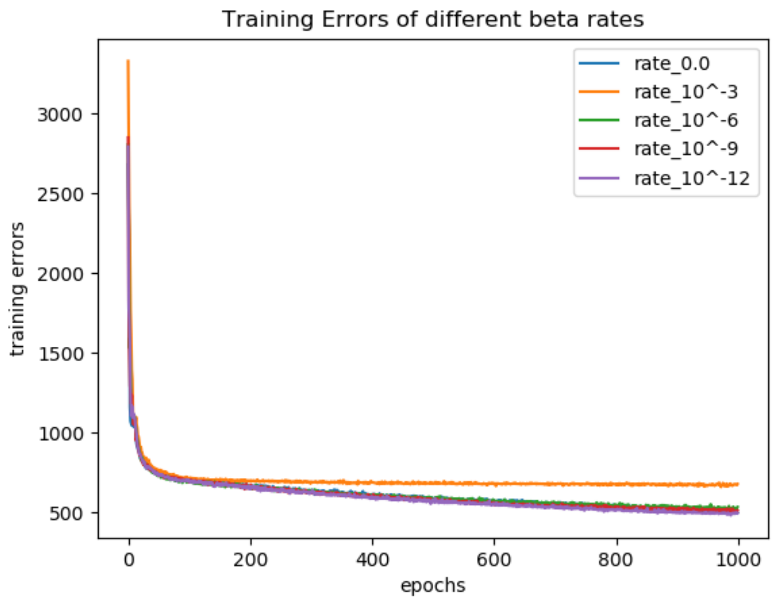
```
rates = [0.0, 10**-3, 10**-6, 10**-9, 10**-12]
```

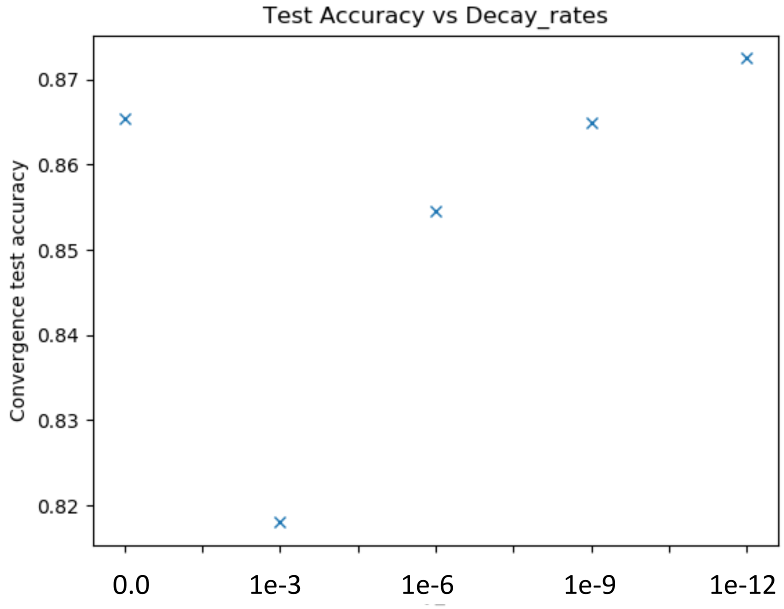
Train and evaluate for each beta rate (decay parameters)

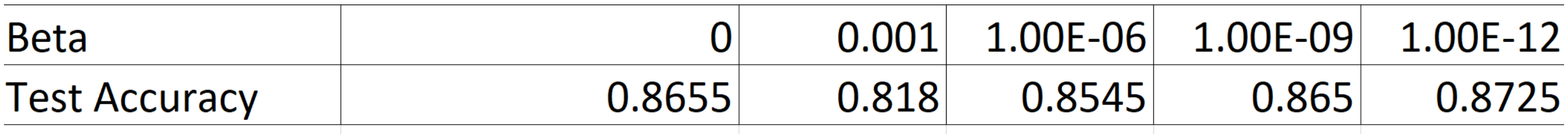


```
for i in rates:

    print()
    print('Evaluating beta rate %g' % (i))
    print('==============================')
    accuracy_batch, error_batch = train(i)
    accuracies.append(accuracy_batch)
    errors.append(error_batch)
```

4.2 Experiments and Results







Beta	0	0.001	1.00E-06	1.00E-09	1.00E-12
Test Accuracy	0.8655	0.818	0.8545	0.865	0.8725

4.3 Conclusion

The experiment was conducted with the optimal batch size of 8 and optimal hidden layer neurons of 10 found in section 2 and 3 respectively. Generally, the training errors decreases over the training epochs as the network trains and learns.

Inferring from the test accuracies at convergence (table in 3rd quadrant) 1e-12 gives the best test accuracy amongst all the beta rates. Though there was a decrement in test accuracy from 0.0 to 1e-3, it increases again until 1e-12.

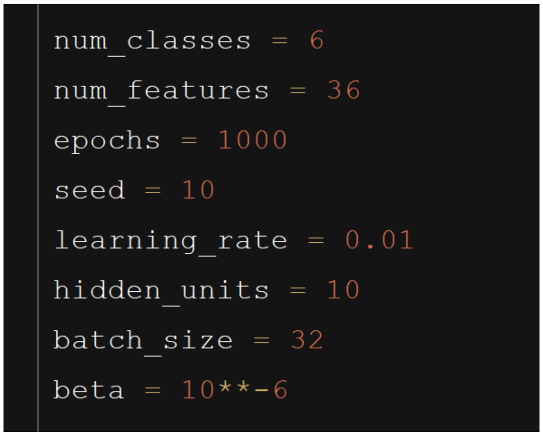
Weights tends to attain large values to minimize training error during overfitting. The purpose of the decay parameter is to penalize large weights and hence reduce overfitting.

Therefore, from the statistics obtained above, the decay parameter of 1e-12 is chosen to be the optimal decay parameter since it gives the best test accuracy amongst all.

5. Design a 4-Layer network

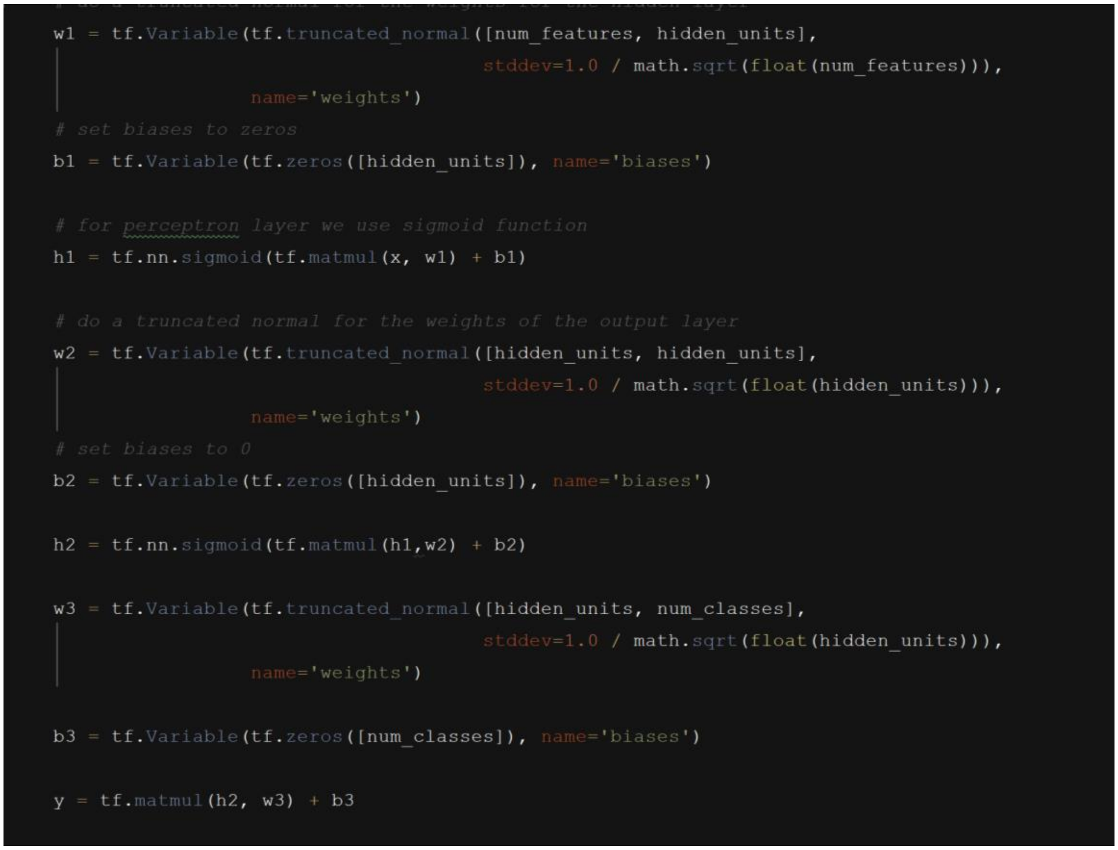
5.1 Methods

Specifications: Decay parameters of 10^-6, batch size 32



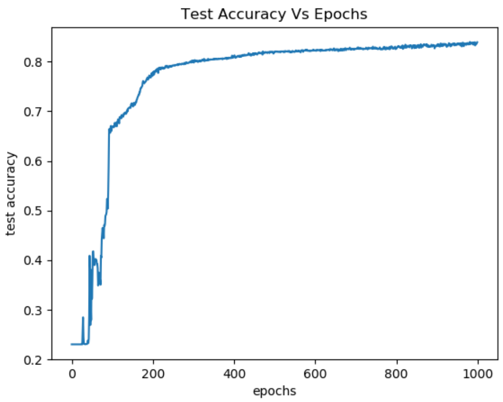
```
num_classes = 6
num_features = 36
epochs = 1000
seed = 10
learning_rate = 0.01
hidden_units = 10
batch_size = 32
beta = 10**-6
```

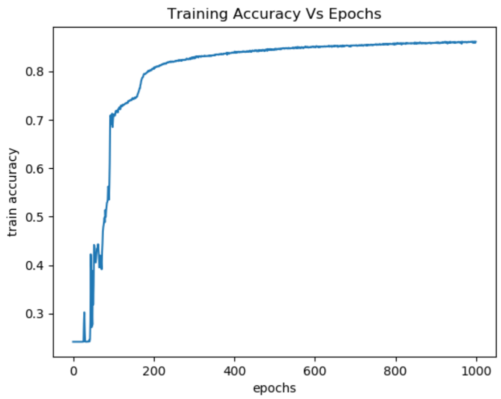
2 hidden layers of 10 perceptron each

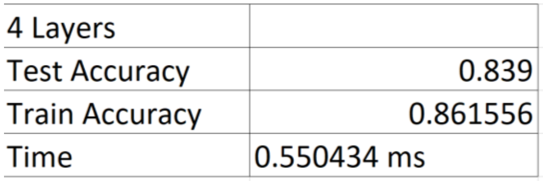


```
# do a truncated normal for the weights for the hidden layer
w1 = tf.Variable(tf.truncated_normal([num_features, hidden_units],
                                     stddev=1.0 / math.sqrt(float(num_features))),
                 name='weights')
# set biases to zeros
b1 = tf.Variable(tf.zeros([hidden_units]), name='biases')

# for perceptron layer we use sigmoid function
h1 = tf.nn.sigmoid(tf.matmul(x, w1) + b1)

# do a truncated normal for the weights of the output layer
w2 = tf.Variable(tf.truncated_normal([hidden_units, hidden_units],
                                     stddev=1.0 / math.sqrt(float(hidden_units))),
                 name='weights')
# set biases to 0
b2 = tf.Variable(tf.zeros([hidden_units]), name='biases')

h2 = tf.nn.sigmoid(tf.matmul(h1,w2) + b2)

w3 = tf.Variable(tf.truncated_normal([hidden_units, num_classes],
                                     stddev=1.0 / math.sqrt(float(hidden_units))),
                 name='weights')

b3 = tf.Variable(tf.zeros([num_classes]), name='biases')

y = tf.matmul(h2, w3) + b3
```

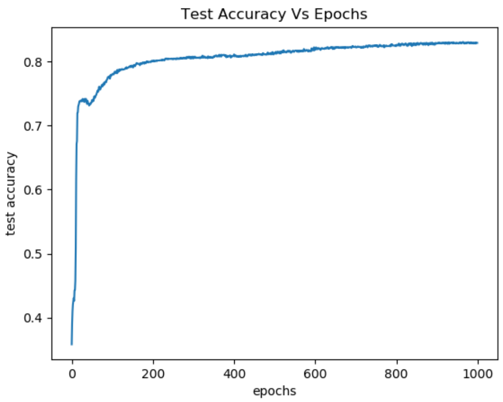
5.2 Experiment and Results

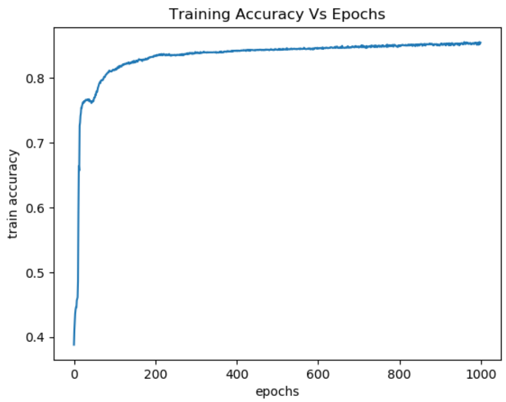

Test Accuracy Vs Epochs

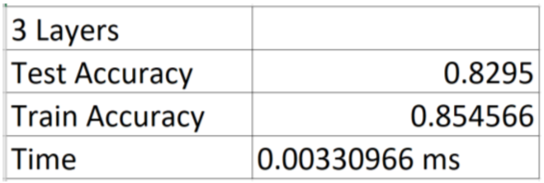

Training Accuracy Vs Epochs



4 Layers	
Test Accuracy	0.839
Train Accuracy	0.861556
Time	0.550434 ms


Test Accuracy Vs Epochs


Training Accuracy Vs Epochs



3 Layers	
Test Accuracy	0.8295
Train Accuracy	0.854566
Time	0.00330966 ms

5.3 Conclusion

The graphs and statistics in the first row of the table shows the outcomes of 4-Layer network training and the row below shows the performances of the 3-Layer network with the same specifications with one less layer of hidden neurons.

It can be observed that the for the 3-Layer network learns faster than the 4-Layer network as interpreted from the sudden steep increase in test accuracy from the first 0-100 epochs. It is also evident that the 4-Layer network’s learning is less stable in the first 0-200 epochs with the fluctuations.

However, at convergence, 4-Layer network can achieve better training accuracy and test accuracies than the 3-Layer network: 86.16% and 83.9% respectively as compared to 82.95% and 85.46%.

It is also worthwhile to take note that the time taken to update parameters per epoch for 4-layer is higher than 3-layers by (0.55 / 0.0033) = 166 folds though it is too short of a time for humans to notice since it is in milli seconds. Therefore, it can be concluded that for this experiment, a 3-layer network would suffice since the test and training accuracies are close to that of 4-layer’s with high advantage in terms of time complexity especially when training large data sets.

Part B: Regression Problem

Introduction

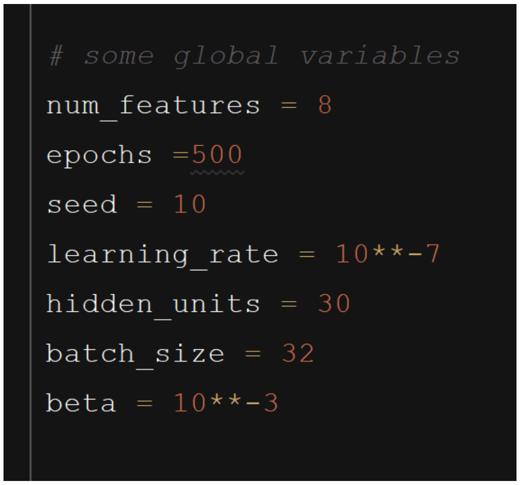
The dataset given has information of the block groups in California from the 1990 Cens us which includes 1425.5 individuals in a close knitted area. The main objective of solving this problem is to train the artificial neural network to be able to predict housing prices based on 8 input features: median house value, median income, housing median, age, total rooms, total bedrooms, population, households and latitude and longitude. The dependent variable is ln(median house value). There is a total of 20,640 data entries of which 70% of the data is used as training data for the network to train upon and 30% of the data is used as the unseen data for the network's performance to be tested upon.

From the training data a further 5-fold cross validation is done to evaluate the validation and cross validation errors. Each fold there's 4 parts used for training and 1 part used for testing which ensures that all the entries are eventually used for both training and testing. Performance is then evaluated based on cross validation errors as well as test errors for selection of parameters such as optimal learning rate and optimal hidden layer number of neurons. Since this problem requires us to predict continuous outcomes (median house value), a linear output layer is implemented for the design of the neural network.

1. Design a 3-layer feedforward neural network containing

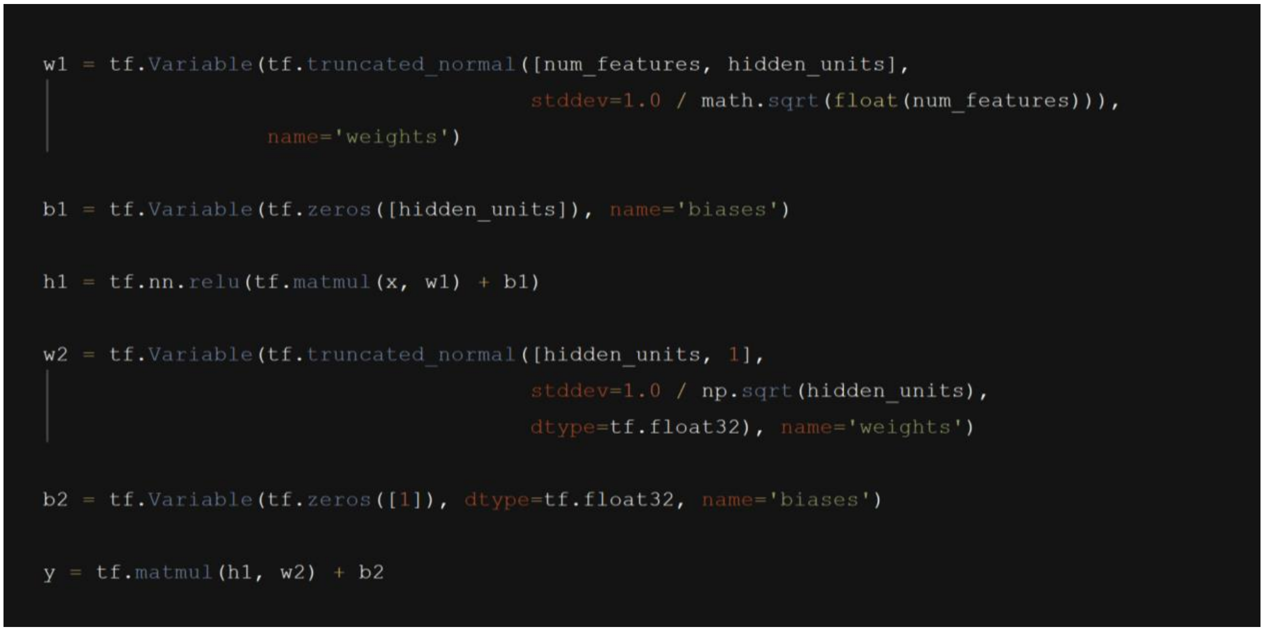
1.1 Method

Specifications: Learning rate = 10^-7 | weight decay parameter = 10^-3 | Batch size = 32 | Hidden layer neurons = 30 units | Epochs = 500



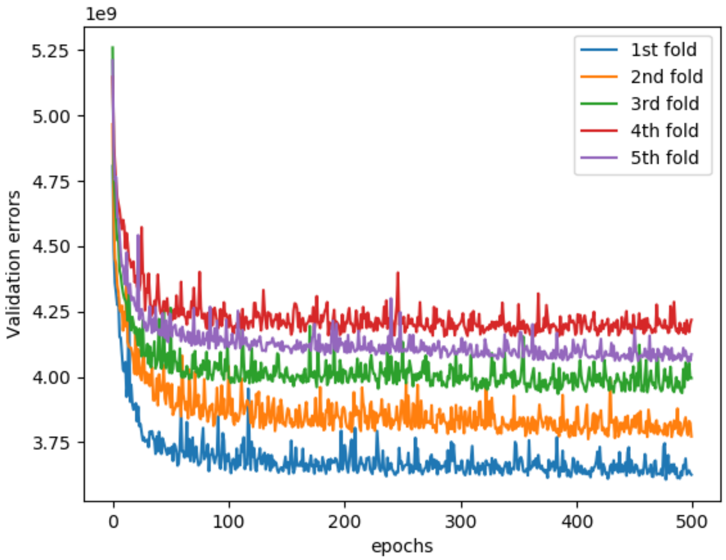
```
# some global variables
num_features = 8
epochs =500
seed = 10
learning_rate = 10**-7
hidden_units = 30
batch_size = 32
beta = 10**-3
```

Input Layer | A hidden-layer having ReLu Activation Function | Linear Output layer



```
w1 = tf.Variable(tf.truncated_normal([num_features, hidden_units],
                                      stddev=1.0 / math.sqrt(float(num_features))),
                 name='weights')

b1 = tf.Variable(tf.zeros([hidden_units]), name='biases')

h1 = tf.nn.relu(tf.matmul(x, w1) + b1)

w2 = tf.Variable(tf.truncated_normal([hidden_units, 1],
                                      stddev=1.0 / np.sqrt(hidden_units),
                                      dtype=tf.float32), name='weights')

b2 = tf.Variable(tf.zeros([1]), dtype=tf.float32, name='biases')

y = tf.matmul(h1, w2) + b2
```

1.2 Experiments and Results



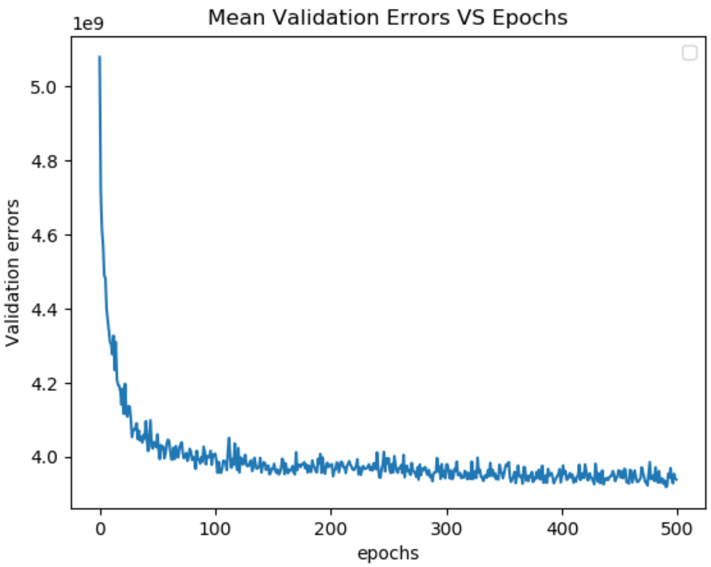
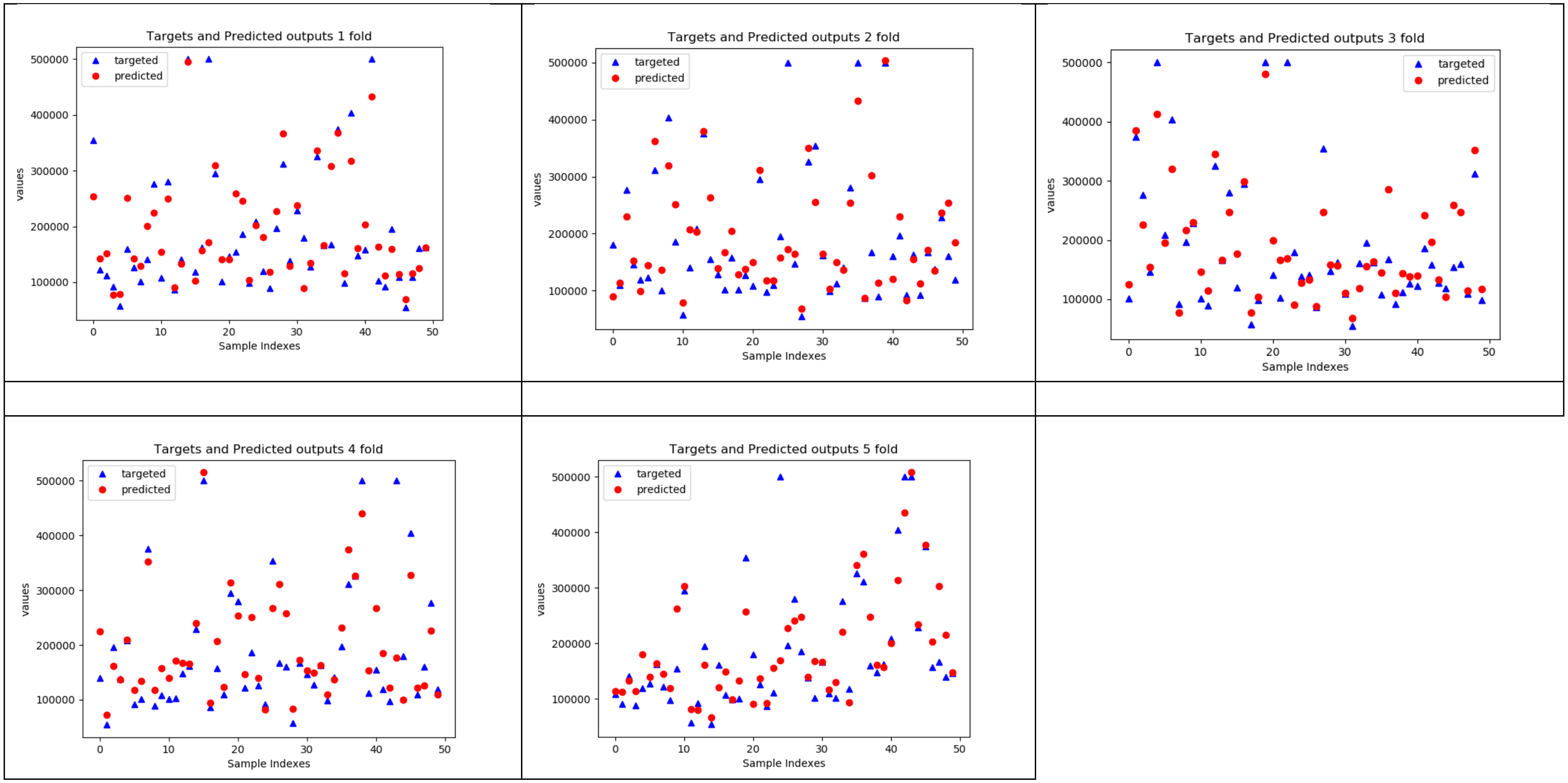


Figure on the left shows individual Validation Errors Against Epochs for each fold. Evidently, for each fold of datasets the general trend is a downward slopping curve indicating reduction in validation errors. However, there are different validation errors at convergence for each Fold with 1st Fold achieving the best (lowest) validation error at convergence and 4th Fold achieving the worst (highest) validation errors at convergence.

Figure on the right shows the mean validation errors across all the folds against epochs.



Targets and Predicted outputs for each Fold across 1 to 5.

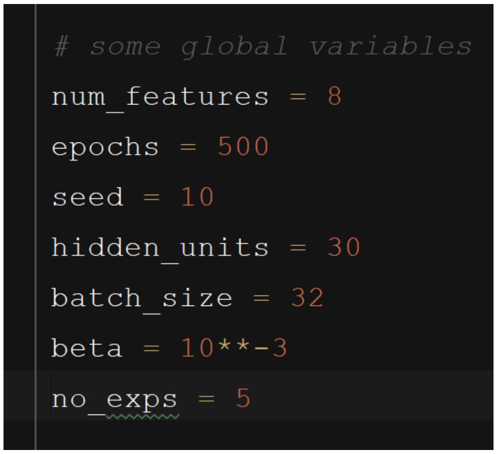
1.4 Conclusion

Generally, the trend is a downward slopping curve which indicates a reduction in validation errors as the network trains over the range of epochs.

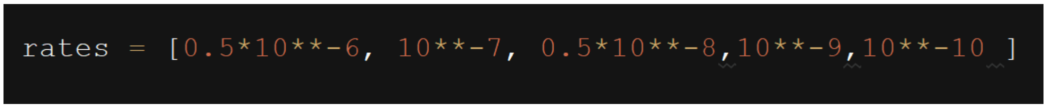
2. Find the optimal learning rate for 3 layer network

2.1 Method

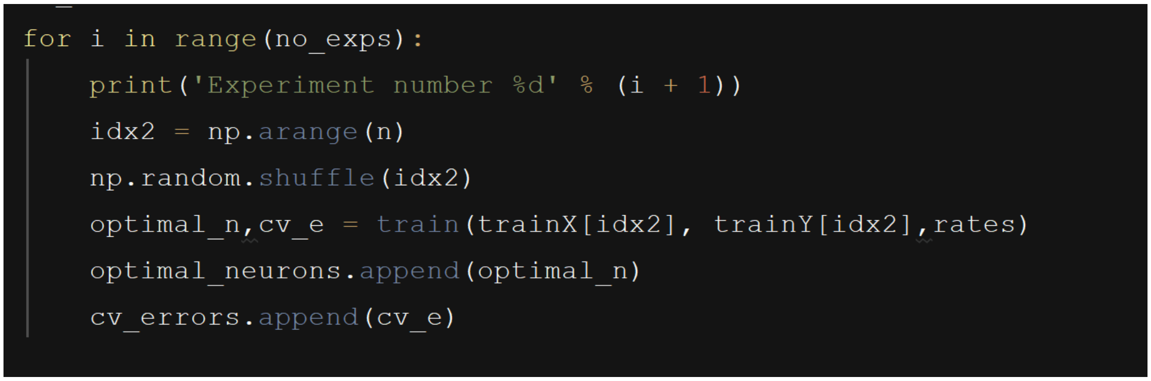
The experiment was done by initially computing the individual validation errors for each fold across 1 to 5 which runs for 500 epochs. Afterwards, taking a mean across these validation errors gives a cross validation errors across all folds for each learning rate. The experiment was then repeated 5 times to get the mode of the optimum learning rate that achieves the lowest cross validation errors across 5 experiments.



```
# some global variables
num_features = 8
epochs = 500
seed = 10
hidden_units = 30
batch_size = 32
beta = 10**-3
no_exps = 5
```

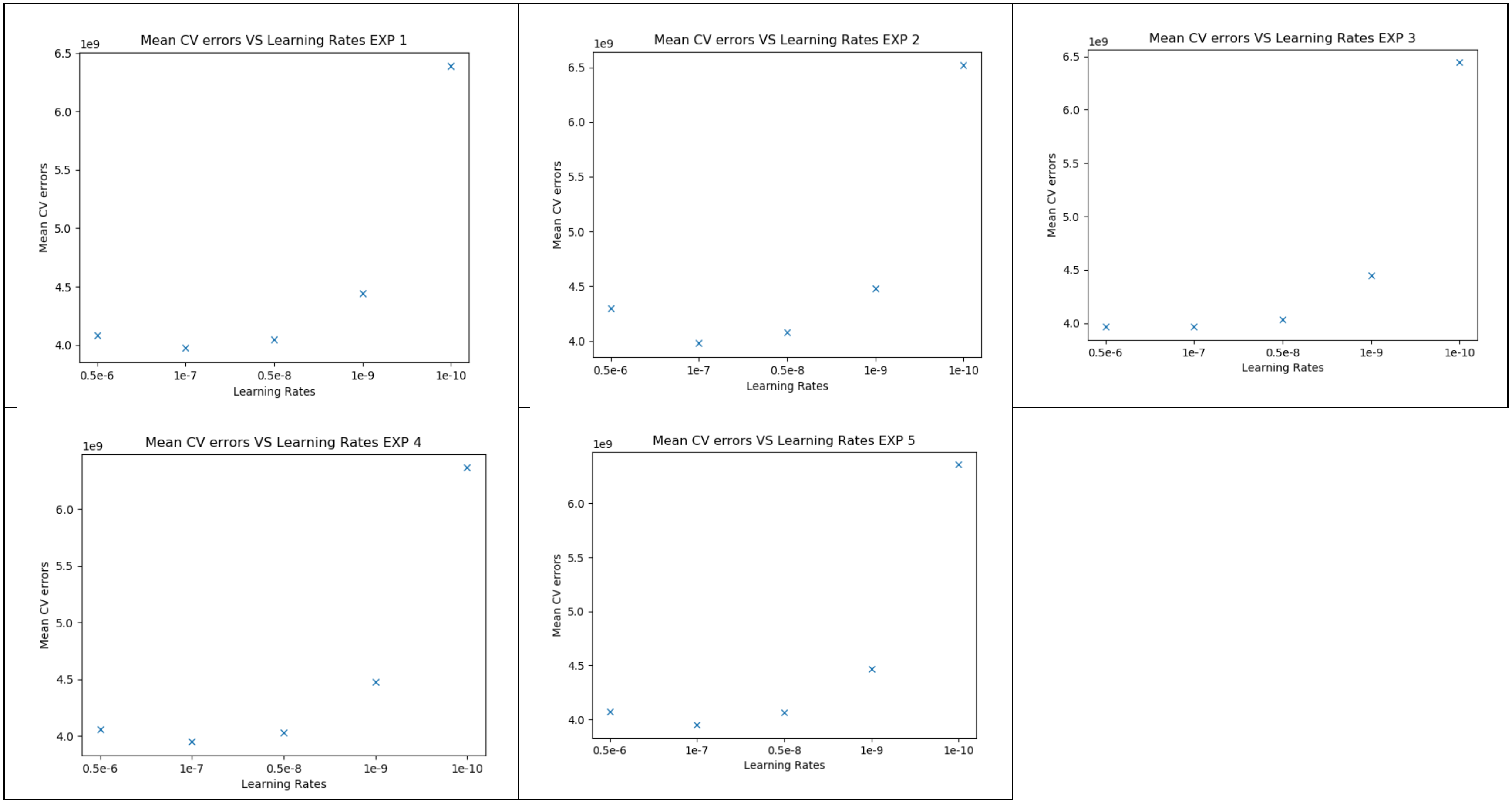


```
rates = [0.5*10**-6, 10**-7, 0.5*10**-8,10**-9,10**-10 ]
```



```
for i in range(no_exps):
    print('Experiment number %d' % (i + 1))
    idx2 = np.arange(n)
    np.random.shuffle(idx2)
    optimal_n,cv_e = train(trainX[idx2], trainY[idx2],rates)
    optimal_neurons.append(optimal_n)
    cv_errors.append(cv_e)
```

2.1 Experiments and Results



Mean CV errors VS Learning Rates EXP 1

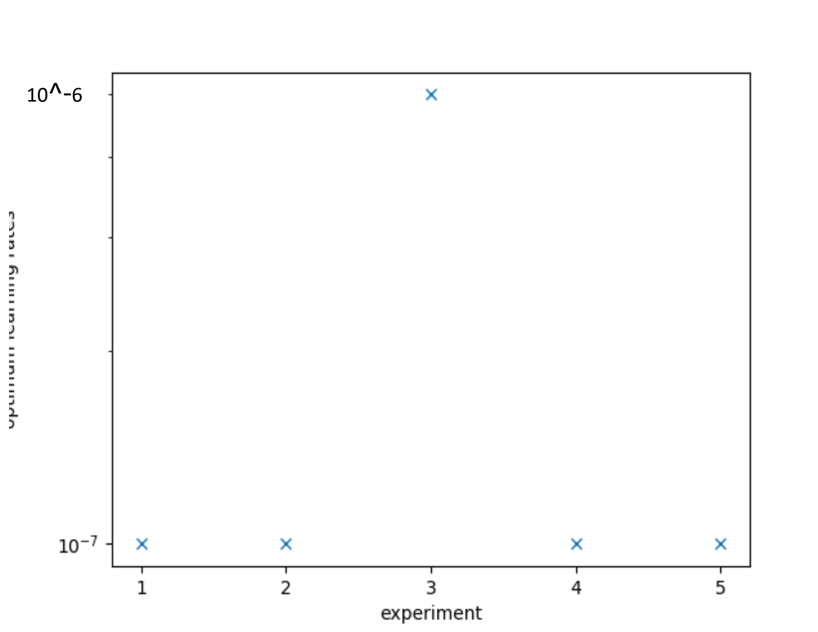
Mean CV errors VS Learning Rates EXP 2

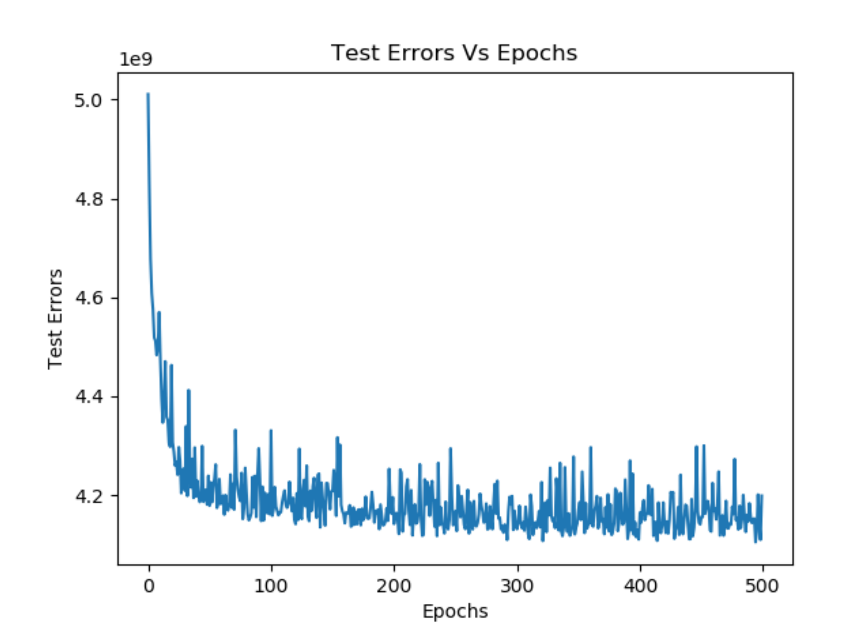
Mean CV errors VS Learning Rates EXP 3

Mean CV errors VS Learning Rates EXP 4

Mean CV errors VS Learning Rates EXP 5

Above table shows the graphs of mean cross validation errors against each learning rate for different experiments conducted.





From the experiments conducted the optimal learning rate that is most frequently concluded is 10^-7 and the graph on the left-hand side conveys that.

The graph on the right shows the test errors against epochs for the optimal learning rate of 10^-7. A general downwards slopping trends indicates that the training error decreases as the network trains more epochs.

2.3 Conclusions

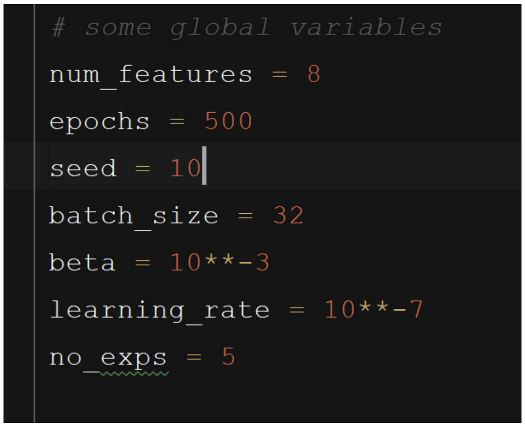
From the mean cross validation errors against learning rates graphs, it can be concluded that generally the cross-validation errors increase after

1e-7 learning rate after it decreases from 1e-6. Hence , having learning rates could not be good as it could result in undesired test errors although it may give more stable learning.

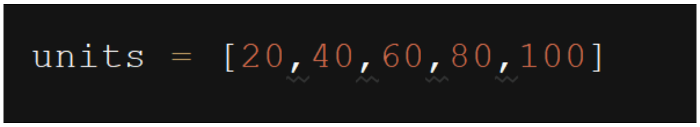
3. Find the optimal number of hidden neurons for the 3-layer network

3.1 Methods

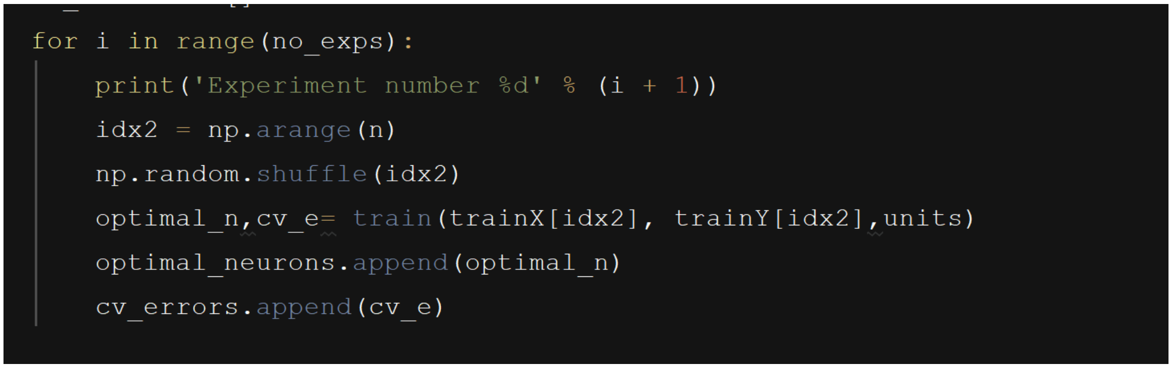
The experiment was done by initially computing the individual validation errors for each fold across 1 to 5 which runs for 500 epochs. Afterwards, taking a mean across these validation errors gives a cross validation errors across all folds for each number of hidden neurons. The experiment was then repeated 5 times to get the mode of the optimum learning rate that achieves the lowest cross validation errors across 5 experiments. All the experiment's learning rates are optimal learning rate found in section 2 of 1e-7.



```
# some global variables
num_features = 8
epochs = 500
seed = 10
batch_size = 32
beta = 10**-3
learning_rate = 10**-7
no_exps = 5
```

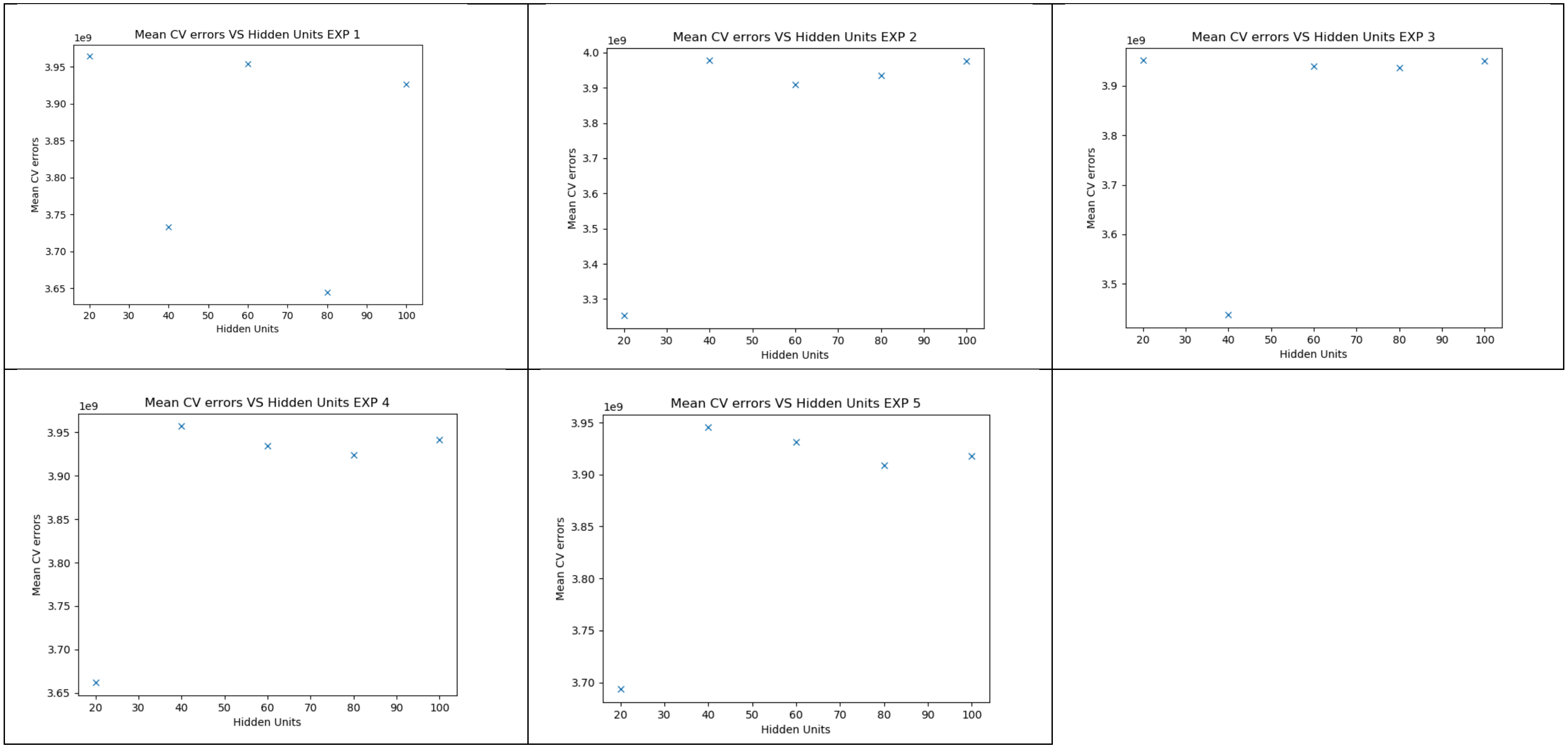


```
units = [20,40,60,80,100]
```

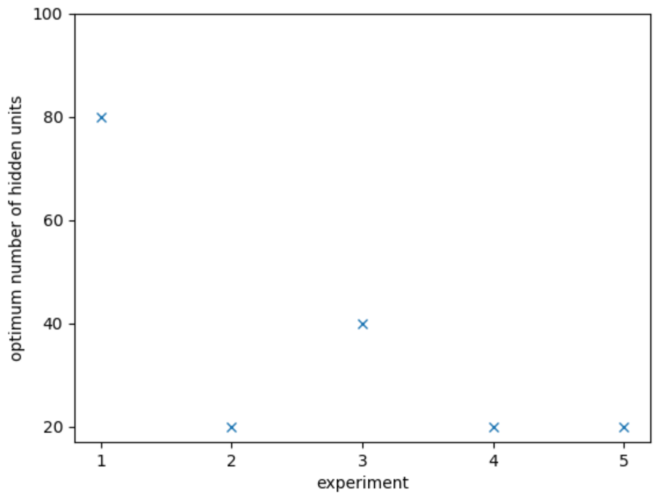


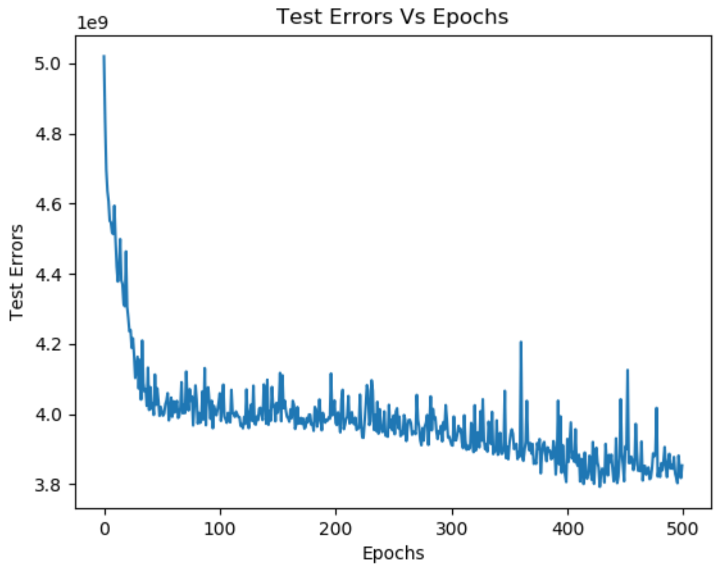
```
for i in range(no_exps):
    print('Experiment number %d' % (i + 1))
    idx2 = np.arange(n)
    np.random.shuffle(idx2)
    optimal_n,cv_e= train(trainX[idx2], trainY[idx2],units)
    optimal_neurons.append(optimal_n)
    cv_errors.append(cv_e)
```

3.2 Experiments and Results



Above table shows the graphs of mean cross validation errors against each number of hidden neurons for different experiments conducted.





From the experiments conducted, the optimal number of hidden neurons that is most frequently concluded is 20 and the graph on the left-hand side conveys such information. This is in line with the finding from part A classification problem where we identified that having a low number of neurons in hidden layer would mean reduction of overfitting. The graph on the right shows the test errors against epochs for the optimal hidden neurons of 20. A general downwards slopping trends indicates that the training error decreases as the network trains more epochs.

3.3 Conclusions

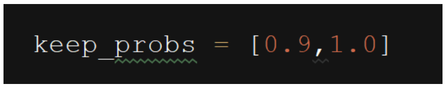
From the mean cross validation errors against different sizes of hidden layer neurons, it is observable that as the number hidden neurons increases, the validation errors generally become larger as well. This is due to overfitting which resulting from the network attempting to remember the training patterns with large parameters. The findings here also confirms the finding in partA, question 3 for finding optimal hidden layer neurons in   classification problem.

4. Design a four-layer network and a five-layer neural network

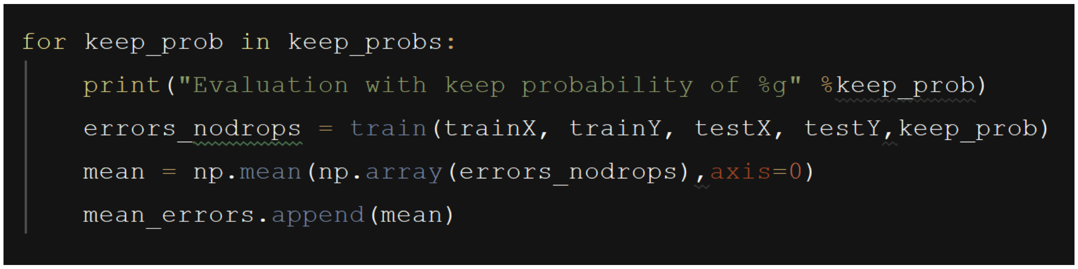
4.1 Methods

The networks 4-Layer and 5-Layer designed in this section have first layer of hidden neurons having the optimal number of hidden neurons found in section 3.

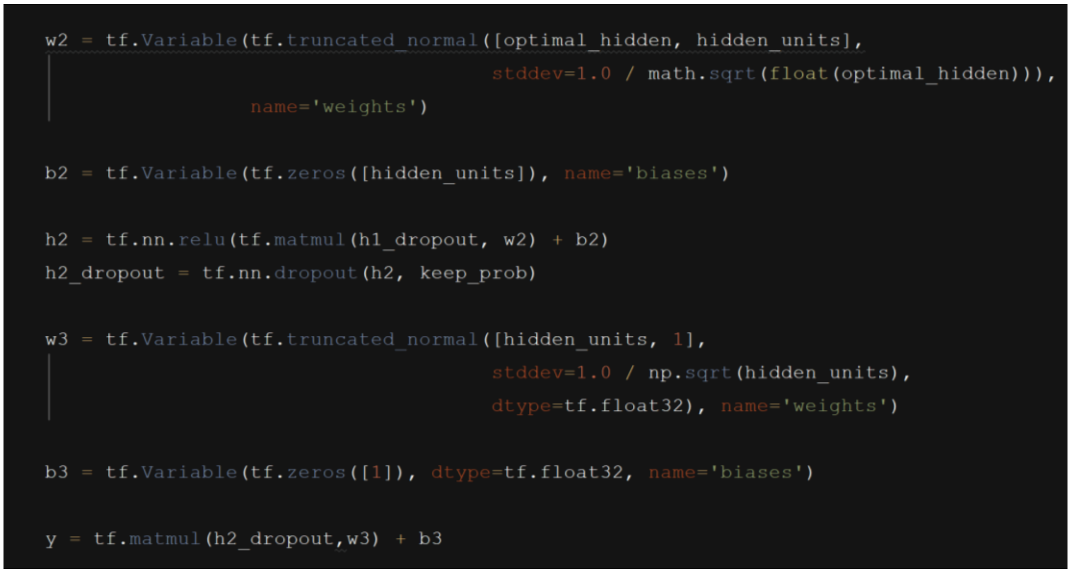
For each of the 4-Layer and 5-layer networks, 2 different training were conducted one using dropouts which corresponds to a keep_probs of 1.0 and one with introduced dropouts keep_probs 0.9. The test errors were then recorded trainings with and without dropouts for analysis. The 3-Layer network designed in section 1 is also modified such that its tests errors with and without dropouts can be identified for comparison with the 4-Layer and 5-Layer networks.



```
keep_probs = [0.9,1.0]
```

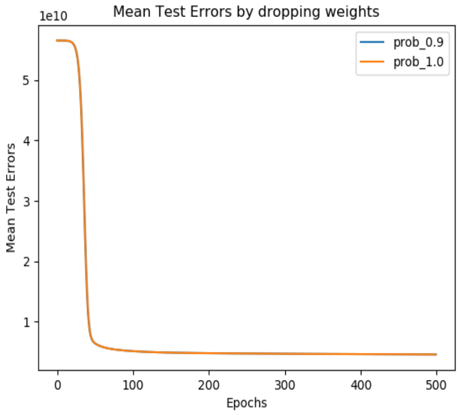


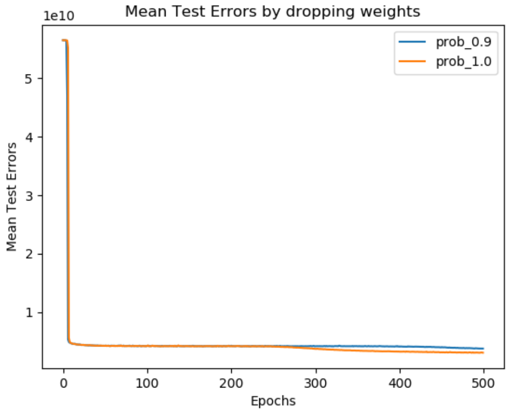
```
for keep_prob in keep_probs:
    print("Evaluation with keep probability of %g" %keep_prob)
    errors_nodrops = train(trainX, trainY, testX, testY,keep_prob)
    mean = np.mean(np.array(errors_nodrops),axis=0)
    mean_errors.append(mean)
```

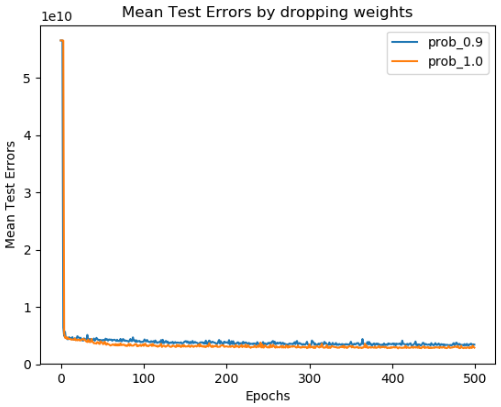


```
w2 = tf.Variable(tf.truncated_normal([optimal_hidden, hidden_units],
                                    stddev=1.0 / math.sqrt(float(optimal_hidden))),
                 name='weights')

b2 = tf.Variable(tf.zeros([hidden_units]), name='biases')

h2 = tf.nn.relu(tf.matmul(h1_dropout, w2) + b2)
h2_dropout = tf.nn.dropout(h2, keep_prob)

w3 = tf.Variable(tf.truncated_normal([hidden_units, 1],
                                    stddev=1.0 / np.sqrt(hidden_units),
                                    dtype=tf.float32), name='weights')

b3 = tf.Variable(tf.zeros([1]), dtype=tf.float32, name='biases')

y = tf.matmul(h2_dropout,w3) + b3
```

4.2 Experiments and Results







4.3 Conclusions

From the above data it can be observed that networks with more than 1 hidden neuron layers (4 & 5-Layers) learns faster than the network with only one hidden layer (3-Layer) as the curve drops very steeply. As for 3-Layer network there is no observable difference in mean test errors against epochs for both the dropouts and without dropouts. It could be attributed to the fact that the number of hidden neurons is too little for the effect to be noticeable. However, as for the 4-Layer and 5-Layer networks, we can see a slight difference in the mean test errors towards convergence. It can also be noted that 5-Layer network's learning is less stable than the other 2 as it is evident that there are more fluctuations in mean test errors even towards convergence.

References

https://archive.ics.uci.edu/ml/datasets/Statlog+(Landsat+Satellite)

http://www.dcc.fc.up.pt/~ltorgo/Regression/cal_housing.html

https://stats.stackexchange.com/questions/181/how-to-choose-the-number-of-hidden-layers-and-nodes-in-a-feedforward-neural-netw