Problem Statement:

We are trying to improve the performance of our model which use the Spamsbase data set.

Proposed Solution:

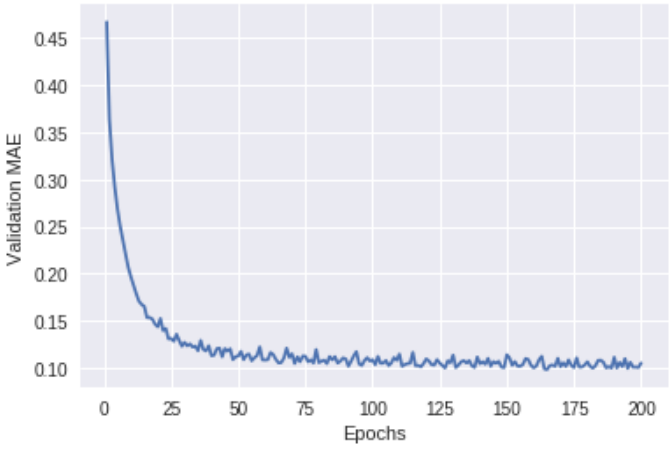
Loading Data:

1. Dataset -http://archive.ics.uci.edu/ml/machine-learning-databases/communities/communities.data
2. I took some Data Cleaning steps on our data set and that were needed to train our model with higher percentage of accuracy.
   a. Column "Community Name" has been deleted because of incorrect data. Column type is string we have its data in another column Community Number.
   b. Multiple columns have 1675 missing values. These 24 columns have been dropped since we do not have enough available information to make reasonable imputations.
   c. In 29th Column we have one blank value. I put the mean into it.
   d. I devide the data into train and testing sets using SciKitLearn'strain_test_split method, with an 80-20 ratio.
   e. Final Dataset shapes: x_train: (1595, 158) y_train: (1595, 158) x_test: (399, 158) y_test: (399,)

Implementation details:

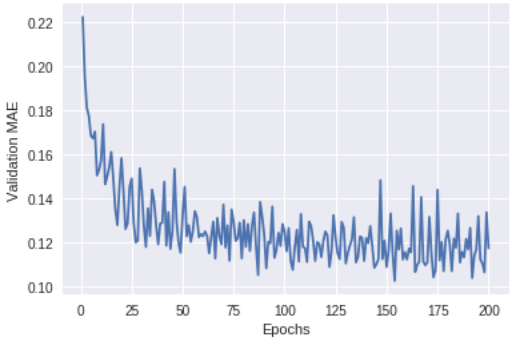
Base Model

1. 4-fold cross validation is being used to train the Network. • The base model have 1 layer of 8 Neurons at training stage.
2. No. of Layers: 1, No. of Neurons: 8, Loss: MSE, Optimizer: RMSProp, Metrics: ['mae'] Activation: Relu for the Hidden Layers.
3. Mean Absolute Error :

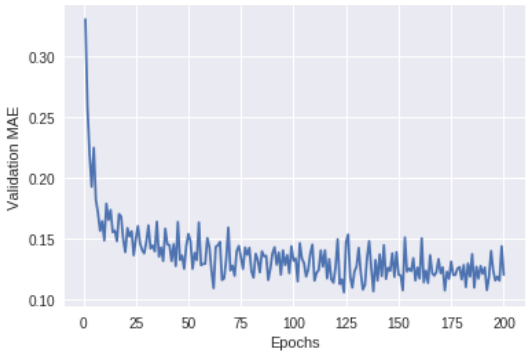


Tuning with Layers:

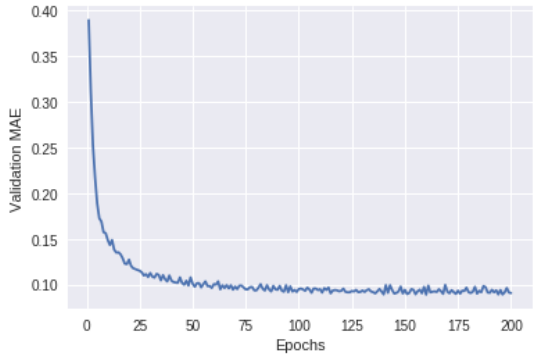
Layer 1, No Neurons: 32



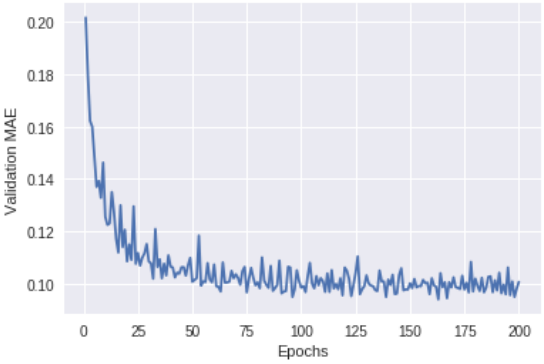
Layer 1, No Neurons: 64



Layer 2, No Neurons: [8, 8]



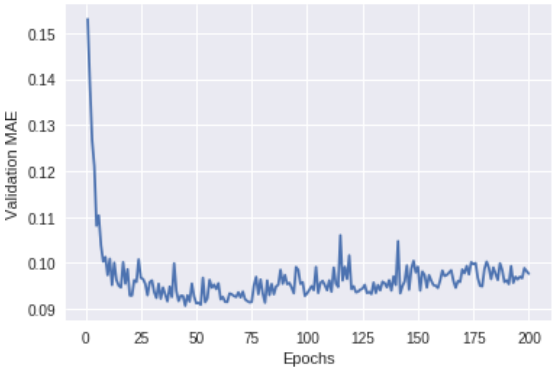
Layer 2, No Neurons: [32, 8]



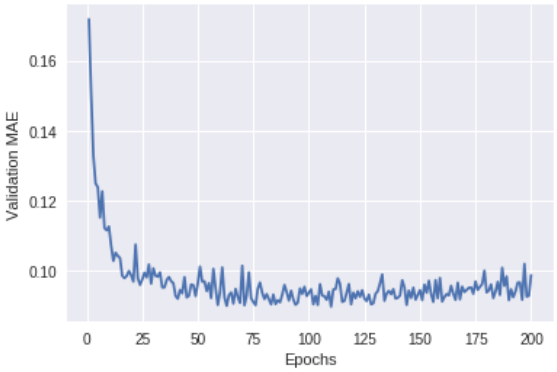
Based on the above Mean Absolute Error graphs, a 2 layer model with [8, 8] Neurons will be used.

Tuning Batch Size

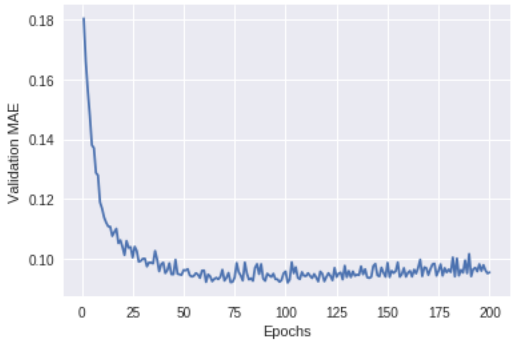
Batch Size 16



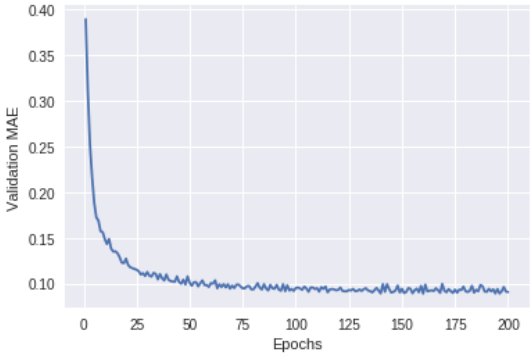
Batch size 128



Batch Size 256



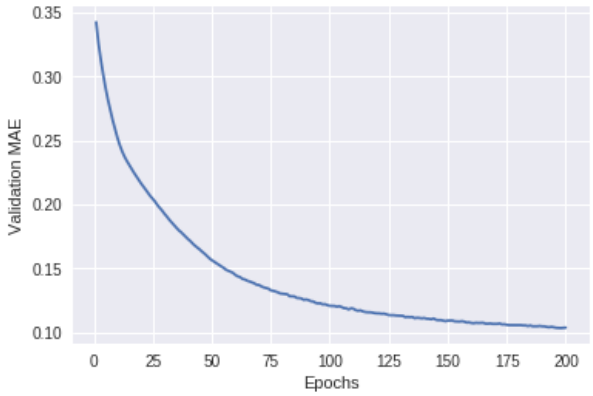
Batch size 512



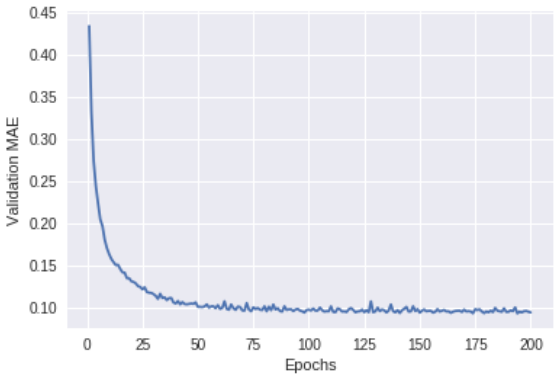
We cannot identify any significant changes in changing no of layers. So our prior choice will be same Layer 2 [8, 8] Neurons with batch size 512.

Tuning on Learning Rate:

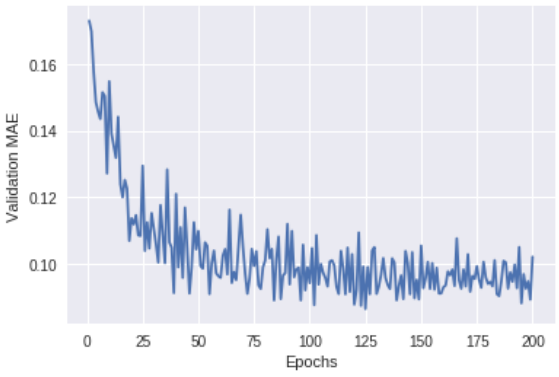
Rate: 0.0001



Rate: 0.001



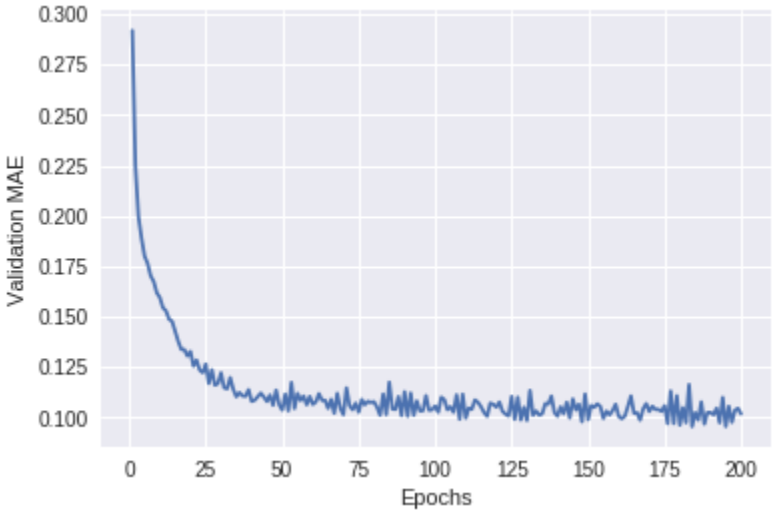
Rate: 0.01



Learning Rate of 0.001 gives a good performance.

Regularization:

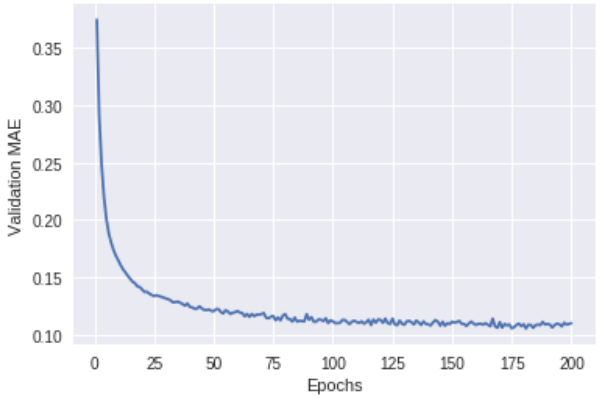
Regularization Rate (L1 0.001), (L1 0.001), (L1 0.005), (L1 0.0005), (L2 0.001), (L2 0.001), (L2 0.005), and (L2 0.0005)



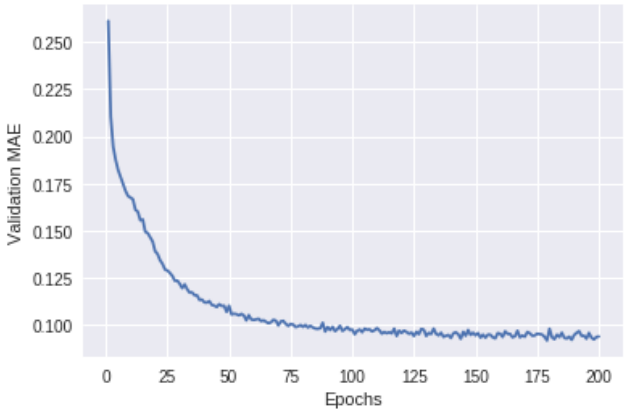
L1 Regularization with a rate of 0.005 leads to the lowest Validation MAE

Adding Dropouts:

Dropout = 0.5



Dropout = 0.2



No improvement has been observed in Dropouts, It is not use full for final model.

Results and Discussion:

Final Model:

#Number of layers: 2
#Layer Size: [8, 8]
#Epochs: 200
#Batch_size = 512
#Regularization = L1
#Regularization Rate: 0.005
#Learning Rate: 0.001

```
### Final Model
epochs = 200
batch_size = 512

final_model = define_model(num_layers = 2, num_neurons = [8, 8], input_shape = (x_train.shape[1],), optimizer_lr = 0.001,
                                regularizer="L1", reg_rate = 0.0005)
final_model.fit(x_train, y_train,
                  epochs = epochs,
                  batch_size = batch_size,
                  verbose = 0)

test_mse, test_mae = final_model.evaluate(x_test, y_test)

print(test_mae)
```

399/399 [==============================] - 1s 3ms/step
0.08791667968034744

Result Mean Accuracy Error: 0.087