Final Project

Deliverable 2

Content

Overview

This paper introduces an artificial neural network model that predicts the suicide rate and covers the experiments for arriving at this final model. The total experimental plan takes more than 7 hours, exploring the optimal setting for optimizer, learning rate, decay rate/momentum, training-testing split, activation function, unit numbers of layers, and epoch number.

Objective

The goal is to construct an artificial neural network regressor that predicts the suicide rate(suicides/100k population) of a group of people based 7 predictors including nationality, gender, age, time(year), population, generation, and GDP. This prediction is important and meaningful because it could be used as a baseline for measuring progress in suicide prevention activities for a specific country. In addition, it can be easily analyzed which subgroup has a higher suicide rate and the government could take a note of it.

The dataset has 27820 records covering the number of suicides from 1985 to 2016 in 101 countries, split by sex and age groups, plus the total population in according group. (Website:https://www.kaggle.com/russellyates88/suicide-rates-overview-1985-to-2016

Data Description & Choices

Column Name	Data type	Example	Whether/How Used in My Model
suicides/100k pop	Numeric	5.99	Response
country	Categorical (101 classes)	Austria	Predictor
year	Numeric (1985-2016)	1985	Predictor
sex	Categorical (2 classes)	Female, Male	Predictor
age	Categorical (6 classes)	15-24	Predictor
population	Numeric	430000	Predictor
gdp_per_capita ($)	Numeric	9372	Predictor
generation	Categorical (6 classes)	Generation X	Predictor
gdp_for_year ($)	Numeric	4800000000	Not used because it can be derived by "DGP/capita × population"
country-year	Categorical (2321 classes)	Albania1987	Not used because I already have chosen country and year as predictors.
suicides number	Numeric	23	Not used because I already have chosen "suicides/100k pop" as the response.
HDI for year	Numeric	0.78	Not used because 70% of the data is missing.

The original dataset has 12 columns including country, year, sex, age group, suicide number, population of the country, suicide number per 100 thousand people, a combination index of year and country, Human Development Index(HDI) of the year, Gross domestic product(GDP) for the year and GDP per capita in $. However, I chose only 7 of them as predictors, including country, year, sex, age group, generation, population of the country, and GDP for the year. I listed the reason why I did not choose the other columns and the data types of those columns in the following table.

Metrics

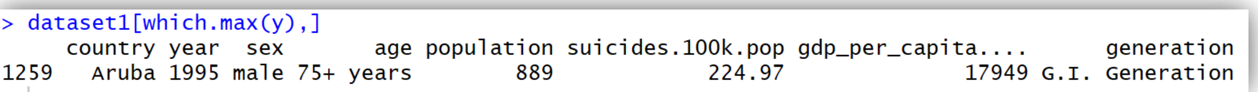
The goal is to predict a numerical response. Therefore, the type of ANN model is a regressor. Correspondingly, I will use Mean Squared Error(MSE) as the performance metrics. In statistics, the mean squared error measures the average of the squares of the errors—that is, the average squared difference between the estimated values and what is estimated. The MSE is always non-negative, and values closer to zero are better. I choose using MSE as metrics because it is a common measure for regression problem, and it clearly measures the performance of the models by measuring the errors. I will use test MSE as final metrics rather than training MSE. Moreover, I will calculate the Root Mean Squared Error(RMSE) to interpret the result better.

Input Variables & Preprocessing

As mentioned above, there are seven predictors and one numeric response.

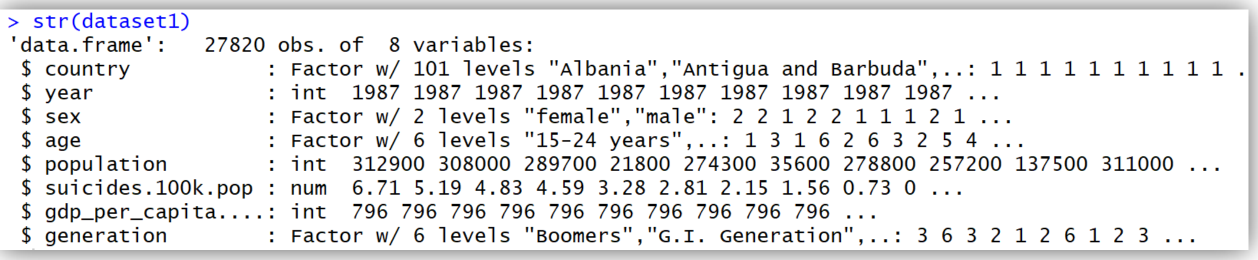
Column Name	Data type	Example	Whether/How Used in My Model
suicides/100k pop	Numeric	5.99	Response
country	Categorical (101 classes)	Austria	Predictor
year	Numeric (1985-2016)	1985	Predictor
sex	Categorical (2 classes)	Female, Male	Predictor
age	Categorical (6 classes)	15-24	Predictor
population	Numeric	430000	Predictor
gdp_per_capita ($)	Numeric	9372	Predictor
generation	Categorical (6 classes)	Generation X	Predictor

Taking a look at the numeric response, the suicide rate, we can find that the largest suicide rate is 224.97 suicides per 100,000 people, while the lowest suicide rate is 0 suicides per 100,000 people. For example, as follows, the suicide rate for the old man(75+Y) in Aruba in 1995 is the highest.



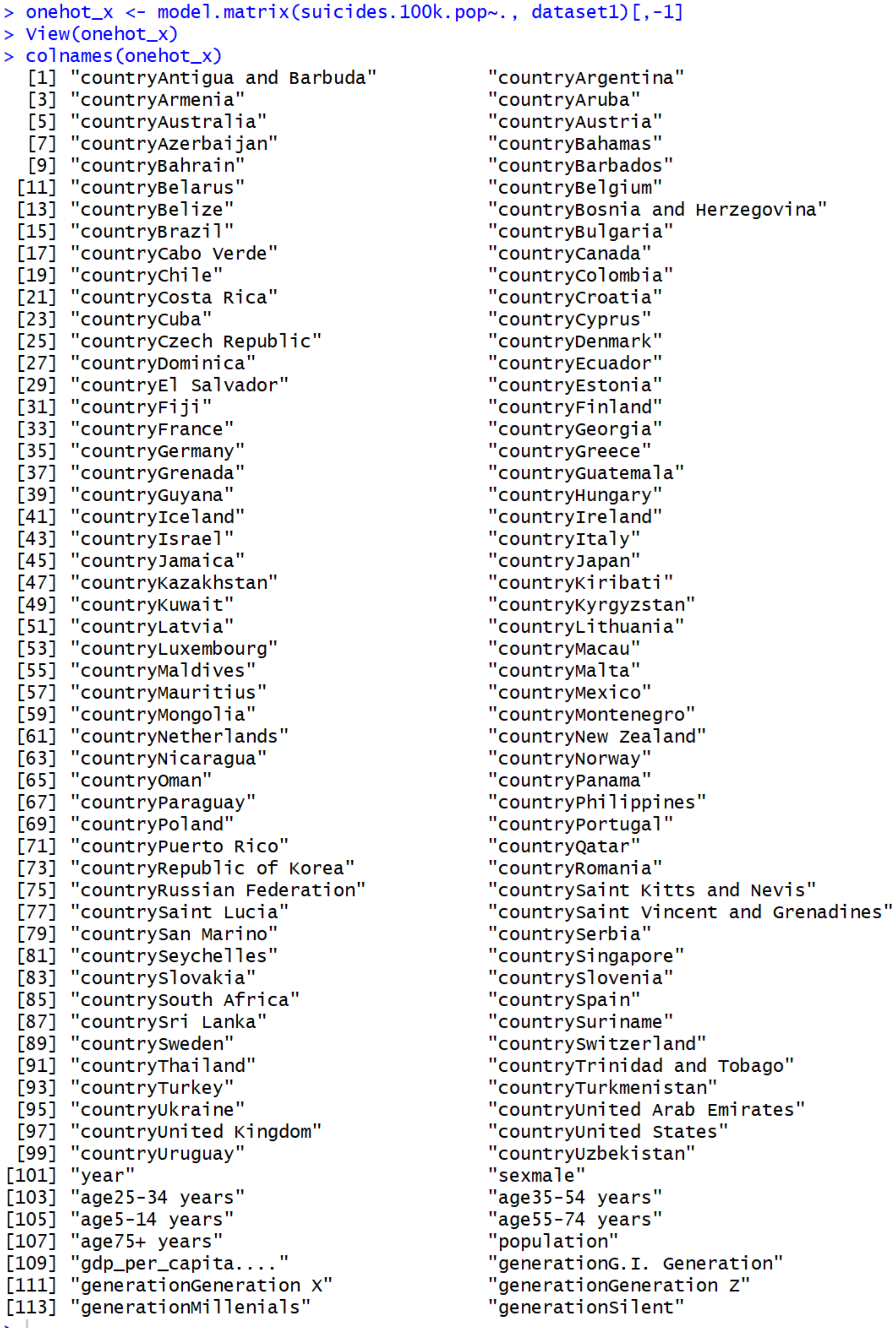
```
> dataset1[which.max(y),]
     country year  sex       age population suicides.100k.pop gdp_per_capita....       generation
1259   Aruba 1995 male 75+ years        889            224.97              17949 G.I. Generation
```

Among the predictors, “country”, “sex”, “age group”, “generation” are categorical data. However, the original data type is String, so I need to turn them into categorical data. I used R to do so and the result is as follows.



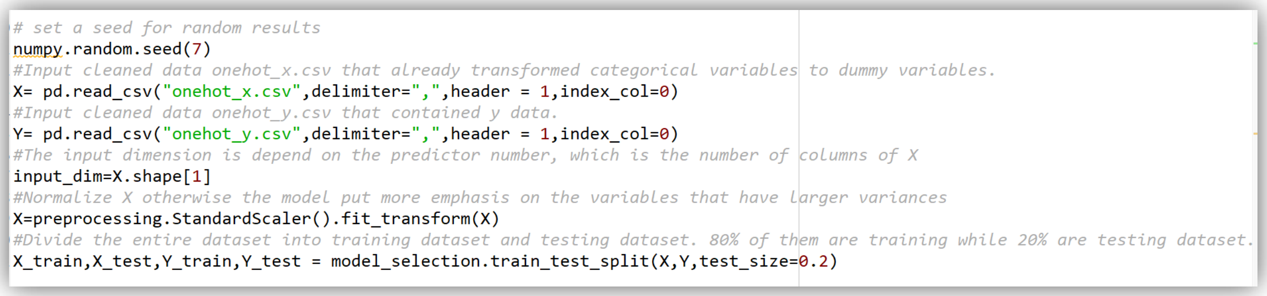
```
> str(dataset1)
'data.frame':	27820 obs. of  8 variables:
 $ country           : Factor w/ 101 levels "Albania","Antigua and Barbuda",..: 1 1 1 1 1 1 1 1 1 1 .
 $ year              : int  1987 1987 1987 1987 1987 1987 1987 1987 1987 1987 ...
 $ sex               : Factor w/ 2 levels "female","male": 2 2 1 2 2 1 1 1 2 1 ...
 $ age               : Factor w/ 6 levels "15-24 years",..: 1 3 1 6 2 6 3 2 5 4 ...
 $ population        : int  312900 308000 289700 21800 274300 35600 278800 257200 137500 311000 ...
 $ suicides.100k.pop : num  6.71 5.19 4.83 4.59 3.28 2.81 2.15 1.56 0.73 0 ...
 $ gdp_per_capita....: int  796 796 796 796 796 796 796 796 796 796 ...
 $ generation        : Factor w/ 6 levels "Boomers","G.I. Generation",..: 3 6 3 2 1 2 6 1 2 3 ...
```

Next, I need to turn those categorical variables into dummy variables. I also used R to do so by a function called “model.matrix”. The transformed columns are as follows. There are 114 columns of X.



```
> onehot_x <- model.matrix(suicides.100k.pop~., dataset1)[,-1]
> View(onehot_x)
> colnames(onehot_x)
  [1] "countryAntigua and Barbuda"          "countryArgentina"
  [3] "countryArmenia"                      "countryAruba"
  [5] "countryAustralia"                    "countryAustria"
  [7] "countryAzerbaijan"                   "countryBahamas"
  [9] "countryBahrain"                      "countryBarbados"
 [11] "countryBelarus"                      "countryBelgium"
 [13] "countryBelize"                       "countryBosnia and Herzegovina"
 [15] "countryBrazil"                       "countryBulgaria"
 [17] "countryCabo Verde"                   "countryCanada"
 [19] "countryChile"                        "countryColombia"
 [21] "countryCosta Rica"                   "countryCroatia"
 [23] "countryCuba"                         "countryCyprus"
 [25] "countryCzech Republic"               "countryDenmark"
 [27] "countryDominica"                     "countryEcuador"
 [29] "countryEl Salvador"                  "countryEstonia"
 [31] "countryFiji"                         "countryFinland"
 [33] "countryFrance"                       "countryGeorgia"
 [35] "countryGermany"                      "countryGreece"
 [37] "countryGrenada"                      "countryGuatemala"
 [39] "countryGuyana"                       "countryHungary"
 [41] "countryIceland"                      "countryIreland"
 [43] "countryIsrael"                       "countryItaly"
 [45] "countryJamaica"                      "countryJapan"
 [47] "countryKazakhstan"                   "countryKiribati"
 [49] "countryKuwait"                       "countryKyrgyzstan"
 [51] "countryLatvia"                       "countryLithuania"
 [53] "countryLuxembourg"                   "countryMacau"
 [55] "countryMaldives"                     "countryMalta"
 [57] "countryMauritius"                    "countryMexico"
 [59] "countryMongolia"                     "countryMontenegro"
 [61] "countryNetherlands"                  "countryNew Zealand"
 [63] "countryNicaragua"                    "countryNorway"
 [65] "countryOman"                         "countryPanama"
 [67] "countryParaguay"                     "countryPhilippines"
 [69] "countryPoland"                       "countryPortugal"
 [71] "countryPuerto Rico"                  "countryQatar"
 [73] "countryRepublic of Korea"            "countryRomania"
 [75] "countryRussian Federation"           "countrySaint Kitts and Nevis"
 [77] "countrySaint Lucia"                  "countrySaint Vincent and Grenadines"
 [79] "countrySan Marino"                   "countrySerbia"
 [81] "countrySeychelles"                   "countrySingapore"
 [83] "countrySlovakia"                     "countrySlovenia"
 [85] "countrySouth Africa"                 "countrySpain"
 [87] "countrySri Lanka"                    "countrySuriname"
 [89] "countrySweden"                       "countrySwitzerland"
 [91] "countryThailand"                     "countryTrinidad and Tobago"
 [93] "countryTurkey"                       "countryTurkmenistan"
 [95] "countryUkraine"                      "countryUnited Arab Emirates"
 [97] "countryUnited Kingdom"               "countryUnited States"
 [99] "countryUruguay"                      "countryUzbekistan"
[101] "year"                                "sexmale"
[103] "age25-34 years"                      "age35-54 years"
[105] "age5-14 years"                       "age55-74 years"
[107] "age75+ years"                        "population"
[109] "gdp_per_capita...."                  "generationG.I. Generation"
[111] "generationGeneration X"              "generationGeneration Z"
[113] "generationMillenials"                "generationSilent"
```

Then I saved the transformed X and Y in two separate tables by R and use python to read it.
Finally, I used processing package to scale X as follows and divide the whole dataset into training dataset and testing dataset.



```
# set a seed for random results
numpy.random.seed(7)
#Input cleaned data onehot_x.csv that already transformed categorical variables to dummy variables.
X= pd.read_csv("onehot_x.csv",delimiter=",",header = 1,index_col=0)
#Input cleaned data onehot_y.csv that contained y data.
Y= pd.read_csv("onehot_y.csv",delimiter=",",header = 1,index_col=0)
#The input dimension is depend on the predictor number, which is the number of columns of X
input_dim=X.shape[1]
#Normalize X otherwise the model put more emphasis on the variables that have larger variances
X=preprocessing.StandardScaler().fit_transform(X)
#Divide the entire dataset into training dataset and testing dataset. 80% of them are training while 20% are testing dataset.
X_train,X_test,Y_train,Y_test = model_selection.train_test_split(X,Y,test_size=0.2)
```

The input dimension is calculated by counting the number of columns in X.

Experimental Plan

The experimental plan covers the settings about optimizer, learning rate, decay rate/momentum, training-testing split, activation function, unit numbers of layers, and epoch number.

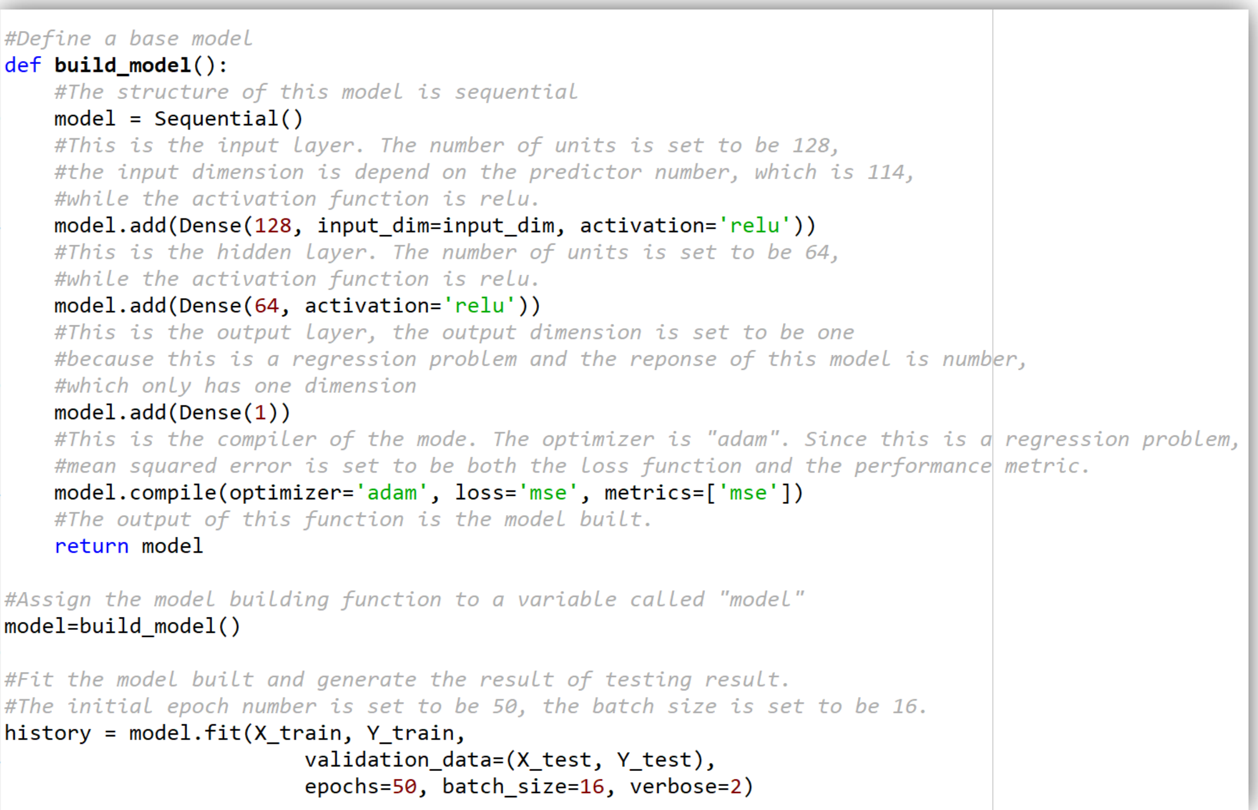
1. Build a base model and measure the performance by testing MSE. Plot the testing MSE and training MSE against epoch number.
2. Try which optimizer is the best by using grid search, taking consideration of “SGD”, “RMSprop”, and “Adam”. Measure the performance of the improved model with best optimizer by testing MSE. Plot the testing MSE and training MSE against epoch number.
3. Using the best optimizer I found in the previous step, and try which activation function is suitable, taking consideration of 'relu', 'tanh', 'sigmoid', 'linear'. Measure the performance of the improved model with best optimizer and action functions by testing MSE. Plot the testing MSE and training MSE against epoch number.
4. Using the optimal optimizer and activations I found, try different unit numbers and layer size. Measure the performance of the improved model with best optimizer, action functions, unit numbers by testing MSE. Plot the testing MSE and training MSE against epoch number.
5. Using the optimal optimizer, activations, and layers I found, try different learning and decay rate. Measure the performance of the improved model with best optimizer, action functions, unit numbers, learning rate and decay rate by testing MSE. Plot the testing MSE and training MSE against epoch number.
6. Try different training set proportion (80/20 or 70/30). Compare the result.
7. Determine the best epoch number for final model.
8. Build the best model with best epoch number and measure the result.

Detailed Result of Experimental Plan

Base Model

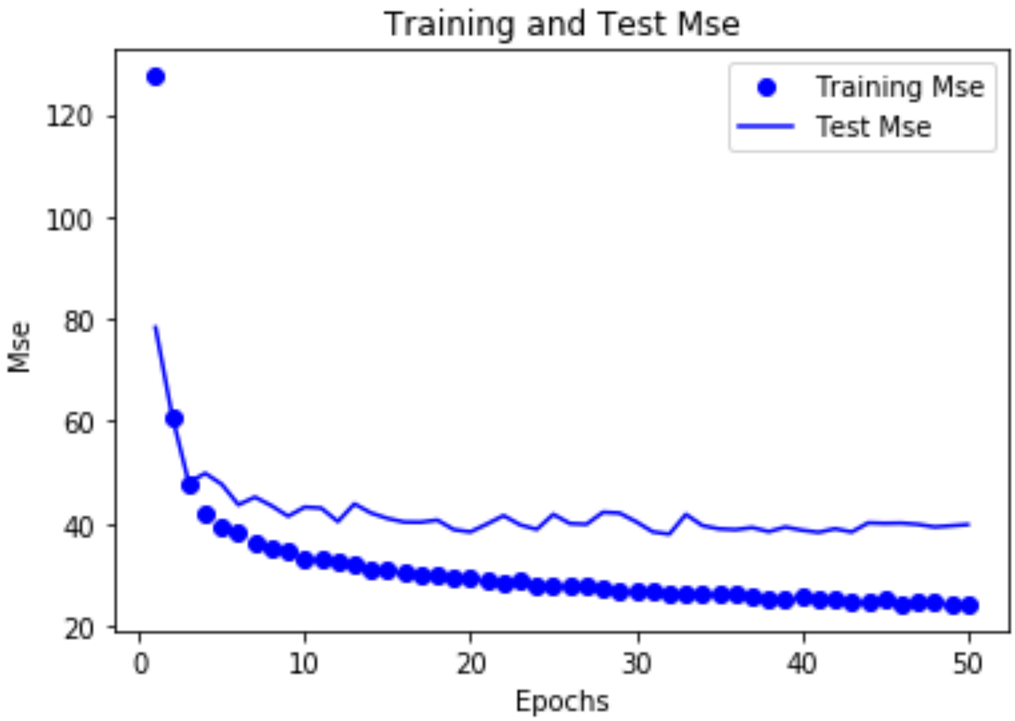
First of all, I built a base model without any experiment. This is the baseline of all the experiment.

The base model is as follows. There are three layers. The input dimension is determined by the column number of X. The output dimension is determined by Y. Since Y is numeric, the output dimension is also 1. The number of units in first layer is 128 while the number of units in second layer is set to be 64. Since the metrics I chose is MSE, both loss function and metrics are set to be MSE. The optimizer is set to be "adam". I use 50 as epoch number and 16 as batch size.



```
#Define a base model
def build_model():
    #The structure of this model is sequential
    model = Sequential()
    #This is the input layer. The number of units is set to be 128,
    #the input dimension is depend on the predictor number, which is 114,
    #while the activation function is relu.
    model.add(Dense(128, input_dim=input_dim, activation='relu'))
    #This is the hidden layer. The number of units is set to be 64,
    #while the activation function is relu.
    model.add(Dense(64, activation='relu'))
    #This is the output layer, the output dimension is set to be one
    #because this is a regression problem and the reponse of this model is number,
    #which only has one dimension
    model.add(Dense(1))
    #This is the compiler of the mode. The optimizer is "adam". Since this is a regression problem,
    #mean squared error is set to be both the loss function and the performance metric.
    model.compile(optimizer='adam', loss='mse', metrics=['mse'])
    #The output of this function is the model built.
    return model

#Assign the model building function to a variable called "model"
model=build_model()

#Fit the model built and generate the result of testing result.
#The initial epoch number is set to be 50, the batch size is set to be 16.
history = model.fit(X_train, Y_train,
                        validation_data=(X_test, Y_test),
                        epochs=50, batch_size=16, verbose=2)
```

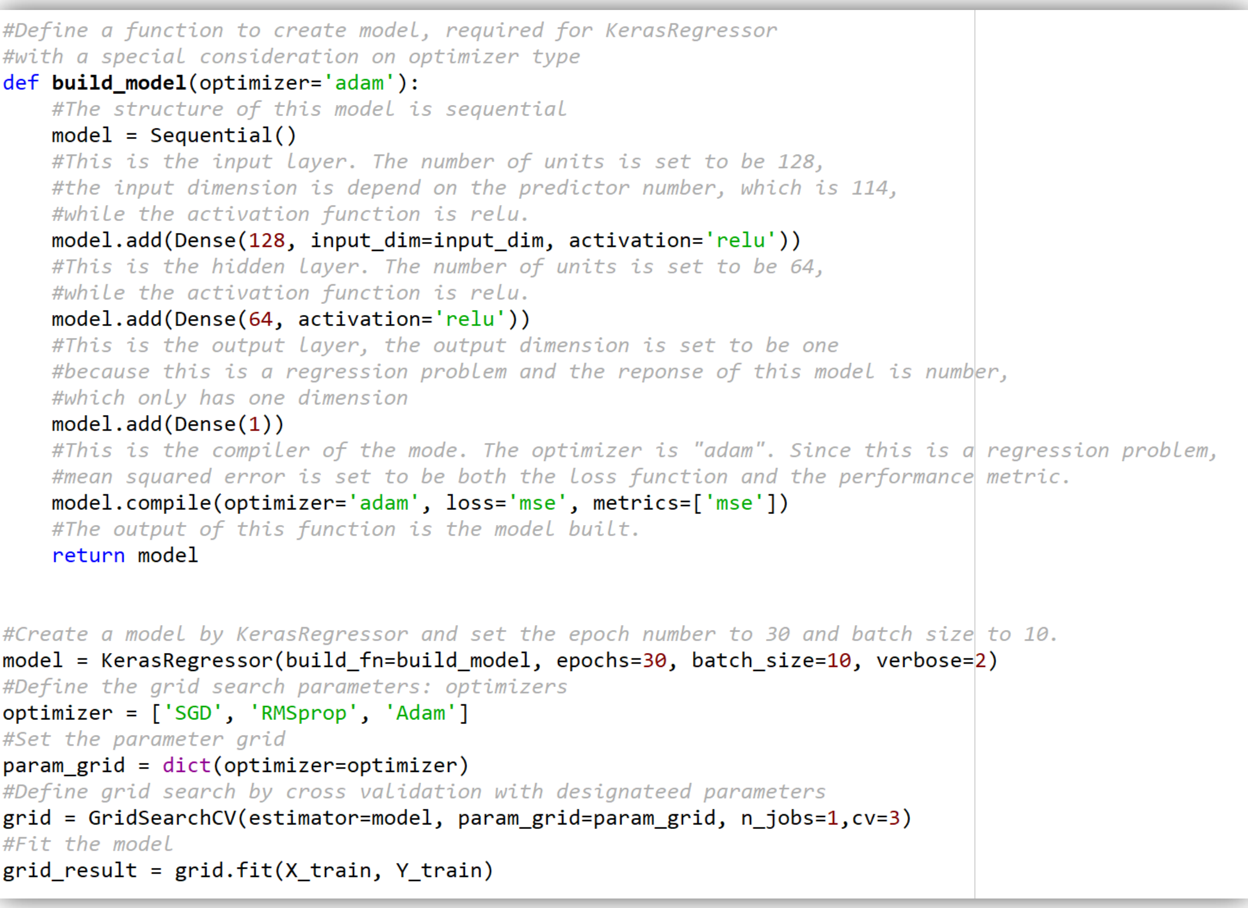
The result is as below. When the epoch number reaches 20, test MSE stops decreasing while the testing MSE will continue decreasing to approach 20. The best test MSE is **40**, which is the baseline of performance measure of all the following experiments.



Base Optimizer

In this section, I tried which optimizer is the best by using grid search, taking consideration of “SGD”, “RMSprop”, and “Adam”. The codes are as follows.

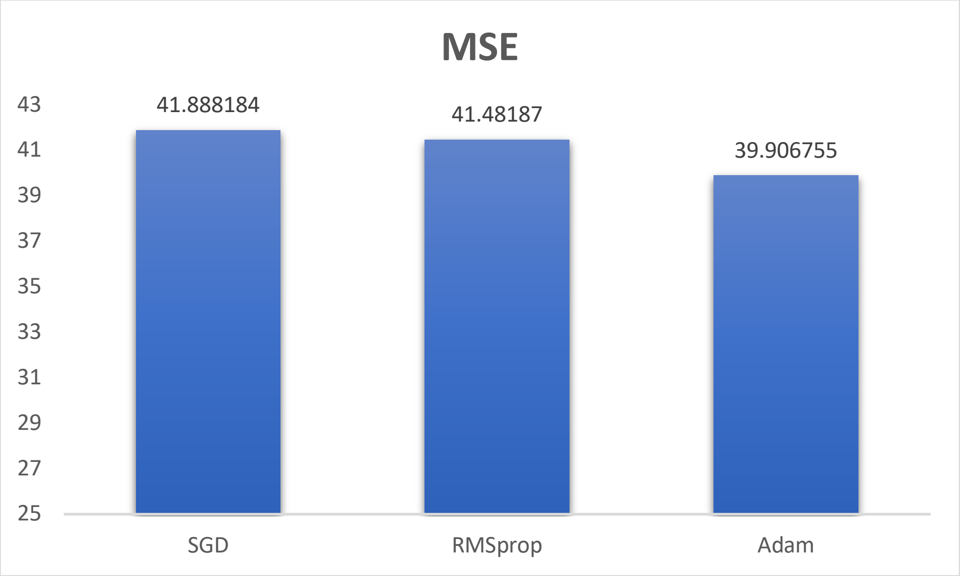
Except optimizer, everything keeps the same as the base model.



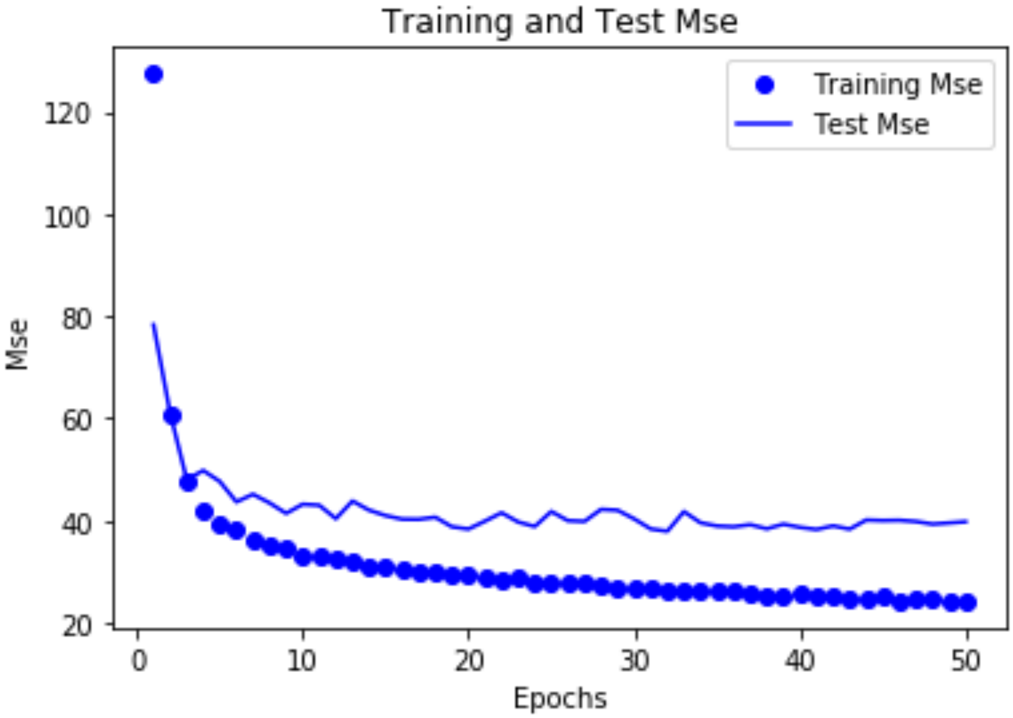
```python
#Define a function to create model, required for KerasRegressor
#with a special consideration on optimizer type
def build_model(optimizer='adam'):
    #The structure of this model is sequential
    model = Sequential()
    #This is the input layer. The number of units is set to be 128,
    #the input dimension is depend on the predictor number, which is 114,
    #while the activation function is relu.
    model.add(Dense(128, input_dim=input_dim, activation='relu'))
    #This is the hidden layer. The number of units is set to be 64,
    #while the activation function is relu.
    model.add(Dense(64, activation='relu'))
    #This is the output layer, the output dimension is set to be one
    #because this is a regression problem and the reponse of this model is number,
    #which only has one dimension
    model.add(Dense(1))
    #This is the compiler of the mode. The optimizer is "adam". Since this is a regression problem,
    #mean squared error is set to be both the loss function and the performance metric.
    model.compile(optimizer='adam', loss='mse', metrics=['mse'])
    #The output of this function is the model built.
    return model


#Create a model by KerasRegressor and set the epoch number to 30 and batch size to 10.
model = KerasRegressor(build_fn=build_model, epochs=30, batch_size=10, verbose=2)
#Define the grid search parameters: optimizers
optimizer = ['SGD', 'RMSprop', 'Adam']
#Set the parameter grid
param_grid = dict(optimizer=optimizer)
#Define grid search by cross validation with designateed parameters
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=1,cv=3)
#Fit the model
grid_result = grid.fit(X_train, Y_train)
```

The results are as follows:

- Time required for training**: 0:07:43.107312**
- MSE: 41.888184 (SD: 4.194474) with: {'optimizer': 'SGD'}
- MSE: 41.481870 (SD: 4.431057) with: {'optimizer': 'RMSprop'}
- MSE: 39.906755 (SD: 5.136570) with: {'optimizer': 'Adam'}
- The best optimizer is Adam.



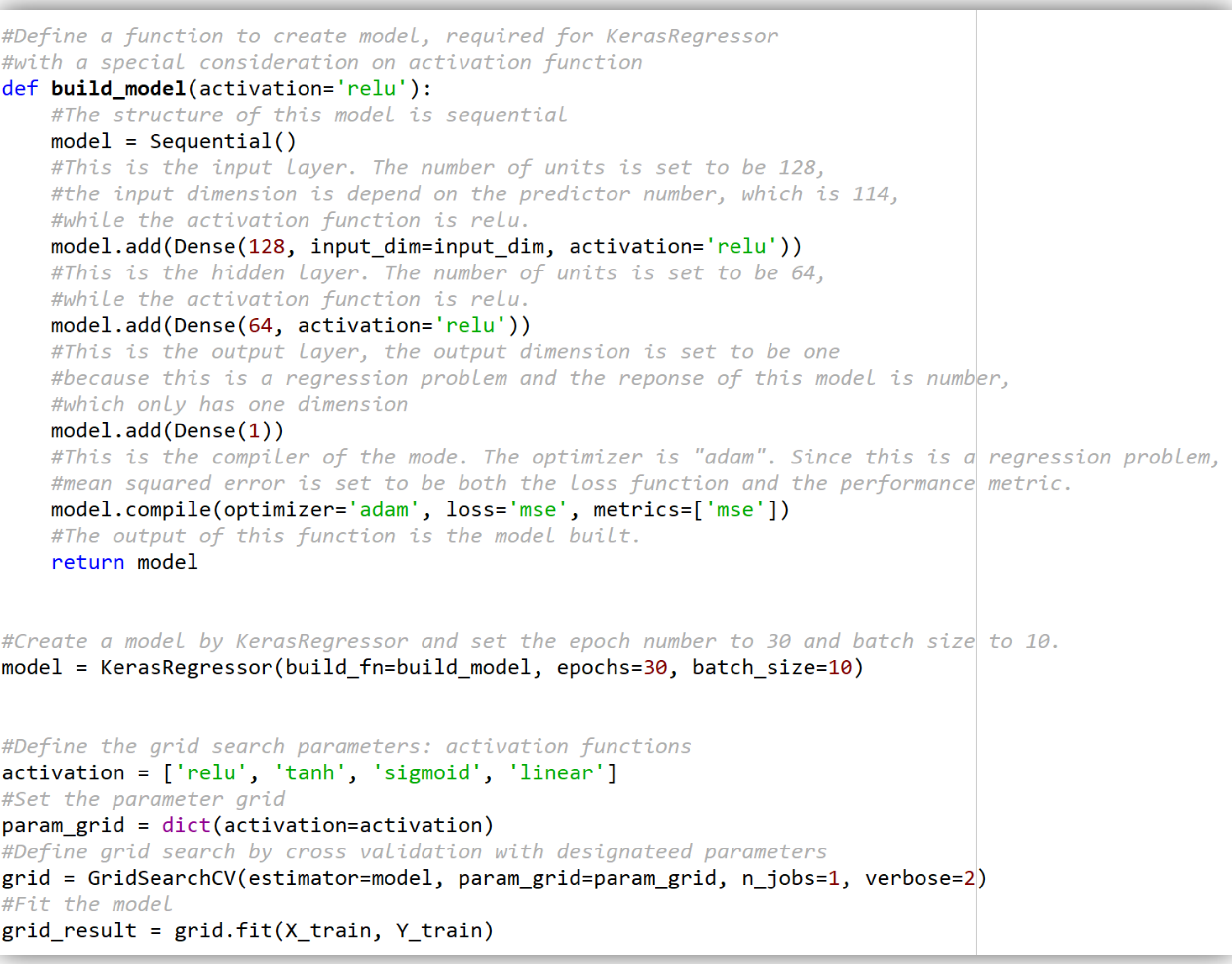
It is concluded that Adam is the best optimizer and I measured the performance the base model with best optimizer. The best test MSE is **40**. It is the same as the base model since I used “Adam” in the base model.



Activation Function

Based on the best optimizer “Adam”, I tried which activation function is suitable, taking consideration of 'relu', 'tanh', 'sigmoid', and 'linear'. The codes are as follows.

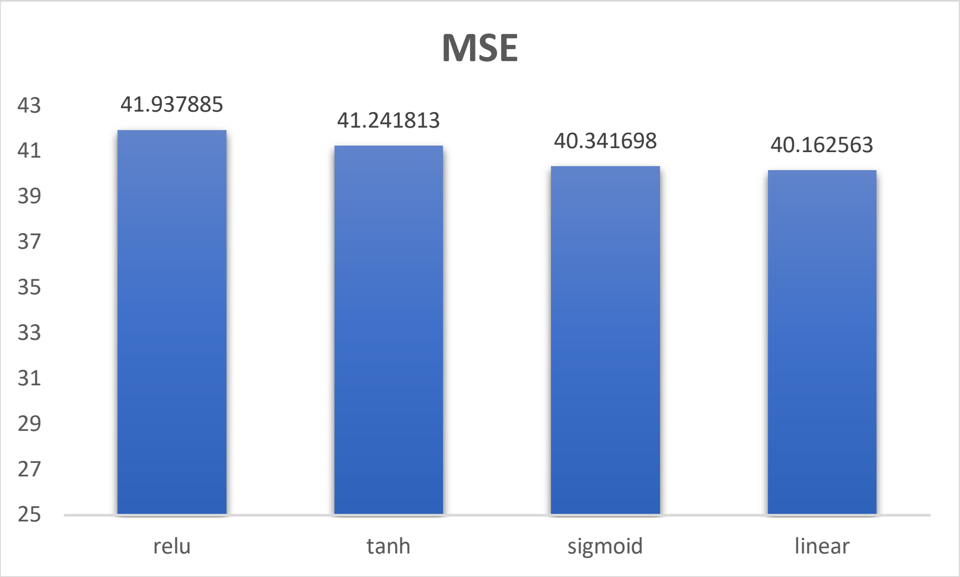
Except optimizer and activation function, everything keeps the same as the base model.



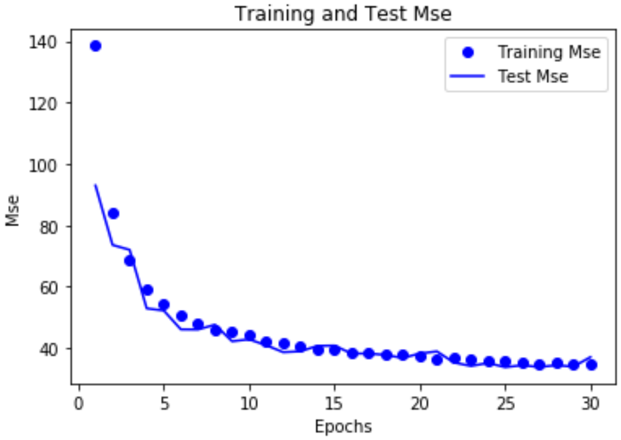
```python
#Define a function to create model, required for KerasRegressor
#with a special consideration on activation function
def build_model(activation='relu'):
    #The structure of this model is sequential
    model = Sequential()
    #This is the input layer. The number of units is set to be 128,
    #the input dimension is depend on the predictor number, which is 114,
    #while the activation function is relu.
    model.add(Dense(128, input_dim=input_dim, activation='relu'))
    #This is the hidden layer. The number of units is set to be 64,
    #while the activation function is relu.
    model.add(Dense(64, activation='relu'))
    #This is the output layer, the output dimension is set to be one
    #because this is a regression problem and the reponse of this model is number,
    #which only has one dimension
    model.add(Dense(1))
    #This is the compiler of the mode. The optimizer is "adam". Since this is a regression problem,
    #mean squared error is set to be both the loss function and the performance metric.
    model.compile(optimizer='adam', loss='mse', metrics=['mse'])
    #The output of this function is the model built.
    return model


#Create a model by KerasRegressor and set the epoch number to 30 and batch size to 10.
model = KerasRegressor(build_fn=build_model, epochs=30, batch_size=10)


#Define the grid search parameters: activation functions
activation = ['relu', 'tanh', 'sigmoid', 'linear']
#Set the parameter grid
param_grid = dict(activation=activation)
#Define grid search by cross validation with designateed parameters
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=1, verbose=2)
#Fit the model
grid_result = grid.fit(X_train, Y_train)
```

The results are as follows:

- Time required for training**: 0:12:53.906330**
- MSE: 41.937885 (SD: 4.465415) with: {'activation': 'relu'}
- MSE: 41.241813 (SD: 5.125712) with: {'activation': 'tanh'}
- MSE: 40.341698 (SD: 5.316120) with: {'activation': 'sigmoid'}
- MSE: 40.162563 (SD: 4.440145) with: {'activation': 'linear'}
- The best optimizer is “linear”.



It is concluded that “linear” is the best activation function and I measured the performance the base model with best optimizer. The best test MSE is reduced to **37.0404** from 40. In addition ,we can see that the testing MSE is very close to training MSE, avoiding overfitting perfectly.



Best Unit Numbers and Layer Size

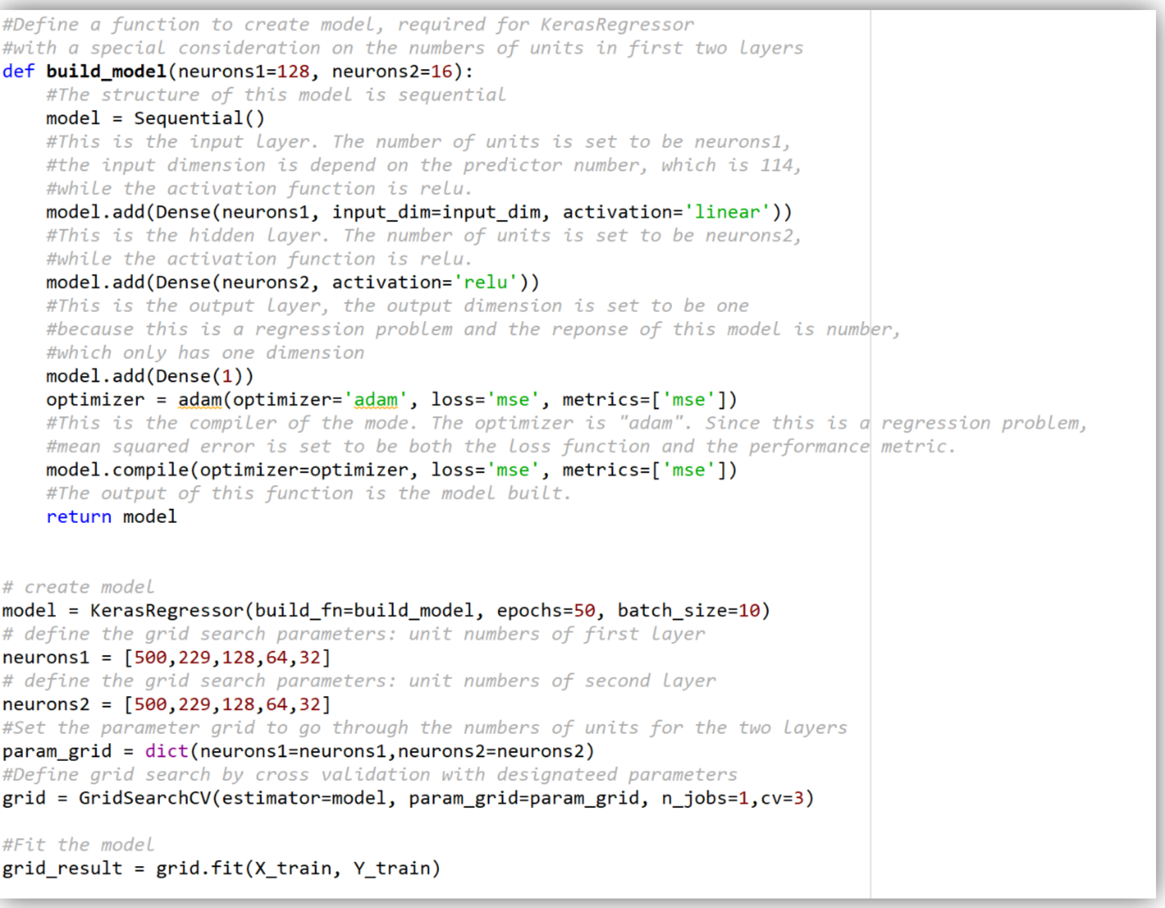
Based on the best optimizer “Adam” and optimal activation function “linear” in first layer, I tried different unit numbers and size for the first two layers.

Except optimizer, activation function and unit numbers, everything keeps the same as the base model.

I tried [500,229,128,64,32] as the unit number of the first layer, and [500,229,128,64,32] as the unit number of the second layer.

The reason why I tried 229 is because it is number of **2*n+1**(n is the input dimension).

The code is as follows.

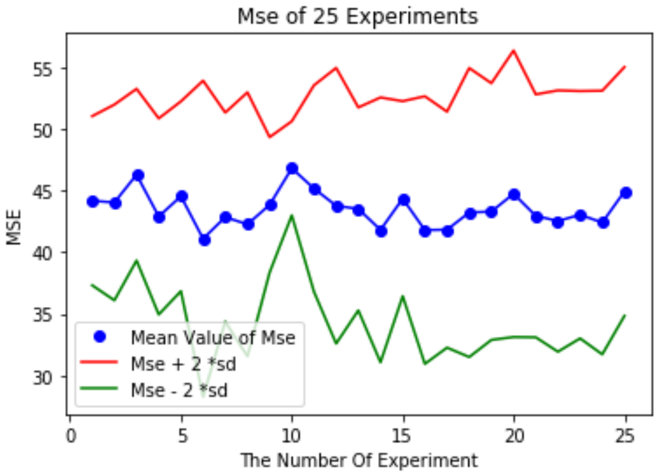


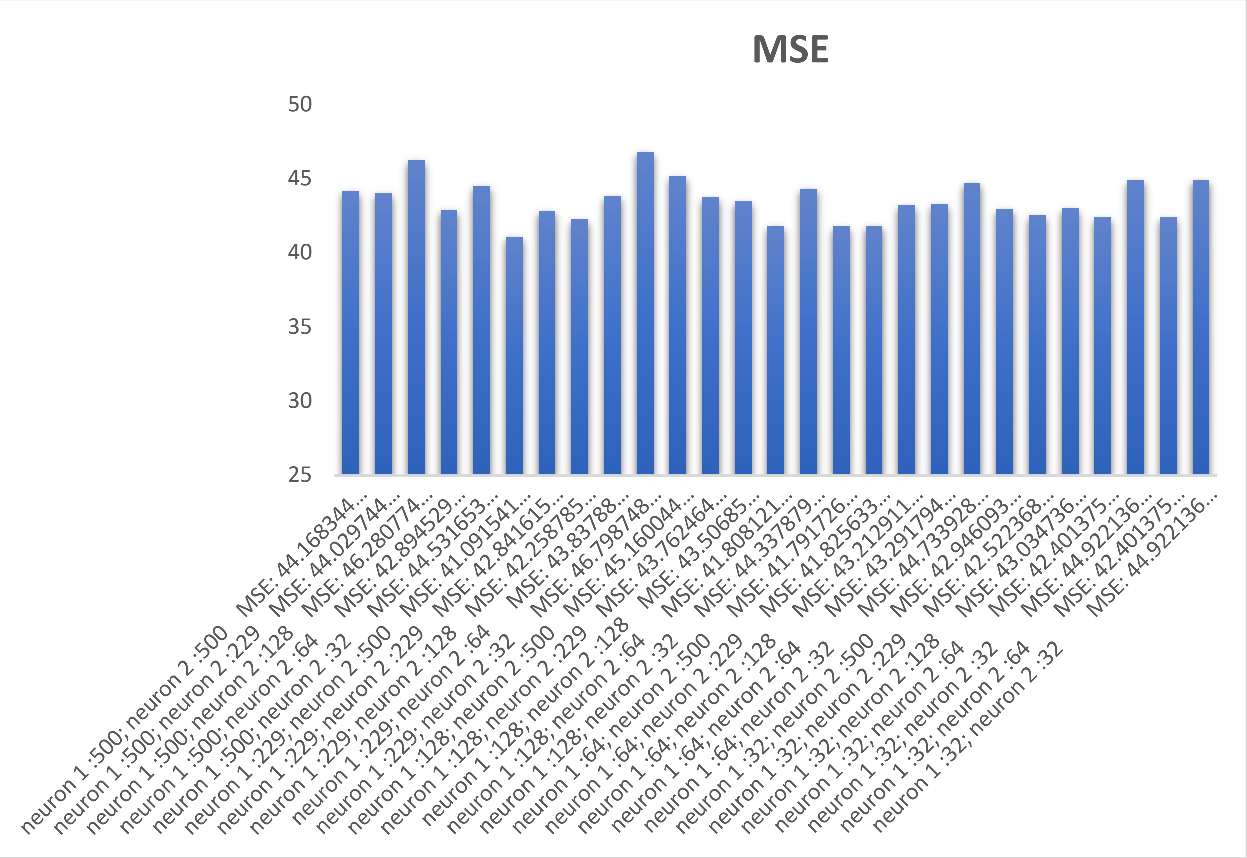
```
#Define a function to create model, required for KerasRegressor
#with a special consideration on the numbers of units in first two layers
def build_model(neurons1=128, neurons2=16):
    #The structure of this model is sequential
    model = Sequential()
    #This is the input layer. The number of units is set to be neurons1,
    #the input dimension is depend on the predictor number, which is 114,
    #while the activation function is relu.
    model.add(Dense(neurons1, input_dim=input_dim, activation='linear'))
    #This is the hidden layer. The number of units is set to be neurons2,
    #while the activation function is relu.
    model.add(Dense(neurons2, activation='relu'))
    #This is the output layer, the output dimension is set to be one
    #because this is a regression problem and the reponse of this model is number,
    #which only has one dimension
    model.add(Dense(1))
    optimizer = adam(optimizer='adam', loss='mse', metrics=['mse'])
    #This is the compiler of the mode. The optimizer is "adam". Since this is a regression problem,
    #mean squared error is set to be both the loss function and the performance metric.
    model.compile(optimizer=optimizer, loss='mse', metrics=['mse'])
    #The output of this function is the model built.
    return model


# create model
model = KerasRegressor(build_fn=build_model, epochs=50, batch_size=10)
# define the grid search parameters: unit numbers of first layer
neurons1 = [500,229,128,64,32]
# define the grid search parameters: unit numbers of second layer
neurons2 = [500,229,128,64,32]
#Set the parameter grid to go through the numbers of units for the two layers
param_grid = dict(neurons1=neurons1,neurons2=neurons2)
#Define grid search by cross validation with designateed parameters
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=1,cv=3)

#Fit the model
grid_result = grid.fit(X_train, Y_train)
```

The results are as follows:

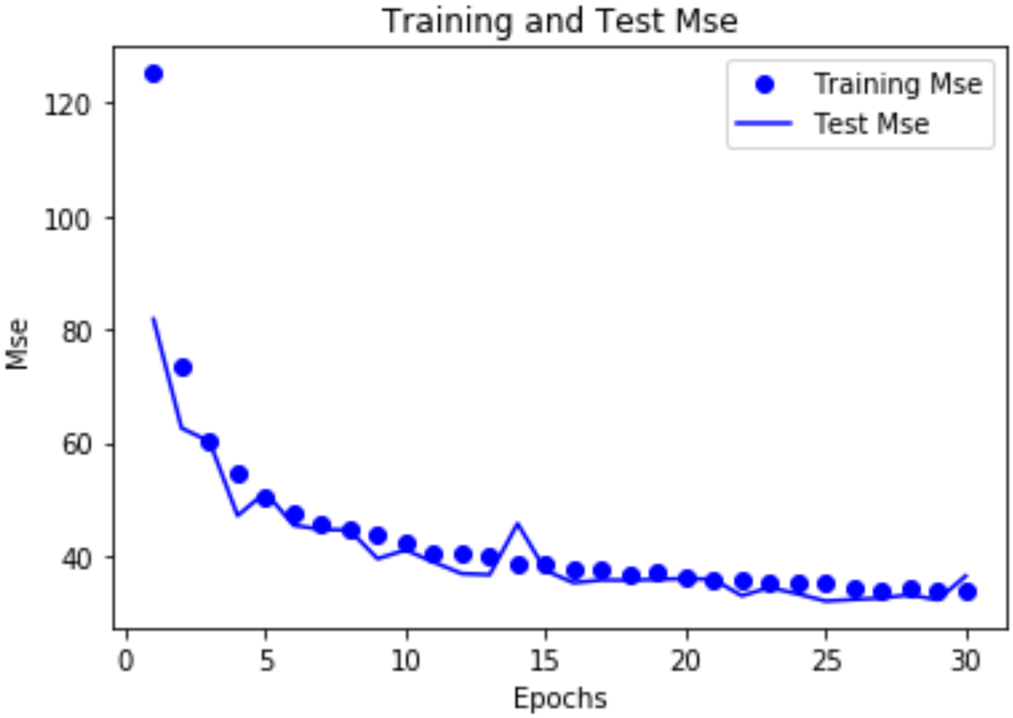
- Time required for training: **4:24:28.706118**
- neuron 1 :500; neuron 2 :500 MSE: 44.168344 SD: 3.426388
- neuron 1 :500; neuron 2 :229 MSE: 44.029744 SD: 3.964448
- neuron 1 :500; neuron 2 :128 MSE: 46.280774 SD: 3.477291
- neuron 1 :500; neuron 2 :64 MSE: 42.894529 SD: 3.972258
- neuron 1 :500; neuron 2 :32 MSE: 44.531653 SD: 3.843619
- neuron 1 :229; neuron 2 :500 MSE: 41.091541 SD: 6.402213
- neuron 1 :229; neuron 2 :229 MSE: 42.841615 SD: 4.235783
- neuron 1 :229; neuron 2 :128 MSE: 42.258785 SD: 5.345892
- neuron 1 :229; neuron 2 :64 MSE: 43.83788 SD: 2.740758
- neuron 1 :229; neuron 2 :32 MSE: 46.798748 SD: 1.90541
- neuron 1 :128; neuron 2 :500 MSE: 45.160044 SD: 4.18199
- neuron 1 :128; neuron 2 :229 MSE: 43.762464 SD: 5.583879
- neuron 1 :128; neuron 2 :128 MSE: 43.50685 SD: 4.110146
- neuron 1 :128; neuron 2 :64 MSE: 41.808121 SD: 5.364137
- neuron 1 :128; neuron 2 :32 MSE: 44.337879 SD: 3.948221
- neuron 1 :64; neuron 2 :500 MSE: 41.791726 SD: 5.420483
- neuron 1 :64; neuron 2 :229 MSE: 41.825633 SD: 4.779529

- neuron 1 :64; neuron 2 :128 MSE: 43.212911 SD: 5.853408
- neuron 1 :64; neuron 2 :64 MSE: 43.291794 SD: 5.199115
- neuron 1 :64; neuron 2 :32 MSE: 44.733928 SD: 5.804268
- neuron 1 :32; neuron 2 :500 MSE: 42.946093 SD: 4.920273
- neuron 1 :32; neuron 2 :229 MSE: 42.522368 SD: 5.292057
- neuron 1 :32; neuron 2 :128 MSE: 43.034736 SD: 5.009574
- neuron 1 :32; neuron 2 :64 MSE: 42.401375 SD: 5.33976
- neuron 1 :32; neuron 2 :32 MSE: 44.922136 SD: 5.039919
- neuron 1 :32; neuron 2 :64 MSE: 42.401375 SD: 5.33976
- neuron 1 :32; neuron 2 :32 MSE: 44.922136 SD: 5.039919
- The best unit number for layer 1 is 229, and the best unit number for layer 2 is 500.





It is concluded the best unit number for layer 1 is 229, and the best unit number for layer 2 is 500. In addition, I measured the performance the base model with best optimizer “adam”, best activation function “linear” in first layer, and best number of units. The best test MSE is **36.4415**, which is better than all the previous experiments.



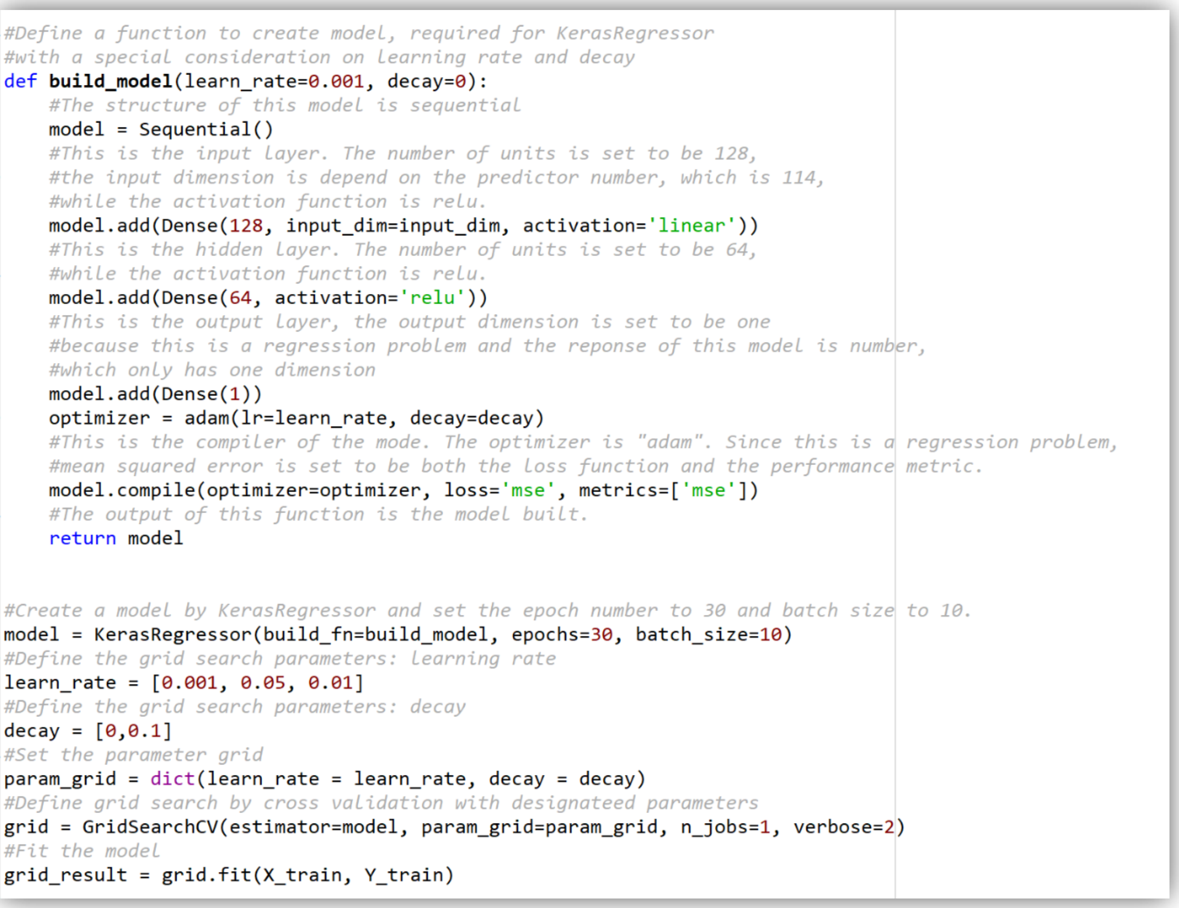
Learning Rate and Momentum(Decay)

Previously, it is concluded that the best optimizer is Adam. Therefore, this experiment should base on Adam. However, Adam only has learning rate but no parameters to adjust momentum. Therefore, I tried different decay rate instead of momentum.

Based on the best optimizer "Adam" and optimal activation function "linear" in first layer, I tried different learning rates and decay rates.

I tried [0.001, 0.05, 0.01] as learning rates of Adam and [0,0.1] as the decay rates.

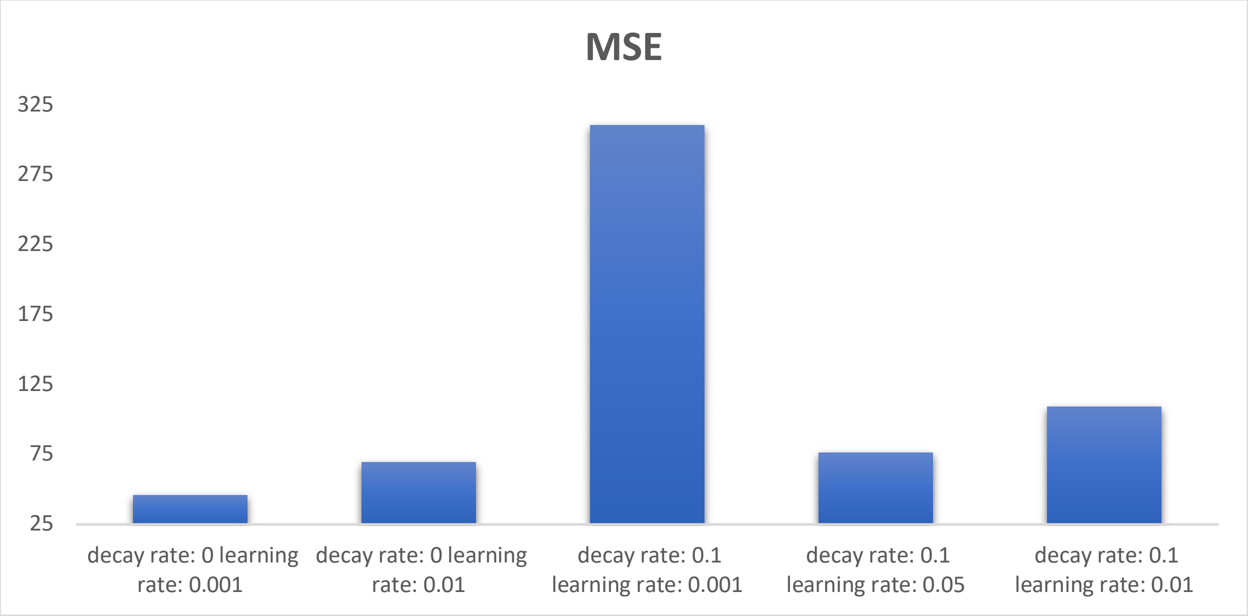
Except optimizer, activation function and unit numbers, everything keeps the same as the base model.



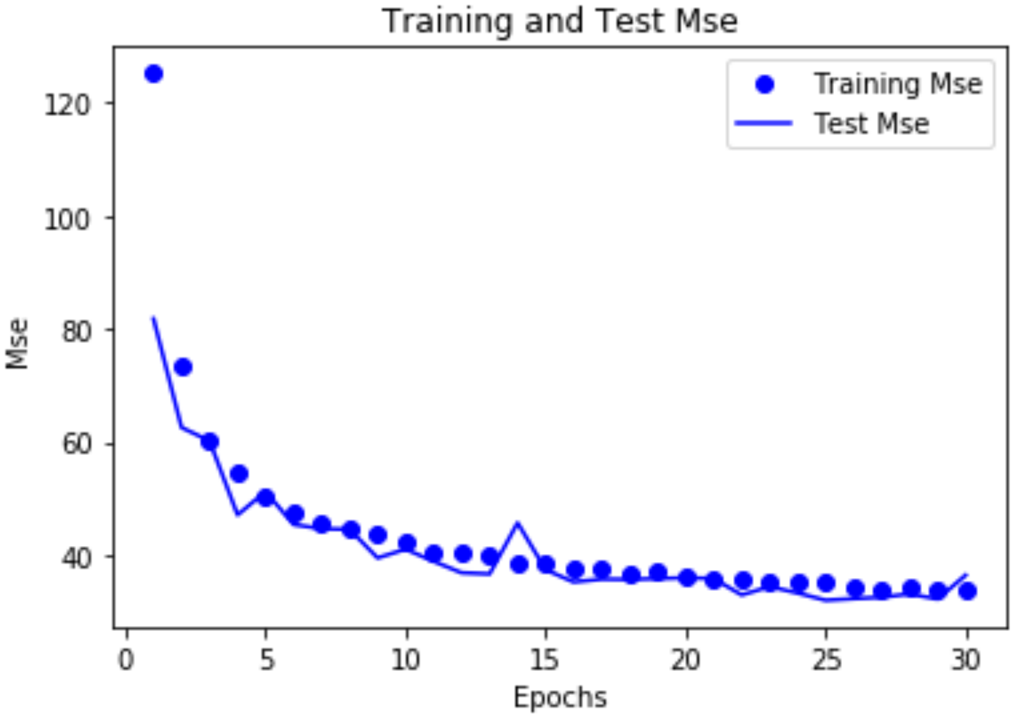
```python
#Define a function to create model, required for KerasRegressor
#with a special consideration on learning rate and decay
def build_model(learn_rate=0.001, decay=0):
    #The structure of this model is sequential
    model = Sequential()
    #This is the input layer. The number of units is set to be 128,
    #the input dimension is depend on the predictor number, which is 114,
    #while the activation function is relu.
    model.add(Dense(128, input_dim=input_dim, activation='linear'))
    #This is the hidden layer. The number of units is set to be 64,
    #while the activation function is relu.
    model.add(Dense(64, activation='relu'))
    #This is the output layer, the output dimension is set to be one
    #because this is a regression problem and the reponse of this model is number,
    #which only has one dimension
    model.add(Dense(1))
    optimizer = adam(lr=learn_rate, decay=decay)
    #This is the compiler of the mode. The optimizer is "adam". Since this is a regression problem,
    #mean squared error is set to be both the loss function and the performance metric.
    model.compile(optimizer=optimizer, loss='mse', metrics=['mse'])
    #The output of this function is the model built.
    return model


#Create a model by KerasRegressor and set the epoch number to 30 and batch size to 10.
model = KerasRegressor(build_fn=build_model, epochs=30, batch_size=10)
#Define the grid search parameters: learning rate
learn_rate = [0.001, 0.05, 0.01]
#Define the grid search parameters: decay
decay = [0,0.1]
#Set the parameter grid
param_grid = dict(learn_rate = learn_rate, decay = decay)
#Define grid search by cross validation with designateed parameters
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=1, verbose=2)
#Fit the model
grid_result = grid.fit(X_train, Y_train)
```

The results are as follows:

- Time required for training: **1:00:27.676657**
- decay rate: 0 learning rate: 0.001 MSE: 46.095426 SD: 4.748215
- decay rate: 0 learning rate: 0.05 MSE: 579254.846872 SD: 815584.38484
- decay rate: 0 learning rate: 0.01 MSE: 69.758648 SD: 7.716279
- decay rate: 0.1 learning rate: 0.001 MSE: 310.513672 SD: 18.898059
- decay rate: 0.1 learning rate: 0.05 MSE: 76.534455 SD: 3.496928
- decay rate: 0.1 learning rate: 0.01 MSE: 109.239686 SD: 3.625726
- The best decay rate is 0 and best learning rate is 0.001.



It is concluded that the best decay rate is 0 and best learning rate is 0.001.The best test MSE is **36.4415**. The result is the same as previous experiment because 0 is the default decay rate and 0.001 is the default learning rate.



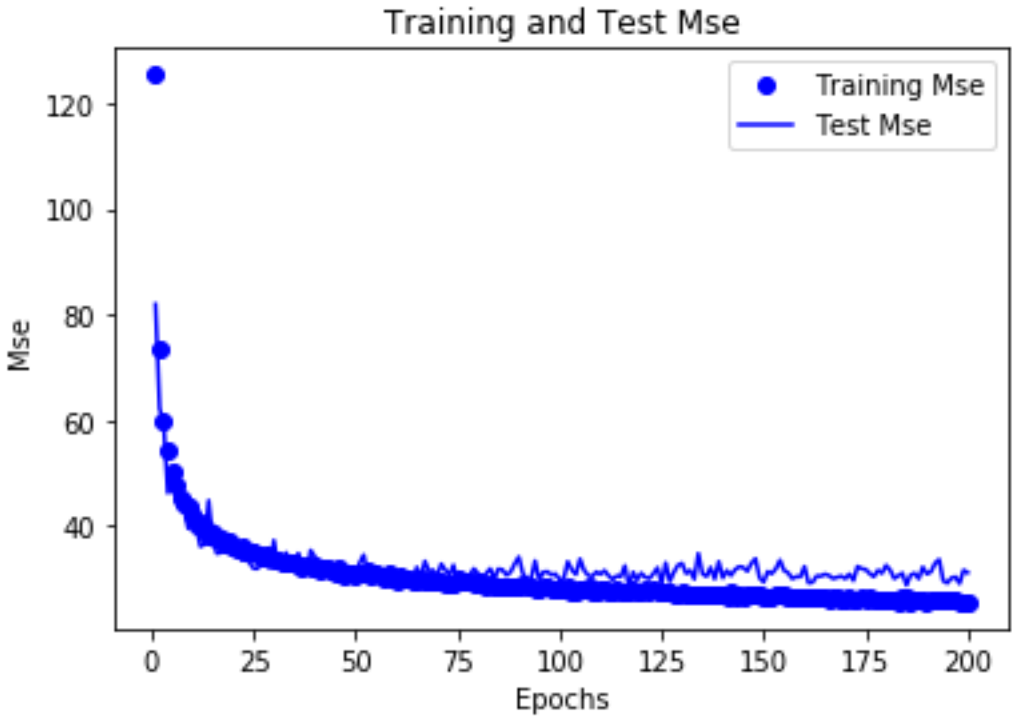
Best Training-Test Split

In this section, I will try whether 80/20 split is superior than 70/30 split of dataset based on the adjusted model with optimal optimizer, optimal learning rate and decay rate, optimal unit number and optimal activation function.

The result of 80/20 split is as follows.

The training MSE can be as low as 25.7130 while the test MSE **31.2162**. In addition, the graph shows that when the epoch number reaches 75, the test MSE stops decreasing.

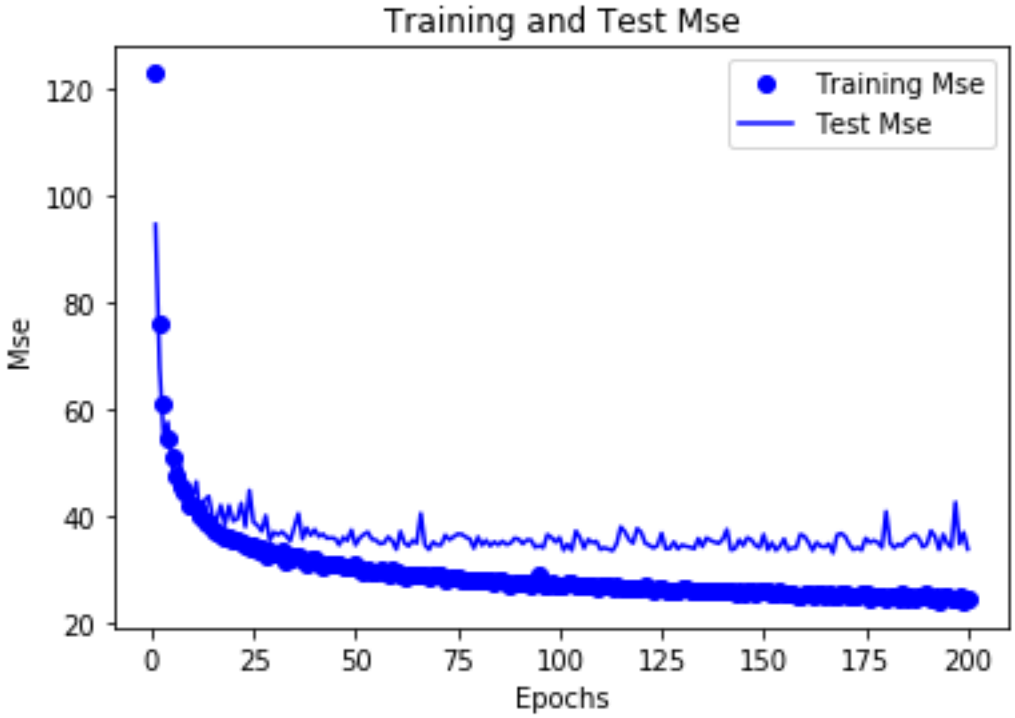
Time required for training is **0:48:59.714306**.



The result of 70/30 split is as follows.

The training MSE can be as low as **24.5610** while the test MSE **: 33.7864**. In addition, the graph shows that when the epoch number reaches 75, the test MSE stops decreasing.

Time required for training is **0:44:08.671238**.

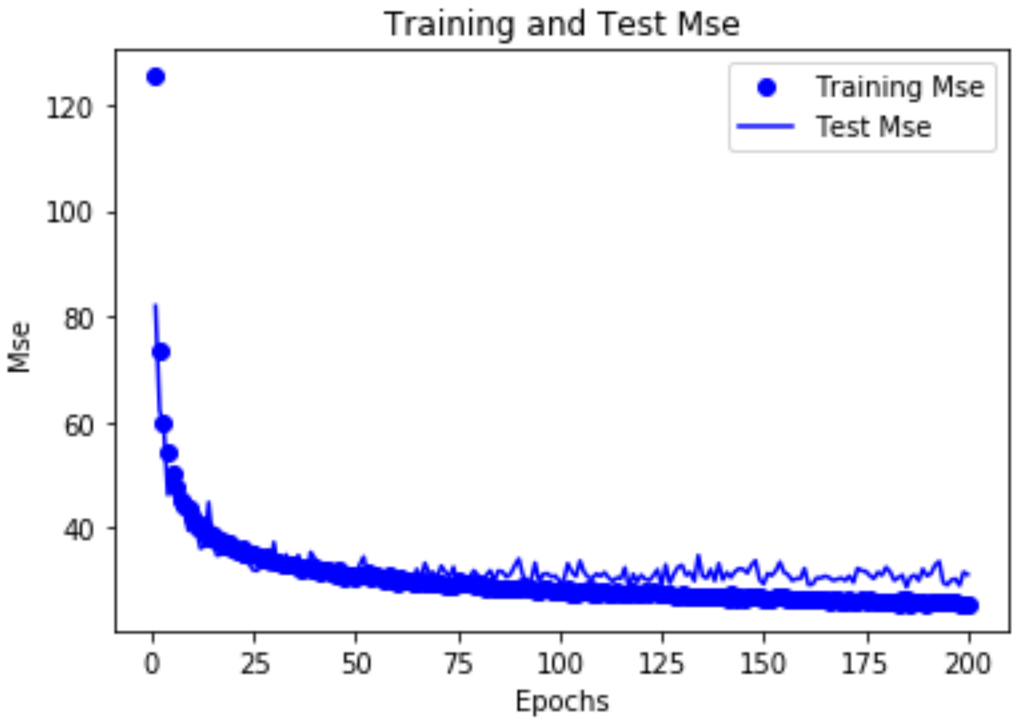


It can be concluded that 80/20 training-testing split is better than 70/30, since the test MSE of 80/20 split is lower than 70/30 split. However, it takes longer time to train the data since there are more training data.

Best Epoch Number

Finally, based on the best model with best parameters and 80/20 split, best epoch number can be determined.

According to the result below, we can set **75** as the best epoch number since further training does not help decrease test MSE.

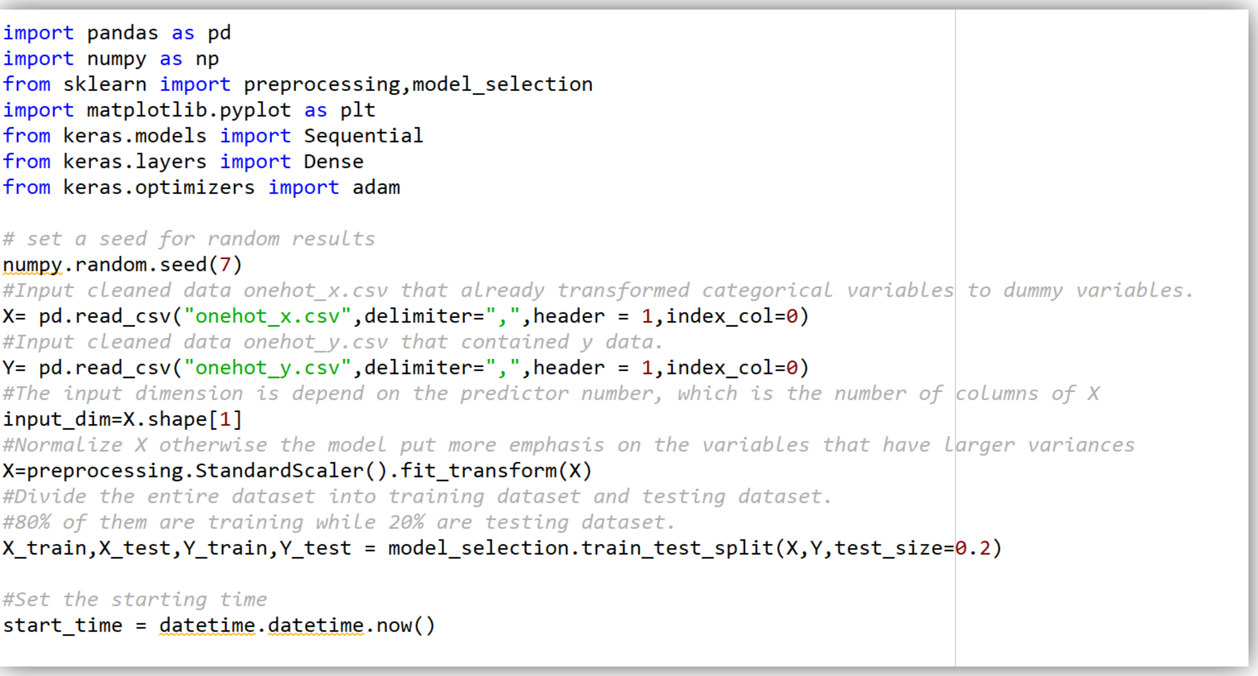


Final Model

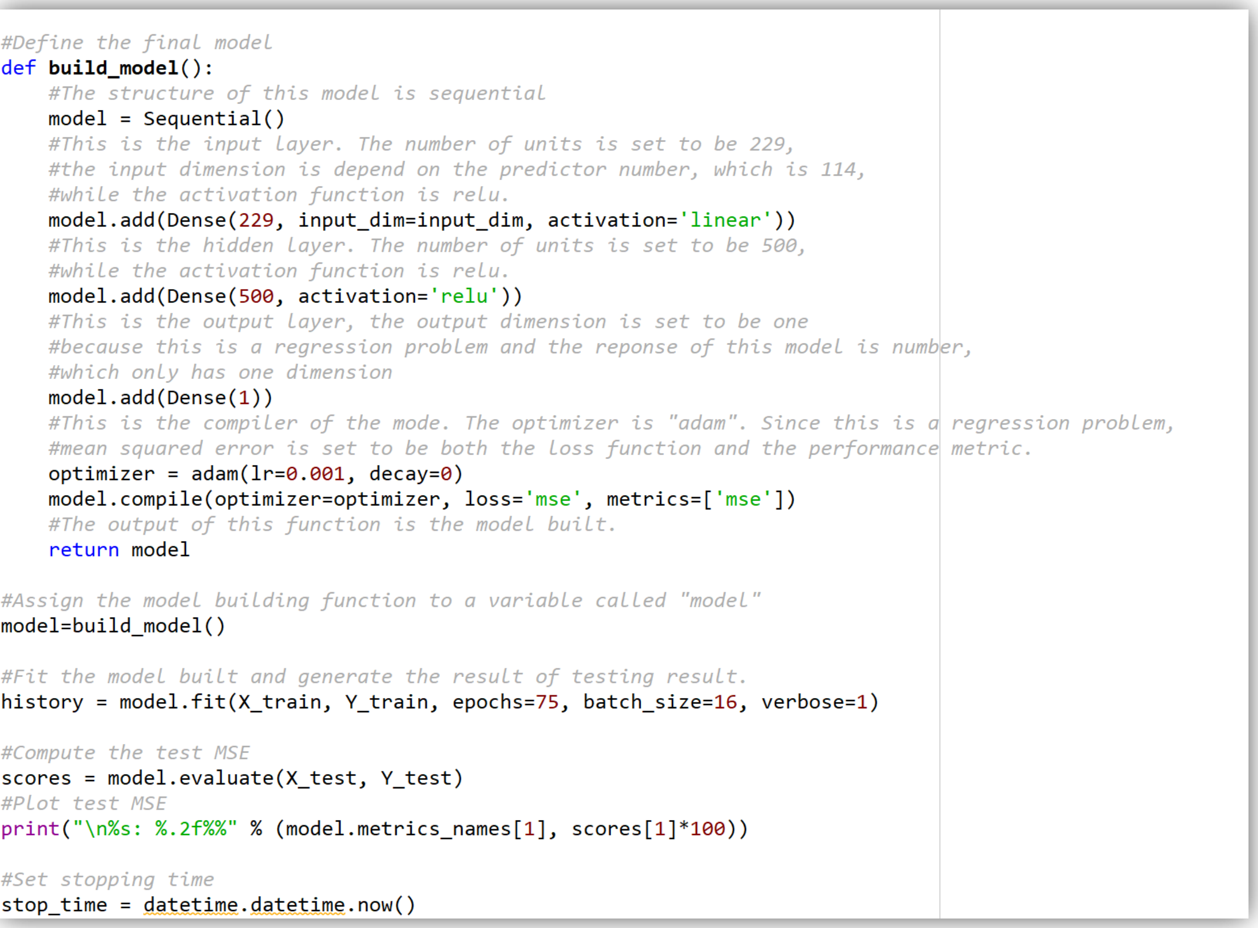
Explanation

1. The entire dataset, which involves 27820 records, is used. 80% of the data are used as training dataset while 20% are used as testing dataset. There are seven predictors and the transformed and scaled X has 114 columns, including many dummy variables.
2. The final model has a sequential structure with three dense layers.
   - The input dimension of the first layer is 114, corresponding to the column number of X. The unit number is 229 and the activation function is “linear”.
   - The unit number of second dense layer is 500 and the activation function is “relu”.
   - The output dimension of the third layer is 1 because the response is numeric.
   - The optimizer is “adam”. The learning rate is 0.001 and the decay rate is 0.
   - The metrics and loss function is set to be means squared value.
   - The epoch number is 75 and the batch size is 16.

Code

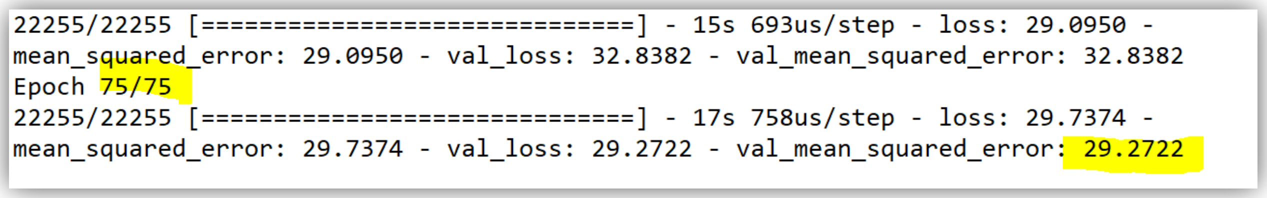


```
import pandas as pd
import numpy as np
from sklearn import preprocessing,model_selection
import matplotlib.pyplot as plt
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import adam

# set a seed for random results
numpy.random.seed(7)
#Input cleaned data onehot_x.csv that already transformed categorical variables to dummy variables.
X= pd.read_csv("onehot_x.csv",delimiter=",",header = 1,index_col=0)
#Input cleaned data onehot_y.csv that contained y data.
Y= pd.read_csv("onehot_y.csv",delimiter=",",header = 1,index_col=0)
#The input dimension is depend on the predictor number, which is the number of columns of X
input_dim=X.shape[1]
#Normalize X otherwise the model put more emphasis on the variables that have larger variances
X=preprocessing.StandardScaler().fit_transform(X)
#Divide the entire dataset into training dataset and testing dataset.
#80% of them are training while 20% are testing dataset.
X_train,X_test,Y_train,Y_test = model_selection.train_test_split(X,Y,test_size=0.2)

#Set the starting time
start_time = datetime.datetime.now()
```



```
#Define the final model
def build_model():
    #The structure of this model is sequential
    model = Sequential()
    #This is the input layer. The number of units is set to be 229,
    #the input dimension is depend on the predictor number, which is 114,
    #while the activation function is relu.
    model.add(Dense(229, input_dim=input_dim, activation='linear'))
    #This is the hidden layer. The number of units is set to be 500,
    #while the activation function is relu.
    model.add(Dense(500, activation='relu'))
    #This is the output layer, the output dimension is set to be one
    #because this is a regression problem and the reponse of this model is number,
    #which only has one dimension
    model.add(Dense(1))
    #This is the compiler of the mode. The optimizer is "adam". Since this is a regression problem,
    #mean squared error is set to be both the loss function and the performance metric.
    optimizer = adam(lr=0.001, decay=0)
    model.compile(optimizer=optimizer, loss='mse', metrics=['mse'])
    #The output of this function is the model built.
    return model

#Assign the model building function to a variable called "model"
model=build_model()

#Fit the model built and generate the result of testing result.
history = model.fit(X_train, Y_train, epochs=75, batch_size=16, verbose=1)

#Compute the test MSE
scores = model.evaluate(X_test, Y_test)
#Plot test MSE
print("\n%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))

#Set stopping time
stop_time = datetime.datetime.now()
```

Result of Metrics for This Problem



```
22255/22255 [==============================] - 15s 693us/step - loss: 29.0950 -
mean_squared_error: 29.0950 - val_loss: 32.8382 - val_mean_squared_error: 32.8382
Epoch 75/75
22255/22255 [==============================] - 17s 758us/step - loss: 29.7374 -
mean_squared_error: 29.7374 - val_loss: 29.2722 - val_mean_squared_error: 29.2722
```

As shown above, the test MSE of final model is 29.27. In addition, the root MSE (RMSE) is 5.41. In order word, the average squared prediction error is just 29.27, and the average difference between the prediction of suicide rate for a subgroup and the actual result is just 5.41 suicides per 100,000 people. Considering the range of response is from 0 suicides per 100,000 people to 224.97 suicides per 100,000 people, the accuracy of this model is really high and the performance is really great.

Moreover, comparing with the base model, whose test MSE is 40, the best model reduced test MSE by 26.8%. It shows that the experiment plan is implemented successfully and efficiently.

Validation of Final Model

The final model is validated because of three reasons.

First, in this case, testing set and training set is used. The model is solely trained by training set. Therefore, the test MSE is an accurate estimate of performance of the model in real practice.

Second, a thorough hyperparameter optimization is conducted here. The optimal optimizer, unit numbers and layer size, activation functions, training – testing set split, and epoch number are used. Therefore, the final model is one of the greatest model based on the purpose of this paper and data from this dataset.

Third, when implementing the experiment plan, GridSearchCV is used. It not only goes through the parameter candidates but also involves cross validation to obtain stable and valid results. Therefore, the experiments are reliable.

Other Considerations

Feedforward, Recurrent, or Backpropagation

The method involved in this paper is backpropagation. The best optimizer I chose is “Adam”. Initially, there are weights associate with different units. Then the performance / loss of this initial model is measured. Next, the weight is adjusted, and the relevant performance/loss is measured again. The model learns by seeing whether performance results are improved when it adjusts the weights in certain directions. In other words, the learning process is triggered by the results generated each time. Therefore, it is backpropagation.

Type of Training – Supervised or Unsupervised

Only supervised training is involved in this paper since the data is labeled – we already know the response, that is, the number of suicides per 100,000 people. The model is trained based on those labels.

Input Units and Output Units

Both the input unit number and output unit number are determined by the data. The input dimension is determined by the column number of X, which is 114, while the output dimension is determined by Y. Since Y is numeric, the output dimension is 1.

Size of Dataset and Records Used

There are 27820 records in the dataset, and I used all of them in the modeling, 80% of which is training set while 20% of which is used as testing set. I used all the available data since the running time is tolerable and the model can be more accurate due to more training data.

Discussion & Further Work

Due to the time limit, different parameter is optimized locally rather than globally. Therefore, maybe there are some great combinations of parameters that are not discussed. Moreover, when implementing grid search, not all possibilities are discussed. For example, only [32,64,128,229,500] is discussed for the unit number of first two layers but maybe 400 is the best choice. Furthermore, this model has not been tested by outside data.

Correspondingly, there are two things that can be done in the future. First, a global grid search can be done based on more powerful computation units such as AWS cloud services. Second, the suicide data after 2016 can be collected, and be used to test this final model.