Assignment 2 Report: Neural Networks

Course: ECE 9063 Data Analytics and Foundations

Submitted to: Katarina Grolinger

Student Name: Chunyang Jia

Student Number: 251001556

October 30, 2018

Table of Contents

Description of the forecasting problem

Different people may hold vastly different opinions on the fair value of a particular house. While this statement may sound general, in real life there are certain common factors that people prefer, say the overall lot area, the number of bedrooms, total square feet, neighborhood.

This dataset contains 80 attributes and the target "sales price" for 1460 cases of housing transactions. In this Assignment, I am going to train a Neural Network model to predict the Sales Price giving its features. I will also compare how closely my predictions are to the real sales price.

Overview of network architecture

Briefly speaking, I have used the Keras library to construct my neural network structure. For any given neural networks, there are many important parameters we need to specify. I started my neural network from the "default" model for beginners. My **basic model** contains the following features:

1. **Number of Layers**: Neuron network is made of layers of neurons. The first layer is called input layer and the last one is called output layer. All the middle ones are called hidden layers.
2. **Input Layer**: The input layer is composed of neurons that bring in the initial data into the network. Usually the number of neurons in input layers match the number of features we have in the data set. In this case, 19 features correspond to 19 neurons.
3. **Hidden Layer**: Hidden layer refers to those between input and output layer. Hidden layer can contain many layers and each layer can have many neurons. General rule is that complicated layers can learn more through large data sets but may have a higher chance of overfitting. Shallower hidden layers may have underfitting issues.

4. **Output Layer**: The output layer is the layer where we receive the output of the neural network. For this assignment, 1 output neuron corresponds to our target feature, "Sale Price". That's the goal of our neural network to predict.
5. **Initial Weight and Bias of neurons**: Keras provide the method initializer class to enable the users on initialize weights and biases before training starts. By default, Keras initialize all weights and biases automatically.
6. **Activation Function**: For the hidden layer neurons, I have chosen the "relu" function as the default activation function. Relu function can be written as f(x) = max(x,0). The advantage is that its derivative is always 1 when input is positive hence resulting faster learning than the sigmoid activation functions. Relu function may cause some "dead" neurons due to 0 output so some researchers proposed Leaky Relu function. The leaky Relu would output x in the positive region and a pre-set parameters times x in the negative regions instead of 0. Other commonly used is sigmoid function which however, can be slow at the far end as the slope of the function is very small approaching to 0. I am going to use Relu, tanh function in the model training process. Details will be covered in the next section.
7. **Optimizer** (Adam vs SGD): I am using the "Adam" optimizer proposed researchers and recommended by Keras as the default optimizers. In general, different optimizers tend to control the learning rate. Learning rate can is the speed we update the neural networks weight and biases. So, it can be written as $W = W - \eta * \nabla f(x)$, $B = B - \eta * \nabla f(x)$, where W is the weight, B is the bias and η is the learning rate, $\nabla f(x)$ is the derivative of our cost function.
    - For SGD optimizer, there are learning rate, decay, momentum and nestrov

momentum.

- For Adam optimizer, there are learning rate, beta_1, beta_2, epsilon, decay and AMSGrad variant. For this assignment, in order to simply this approach, I implemented this Adam optimizer. Further work on can be done on the beta_1, beta_2, decay, epsilon and AMSGrad variant. This method is firstly proposed by Diederik Kingma and Jimmy Ba in their 2015 ICLR paper titled "Adam: A method for Stochastic Optimization".

8. **Loss Function**: Predicting housing sales price is a regression problem. Two most common ones are listed below:

    - Mean Square Error (Quadratic Loss, L2 loss): the MSE is the sum of the squared differences between the label value and predicted value then divided by the number of samples. The equation can be written as:

$$MSE = \frac{\sum_{i=1}^{n} (y_i - y_i^p)^2}{n}$$

    - Mean Absolute Error (L1 loss): the MAE is the sum of the absolute differences of the label value and predicted value then divided by the number of samples. The equations can be written as:

$$MAE = \frac{\sum_{i=1}^{n} |y_i - y_i^p|}{n}$$

    - **Comparison**: In general, MSE is better because at the bottom, the gradient

of MSE is smaller, making it easier to find the lowest point. The gradient of MAE is always the same and hence harder to get to the bottom. However, the MSE will square the error which makes it bigger and when our data is corrupted with abnormal data points, MAE should be chosen over the MSE.

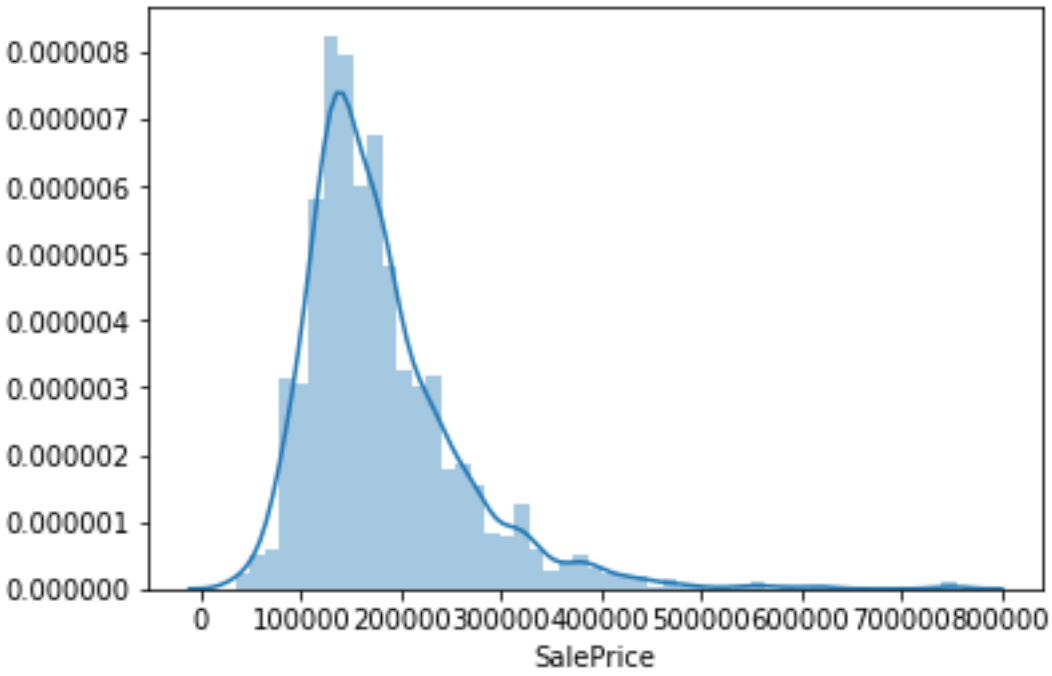
9. **Batch Size**: Batch Size refers to number of samples per gradient update. If unspecified, Keras defaults to 32.
    - When the batch size is set to the total number of samples, it's called batch gradient descent.
    - When batch size is set to 1 sample per update, it's called stochastic gradient descent.
    - When batch size is between 1 and total number of samples, it's called mini-batch gradient descent.
10. **Epochs**: Epochs refer to number of iterations to train the model. One epoch is an iteration over the entire training data set provided. Lower epochs can lead to underfitting while higher epochs will tend to overfitting problems. This can be observed by the decrease of loss errors but an increase in validation errors.
11. **Validation Split factor**: Keras provided this parameter for us to further divide the datasets to training and validation sets. It greatly helps us observe the potential of overfitting problems. Ideally the losses of both training and validation sets should decrease at the same time. In an overfitting situation, the losses of training sets continue to decrease while the losses of validation sets start to tick up. That's where we should stop. In my code, I used 0.1, equivalent to 10%, of my datasets to be the validation sets.

Description of Implementation of Neural Network

1. **Initial Data Reading and description of target data:** For the first step, I imported the pandas library and read in the original pandas dataframe. It has 1460 observations and 81 columns including our target value sale price. Then I used the "pandas describe function" on the sale price to get a brief idea of sale price for the 1460 observations. Result is as shown in the following picture. So, we the average of housing sales is 180K with a standard deviation of about 80K.

```
count      1460.000000
mean     180921.195890
std       79442.502883
min       34900.000000
25%      129975.000000
50%      163000.000000
75%      214000.000000
max      755000.000000
Name: SalePrice, dtype: float64
```

Here is the histogram of the sale price



2. **Feature Selection**: Among the 80 columns provided, I have chosen 19 important features. Also, to simply the problem, I chose 19 important continuous features and neglected the categorical features. Also, I did used the pandas correlation function to check their individual correlation to the "Sale Price" to help me quickly sort out the most relevant features. Attached table below is the summary of them. For continuous features, the Nan value can be replaced by 0.

Feature Name	Detail Description	Feature Name	Detail Description
SalePrice	the property's sale price in dollars (Label)	GrLivArea	Above grade (ground) living area square feet
LotFrontage	Linear feet of street connected to property	FullBath	Full bathrooms above grade
LotArea	Lot size in square feet	BedroomAbvGr	Number of bedrooms above basement level
OverallQual	Overall material and finish quality	KitchenAbovGr	Number of kitchens

OverallCond	Overall condition rating	TotRomsAbvGrd	Total rooms above grade (does not include bathrooms)
YearBuilt	Original construction date	Fireplaces	Number of fireplaces
YearRemodAdd	Remodel date	GarageCars	Size of garage in car capacity
TotalBsmtSF	Total square feet of basement area	WoodDeckSF	Wood deck area in square feet
1stFlrSF	First Floor Square Feet	PoolArea	Pool area in square feet
2ndFlrSF	Second Floor Square Feet	YrSold	Year Sold

3. **Removing outliers and filling Nan values**: For null values in some columns, I used the fillna method provided by pandas library to fill them with zeros as they are all numerical type. Also, from the previous step, most of the housing transactions are below 500K range. I used a for loop and removed the 9 outlier transactions. Outliers can have a big impact on our loss function Mean Squared Errors and hence, by doing this, it's will help us optimize the model in the model training step.
4. **Training and Testing data sets splitting**: In this step, I firstly separated the data into X dataframe which contains the 19 features and y series which contains our target set. Then divide them further into X_train, X_test, y_train, y_test. Our basic process is:
    i. Use the X_train and y_train to train the model by using the back-propagation strategy. Keras automatically does it on the back end to update the weights and biases based on the errors generated.
    ii. Then after training the model, use X_test to predict the "Sale Price" for this set, namely y_predict.

iii. Finally, we compare y_predict to y_test for the evaluate our model performance.

Name ▲	Type	Size
X	DataFrame	(1451, 19)
X_test	float64	(291, 19)
X_train	float64	(1160, 19)
df_corr	DataFrame	(38, 38)
df_data	DataFrame	(1451, 20)
y	Series	(1451,)
y1_predict	float32	(291, 1)
y_test	Series	(291,)
y_train	Series	(1160,)

5. **Data Pre-Processing on input data**: In this step, for input features, I normalized the input data using Scikit-learn library StandardScaler class to scale the data into more like a standard normal distributed data. The logic is that StandardScaler removes the mean and scales the data to standard deviation to 1.

6. **Model Architecture and Training:** My basic approach is that I will Build Model_1 and tune the number of hidden layers, neurons in each hidden layer, activation function and loss function. **Model_1** is the main model. **Model_2** is the Model_1 with modifications on number of hidden layers. **Model_3** is the Model_1 with modifications on neurons in each hidden layer. **Model_4** is the Model_1 with modifications on learning rate. **Model_5** is the Model_1 with modifications on loss function.

- **Here is the summary of their major difference.**

Features	Model 1	Model 2	Model 3	Model 4	Model 5
Total Number of Layer	12	17	12	12	12
Hidden Layer	10	15	10	15	10
Hidden Layer Neurons	60	60	90	60	60
Activation Function	relu	relu	relu	relu	relu
Learning Rate	0.001	0.001	0.001	0.005	0.001
Loss Function	MSE	MSE	MSE	MSE	MAE

- Other common factors to consider:

a) **Validation split factor**: 10% of the training data set will be used to validate the dataset to monitor the overfitting problems. My model was trained on 1044 samples and validated on 116 samples.

b) **Epochs and batch size**: I have tried a few runs and find batch size is 4 and the epochs are the ones I will tune for the sets to find out the optimal epochs.

c) **Activation function**: Rectified linear Unit. I have also tried tanh and sigmoid functions, but they converge very slow, so slow that I may need an epoch of 10000 times, which might take me hours to run. Also, if I tune up the learning rate greatly, the loss function looks very turbulent. Hence, I will leave that to future work.

d) **Initial weight and biases**: Keras randomly initialize the weights and biases. This may cause some minor differences in the final loss and results.

- Error Results after several trials.

Model 1 – basic model			
Epochs	50	Optimal (75)	500
Root Mean Squared Error	32541	30263	34304
Mean Absolute Error	22884	20710	22695
Model 2 – model with more hidden layers			
Epochs	50	Optimal (120)	500
Root Mean Squared Error	30512	30630	29571
Mean Absolute Error	20396	18973	20237
Model 3 – model with more neurons in each hidden layer			
Epochs	50	Optimal (110)	500
Root Mean Squared Error	32204	29701	31351
Mean Absolute Error	21150	20312	20776
Model 4 – model with 5 times learning rate			
Epochs	50	Optimal (80)	500
Root Mean Squared Error	40103	30233	30212
Mean Absolute Error	25617	19594	20668
Model 5 – model with MAE loss function			
Epochs	50	Optimal (70)	500
Root Mean Squared Error	32401	32437	31283
Mean Absolute Error	19130	20655	20896

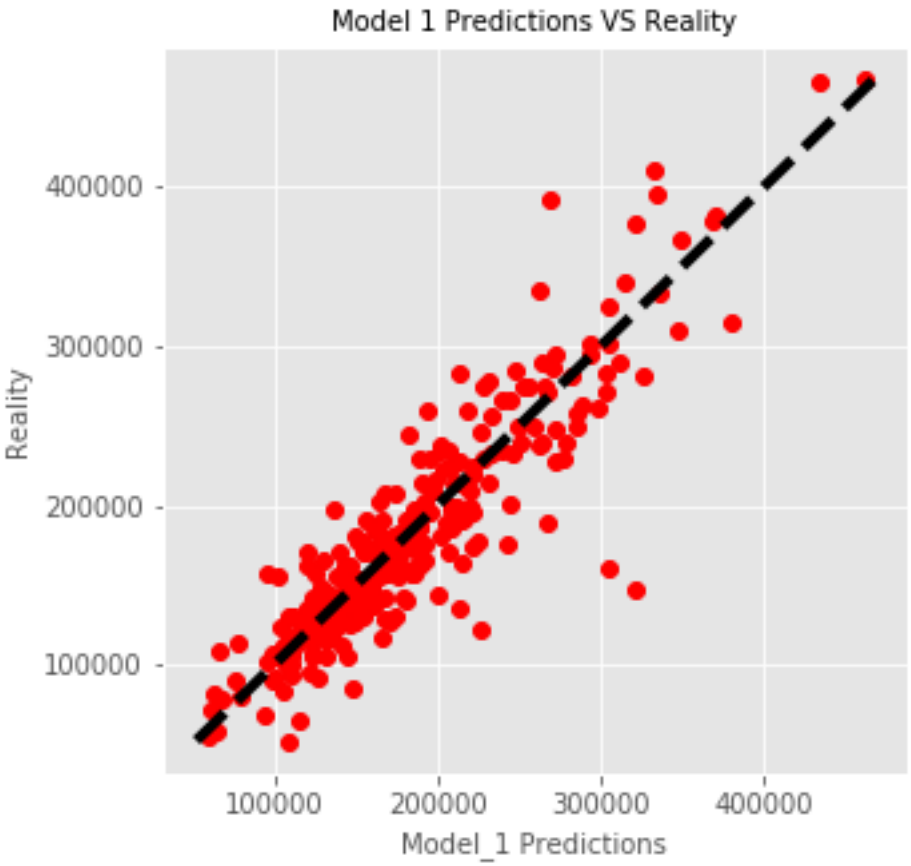
Results Comparison and Conclusion

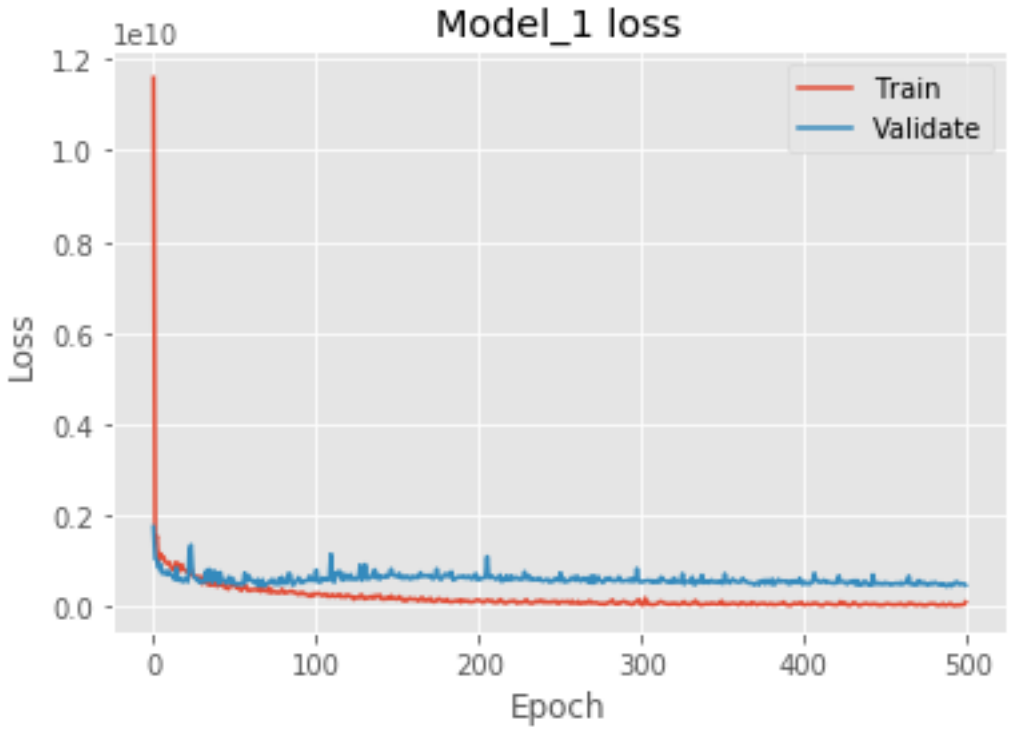
- Final Results comparison on the five Neural Network and previous work done through linear regression and Support Vector Machine models.

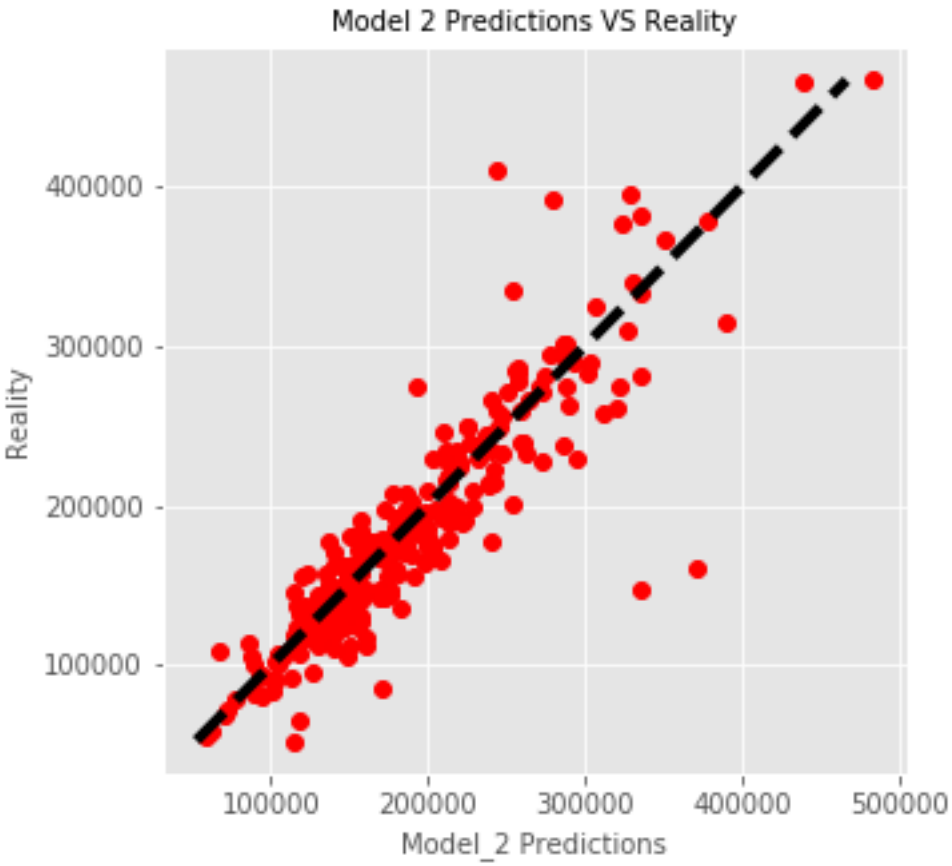
  Please see the table attached.

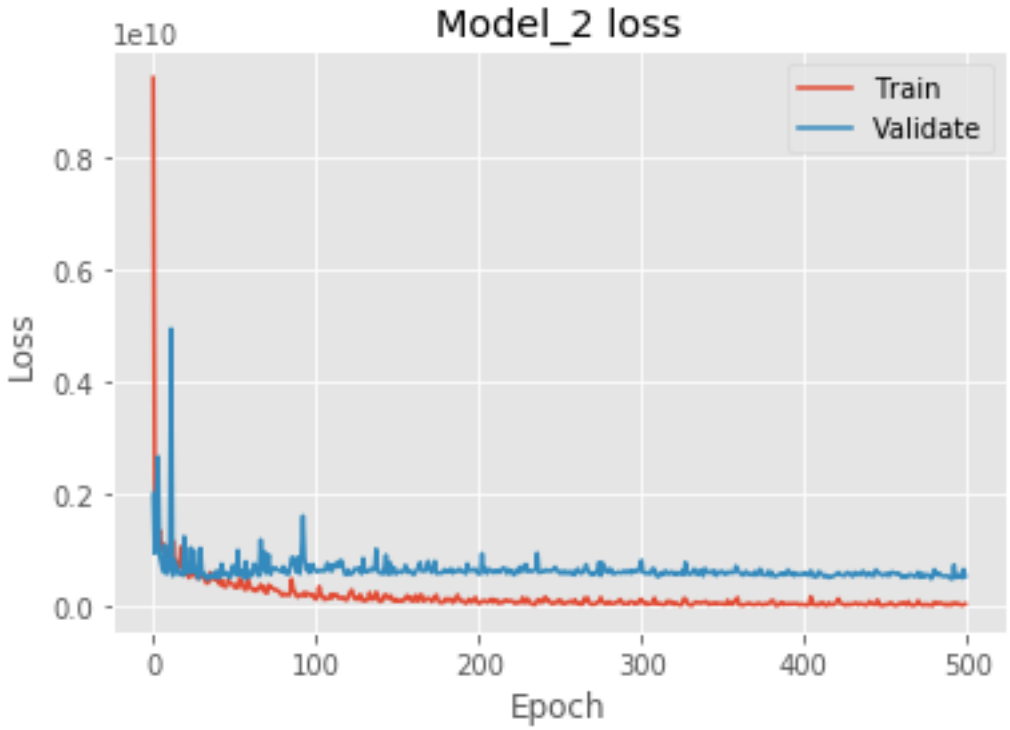
Model Name	MAE	MSE
Model 1	20710	30263
Model 2	18973	30630
Model 3	20312	29701
Model 4	19594	30233
Model 5	20655	32437
Linear Regression	22089	43507
Support Vector Machine	52684	74236

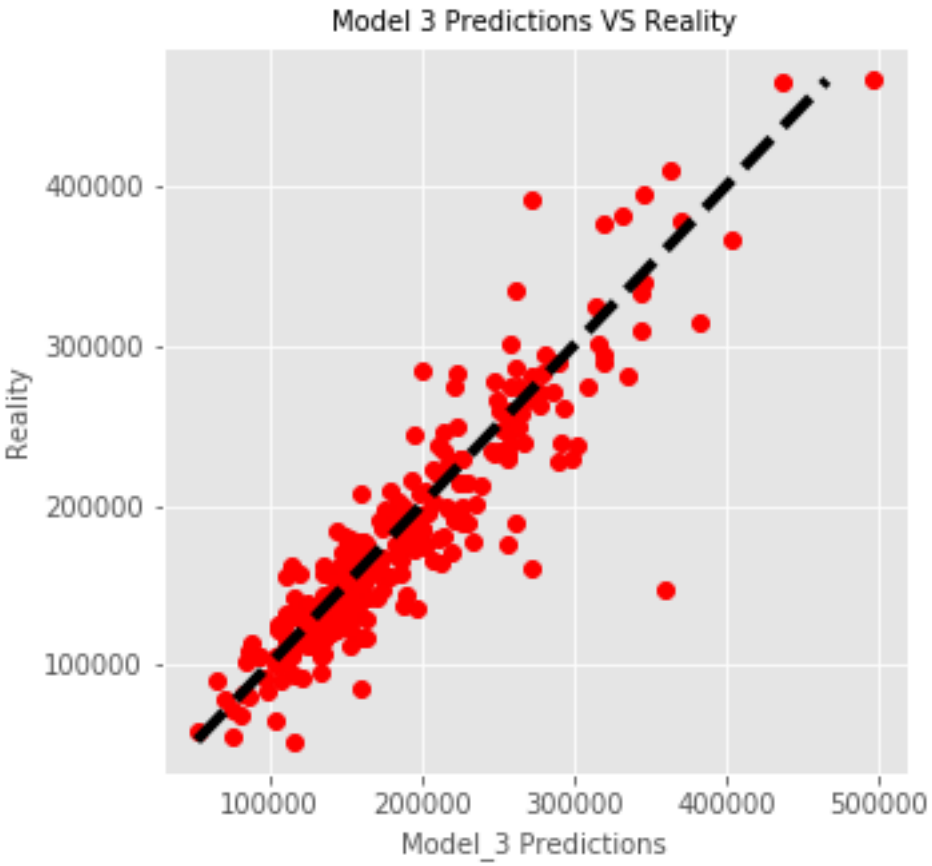
In Model 2 and 3, I achieved the best results of 18K MAE and 30K MSE and in general, neural network presented a better result than linear regression and support vector machine. However, if given more data and more features, I am confident neural network will even produce a better result.

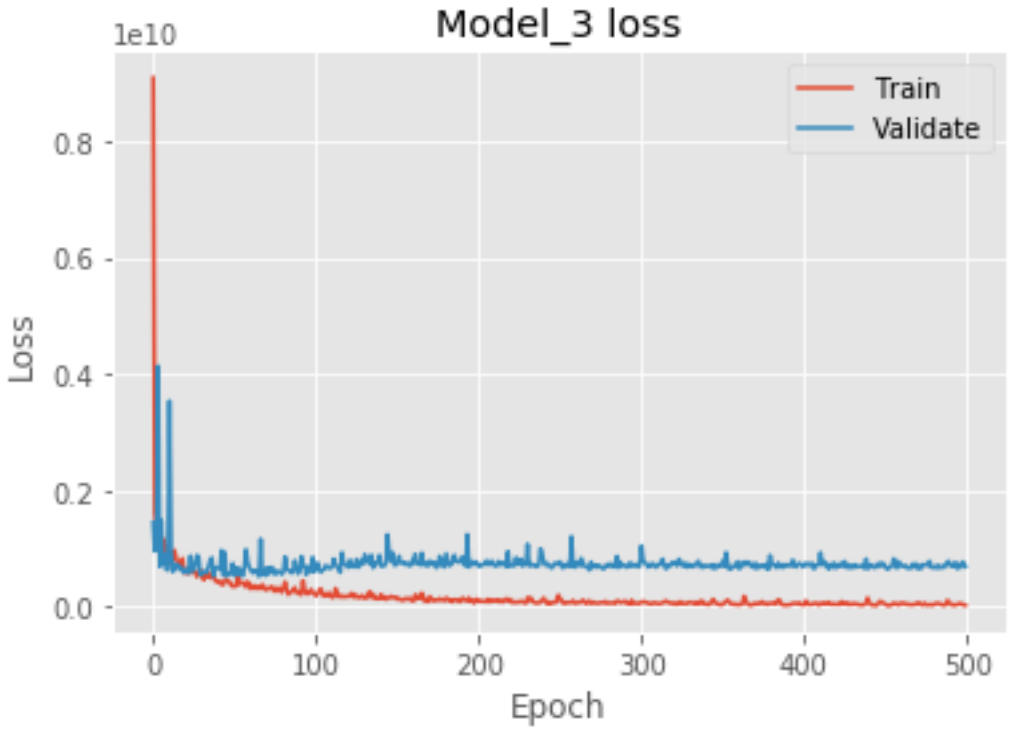

Model 1 Predictions VS Reality
Reality
400000
300000
200000
100000
100000
200000
300000
400000
Model_1 Predictions

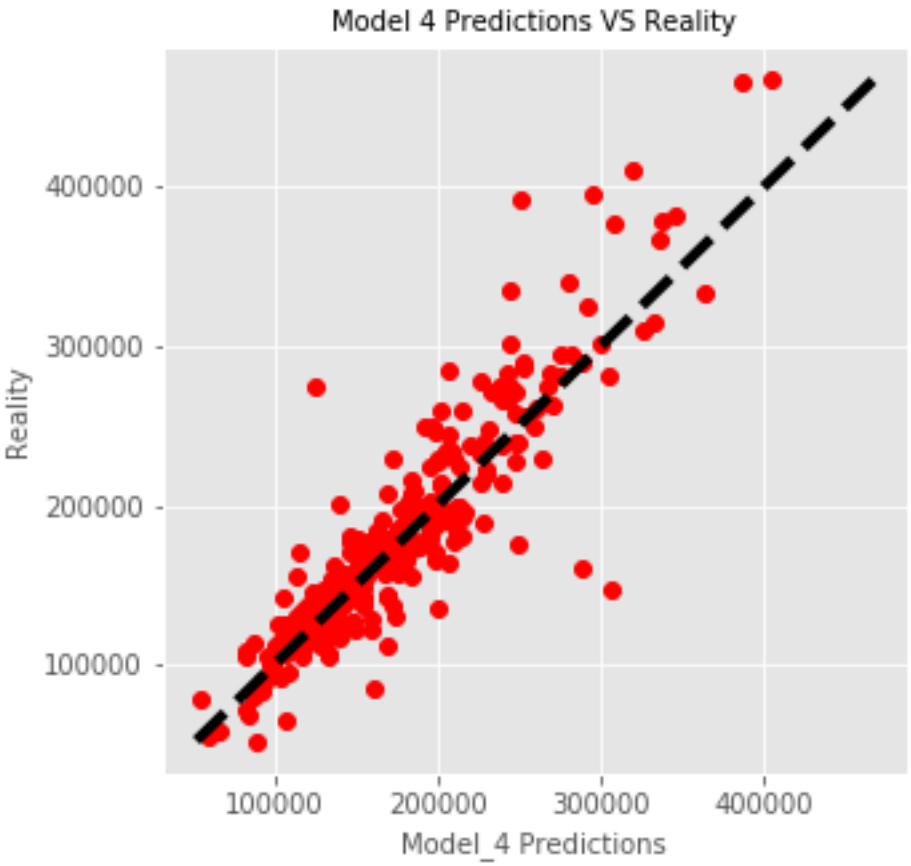

Model_1 loss
1e10
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Loss
Train
Validate
0
100
200
300
400
500
Epoch

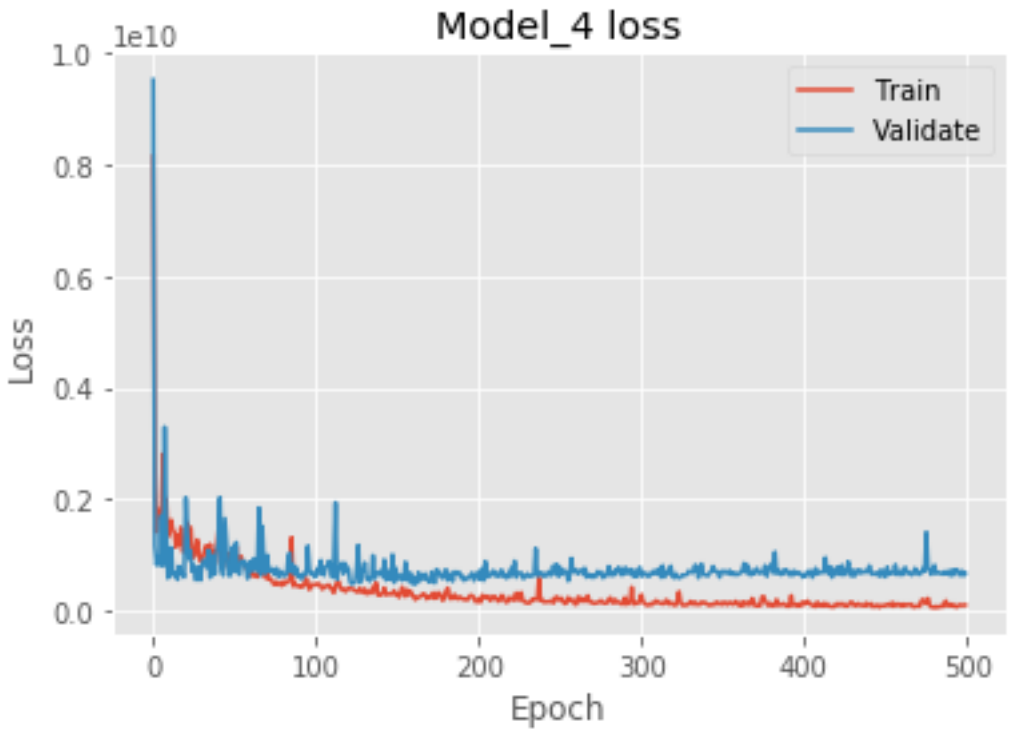

Model 2 Predictions VS Reality
Reality
400000
300000
200000
100000
100000
200000
300000
400000
500000
Model_2 Predictions

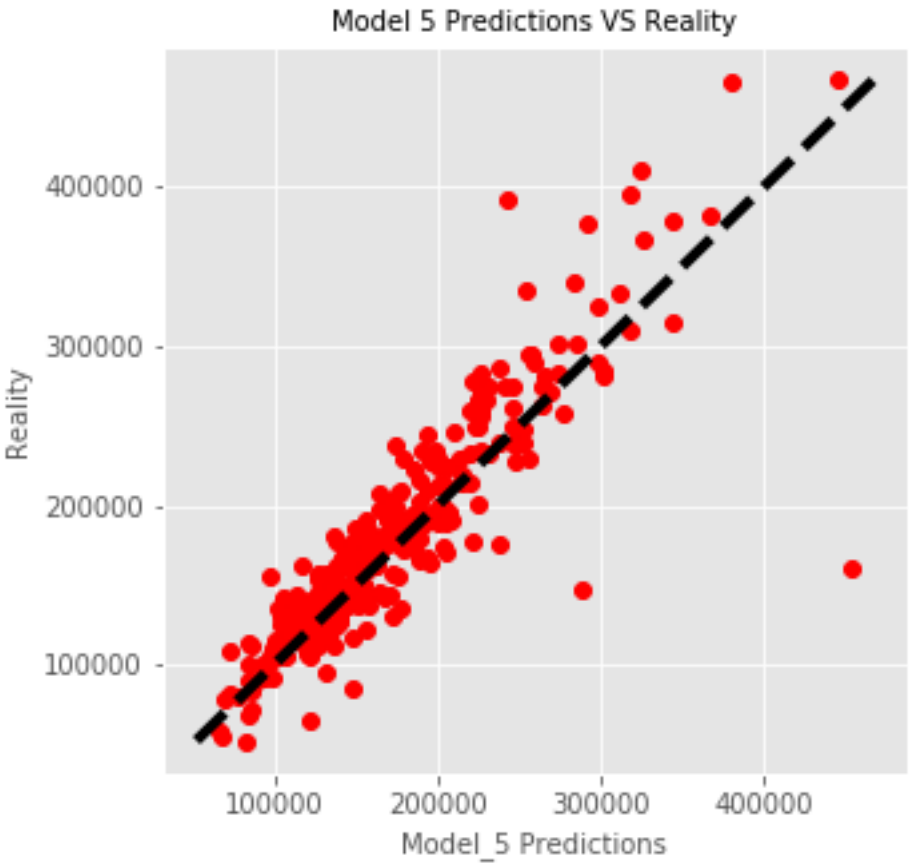

Model_2 loss
1e10
Train
Validate
Loss
0.8
0.6
0.4
0.2
0.0
0
100
200
300
400
500
Epoch

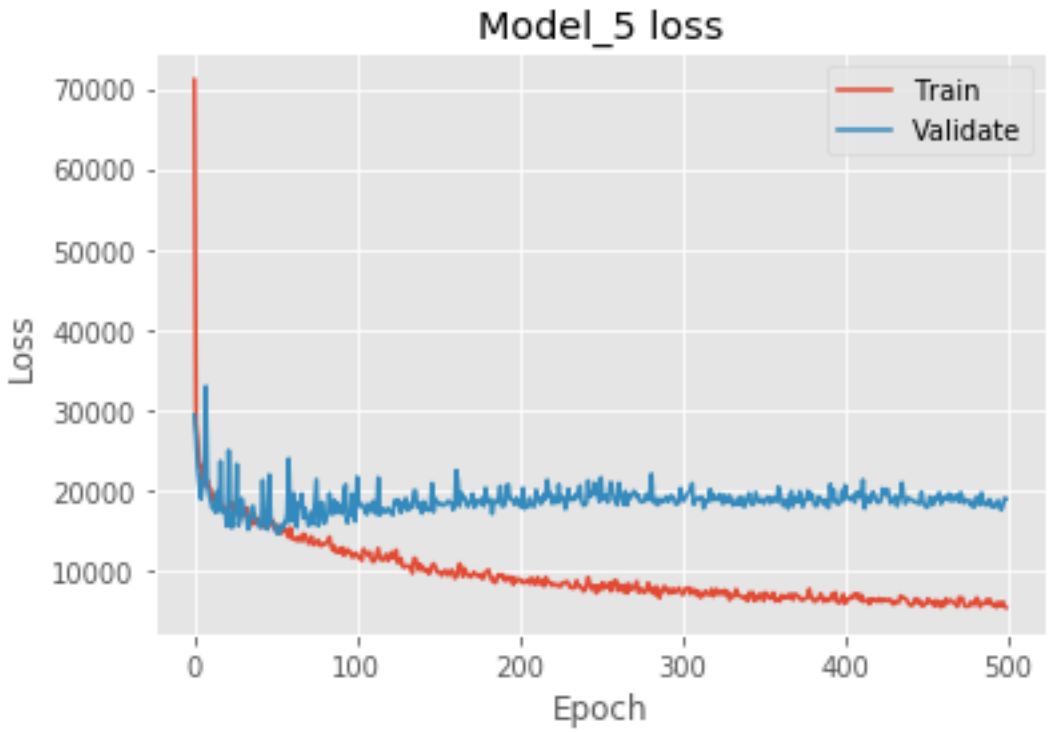

Model 3 Predictions VS Reality
Reality
400000
300000
200000
100000
100000
200000
300000
400000
500000
Model_3 Predictions

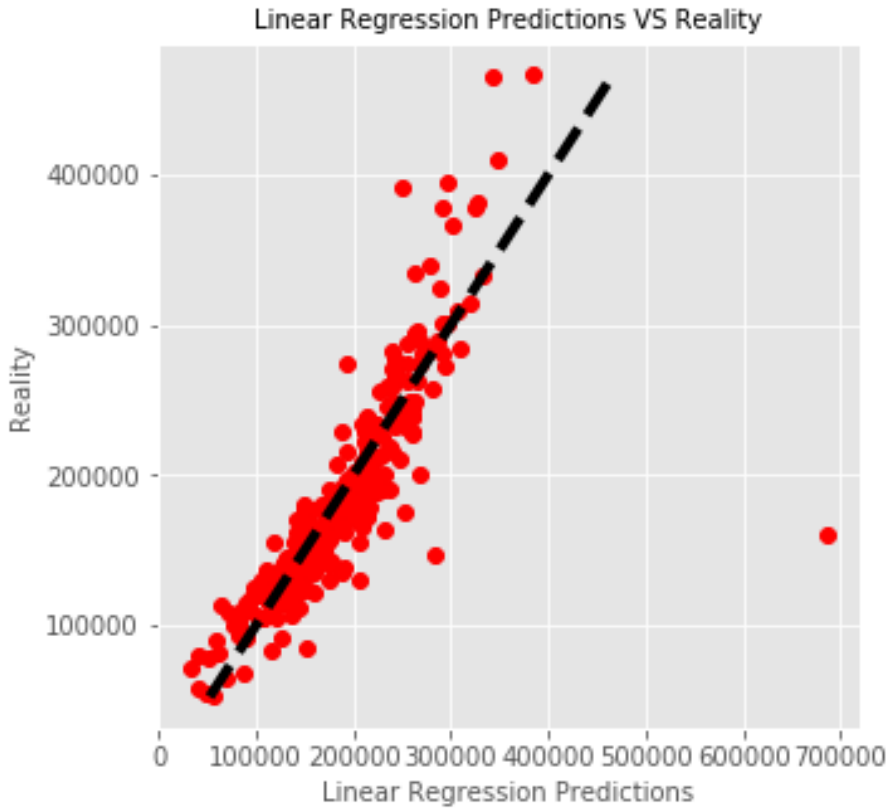

Model_3 loss
1e10
Train
Validate
Loss
0.8
0.6
0.4
0.2
0.0
0
100
200
300
400
500
Epoch

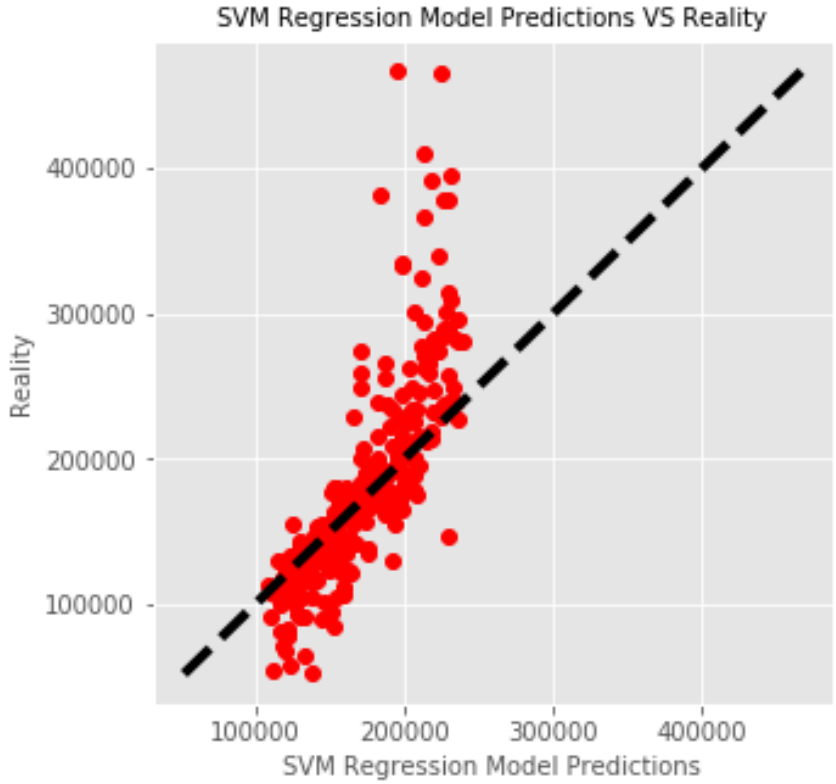

Model 4 Predictions VS Reality
Reality
400000
300000
200000
100000
100000
200000
300000
400000
Model_4 Predictions


Model_4 loss
1e10
1.0
0.8
0.6
0.4
0.2
0.0
Loss
Train
Validate
0
100
200
300
400
500
Epoch


Model 5 Predictions VS Reality
Reality
400000
300000
200000
100000
100000
200000
300000
400000
Model_5 Predictions


Model_5 loss
Train
Validate
Loss
70000
60000
50000
40000
30000
20000
10000
0
100
200
300
400
500
Epoch


Linear Regression Predictions VS Reality
Reality
400000
300000
200000
100000
0
100000
200000
300000
400000
500000
600000
700000
Linear Regression Predictions


SVM Regression Model Predictions VS Reality
Reality
400000
300000
200000
100000
100000
200000
300000
400000
SVM Regression Model Predictions

- **Conclusion**: Tuning the network is a complicated process. Many random factors may affect the final result, for example initial weights and biases which decide where we start on the gradient hill. Also, in the 5 models, I used Adam optimizer which results a changing learning rate. Batch size is another factor as well. But in general, we can draw the following conclusions:
    - Deeper neural network has more learning capacity and can generate better results if given more trainings and data
    - Loss on neural network with bigger learning rate would converge quicker but also less stable. The final result is worse than the ones with smaller learning rate, especially when it gets to the minimal points.
    - Learning rate increase will make our model converges faster but also make the losses more unstable when it reaches the target. That's why in many latest researches, they introduced decay factor.
    - For regression problems, mean squared errors are easier to solve than mean absolute errors, resulting in better results. But mean squared errors are less resistant to outlier data.
    - Comparing to Linear regression and SVM models, in this case, the Neural network results are better than them. But given the flexibility, further improvement is quite possible. The more data and feature we have; the better results neural network will generate.

Python Code

See below and check file attached *Assignment_2_NN.py* and *Assignment_1.py*

```
use Task_1_ECE9014;
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Thu Oct 25 22:28:53 2018
@author: chunyangjia
"""

# !/usr/bin/env python3
# -*- coding: utf-8 -*-

"""
Created on Mon Oct 22 14:01:58 2018

@author: chunyangjia
"""

# 1. Read in the data and check correlation
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

train_path =
'/Users/chunyangjia/Desktop/Western_University/ECE9063A/Assignment_2/housing_price/train.csv'
df_data = pd.read_csv(train_path)
print(df_data.shape)
print(df_data['SalePrice'].describe())
sns.distplot(df_data['SalePrice'])
plt.show()

# 2. Feature Selection and pre-data modification
df_corr = df_data.corr()
df_data.index = df_data['Id']
df_data = df_data[['LotFrontage','LotArea','OverallQual','OverallCond',
              'YearBuilt','YearRemodAdd','TotalBsmtSF','1stFlrSF','2ndFlrSF',
              'GrLivArea','FullBath','BedroomAbvGr','KitchenAbvGr',
              'TotRmsAbvGrd','Fireplaces','GarageCars','WoodDeckSF',
              'PoolArea','YrSold','SalePrice']]
df_data.fillna(0,inplace = True)

# 3. Removing outliers and filling Nan values
df_data.fillna(0,inplace = True)
for i in range (1,1460):
    if df_data['SalePrice'][i] > 500000:
        print(2)
        df_data.drop(i, inplace = True)


# 4. training and testing sets split
X = df_data.loc[:,'LotFrontage':'YrSold']
y = df_data['SalePrice']

```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,shuffle =
    False)

# 5.Data preprocessing
from sklearn.preprocessing import StandardScaler

scalerX = StandardScaler().fit(X_train)
X_train = scalerX.transform(X_train)
X_test = scalerX.transform(X_test)


#6.Model Training and Tuning
from keras import models
from keras import layers
from keras import losses
from keras import optimizers

#Model_1 training
Model_1 = models.Sequential()
Model_1.add(layers.Dense(19, input_dim= 19))
for i in range(0,10):
    Model_1.add(layers.Dense(60, activation = 'relu'))
Model_1.add(layers.Dense(1))
Model_1.compile(optimizer='adam', loss = losses.mean_squared_error)
Model_1.summary()
history1 = Model_1.fit(X_train, y_train,validation_split=0.1,
                       epochs = 50, batch_size = 4)
y1_predict = Model_1.predict(X_test)
Model_1.save('Model_1.h5')

#Model_2 trainig
Model_2 = models.Sequential()
Model_2.add(layers.Dense(19, input_dim= 19))
for i in range(0,15):
    Model_2.add(layers.Dense(60, activation = 'relu'))
Model_2.add(layers.Dense(1))
Model_2.compile(optimizer = 'adam', loss = losses.mean_squared_error)
Model_2.summary()
history2 = Model_2.fit(X_train, y_train,validation_split = 0.1,
                       epochs = 50, batch_size = 4)
y2_predict = Model_2.predict(X_test)
Model_2.save('Model_2.h5')

# Model_3 trainig
Model_3 = models.Sequential()
Model_3.add(layers.Dense(19, input_dim= 19))
for i in range(0,10):
    Model_3.add(layers.Dense(90, activation = 'relu'))
Model_3.add(layers.Dense(1))
Model_3.compile(optimizer = 'adam', loss = losses.mean_squared_error)
Model_3.summary()
history3 = Model_3.fit(X_train, y_train,validation_split = 0.1,
                       epochs = 50, batch_size = 4)
y3_predict = Model_3.predict(X_test)
Model_3.save('Model_3.h5')

# Model_4 training
Model_4 = models.Sequential()
Model_4.add(layers.Dense(19, input_dim= 19))
for i in range(0,10):
```

```
    Model_4.add(layers.Dense(60, activation = 'tanh'))
Model_4.add(layers.Dense(1))
adam = optimizers.Adam(lr = 0.005)
Model_4.compile(optimizer = adam, loss = losses.mean_squared_error)
Model_4.summary()
history4 = Model_4.fit(X_train, y_train,validation_split = 0.1,
                       epochs = 50, batch_size = 4)
y4_predict = Model_4.predict(X_test)
Model_4.save('Model_4.h5')

# Model_5 trainig
Model_5 = models.Sequential()
Model_5.add(layers.Dense(19, input_dim= 19))
for i in range(0,10):
    Model_5.add(layers.Dense(60, activation = 'relu'))
Model_5.add(layers.Dense(1))
Model_5.compile(optimizer = 'adam', loss = losses.mean_absolute_error)
Model_5.summary()
history5 = Model_5.fit(X_train, y_train,validation_split = 0.1,
                       epochs = 50, batch_size = 8)
y5_predict = Model_5.predict(X_test)
Model_5.save('Model_5.h5')



from sklearn import metrics
print("RMSE for NN Model 1: ", np.sqrt(metrics.mean_squared_error(y_test,y1_predict)))
print("MAE for NN Model 1: ", metrics.mean_absolute_error(y_test,y1_predict))

print("RMSE for NN Model 2: ", np.sqrt(metrics.mean_squared_error(y_test,y2_predict)))
print("MAE for NN Model 2: ", metrics.mean_absolute_error(y_test,y2_predict))

print("RMSE for NN Model 3: ", np.sqrt(metrics.mean_squared_error(y_test,y3_predict)))
print("MAE for NN Model 3: ", metrics.mean_absolute_error(y_test,y3_predict))

print("RMSE for NN Model 4: ", np.sqrt(metrics.mean_squared_error(y_test,y4_predict)))
print("MAE for NN Model 4: ", metrics.mean_absolute_error(y_test,y4_predict))

print("RMSE for NN Model 5: ", np.sqrt(metrics.mean_squared_error(y_test,y5_predict)))
print("MAE for NN Model 5: ", metrics.mean_absolute_error(y_test,y5_predict))

plt.plot(history1.history['loss'])
plt.plot(history1.history['val_loss'])
plt.title('Model_1 loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validate'], loc='upper right')
plt.show()

fig, ax = plt.subplots(figsize = (5,5))
plt.style.use('ggplot')
plt.plot(y1_predict,y_test,'ro')
plt.xlabel('Model_1 Predictions', fontsize = 10)
plt.ylabel('Reality', fontsize = 10)
plt.title('Model 1 Predictions VS Reality', fontsize = 10)
ax.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--', lw=4)
plt.show()

plt.plot(history2.history['loss'])
plt.plot(history2.history['val_loss'])
plt.title('Model_2 loss')
```

```
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validate'], loc='upper right')
plt.show()

fig, ax = plt.subplots(figsize = (5,5))
plt.style.use('ggplot')
plt.plot(y2_predict,y_test,'ro')
plt.xlabel('Model_2 Predictions', fontsize = 10)
plt.ylabel('Reality', fontsize = 10)
plt.title('Model 2 Predictions VS Reality', fontsize = 10)
ax.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--', lw=4)
plt.show()

plt.plot(history3.history['loss'])
plt.plot(history3.history['val_loss'])
plt.title('Model_3 loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validate'], loc='upper right')
plt.show()

fig, ax = plt.subplots(figsize = (5,5))
plt.style.use('ggplot')
plt.plot(y3_predict,y_test,'ro')
plt.xlabel('Model_3 Predictions', fontsize = 10)
plt.ylabel('Reality', fontsize = 10)
plt.title('Model 3 Predictions VS Reality', fontsize = 10)
ax.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--', lw=4)
plt.show()

plt.plot(history4.history['loss'])
plt.plot(history4.history['val_loss'])
plt.title('Model_4 loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validate'], loc='upper right')
plt.show()

fig, ax = plt.subplots(figsize = (5,5))
plt.style.use('ggplot')
plt.plot(y4_predict,y_test,'ro')
plt.xlabel('Model_4 Predictions', fontsize = 10)
plt.ylabel('Reality', fontsize = 10)
plt.title('Model 4 Predictions VS Reality', fontsize = 10)
ax.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--', lw=4)
plt.show()

plt.plot(history5.history['loss'])
plt.plot(history5.history['val_loss'])
plt.title('Model_5 loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validate'], loc='upper right')
plt.show()

fig, ax = plt.subplots(figsize = (5,5))
plt.style.use('ggplot')
plt.plot(y5_predict,y_test,'ro')
plt.xlabel('Model_5 Predictions', fontsize = 10)
plt.ylabel('Reality', fontsize = 10)
```

```
plt.title('Model 5 Predictions VS Reality', fontsize = 10)
ax.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--', lw=4)
plt.show()
```

#Python Code for linear regression and SVM Models

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Mon Oct 22 14:01:58 2018

@author: chunyangjia
"""

import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split

#bring in the data and make the id into index
train_path =
    '/Users/chunyangjia/Desktop/Western_University/ECE9063A/Assignment_2/housing_price/trai
   n.csv'
df_data = pd.read_csv(train_path)
print (df_data.head())
df_corr = df_data.corr()
############################################################

df_data.index = df_data['Id']
df_data = df_data[['LotFrontage','LotArea','OverallQual','OverallCond',
               'YearBuilt','YearRemodAdd','TotalBsmtSF','1stFlrSF','2ndFlrSF',
               'GrLivArea','FullBath','BedroomAbvGr','KitchenAbvGr',
               'TotRmsAbvGrd','Fireplaces','GarageCars','WoodDeckSF',
               'PoolArea','YrSold','SalePrice']]
df_data.fillna(0,inplace = True)
print (df_data.head())

for i in range (1,1460):
    if df_data['SalePrice'][i] > 500000:
        print(2)
        df_data.drop(i, inplace = True)

#training and testing sets split
X = df_data.loc[:,'LotFrontage':'YrSold']
y = df_data['SalePrice']

from sklearn.preprocessing import StandardScaler
scalerx = StandardScaler()
#scalery = StandardScaler()
```

```
X = scalerx.fit_transform(X)
#y = scalery.fit_transform(y)

X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.2,shuffle =
    False)


############################################################
#Linear Regression

from sklearn.linear_model import LinearRegression
Model_LR = LinearRegression()
Model_LR.fit(X_train,y_train)

y_LR_pred = Model_LR.predict(X_test)

from sklearn import metrics

print("RMSE for linear regression: ",
    np.sqrt(metrics.mean_squared_error(y_test,y_LR_pred)))
print("MAE for linear regression: ", metrics.mean_absolute_error(y_test,y_LR_pred))
#df_linear = pd.DataFrame({'Actual':y_test, 'Linear Regression Predicted':y_pred})

#print(df_linear)
############################################################

from sklearn import svm

SVM_model = svm.SVR(C = 1000, epsilon = 0.0001)
SVM_model.fit(X_train,y_train)

y_svm_pred = SVM_model.predict(X_test)

print("RMSE for SVR: ", np.sqrt(metrics.mean_squared_error(y_test,y_svm_pred)))
print("MAE for SVR: ", metrics.mean_absolute_error(y_test,y_svm_pred))
df_SVM = pd.DataFrame({'Actual':y_test, 'SVM_Predicted':y_svm_pred,'Linear Regression
    Predicted':y_LR_pred})


############################################################

import matplotlib.pyplot as plt

fig, ax = plt.subplots(figsize = (5,5))
plt.style.use('ggplot')
plt.plot(y_LR_pred,y_test,'ro')
plt.xlabel('Linear Regression Predictions', fontsize = 10)
plt.ylabel('Reality', fontsize = 10)
plt.title('Linear Regression Predictions VS Reality', fontsize = 10)
ax.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--', lw=4)
plt.show()

fig, ax = plt.subplots(figsize = (5,5))
plt.style.use('ggplot')
plt.plot(y_svm_pred,y_test,'ro')
plt.xlabel('SVM Regression Model Predictions', fontsize = 10)
plt.ylabel('Reality', fontsize = 10)
plt.title('SVM Regression Model Predictions VS Reality', fontsize = 10)
ax.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--', lw=4)
plt.show()

```