AI ANN Regressor Assignment 2
Xiao(Megan)Ling

1. **Object**
   We are going to use UCI red wine dataset predict red wine quality using an Artificial Neural Network. With given data points about Portuguese "Vinho Verde" wine, such as fixed acidity, residual sugar, etc., we will be able to determine wine qualities based on a model designed.

2. **The Final Model & Training Algorithm**

```
def build_model():
    model = Sequential()
    model.add(Dense(12, activation='relu', input_shape=(x_train.shape[1],)))
    model.add(Dense(12, activation='relu'))
    model.add(Dense(12, activation='relu'))
    model.add(Dense(1))
    model.compile(loss='mean_squared_error', optimizer='rmsprop', metrics=['mae'])
    return model

"""
.
"""
Final Model Application
"""
model = build_model()
test_mae_score = []
model.fit(x_train, y_train, epochs=80, batch_size=16, verbose=0)
test_mse_score, test_mae_score = model.evaluate(x_test, y_test)
print(test_mae_score)
```

The Final Model has 3 hidden layers using RMSprop as model compile optimizer and MAE as metrics.
The training algorithm here is Artificial Neural Network Algorithms which consists of different layers with weighted connections for analyzing and learning data.

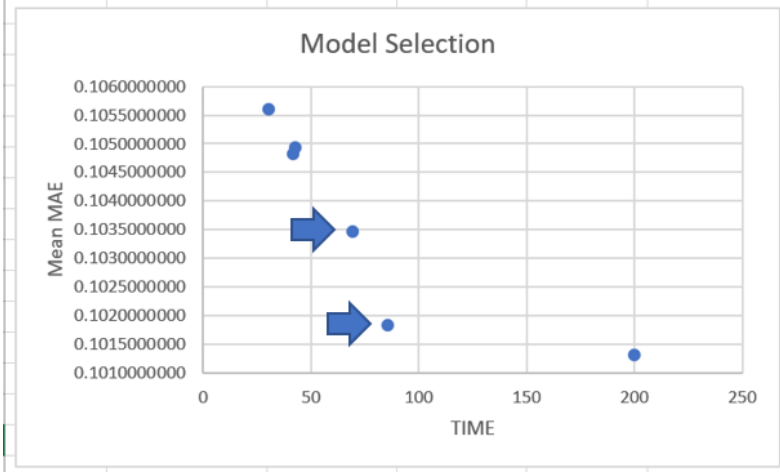
3. **The experimental plan for arriving at the final model and 5. How long it took to run all the models in your experimental plan**

To arrive at the final model, I changed parameters such as the number of Epochs, the size of the hidden layers, and the number of partitions K.

First, I designed model with RMSprop as compile optimizers, K=10, with Epochs = 500 and 2 hidden layers. The RMSprop is a good choice for a recurrent neural network. I applied the same parameters on different model optimizers, Nadam and Adam. I chose Adam because it quires low memory and Nadam is Adam RMSprop with Nesterov momentum.

Then I changed parameters tried K=5, Epochs =200 to see if the MAE would decrease. As the chart shows below that the Mean MAE from each training session:

K	Num_epochs	Optimizer	? Layers	Run time	Mean MAE	Best MAE	Best Epochs
10	500	Nadam	2	199.72157	0.1013023757	0.098494790	391
10	500	RMSprop	2	85.664288	0.1018276161	0.097885619	266
10	200	Adam	2	69.444748	0.1034636472	0.099549006	254
5	200	Nadam	2	41.504548	0.1048140109	0.098500708	176
5	200	RMSprop	2	42.583306	0.1049321602	0.099982675	163
5	200	Adam	2	30.459435	0.1055982604	0.099820368	193

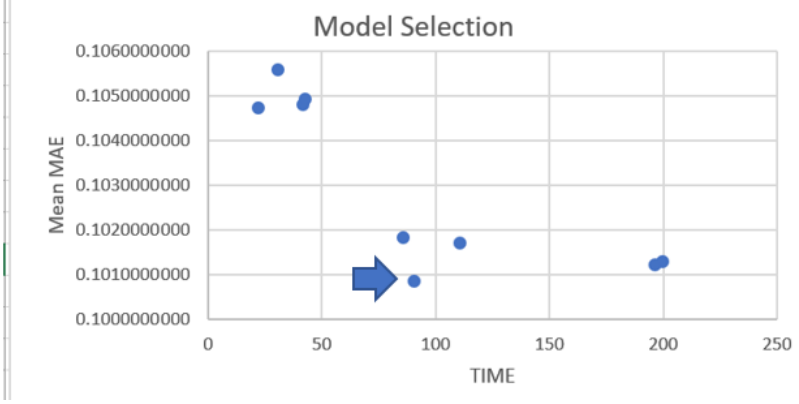


According to the results, the Best Mean MAE appears when using Nadam Optimizer with K=10 and 500 epochs. However, combined with time spend running the training model, it runs too slow compared with other training models from Adam and RMSprop has similar Mean MAE level.

To save running time, I set the number of epochs to 200, but the Mean MAE dropped dramatically. Then I ran RMSprop and Adam models with parameters K=10, Epochs =500 with 2/3 neural layers. The Scatter plot shows that based on a time efficiency rule, when K=10, Epochs =500, 3 neural layers, RMSprop has the best performance in the training data.

As a result, I selected K=10, Epochs =500, 3 neural layers, as my parameters for the final RMSprop model.

K	Num_epochs	Optimizer	? Layers	Run time	Mean MAE	Best MAE	Best Epochs
10	**500**	**RMSprop**	**3**	**90.391225**	**0.1008604196**	**0.097160457**	**129**
10	500	Adam	3	196.11444	0.1012200484	0.098546245	94
10	500	Nadam	2	199.72157	0.1013023757	0.098494790	391
10	500	Adam	2	110.58813	0.1017054446	0.098623392	286
10	**500**	**RMSprop**	**2**	**85.664288**	**0.1018276161**	**0.097885619**	**266**
10	200	Adam	3	22.08059	0.1047363839	0.099848599	190
5	200	Nadam	2	41.504548	0.1048140109	0.098500708	176
5	200	RMSprop	2	42.583306	0.1049321602	0.099982675	163
5	200	Adam	2	30.459435	0.1055982604	0.099820368	193



Run Test Result Using	
RMSprop	
K = 10, 3 Layers	
Num_Epochs = 500	
Test MAE	**0.097160457**

6. An explanation of the input variables and any preprocessing steps you took

I input data through “pd.read_csv” to load in red wine-quality data. Since we are going to predict the wine quality, I isolate the quality labels from the rest of the data set in “y_vars” and the rest into “x_vars.”

```
"""
Load Data Section
"""
start = time.time()

data = pd.read_csv("winequality-red.csv",
                   delimiter=";", header=0,
                   names = ['f_acidity', 'v_acidity', 'citric Acid', 'r_sugar',
                            'chlorides', 'free_so2',
                            'total_so2', 'density', 'pH',
                            'sulphates', 'alcohol', 'quality'])

x_vars = ['f_acidity', 'v_acidity', 'citric Acid', 'r_sugar', 'chlorides',
          'free_so2','total_so2', 'density', 'pH', 'sulphates', 'alcohol']

y_vars = ['quality']
```

In the data pretreat step, I preprocess X and Y data before fitting into predictor. It will convert unscaled data into a given range. Then the data was split into train and test set into a 70/30 proportion through “train_test_split” function. To make sure the axis along which the means are computed properly, the training and test dataset mean on axis 0 will be the mean of all the rows in each column.

```
"""
Pretreat Data Section
"""
x_data = data[x_vars]
y_data = data[y_vars]

print (len(x_data))
print (len(y_data))

x_data_MinMax = preprocessing.MinMaxScaler()
y_data_MinMax = preprocessing.MinMaxScaler()

x_data.values
x_data = np.array(x_data).reshape((len(x_data), 11))

y_data.values
y_data = np.array(y_data).reshape((len(y_data), 1))

x_data = x_data_MinMax.fit_transform(x_data)
y_data = y_data_MinMax.fit_transform(y_data)

x_train, x_test, y_train, y_test = train_test_split(x_data, y_data, test_size=0.33, random_state=7)

x_train.mean(axis=0)
y_train.mean(axis=0)
x_test.mean(axis=0)
y_test.mean(axis=0)
print(x_train.shape)
```

7. **An explanation of your metrics and justification for your choice**

```
"""
Define Model Section - RMSPROP
"""
def build_model():
    model = Sequential()
    model.add(Dense(12, activation='relu', input_shape=(x_train.shape[1],)))
    model.add(Dense(12, activation='relu'))
    model.add(Dense(1))
    model.compile(loss='mean_squared_error', optimizer='rmsprop', metrics=['mae'])
    return model
```

After preparing data and splitting train and test sets, it's easy to find out the training data with a shape of 1071 rows and 11 columns by "x_train.shape." To prevent overfitting, I selected a very small network with three hidden layers, each with 12 units. The network ends with a single unit and no activation. Since we are building up a regressor model, the network is free to learn to predict values in any range to generate a linear result. Generally, the MSE loss function is wildly applied when compiling the network. To monitor the training result, MAE metric is introduced to show the difference between the predictions and the targets.

8. **An explanation of your method to validate the model**

```
"""
K-fold validation
"""
k = 10
num_val_samples = len(x_train)// k

num_epochs = 500
all_mae_histories = []
for i in range(k):
    print('processing fold #', i)
    val_data = x_train[i * num_val_samples: (i + 1) * num_val_samples]
    val_targets = y_train[i * num_val_samples: (i + 1) * num_val_samples]
    partial_x_train = np.concatenate([x_train[:i * num_val_samples],
                                      x_train[(i + 1) * num_val_samples:]],axis=0)
    partial_y_train = np.concatenate([y_train[:i * num_val_samples],
                                      y_train[(i + 1) * num_val_samples:]],axis=0)
    model = build_model()
    history = model.fit(partial_x_train, partial_y_train,
                        validation_data=(val_data, val_targets),
                        epochs=num_epochs, batch_size=50, verbose=0)
    mae_history = history.history['val_mean_absolute_error']
    all_mae_histories.append(mae_history)

average_mae_history = [np.mean([x[i] for x in all_mae_histories]) for i in range(num_epochs)]
```

To evaluate my network while keep adjusting its parameters, data was also split into a training set and a validation set. The best solution in such solution is to use K-Fold cross-validation. Normally, K is suggested equal to 5 or 10. The K-Fold training method is splitting the available training data into K partitions, implementing K identical models, and training each one on K-1 partitions while evaluating on the remaining partition. After validating each model selected, I applied the final production model on all of the training data with the best model.

9. **Your result in terms of appropriate metrics for the objective and problem**

Mean Absolute Error measures the average magnitude of the errors in a set of predictions. Since MSE already worked as the loss function, it's prevalent to take MAE as a metric. In the red wine quality prediction scenario, MAE quantifies how close predictions are to the eventual outcomes.
The final test MAE result of the final model is 0.097, which means nearly 91% of the wine quality predictions are correct.

A copy of my code

```
# -*- coding: utf-8 -*-
"""
Created on Tue Feb 19 10:37:29 2019

@author: megan
"""
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
Import Libraries Section
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
from keras.models import Sequential
from keras.layers import Dense
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import time

import sys
if "C:\\My_Python_Lib" not in sys.path:
    sys.path.append("C:\\My_Python_Lib")
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
Load Data Section
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
start = time.time()

data = pd.read_csv("winequality-red.csv",
                   delimiter=";", header=0,
                   names = ['f_acidity', 'v_acidity', 'citric Acid', 'r_sugar',
                            'chlorides', 'free_so2',
                            'total_so2', 'density', 'pH',
                            'sulphates', 'alcohol', 'quality'])

x_vars = ['f_acidity', 'v_acidity', 'citric Acid', 'r_sugar', 'chlorides',
          'free_so2','total_so2', 'density', 'pH', 'sulphates', 'alcohol']

y_vars = ['quality']
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
Pretreat Data Section
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
x_data = data[x_vars]
y_data = data[y_vars]

print (len(x_data))
print (len(y_data))

x_data_MinMax = preprocessing.MinMaxScaler()
y_data_MinMax = preprocessing.MinMaxScaler()

x_data.values
x_data = np.array(x_data).reshape((len(x_data), 11))
```

```
y_data.values
y_data = np.array(y_data).reshape((len(y_data), 1))

x_data = x_data_MinMax.fit_transform(x_data)
y_data = y_data_MinMax.fit_transform(y_data)

x_train, x_test, y_train, y_test = train_test_split(x_data, y_data, test_size=0.33, random_state=7)

x_train.mean(axis=0)
y_train.mean(axis=0)
x_test.mean(axis=0)
y_test.mean(axis=0)
print(x_train.shape)
"""
Define Model Section - RMSPROP
"""
def build_model():
    model = Sequential()
    model.add(Dense(12, activation='relu', input_shape=(x_train.shape[1],)))
    model.add(Dense(12, activation='relu'))
    model.add(Dense(1))
    model.compile(loss='mean_squared_error', optimizer='rmsprop', metrics=['mae'])
    return model

"""
K-fold validation
"""
k = 10
num_val_samples = len(x_train)// k

num_epochs = 500
all_mae_histories = []
for i in range(k):
    print('processing fold #', i)
    val_data = x_train[i * num_val_samples: (i + 1) * num_val_samples]
    val_targets = y_train[i * num_val_samples: (i + 1) * num_val_samples]
    partial_x_train = np.concatenate([x_train[:i * num_val_samples],
                                      x_train[(i + 1) * num_val_samples:]],axis=0)
    partial_y_train = np.concatenate([y_train[:i * num_val_samples],
                                      y_train[(i + 1) * num_val_samples:]],axis=0)
    model = build_model()
    history = model.fit(partial_x_train, partial_y_train,
                        validation_data=(val_data, val_targets),
                        epochs=num_epochs, batch_size=50, verbose=0)
    mae_history = history.history['val_mean_absolute_error']
    all_mae_histories.append(mae_history)

average_mae_history = [np.mean([x[i] for x in all_mae_histories]) for i in range(num_epochs)]
end = time.time()
print(end - start)

"""
Show & Plot output Section
"""
print(np.mean(average_mae_history))
```

```
print(min(average_mae_history))
print('Epoch for min Mae:', average_mae_history.index(min(average_mae_history)))

plt.plot(range(1, len(average_mae_history) + 1), average_mae_history)
plt.xlabel('Epochs')
plt.ylabel('Validation MAE')
plt.show()

""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
Final Model Application
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
model = build_model()
test_mae_score = []
model.fit(x_train, y_train, epochs=80, batch_size=16, verbose=0)
test_mse_score, test_mae_score = model.evaluate(x_test, y_test)
print(test_mae_score)
```