1. The objective of this artificial neural network is to classify the quality of a red wine on an ordinal scale represented by integers from 0 to 10.
2.

```
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
Import Libraries Section
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
from matplotlib import pyplot
from keras import models
from keras import layers
#from keras.utils import to_categorical
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
from datetime import timedelta
import datetime
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
Load Data Section
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
#Fix random weights for repeatability
np.random.seed(7)

#Import the data
dataframe=pd.read_csv('C:\\Users\\chery\\Grad School\\BUAD 5802 -AI II\\M3-
Intermediate Feed-Forward Neural Networks\\Submission\\winequality-
white.csv',delimiter=";")
dataframe=dataframe.replace(np.nan,0)
dataset=dataframe.values

#print ("xTrain.shape",xTrain.shape)
#print ("len(yTrain)",len(yTrain))
#print("yTrain_labels",yTrain)

#print ("xTest.shape",xTest.shape)
#print ("len(yTest)",len(yTest))
#print("yTest_labels",yTest)

# Start the timer for run time
start_time = datetime.datetime.now()
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
Parameters Section
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
#Set model parameters
num_nodes_first = 11
```

```
num_nodes_second = 11
num_classes = 11
num_epochs = 13 #Test loss flatlines after about 13 epochs
size_batch = 50 #The weights will be updated after ever 50 data points, default is 1 and will take longer to run
verbose_setting=1

""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
Pretreat Data Section
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
#Separate variables
X=dataset[:,0:11]
Y=dataset[:,11]

#Encode YCat to represent an ordinal variable
YCat=np.empty(shape=(4898,11))
for i in range(len(Y)):
    ext=np.empty(shape=(1,11))
    for j in range(11):
        if j < Y[i]:
            ext[0][j]=1
        else:
            ext[0][j]=0
    YCat[i]=ext

#Split into training and test data
xTrain, xTest, yTrain, yTest = train_test_split(X, YCat, test_size = 0.2, random_state = 0)

""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
Define Model Section
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
#Create the feedforward  netowrk
network = models.Sequential()
#Add the first hidden layer specifying the input shape
network.add(layers.Dense(num_nodes_first, activation='linear'))
#Add a second hidden layer
network.add(layers.Dense(num_nodes_second, activation='linear'))
#Add another dense layer--the output layer
network.add(layers.Dense(num_classes, activation='sigmoid'))
#Compile the model
network.compile(optimizer='rmsprop',
                loss='categorical_crossentropy',
                metrics=['accuracy'])

""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
Fit Model Section
```

```
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
#Train the model
estimator=network.fit(xTrain, yTrain, epochs = num_epochs, batch_size = size_batch,
verbose = verbose_setting)

""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
Show output Section
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
#Print a summary of the model
network.summary()

#Print the loss and accuracy
test_loss, test_acc = network.evaluate(xTest, yTest)
print('Test Loss:', test_loss)
print('Test Accuracy:', test_acc)

#Plot the loss and accuracy
pyplot.plot(estimator.history['loss'])
pyplot.plot(estimator.history['accuracy'])
pyplot.show()

#Print the time it took to train the model
stop_time = datetime.datetime.now()

#Initialize the cumulative training time calculator
#totalRunTime=timedelta(seconds=0)
run_time=stop_time-start_time
print ("Time required for training:",stop_time - start_time)

totalRunTime=totalRunTime+run_time
```

3. This network uses two hidden layers with 11 hidden units and linear activation with an output layer that has sigmoid activation. The loss function is categorical crossentropy and the optimizer is RMSProp. I used backpropogation through 13 epochs to train the model.
4. I started by separating my data into training and test data. My first approach involved representing the wine quality as a typical categorical variable using two hidden layers which used relu activation with 11 hidden units each and a final layer with softmax activation. This resulted in a test accuracy of not even 50%. I decided I would try a different approach involving representing wine quality as an ordinal variable. This model still had two hidden layers, but they used linear activation and the final layer used sigmoid activation. I trained it through 50 epochs and the RMSProp optimizer with

categorical crossentopy as the loss function. The improvement in accuracy was substantial. I tried limiting the X variable range to between 0 and 1, but found that it did not impact the model well. Then, I tested varying amounts of nodes, but found that 11 gave me the best results. Finally, I reduced the epochs to 13 to avoid overfitting.

5. The total of all model run times was 5 minutes and 55 seconds.
6. The input variables used in the model were chemical components of the wines including fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free sulfur dioxide, total sulfur dioxide, pH, sulphates, and alcohol as well as a measure of density. I processed the wine quality variable by turning each value into an array where each index up to that which represented the quality of the wine quality was 1 and all after were 0.
7. I used categorical crossentropy as the loss function with accuracy as the metric. Accuracy is a simple way to determine how well the model is correctly placing data into each category.
8. The accuracy for the final model was 99.8%--much better than random luck at 9.1%