#This AI model can be used to predict if sonar signals are being bounced against a metal cylinder or rock. The model uses a data file that contains 60 attributes that are obtained

#by bouncing 208 signals from different angles. Each instance then sets a label to either R (rock) or M (metal). The data for these sonar sginals is saved in a CSV file called

#sonar.csv. The program uses methods and classes from Google's tensorflow model instead of creating relu, softmax, optimizer and loss functions from scratch.

#Some Observations about this AI model

# 1. The model currently predicts with about 80-85% accuracy as can be seen from the results shown at the end of this program by comparing predicted and actual vlaues.

# This accuracy can be improved significantly if more data from these sonar signals can be obtained. The most important factor in training an AI model is the availibilty of

# large and precise dataset. The dataset that does not have missing values and also has few ouliers. Since this model is only using a dataset of 208 signals it can be improved

# if more data is made available.

# 2. Models and layers used, can adding more layers and epoch help improve accuracy.

# 3. Loss and accuracy functions used.

# 4. There are print statements in this program that are added just to understand the flow of the program and for debug purposes. These print statements can be commented out or taken

#    out if needed.

# Import necessary calsses from tensorflow, panda datasets, models and layers

import tensorflow as tflow

import pandas as pan

import numpy as npy

from tensorflow.keras import datasets, layers, models

from tensorflow.keras.utils import to\_categorical

from sklearn.model\_selection import train\_test\_split

#define the one hot encode function to transform the text labels 'R' and 'M' to an array of 1 and zeros.

def hot\_encode\_one(lbls):

    n\_lbls = len(lbls)

    n\_unique\_labels = len(npy.unique(lbls))

    hot\_encode\_one = npy.zeros((n\_lbls,n\_unique\_labels))

    hot\_encode\_one[npy.arange(n\_lbls), lbls] = 1

    return hot\_encode\_one

def main():

  #Import sonar.csv data file into a panda dataset and print the header to verify file was loaded into the panda dataset succesfully

  datafile = pan.read\_csv("sonar.csv")

  datafile.head()

  #separate attributes from lables in their respective arrays

  attributes = datafile.iloc[:,0:60].values

  labels = datafile.iloc[:,60].values

  print(attributes[0:5])

  print(labels[0:5])

  print(attributes.shape)

  print(labels.shape)

  #Convert text based labels 'R' and 'M' to 1 and zero array

  from sklearn.preprocessing import LabelEncoder

  encoder =  LabelEncoder()

  labels1 = encoder.fit\_transform(labels)

  print(labels1)

  #Call hot\_encode\_one function to change 1 and zero to one hot code value

  one\_hot\_labels = hot\_encode\_one(labels1)

  print(one\_hot\_labels[0:5])

  #print (one\_hot\_labels)

  #Split the attribute file and its respective labels into training data and test data. 20% of total data will be used as test data to test the accuracy of the

  #model.

  train\_attr, test\_attr, train\_lbls, test\_lbls = train\_test\_split(attributes, one\_hot\_labels, test\_size=0.2, random\_state=0)

  #Define model use two layers with 128 and 56 nodes each. Final layer will use softmax.

  model = tflow.keras.Sequential([

          tflow.keras.layers.Dense(128, activation='relu'),

          tflow.keras.layers.Dense(56, activation='relu'),

          tflow.keras.layers.Dense(2, activation='softmax')

          ])

  #Complie model use 'rmsprop' optimizer, this optimizer can aslo be chnaged to 'Adam' to see if that helps improve accuracy.

  #Use 'Categorical\_crossentropy' to calculate and manage loss since we are using one hot code instead of integer as the label.

  model.compile(optimizer='rmsprop',

                loss='categorical\_crossentropy',

                metrics=['accuracy'])

  #Now train model on the training data

  #We can adjust epoch size here to see if our model trains any better.

  model.fit(train\_attr, train\_lbls, batch\_size=50, epochs=100)

  #evaluate model against test data to calculate loss and accuracy

  loss, accuracy = model.evaluate(test\_attr, test\_lbls, verbose=0)

  print('Test loss:', loss)

  print('Test accuracy:', accuracy)

  #Make predictions against test data.

  pred\_lbls = model.predict(test\_attr)

  pred\_lbls

  actual = npy.argmax(test\_lbls,axis=1)

  predicted = npy.argmax(pred\_lbls,axis=1)

  #print(f"Actual: {actual}")

  #print(f"Predicted: {predicted}")

  print (actual)

  print (predicted)

main()