import numpy as np

from keras.datasets import imdb

from keras import models

from keras import layers

from keras import optimizers

from keras import losses

from keras import metrics

import matplotlib.pyplot as plt

%matplotlib inline

# Load the data, keeping only 10,000 of the most frequently occuring words

(train\_data, train\_labels), (test\_data, test\_labels) = imdb.load\_data(num\_words = 10000)

train\_data[:2]

train\_labels

# Check the first label

train\_labels[0]

# Since we restricted ourselves to the top 10000 frequent words, no word index should exceed 10000

# we'll verify this below

# Here is a list of maximum indexes in every review --- we search the maximum index in this list of max indexes

print(type([max(sequence) for sequence in train\_data]))

# Find the maximum of all max indexes

max([max(sequence) for sequence in train\_data])

# Let's quickly decode a review

# step 1: load the dictionary mappings from word to integer index

word\_index = imdb.get\_word\_index()

# step 2: reverse word index to map integer indexes to their respective words

reverse\_word\_index = dict([(value, key) for (key, value) in word\_index.items()])

# Step 3: decode the review, mapping integer indices to words

# indices are off by 3 because 0, 1, and 2 are reserverd indices for "padding", "Start of sequence" and "unknown"

decoded\_review = ' '.join([reverse\_word\_index.get(i-3, '?') for i in train\_data[0]])

decoded\_review

len(reverse\_word\_index)

def vectorize\_sequences(sequences, dimension=10000):

results = np.zeros((len(sequences), dimension)) # Creates an all zero matrix of shape (len(sequences),10K)

for i,sequence in enumerate(sequences):

results[i,sequence] = 1 # Sets specific indices of results[i] to 1s

return results

# Vectorize training Data

X\_train = vectorize\_sequences(train\_data)

# Vectorize testing Data

X\_test = vectorize\_sequences(test\_data)

X\_train[0]

X\_train.shape

y\_train = np.asarray(train\_labels).astype('float32')

y\_test = np.asarray(test\_labels).astype('float32')

model = models.Sequential()

model.add(layers.Dense(16, activation='relu', input\_shape=(10000,)))

model.add(layers.Dense(16, activation='relu'))

model.add(layers.Dense(1, activation='sigmoid'))

model.compile(

optimizer=optimizers.RMSprop(learning\_rate=0.001),

loss = losses.binary\_crossentropy,

metrics = [metrics.binary\_accuracy]

)

# Input for Validation

X\_val = X\_train[:10000]

partial\_X\_train = X\_train[10000:]

# Labels for validation

y\_val = y\_train[:10000]

partial\_y\_train = y\_train[10000:]

history = model.fit(

partial\_X\_train,

partial\_y\_train,

epochs=20,

batch\_size=512,

validation\_data=(X\_val, y\_val)

)

history\_dict = history.history

history\_dict.keys()

# Plotting losses

loss\_values = history\_dict['loss']

val\_loss\_values = history\_dict['val\_loss']

epochs = range(1, len(loss\_values) + 1)

plt.plot(epochs, loss\_values, 'g', label="Training Loss")

plt.plot(epochs, val\_loss\_values, 'b', label="Validation Loss")

plt.title('Training and Validation Loss')

plt.xlabel('Epochs')

plt.ylabel('Loss Value')

plt.legend()

plt.show()

# Training and Validation Accuracy

acc\_values = history\_dict['binary\_accuracy']

val\_acc\_values = history\_dict['val\_binary\_accuracy']

epochs = range(1, len(loss\_values) + 1)

plt.plot(epochs, acc\_values, 'g', label="Training Accuracy")

plt.plot(epochs, val\_acc\_values, 'b', label="Validation Accuracy")

plt.title('Training and Validation Accuraccy')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.legend()

plt.show()

model.fit(

partial\_X\_train,

partial\_y\_train,

epochs=3,

batch\_size=512,

validation\_data=(X\_val, y\_val)

)

# Making Predictions for testing data

np.set\_printoptions(suppress=True)

result = model.predict(X\_test)

y\_pred = np.zeros(len(result))

for i, score in enumerate(result):

y\_pred[i] = np.round(score)

mae = metrics.mean\_absolute\_error(y\_pred, y\_test)

mae

**Code 2**

import numpy as np

from tensorflow import keras

from tensorflow.keras.datasets import imdb

from tensorflow.keras.preprocessing.sequence import pad\_sequences

# Set the hyperparameters

max\_features = 10000 # Number of words to consider as features

max\_len = 200 # Maximum length of each sequence (in words)

batch\_size = 32

epochs = 10

# Load the IMDB dataset

(X\_train, y\_train), (X\_test, y\_test) = imdb.load\_data(num\_words=max\_features)

# Pad the sequences to have a consistent length

X\_train = pad\_sequences(X\_train, maxlen=max\_len)

X\_test = pad\_sequences(X\_test, maxlen=max\_len)

# Create a sequential model with an embedding layer and two dense layers

model = keras.Sequential([

keras.layers.Embedding(input\_dim=max\_features, output\_dim=128, input\_length=max\_len),

keras.layers.Flatten(),

keras.layers.Dense(64, activation='relu'),

keras.layers.Dense(1, activation='sigmoid')

])

# Compile the model with binary crossentropy loss and adam optimizer

model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])

# Train the model on the training data

model.fit(X\_train, y\_train, batch\_size=batch\_size, epochs=epochs, validation\_data=(X\_test, y\_test))

# Evaluate the model on the testing data

test\_loss, test\_acc = model.evaluate(X\_test, y\_test) # Print the test accuracy

print('Test accuracy:', test\_acc)