import tensorflow as tf

tf.test.gpu\_device\_name()

Out[1]: '/device:GPU:0'

import pandas as pd

data = pd.read\_csv('Womens Clothing E-Commerce Reviews.csv', header=None)

X = data.iloc[1:, 4].values

y = data.iloc[1:, 5].values

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, random\_state = 0)

import numpy as np

glove\_file = 'glove.6B.50d.txt'

with open(glove\_file, 'r', encoding="utf8") as f:

word\_to\_vec\_map = {}

for line in f:

line = line.strip().split()

curr\_word = line[0]

word\_to\_vec\_map[curr\_word] = np.array(line[1:], dtype=np.float64)

import gensim

dataTrain\_as\_lists\_of\_words = []

for i in range(len(X\_train)):

a\_piece\_of\_sentence = str(X\_train[i])

single\_sentence\_as\_list\_of\_words = gensim.utils.simple\_preprocess( a\_piece\_of\_sentence )

dataTrain\_as\_lists\_of\_words.append(single\_sentence\_as\_list\_of\_words)

dataTest\_as\_lists\_of\_words = []

for i in range(len(X\_test)):

a\_piece\_of\_sentence = str(X\_test[i])

single\_sentence\_as\_list\_of\_words = gensim.utils.simple\_preprocess( a\_piece\_of\_sentence )

dataTest\_as\_lists\_of\_words.append(single\_sentence\_as\_list\_of\_words)

SENTENCE\_LENGTH = 100

EMBEDDED\_VECTOR\_DIM = 50

list\_of\_words = dataTrain\_as\_lists\_of\_words[0]

sentense\_word2vec = np.zeros((SENTENCE\_LENGTH, EMBEDDED\_VECTOR\_DIM))

for word\_nr in range( min(SENTENCE\_LENGTH, len(list\_of\_words)) ):

word = list\_of\_words[word\_nr]

try:

word\_vec = word\_to\_vec\_map[word]

sentense\_word2vec[word\_nr,:] = word\_vec

except:

sentense\_word2vec[word\_nr,:] = np.zeros((EMBEDDED\_VECTOR\_DIM))

print(sentense\_word2vec)

[[-0.077432 -0.17968 1.0954 ... 0.79036 -0.14109 0.63367 ]

[ 0.21637 -0.16276 -0.21876 ... 0.64911 0.19922 0.45611 ]

[ 0.45323 0.059811 -0.10577 ... 0.5324 -0.25103 0.62546 ]

...

[ 0. 0. 0. ... 0. 0. 0. ]

[ 0. 0. 0. ... 0. 0. 0. ]

[ 0. 0. 0. ... 0. 0. 0. ]]

trainX = []

for sentence\_nr in range(len(dataTrain\_as\_lists\_of\_words)):

list\_of\_words = dataTrain\_as\_lists\_of\_words[sentence\_nr]

sentense\_word2vec = np.zeros((SENTENCE\_LENGTH, EMBEDDED\_VECTOR\_DIM))

for word\_nr in range( min(SENTENCE\_LENGTH, len(list\_of\_words)) ):

word = list\_of\_words[word\_nr]

try:

word\_vec = word\_to\_vec\_map[word]

sentense\_word2vec[word\_nr,:] = word\_vec

except:

sentense\_word2vec[word\_nr,:] = np.zeros((EMBEDDED\_VECTOR\_DIM))

trainX.append(sentense\_word2vec)

trainX = np.array(trainX)

print(trainX.shape)

trainY = np.array(y\_train)

trainY = trainY.astype(int)

print(trainY.shape)

(18788, 100, 50)

(18788,)

testX = []

for sentence\_nr in range(len(dataTest\_as\_lists\_of\_words)):

list\_of\_words = dataTest\_as\_lists\_of\_words[sentence\_nr]

sentense\_word2vec = np.zeros((SENTENCE\_LENGTH, EMBEDDED\_VECTOR\_DIM))

for word\_nr in range( min(SENTENCE\_LENGTH, len(list\_of\_words)) ):

word = list\_of\_words[word\_nr]

try:

word\_vec = word\_to\_vec\_map[word]

sentense\_word2vec[word\_nr,:] = word\_vec

except:

sentense\_word2vec[word\_nr,:] = np.zeros((EMBEDDED\_VECTOR\_DIM))

testX.append(sentense\_word2vec)

testX = np.array(testX)

print(testX.shape)

testY = np.array(y\_test)

testY = testY.astype(int)

print(testY.shape)

(4698, 100, 50)

(4698,)

from keras.models import Sequential

from keras.layers import Dense

from keras.layers import LSTM

model = Sequential()

model.add(LSTM(100, input\_shape=(SENTENCE\_LENGTH, EMBEDDED\_VECTOR\_DIM)))

model.add(Dense(8, activation='softmax'))

model.compile(loss='sparse\_categorical\_crossentropy', optimizer='adam', metrics=['sparse\_categorical\_accuracy'])

model.summary()

history = model.fit(trainX,

trainY,

epochs=20,

batch\_size=32,

verbose=1,

validation\_data=(testX, testY))

predY = model.predict(testX)

predY = np.argmax(predY, axis=1)

Using TensorFlow backend.

Model: "sequential\_1"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

lstm\_1 (LSTM) (None, 100) 60400

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_1 (Dense) (None, 8) 808

=================================================================

Total params: 61,208

Trainable params: 61,208

Non-trainable params: 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Train on 18788 samples, validate on 4698 samples

Epoch 1/20

18788/18788 [==============================] - 29s 2ms/step - loss: 1.2325 - sparse\_categorical\_accuracy: 0.5580 - val\_loss: 1.1795 - val\_sparse\_categorical\_accuracy: 0.5583

Epoch 2/20

18788/18788 [==============================] - 29s 2ms/step - loss: 1.0993 - sparse\_categorical\_accuracy: 0.5696 - val\_loss: 1.0550 - val\_sparse\_categorical\_accuracy: 0.5885

Epoch 3/20

18788/18788 [==============================] - 29s 2ms/step - loss: 1.0037 - sparse\_categorical\_accuracy: 0.5942 - val\_loss: 0.9856 - val\_sparse\_categorical\_accuracy: 0.5888

Epoch 4/20

18788/18788 [==============================] - 29s 2ms/step - loss: 0.9594 - sparse\_categorical\_accuracy: 0.6059 - val\_loss: 0.9602 - val\_sparse\_categorical\_accuracy: 0.6081

Epoch 5/20

18788/18788 [==============================] - 29s 2ms/step - loss: 0.9262 - sparse\_categorical\_accuracy: 0.6160 - val\_loss: 0.9360 - val\_sparse\_categorical\_accuracy: 0.6201

Epoch 6/20

18788/18788 [==============================] - 29s 2ms/step - loss: 0.8924 - sparse\_categorical\_accuracy: 0.6264 - val\_loss: 0.9231 - val\_sparse\_categorical\_accuracy: 0.6130

Epoch 7/20

18788/18788 [==============================] - 29s 2ms/step - loss: 0.8743 - sparse\_categorical\_accuracy: 0.6347 - val\_loss: 0.8943 - val\_sparse\_categorical\_accuracy: 0.6296

Epoch 8/20

18788/18788 [==============================] - 29s 2ms/step - loss: 0.8478 - sparse\_categorical\_accuracy: 0.6413 - val\_loss: 0.8952 - val\_sparse\_categorical\_accuracy: 0.6311

Epoch 9/20

18788/18788 [==============================] - 29s 2ms/step - loss: 0.8231 - sparse\_categorical\_accuracy: 0.6498 - val\_loss: 0.9022 - val\_sparse\_categorical\_accuracy: 0.6213

Epoch 10/20

18788/18788 [==============================] - 29s 2ms/step - loss: 0.7990 - sparse\_categorical\_accuracy: 0.6606 - val\_loss: 0.8847 - val\_sparse\_categorical\_accuracy: 0.6371

Epoch 11/20

18788/18788 [==============================] - 29s 2ms/step - loss: 0.7838 - sparse\_categorical\_accuracy: 0.6680 - val\_loss: 0.8753 - val\_sparse\_categorical\_accuracy: 0.6358

Epoch 12/20

18788/18788 [==============================] - 29s 2ms/step - loss: 0.7594 - sparse\_categorical\_accuracy: 0.6778 - val\_loss: 0.8833 - val\_sparse\_categorical\_accuracy: 0.6347

Epoch 13/20

18788/18788 [==============================] - 29s 2ms/step - loss: 0.7378 - sparse\_categorical\_accuracy: 0.6882 - val\_loss: 0.9086 - val\_sparse\_categorical\_accuracy: 0.6260

Epoch 14/20

18788/18788 [==============================] - 29s 2ms/step - loss: 0.7142 - sparse\_categorical\_accuracy: 0.6982 - val\_loss: 0.9530 - val\_sparse\_categorical\_accuracy: 0.6379

Epoch 15/20

18788/18788 [==============================] - 29s 2ms/step - loss: 0.6903 - sparse\_categorical\_accuracy: 0.7113 - val\_loss: 0.9211 - val\_sparse\_categorical\_accuracy: 0.63796827 - sparse\_categorical\_accuracy: 0.7123

Epoch 16/20

18788/18788 [==============================] - 28s 2ms/step - loss: 0.6642 - sparse\_categorical\_accuracy: 0.7231 - val\_loss: 0.9632 - val\_sparse\_categorical\_accuracy: 0.6262

Epoch 17/20

18788/18788 [==============================] - 29s 2ms/step - loss: 0.6368 - sparse\_categorical\_accuracy: 0.7359 - val\_loss: 0.9751 - val\_sparse\_categorical\_accuracy: 0.6032

Epoch 18/20

18788/18788 [==============================] - 29s 2ms/step - loss: 0.6136 - sparse\_categorical\_accuracy: 0.7430 - val\_loss: 1.0388 - val\_sparse\_categorical\_accuracy: 0.6324

Epoch 19/20

18788/18788 [==============================] - 29s 2ms/step - loss: 0.5842 - sparse\_categorical\_accuracy: 0.7627 - val\_loss: 1.0345 - val\_sparse\_categorical\_accuracy: 0.6132

Epoch 20/20

18788/18788 [==============================] - 29s 2ms/step - loss: 0.5596 - sparse\_categorical\_accuracy: 0.7747 - val\_loss: 1.0692 - val\_sparse\_categorical\_accuracy: 0.6211

import matplotlib.pyplot as plt

acc = history.history['sparse\_categorical\_accuracy']

val\_acc = history.history['val\_sparse\_categorical\_accuracy']

loss = history.history['loss']

val\_loss = history.history['val\_loss']

epochs = range(len(acc))

plt.plot(epochs, acc, 'bo', label='Training acc')

plt.plot(epochs, val\_acc, 'r', label='Validation acc')

plt.title('Training and validation accuracy')

plt.legend()

plt.figure()

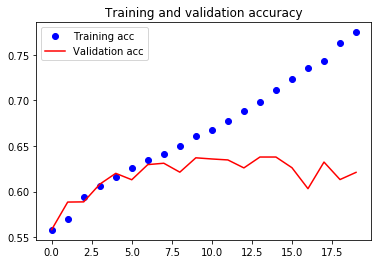
plt.plot(epochs, loss, 'bo', label='Training loss')

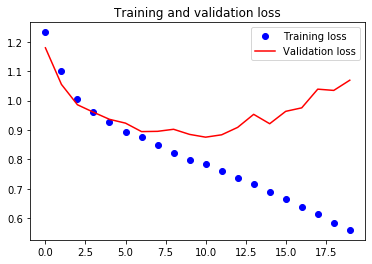
plt.plot(epochs, val\_loss, 'r', label='Validation loss')

plt.title('Training and validation loss')

plt.legend()

plt.show()





from sklearn.metrics import mean\_squared\_error as mse

from sklearn.metrics import mean\_absolute\_error as mae

print("The Root Mean Square Error is:", np.sqrt(mse(predY, testY)))

print("The Mean Absolute Error is:", mae(predY, testY))

The Root Mean Square Error is: 0.8655951721957357

The Mean Absolute Error is: 0.4836100468284376