```
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']

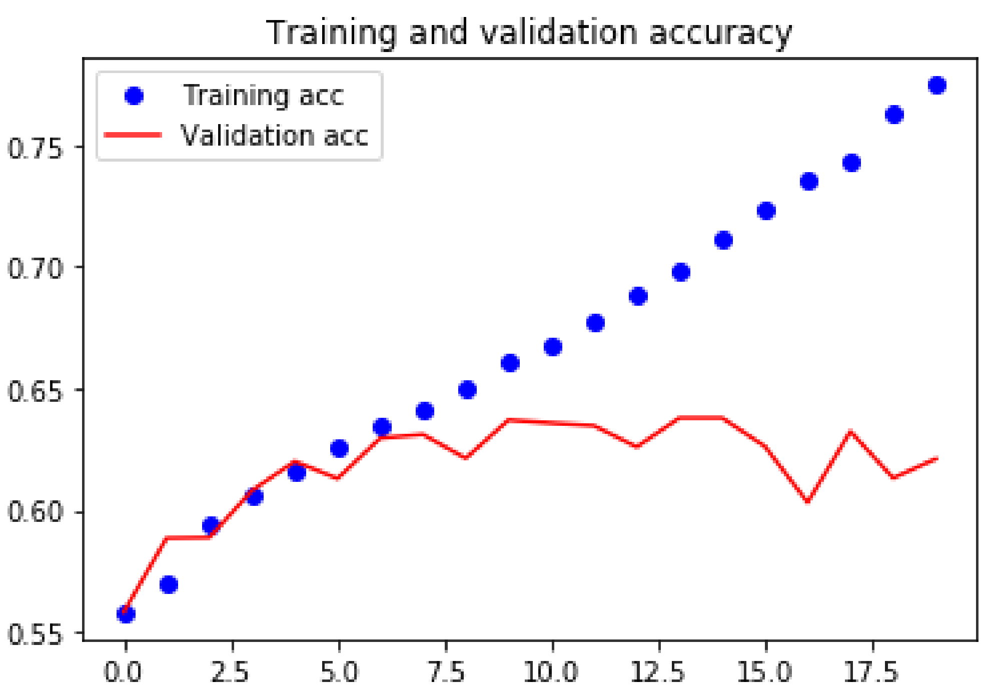
loss = history.history['loss']

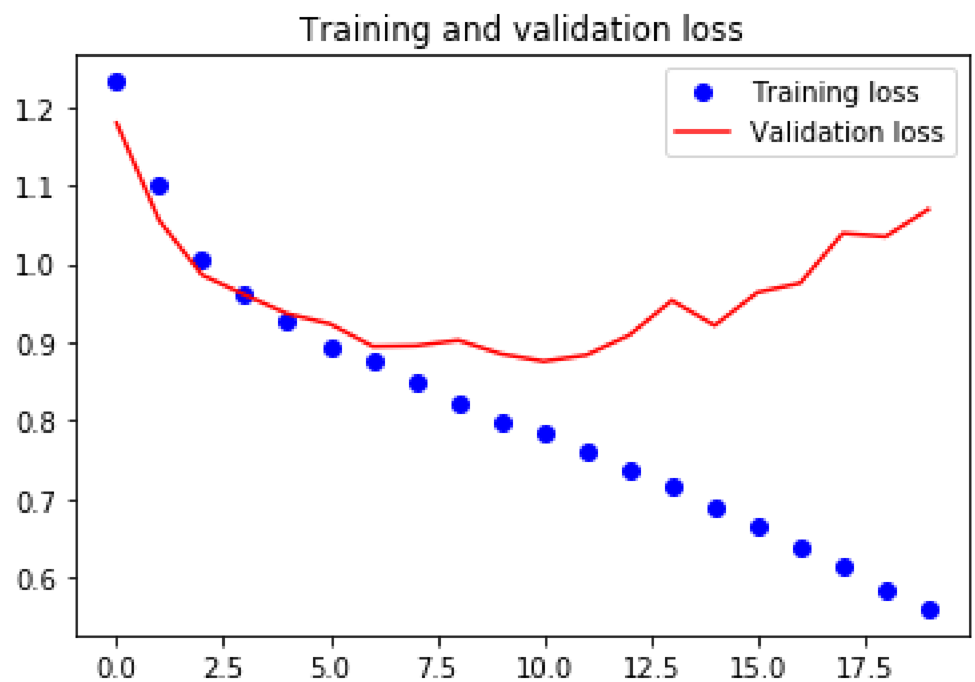
val_loss = history.history['val_loss']

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