Irika Katiyar
ikatiyar@usc.edu

I used classification for my model since it's trying to classify whether a tweet is negative (0) or positive (4) or neutral (2).

For data preprocessing, I did something similar to what was done in the Lesson 6 RNN example we did in the club meeting (screenshots of the file are below in case the link in the colab doesn't work). I first loaded the tweets and the labels from the data frame I created out of the csv file. I next padded and then created the embedding matrix and word embeddings for the various tweets using the GloVe pre trained word embeddings we used in the Lesson 6 RNN folder (I just mapped to the same file to avoid having to download it onto my computer)

I trained the model on the tweets and the valence and the target values for the testing was the valence value for a given tweet.

Value	Domain
Tweet	string
Valence	{0, 2, 4}

For my model, I attempted to make an RNN, with an embedding layer (using the embedding matrix created during preprocessing), 2 LSTM layers (with output sizes of 64), and 2 Dense layers (one with size 32, and the last with size 1 since that was the final output). The 2 LSTM layers and one Dense layer had relu as the activation, and the last Dense layer that represents the output was sigmoid as the activation. I tried different variations of the number of layers to use, and this seemed to output the best results with a reasonable amount of time. For the loss, I used binary_crossentropy since based on the data we were deciding between 0 and 4 (as there were no 2 values in the data set), and the optimization was RMSprop since that one seemed to be good for recurrent models, according to what we learned during our club meeting. I used a validation split of 0.5 for my model, similar to the example we did during our club meeting.

I tried different test ratios, different loss functions, different numbers of layers, and I was either getting the problem of overfitting (100% accuracy) or very low accuracy, nothing quite in the middle. The model most definitely doesn't work since my val_loss and val_accuracy are very off, however I did learn a lot from this exercise, understanding better how RNNs work and how the different layers being added and removed affect the model.

Load Data file (in case link in Colab doesn't work)

load_data.py

```python
import pickle
import numpy as np
from keras.preprocessing.text import text_to_word_sequence
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
import os

EMBEDDING_DIM = 50

def load_data(data_df, EMBEDDINGS_DIR):
  #Load tweets, labels from dataframe
  print("1: Setting tweets and labels to columns of df")
  tweets = data_df["tweet"]
  labels = data_df["valence"]

  #Tokenize the tweets
  print("2: tokenizing tweets")
  tokenizer = Tokenizer()
  #updating internal vocabulary
  tokenizer.fit_on_texts(tweets)

  #assigning unique values to each word
  sequences = tokenizer.texts_to_sequences(tweets)
  word_index = tokenizer.word_index

  #Pad sequences to ensure samples are the same size
  print("3: padding")
  training_data = pad_sequences(sequences)

  #Loading word embeddings
  print("4: loading word embeddings")
  embeddings_index = {}
  f = open(EMBEDDINGS_DIR,'rb')
  #for each embedding
  for line in f:
    values = line.split()
    #decode word using UTF-8
    word = values[0].decode('UTF-8')
    coefs = np.asarray(values[1:], dtype='float32')
    #value at word index will be embedding
    embeddings_index[word] = coefs
  f.close()

  print("5: finding word embeddings")
  # prepare word embedding matrix
  num_words = len(word_index)+1
  embedding_matrix = np.zeros((num_words, EMBEDDING_DIM))
  #for each word in tweet
  for word, i in word_index.items():
      if i >= num_words:
          continue
      #find embedding for word
      embedding_vector = embeddings_index.get(word)
      #if embedding does exist then add it to the matrix
      if embedding_vector is not None:
          # words not found in embedding index will be all-zeros.
          embedding_matrix[i] = embedding_vector

  return tweets, training_data, labels, word_index, embedding_matrix
```