```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
from __future__ import division, print_function
from gensim import models
from keras.callbacks import ModelCheckpoint
from keras.layers import Dense, Dropout, Reshape, Flatten, concatenate, Input, Conv1D, Global
from keras.layers.recurrent import LSTM
from keras.models import Sequential
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.models import Model
from sklearn.model_selection import train_test_split
import numpy as np
import pandas as pd
import os
import collections
import re
import string
```

```
data = pd.read_csv('/content/imdb_labelled.tsv', header = None, delimiter='\t')
```

```
data.columns = ['Text', 'Label']
```

```
data.head()
```

	Text	Label
0	A very, very, very slow-moving, aimless movie ...	0
1	Not sure who was more lost - the flat characte...	0
2	Attempting artiness with black & white and cle...	0
3	Very little music or anything to speak of.	0
4	The best scene in the movie was when Gerardo i...	1

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
data.Label.unique()
```

```
array([0, 1])
```

```
data.shape
```

```
(748, 2)
```

```
pos = []
neg = []
for l in data.Label:
    if l == 0:
        pos.append(0)
        neg.append(1)
    elif l == 1:
        pos.append(1)
        neg.append(0)
```

```
data['Pos']= pos
data['Neg']= neg
```

```
data.head()
```

	Text	Label	Pos	Neg
0	A very, very, very slow-moving, aimless movie ...	0	0	1
1	Not sure who was more lost - the flat characte...	0	0	1
2	Attempting artiness with black & white and cle...	0	0	1
3	Very little music or anything to speak of.	0	0	1
4	The best scene in the movie was when Gerardo i...	1	1	0

```
def remove_punct(text):
    text_nopunct = ''
    text_nopunct = re.sub('['+string.punctuation+']', '', text)
    return text_nopunct

data['Text_Clean'] = data['Text'].apply(lambda x: remove_punct(x))
```

```
from nltk import word_tokenize, WordNetLemmatizer
import nltk
nltk.download('punkt')
tokens = [word_tokenize(sen) for sen in data.Text_Clean]
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
```

```
[nltk_data]   Unzipping tokenizers/punkt.zip.
```

```
def lower_token(tokens):
    return [w.lower() for w in tokens]

lower_tokens = [lower_token(token) for token in tokens]
```

```
import nltk
nltk.download("stopwords")
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
True
```

```
import nltk
from nltk.corpus import stopwords
nltk.download("stopwords")
stoplist = stopwords.words('english')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```
def remove_stop_words(tokens):
    return [word for word in tokens if word not in stoplist]

filtered_words = [remove_stop_words(sen) for sen in lower_tokens]

result = [' '.join(sen) for sen in filtered_words]

data['Text_Final'] = result

data['tokens'] = filtered_words

data = data[['Text_Final', 'tokens', 'Label', 'Pos', 'Neg']]

data[:4]
```

Text_Final	tokens	Label	Pos	Neg

```
data_train, data_test = train_test_split(data, test_size=0.10, random_state=42)
```

```
all_training_words = [word for tokens in data_train["tokens"] for word in tokens]
training_sentence_lengths = [len(tokens) for tokens in data_train["tokens"]]
TRAINING_VOCAB = sorted(list(set(all_training_words)))
print("%s words total, with a vocabulary size of %s" % (len(all_training_words), len(TRAINING
print("Max sentence length is %s" % max(training_sentence_lengths))
```

```
7218 words total, with a vocabulary size of 2881
Max sentence length is 789
```

```
all_test_words = [word for tokens in data_test["tokens"] for word in tokens]
test_sentence_lengths = [len(tokens) for tokens in data_test["tokens"]]
TEST_VOCAB = sorted(list(set(all_test_words)))
print("%s words total, with a vocabulary size of %s" % (len(all_test_words), len(TEST_VOCAB))
print("Max sentence length is %s" % max(test_sentence_lengths))
```

```
580 words total, with a vocabulary size of 457
Max sentence length is 24
```

```
word2vec_path = '/content/drive/MyDrive/GoogleNews-vectors-negative300.bin.gz'
word2vec = models.KeyedVectors.load_word2vec_format(word2vec_path, binary=True)
```

```
def get_average_word2vec(tokens_list, vector, generate_missing=False, k=300):
    if len(tokens_list)<1:
        return np.zeros(k)
    if generate_missing:
        vectorized = [vector[word] if word in vector else np.random.rand(k) for word in toker
    else:
        vectorized = [vector[word] if word in vector else np.zeros(k) for word in tokens_list
    length = len(vectorized)
    summed = np.sum(vectorized, axis=0)
    averaged = np.divide(summed, length)
    return averaged

def get_word2vec_embeddings(vectors, clean_comments, generate_missing=False):
    embeddings = clean_comments['tokens'].apply(lambda x: get_average_word2vec(x, vectors, ge
    return list(embeddings)
```

```
training_embeddings = get_word2vec_embeddings(word2vec, data_train, generate_missing=True)
```

```
MAX_SEQUENCE_LENGTH = 50
EMBEDDING_DIM = 300
```

```
tokenizer = Tokenizer(num_words=len(TRAINING_VOCAB), lower=True, char_level=False)
tokenizer.fit_on_texts(data_train["Text_Final"].tolist())
training_sequences = tokenizer.texts_to_sequences(data_train["Text_Final"].tolist())

train_word_index = tokenizer.word_index
print('Found %s unique tokens.' % len(train_word_index))
```

```
Found 2881 unique tokens.
```

```
train_cnn_data = pad_sequences(training_sequences, maxlen=MAX_SEQUENCE_LENGTH)
```

```
train_embedding_weights = np.zeros((len(train_word_index)+1, EMBEDDING_DIM))
for word,index in train_word_index.items():
    train_embedding_weights[index,:] = word2vec[word] if word in word2vec else np.random.ranc
print(train_embedding_weights.shape)
```

```
(2882, 300)
```

```
test_sequences = tokenizer.texts_to_sequences(data_test["Text_Final"].tolist())
test_cnn_data = pad_sequences(test_sequences, maxlen=MAX_SEQUENCE_LENGTH)
```

```
def ConvNet(embeddings, max_sequence_length, num_words, embedding_dim, labels_index):

    embedding_layer = Embedding(num_words,
                            embedding_dim,
                            weights=[embeddings],
                            input_length=max_sequence_length,
                            trainable=False)

    sequence_input = Input(shape=(max_sequence_length,), dtype='int32')
    embedded_sequences = embedding_layer(sequence_input)

    convs = []
    filter_sizes = [2,3,4,5,6]

    for filter_size in filter_sizes:
        l_conv = Conv1D(filters=200, kernel_size=filter_size, activation='relu')(embedded_sec
        l_pool = GlobalMaxPooling1D()(l_conv)
        convs.append(l_pool)


    l_merge = concatenate(convs, axis=1)

    x = Dropout(0.1)(l_merge)
    x = Dense(128, activation='relu')(x)
    x = Dropout(0.2)(x)
    preds = Dense(labels_index, activation='sigmoid')(x)
```

```
    model = Model(sequence_input, preds)
    model.compile(loss='binary_crossentropy',
                  optimizer='adam',
                  metrics=['acc'])
    model.summary()
    return model

label_names = ['Pos', 'Neg']


y_train = data_train[label_names].values


x_train = train_cnn_data
y_tr = y_train


model = ConvNet(train_embedding_weights, MAX_SEQUENCE_LENGTH, len(train_word_index)+1, EMBEDD
                len(list(label_names)))
```

```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_1 (InputLayer)           [(None, 50)]         0           []

 embedding (Embedding)          (None, 50, 300)      864600      ['input_1[0][0]']

 conv1d (Conv1D)                (None, 49, 200)      120200      ['embedding[0][0]']

 conv1d_1 (Conv1D)              (None, 48, 200)      180200      ['embedding[0][0]']

 conv1d_2 (Conv1D)              (None, 47, 200)      240200      ['embedding[0][0]']

 conv1d_3 (Conv1D)              (None, 46, 200)      300200      ['embedding[0][0]']

 conv1d_4 (Conv1D)              (None, 45, 200)      360200      ['embedding[0][0]']

 global_max_pooling1d (GlobalMa  (None, 200)         0           ['conv1d[0][0]']
 xPooling1D)

 global_max_pooling1d_1 (Global  (None, 200)         0           ['conv1d_1[0][0]']
 MaxPooling1D)

 global_max_pooling1d_2 (Global  (None, 200)         0           ['conv1d_2[0][0]']
 MaxPooling1D)

 global_max_pooling1d_3 (Global  (None, 200)         0           ['conv1d_3[0][0]']
 MaxPooling1D)

 global_max_pooling1d_4 (Global  (None, 200)         0           ['conv1d_4[0][0]']
 MaxPooling1D)

 concatenate (Concatenate)      (None, 1000)         0           ['global_max_pooling1d
                                                                   'global_max_pooling1d
```

```
                                                                    'global_max_pooling1d
                                                                    'global_max_pooling1d
                                                                    'global_max_pooling1d

 dropout (Dropout)              (None, 1000)         0           ['concatenate[0][0]']

 dense (Dense)                  (None, 128)          128128      ['dropout[0][0]']

 dropout_1 (Dropout)            (None, 128)          0           ['dense[0][0]']

 dense_1 (Dense)                (None, 2)            258         ['dropout_1[0][0]']

==================================================================================================
Total params: 2,193,986
Trainable params: 1,329,386
Non-trainable params: 864,600
__________________________________________________________________________________________________
```

```
num_epochs = 3
batch_size = 34
```

```
hist = model.fit(x_train, y_tr, epochs=num_epochs, validation_split=0.1, shuffle=True, batch_
```

```
Epoch 1/3
18/18 [==============================] - 6s 195ms/step - loss: 0.7058 - acc: 0.5570 - v
Epoch 2/3
18/18 [==============================] - 3s 155ms/step - loss: 0.5231 - acc: 0.8397 - v
Epoch 3/3
18/18 [==============================] - 3s 156ms/step - loss: 0.3469 - acc: 0.8876 - v
```

```
predictions = model.predict(test_cnn_data, batch_size=1024, verbose=1)
```

```
1/1 [==============================] - 1s 509ms/step
```

```
labels = [1, 0]
```

```
prediction_labels=[]
for p in predictions:
    prediction_labels.append(labels[np.argmax(p)])
```

```
sum(data_test.Label==prediction_labels)/len(prediction_labels)
```

```
0.7866666666666666
```

```
data_test.Label.value_counts()
```

```
0    44
```

```
1    31
Name: Label, dtype: int64
```

✓ 0s completed at 12:29 PM