Sentiment Analysis using LSTM

- **By:** - Vikas Velmurugan
- **Date Created:** 28/9/23

- **What is sentiment Analysis?**
  - Sentiment Analysis is an NLP application that identifies a text corpus's emotional or sentimental tone or opinion. Usually, emotions or attitudes towards a topic can be positive, negative or neutral. This makes sentiment analysis a text classification task.
  - Examples of positive, negative and neutral expressions are:
    - I enjoyed the movie! - Positive
    - I am not sure if I liked the movie - Neutral
    - It was the most terrible movie I have ever seen - Negative

STEPS:

- Load the dataset (50K IMDB Movie Review)
- Clean Dataset
- Encode Sentiments
- Split Dataset
- Tokenize and Pad/Truncate Reviews
- Build Architecture/Model
- Train and Test

In [24]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", for
ce_remount=True).
```

In [41]:
```
# Importing necessary libraries
import pandas as pd    # to load dataset
import numpy as np     # for mathematic equation
from nltk.corpus import stopwords   # to get collection of stopwords
from sklearn.model_selection import train_test_split       # for splitting dataset
from tensorflow.keras.preprocessing.text import Tokenizer  # to encode text to int
from tensorflow.keras.preprocessing.sequence import pad_sequences   # to do padding or truncating
from tensorflow.keras.models import Sequential     # the model
from tensorflow.keras.layers import Embedding, LSTM, Dense # layers of the architecture
from tensorflow.keras.callbacks import ModelCheckpoint   # save model
from tensorflow.keras.models import load_model   # load saved model
import re
```

In [42]:
```
#reading the dataset
data=pd.read_csv('/content/drive/MyDrive/AI WORLD/NLP/Sentiment Analysis with LSTM/IMDB Dataset.csv')
```

Dataset link : https://www.kaggle.com/datasets/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews

In [43]:
```
data.head(20)
```

Out[43]:

	review	sentiment
0	One of the other reviewers has mentioned that ...	positive
1	A wonderful little production. The...	positive
2	I thought this was a wonderful way to spend ti...	positive
3	Basically there's a family where a little boy ...	negative
4	Petter Mattei's "Love in the Time of Money" is...	positive
5	Probably my all-time favorite movie, a story o...	positive
6	I sure would like to see a resurrection of a u...	positive
7	This show was an amazing, fresh & innovative i...	negative
8	Encouraged by the positive comments about this...	negative
9	If you like original gut wrenching laughter yo...	positive
10	Phil the Alien is one of those quirky films wh...	negative
11	I saw this movie when I was about 12 when it c...	negative
12	So im not a big fan of Boll's work but then ag...	negative
13	The cast played Shakespeare. Shakes...	negative
14	This a fantastic movie of three prisoners who ...	positive
15	Kind of drawn in by the erotic scenes, only to...	negative
16	Some films just simply should not be remade. T...	positive
17	This movie made it into one of my top 10 most ...	negative
18	I remember this film,it was the first film i h...	positive
19	An awful film! It must have been up against so...	negative

In [45]:
```
import nltk
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

Out[45]:
```
True
```

In [46]:
```
english_stops = set(stopwords.words('english'))
```

LOAD AND CLEAN DATASET

- In the original dataset, the reviews are still dirty. There are still html tags, numbers, uppercase, and punctuations. This will not be good for training, so in load_dataset() function, beside loading the dataset using pandas, I also pre-process the reviews by removing html tags, non alphabet (punctuations and numbers), stop words, and lower case all of the reviews.

ENCODE SENTIMENTS

- In the same function, I also encode the sentiments into integers (0 and 1). Where 0 is for negative sentiments and 1 is for positive sentiments.

In [48]:
```
def load_dataset():
    data = pd.read_csv('/content/drive/MyDrive/AI WORLD/NLP/Sentiment Analysis with LSTM/IMDB Dataset.csv')
    x_data = data['review']       # Reviews/Input
    y_data = data['sentiment']    # Sentiment/Output

    # PRE-PROCESS REVIEW
    x_data = x_data.replace({'<.*?>': ''}, regex = True)          # remove html tag
    x_data = x_data.replace({'[^A-Za-z]': ' '}, regex = True)     # remove non alphabet
    x_data = x_data.apply(lambda review: [w for w in review.split() if w not in english_stops])  # remove stop
    x_data = x_data.apply(lambda review: [w.lower() for w in review])   # lower case

    # ENCODE SENTIMENT -> 0 & 1
    y_data = y_data.replace('positive', 1)
    y_data = y_data.replace('negative', 0)

    return x_data, y_data

x_data, y_data = load_dataset()
```

Splitting Datset into training and testing

- We need to shuffle the data because in the original dataset, the reviews and sentiments are in order, where they list positive reviews first and then negative reviews. By shuffling the data, it will be distributed equally in the model, so it will be more accurate for predictions.

In [49]:
```
x_train, x_test, y_train, y_test = train_test_split(x_data, y_data, test_size = 0.2)
```

- Function for getting the maximum review length, by calculating the mean of all the reviews length (using numpy.mean)

In [50]:
```
def get_max_length():
    review_length = []
    for review in x_train:
        review_length.append(len(review))

    return int(np.ceil(np.mean(review_length)))
```

Tokenizing and Pad/Truncate Reviews

- A Neural Network only accepts numeric data, so we need to encode the reviews. I use tensorflow.keras.preprocessing.text.Tokenizer to encode the reviews into integers, where each unique word is automatically indexed (using fit_on_texts method) based on x_train.
- x_train and x_test is converted into integers using texts_to_sequences method.
- Each reviews has a different length, so we need to add padding (by adding 0) or truncating the words to the same length (in this case, it is the mean of all reviews length) using tensorflow.keras.preprocessing.sequence.pad_sequences.
- post, pad or truncate the words in the back of a sentence
- pre, pad or truncate the words in front of a sentence

In [51]:
```
token = Tokenizer(lower=True)
token.fit_on_texts(x_train)
x_train = token.texts_to_sequences(x_train)
x_test = token.texts_to_sequences(x_test)

max_length = get_max_length()

x_train = pad_sequences(x_train, maxlen=max_length, padding='post', truncating='post')
x_test = pad_sequences(x_test, maxlen=max_length, padding='post', truncating='post')

total_words = len(token.word_index) + 1   # add 1 because of 0 padding
```

Build Architecture Model

- **Embedding Layer:** in simple terms, it creates word vectors of each word in the word_index and group words that are related or have similar meaning by analyzing other words around them.
- **LSTM Layer:** to make a decision to keep or throw away data by considering the current input, previous output, and previous memory. There are some important components in LSTM.
- **Forget Gate:** decides information is to be kept or thrown away
- **Input Gate:** updates cell state by passing previous output and current input into sigmoid activation function Cell State, calculate new cell state, it is multiplied by forget vector (drop value if multiplied by a near 0), add it with the output from input gate to update the cell state value.
- **Ouput Gate:** decides the next hidden state and used for predictions
- **Dense Layer:** compute the input with the weight matrix and bias (optional), and using an activation function. I use Sigmoid activation function for this work because the output is only 0 or 1.
- The optimizer is Adam and the loss function is Binary Crossentropy because again the output is only 0 and 1, which is a binary number.

In [52]:
```
# ARCHITECTURE
EMBED_DIM = 32
LSTM_OUT = 64

model = Sequential()
model.add(Embedding(total_words, EMBED_DIM, input_length = max_length))
model.add(LSTM(LSTM_OUT))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])

print(model.summary())
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_2 (Embedding)     (None, 130, 32)           2961184

 lstm_2 (LSTM)               (None, 64)                24832

 dense_2 (Dense)             (None, 1)                 65

=================================================================
Total params: 2986081 (11.39 MB)
Trainable params: 2986081 (11.39 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
```

TRAINING

- For training, we need to fit our x_train (input) and y_train (output/label) data. For this training, I use a mini-batch learning method with a batch_size of 128 and 6 epochs.
- Also, I added a callback called checkpoint to save the model locally for every epoch if its accuracy improved from the previous epoch.

In [56]:
```
checkpoint = ModelCheckpoint(
    '/content/drive/MyDrive/AI WORLD/NLP/Sentiment Analysis with LSTM/models/LSTM.h5',
    monitor='accuracy',
    save_best_only=True,
    verbose=1
)
```

In [58]:
```
model.fit(x_train, y_train, batch_size = 128, epochs = 6, verbose=1, validation_split=0.2, callbacks=[checkpoin
```

```
Epoch 1/6
245/250 [============================>.] - ETA: 0s - loss: 0.0250 - accuracy: 0.9949
Epoch 1: accuracy improved from -inf to 0.99488, saving model to /content/drive/MyDrive/AI WORLD/NLP/Sentiment
Analysis with LSTM/models/LSTM.h5
250/250 [==============================] - 8s 33ms/step - loss: 0.0251 - accuracy: 0.9949 - val_loss: 0.5731 -
val_accuracy: 0.8618
Epoch 2/6
  8/250 [..............................] - ETA: 1s - loss: 0.0235 - accuracy: 0.9971
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3000: UserWarning: You are saving your mod
el as an HDF5 file via `model.save()`. This file format is considered legacy. We recommend using instead the na
tive Keras format, e.g. `model.save('my_model.keras')`.
  saving_api.save_model(
247/250 [============================>.] - ETA: 0s - loss: 0.0223 - accuracy: 0.9953
Epoch 2: accuracy improved from 0.99488 to 0.99525, saving model to /content/drive/MyDrive/AI WORLD/NLP/Sentime
nt Analysis with LSTM/models/LSTM.h5
250/250 [==============================] - 7s 28ms/step - loss: 0.0224 - accuracy: 0.9952 - val_loss: 0.5430 -
val_accuracy: 0.8619
Epoch 3/6
249/250 [============================>.] - ETA: 0s - loss: 0.0231 - accuracy: 0.9947
Epoch 3: accuracy did not improve from 0.99525
250/250 [==============================] - 8s 31ms/step - loss: 0.0231 - accuracy: 0.9947 - val_loss: 0.5405 -
val_accuracy: 0.8616
Epoch 4/6
246/250 [============================>.] - ETA: 0s - loss: 0.0237 - accuracy: 0.9945
Epoch 4: accuracy did not improve from 0.99525
250/250 [==============================] - 6s 26ms/step - loss: 0.0237 - accuracy: 0.9945 - val_loss: 0.5995 -
val_accuracy: 0.8566
Epoch 5/6
248/250 [============================>.] - ETA: 0s - loss: 0.0285 - accuracy: 0.9933
Epoch 5: accuracy did not improve from 0.99525
250/250 [==============================] - 7s 27ms/step - loss: 0.0285 - accuracy: 0.9933 - val_loss: 0.5295 -
val_accuracy: 0.8490
Epoch 6/6
250/250 [==============================] - ETA: 0s - loss: 0.0305 - accuracy: 0.9927
Epoch 6: accuracy did not improve from 0.99525
250/250 [==============================] - 4s 16ms/step - loss: 0.0305 - accuracy: 0.9927 - val_loss: 0.5265 -
val_accuracy: 0.8602
```

Out[58]:
```
<keras.src.callbacks.History at 0x7e2f00c94cd0>
```

- To evaluate the model, we need to predict the sentiment using our x_test data and comparing the predictions with y_test (expected output) data. Then, we calculate the accuracy of the model by dividing numbers of correct prediction with the total data. Resulted an **accuracy of 85.79%.**

In [59]:
```
score=model.evaluate(x_test,y_test,verbose=1)
```

```
313/313 [==============================] - 2s 5ms/step - loss: 0.5237 - accuracy: 0.8579
```

LOADING SAVED MODEL

In [60]:
```
loaded_model = load_model('/content/drive/MyDrive/AI WORLD/NLP/Sentiment Analysis with LSTM/models/LSTM.h5')
```

In [61]:
```
review = str(input('Movie Review: '))
```

```
Movie Review: Nothing was typical about this. Everything was beautifully done in this movie, the story, the flo
w, the scenario, everything. I highly recommend it for mystery lovers, for anyone who wants to watch a good mov
ie!
```

The input must be preprocessed before it is passed to the model to predict.

In [62]:
```
# Pre-process input
regex = re.compile(r'[^a-zA-Z\s]')
review = regex.sub('', review)
print('Cleaned: ', review)

words = review.split(' ')
filtered = [w for w in words if w not in english_stops]
filtered = ' '.join(filtered)
filtered = [filtered.lower()]

print('Filtered: ', filtered)
```

```
Cleaned:  Nothing was typical about this Everything was beautifully done in this movie the story the flow the s
cenario everything I highly recommend it for mystery lovers for anyone who wants to watch a good movie
Filtered:  ['nothing typical everything beautifully done movie story flow scenario everything i highly recommen
d mystery lovers anyone wants watch good movie']
```

Once again, we need to tokenize and encode the words. I use the tokenizer which was previously declared because we want to encode the words based on words that are known by the model.

In [63]:
```
tokenize_words = token.texts_to_sequences(filtered)
tokenize_words = pad_sequences(tokenize_words, maxlen=max_length, padding='post', truncating='post')
print(tokenize_words)
```

```
[[  76  680  172 1176  130    3   15 2676 2728  172    1  446  281  687
  1691  151  398   33    9    3    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0]]
```

In [64]:
```
result = loaded_model.predict(tokenize_words)
print(result)
```

```
1/1 [==============================] - 0s 326ms/step
[[0.99831116]]
```

If the confidence score is close to 0, then the statement is negative. On the other hand, if the confidence score is close to 1, then the statement is positive. I use a threshold of 0.7 to determine which confidence score is positive and negative, so if it is equal or greater than 0.7, it is positive and if it is less than 0.7, it is negative

In [65]:
```
if result >= 0.7:
    print('positive')
else:
    print('negative')
```

```
positive
```

THANK YOU! HAPPY LEARNING :)

In [ ]: