DL Lab # 10 : Design and implement a LSTM model for Sentiment Analysis

```
import re
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
import keras
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
import math
import nltk

data = pd.read_csv('IMDB Dataset.csv', error_bad_lines=False, engine="python")
data.head()
```

```
<ipython-input-10-764701553c0d>:1: FutureWarning: The error_bad_lines argument has been deprecated and
will be removed in a future version. Use on_bad_lines in the future.


  data = pd.read_csv('IMDB Dataset.csv', error_bad_lines=False, engine="python")
Skipping line 19793: unexpected end of data
```

	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

```
print(data.shape)
print(data.dtypes)
```

```
(19791, 2)
review       object
sentiment    object
dtype: object
```

```
def remove_tags(string):
    removelist = ""
    result = re.sub('','',string)
    result = re.sub('https://.*','',result)
    result = re.sub(r'[^w'+removelist+']', ' ',result)
    result = result.lower()
    return result
data['review']=data['review'].apply(lambda cw : remove_tags(cw))

# nltk.download('stopwords')
from nltk.corpus import stopwords
stop_words = set(stopwords.words('english'))
data['review'] = data['review'].apply(lambda x: ' '.join([word for word in x.split() if word not in
(stop_words)]))
```

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

```
# nltk.download("wordnet")
```

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

True
```

```
w_tokenizer = nltk.tokenize.WhitespaceTokenizer()
lemmatizer = nltk.stem.WordNetLemmatizer()
def lemmatize_text(text):
    st = ""
```

```
    for w in w_tokenizer.tokenize(text):
        st = st + lemmatizer.lemmatize(w) + " "
    return st
data['review'] = data.review.apply(lemmatize_text)
data

s = 0.0
for i in data['review']:
    word_list = i.split()
    s = s + len(word_list)
print("Average length of each review : ",s/data.shape[0])
pos = 0
for i in range(data.shape[0]):
    if data.iloc[i]['sentiment'] == 'positive':
        pos = pos + 1
neg = data.shape[0]-pos
print("Percentage of reviews with positive sentiment is "+str(pos/data.shape[0]*100)+"%")
print("Percentage of reviews with negative sentiment is "+str(neg/data.shape[0]*100)+"%")
```

```
Average length of each review :  18.714
Percentage of reviews with positive sentiment is 50.0%
Percentage of reviews with negative sentiment is 50.0%
```

```
reviews = data['review'].values
labels = data['sentiment'].values
encoder = LabelEncoder()
encoded_labels = encoder.fit_transform(labels)

train_sentences, test_sentences, train_labels, test_labels = train_test_split(reviews, encoded_labels,
stratify = encoded_labels)

# Hyperparameters of the model

vocab_size = 3000
oov_tok = ''
embedding_dim = 100
max_length = 200
padding_type='post'
trunc_type='post'

tokenizer = Tokenizer(num_words = vocab_size, oov_token=oov_tok)
tokenizer.fit_on_texts(train_sentences)
word_index = tokenizer.word_index

train_sequences = tokenizer.texts_to_sequences(train_sentences)
train_padded = pad_sequences(train_sequences, padding='post', maxlen=max_length)

test_sequences = tokenizer.texts_to_sequences(test_sentences)
test_padded = pad_sequences(test_sequences, padding='post', maxlen=max_length)

# model initialization
model = keras.Sequential([
    keras.layers.Embedding(vocab_size, embedding_dim, input_length=max_length),
    keras.layers.Bidirectional(keras.layers.LSTM(64)),
    keras.layers.Dense(24, activation='relu'),
    keras.layers.Dense(1, activation='sigmoid')
])
# compile model
model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
# model summary
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
```

```
 embedding (Embedding)       (None, 200, 100)          300000

 bidirectional (Bidirectiona  (None, 128)              84480
 l)

 dense (Dense)               (None, 24)                3096

 dense_1 (Dense)             (None, 1)                 25

=================================================================
Total params: 387,601
Trainable params: 387,601
Non-trainable params: 0
_________________________________________________________________
```

```
num_epochs = 10
history = model.fit(train_padded, train_labels,
                    epochs=num_epochs, verbose=1,
                    validation_split=0.1)
```

```
Epoch 1/10
1055/1055 [==============================] - 169s 160ms/step - loss: 0.6924 - accuracy: 0.5144 - val_loss:
0.6936 - val_accuracy: 0.5032
Epoch 2/10
1055/1055 [==============================] - 166s 157ms/step - loss: 0.6923 - accuracy: 0.5171 - val_loss:
0.6934 - val_accuracy: 0.5088
Epoch 3/10
1055/1055 [==============================] - 166s 157ms/step - loss: 0.6921 - accuracy: 0.5176 - val_loss:
0.6938 - val_accuracy: 0.5075
Epoch 4/10
1055/1055 [==============================] - 165s 157ms/step - loss: 0.6921 - accuracy: 0.5186 - val_loss:
0.6935 - val_accuracy: 0.4971
Epoch 5/10
1055/1055 [==============================] - 164s 155ms/step - loss: 0.6921 - accuracy: 0.5185 - val_loss:
0.6941 - val_accuracy: 0.5003
Epoch 6/10
1055/1055 [==============================] - 167s 158ms/step - loss: 0.6920 - accuracy: 0.5183 - val_loss:
0.6942 - val_accuracy: 0.4971
Epoch 7/10
1055/1055 [==============================] - 166s 157ms/step - loss: 0.6921 - accuracy: 0.5183 - val_loss:
0.6938 - val_accuracy: 0.4997
Epoch 8/10
1055/1055 [==============================] - 163s 155ms/step - loss: 0.6920 - accuracy: 0.5182 - val_loss:
0.6942 - val_accuracy: 0.5029
Epoch 9/10
1055/1055 [==============================] - 164s 156ms/step - loss: 0.6920 - accuracy: 0.5188 - val_loss:
0.6937 - val_accuracy: 0.5088
Epoch 10/10
1055/1055 [==============================] - 164s 155ms/step - loss: 0.6919 - accuracy: 0.5198 - val_loss:
0.6938 - val_accuracy: 0.5021
```

```
model.evaluate(test_sentences)
```

```
prediction = model.predict(test_padded)
# Get labels based on probability 1 if p>= 0.5 else 0
pred_labels = []
for i in prediction:
    if i >= 0.5:
        pred_labels.append(1)
    else:
        pred_labels.append(0)
print("Accuracy of prediction on test set : ", accuracy_score(test_labels,pred_labels))
```

```
391/391 [==============================] - 14s 35ms/step
Accuracy of prediction on test set :  0.51256
```

```
# reviews on which we need to predict
sentence = ["The movie was very touching and heart whelming",
```

```
            "I have never seen a terrible movie like this",
            "the movie plot is terrible but it had good acting"]

sequences = tokenizer.texts_to_sequences(sentence)

padded = pad_sequences(sequences, padding='post', maxlen=max_length)

prediction = model.predict(padded)
pred_labels = []
for i in prediction:
    if i >= 0.5:
        pred_labels.append(1)
    else:
        pred_labels.append(0)
for i in range(len(sentence)):
    print(sentence[i])
    if pred_labels[i] == 1:
        s = 'Positive'
    else:
        s = 'Negative'
    print("Predicted sentiment : ",s)
```

```
1/1 [==============================] - 0s 34ms/step
The movie was very touching and heart whelming
Predicted sentiment :  Positive
I have never seen a terrible movie like this
Predicted sentiment :  Positive
the movie plot is terrible but it had good acting
Predicted sentiment :  Positive
```