Sentiment Analysis

Old work related to the paper ' Convolutional Neural Networks for Sentence Classification '

- https://arxiv.org/pdf/1408.5882.pdf

It train a simple (CNN) with one layer of convolution on top of word vectors obtained from an unsupervised neural language model. These vectors were trained by Mikolov etal. (2013) on 100 billion words of Google News, and are publicly available.1 We initially keep the word vectors static and learn only the other parameters of the model. Despite little tuning of hyperparameters, this simple model achieves excellent results on multiple benchmarks.

What is new work we add on the old one and our contributions?

Instead of unsing only the CNN to make the Sentence Classification we will use CNN in as well as LSTM to generate a combination model of them(CNN-LSTM) and (LSTM-CNN)

CNNs

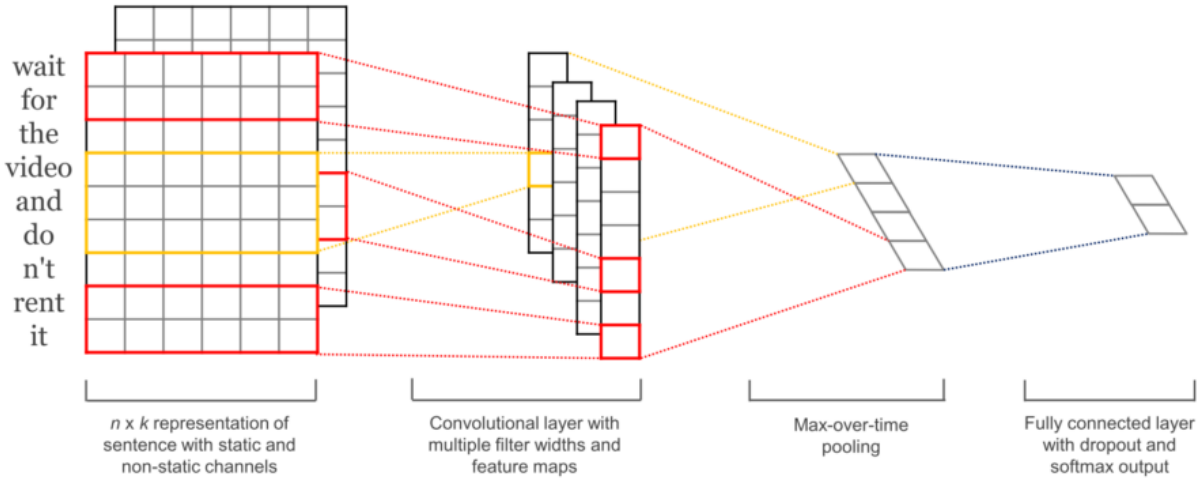
Convolutional Neural Networks (CNNs) are networks initially created for image-related tasks that can learn to capture specific features regardless of locality.

For a more concrete example of that, imagine we use CNNs to distinguish pictures of Cars vs. pictures of Dogs. Since CNNs learn to capture features regardless of where these might be, the CNN will learn that cars have wheels, and every time it sees a wheel, regardless of where it is on the picture, that feature will activate.

In our particular case, it could capture a negative phrase such as "don't like" regardless of where it happens in the tweet.

- I don't like watching those types of films
- That's the one thing I really don't like.
- I saw the movie, and I don't like how it ended.

```
import os
from IPython.display import Image
Image(filename="../input/images/CNNs.png")
```



LSTMs

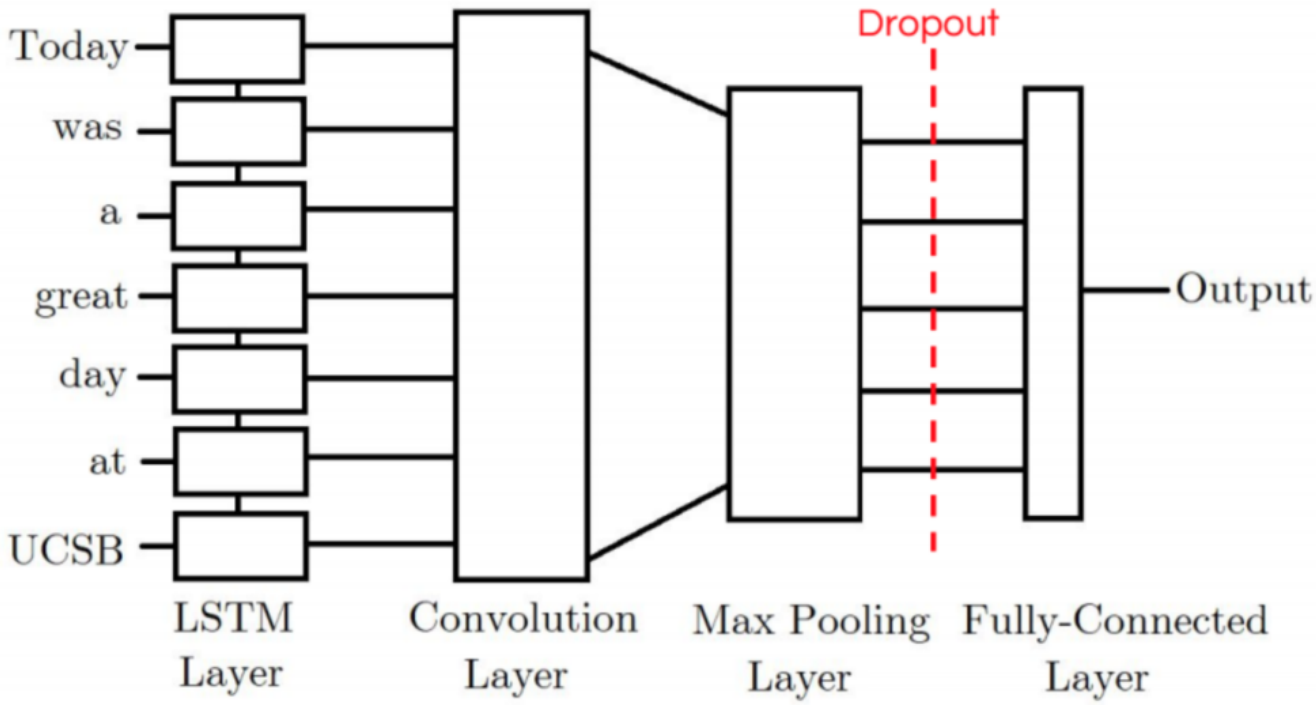
Long-Term Short Term Memory (LSTMs) are a type of network that has a memory that "remembers" previous data from the input and makes decisions based on that knowledge. These networks are more directly suited for written data inputs, since each word in a sentence has meaning based on the surrounding words (previous and upcoming words).

In our particular case, it is possible that an LSTM could allow us to capture changing sentiment in a tweet. For example, a sentence such as: At first I loved it, but then I ended up hating it. has words with conflicting sentiments that would end-up confusing a simple Feed-Forward network. The LSTM, on the other hand, could learn that sentiments expressed towards the end of a sentence mean more than those expressed at the start.

CNN-LSTM Model

The first model I tried was the CNN-LSTM Model. Our CNN-LSTM model combination consists of an initial convolution layer which will receive word embeddings as input. Its output will then be pooled to a smaller dimension which is then fed into an LSTM layer. The intuition behind this model is that the convolution layer will extract local features and the LSTM layer will then be able to use the ordering of said features to learn about the input's text ordering. In practice, this model is not as powerful as our other LSTM-CNN model proposed.

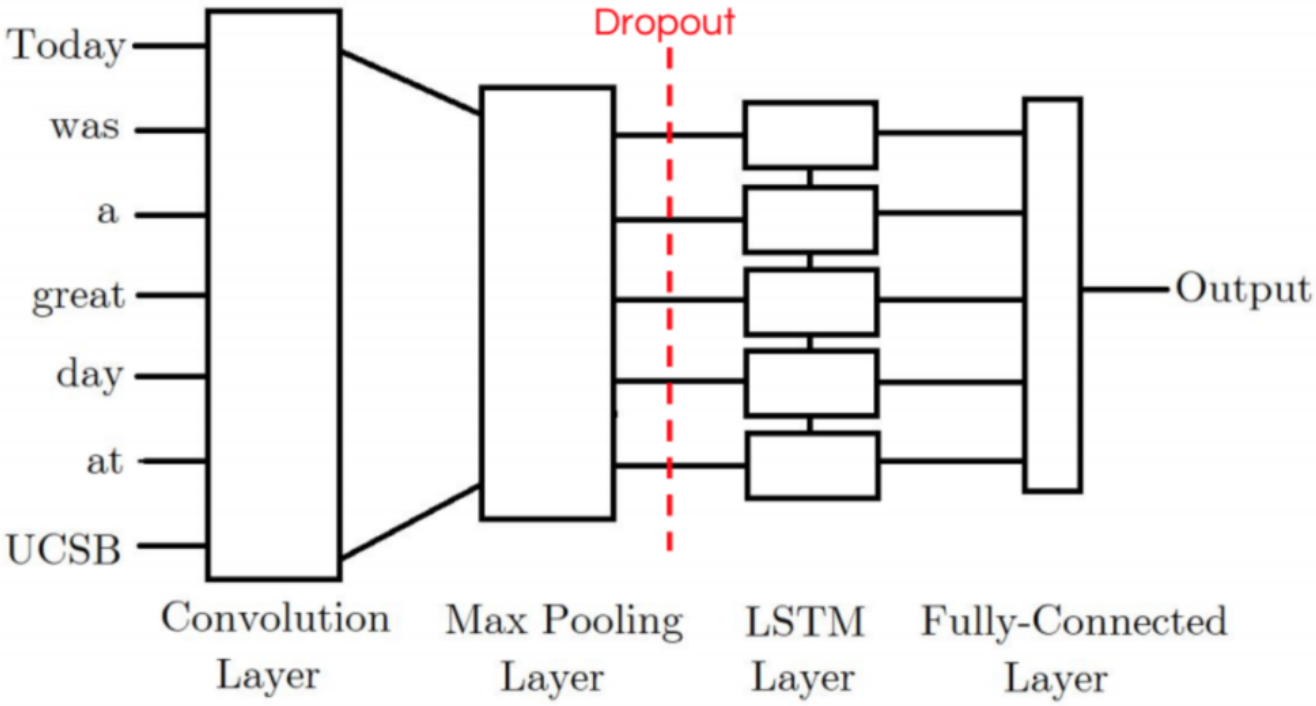
```
Image(filename="../input/images/LSTM_CNN_image.png")
```



LSTM-CNN Model

Our CNN-LSTM model consists of an initial LSTM layer which will receive word embeddings for each token in the tweet as inputs. The intuition is that its output tokens will store information not only of the initial token, but also any previous tokens; In other words, the LSTM layer is generating a new encoding for the original input. The output of the LSTM layer is then fed into a convolution layer which we expect will extract local features. Finally the convolution layer's output will be pooled to a smaller dimension and ultimately outputted as either a positive or negative label.

```
Image(filename="../input/images/CNN-LSTM_image.png")
```



```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets pr
# You can also write temporary files to /kaggle/temp/, but they won't be saved outs
```

```
/kaggle/input/images/LSTM_CNN_image.png
/kaggle/input/images/CNN-LSTM_image.png
/kaggle/input/images/CNNs.png
/kaggle/input/imdb-dataset-of-50k-movie-reviews/IMDB Dataset.csv
```

load needed libraries

```
import os
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize
from wordcloud import WordCloud,STOPWORDS
from bs4 import BeautifulSoup
import re,string,unicodedata
import os
from IPython.display import Image

from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.naive_bayes import GaussianNB, MultinomialNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix
#from xgboost.sklearn import XGBClassifier

import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.layers import Dense,Input, Embedding,LSTM,Dropout,Conv1D, Max
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.utils import plot_model
#import transformers
#import tokenizers
```

load our data

```
data = pd.read_csv('/kaggle/input/imdb-dataset-of-50k-movie-reviews/IMDB Dataset.cs
df= data.copy()
```

Data preprocessing

```
data.head()
```

	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

```
data.shape
```

```
(50000, 2)
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 2 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   review     50000 non-null  object
 1   sentiment  50000 non-null  object
dtypes: object(2)
memory usage: 781.4+ KB
```

```
data.describe()
```

	review	sentiment
count	50000	50000
unique	49582	2
top	Loved today's show!!! It was a variety and not...	positive
freq	5	25000

```
data['sentiment'].value_counts()
```

```
positive    25000
negative    25000
Name: sentiment, dtype: int64
```

Data Cleaning

```
data.isnull().sum()
```

```
review       0
sentiment    0
dtype: int64
```

```
data.duplicated().sum()
```

```
418
```

```
data.drop_duplicates(inplace = True)
```

```
data.shape
```

```
(49582, 2)
```

```
stop = stopwords.words('english')
wl = WordNetLemmatizer()
```

```
mapping = {"ain't": "is not", "aren't": "are not","can't": "cannot",
           "'cause": "because", "could've": "could have", "couldn't": "could not",
           "didn't": "did not",  "doesn't": "does not", "don't": "do not", "hadn't"
           "hasn't": "has not", "haven't": "have not", "he'd": "he would","he'll":
           "he's": "he is", "how'd": "how did", "how'd'y": "how do you", "how'll":
           "how's": "how is",  "I'd": "I would", "I'd've": "I would have", "I'll":
           "I'll've": "I will have","I'm": "I am", "I've": "I have", "i'd": "i woul
           "i'd've": "i would have", "i'll": "i will",  "i'll've": "i will have",
           "i'm": "i am", "i've": "i have", "isn't": "is not", "it'd": "it would",
           "it'd've": "it would have", "it'll": "it will", "it'll've": "it will hav
           "it's": "it is", "let's": "let us", "ma'am": "madam", "mayn't": "may not
           "might've": "might have","mightn't": "might not","mightn't've": "might n
           "must've": "must have", "mustn't": "must not", "mustn't've": "must not h
           "needn't": "need not", "needn't've": "need not have","o'clock": "of the
           "oughtn't": "ought not", "oughtn't've": "ought not have", "shan't": "sha
           "sha'n't": "shall not", "shan't've": "shall not have", "she'd": "she wou
           "she'd've": "she would have", "she'll": "she will", "she'll've": "she wi
           "she's": "she is", "should've": "should have", "shouldn't": "should not"
           "shouldn't've": "should not have", "so've": "so have","so's": "so as", "
           "that'd": "that would", "that'd've": "that would have", "that's": "that
           "there'd": "there would", "there'd've": "there would have", "there's": "
           "here's": "here is","they'd": "they would", "they'd've": "they would hav
           "they'll": "they will", "they'll've": "they will have", "they're": "they
           "they've": "they have", "to've": "to have", "wasn't": "was not", "we'd":
           "we'd've": "we would have", "we'll": "we will", "we'll've": "we will hav
           "we're": "we are", "we've": "we have", "weren't": "were not",
           "what'll": "what will", "what'll've": "what will have","what're": "what
           "what's": "what is", "what've": "what have", "when's": "when is", "when"
           "where'd": "where did", "where's": "where is", "where've": "where have",
           "who'll've": "who will have", "who's": "who is", "who've": "who have", "
           "why've": "why have", "will've": "will have", "won't": "will not", "won'
           "would've": "would have", "wouldn't": "would not", "wouldn't've": "would
           "y'all": "you all", "y'all'd": "you all would","y'all'd've": "you all wo
           "y'all're": "you all are","y'all've": "you all have","you'd": "you would
           "you'd've": "you would have", "you'll": "you will", "you'll've": "you wi
           "you're": "you are", "you've": "you have" }


import nltk
nltk.download('wordnet')
```

```
    [nltk_data] Error loading wordnet: <urlopen error [Errno -3] Temporary
    [nltk_data]     failure in name resolution>
    False
```

```
#function to clean data

def clean_text(text,lemmatize = True):
    soup = BeautifulSoup(text, "html.parser") #remove html tags
    text = soup.get_text()
    text = ' '.join([mapping[t] if t in mapping else t for t in text.split(" ")]) #
    emoji_clean= re.compile("["
                            u"\U0001F600-\U0001F64F"  # emoticons
                            u"\U0001F300-\U0001F5FF"  # symbols & pictographs
                            u"\U0001F680-\U0001F6FF"  # transport & map symbols
                            u"\U0001F1E0-\U0001F1FF"  # flags (iOS)
                            u"\U00002702-\U000027B0"
                            u"\U000024C2-\U0001F251"
                            "]+", flags=re.UNICODE)
    text = emoji_clean.sub(r'',text)
    text = re.sub(r'\.(?=\S)', '. ',text) #add space after full stop
    text = re.sub(r'http\S+', '', text) #remove urls
    text = "".join([word.lower() for word in text if word not in string.punctuation
    #tokens = re.split('\W+', text) #create tokens
    if lemmatize:
        text = " ".join([wl.lemmatize(word) for word in text.split() if word not in
    else:
        text = " ".join([word for word in text.split() if word not in stop and word
    return text


data['review']=data['review'].apply(clean_text,lemmatize = True)


#converting target variable to numeric labels
data.sentiment = [ 1 if each == "positive" else 0 for each in data.sentiment]
data.head()
```

	review	sentiment
0	one reviewer mentioned watching oz episode hoo...	1
1	wonderful little production filming technique ...	1
2	thought wonderful way spend time hot summer we...	1
3	basically family little boy jake think zombie ...	0
4	petter matteis love time money visually stunni...	1

Spliting the training dataset

```
#splitting into train and test
train, test= train_test_split(data, test_size=0.2, random_state=42)

#train dataset
Xtrain, ytrain = train['review'], train['sentiment']

#test dataset
Xtest, ytest = test['review'], test['sentiment']

print(Xtrain.shape,ytrain.shape)
print(Xtest.shape,ytest.shape)
```

```
(39665,) (39665,)
(9917,) (9917,)
```

Vectorizing data

```
vect = TfidfVectorizer()
Xtrain_vect= vect.fit_transform(Xtrain)
Xtest_vect = vect.transform(Xtest)


count_vect = CountVectorizer()
Xtrain_count = count_vect.fit_transform(Xtrain)
Xtest_count = count_vect.transform(Xtest)
```

1- LSTM model

```
MAX_VOCAB_SIZE = 10000
tokenizer = Tokenizer(num_words = MAX_VOCAB_SIZE,oov_token="<oov>")
tokenizer.fit_on_texts(Xtrain)
word_index = tokenizer.word_index
#print(word_index)
V = len(word_index)
print("Vocabulary of the dataset is : ",V)
```

```
Vocabulary of the dataset is :  126096
```

```
##create sequences of reviews
seq_train = tokenizer.texts_to_sequences(Xtrain)
seq_test =  tokenizer.texts_to_sequences(Xtest)
```

```
#choice of maximum length of sequences
seq_len_list = [len(i) for i in seq_train + seq_test]

#if we take the direct maximum then
max_len=max(seq_len_list)
print('Maximum length of sequence in the list: {}'.format(max_len))
```

```
Maximum length of sequence in the list: 1406
```

```
# when setting the maximum length of sequence, variability around the average is us
max_seq_len = np.mean(seq_len_list) + 2 * np.std(seq_len_list)
max_seq_len = int(max_seq_len)
print('Maximum length of the sequence when considering data only two standard devia
```

```
Maximum length of the sequence when considering data only two standard deviat
```

```
perc_covered = np.sum(np.array(seq_len_list) < max_seq_len) / len(seq_len_list)*100
print('The above calculated number coveres approximately {} % of data'.format(np.ro
```

```
The above calculated number coveres approximately 94.51 % of data
```

```
#create padded sequences
pad_train=pad_sequences(seq_train,truncating = 'post', padding = 'pre',maxlen=max_s
pad_test=pad_sequences(seq_test,truncating = 'post', padding = 'pre',maxlen=max_seq
```

```
#Splitting training set for validation purposes
Xtrain,Xval,ytrain,yval=train_test_split(pad_train,ytrain,
                                         test_size=0.2,random_state=10)


def lstm_model(Xtrain,Xval,ytrain,yval,V,D,maxlen,epochs):

    print("----Building the model----")
    i = Input(shape=(maxlen,))
    x = Embedding(V + 1, D,input_length = maxlen)(i)
    x = BatchNormalization()(x)
    x = Dropout(0.3)(x)
    x = Conv1D(32,5,activation = 'relu')(x)
    x = Dropout(0.3)(x)
    x = MaxPooling1D(2)(x)
    x = Bidirectional(LSTM(128,return_sequences=True))(x)
    x = LSTM(64)(x)
    x = Dropout(0.5)(x)
    x = Dense(1, activation='sigmoid')(x)
    model = Model(i, x)
    model.summary()

    #Training the LSTM
    print("----Training the network----")
    model.compile(optimizer= Adam(0.0005),
              loss='binary_crossentropy',
              metrics=['accuracy'])

    r = model.fit(Xtrain,ytrain,
                  validation_data = (Xval,yval),
                  epochs = epochs,
                  verbose = 2,
                  batch_size = 32)
                  #callbacks = callbacks
    print("Train score:", model.evaluate(Xtrain,ytrain))
    print("Validation score:", model.evaluate(Xval,yval))
    n_epochs = len(r.history['loss'])

    return r,model,n_epochs


D = 64 #embedding dims
epochs = 5
r,model,n_epochs = lstm_model(Xtrain,Xval,ytrain,yval,V,D,max_seq_len,epochs)
```

```
----Building the model----
2021-12-12 22:09:27.829332: I tensorflow/stream_executor/cuda/cuda_gpu_executo
2021-12-12 22:09:27.930067: I tensorflow/stream_executor/cuda/cuda_gpu_executo
2021-12-12 22:09:27.930789: I tensorflow/stream_executor/cuda/cuda_gpu_executo
2021-12-12 22:09:27.931935: I tensorflow/core/platform/cpu_feature_guard.cc:14
To enable them in other operations, rebuild TensorFlow with the appropriate co
2021-12-12 22:09:27.933150: I tensorflow/stream_executor/cuda/cuda_gpu_executo
2021-12-12 22:09:27.933817: I tensorflow/stream_executor/cuda/cuda_gpu_executo
2021-12-12 22:09:27.934466: I tensorflow/stream_executor/cuda/cuda_gpu_executo
2021-12-12 22:09:29.782258: I tensorflow/stream_executor/cuda/cuda_gpu_executo
2021-12-12 22:09:29.783036: I tensorflow/stream_executor/cuda/cuda_gpu_executo
2021-12-12 22:09:29.783740: I tensorflow/stream_executor/cuda/cuda_gpu_executo
2021-12-12 22:09:29.784320: I tensorflow/core/common_runtime/gpu/gpu_device.cc
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 293)]             0
_________________________________________________________________
embedding (Embedding)        (None, 293, 64)           8070208
_________________________________________________________________
batch_normalization (BatchNo (None, 293, 64)           256
_________________________________________________________________
dropout (Dropout)            (None, 293, 64)           0
_________________________________________________________________
conv1d (Conv1D)              (None, 289, 32)           10272
_________________________________________________________________
dropout_1 (Dropout)          (None, 289, 32)           0
_________________________________________________________________
max_pooling1d (MaxPooling1D) (None, 144, 32)           0
_________________________________________________________________
bidirectional (Bidirectional (None, 144, 256)          164864
_________________________________________________________________
lstm_1 (LSTM)                (None, 64)                82176
_________________________________________________________________
dropout_2 (Dropout)          (None, 64)                0
_________________________________________________________________
dense (Dense)                (None, 1)                 65
=================================================================
Total params: 8,327,841
Trainable params: 8,327,713
Non-trainable params: 128
_________________________________________________________________
----Training the network----
Epoch 1/5
2021-12-12 22:09:30.880447: I tensorflow/compiler/mlir/mlir_graph_optimization
2021-12-12 22:09:35.216930: I tensorflow/stream_executor/cuda/cuda_dnn.cc:369]
992/992 - 36s - loss: 0.4159 - accuracy: 0.7981 - val_loss: 0.2849 - val_accu
Epoch 2/5
992/992 - 25s - loss: 0.2508 - accuracy: 0.8995 - val_loss: 0.2795 - val_accu
Epoch 3/5
```

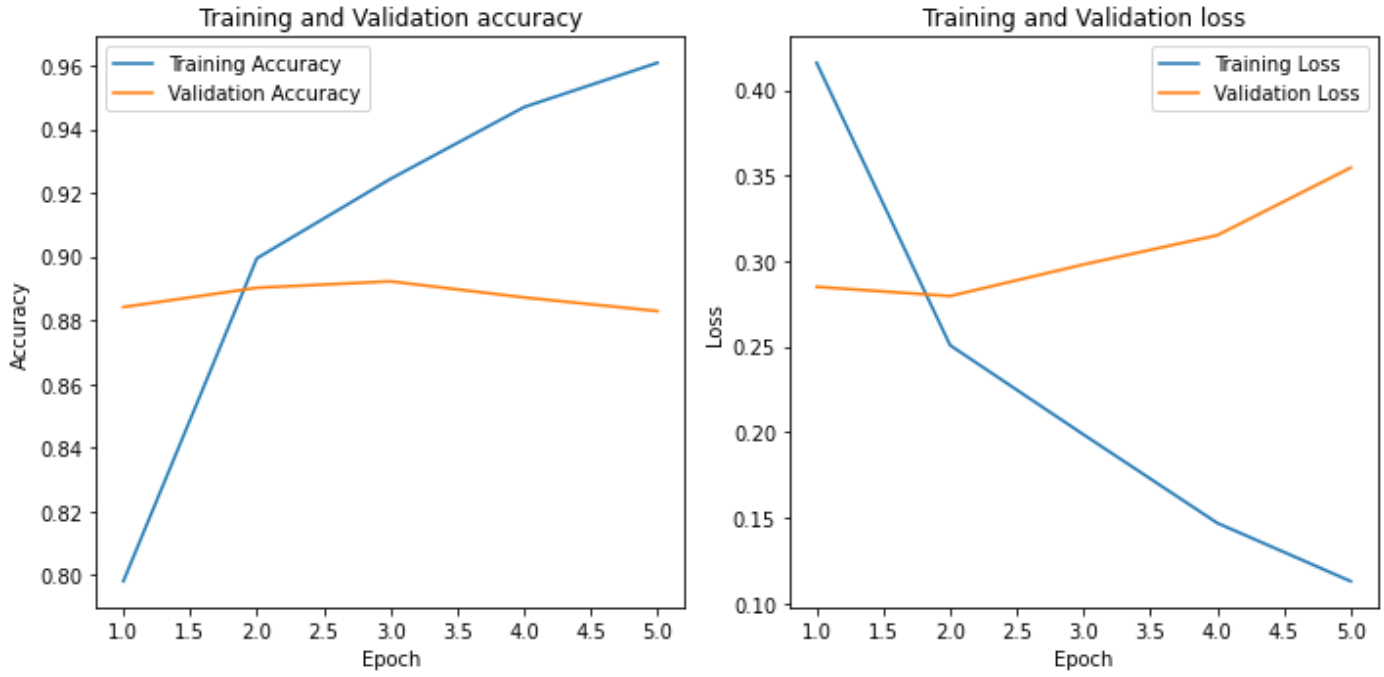
```
Epoch 3/5
992/992 - 25s - loss: 0.1985 - accuracy: 0.9244 - val_loss: 0.2980 - val_accu
Epoch 4/5
992/992 - 25s - loss: 0.1469 - accuracy: 0.9469 - val_loss: 0.3151 - val_accu
Epoch 5/5
992/992 - 25s - loss: 0.1128 - accuracy: 0.9608 - val_loss: 0.3546 - val_accu
992/992 [==============================] - 9s 9ms/step - loss: 0.0440 - accura
Train score: [0.04403455927968025, 0.9884659051895142]
248/248 [==============================] - 2s 9ms/step - loss: 0.3546 - accura
Validation score: [0.3546118140220642, 0.8828942179679871]
```

```
def plotLearningCurve(history,epochs):

    epochRange = range(1,epochs+1)
    fig , ax = plt.subplots(1,2,figsize = (10,5))

    ax[0].plot(epochRange,history.history['accuracy'],label = 'Training Accuracy')
    ax[0].plot(epochRange,history.history['val_accuracy'],label = 'Validation Accur
    ax[0].set_title('Training and Validation accuracy')
    ax[0].set_xlabel('Epoch')
    ax[0].set_ylabel('Accuracy')
    ax[0].legend()
    ax[1].plot(epochRange,history.history['loss'],label = 'Training Loss')
    ax[1].plot(epochRange,history.history['val_loss'],label = 'Validation Loss')
    ax[1].set_title('Training and Validation loss')
    ax[1].set_xlabel('Epoch')
    ax[1].set_ylabel('Loss')
    ax[1].legend()
    fig.tight_layout()
    plt.show()
```

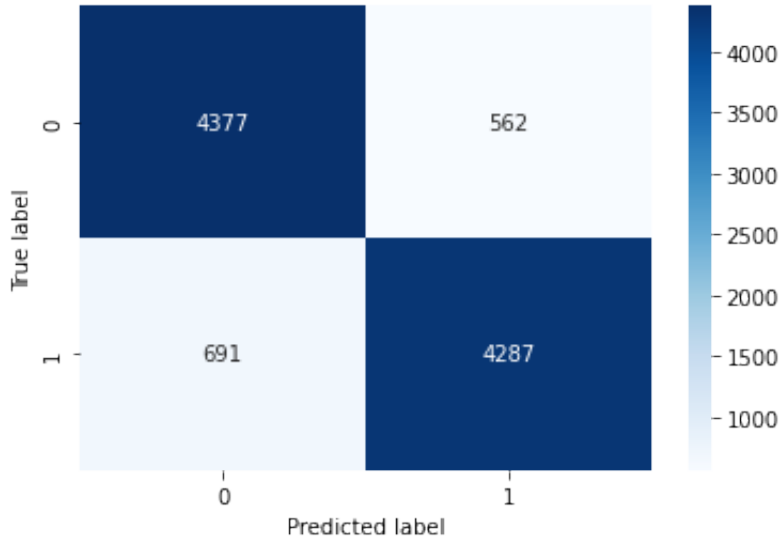
```
plotLearningCurve(r,n_epochs)
```




```
print("Evaluate Model Performance on Test set")
result = model.evaluate(pad_test,ytest)
print(dict(zip(model.metrics_names, result)))
```

```
Evaluate Model Performance on Test set
310/310 [==============================] - 3s 10ms/step - loss: 0.3779 - accu
{'loss': 0.37792181968688965, 'accuracy': 0.8736513257026672}
```

```
#Generate predictions for the test dataset
ypred = model.predict(pad_test)
ypred = ypred>0.5
#Get the confusion matrix
cf_matrix = confusion_matrix(ytest, ypred)
sns.heatmap(cf_matrix,annot = True,fmt ='g', cmap='Blues')
plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.show()
```



2-CNN model

```
from sklearn.metrics import classification_report, confusion_matrix, roc_curve, roc

from sklearn.preprocessing import StandardScaler, MinMaxScaler

from sklearn.linear_model import LogisticRegression

from sklearn.model_selection import train_test_split

from sklearn import datasets

from tensorflow.keras.preprocessing import sequence
from sklearn.datasets import fetch_20newsgroups
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Input, Dense, Embedding, Conv1D, MaxPool1D, Dro
from tensorflow.keras.layers import Flatten
from tensorflow.keras.preprocessing import sequence
import numpy as np
import string
import re


df.sentiment = df.sentiment.map({ 'negative': 0, 'positive': 1 })

text = df.review.tolist()
label = df.sentiment.tolist()


X_train, X_test, y_train, y_test = train_test_split(text, label, test_size=0.2, ran


translator = str.maketrans(string.punctuation, ' '*len(string.punctuation)) #map pu
```

```
X_train_clean = []
X_test_clean = []
clean = re.compile(r'<[^>]+>')
for i, test in enumerate(X_train):
    tmp_text = test.lower()
    tmp_text = tmp_text.replace('\n', '')
    tmp_text = clean.sub('', tmp_text)
    tmp_text = tmp_text.translate(translator)
    X_train_clean.append(tmp_text)

for i, test in enumerate(X_test):
    tmp_text = test.lower()
    tmp_text = tmp_text.replace('\n', '')
    tmp_text = clean.sub('', tmp_text)
    tmp_text = tmp_text.translate(translator)
    X_test_clean.append(tmp_text)

X_train_clean = np.array(X_train_clean)
X_test_clean = np.array(X_test_clean)

X_train = X_train_clean
X_test = X_test_clean


top_words = 40000
tokenizer = Tokenizer(num_words=top_words)
tokenizer.fit_on_texts(X_train)
X_train = tokenizer.texts_to_sequences(X_train)
X_test = tokenizer.texts_to_sequences(X_test)


max_words = 100
X_train = sequence.pad_sequences(X_train, maxlen=max_words, padding='post')
X_test = sequence.pad_sequences(X_test, maxlen=max_words, padding='post')


y_train = np.array(y_train)
y_test = np.array(y_test)


X_train.shape
```

```
(40000, 100)
```

Double-click (or enter) to edit

```
model = Sequential()
model.add(Embedding(20000,32, input_length=100))
model.add(Conv1D(256, 3, activation='relu', padding='same'))
model.add(MaxPool1D(2))
model.add(Dropout(0.2))
model.add(Conv1D(128, 3, activation='relu', padding='same'))
model.add(MaxPool1D(2))
model.add(Dropout(0.2))
model.add(Flatten())
model.add(Dense(250, activation='relu'))

model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding_1 (Embedding)      (None, 100, 32)           640000    
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 100, 256)          24832     
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 50, 256)           0         
_________________________________________________________________
dropout_3 (Dropout)          (None, 50, 256)           0         
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 50, 128)           98432     
_________________________________________________________________
max_pooling1d_2 (MaxPooling1 (None, 25, 128)           0         
_________________________________________________________________
dropout_4 (Dropout)          (None, 25, 128)           0         
_________________________________________________________________
flatten (Flatten)            (None, 3200)              0         
_________________________________________________________________
dense_1 (Dense)              (None, 250)               800250    
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 251       
=================================================================
Total params: 1,563,765
Trainable params: 1,563,765
Non-trainable params: 0
_________________________________________________________________
```

```
model.fit(X_train, y_train, validation_data=(X_test,y_test), epochs=20, batch_size=
```

```
Epoch 1/20
313/313 - 3s - loss: 0.4412 - accuracy: 0.7645 - val_loss: 0.3008 - val_accura
Epoch 2/20
313/313 - 2s - loss: 0.2369 - accuracy: 0.9067 - val_loss: 0.2934 - val_accura
Epoch 3/20
313/313 - 2s - loss: 0.1531 - accuracy: 0.9426 - val_loss: 0.3477 - val_accura
Epoch 4/20
313/313 - 2s - loss: 0.0977 - accuracy: 0.9652 - val_loss: 0.4190 - val_accura
Epoch 5/20
313/313 - 2s - loss: 0.0564 - accuracy: 0.9809 - val_loss: 0.5960 - val_accura
Epoch 6/20
313/313 - 2s - loss: 0.0348 - accuracy: 0.9879 - val_loss: 0.6625 - val_accura
Epoch 7/20
313/313 - 2s - loss: 0.0324 - accuracy: 0.9883 - val_loss: 0.7332 - val_accura
Epoch 8/20
313/313 - 2s - loss: 0.0210 - accuracy: 0.9924 - val_loss: 0.8565 - val_accura
Epoch 9/20
313/313 - 2s - loss: 0.0182 - accuracy: 0.9942 - val_loss: 0.8772 - val_accura
Epoch 10/20
313/313 - 2s - loss: 0.0143 - accuracy: 0.9950 - val_loss: 0.8995 - val_accura
Epoch 11/20
313/313 - 2s - loss: 0.0132 - accuracy: 0.9953 - val_loss: 0.9881 - val_accura
Epoch 12/20
313/313 - 2s - loss: 0.0124 - accuracy: 0.9959 - val_loss: 1.1098 - val_accura
Epoch 13/20
313/313 - 2s - loss: 0.0114 - accuracy: 0.9963 - val_loss: 1.0606 - val_accura
Epoch 14/20
313/313 - 2s - loss: 0.0091 - accuracy: 0.9968 - val_loss: 1.1989 - val_accura
Epoch 15/20
313/313 - 2s - loss: 0.0085 - accuracy: 0.9967 - val_loss: 1.1908 - val_accura
Epoch 16/20
313/313 - 2s - loss: 0.0075 - accuracy: 0.9973 - val_loss: 1.3590 - val_accura
Epoch 17/20
313/313 - 2s - loss: 0.0091 - accuracy: 0.9972 - val_loss: 1.1516 - val_accura
Epoch 18/20
313/313 - 2s - loss: 0.0092 - accuracy: 0.9968 - val_loss: 1.0704 - val_accura
Epoch 19/20
313/313 - 2s - loss: 0.0061 - accuracy: 0.9980 - val_loss: 1.4095 - val_accura
Epoch 20/20
313/313 - 2s - loss: 0.0068 - accuracy: 0.9976 - val_loss: 1.3270 - val_accura
<keras.callbacks.History at 0x7fb7580f6410>
```

```
print("Evaluate Model Performance on Test set")
result = model.evaluate(X_test,y_test)
print(dict(zip(model.metrics_names, result)))
```

```
Evaluate Model Performance on Test set
313/313 [==============================] - 1s 2ms/step - loss: 1.3270 - accura
{'loss': 1.3270049095153809, 'accuracy': 0.8485000133514404}
```

3- LSTM-CNN Model

```
model = Sequential()
model.add(Embedding(20000,32, input_length=100))
model.add(Conv1D(256, 3, activation='relu', input_shape=(178, 1), padding='same'))
model.add(MaxPool1D(2))
model.add(Dropout(0.2))
model.add(Conv1D(128, 3, activation='relu', padding='same'))
model.add(MaxPool1D(2))
model.add(Dropout(0.2))
model.add(LSTM(64, return_sequences=True))
model.add(LSTM(32))
model.add(Flatten())
model.add(Dense(250, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding_2 (Embedding)      (None, 100, 32)           640000    
_________________________________________________________________
conv1d_3 (Conv1D)            (None, 100, 256)          24832     
_________________________________________________________________
max_pooling1d_3 (MaxPooling1 (None, 50, 256)           0         
_________________________________________________________________
dropout_5 (Dropout)          (None, 50, 256)           0         
_________________________________________________________________
conv1d_4 (Conv1D)            (None, 50, 128)           98432     
_________________________________________________________________
max_pooling1d_4 (MaxPooling1 (None, 25, 128)           0         
_________________________________________________________________
dropout_6 (Dropout)          (None, 25, 128)           0         
_________________________________________________________________
lstm_2 (LSTM)                (None, 25, 64)            49408     
_________________________________________________________________
lstm_3 (LSTM)                (None, 32)                12416     
_________________________________________________________________
flatten_1 (Flatten)          (None, 32)                0         
_________________________________________________________________
dense_3 (Dense)              (None, 250)               8250      
_________________________________________________________________
dropout_7 (Dropout)          (None, 250)               0         
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 251       
=================================================================
Total params: 833,589
Trainable params: 833,589
Non-trainable params: 0
_________________________________________________________________
```

```
model.fit(X_train, y_train, validation_data=(X_test,y_test), epochs=20, batch_size=
```

```
Epoch 1/20
313/313 - 6s - loss: 0.4301 - accuracy: 0.7793 - val_loss: 0.3113 - val_accura
Epoch 2/20
313/313 - 3s - loss: 0.2304 - accuracy: 0.9087 - val_loss: 0.2933 - val_accura
Epoch 3/20
313/313 - 3s - loss: 0.1571 - accuracy: 0.9412 - val_loss: 0.3194 - val_accura
Epoch 4/20
313/313 - 3s - loss: 0.0962 - accuracy: 0.9670 - val_loss: 0.4530 - val_accura
Epoch 5/20
313/313 - 3s - loss: 0.0599 - accuracy: 0.9801 - val_loss: 0.5109 - val_accura
Epoch 6/20
313/313 - 3s - loss: 0.0412 - accuracy: 0.9868 - val_loss: 0.5940 - val_accura
Epoch 7/20
313/313 - 3s - loss: 0.0291 - accuracy: 0.9916 - val_loss: 0.6541 - val_accura
Epoch 8/20
313/313 - 3s - loss: 0.0261 - accuracy: 0.9920 - val_loss: 0.6180 - val_accura
Epoch 9/20
313/313 - 3s - loss: 0.0225 - accuracy: 0.9934 - val_loss: 0.6249 - val_accura
Epoch 10/20
313/313 - 3s - loss: 0.0178 - accuracy: 0.9950 - val_loss: 0.7083 - val_accura
Epoch 11/20
313/313 - 3s - loss: 0.0145 - accuracy: 0.9959 - val_loss: 0.7366 - val_accura
Epoch 12/20
313/313 - 3s - loss: 0.0163 - accuracy: 0.9951 - val_loss: 0.7254 - val_accura
Epoch 13/20
313/313 - 3s - loss: 0.0101 - accuracy: 0.9971 - val_loss: 0.7588 - val_accura
Epoch 14/20
313/313 - 3s - loss: 0.0104 - accuracy: 0.9969 - val_loss: 0.8731 - val_accura
Epoch 15/20
313/313 - 3s - loss: 0.0118 - accuracy: 0.9965 - val_loss: 0.6483 - val_accura
Epoch 16/20
313/313 - 3s - loss: 0.0103 - accuracy: 0.9970 - val_loss: 0.8171 - val_accura
Epoch 17/20
313/313 - 3s - loss: 0.0090 - accuracy: 0.9975 - val_loss: 0.7759 - val_accura
Epoch 18/20
313/313 - 3s - loss: 0.0092 - accuracy: 0.9974 - val_loss: 0.8401 - val_accura
Epoch 19/20
313/313 - 3s - loss: 0.0094 - accuracy: 0.9969 - val_loss: 0.8314 - val_accura
Epoch 20/20
313/313 - 3s - loss: 0.0072 - accuracy: 0.9979 - val_loss: 0.8461 - val_accura
<keras.callbacks.History at 0x7fb7586968d0>
```

```
print("Evaluate Model Performance on Test set")
result = model.evaluate(X_test,y_test)
print(dict(zip(model.metrics_names, result)))
```

```
Evaluate Model Performance on Test set
313/313 [==============================] - 1s 4ms/step - loss: 0.8461 - accura
{'loss': 0.8460676670074463, 'accuracy': 0.8507999777793884}
```

Sources

- http://konukoii.com/blog/2018/02/19/twitter-sentiment-analysis-using-combined-lstm-cnn-models/
- https://arxiv.org/pdf/1408.5882.pdf
- https://github.com/pytorch/ignite/blob/master/examples/notebooks/TextCNN.ipynb
- https://www.kaggle.com/raghav2002sharma/sentiment-classifier-with-cnn-bi-lstm
- https://www.kaggle.com/ashrafkhan94/imdb-review-comparison-using-cnn-lstm-bert
- https://www.kaggle.com/parth05rohilla/bi-lstm-and-cnn-model-top-10/notebook
- https://www.kaggle.com/c/movie-review-sentiment-analysis-kernels-only/code?competitionId=10025&searchQuery=cnn
- https://colab.research.google.com/github/d2l-ai/d2l-en-colab/blob/master/chapter_natural-language-processing-applications/sentiment-analysis-cnn.ipynb
- https://www.kaggle.com/nafisur/keras-models-lstm-cnn-gru-bidirectional-glove
- https://www.kaggle.com/derrelldsouza/imdb-sentiment-analysis-eda-ml-lstm-bert#5.-Predictive-Modelling-using-Deep-Learning
- https://www.kaggle.com/clementbrehard/imdb-conv1d-lstm