# # Submitted by Navneet Das 3433 Comp A

```
***'
Assignment 2-A Deep Learning
Binary classification using Deep Neural
Networks Example: Classify movie reviews into positive" reviews and "negative" reviews,
just based on the text content of the reviews. Use IMDB dataset
'***
```

In [3]:

```
import nltk
nltk.download('punkt')
# import nltk
nltk.download('wordnet')
nltk.download('stopwords')
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\navne\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\navne\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\navne\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Out[3]:

```
True
```

In [4]:

```
import pandas as pd
import numpy as np
import re

import string
from string import digits

import numpy as np

import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.utils import to_categorical


from nltk.tokenize import word_tokenize
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer

import matplotlib.pyplot as plt
import nltk
nltk.download('stopwords')
lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words('english'))
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\navne\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

In [5]:

```
df = pd.read_csv("DeepLearningData/IMDB Dataset.csv")
df.head()
```

Out[5]:

| | review | sentiment |
|---|---|---|
| **0** | One of the other reviewers has mentioned that ... | positive |
| **1** | A wonderful little production. <br /><br />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 |

In [6]:

```
from sklearn import preprocessing
le =  preprocessing.LabelEncoder()
df["sentiment"] = le.fit_transform(df['sentiment'])
```

```
In [7]: df.head
```

```
Out[7]: <bound method NDFrame.head of                                                   review  sentiment
0      One of the other reviewers has mentioned that ...          1
1      A wonderful little production. <br /><br />The...          1
2      I thought this was a wonderful way to spend ti...          1
3      Basically there's a family where a little boy ...          0
4      Petter Mattei's "Love in the Time of Money" is...          1
...                                                  ...        ...
49995  I thought this movie did a down right good job...          1
49996  Bad plot, bad dialogue, bad acting, idiotic di...          0
49997  I am a Catholic taught in parochial elementary...          0
49998  I'm going to have to disagree with the previou...          0
49999  No one expects the Star Trek movies to be high...          0

[50000 rows x 2 columns]>
```

```
In [8]: df.isnull().sum()
```

```
Out[8]: review       0
sentiment    0
dtype: int64
```

```
In [9]: X = df["review"]
y = df["sentiment"]
```

```
In [10]: def stringprocess(text):
    text = re.sub(r"what's", "what is ", text)
    text = re.sub(r"\'s", " is", text)
    text = re.sub(r"\'ve", " have ", text)
    text = re.sub(r"can't", "cannot ", text)
    text = re.sub(r"n't", " not ", text)
    text = re.sub(r"i'm", "i am ", text)
    text = re.sub(r"\'re", " are ", text)
    text = re.sub(r"\'d", " would ", text)
    text = re.sub(r"\'ll", " will ", text)
    text = re.sub(r"\'scuse", " excuse ", text)
    text = re.sub('\W', ' ', text)
    text = re.sub('\s+', ' ', text)
    text = text.strip(' ')

    return text
```

```
In [11]: def  textpreprocess(text):

    text = map(lambda x: x.lower(), text)
    text = map(lambda x: re.sub(r"https?://\S+|www\.\S+", "", x), text)
    text = map(lambda x: re.sub(re.compile(r"<.*?>|&([a-z0-9]+|#[0-9]{1,6}|#x[0-9a-f]{1,6});"),"", x), text)
    text = map(lambda x: re.sub(r'[^\x00-\x7f]',r' ', x), text)

    text = map(lambda x: x.translate(str.maketrans('', '', string.punctuation)), text) # Remove punctuations


    remove_digits = str.maketrans('', '', digits)
    text = [i.translate(remove_digits) for i in text]
    text = [w for w in text if not w in stop_words]
    text = ' '.join([lemmatizer.lemmatize(w) for w in text])
    text = text.strip()
    return text
```

```
In [13]: X = X.apply(lambda x: stringprocess(x))
word_tokens = X.apply(lambda x: word_tokenize(x))

preprocess_text = word_tokens.apply(lambda x: textpreprocess(x))
preprocess_text[0]
```

```
Out[13]: 'one reviewer mentioned watching  oz episode hooked right exactly happened br br first thing struck oz brutality unflinching sc
ene violence set right word go trust show faint hearted timid show pull punch regard drug sex violence hardcore classic use wor
d br br called oz nickname given oswald maximum security state penitentary focus mainly emerald city experimental section priso
n cell glass front face inwards privacy high agenda em city home many aryan muslim gangsta latino christian italian irish scuff
le death stare dodgy dealing shady agreement never far away br br would say main appeal show due fact go show would dare forget
pretty picture painted mainstream audience forget charm forget romance oz mess around first episode ever saw struck nasty surre
al could say ready watched developed taste oz got accustomed high level graphic violence violence injustice crooked guard sold
nickel inmate kill order get away well mannered middle class inmate turned prison bitch due lack street skill prison experience
watching oz may become comfortable uncomfortable viewing thats get touch darker side'
```

In [14]:
```
training_portion = 0.8
train_size = int(len(preprocess_text) * training_portion)

train_data = preprocess_text[0: train_size]
train_labels = np.array(y[0: train_size])

validation_data = preprocess_text[train_size:]
validation_labels = np.array(y[train_size:])


print(len(train_data))
print(len(train_labels))
print(len(validation_data))
print(len(validation_labels))
```

```
40000
40000
10000
10000
```

In [15]:
```
vocab_size = 500
oov_tok = '<OOV>'

tokenizer = Tokenizer(num_words = vocab_size, oov_token=oov_tok)
tokenizer.fit_on_texts(train_data)
word_index = tokenizer.word_index
dict(list(word_index.items())[0:10])
```

Out[15]:
```
{'<OOV>': 1,
 'br': 2,
 'movie': 3,
 'film': 4,
 'one': 5,
 'like': 6,
 'would': 7,
 'time': 8,
 'good': 9,
 'character': 10}
```

In [16]:
```
train_sequences = tokenizer.texts_to_sequences(train_data)
print(train_sequences[10])
```

```
[1, 1, 5, 1, 4, 1, 332, 96, 1, 172, 153, 1, 1, 2, 2, 25, 1, 94, 69, 3, 1, 59, 285, 1, 69, 1, 2, 2, 251, 217, 4, 1, 42, 183, 94,
121, 10, 1, 313, 439, 2, 2, 1, 4, 7, 1, 1, 1, 1, 2, 2, 57, 1, 51, 124, 305, 73, 1]
```

In [17]:
```
embedding_dim = 50
max_length = 70

trunc_type = 'post'
padding_type = 'post'
```

In [18]:
```
train_padded = pad_sequences(train_sequences, maxlen=max_length, padding=padding_type, truncating=trunc_type)
print(len(train_sequences[0]))
print(len(train_padded[0]))
```

```
170
70
```

In [19]:
```
train_padded[0]
```

Out[19]:
```
array([  5,   1,   1,  66,   1, 174,   1, 102, 494, 486,   2,   2,  25,
        28,   1,   1,   1,   1,  18, 449, 114, 102, 244,  32,   1,  26,
         1,   1,   1,  26,   1,   1,   1,   1, 266, 449,   1, 218, 254,
       244,   2,   2, 325,   1,   1, 255,   1,   1,   1,   1,   1,   1,
         1,   1, 382,   1,   1,   1,   1,   1,   1, 223,   1,   1, 200,
         1,   1, 382, 238,  39])
```

In [20]:
```
validation_sequences = tokenizer.texts_to_sequences(validation_data)
validation_padded = pad_sequences(validation_sequences, maxlen=max_length, padding=padding_type, truncating=trunc_type)

print(len(validation_sequences))
print(validation_padded.shape)
```

```
10000
(10000, 70)
```

In [21]:
```
reverse_word_index = dict([(value, key) for (key, value) in word_index.items()])

def decode_data(text):
    return ' '.join([reverse_word_index.get(i, '?') for i in text])
print(decode_data(train_padded[10]))
print('---')
print(train_data[10])
```

```
<OOV> <OOV> one <OOV> film <OOV> based around <OOV> everything rather <OOV> <OOV> br br first <OOV> pretty funny movie <OOV> fi
nd joke <OOV> funny <OOV> br br low budget film <OOV> never problem pretty interesting character <OOV> lost interest br br <OOV
> film would <OOV> <OOV> <OOV> <OOV> br br something <OOV> better try brother another <OOV> ? ? ? ? ? ? ? ? ? ? ? ?
---
phil alien one quirky film humour based around oddness everything rather actual punchlines br br first odd pretty funny movie p
rogressed find joke oddness funny anymore br br low budget film thats never problem pretty interesting character eventually los
t interest br br imagine film would appeal stoner currently partaking br br something similar better try brother another planet
```

In [22]:
```
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size, embedding_dim),
    tf.keras.layers.LSTM(64,activation='relu'),
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dense(16, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
model.summary()
```

```
Model: "sequential"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, None, 50)          25000

 lstm (LSTM)                 (None, 64)                29440

 dense (Dense)               (None, 32)                2080

 dense_1 (Dense)             (None, 16)                528

 dense_2 (Dense)             (None, 1)                 17

=================================================================
Total params: 57,065
Trainable params: 57,065
Non-trainable params: 0
_________________________________________________________________
```

In [23]:
```
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

In [24]:
```
from keras.utils.vis_utils import plot_model
plot_model(model, to_file='model_plot.png', show_shapes=True, show_layer_names=True)
```

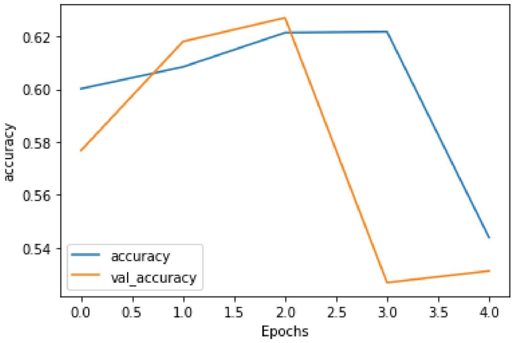
```
You must install pydot (`pip install pydot`) and install graphviz (see instructions at https://graphviz.gitlab.io/download/) (h
ttps://graphviz.gitlab.io/download/)) for plot_model to work.
```

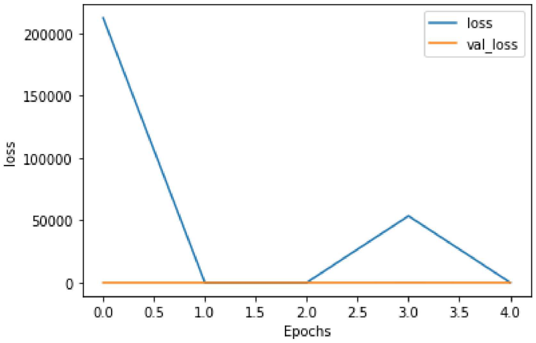
In [25]:
```
num_epochs = 5
history = model.fit(train_padded, train_labels, epochs=num_epochs, validation_data=(validation_padded, validation_labels), verbos
```

```
Epoch 1/5
1250/1250 - 29s - loss: 212559.0156 - accuracy: 0.6002 - val_loss: 0.6744 - val_accuracy: 0.5768 - 29s/epoch - 23ms/step
Epoch 2/5
1250/1250 - 27s - loss: 0.6560 - accuracy: 0.6085 - val_loss: 0.6473 - val_accuracy: 0.6181 - 27s/epoch - 22ms/step
Epoch 3/5
1250/1250 - 27s - loss: 3.2554 - accuracy: 0.6215 - val_loss: 0.6423 - val_accuracy: 0.6271 - 27s/epoch - 22ms/step
Epoch 4/5
1250/1250 - 25s - loss: 53488.5508 - accuracy: 0.6218 - val_loss: 35.6841 - val_accuracy: 0.5268 - 25s/epoch - 20ms/step
Epoch 5/5
1250/1250 - 26s - loss: 5.9972 - accuracy: 0.5439 - val_loss: 7.2854 - val_accuracy: 0.5312 - 26s/epoch - 20ms/step
```

```
In [26]: def plot_graphs(history, string):
             plt.plot(history.history[string])
             plt.plot(history.history['val_'+string])
             plt.xlabel("Epochs")
             plt.ylabel(string)
             plt.legend([string, 'val_'+string])
             plt.show()

         plot_graphs(history, "accuracy")
         plot_graphs(history, "loss")
```





```
In [27]:
         seed_text = "wonderful little production br br filming technique unassuming old time bbc fashion give comforting sometimes discom
         token_list = tokenizer.texts_to_sequences([seed_text])[0]
         token_list = pad_sequences([token_list], maxlen=max_length-1, padding=padding_type, truncating=trunc_type)
         predicted = (model.predict(token_list, verbose=0) > 0.5).astype("int32")

         if predicted[0][0] == 0:
             print("Negative")
         else:
             print("Positive")
```

```
Positive
```

```
In [28]: preprocess_text[1]
```

```
Out[28]: 'wonderful little production br br filming technique unassuming old time bbc fashion give comforting sometimes discomforting se
         nse realism entire piece br br actor extremely well chosen michael sheen got polari voice pat truly see seamless editing guided
         reference williams diary entry well worth watching terrificly written performed piece masterful production one great master com
         edy life br br realism really come home little thing fantasy guard rather use traditional would ream technique remains solid di
         sappears play knowledge sens particularly scene concerning orton halliwell set particularly flat halliwell mural decorating eve
         ry surface terribly well done'
```

```
In [ ]:
```