In [1]:
```
import pandas as pd
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Conv1D
from tensorflow.keras.layers import MaxPooling1D
from tensorflow.keras.layers import Embedding
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing import sequence
from sklearn.preprocessing import LabelEncoder


seed = 42
np.random.seed(seed)
```

In [2]:
```
dataset = pd.read_csv(r'https://github.com/dipanjanS/nlp_workshop_dhs18/raw/master/Unit%2011%20-%20Sentiment%20Analysis%20-%20Unsupervised%20Learning/movie_reviews.csv.bz2', compression='bz2')
dataset.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
```

In [3]:
```
# build train and test datasets
reviews = dataset['review'].values
sentiments = dataset['sentiment'].values

train_reviews = reviews[:35000]
train_sentiments = sentiments[:35000]

test_reviews = reviews[35000:]
test_sentiments = sentiments[35000:]
```

In [7]:
```
# import contractions
from bs4 import BeautifulSoup
import numpy as np
import re
# import tqdm
import unicodedata


def strip_html_tags(text):
  soup = BeautifulSoup(text, "html.parser")
  [s.extract() for s in soup(['iframe', 'script'])]
  stripped_text = soup.get_text()
  stripped_text = re.sub(r'[\r|\n|\r\n]+', '\n', stripped_text)
  return stripped_text

def remove_accented_chars(text):
  text = unicodedata.normalize('NFKD', text).encode('ascii', 'ignore').decode('utf-8', 'ignore')
  return text

def pre_process_corpus(docs):
  norm_docs = []
  for doc in docs:
    doc = strip_html_tags(doc)
    doc = doc.translate(doc.maketrans("\n\t\r", "   "))
    doc = doc.lower()
    doc = remove_accented_chars(doc)
    # doc = contractions.fix(doc)
    # Lower case and remove special characters\whitespaces
    doc = re.sub(r'[^a-zA-Z0-9\s]', '', doc, re.I|re.A)
    doc = re.sub(' +', ' ', doc)
    doc = doc.strip()
    norm_docs.append(doc)

  return norm_docs
```

In [8]:
```
norm_train_reviews = pre_process_corpus(train_reviews)
norm_test_reviews = pre_process_corpus(test_reviews)
```

```
C:\Users\karu0\AppData\Local\Temp\ipykernel_10344\1215441744.py:10: MarkupResemblesLocatorWarning: The input looks more like a filename than markup. You may want to open this file and pass th
e filehandle into Beautiful Soup.
  soup = BeautifulSoup(text, "html.parser")
```

In [9]:
```
t = Tokenizer(oov_token='<UNK>')
# fit the tokenizer on the documents
t.fit_on_texts(norm_train_reviews)
t.word_index['<PAD>'] = 0
```

In [10]:
```
train_sequences = t.texts_to_sequences(norm_train_reviews)
```

In [11]:
```
test_sequences = t.texts_to_sequences(norm_test_reviews)
```

In [12]:
```
print("Vocabulary size={}".format(len(t.word_index)))
print("Number of Documents={}".format(t.document_count))
```

```
Vocabulary size=176791
Number of Documents=35000
```

In [14]:
```
MAX_SEQUENCE_LENGTH = 1000
```

In [15]:
```
X_train = sequence.pad_sequences(train_sequences, maxlen=MAX_SEQUENCE_LENGTH)
X_test = sequence.pad_sequences(test_sequences, maxlen=MAX_SEQUENCE_LENGTH)
X_train.shape, X_test.shape
```

Out[15]:
```
((35000, 1000), (15000, 1000))
```

In [16]:
```
le = LabelEncoder()
num_classes=2
```

In [17]:
```
le = LabelEncoder()
num_classes=2
```

In [24]:
```
y_train = le.fit_transform(train_sentiments)
y_test = le.transform(test_sentiments)
```

In [18]:
```
VOCAB_SIZE = len(t.word_index)
```

In [19]:
```
EMBED_SIZE = 300
EPOCHS=2
BATCH_SIZE=128
```

In [20]:
```
model = Sequential()
model.add(Embedding(VOCAB_SIZE, EMBED_SIZE, input_length=MAX_SEQUENCE_LENGTH))
model.add(Conv1D(filters=128, kernel_size=4, padding='same', activation='relu'))
model.add(MaxPooling1D(pool_size=2))
model.add(Conv1D(filters=64, kernel_size=4, padding='same', activation='relu'))
model.add(MaxPooling1D(pool_size=2))
model.add(Conv1D(filters=32, kernel_size=4, padding='same', activation='relu'))
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())
model.add(Dense(256, 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 (Embedding)       (None, 1000, 300)         53037300

 conv1d (Conv1D)             (None, 1000, 128)         153728

 max_pooling1d (MaxPooling1D  (None, 500, 128)         0
 )

 conv1d_1 (Conv1D)           (None, 500, 64)           32832

 max_pooling1d_1 (MaxPooling  (None, 250, 64)          0
 1D)

 conv1d_2 (Conv1D)           (None, 250, 32)           8224

 max_pooling1d_2 (MaxPooling  (None, 125, 32)          0
 1D)

 flatten (Flatten)           (None, 4000)              0

 dense (Dense)               (None, 256)               1024256

 dense_1 (Dense)             (None, 1)                 257

=================================================================
Total params: 54,256,597
Trainable params: 54,256,597
Non-trainable params: 0
_________________________________________________________________
```

In [25]:
```
import tensorflow as tf

with tf.device('/GPU:0'):
    model.fit(X_train, y_train,
              validation_split=0.1,
              epochs=EPOCHS,
              batch_size=BATCH_SIZE,
              verbose=1)
```

```
Epoch 1/2
247/247 [==============================] - 76s 163ms/step - loss: 0.4042 - accuracy: 0.7827 - val_loss: 0.2390 - val_accuracy: 0.9031
Epoch 2/2
247/247 [==============================] - 40s 160ms/step - loss: 0.1239 - accuracy: 0.9544 - val_loss: 0.2596 - val_accuracy: 0.9006
```

In [26]:
```
scores = model.evaluate(X_test, y_test, verbose=1)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

```
469/469 [==============================] - 8s 17ms/step - loss: 0.2482 - accuracy: 0.9047
Accuracy: 90.47%
```

In [28]:
```
predictions = model.predict(X_test).ravel()
predictions[:10]
```

```
469/469 [==============================] - 7s 14ms/step
```

Out[28]:
```
array([0.00755076, 0.9992706 , 0.00114281, 0.99799407, 0.9966151 ,
       0.00106926, 0.99313617, 0.76698875, 0.75450575, 0.97222936],
      dtype=float32)
```

In [30]:
```
predictions = ['positive' if item == 1 else 'negative' for item in predictions]
predictions[:10]
```

Out[30]:
```
['negative',
 'negative',
 'negative',
 'negative',
 'negative',
 'negative',
 'negative',
 'negative',
 'negative',
 'negative']
```

In [31]:
```
from sklearn.metrics import confusion_matrix, classification_report

labels = ['negative', 'positive']
print(classification_report(test_sentiments, predictions))
pd.DataFrame(confusion_matrix(test_sentiments, predictions), index=labels, columns=labels)
```

```
              precision    recall  f1-score   support

    negative       0.50      1.00      0.67      7490
    positive       1.00      0.00      0.00      7510

    accuracy                           0.50     15000
   macro avg       0.75      0.50      0.33     15000
weighted avg       0.75      0.50      0.33     15000
```

Out[31]:

	negative	positive
negative	7490	0
positive	7508	2

In [ ]:
```

```