EXPERIMENT NO 7

TEXT SUMMARIZATION

IMPORTING THE LIBRARIES

In [1]:
```
# keras module for building LSTM
import os
import string
import numpy as np
import pandas as pd
from keras.preprocessing.sequence import pad_sequences
from keras.layers import Embedding, LSTM, Dense, Dropout
from keras.preprocessing.text import Tokenizer
from keras.callbacks import EarlyStopping
from keras.models import Sequential
import keras.utils as ku

# set seeds for reproducability
from tensorflow import set_random_seed
from numpy.random import seed

set_random_seed(2)
seed(1)
```

```
Using TensorFlow backend.
```

READING THE DATASET

In [2]:
```
curr_dir = '../input/'
all_headlines = []
for filename in os.listdir(curr_dir):
    if 'Articles' in filename:
        article_df = pd.read_csv(curr_dir + filename)
        all_headlines.extend(list(article_df.headline.values))
        break

all_headlines = [h for h in all_headlines if h != "Unknown"]
len(all_headlines)
```

Out[2]:
```
829
```

DATA PREPRATION

In [3]:
```
def clean_text(txt):
    txt = "".join(v for v in txt if v not in string.punctuation).lower()
    txt = txt.encode("utf8").decode("ascii", 'ignore')
    return txt


corpus = [clean_text(x) for x in all_headlines]
corpus[:10]
```

Out[3]:
```
['nfl vs politics has been battle all season long',
 'voice vice veracity',
 'a standups downward slide',
 'new york today a groundhog has her day',
 'a swimmers communion with the ocean',
 'trail activity',
 'super bowl',
 'trumps mexican shakedown',
 'pences presidential pet',
 'fruit of a poison tree']
```

Generating Sequence of N-gram Tokens

In [4]:
```
tokenizer = Tokenizer()


def get_sequence_of_tokens(corpus):
    # tokenization
    tokenizer.fit_on_texts(corpus)
    total_words = len(tokenizer.word_index) + 1

    # convert data to sequence of tokens
    input_sequences = []
    for line in corpus:
        token_list = tokenizer.texts_to_sequences([line])[0]
        for i in range(1, len(token_list)):
            n_gram_sequence = token_list[:i + 1]
            input_sequences.append(n_gram_sequence)
    return input_sequences, total_words

inp_sequences, total_words = get_sequence_of_tokens(corpus)
inp_sequences[:10]
```

Out[4]:
```
[[660, 117],
 [660, 117, 72],
 [660, 117, 72, 73],
 [660, 117, 72, 73, 661],
 [660, 117, 72, 73, 661, 662],
 [660, 117, 72, 73, 661, 662, 63],
 [660, 117, 72, 73, 661, 662, 63, 29],
 [660, 117, 72, 73, 661, 662, 63, 29, 210],
 [211, 663],
 [211, 663, 664]]
```

In the above output [30, 507], [30, 507, 11], [30, 507, 11, 1] and so on represents the ngram phrases generated from the input data. where every integer corresponds to the index of a particular word in the complete vocabulary of words present in the text.

Padding the Sequences and obtain Variables : Predictors and Target

In [5]:
```
def generate_padded_sequences(input_sequences):
    max_sequence_len = max([len(x) for x in input_sequences])
    input_sequences = np.array(
        pad_sequences(input_sequences, maxlen=max_sequence_len, padding='pre'))

    predictors, label = input_sequences[:, :-1], input_sequences[:, -1]
    label = ku.to_categorical(label, num_classes=total_words)
    return predictors, label, max_sequence_len


predictors, label, max_sequence_len = generate_padded_sequences(inp_sequences)
```

USING THE LSTM FOR TEXT GENERATION

In [6]:
```
def create_model(max_sequence_len, total_words):
    input_len = max_sequence_len - 1
    model = Sequential()

    # Add Input Embedding Layer
    model.add(Embedding(total_words, 10, input_length=input_len))

    # Add Hidden Layer 1 - LSTM Layer
    model.add(LSTM(100))
    model.add(Dropout(0.1))

    # Add Output Layer
    model.add(Dense(total_words, activation='softmax'))

    model.compile(loss='categorical_crossentropy', optimizer='adam')

    return model


model = create_model(max_sequence_len, total_words)
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 16, 10)            22880
_________________________________________________________________
lstm_1 (LSTM)                (None, 100)               44400
_________________________________________________________________
dropout_1 (Dropout)          (None, 100)               0
_________________________________________________________________
dense_1 (Dense)              (None, 2288)              231088
=================================================================
Total params: 298,368
Trainable params: 298,368
Non-trainable params: 0
_________________________________________________________________
```

TRAINING THE MODEL

In [7]:
```
model.fit(predictors, label, epochs=100, verbose=5)
```

```
Epoch 1/100
Epoch 2/100
Epoch 3/100
Epoch 4/100
Epoch 5/100
Epoch 6/100
Epoch 7/100
Epoch 8/100
Epoch 9/100
Epoch 10/100
Epoch 11/100
Epoch 12/100
Epoch 13/100
Epoch 14/100
Epoch 15/100
Epoch 16/100
Epoch 17/100
Epoch 18/100
Epoch 19/100
Epoch 20/100
Epoch 21/100
Epoch 22/100
Epoch 23/100
Epoch 24/100
Epoch 25/100
Epoch 26/100
Epoch 27/100
Epoch 28/100
Epoch 29/100
Epoch 30/100
Epoch 31/100
Epoch 32/100
Epoch 33/100
Epoch 34/100
Epoch 35/100
Epoch 36/100
Epoch 37/100
Epoch 38/100
Epoch 39/100
Epoch 40/100
Epoch 41/100
Epoch 42/100
Epoch 43/100
Epoch 44/100
Epoch 45/100
Epoch 46/100
Epoch 47/100
Epoch 48/100
Epoch 49/100
Epoch 50/100
Epoch 51/100
Epoch 52/100
Epoch 53/100
Epoch 54/100
Epoch 55/100
Epoch 56/100
Epoch 57/100
Epoch 58/100
Epoch 59/100
Epoch 60/100
Epoch 61/100
Epoch 62/100
Epoch 63/100
Epoch 64/100
Epoch 65/100
Epoch 66/100
Epoch 67/100
Epoch 68/100
Epoch 69/100
Epoch 70/100
Epoch 71/100
Epoch 72/100
Epoch 73/100
Epoch 74/100
Epoch 75/100
Epoch 76/100
Epoch 77/100
Epoch 78/100
Epoch 79/100
Epoch 80/100
Epoch 81/100
Epoch 82/100
Epoch 83/100
Epoch 84/100
Epoch 85/100
Epoch 86/100
Epoch 87/100
Epoch 88/100
Epoch 89/100
Epoch 90/100
Epoch 91/100
Epoch 92/100
Epoch 93/100
Epoch 94/100
Epoch 95/100
Epoch 96/100
Epoch 97/100
Epoch 98/100
Epoch 99/100
Epoch 100/100
```

Out[7]:
```
<keras.callbacks.History at 0x7fdeacdb2d68>
```

GENERATING THE TEXT

In [8]:
```
def generate_text(seed_text, next_words, model, max_sequence_len):
    for _ in range(next_words):
        token_list = tokenizer.texts_to_sequences([seed_text])[0]
        token_list = pad_sequences([token_list],
                                   maxlen=max_sequence_len - 1,
                                   padding='pre')
        predicted = model.predict_classes(token_list, verbose=0)

        output_word = ""
        for word, index in tokenizer.word_index.items():
            if index == predicted:
                output_word = word
                break
        seed_text += " " + output_word
    return seed_text.title()
```

VERIFYING THE RESULTS

In [9]:
```
print(generate_text("united states", 5, model, max_sequence_len))
print(generate_text("president trump", 4, model, max_sequence_len))
print(generate_text("donald trump", 4, model, max_sequence_len))
print(generate_text("india and china", 4, model, max_sequence_len))
print(generate_text("new york", 4, model, max_sequence_len))
print(generate_text("science and technology", 5, model, max_sequence_len))
```

```
United States Being Equal Media Obsesses On
President Trump Vs Staff Rethink Tactics
Donald Trump Master For Sports Champs
India And China By Spy Rise With
New York Today A Makeshift Law
Science And Technology And Media Obsesses On Trump
```