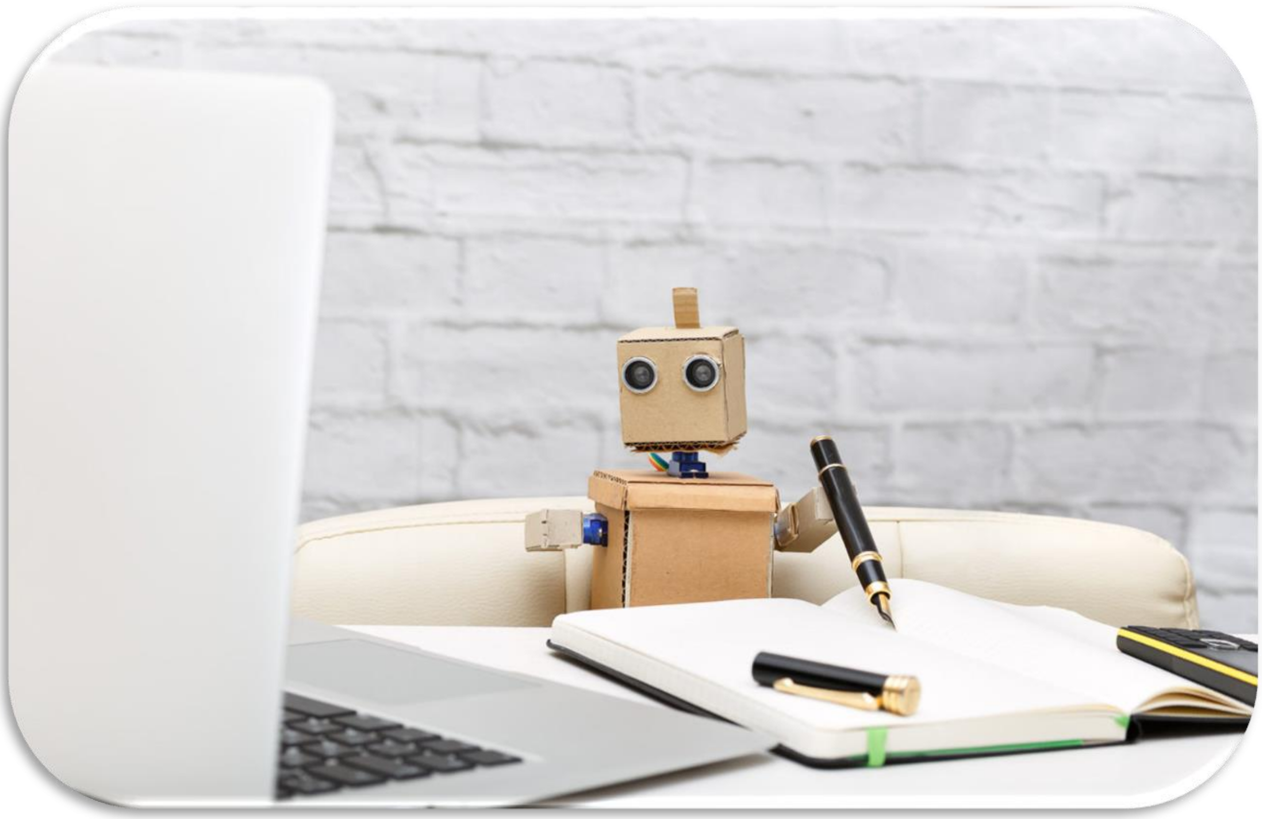
# Generating Poems Using Natural Language Processing (NLP) with Python

The Ultimate Guide for Text Generation and Summarization by Predicting the Next Set of Words using LSTM.



Keras TensorFlow NLTK

**Blog By: Chaithanya Vamshi Sai**

**Student Id: 21152797**

## Introduction

Natural Language Processing (NLP) is an emerging field and a subset of machine learning which aims to train computers to understand human languages. The most common application of NLP is Text Generation and Summarization.

Text generation comes in handy in the world of creative arts through songwriting, poems, short stories, and novels.

In this blog, I will demonstrate and show how we can harness the power of Natural Language Processing (NLP) with Deep Learning by building and training a neural network that generates poems and predicts the next set of words from the seed text using LSTM.

## Steps to Implement Text Generation with LSTM Neural Networks in Python with Keras and NLTK

1. Problem Statement
2. Importing Libraries
3. Loading the Data
4. Data Pre-processing
5. Tokenize the Data
6. Model Building
7. Model Evaluation Results: Accuracy and Loss
8. Predicting the next 25 words in a Poem

## 1. Problem Statement

The objective of this task is that we are given a Gutenberg Dataset and use the book 'blake-poems.txt' file corpus as the source text.

Using this data, we need to train a neural network model that generates poems by predicting the next set of words from the seed text in a Poem using LSTM, Keras and NLTK.

### Why LSTM?

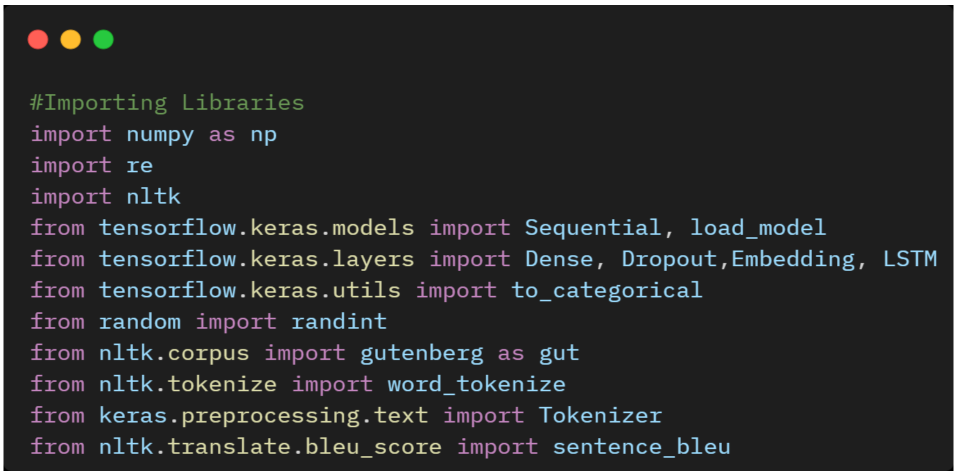
LSTM stands for long short-term memory, and it is a building unit for layers of a recurrent neural network. The LSTM unit is made of a cell, an input, an output and a forget gate.

These are responsible for memorizing over a certain time, where the gates regulate how much of the data is kept and can have a memory about the previous data. Hence it can generate a new pattern using previous data.

LSTMs are mainly used for text generation, and it's preferred over RNNs because of RNNs vanishing and exploding gradients problems.

## 2. Importing Libraries

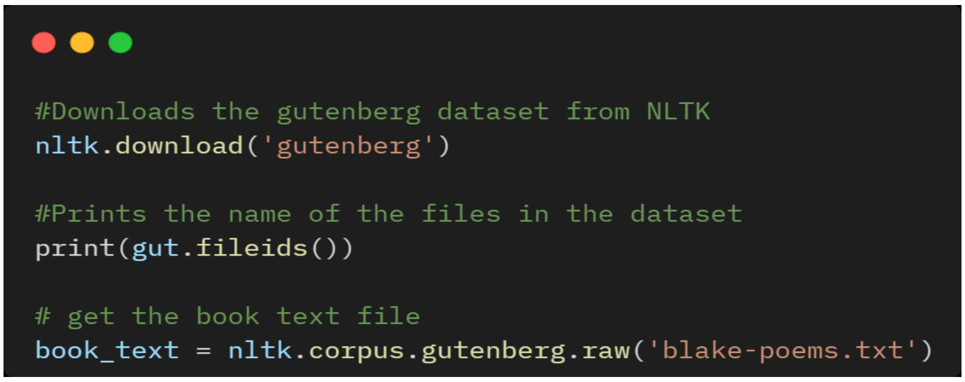
- **NLTK:** It is a toolkit built for working with NLP in Python which contains text processing libraries for tokenization, parsing, classification etc.
- **Keras:** It is a powerful and easy-to-use free open-source Python library and high-level API for developing and evaluating deep learning models.



```python
#Importing Libraries
import numpy as np
import re
import nltk
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.layers import Dense, Dropout,Embedding, LSTM
from tensorflow.keras.utils import to_categorical
from random import randint
from nltk.corpus import gutenberg as gut
from nltk.tokenize import word_tokenize
from keras.preprocessing.text import Tokenizer
from nltk.translate.bleu_score import sentence_bleu
```

## 3. Loading the Dataset

- Download the Gutenberg dataset from NLTK
- Print the name of the files in the dataset
- Get the book 'blake-poems.txt' text file from NLTK



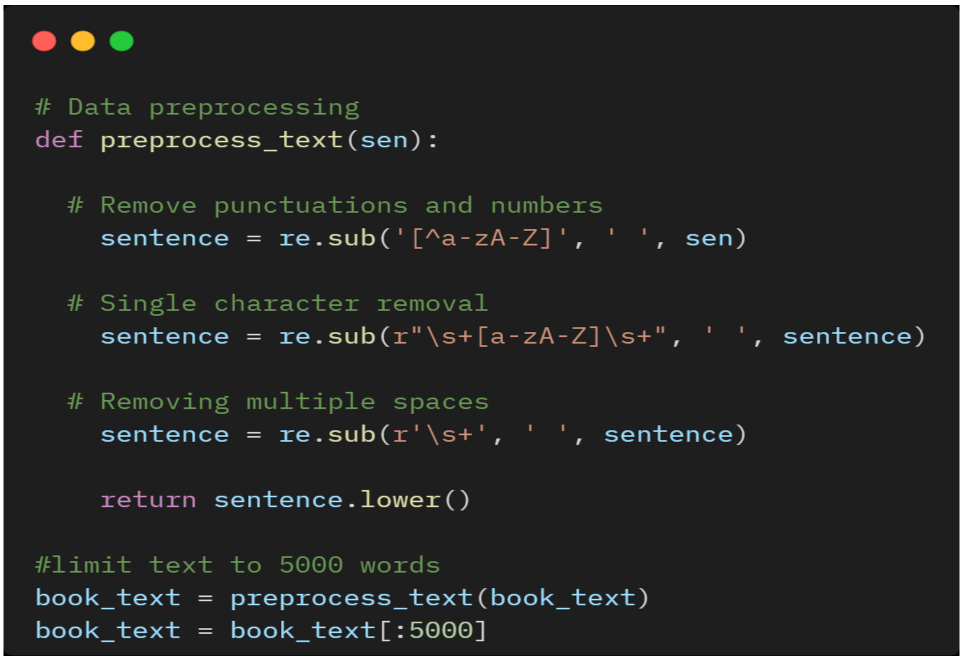
```python
#Downloads the gutenberg dataset from NLTK
nltk.download('gutenberg')

#Prints the name of the files in the dataset
print(gut.fileids())

# get the book text file
book_text = nltk.corpus.gutenberg.raw('blake-poems.txt')
```

## 4. Data Pre-processing

The pre-processing of the text data is an essential step as it makes it easier to extract information from the text and apply deep learning algorithms to it.

The objective of this step is to Inspect data and clean noise that is irrelevant such as punctuation, special characters, numbers, and terms that don't carry much weightage in context to the text.

We will perform some basic text pre-processing on the text like Remove punctuations and numbers, Single character removal, removing multiple spaces, limiting text to 5000 etc. so that text is good enough to build the required model on.



```python
# Data preprocessing
def preprocess_text(sen):

  # Remove punctuations and numbers
    sentence = re.sub('[^a-zA-Z]', ' ', sen)

  # Single character removal
    sentence = re.sub(r"\s+[a-zA-Z]\s+", ' ', sentence)

  # Removing multiple spaces
    sentence = re.sub(r'\s+', ' ', sentence)

    return sentence.lower()

#limit text to 5000 words
book_text = preprocess_text(book_text)
book_text = book_text[:5000]
```

## 5. Tokenization

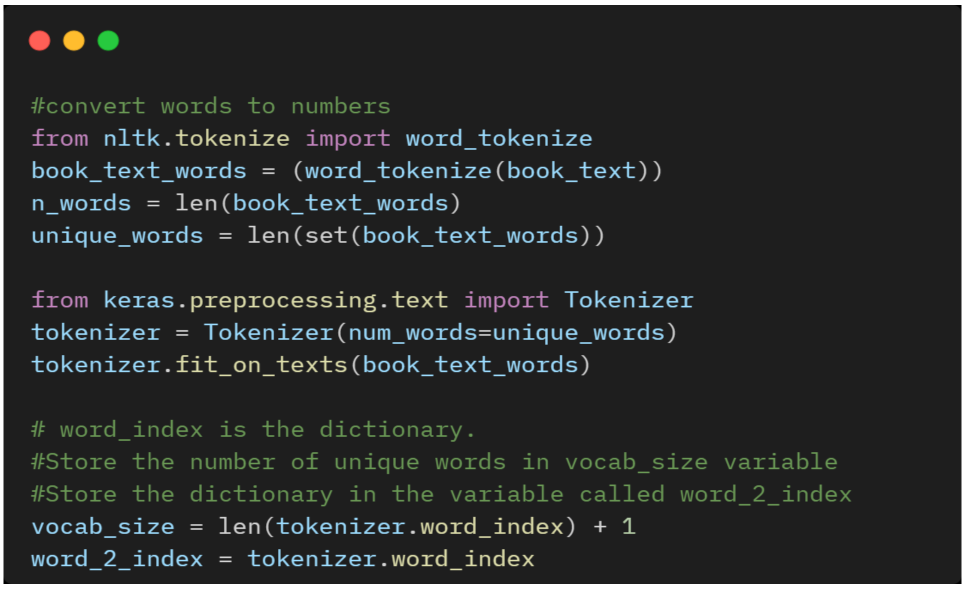
As we all know machine learning and neural networks algorithms cannot work on text data directly, they simply don't recognize it.

Hence, we need to convert our words in the sentences to numerical values such that our model can figure out what exactly is going on.

For this purpose, especially, in Natural Language Processing (NLP), "Tokenization" plays are vital role.

Tokens are individual terms or words, and tokenization is the process of splitting a string of text into tokens.

Tokenization serves as the base of almost every possible model based on NLP. It just assigns each unique word a different number which we can check out by printing tokenizer.word_index.

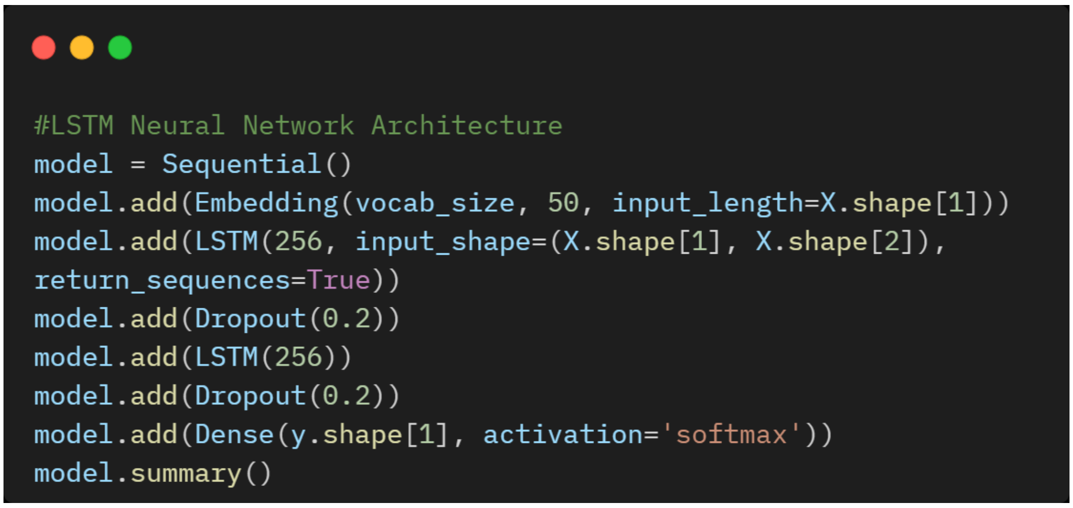


```python
#convert words to numbers
from nltk.tokenize import word_tokenize
book_text_words = (word_tokenize(book_text))
n_words = len(book_text_words)
unique_words = len(set(book_text_words))

from keras.preprocessing.text import Tokenizer
tokenizer = Tokenizer(num_words=unique_words)
tokenizer.fit_on_texts(book_text_words)

# word_index is the dictionary.
#Store the number of unique words in vocab_size variable
#Store the dictionary in the variable called word_2_index
vocab_size = len(tokenizer.word_index) + 1
word_2_index = tokenizer.word_index
```

## 6. Model Building

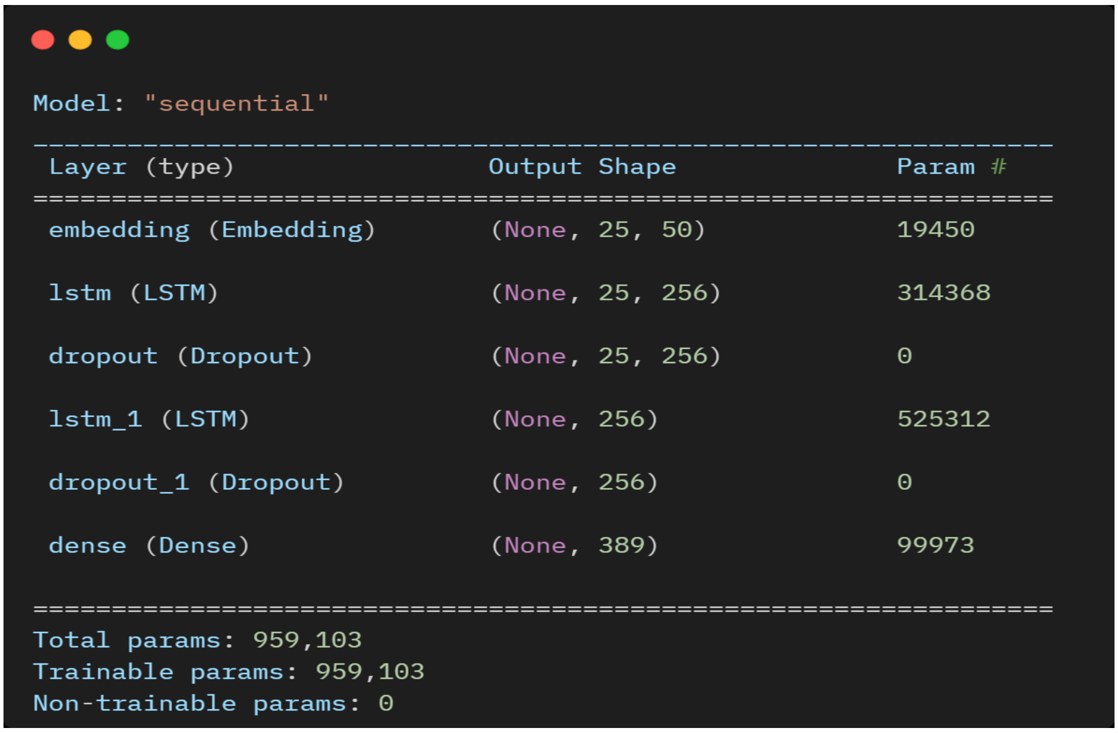
### a) Defining the LSTM Neural Network Model

- we will use an **Embedding Layer** to learn the representation of words, and a Long Short-Term Memory (LSTM) recurrent neural network to learn to predict words based on their context and we use 50 as the size of the embedding vector space.
- We will **use two LSTM hidden layers** with **256** memory units each. More memory cells and a deeper network may achieve better results.
- **Dropout** is a regularization method, where a proportion of nodes in the layer are randomly ignored by setting their weights to zero for each training sample. This technique also improves generalization and reduces overfitting. Hence, to avoid that we will be using a Dropout layer with a probability of **20%.**
- The **output layer** predicts the next word as a single vector the size of the vocabulary with a probability for each word in the vocabulary.
- **SoftMax activation function** is used to ensure the outputs have the characteristics of normalized probabilities.



```python
#LSTM Neural Network Architecture
model = Sequential()
model.add(Embedding(vocab_size, 50, input_length=X.shape[1]))
model.add(LSTM(256, input_shape=(X.shape[1], X.shape[2]),
return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(256))
model.add(Dropout(0.2))
model.add(Dense(y.shape[1], activation='softmax'))
model.summary()
```

## b) LSTM Neural Network Architecture Summary

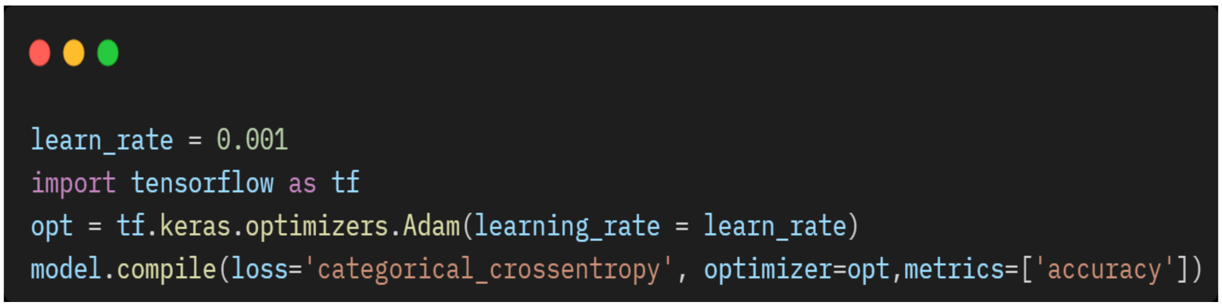


```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 25, 50)            19450

 lstm (LSTM)                 (None, 25, 256)           314368

 dropout (Dropout)           (None, 25, 256)           0

 lstm_1 (LSTM)               (None, 256)               525312

 dropout_1 (Dropout)         (None, 256)               0

 dense (Dense)               (None, 389)               99973

=================================================================
Total params: 959,103
Trainable params: 959,103
Non-trainable params: 0
```

## c) Compiling the LSTM Model

Once our layers are added to the model, we need to set up a score function, a loss function, and an optimization algorithm.

- **Loss function:** To measure how poorly our model performs on images with known labels and we will use “categorical_crossentropy".
- **Optimizer:** This function will iteratively improve parameters to minimize the loss. We will go with the "Adam" optimizer.
- **Evaluation Metric:** "Accuracy" is used to evaluate the LSTM model's performance.
- **Learning rate**: The amount of change to the model during each step of this search process, or the step size. The learning rate hyperparameter controls the rate or speed at which the model learns. We will use 0.001 as the learning rate for the LSTM model.



```
learn_rate = 0.001
import tensorflow as tf
opt = tf.keras.optimizers.Adam(learning_rate = learn_rate)
model.compile(loss='categorical_crossentropy', optimizer=opt,metrics=['accuracy'])
```

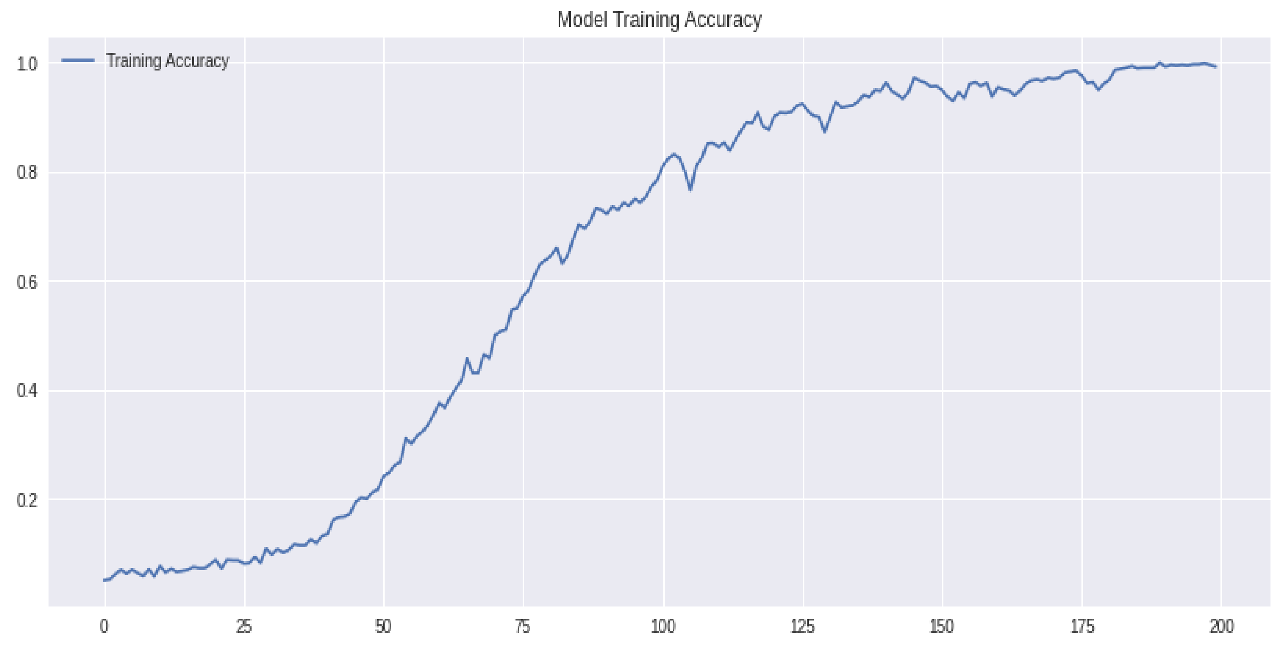
## d) Model Training

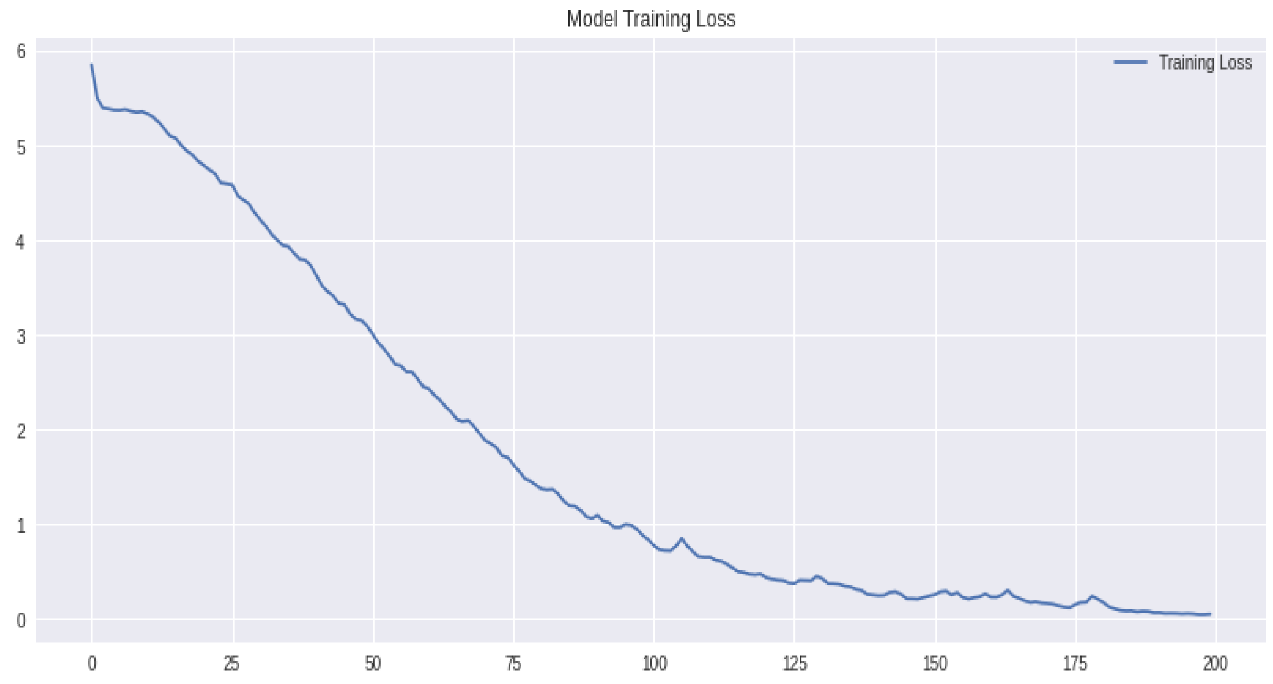
To start training, we will use the “model.fit” method to train the data and parameters with epochs = 200 and batch_size=64.

We have trained the model using 200 epochs and got a loss of 0.0287 and an accuracy of 100%. Below is the visual summary of the LSTM model Accuracy and Loss.

Python Implementation Source Code of the Project is available on **GitHub**

**Results:**
**Model Accuracy:** 100.00%
**Model Loss:** 0.02874165028333664





## 7. Predicting the Next 25 Words of a Poem

Finally, we will input a seed text which will be the origin from which the poem will be generated and set the next words to 25.

**Poem 1:**

**Seed word sequence** gives his light and gives his heat away and flowers and trees and beasts and men receive Comfort in morning joy in the noonday and

**Predicted words** we are put on earth little space that we may learn to bear the beams of love and these black bodies and this sunburnt face

**BLEU Score for predicted words:** 1.0

**Poem 2:**

**Seed word sequence** gives his light and gives his heat away and flowers and trees and beasts and men receive Comfort in morning joy in the noonday and

**Predicted words** is but cloud and like shady grove for when our souls have learn the heat to bear The cloud will vanish, we shall hear his

**BLEU Score for predicted words:** 1.0

**Poem 3:**

**Seed word sequence** gives his light and gives his heat away and flowers and trees and beasts and men receive Comfort in morning joy in the noonday and

**Predicted words** voice saying come out from the grove my love and care and round my golden tent like lambs rejoice thus did my mother say and

**BLEU Score for predicted words:** 1.0

## 8. Conclusion

In this blog, we discussed how to implement a neural network that generates poems and predicts the next set of words from the seed text using LSTM, Keras and NLTK. Though some parts of the poem sound meaningless, the model can be tweaked to gain higher accuracy, lower loss, and predict more meaningful poems.

## 9. References

- https://machinelearningmastery.com/text-generation-lstm-recurrent-neural-networks-python-keras/
- https://machinelearningmastery.com/calculate-bleu-score-for-text-python/