Short Story Generator using Bidirectional LSTM

Let's read the dataset

In [1]:
```
import pandas as pd

train_path = "dataset/train.csv"

train = pd.read_csv(train_path)
train['text'] = train['text'].fillna('').astype(str)
```

In [2]:
```
print("Number of rows and columns in train dataset:", train.shape)
print("\nColumn names:")
print(train.columns)
print("\nData types of columns:")
print(train.dtypes)
print("\nBasic statistics of numerical columns:")
print(train.describe())
```

```
Number of rows and columns in train dataset: (2119719, 1)

Column names:
Index(['text'], dtype='object')

Data types of columns:
text    object
dtype: object

Basic statistics of numerical columns:
           text
count   2119719
unique  1799249
top
freq        230
```

In [3]:
```
train.head()
```

Out[3]:

	text
0	One day, a little girl named Lily found a need...
1	Once upon a time, there was a little car named...
2	One day, a little fish named Fin was swimming ...
3	Once upon a time, in a land full of trees, the...
4	Once upon a time, there was a little girl name...

Trim Dataset

The original dataset is very large. We trim it to be able to run it on local computer. Feel free to comment this line or change your desired dataset size based on your configuration

In [28]:
```
dataset_size = 100
train = train[:dataset_size]
```

In [29]:
```
train.shape
```

Out[29]:
```
(100, 1)
```

Make a tokenizer to build set of tokens from the train dataset

In [30]:
```
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences


# Tokenize train data
tokenizer = Tokenizer()
tokenizer.fit_on_texts(train['text'])
total_words = len(tokenizer.word_index) + 1
```

In [44]:
```
total_words
```

Out[44]:
```
1491
```

Now we need to convert our text based inputs to numerical inputs. texts_to_squences converts text input to the numerical vector.

In [31]:
```
# Prepare input sequences
input_sequences = []
for line in train['text']:
    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)
```

In [50]:
```
print("Example input:")
print(line)
print("<------------------------------------->\n\n")
print("Converted vector:")
print(tokenizer.texts_to_sequences([line])[0])
```

```
Example input:
John and Sarah were playing together in their backyard when they found a piece of metal. It was shin
y and reflective and they couldn't wait to show their parents.

John asked Sarah, "What should we do with the metal?"

Sarah thought for a moment, then said, "Let's take it to Mommy and Daddy!" With that, they ran off e
xcitedly, ready to surprise their parents.

They raced into the house, and shouted, "Mommy, Daddy! Look what we found!"

Their parents were very surprised and asked, "Where did you find this piece of metal?"

John and Sarah were so proud of their discovery, and couldn't wait to tell the story. They recounted
that they found the metal outside in the backyard and it was so shiny and reflective.

Their parents smiled, and said, "Well, why don't you two take it around the neighbourhood and see if
you can return it to its rightful owner. If nobody takes it, you two can keep it!".

John and Sarah were so cheerful and excited about the prospect of helping find the true owner of the
metal, that they grabbed it and set off, ready to call on their neighbours.
<------------------------------------->


Converted vector:
[596, 2, 260, 40, 81, 48, 10, 91, 724, 45, 9, 65, 3, 461, 19, 357, 6, 5, 388, 2, 948, 2, 9, 156, 25
1, 4, 252, 91, 406, 596, 67, 260, 98, 287, 115, 118, 16, 1, 357, 260, 96, 31, 3, 329, 52, 14, 131, 1
42, 6, 4, 108, 2, 219, 16, 17, 9, 82, 166, 922, 362, 4, 355, 91, 406, 9, 747, 160, 1, 141, 2, 475, 1
08, 219, 195, 98, 115, 65, 91, 406, 40, 30, 889, 2, 67, 447, 97, 21, 174, 87, 461, 19, 357, 596, 2,
260, 40, 13, 210, 19, 91, 1481, 2, 156, 251, 4, 846, 1, 339, 9, 1482, 17, 9, 65, 1, 357, 140, 10, 1,
724, 2, 6, 5, 13, 388, 2, 948, 91, 406, 44, 2, 14, 690, 215, 150, 21, 186, 142, 6, 68, 1, 1483, 2, 1
53, 175, 21, 55, 1484, 6, 4, 258, 1485, 949, 175, 883, 1486, 6, 21, 186, 55, 384, 6, 596, 2, 260, 4
0, 13, 1487, 2, 106, 126, 1, 1488, 19, 421, 174, 1, 668, 949, 19, 1, 357, 17, 9, 482, 6, 2, 1489, 16
6, 362, 4, 735, 34, 91, 1490]
```

In [32]:
```
len(input_sequences)
```
Out[32]:
```
14700
```

The input text files includes stories with different lengths. We convert them to the same size input sequences using pad_sequences function. We use the length of longest story for finding the padding size.

In [33]:
```
# Pad sequences
max_sequence_len = max([len(seq) for seq in input_sequences])
input_sequences = pad_sequences(input_sequences, maxlen=max_sequence_len, padding='pre')
```

There are infinite ways to define predictor labels. For example one can get first i words of a particular text as an input and the i+1 word as the label. We use the entire text except the last one as an input and the last word as a label.

In [34]:
```
import numpy as np
# Create predictors and label
predictors, label = input_sequences[:, :-1],input_sequences[:, -1]
label = np.array(label)
```

In [35]:
```
predictors.shape
```
Out[35]:
```
(14700, 211)
```

In [36]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Bidirectional, Dense
```

In [37]:
```
# Build the model
model = Sequential()
model.add(Embedding(total_words, 100, input_length=max_sequence_len-1))
model.add(Bidirectional(LSTM(150)))
model.add(Dense(total_words, activation='softmax'))
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

In [38]:
```
model.summary()
```
```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding_2 (Embedding)     (None, 211, 100)          149100    
                                                                 
 bidirectional_2 (Bidirecti  (None, 300)               301200    
 onal)                                                           
                                                                 
 dense_2 (Dense)             (None, 1491)              448791    
                                                                 
=================================================================
Total params: 899091 (3.43 MB)
Trainable params: 899091 (3.43 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [39]:
```python
# Train the model
model.fit(predictors, label, epochs=100, verbose=1)
```

```
Epoch 1/100
460/460 [==============================] - 636s 1s/step - loss: 5.9186 - accuracy: 0.0689
Epoch 2/100
460/460 [==============================] - 626s 1s/step - loss: 5.2878 - accuracy: 0.1137
Epoch 3/100
460/460 [==============================] - 631s 1s/step - loss: 4.7095 - accuracy: 0.1783
Epoch 4/100
460/460 [==============================] - 609s 1s/step - loss: 4.2156 - accuracy: 0.2143
Epoch 5/100
460/460 [==============================] - 610s 1s/step - loss: 3.8192 - accuracy: 0.2495
Epoch 6/100
460/460 [==============================] - 612s 1s/step - loss: 3.4718 - accuracy: 0.2806
Epoch 7/100
460/460 [==============================] - 615s 1s/step - loss: 3.1624 - accuracy: 0.3134
Epoch 8/100
460/460 [==============================] - 613s 1s/step - loss: 2.8747 - accuracy: 0.3565
Epoch 9/100
460/460 [==============================] - 613s 1s/step - loss: 2.6117 - accuracy: 0.4024
Epoch 10/100
```

In [42]:
```python
def generate_text(seed_text, next_words, model, max_sequence_len, tokenizer):
    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_probs = model.predict(token_list, verbose=0)
        predicted = np.argmax(predicted_probs)
        output_word = ""
        for word, index in tokenizer.word_index.items():
            if index == predicted:
                output_word = word
                break
        seed_text += " " + output_word
    return seed_text
```

In [43]:
```python
# Keywords for text generation
keywords = ["girl", "dog"]
seed_text = ' '.join(keywords)  # Seed text with keywords

# Generate text based on keywords
generated_text = generate_text(seed_text, 100, model, max_sequence_len, tokenizer)
print(generated_text)
```

```
girl dog a girl named mia went for a walk she saw a big scary house it had a tall door and small win
dows mia was brave so she went inside the house in the house mia saw a birdcage inside the birdcage
there was a little bird the bird was sad it wanted to fly and be free mia wanted to help the bird mi
a opened the birdcage door the bird flew out and was happy it was not scary anymore mia and the bird
were friends they played and had fun all day and they decided to play hide and
```

See impact of train data

Let's find the input sequences which include "mia.

In [65]:
```python
similar_texts = []
target_words = ["Mia", "mia"]
for line in train['text']:
    for w in target_words:
        if w in line:
            similar_texts.append(line)
            break
```

```
In [66]: len(similar_texts)
```

```
Out[66]: 2
```

```
In [67]: for i, txt in enumerate(similar_texts):
             print(i,"):\n", txt)
```

```
0 ):
 One day, a girl named Mia went for a walk. She saw a big, scary house. It had a tall door and small
windows. Mia was brave, so she went inside the house.

In the house, Mia saw a birdcage. Inside the birdcage, there was a little bird. The bird was sad. It
wanted to fly and be free. Mia wanted to help the bird.

Mia opened the birdcage door. The bird flew out and was happy. It was not scary anymore. Mia and the
bird were friends. They played and had fun all day.
1 ):
 Once there was a little girl called Mia who loved to jump. Everywhere she went, she jumped. When wa
lking to school, she would jump on the sidewalk. At the park, she would jump into the sandbox.

One day Mia was at the supermarket and she saw something unusual. She saw a lawyer. Mia had never se
en a lawyer before so it made her very curious. She wanted to know what a lawyer did and why he was
so dressed up. So, Mia jumped right up to the lawyer and asked him.

The lawyer was very confused. He had never seen a little girl so eager to talk to him. He tried to e
xplain but Mia kept on jumping and interrupting.

Soon enough the store manager got involved. He explained to Mia that it was not appropriate to engag
e with strangers and it was wrong to interrupt people when they were talking.

Mia was very sorry for her behaviour and decided to never do something like this again. She had lear
ned her lesson that it is important to be respectful to all strangers.
```

We can see since the input size was small, the generated story has a lot of similarity with the train samples inlcuding word "mia". This issue gan get solved by using the larger train dataset. Let's do it if you have enough Memory. :)

```
In [ ]:
```