Ungraded Lab: Generating Text with Neural Networks

For this week, you will look at techniques to prepare data and build models for text generation. You will train a neural network with lyrics from an Irish song then let it make a new song for you. Though this might sound like a more complex application, you'll soon see that the process is very similar to the ones you've been using in the previous weeks. Only minor modifications are needed. Let's see what these are in the next sections.

Imports

First, you will import the required libraries. You've used all of these already in the previous labs.

In [1]:
```
import tensorflow as tf
import numpy as np
```

Building the Word Vocabulary

The dataset is the lyrics of *Lanigan's Ball*, a traditional Irish song. You will split it per line then use a `TextVectorization` layer to build the vocabulary.

In [2]:
```
# Define the lyrics of the song
data = "In the town of Athy one Jeremy Lanigan \n Battered away til he hadnt a pound. \nHis father died and made him a man aga

# Split the long string per line and put in a list
corpus = data.lower().split("\n")

# Preview the result
print(corpus)
```

```
['in the town of athy one jeremy lanigan ', ' battered away til he hadnt a pound. ', 'his father died and made him a man again
', ' left him a farm and ten acres of ground. ', 'he gave a grand party for friends and relations ', 'who didnt forget him when
come to the wall, ', 'and if youll but listen ill make your eyes glisten ', 'of the rows and the ructions of lanigans ball. ',
'myself to be sure got free invitation, ', 'for all the nice girls and boys i might ask, ', 'and just in a minute both friends
and relations ', 'were dancing round merry as bees round a cask. ', 'judy odaly, that nice little milliner, ', 'she tipped me a
wink for to give her a call, ', 'and i soon arrived with peggy mcgilligan ', 'just in time for lanigans ball. ', 'there were la
shings of punch and wine for the ladies, ', 'potatoes and cakes; there was bacon and tea, ', 'there were the nolans, dolans, og
radys ', 'courting the girls and dancing away. ', 'songs they went round as plenty as water, ', 'the harp that once sounded in
taras old hall,', 'sweet nelly gray and the rat catchers daughter,', 'all singing together at lanigans ball. ', 'they were doin
g all kinds of nonsensical polkas ', 'all round the room in a whirligig. ', 'julia and i, we banished their nonsense ', 'and ti
pped them the twist of a reel and a jig. ', 'ach mavrone, how the girls got all mad at me ', 'danced til youd think the ceiling
would fall. ', 'for i spent three weeks at brooks academy ', 'learning new steps for lanigans ball. ', 'three long weeks i spen
t up in dublin, ', 'three long weeks to learn nothing at all,', ' three long weeks i spent up in dublin, ', 'learning new steps
for lanigans ball. ', 'she stepped out and i stepped in again, ', 'i stepped out and she stepped in again, ', 'she stepped out
and i stepped in again, ', 'learning new steps for lanigans ball. ', 'boys were all merry and the girls they were hearty ', 'an
d danced all around in couples and groups, ', 'til an accident happened, young terrance mccarthy ', 'put his right leg through
miss finnertys hoops. ', 'poor creature fainted and cried meelia murther, ', 'called for her brothers and gathered them all. ',
'carmody swore that hed go no further ', 'til he had satisfaction at lanigans ball. ', 'in the midst of the row miss kerrigan f
ainted, ', 'her cheeks at the same time as red as a rose. ', 'some of the lads declared she was painted, ', 'she took a small d
rop too much, i suppose. ', 'her sweetheart, ned morgan, so powerful and able, ', 'when he saw his fair colleen stretched out b
y the wall, ', 'tore the left leg from under the table ', 'and smashed all the chaneys at lanigans ball. ', 'boys, oh boys, twa
s then there were runctions. ', 'myself got a lick from big phelim mchugh. ', 'i soon replied to his introduction ', 'and kicke
d up a terrible hullabaloo. ', 'old casey, the piper, was near being strangled. ', 'they squeezed up his pipes, bellows, chante
rs and all. ', 'the girls, in their ribbons, they got all entangled ', 'and that put an end to lanigans ball.']
```

In [3]:
```
# Initialize the vectorization layer
vectorize_layer = tf.keras.layers.TextVectorization()

# Build the vocabulary
vectorize_layer.adapt(corpus)
```

You can view the results with the cell below. The resulting vocabulary is 262 words (not including the special tokens for padding and out-of-vocabulary words). You will these variables later.

In [4]:
```
# Get the vocabulary and its size
vocabulary = vectorize_layer.get_vocabulary()
vocab_size = len(vocabulary)

print(f'{vocabulary}')
print(f'{vocab_size}')
```

```
['', '[UNK]', 'and', 'the', 'a', 'in', 'all', 'i', 'for', 'of', 'lanigans', 'ball', 'were', 'at', 'to', 'stepped', 'she', 'they
', 'his', 'girls', 'as', 'weeks', 'up', 'til', 'three', 'there', 'that', 'round', 'out', 'her', 'he', 'got', 'boys', 'again', '
was', 'steps', 'spent', 'new', 'long', 'learning', 'him', 'when', 'wall', 'tipped', 'time', 'them', 'their', 'soon', 'relations
', 'put', 'old', 'nice', 'myself', 'miss', 'merry', 'me', 'leg', 'left', 'just', 'from', 'friends', 'fainted', 'dublin', 'danci
ng', 'danced', 'away', 'an', 'your', 'young', 'youll', 'youd', 'would', 'with', 'wink', 'wine', 'who', 'whirligig', 'went', 'we
', 'water', 'under', 'twist', 'twas', 'town', 'tore', 'took', 'too', 'together', 'through', 'think', 'then', 'terrible', 'terra
nce', 'ten', 'tea', 'taras', 'table', 'swore', 'sweetheart', 'sweet', 'sure', 'suppose', 'stretched', 'strangled', 'squeezed',
'sounded', 'songs', 'some', 'so', 'smashed', 'small', 'singing', 'saw', 'satisfaction', 'same', 'runctions', 'ructions', 'rows'
, 'row', 'rose', 'room', 'right', 'ribbons', 'replied', 'reel', 'red', 'rat', 'punch', 'powerful', 'pound', 'potatoes', 'poor',
'polkas', 'plenty', 'pipes', 'piper', 'phelim', 'peggy', 'party', 'painted', 'one', 'once', 'oh', 'ogradys', 'odaly', 'nothing'
, 'nonsensical', 'nonsense', 'nolans', 'no', 'nelly', 'ned', 'near', 'murther', 'much', 'morgan', 'minute', 'milliner', 'might'
, 'midst', 'meelia', 'mchugh', 'mcgilligan', 'mccarthy', 'mavrone', 'man', 'make', 'made', 'mad', 'little', 'listen', 'lick', '
learn', 'lashings', 'lanigan', 'lads', 'ladies', 'kinds', 'kicked', 'kerrigan', 'julia', 'judy', 'jig', 'jeremy', 'invitation',
'introduction', 'ill', 'if', 'hullabaloo', 'how', 'hoops', 'hed', 'hearty', 'harp', 'happened', 'hall', 'hadnt', 'had', 'groups
', 'ground', 'gray', 'grand', 'go', 'glisten', 'give', 'gave', 'gathered', 'further', 'free', 'forget', 'finnertys', 'father',
'farm', 'fall', 'fair', 'eyes', 'entangled', 'end', 'drop', 'dolans', 'doing', 'died', 'didnt', 'declared', 'daughter', 'cried'
, 'creature', 'courting', 'couples', 'come', 'colleen', 'cheeks', 'chanters', 'chaneys', 'ceiling', 'catchers', 'cask', 'casey'
, 'carmody', 'called', 'call', 'cakes', 'by', 'but', 'brothers', 'brooks', 'both', 'big', 'bellows', 'being', 'bees', 'be', 'ba
ttered', 'banished', 'bacon', 'athy', 'ask', 'arrived', 'around', 'acres', 'ach', 'accident', 'academy', 'able']
264
```

Preprocessing the Dataset

As discussed in the lectures, you will take each line of the song and create inputs and labels from it. For example, if you only have one sentence: "I am using Tensorflow", you want the model to learn the next word given any subphrase of this sentence:

```
INPUT              LABEL
-----------------------------
I             ---> am
I am          ---> using
I am using    ---> Tensorflow
```

The next cell shows how to implement this concept in code. The result would be inputs as padded sequences, and labels as one-hot encoded arrays.

In [5]:
```
# Initialize the sequences list
input_sequences = []

# Loop over every line
for line in corpus:

        # Generate the integer sequence of the current line
        sequence = vectorize_layer(line).numpy()

        # Loop over the line several times to generate the subphrases
        for i in range(1, len(sequence)):

                # Generate the subphrase
                n_gram_sequence = sequence[:i+1]

                # Append the subphrase to the sequences list
                input_sequences.append(n_gram_sequence)

# Get the length of the longest line
max_sequence_len = max([len(x) for x in input_sequences])

# Pad all sequences
input_sequences = np.array(tf.keras.utils.pad_sequences(input_sequences, maxlen=max_sequence_len, padding='pre'))

# Create inputs and label by splitting the last token in the subphrases
xs, labels = input_sequences[:,:-1],input_sequences[:,-1]

# Convert the label into one-hot arrays
ys = tf.keras.utils.to_categorical(labels, num_classes=vocab_size)
```

Check the result for the first line of the song. The particular line and the expected token sequence is shown in the cell below:

In [6]:
```
# Get sample sentence
sentence = corpus[0].split()
print(f'sample sentence: {sentence}')

# Initialize token list
token_list = []

# Look up the indices of each word and append to the list
for word in sentence:
```

```
  token_list.append(vocabulary.index(word))

# Print the token list
print(token_list)
```

```
sample sentence: ['in', 'the', 'town', 'of', 'athy', 'one', 'jeremy', 'lanigan']
[5, 3, 83, 9, 255, 140, 183, 174]
```

Since there are 8 tokens here, you can expect to find this particular line in the first 7 elements of the `xs` that you generated earlier. If we get the longest subphrase generated, that should be found in `xs[6]` . See the padded token sequence below:

In [7]:
```
def sequence_to_text(sequence, vocabulary):
  '''utility to convert integer sequence back to text'''

  # Loop through the integer sequence and look up the word from the vocabulary
  words = [vocabulary[index] for index in sequence]

  # Combine the words into one sentence
  text = tf.strings.reduce_join(words, separator=' ').numpy().decode()

  return text
```

In [8]:
```
# Pick element
elem_number = 6

# Print token list and phrase
print(f'token list: {xs[elem_number]}')
print(f'decoded to text: {sequence_to_text(xs[elem_number], vocabulary)}')
```

```
token list: [  0   0   0   5   3  83   9 255 140 183]
decoded to text:    in the town of athy one jeremy
```

If you print out the label, it should show `174` because that is the next word in the phrase (i.e. `lanigan`). See the one-hot encoded form below. You can use the `np.argmax()` method to get the index of the 'hot' label.

In [9]:
```
# Print label
print(f'one-hot label: {ys[elem_number]}')
print(f'index of label: {np.argmax(ys[elem_number])}')
```

```
one-hot label: [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
index of label: 174
```

If you pick the element before that, you will see the same subphrase as above minus one word:

In [10]:
```
# Pick element
elem_number = 5

# Print token list and phrase
print(f'token list: {xs[elem_number]}')
print(f'decoded to text: {sequence_to_text(xs[elem_number], vocabulary)}')

# Print label
print(f'one-hot label: {ys[elem_number]}')
print(f'index of label: {np.argmax(ys[elem_number])}')
```

```
token list: [  0   0   0   0   5   3  83   9 255 140]
decoded to text:     in the town of athy one
one-hot label: [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
index of label: 183
```

Build the Model

Next, you will build the model with basically the same layers as before. The main difference is you will remove the sigmoid output and use a softmax activated `Dense` layer instead. This output layer will have one neuron for each word in the vocabulary. So given an input token list, the output array of the final layer will have the probabilities for each word.

In [11]:
```
# Build the model
model = tf.keras.models.Sequential([
            tf.keras.Input(shape=(max_sequence_len-1,)),
            tf.keras.layers.Embedding(vocab_size, 64),
            tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(20)),
            tf.keras.layers.Dense(vocab_size, activation='softmax')
])

# Use categorical crossentropy because this is a multi-class problem
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

# Print the model summary
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 10, 64)	16,896
bidirectional (Bidirectional)	(None, 40)	13,600
dense (Dense)	(None, 264)	10,824

Total params: 41,320 (161.41 KB)
Trainable params: 41,320 (161.41 KB)
Non-trainable params: 0 (0.00 B)

Train the model

You can now train the model. We have a relatively small vocabulary so it will only take a couple of minutes to complete 500 epochs.

In [12]:
```
# Train the model
history = model.fit(xs, ys, epochs=500)
```

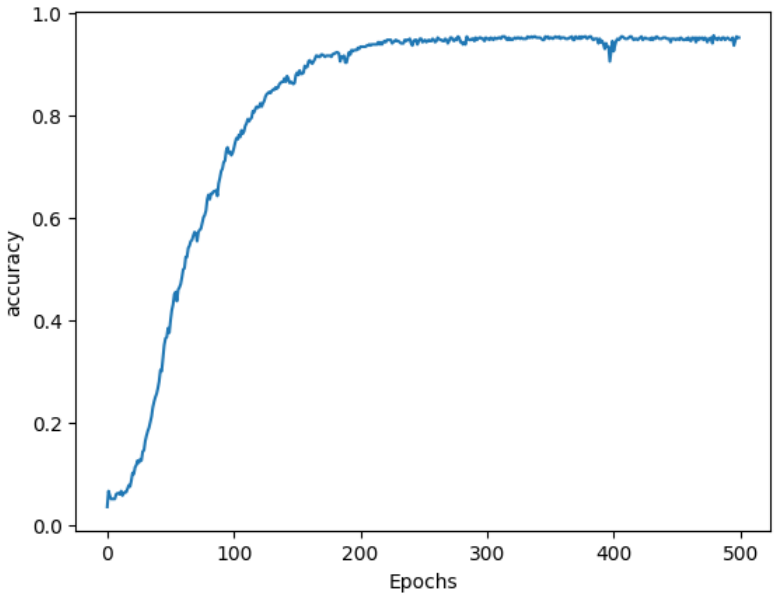
```
Epoch 481/500
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9427 - loss: 0.1426
Epoch 482/500
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9454 - loss: 0.1332
Epoch 483/500
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9640 - loss: 0.1048
Epoch 484/500
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9515 - loss: 0.1321
Epoch 485/500
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9594 - loss: 0.1126
Epoch 486/500
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9480 - loss: 0.1332
Epoch 487/500
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9502 - loss: 0.1175
Epoch 488/500
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9558 - loss: 0.1174
Epoch 489/500
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9521 - loss: 0.1165
Epoch 490/500
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9531 - loss: 0.1184
Epoch 491/500
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9504 - loss: 0.1184
Epoch 492/500
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9502 - loss: 0.1235
Epoch 493/500
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9403 - loss: 0.1358
Epoch 494/500
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9581 - loss: 0.1158
Epoch 495/500
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9502 - loss: 0.1225
Epoch 496/500
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9227 - loss: 0.1386
Epoch 497/500
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9484 - loss: 0.1134
Epoch 498/500
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9434 - loss: 0.1302
Epoch 499/500
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9508 - loss: 0.1100
Epoch 500/500
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9523 - loss: 0.1241
```

You can visualize the results with the utility below. With the default settings, you should see around 95% accuracy after 500 epochs.

In [13]:
```python
import matplotlib.pyplot as plt

# Plot utility
def plot_graphs(history, string):
  plt.plot(history.history[string])
  plt.xlabel("Epochs")
  plt.ylabel(string)
  plt.show()

# Visualize the accuracy
plot_graphs(history, 'accuracy')
```



Generating Text

With the model trained, you can now use it to make its own song! The process would look like:

1. Feed a seed text to initiate the process.
2. Model predicts the index of the most probable next word.
3. Look up the index in the reverse word index dictionary
4. Append the next word to the seed text.
5. Feed the result to the model again.

Steps 2 to 5 will repeat until the desired length of the song is reached. See how it is implemented in the code below:

In [14]:
```
# Define seed text
seed_text = "Laurence went to Dublin"

# Define total words to predict
next_words = 100

# Loop until desired length is reached
for _ in range(next_words):

        # Convert the seed text to an integer sequence
        sequence = vectorize_layer(seed_text)

        # Pad the sequence
        sequence = tf.keras.utils.pad_sequences([sequence], maxlen=max_sequence_len-1, padding='pre')

        # Feed to the model and get the probabilities for each index
        probabilities = model.predict(sequence, verbose=0)

        # Get the index with the highest probability
        predicted = np.argmax(probabilities, axis=-1)[0]

        # Ignore if index is 0 because that is just the padding.
        if predicted != 0:

                # Look up the word associated with the index.
                output_word = vocabulary[predicted]

                # Combine with the seed text
                seed_text += " " + output_word

# Print the result
print(seed_text)
```

```
Laurence went to Dublin athy ructions of the wall mchugh your one fall glisten hearty hearty hearty glisten glisten glisten gli
sten hearty accident eyes glisten glisten glisten glisten glisten glisten glisten glisten swore gave me steps for her t
was call call call call call call eyes glisten accident call glisten glisten glisten glisten glisten glisten glisten gl
isten accident eyes eyes glisten accident eyes glisten glisten glisten glisten swore round a call end together her in ground ga
ve brooks call call rose rose call end new ball rose rose me me a a a rose rose rose call call call call hullabaloo glisten
```

In the output above, you might notice frequent repetition of words the longer the sentence gets. There are ways to get around it and the next cell shows one. Basically, instead of getting the index with max probability, you will get the top three indices and choose one at random. See if the output text makes more sense with this approach. This is not the most time efficient solution because it is always sorting the entire array even if you only need the top three. Feel free to improve it and of course, you can also develop your own method of picking the next word.

In [15]:
```
# Define seed text
seed_text = "Laurence went to Dublin"

# Define total words to predict
next_words = 100

# Loop until desired length is reached
for _ in range(next_words):

        # Convert the seed text to an integer sequence
  sequence = vectorize_layer(seed_text)

        # Pad the sequence
  sequence = tf.keras.utils.pad_sequences([sequence], maxlen=max_sequence_len-1, padding='pre')

        # Feed to the model and get the probabilities for each index
  probabilities = model.predict(sequence, verbose=0)

  # Pick a random number from [1,2,3]
  choice = np.random.choice([1,2,3])

  # Sort the probabilities in ascending order
  # and get the random choice from the end of the array
  predicted = np.argsort(probabilities)[0][-choice]

        # Ignore if index is 0 because that is just the padding.
  if predicted != 0:

    # Look up the word associated with the index.
    output_word = vocabulary[predicted]

    # Combine with the seed text
    seed_text += " " + output_word

# Print the result
print(seed_text)
```

```
Laurence went to Dublin there the ructions i might fall make ladies died for til eyes accident ask eyes glisten swore swore thi
nk round a strangled cask water swore learn ask glisten invitation together new red gray together new mchugh ground jig a suppo
se rose hall rose call declared phelim mchugh right mchugh jig right hall declared declared end in red out stepped of a pound p
ainted water water a jig jig eyes eyes glisten swore swore round nelly gray gray were nonsensical hearty accident eyes accident
eyes think creature swore sure together new mchugh a rose call suppose hall hall call jig end
```

Wrap Up

In this lab, you got a first look at preparing data and building a model for text generation. The corpus is fairly small in this particular exercise and in the next lessons, you will be building one from a larger body of text. See you there!

Run the cell below to free up resources for the next lab.

In [16]:
```
# Shutdown the kernel to free up resources.
# Note: You can expect a pop-up when you run this cell. You can safely ignore that and just press `Ok`.

from IPython import get_ipython

k = get_ipython().kernel

k.do_shutdown(restart=False)
```

Out[16]:
```
{'status': 'ok', 'restart': False}
```