Copyright Notice

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 Lanigan's 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.

```
tokenizer = Tokenizer()

data="In the town of Athy one Jeremy Lanigan \n Battered away ... ..."
corpus = data.lower().split("\n")

tokenizer.fit_on_texts(corpus)
total_words = len(tokenizer.word_index) + 1
```

```
tokenizer = Tokenizer()

data="In the town of Athy one Jeremy Lanigan \n Battered away ... ..."
corpus = data.lower().split("\n")

tokenizer.fit_on_texts(corpus)
total_words = len(tokenizer.word_index) + 1
```

```python
tokenizer = Tokenizer()

data="In the town of Athy one Jeremy Lanigan \n Battered away ... ..."
corpus = data.lower().split("\n")

tokenizer.fit_on_texts(corpus)
total_words = len(tokenizer.word_index) + 1
```

```
tokenizer = Tokenizer()

data="In the town of Athy one Jeremy Lanigan \n Battered away ... ..."
corpus = data.lower().split("\n")

tokenizer.fit_on_texts(corpus)
total_words = len(tokenizer.word_index) + 1
```

```
tokenizer = Tokenizer()

data="In the town of Athy one Jeremy Lanigan \n Battered away ... ..."
corpus = data.lower().split("\n")

tokenizer.fit_on_texts(corpus)
total_words = len(tokenizer.word_index) + 1
```

add 1 'cause the outer vocab word

```
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)
```

```
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)
```

initialize the list

```
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)
```

initialize the token_list

In the town of Athy one Jeremy Lanigan

$\Downarrow$

[4 2 66 8 67 68 69 70]

convert the sentence into the list of tokens representing words

```
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)
```

iterate over the token_list

take 1st i+1 words in the i-th sentence

Line:	Input Sequences:
[4 2 66 8 67 68 69 70]	[4 2]
	[4 2 66]
	[4 2 66 8]
	[4 2 66 8 67]
	[4 2 66 8 67 68]
	[4 2 66 8 67 68 69]
	[4 2 66 8 67 68 69 70]

```
max_sequence_len = max([len(x) for x in input_sequences])
```

Find the length of the longest

```
input_sequences =
    np.array(pad_sequences(input_sequences, maxlen=max_sequence_len, padding='pre'))
```

pad all the sequences to have the same length

Line:

[4 2 66 8 67 68 69 70]

Padded Input Sequences:

```
[0 0 0 0 0 0 0 0 0 0 4 2]
[0 0 0 0 0 0 0 0 0 4 2 66]
[0 0 0 0 0 0 0 0 4 2 66 8]
[0 0 0 0 0 0 0 4 2 66 8 67]
[0 0 0 0 0 0 4 2 66 8 67 68]
[0 0 0 0 0 4 2 66 8 67 68 69]
[0 0 0 0 4 2 66 8 67 68 69 70]
```

Padded Input Sequences:

take the last number and set as a label

```
[0 0 0 0 0 0 0 0 0 0 4 2]
[0 0 0 0 0 0 0 0 0 4 2 66]
[0 0 0 0 0 0 0 0 4 2 66 8]
[0 0 0 0 0 0 0 4 2 66 8 67]
[0 0 0 0 0 0 4 2 66 8 67 68]
[0 0 0 0 0 4 2 66 8 67 68 69]
[0 0 0 0 4 2 66 8 67 68 69 70]
```

The rest is input

Padded Input Sequences:

Input (X)

Label (Y)

```
[0 0 0 0 0 0 0 0 0 0 4 2]

[0 0 0 0 0 0 0 0 0 4 2 66]

[0 0 0 0 0 0 0 0 4 2 66 8]

[0 0 0 0 0 0 0 4 2 66 8 67]

[0 0 0 0 0 0 4 2 66 8 67 68]

[0 0 0 0 0 4 2 66 8 67 68 69]

[0 0 0 0 4 2 66 8 67 68 69 70]
```

Padded Input Sequences:

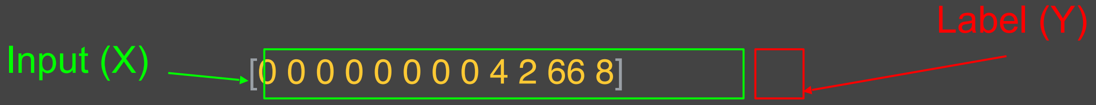
Input (X)

Label (Y)

```
[0 0 0 0 0 0 0 0 0 0 4 2]
[0 0 0 0 0 0 0 0 0 4 2 66]
[0 0 0 0 0 0 0 0 4 2 66 8]
[0 0 0 0 0 0 0 4 2 66 8 67]
[0 0 0 0 0 0 4 2 66 8 67 68]
[0 0 0 0 0 4 2 66 8 67 68 69]
[0 0 0 0 4 2 66 8 67 68 69 70]
```

Padded Input Sequences:

```
[0 0 0 0 0 0 0 0 0 0 4 2]
[0 0 0 0 0 0 0 0 0 4 2 66]
[0 0 0 0 0 0 0 0 4 2 66 8]
[0 0 0 0 0 0 0 4 2 66 8 67]
[0 0 0 0 0 0 4 2 66 8 67 68]
[0 0 0 0 0 4 2 66 8 67 68 69]
[0 0 0 0 4 2 66 8 67 68 69 70]
```



take first n tokens

```
xs = input_sequences[:,:-1]
labels = input_sequences[:,-1]
```

last token is label

```
ys = tf.keras.utils.to_categorical(labels, num_classes=total_words)
```

convert list into categorical
one-hot

```
Sentence: [0 0 0 0 4 2 66 8 67 68 69 70]

       X:[0 0 0 0 4 2 66 8 67 68 69]

   Label:[ 70 ]

      Y: [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
           0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
           0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  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.  0.  0.
           0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
           0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
           0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
           0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
           0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
           0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
```

70

||

set 1 at index 70th

```
Sentence: [0 0 0 0 4 2 66 8 67 68 69 70]

       X: [0 0 0 0 4 2 66 8 67 68 69]

   Label: [ 70 ]

       Y: [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
            0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
            0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  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.  0.  0.
            0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
            0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
            0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
            0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
            0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
            0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
```

```
model = Sequential()
model.add(Embedding(total_words, 64, input_length=max_sequence_len - 1))
model.add((LSTM(20)))
model.add(Dense(total_words, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(xs, ys, epochs=500, verbose=1)
```

```
model = Sequential()
model.add(Embedding(total_words, 64, input_length=max_sequence_len - 1))
model.add((LSTM(20)))
model.add(Dense(total_words, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(xs, ys, epochs=500, verbose=1)
```

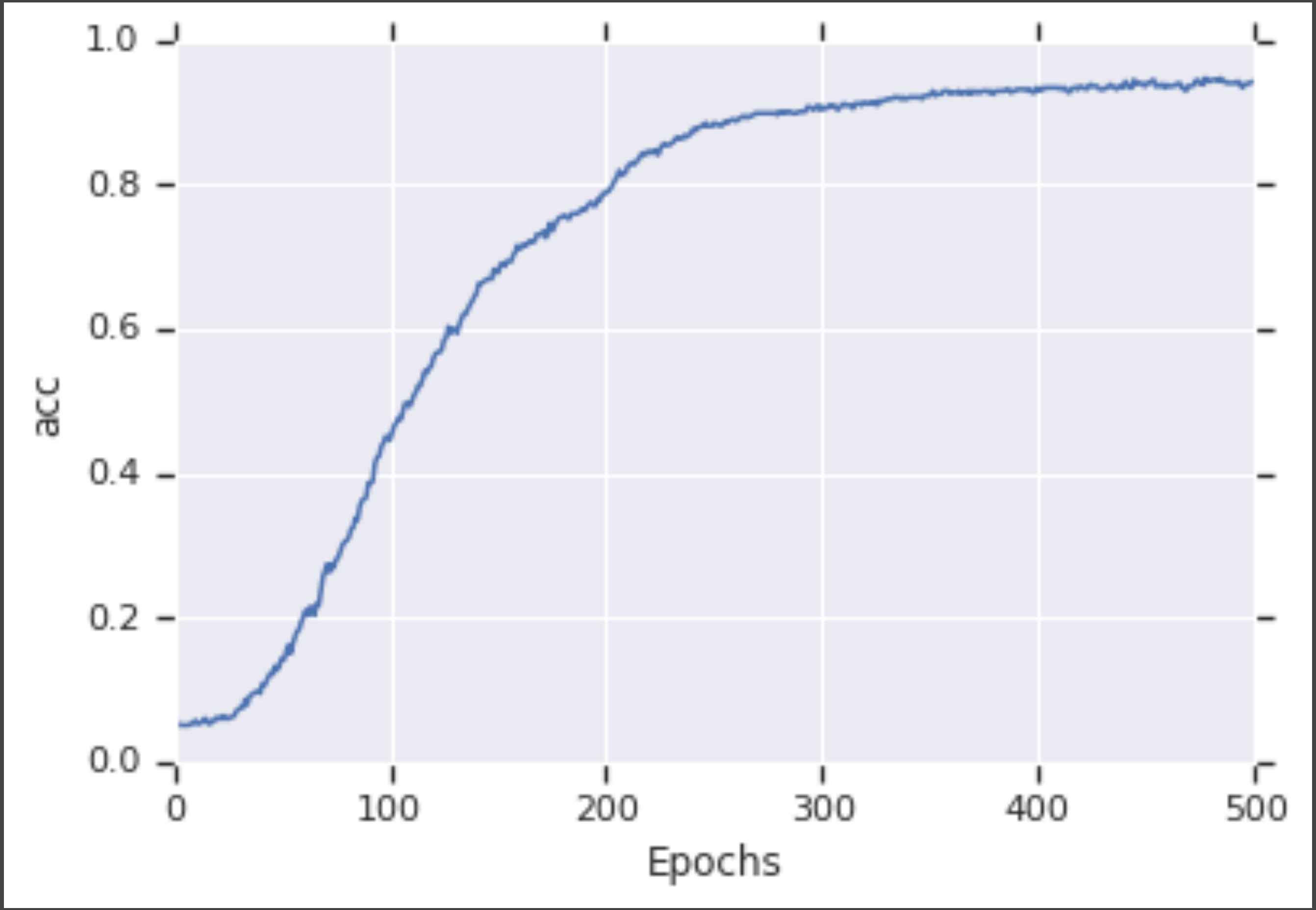
```
model = Sequential()
model.add(Embedding(total_words, 64, input_length=max_sequence_len - 1))
model.add((LSTM(20)))
model.add(Dense(total_words, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(xs, ys, epochs=500, verbose=1)
```

```
model = Sequential()
model.add(Embedding(total_words, 64, input_length=max_sequence_len - 1))
model.add((LSTM(20)))
model.add(Dense(total_words, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(xs, ys, epochs=500, verbose=1)
```

```
model = Sequential()
model.add(Embedding(total_words, 64, input_length=max_sequence_len - 1))
model.add((LSTM(20)))
model.add(Dense(total_words, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(xs, ys, epochs=500, verbose=1)
```

```
model = Sequential()
model.add(Embedding(total_words, 64, input_length=max_sequence_len - 1))
model.add((LSTM(20)))
model.add(Dense(total_words, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(xs, ys, epochs=500, verbose=1)
```

```
model = Sequential()
model.add(Embedding(total_words, 64, input_length=max_sequence_len - 1))
model.add((LSTM(20)))
model.add(Dense(total_words, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(xs, ys, epochs=500, verbose=1)
```


1.0
0.8
0.6
0.4
0.2
0.0
acc
0
100
200
300
400
500
Epochs

Laurence went to dublin round the plenty as red wall me for wall wall
Laurence went to dublin odaly of the nice of lanigans ball ball ball hall
Laurence went to dublin he hadnt a minute both relations hall new relations youd

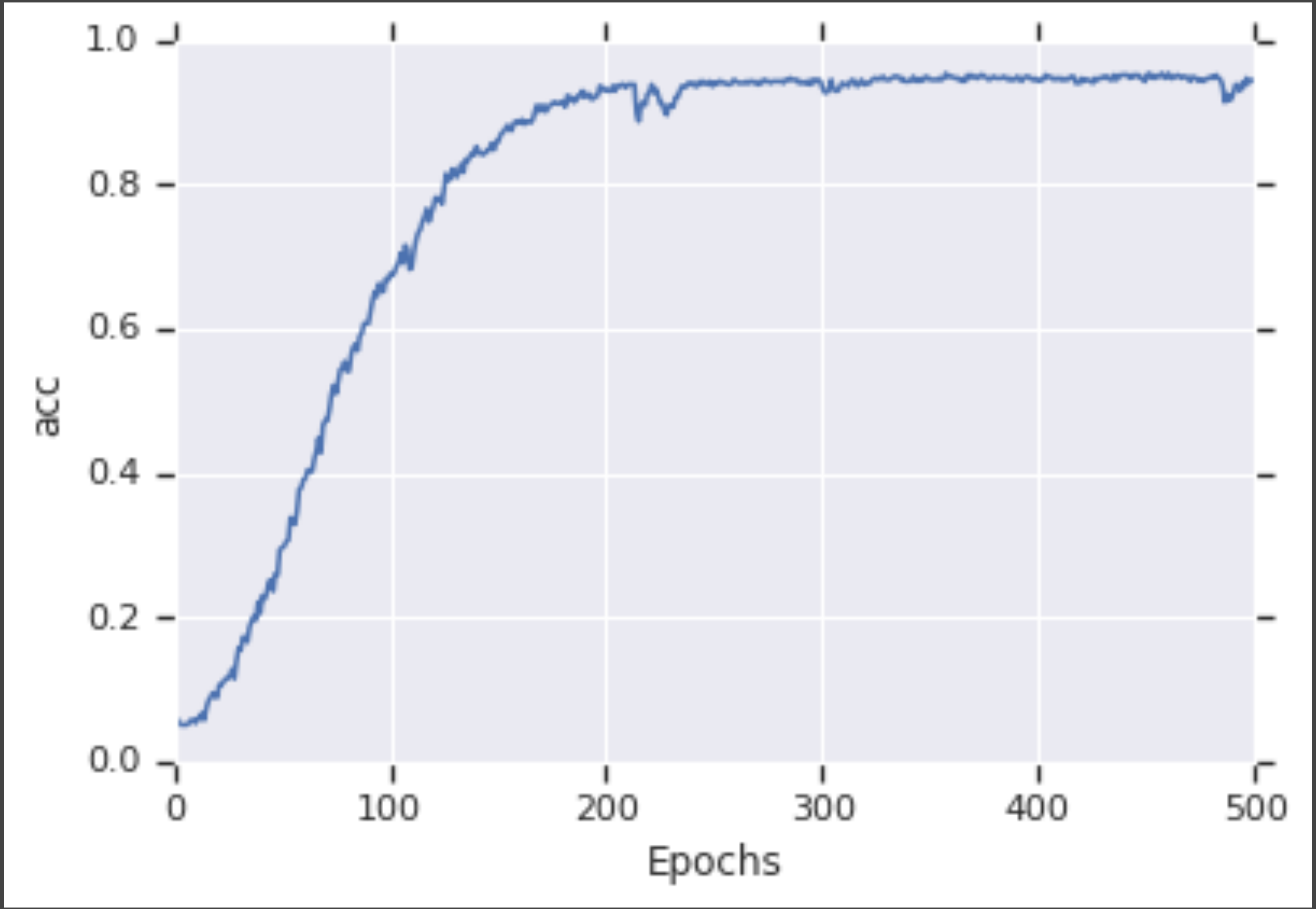
Laurence went to dublin round the plenty as red wall me for wall wall
Laurence went to dublin odaly of the nice of lanigans ball ball ball hall
Laurence went to dublin he hadnt a minute both relations hall new relations youd

Laurence went to dublin round the plenty as red wall me for wall wall
Laurence went to dublin odaly of the nice of lanigans ball ball ball hall
Laurence went to dublin he hadnt a minute both relations hall new relations youd

Laurence went to dublin round the plenty as red wall me for wall wall
Laurence went to dublin odaly of the nice of lanigans ball ball ball hall
Laurence went to dublin he hadnt a minute both relations hall new relations youd

```
model = Sequential()
model.add(Embedding(total_words, 64, input_length=max_sequence_len - 1))
model.add(Bidirectional(LSTM(20)))
model.add(Dense(total_words, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(xs, ys, epochs=500, verbose=1)
```

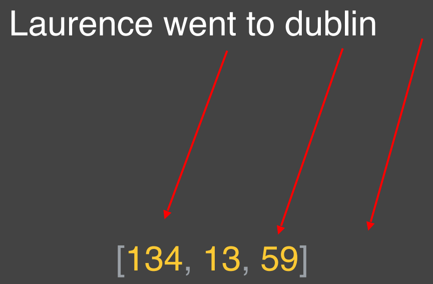
```
model = Sequential()
model.add(Embedding(total_words, 64, input_length=max_sequence_len - 1))
model.add(Bidirectional(LSTM(20)))
model.add(Dense(total_words, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(xs, ys, epochs=500, verbose=1)
```


acc
Epochs
0.0
0.2
0.4
0.6
0.8
1.0
0
100
200
300
400
500

Laurence went to dublin think and wine for lanigans ball entangled in nonsense me
Laurence went to dublin his pipes bellows chanters and all all entangled all kinds
Laurence went to dublin how the room a whirligig ructions long at brooks fainted

Laurence went to dublin

```
token_list = tokenizer.texts_to_sequences([seed_text])[0]
```


Laurence went to dublin
[134, 13, 59]

```
token_list = pad_sequences([token_list], maxlen=max_sequence_len - 1, padding='pre')
```

```
[  0   0   0   0   0   0   0 134  13  59]
```

```
predicted = model.predict(token_list)
predicted = np.argmax(probabilities, axis= - 1)[0]
```

```
output_word = tokenizer.index_word[predicted]
seed_text += " " + output_word
seed_text = "Laurence went to dublin"
next_words = 10

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 = output_word = tokenizer.index_word[predicted]
  seed_text += " " + output_word
print(seed_text)
```

Laurence went to dublin round a cask cask cask cask cask
squeezed forget tea twas make eyes glisten mchugh mchugh
lanigan lanigan glisten glisten glisten glisten glisten
glisten glisten glisten glisten glisten glisten glisten
glisten glisten glisten glisten glisten glisten glisten
glisten glisten glisten glisten glisten glisten glisten
glisten glisten glisten glisten glisten glisten glisten
glisten glisten glisten glisten glisten glisten glisten
glisten glisten glisten glisten glisten glisten glisten
glisten glisten glisten glisten glisten glisten glisten
glisten glisten glisten glisten glisten glisten glisten
glisten glisten glisten glisten glisten glisten glisten
glisten glisten glisten glisten glisten glisten glisten
glisten glisten glisten glisten glisten glisten glisten

```
!wget --no-check-certificate \
    https://storage.googleapis.com/laurencemoroney-blog.appspot.com/irish-lyrics-eof.txt \
    -O /tmp/irish-lyrics-eof.txt
```

```
data = open('/tmp/irish-lyrics-eof.txt').read()
```

```
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'))
adam = Adam(lr=0.01)
model.compile(loss='categorical_crossentropy', optimizer=adam, metrics=['accuracy'])
history = model.fit(xs, ys, epochs=100, verbose=1)
```

```
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'))
adam = Adam(lr=0.01)
model.compile(loss='categorical_crossentropy', optimizer=adam, metrics=['accuracy'])
history = model.fit(xs, ys, epochs=100, verbose=1)
```

```
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'))
adam = Adam(lr=0.01)
model.compile(loss='categorical_crossentropy', optimizer=adam, metrics=['accuracy'])
history = model.fit(xs, ys, epochs=100, verbose=1)
```

```
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'))
adam = Adam(lr=0.01)
model.compile(loss='categorical_crossentropy', optimizer=adam, metrics=['accuracy'])
history = model.fit(xs, ys, epochs=100, verbose=1)
```

```
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'))
adam = Adam(lr=0.01)
model.compile(loss='categorical_crossentropy', optimizer=adam, metrics=['accuracy'])
history = model.fit(xs, ys, epochs=100, verbose=1)
```

Help Me Obi-Wan Kenobi, you're my only hope
my dear
and hope as i did fly with its flavours
along with all its joys
but sure i will build
love you still
gold it did join
do mans run away cross our country
are wedding i was down to
off holyhead wished meself
down among the pigs
played some hearty rigs
me embarrass
find me brother
me chamber she gave me
who storied be irishmen
to greet you
lovely molly
gone away from me home
home to leave the old tin cans
the foemans chain one was shining
sky above i think i love

https://www.tensorflow.org/tutorials/sequences/text_generation