Lab 8 Neural Language Model

A language model predicts the next word in the sequence based on the specific words that have come before it in the sequence.

It is also possible to develop language models at the character level using neural networks. The benefit of character-based language models is their small vocabulary and flexibility in handling any words, punctuation, and other document structure. This comes at the cost of requiring larger models that are slower to train.

Nevertheless, in the field of neural language models, character-based models offer a lot of promise for a general, flexible and powerful approach to language modeling.

As a prerequisite for the lab, make sure to pip install:

- **keras**
- **tensorflow**
- **h5py**

Source Text Creation

To start out with, we'll be using a simple nursery rhyme. It's quite short so we can actually train something on your CPU and see relatively interesting results. Please copy and paste the following text in a text file and save it as "rhyme.txt". Place this in the same directory as this jupyter notebook:

In [ ]:

```
!pip install tensorflow
!pip install keras
!pip install h5py
```

```
Requirement already satisfied: tensorflow in /usr/local/lib/python3.7/dist-packages (2.7.
0)
Requirement already satisfied: keras-preprocessing>=1.1.1 in /usr/local/lib/python3.7/dis
t-packages (from tensorflow) (1.1.2)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.21.0 in /usr/local/lib/pyt
hon3.7/dist-packages (from tensorflow) (0.21.0)
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.7/dist-package
s (from tensorflow) (3.3.0)
Requirement already satisfied: h5py>=2.9.0 in /usr/local/lib/python3.7/dist-packages (fro
m tensorflow) (3.1.0)
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.7/dist-packages (f
rom tensorflow) (1.13.3)
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.7/dist-package
s (from tensorflow) (1.6.3)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.7/dist-packages (fro
m tensorflow) (1.15.0)
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.7/dist-
packages (from tensorflow) (3.10.0.2)
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.7/dist-packa
ges (from tensorflow) (0.2.0)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.7/dist-packages
 (from tensorflow) (1.1.0)
Requirement already satisfied: gast<0.5.0,>=0.2.1 in /usr/local/lib/python3.7/dist-packag
es (from tensorflow) (0.4.0)
Requirement already satisfied: wheel<1.0,>=0.32.0 in /usr/local/lib/python3.7/dist-packag
es (from tensorflow) (0.37.0)
Requirement already satisfied: flatbuffers<3.0,>=1.12 in /usr/local/lib/python3.7/dist-pa
ckages (from tensorflow) (2.0)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.7/dist-packa
ges (from tensorflow) (1.41.1)
Requirement already satisfied: numpy>=1.14.5 in /usr/local/lib/python3.7/dist-packages (f
rom tensorflow) (1.19.5)
```

```
Requirement already satisfied: keras<2.8,>=2.7.0rc0 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (2.7.0)
Requirement already satisfied: protobuf>=3.9.2 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (3.17.3)
Requirement already satisfied: tensorflow-estimator<2.8,~=2.7.0rc0 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (2.7.0)
Requirement already satisfied: libclang>=9.0.1 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (12.0.0)
Requirement already satisfied: absl-py>=0.4.0 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (0.12.0)
Requirement already satisfied: tensorboard~=2.6 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (2.7.0)
Requirement already satisfied: cached-property in /usr/local/lib/python3.7/dist-packages (from h5py>=2.9.0->tensorflow) (1.5.2)
Requirement already satisfied: setuptools>=41.0.0 in /usr/local/lib/python3.7/dist-packages (from tensorboard~=2.6->tensorflow) (57.4.0)
Requirement already satisfied: werkzeug>=0.11.15 in /usr/local/lib/python3.7/dist-packages (from tensorboard~=2.6->tensorflow) (1.0.1)
Requirement already satisfied: google-auth<3,>=1.6.3 in /usr/local/lib/python3.7/dist-packages (from tensorboard~=2.6->tensorflow) (1.35.0)
Requirement already satisfied: tensorboard-plugin-wit>=1.6.0 in /usr/local/lib/python3.7/dist-packages (from tensorboard~=2.6->tensorflow) (1.8.0)
Requirement already satisfied: tensorboard-data-server<0.7.0,>=0.6.0 in /usr/local/lib/python3.7/dist-packages (from tensorboard~=2.6->tensorflow) (0.6.1)
Requirement already satisfied: google-auth-oauthlib<0.5,>=0.4.1 in /usr/local/lib/python3.7/dist-packages (from tensorboard~=2.6->tensorflow) (0.4.6)
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.7/dist-packages (from tensorboard~=2.6->tensorflow) (3.3.4)
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.7/dist-packages (from tensorboard~=2.6->tensorflow) (2.23.0)
Requirement already satisfied: cachetools<5.0,>=2.0.0 in /usr/local/lib/python3.7/dist-packages (from google-auth<3,>=1.6.3->tensorboard~=2.6->tensorflow) (4.2.4)
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.7/dist-packages (from google-auth<3,>=1.6.3->tensorboard~=2.6->tensorflow) (0.2.8)
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.7/dist-packages (from google-auth<3,>=1.6.3->tensorboard~=2.6->tensorflow) (4.7.2)
Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/local/lib/python3.7/dist-packages (from google-auth-oauthlib<0.5,>=0.4.1->tensorboard~=2.6->tensorflow) (1.3.0)
Requirement already satisfied: importlib-metadata in /usr/local/lib/python3.7/dist-packages (from markdown>=2.6.8->tensorboard~=2.6->tensorflow) (4.8.1)
Requirement already satisfied: pyasn1<0.5.0,>=0.4.6 in /usr/local/lib/python3.7/dist-packages (from pyasn1-modules>=0.2.1->google-auth<3,>=1.6.3->tensorboard~=2.6->tensorflow) (0.4.8)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packages (from requests<3,>=2.21.0->tensorboard~=2.6->tensorflow) (3.0.4)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/lib/python3.7/dist-packages (from requests<3,>=2.21.0->tensorboard~=2.6->tensorflow) (1.24.3)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-packages (from requests<3,>=2.21.0->tensorboard~=2.6->tensorflow) (2021.10.8)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (from requests<3,>=2.21.0->tensorboard~=2.6->tensorflow) (2.10)
Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.7/dist-packages (from requests-oauthlib>=0.7.0->google-auth-oauthlib<0.5,>=0.4.1->tensorboard~=2.6->tensorflow) (3.1.1)
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.7/dist-packages (from importlib-metadata->markdown>=2.6.8->tensorboard~=2.6->tensorflow) (3.6.0)
Requirement already satisfied: keras in /usr/local/lib/python3.7/dist-packages (2.7.0)
Requirement already satisfied: h5py in /usr/local/lib/python3.7/dist-packages (3.1.0)
Requirement already satisfied: cached-property in /usr/local/lib/python3.7/dist-packages (from h5py) (1.5.2)
Requirement already satisfied: numpy>=1.14.5 in /usr/local/lib/python3.7/dist-packages (from h5py) (1.19.5)
```

In [ ]:

```
s='Sing a song of sixpence,\
A pocket full of rye.\
Four and twenty blackbirds,\
Baked in a pie.\
When the pie was opened\
The birds began to sing;\
```

```
Wasn't that a dainty dish,\
To set before the king.\
The king was in his counting house,\
Counting out his money;\
The queen was in the parlour,\
Eating bread and honey.\
The maid was in the garden,\
Hanging out the clothes,\
When down came a blackbird\
And pecked off her nose.'

with open('rhymes.txt','w') as f:
  f.write(s)
```

```
Sing a song of sixpence,
A pocket full of rye.
Four and twenty blackbirds,
Baked in a pie.

When the pie was opened
The birds began to sing;
Wasn't that a dainty dish,
To set before the king.

The king was in his counting house,
Counting out his money;
The queen was in the parlour,
Eating bread and honey.

The maid was in the garden,
Hanging out the clothes,
When down came a blackbird
And pecked off her nose.
```

Sequence Generation

A language model must be trained on the text, and in the case of a character-based language model, the input and output sequences must be characters.

The number of characters used as input will also define the number of characters that will need to be provided to the model in order to elicit the first predicted character.

After the first character has been generated, it can be appended to the input sequence and used as input for the model to generate the next character.

Longer sequences offer more context for the model to learn what character to output next but take longer to train and impose more burden on seeding the model when generating text.

We will use an arbitrary length of 10 characters for this model.

There is not a lot of text, and 10 characters is a few words.

We can now transform the raw text into a form that our model can learn; specifically, input and output sequences of characters.

In [ ]:

```
#load doc into memory
def load_doc(filename):
    # open the file as read only
    file = open(filename, 'r')
    # read all text
    text = file.read()
```

```
    # close the file
    file.close()
    return text
# save tokens to file, one dialog per line
def save_doc(lines, filename):
    data = '\n'.join(lines)
    file = open(filename, 'w')
    file.write(data)
    file.close()
```

In [ ]:

```
#load text
raw_text = load_doc('rhymes.txt')
print(raw_text)

# clean
tokens = raw_text.split()
raw_text = ' '.join(tokens)

# organize into sequences of characters
length = 10
sequences = list()
for i in range(length, len(raw_text)):
    # select sequence of tokens
    seq = raw_text[i-length:i+1]
    # store
    sequences.append(seq)
print('Total Sequences: %d' % len(sequences))
```

```
Sing a song of sixpence,A pocket full of rye.Four and twenty blackbirds,Baked in a pie.Wh
en the pie was openedThe birds began to sing;Wasn't that a dainty dish,To set before the
king.The king was in his counting house,Counting out his money;The queen was in the parlo
ur,Eating bread and honey.The maid was in the garden,Hanging out the clothes,When down ca
me a blackbirdAnd pecked off her nose.
Total Sequences: 384
```

In [ ]:

```
# save sequences to file
out_filename = 'char_sequences.txt'
save_doc(sequences, out_filename)
```

Train a Model

In this section, we will develop a neural language model for the prepared sequence data.

The model will read encoded characters and predict the next character in the sequence. A Long Short-Term Memory recurrent neural network hidden layer will be used to learn the context from the input sequence in order to make the predictions.

In [ ]:

```
from numpy import array
from pickle import dump
from tensorflow.keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM


# load doc into memory
def load_doc(filename):
    # open the file as read only
    file = open(filename, 'r')
    # read all text
    text = file.read()
    # close the file
    file.close()
```

```
    return text
```

In [ ]:

```
# load

in_filename = 'char_sequences.txt'
raw_text = load_doc(in_filename)
lines = raw_text.split('\n')
```

The sequences of characters must be encoded as integers.This means that each unique character will be assigned a specific integer value and each sequence of characters will be encoded as a sequence of integers. We can create the mapping given a sorted set of unique characters in the raw input data. The mapping is a dictionary of character values to integer values.

Next, we can process each sequence of characters one at a time and use the dictionary mapping to look up the integer value for each character. The result is a list of integer lists.

We need to know the size of the vocabulary later. We can retrieve this as the size of the dictionary mapping.

In [ ]:

```
# integer encode sequences of characters
chars = sorted(list(set(raw_text)))
mapping = dict((c, i) for i, c in enumerate(chars))
sequences = list()
for line in lines:
    # integer encode line
    encoded_seq = [mapping[char] for char in line]
    # store
    sequences.append(encoded_seq)

# vocabulary size
vocab_size = len(mapping)
print('Vocabulary Size: %d' % vocab_size)

# separate into input and output
sequences = array(sequences)
X, y = sequences[:,:-1], sequences[:,-1]
sequences = [to_categorical(x, num_classes=vocab_size) for x in X]
X = array(sequences)
y = to_categorical(y, num_classes=vocab_size)
```

```
Vocabulary Size: 38
```

The model is defined with an input layer that takes sequences that have 10 time steps and 38 features for the one hot encoded input sequences. Rather than specify these numbers, we use the second and third dimensions on the X input data. This is so that if we change the length of the sequences or size of the vocabulary, we do not need to change the model definition.

The model has a single LSTM hidden layer with 75 memory cells. The model has a fully connected output layer that outputs one vector with a probability distribution across all characters in the vocabulary. A softmax activation function is used on the output layer to ensure the output has the properties of a probability distribution.

The model is learning a multi-class classification problem, therefore we use the categorical log loss intended for this type of problem. The efficient Adam implementation of gradient descent is used to optimize the model and accuracy is reported at the end of each batch update. The model is fit for 50 training epochs.

To Do:

- **Try different numbers of memory cells**
- **Try different types and amounts of recurrent and fully connected layers**
- **Try different lengths of training epochs**
- **Try different sequence lengths and pre-processing of data**
- **Try regularization techniques such as Dropout**

In [ ]:

```python
# define model
model = Sequential()
model.add(LSTM(75, input_shape=(X.shape[1], X.shape[2])))
model.add(Dense(vocab_size, activation='softmax'))
print(model.summary())
# compile model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
# fit model
history=model.fit(X, y, epochs=100)
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 lstm (LSTM)                 (None, 75)                34200     
                                                                 
 dense (Dense)               (None, 38)                2888      
                                                                 
=================================================================
Total params: 37,088
Trainable params: 37,088
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/100
12/12 [==============================] - 2s 10ms/step - loss: 3.6097 - accuracy: 0.0885
Epoch 2/100
12/12 [==============================] - 0s 10ms/step - loss: 3.5061 - accuracy: 0.1562
Epoch 3/100
12/12 [==============================] - 0s 11ms/step - loss: 3.2463 - accuracy: 0.1589
Epoch 4/100
12/12 [==============================] - 0s 11ms/step - loss: 3.1377 - accuracy: 0.1589
Epoch 5/100
12/12 [==============================] - 0s 9ms/step - loss: 3.0630 - accuracy: 0.1589
Epoch 6/100
12/12 [==============================] - 0s 9ms/step - loss: 3.0436 - accuracy: 0.1589
Epoch 7/100
12/12 [==============================] - 0s 9ms/step - loss: 3.0194 - accuracy: 0.1589
Epoch 8/100
12/12 [==============================] - 0s 9ms/step - loss: 3.0017 - accuracy: 0.1589
Epoch 9/100
12/12 [==============================] - 0s 10ms/step - loss: 2.9810 - accuracy: 0.1589
Epoch 10/100
12/12 [==============================] - 0s 9ms/step - loss: 2.9601 - accuracy: 0.1589
Epoch 11/100
12/12 [==============================] - 0s 11ms/step - loss: 2.9328 - accuracy: 0.1589
Epoch 12/100
12/12 [==============================] - 0s 9ms/step - loss: 2.9120 - accuracy: 0.1979
Epoch 13/100
12/12 [==============================] - 0s 10ms/step - loss: 2.8767 - accuracy: 0.1693
Epoch 14/100
12/12 [==============================] - 0s 9ms/step - loss: 2.8407 - accuracy: 0.1875
Epoch 15/100
12/12 [==============================] - 0s 10ms/step - loss: 2.8033 - accuracy: 0.2161
Epoch 16/100
12/12 [==============================] - 0s 9ms/step - loss: 2.7785 - accuracy: 0.1797
Epoch 17/100
12/12 [==============================] - 0s 9ms/step - loss: 2.7451 - accuracy: 0.2214
Epoch 18/100
12/12 [==============================] - 0s 10ms/step - loss: 2.7079 - accuracy: 0.2240
Epoch 19/100
12/12 [==============================] - 0s 10ms/step - loss: 2.6830 - accuracy: 0.2240
Epoch 20/100
12/12 [==============================] - 0s 10ms/step - loss: 2.6664 - accuracy: 0.2266
Epoch 21/100
12/12 [==============================] - 0s 9ms/step - loss: 2.6204 - accuracy: 0.2422
Epoch 22/100
12/12 [==============================] - 0s 9ms/step - loss: 2.5803 - accuracy: 0.2500
Epoch 23/100
```

```
Epoch 23/100
12/12 [==============================] - 0s 10ms/step - loss: 2.5556 - accuracy: 0.2474
Epoch 24/100
12/12 [==============================] - 0s 10ms/step - loss: 2.5200 - accuracy: 0.2734
Epoch 25/100
12/12 [==============================] - 0s 8ms/step - loss: 2.4834 - accuracy: 0.2839
Epoch 26/100
12/12 [==============================] - 0s 9ms/step - loss: 2.4554 - accuracy: 0.2839
Epoch 27/100
12/12 [==============================] - 0s 9ms/step - loss: 2.4317 - accuracy: 0.3255
Epoch 28/100
12/12 [==============================] - 0s 11ms/step - loss: 2.4023 - accuracy: 0.3229
Epoch 29/100
12/12 [==============================] - 0s 10ms/step - loss: 2.3609 - accuracy: 0.3047
Epoch 30/100
12/12 [==============================] - 0s 9ms/step - loss: 2.3194 - accuracy: 0.3516
Epoch 31/100
12/12 [==============================] - 0s 9ms/step - loss: 2.2714 - accuracy: 0.3906
Epoch 32/100
12/12 [==============================] - 0s 9ms/step - loss: 2.2311 - accuracy: 0.3880
Epoch 33/100
12/12 [==============================] - 0s 9ms/step - loss: 2.2113 - accuracy: 0.3828
Epoch 34/100
12/12 [==============================] - 0s 9ms/step - loss: 2.1923 - accuracy: 0.3646
Epoch 35/100
12/12 [==============================] - 0s 10ms/step - loss: 2.1528 - accuracy: 0.4089
Epoch 36/100
12/12 [==============================] - 0s 10ms/step - loss: 2.0957 - accuracy: 0.4089
Epoch 37/100
12/12 [==============================] - 0s 9ms/step - loss: 2.0505 - accuracy: 0.4193
Epoch 38/100
12/12 [==============================] - 0s 9ms/step - loss: 1.9940 - accuracy: 0.4714
Epoch 39/100
12/12 [==============================] - 0s 9ms/step - loss: 1.9763 - accuracy: 0.4453
Epoch 40/100
12/12 [==============================] - 0s 9ms/step - loss: 1.9372 - accuracy: 0.4479
Epoch 41/100
12/12 [==============================] - 0s 9ms/step - loss: 1.9025 - accuracy: 0.4948
Epoch 42/100
12/12 [==============================] - 0s 9ms/step - loss: 1.8712 - accuracy: 0.4792
Epoch 43/100
12/12 [==============================] - 0s 9ms/step - loss: 1.8218 - accuracy: 0.4818
Epoch 44/100
12/12 [==============================] - 0s 9ms/step - loss: 1.7800 - accuracy: 0.5234
Epoch 45/100
12/12 [==============================] - 0s 11ms/step - loss: 1.7361 - accuracy: 0.5182
Epoch 46/100
12/12 [==============================] - 0s 9ms/step - loss: 1.7022 - accuracy: 0.5443
Epoch 47/100
12/12 [==============================] - 0s 9ms/step - loss: 1.6581 - accuracy: 0.5234
Epoch 48/100
12/12 [==============================] - 0s 9ms/step - loss: 1.6165 - accuracy: 0.5677
Epoch 49/100
12/12 [==============================] - 0s 9ms/step - loss: 1.5839 - accuracy: 0.5781
Epoch 50/100
12/12 [==============================] - 0s 10ms/step - loss: 1.5452 - accuracy: 0.5911
Epoch 51/100
12/12 [==============================] - 0s 9ms/step - loss: 1.5372 - accuracy: 0.5938
Epoch 52/100
12/12 [==============================] - 0s 9ms/step - loss: 1.4767 - accuracy: 0.6042
Epoch 53/100
12/12 [==============================] - 0s 10ms/step - loss: 1.4423 - accuracy: 0.6224
Epoch 54/100
12/12 [==============================] - 0s 8ms/step - loss: 1.4146 - accuracy: 0.6536
Epoch 55/100
12/12 [==============================] - 0s 12ms/step - loss: 1.3601 - accuracy: 0.6536
Epoch 56/100
12/12 [==============================] - 0s 9ms/step - loss: 1.3471 - accuracy: 0.6615
Epoch 57/100
12/12 [==============================] - 0s 9ms/step - loss: 1.2864 - accuracy: 0.6771
Epoch 58/100
12/12 [==============================] - 0s 9ms/step - loss: 1.2410 - accuracy: 0.6927
Epoch 59/100
```

```
Epoch 59/100
12/12 [==============================] - 0s 9ms/step - loss: 1.2338 - accuracy: 0.6953
Epoch 60/100
12/12 [==============================] - 0s 8ms/step - loss: 1.1981 - accuracy: 0.7292
Epoch 61/100
12/12 [==============================] - 0s 10ms/step - loss: 1.1700 - accuracy: 0.7318
Epoch 62/100
12/12 [==============================] - 0s 8ms/step - loss: 1.1216 - accuracy: 0.7448
Epoch 63/100
12/12 [==============================] - 0s 9ms/step - loss: 1.1217 - accuracy: 0.7526
Epoch 64/100
12/12 [==============================] - 0s 8ms/step - loss: 1.1066 - accuracy: 0.7552
Epoch 65/100
12/12 [==============================] - 0s 8ms/step - loss: 1.0301 - accuracy: 0.7812
Epoch 66/100
12/12 [==============================] - 0s 11ms/step - loss: 0.9902 - accuracy: 0.7969
Epoch 67/100
12/12 [==============================] - 0s 9ms/step - loss: 0.9604 - accuracy: 0.8177
Epoch 68/100
12/12 [==============================] - 0s 9ms/step - loss: 0.9278 - accuracy: 0.8411
Epoch 69/100
12/12 [==============================] - 0s 8ms/step - loss: 0.8791 - accuracy: 0.8464
Epoch 70/100
12/12 [==============================] - 0s 9ms/step - loss: 0.8443 - accuracy: 0.8490
Epoch 71/100
12/12 [==============================] - 0s 9ms/step - loss: 0.8281 - accuracy: 0.8620
Epoch 72/100
12/12 [==============================] - 0s 9ms/step - loss: 0.7964 - accuracy: 0.8802
Epoch 73/100
12/12 [==============================] - 0s 9ms/step - loss: 0.7528 - accuracy: 0.8984
Epoch 74/100
12/12 [==============================] - 0s 9ms/step - loss: 0.7379 - accuracy: 0.8932
Epoch 75/100
12/12 [==============================] - 0s 9ms/step - loss: 0.7126 - accuracy: 0.9036
Epoch 76/100
12/12 [==============================] - 0s 9ms/step - loss: 0.6787 - accuracy: 0.9062
Epoch 77/100
12/12 [==============================] - 0s 10ms/step - loss: 0.6539 - accuracy: 0.9193
Epoch 78/100
12/12 [==============================] - 0s 10ms/step - loss: 0.6285 - accuracy: 0.9349
Epoch 79/100
12/12 [==============================] - 0s 10ms/step - loss: 0.6047 - accuracy: 0.9375
Epoch 80/100
12/12 [==============================] - 0s 9ms/step - loss: 0.5826 - accuracy: 0.9453
Epoch 81/100
12/12 [==============================] - 0s 9ms/step - loss: 0.5682 - accuracy: 0.9531
Epoch 82/100
12/12 [==============================] - 0s 9ms/step - loss: 0.5364 - accuracy: 0.9531
Epoch 83/100
12/12 [==============================] - 0s 8ms/step - loss: 0.5210 - accuracy: 0.9531
Epoch 84/100
12/12 [==============================] - 0s 9ms/step - loss: 0.5023 - accuracy: 0.9479
Epoch 85/100
12/12 [==============================] - 0s 9ms/step - loss: 0.4836 - accuracy: 0.9714
Epoch 86/100
12/12 [==============================] - 0s 9ms/step - loss: 0.4749 - accuracy: 0.9557
Epoch 87/100
12/12 [==============================] - 0s 10ms/step - loss: 0.4575 - accuracy: 0.9661
Epoch 88/100
12/12 [==============================] - 0s 10ms/step - loss: 0.4398 - accuracy: 0.9688
Epoch 89/100
12/12 [==============================] - 0s 9ms/step - loss: 0.4196 - accuracy: 0.9766
Epoch 90/100
12/12 [==============================] - 0s 9ms/step - loss: 0.3970 - accuracy: 0.9714
Epoch 91/100
12/12 [==============================] - 0s 8ms/step - loss: 0.3866 - accuracy: 0.9766
Epoch 92/100
12/12 [==============================] - 0s 9ms/step - loss: 0.3654 - accuracy: 0.9792
Epoch 93/100
12/12 [==============================] - 0s 8ms/step - loss: 0.3475 - accuracy: 0.9766
Epoch 94/100
12/12 [==============================] - 0s 9ms/step - loss: 0.3336 - accuracy: 0.9818
Epoch 95/100
```

```
Epoch 95/100
12/12 [==============================] - 0s 9ms/step - loss: 0.3272 - accuracy: 0.9818
Epoch 96/100
12/12 [==============================] - 0s 9ms/step - loss: 0.3148 - accuracy: 0.9766
Epoch 97/100
12/12 [==============================] - 0s 11ms/step - loss: 0.2990 - accuracy: 0.9792
Epoch 98/100
12/12 [==============================] - 0s 9ms/step - loss: 0.2860 - accuracy: 0.9792
Epoch 99/100
12/12 [==============================] - 0s 9ms/step - loss: 0.2759 - accuracy: 0.9844
Epoch 100/100
12/12 [==============================] - 0s 9ms/step - loss: 0.2705 - accuracy: 0.9870
```

In [ ]:

```
# save the model to file
model.save('model.h5')
# save the mapping
dump(mapping, open('mapping.pkl', 'wb'))
```

Generating Text

We must provide sequences of 10 characters as input to the model in order to start the generation process. We will pick these manually. A given input sequence will need to be prepared in the same way as preparing the training data for the model.

In [ ]:

```
from pickle import load
import numpy as np
from keras.models import load_model
from tensorflow.keras.utils import to_categorical
from keras.preprocessing.sequence import pad_sequences

# generate a sequence of characters with a language model
def generate_seq(model, mapping, seq_length, seed_text, n_chars):
    in_text = seed_text
    # generate a fixed number of characters
    for _ in range(n_chars):
        # encode the characters as integers
        encoded = [mapping[char] for char in in_text]
        # truncate sequences to a fixed length
        encoded = pad_sequences([encoded], maxlen=seq_length, truncating='pre')
        # one hot encode
        encoded = to_categorical(encoded, num_classes=len(mapping))
        # predict character
        yhat = np.argmax(model.predict(encoded), axis=-1)
        # reverse map integer to character
        out_char = ''
        for char, index in mapping.items():
            if index == yhat:
                out_char = char
                break
        # append to input
        in_text += char
    return in_text

# load the model
model = load_model('model.h5')
# load the mapping
mapping = load(open('mapping.pkl', 'rb'))
```

Running the example generates three sequences of text.

The first is a test to see how the model does at starting from the beginning of the rhyme. The second is a test to see how well it does at beginning in the middle of a line. The final example is a test to see how well it does with a sequence of characters never seen before.

In [ ]:

```
# test start of rhyme
print(generate_seq(model, mapping, 10, 'Sing a son', 20))
# test mid-line
print(generate_seq(model, mapping, 10, 'king was i', 20))
# test not in original
print(generate_seq(model, mapping, 10, 'hello worl', 20))
```

```
Sing a song of sixpence,A pock
king was in his counting house
hello worle.Ta kkig  baiain aa
```

If the results aren't satisfactory, try out the suggestions above or these below:

- **Padding. Update the example to provides sequences line by line only and use padding to fill out each sequence to the maximum line length.**
- **Sequence Length. Experiment with different sequence lengths and see how they impact the behavior of the model.**
- **Tune Model. Experiment with different model configurations, such as the number of memory cells and epochs, and try to develop a better model for fewer resources.**

Deliverables to receive credit

1. **(4 points) Optimize the cells above to tune the model so that it generates text that closely resembles the orginal line from the rhyme, or at least generates sensible words. It's okay if the third example using unseen text still looks somewhat strange though. Again, this is a toy problem, as language models require a lot of computation. This toy problem is great for rapid experimentation to explore different aspects of deep learning language models.**
2. **(3 points) Write a function to split the text corpus file into training and validation and pipe the validation data into the model.fit() function to be able to track validation error per epoch. Lookup Keras documentation to see how this is handled.**
3. **(3 points) Write a summary (methods and results) in the cells below of the different things you applied. You must include your intuitions behind what did work and what did not work well.**
4. **(Extra Credit 2.5 points) Do something even more interesting. Try a different source text. Train a word-level model. We'll leave it up to your creativity to explore and write a summary of your methods and results.**

1: Optimize the cells above to tune the model...

In [ ]:

```
#load text
raw_text = load_doc('rhymes.txt')
print(raw_text)

# clean
tokens = raw_text.split()
raw_text = ' '.join(tokens)

# organize into sequences of characters
length = 15
sequences = list()
for i in range(length, len(raw_text)):
    # select sequence of tokens
    seq = raw_text[i-length:i+1]
    # store
    sequences.append(seq)
print('Total Sequences: %d' % len(sequences))
```

```
Sing a song of sixpence,A pocket full of rye.Four and twenty blackbirds,Baked in a pie.Wh
en the pie was openedThe birds began to sing;Wasn't that a dainty dish,To set before the
king.The king was in his counting house,Counting out his money;The queen was in the parlo
ur,Eating bread and honey.The maid was in the garden,Hanging out the clothes,When down ca
me a blackbirdAnd pecked off her nose.
```

Total Sequences: 379

In [ ]:

```
# save sequences to file
out_filename = 'char_sequences.txt'
save_doc(sequences, out_filename)
```

In [ ]:

```
# load doc into memory
def load_doc(filename):
    # open the file as read only
    file = open(filename, 'r')
    # read all text
    text = file.read()
    # close the file
    file.close()
    return text
```

In [ ]:

```
# load

in_filename = 'char_sequences.txt'
raw_text = load_doc(in_filename)
lines = raw_text.split('\n')
```

In [ ]:

```
# integer encode sequences of characters
chars = sorted(list(set(raw_text)))
mapping = dict((c, i) for i, c in enumerate(chars))
sequences = list()
for line in lines:
    # integer encode line
    encoded_seq = [mapping[char] for char in line]
    # store
    sequences.append(encoded_seq)

# vocabulary size
vocab_size = len(mapping)
print('Vocabulary Size: %d' % vocab_size)

# separate into input and output
sequences = array(sequences)
X, y = sequences[:,:-1], sequences[:,-1]
sequences = [to_categorical(x, num_classes=vocab_size) for x in X]
X = array(sequences)
y = to_categorical(y, num_classes=vocab_size)
```

Vocabulary Size: 38

In [ ]:

```
from keras.layers import Dropout
from keras.layers import ConvLSTM2D
from keras.layers import SimpleRNN

# define model
model = Sequential()
model.add(LSTM(150, input_shape=(X.shape[1], X.shape[2])))
model.add(Dense(vocab_size, activation='softmax'))
model.add(Dropout(0.1, input_shape=(vocab_size,)))


print(model.summary())
# compile model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
# fit model
history=model.fit(X, y, epochs=100, steps_per_epoch=20)
```

```
Model: "sequential_14"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 lstm_14 (LSTM)              (None, 150)               113400    
                                                                 
 dense_18 (Dense)            (None, 38)                5738      
                                                                 
 dropout_13 (Dropout)        (None, 38)                0         
                                                                 
=================================================================
Total params: 119,138
Trainable params: 119,138
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/100
20/20 [==============================] - 3s 23ms/step - loss: 4.8076 - accuracy: 0.1266
Epoch 2/100
20/20 [==============================] - 0s 22ms/step - loss: 4.4219 - accuracy: 0.1187
Epoch 3/100
20/20 [==============================] - 0s 23ms/step - loss: 4.1473 - accuracy: 0.1451
Epoch 4/100
20/20 [==============================] - 0s 22ms/step - loss: 4.2519 - accuracy: 0.1504
Epoch 5/100
20/20 [==============================] - 0s 23ms/step - loss: 4.3212 - accuracy: 0.1425
Epoch 6/100
20/20 [==============================] - 0s 22ms/step - loss: 4.2041 - accuracy: 0.1425
Epoch 7/100
20/20 [==============================] - 0s 24ms/step - loss: 4.2980 - accuracy: 0.1636
Epoch 8/100
20/20 [==============================] - 0s 22ms/step - loss: 4.2777 - accuracy: 0.1451
Epoch 9/100
20/20 [==============================] - 0s 23ms/step - loss: 4.2267 - accuracy: 0.1768
Epoch 10/100
20/20 [==============================] - 0s 22ms/step - loss: 4.1099 - accuracy: 0.1715
Epoch 11/100
20/20 [==============================] - 0s 22ms/step - loss: 4.2725 - accuracy: 0.1741
Epoch 12/100
20/20 [==============================] - 0s 23ms/step - loss: 3.8055 - accuracy: 0.2058
Epoch 13/100
20/20 [==============================] - 0s 22ms/step - loss: 3.7130 - accuracy: 0.1900
Epoch 14/100
20/20 [==============================] - 0s 23ms/step - loss: 3.8985 - accuracy: 0.2032
Epoch 15/100
20/20 [==============================] - 0s 22ms/step - loss: 3.7334 - accuracy: 0.2322
Epoch 16/100
20/20 [==============================] - 0s 23ms/step - loss: 3.6196 - accuracy: 0.2665
Epoch 17/100
20/20 [==============================] - 0s 22ms/step - loss: 3.4792 - accuracy: 0.2533
Epoch 18/100
20/20 [==============================] - 0s 23ms/step - loss: 3.7559 - accuracy: 0.2612
Epoch 19/100
20/20 [==============================] - 0s 22ms/step - loss: 3.8430 - accuracy: 0.2612
Epoch 20/100
20/20 [==============================] - 0s 24ms/step - loss: 3.7944 - accuracy: 0.2982
Epoch 21/100
20/20 [==============================] - 0s 23ms/step - loss: 3.3796 - accuracy: 0.3193
Epoch 22/100
20/20 [==============================] - 0s 23ms/step - loss: 3.3985 - accuracy: 0.3456
Epoch 23/100
20/20 [==============================] - 0s 23ms/step - loss: 3.5367 - accuracy: 0.3509
Epoch 24/100
20/20 [==============================] - 0s 23ms/step - loss: 3.4953 - accuracy: 0.3615
Epoch 25/100
20/20 [==============================] - 0s 23ms/step - loss: 3.1808 - accuracy: 0.4063
Epoch 26/100
20/20 [==============================] - 0s 23ms/step - loss: 3.1002 - accuracy: 0.4248
Epoch 27/100
20/20 [==============================] - 0s 23ms/step - loss: 3.5566 - accuracy: 0.4142
Epoch 28/100
```

```
20/20 [==============================] - 0s 23ms/step - loss: 2.9399 - accuracy: 0.4617
Epoch 29/100
20/20 [==============================] - 0s 24ms/step - loss: 3.1106 - accuracy: 0.4697
Epoch 30/100
20/20 [==============================] - 0s 22ms/step - loss: 2.8178 - accuracy: 0.5224
Epoch 31/100
20/20 [==============================] - 0s 24ms/step - loss: 2.9597 - accuracy: 0.5409
Epoch 32/100
20/20 [==============================] - 0s 22ms/step - loss: 2.3912 - accuracy: 0.5752
Epoch 33/100
20/20 [==============================] - 0s 24ms/step - loss: 2.6414 - accuracy: 0.5620
Epoch 34/100
20/20 [==============================] - 0s 23ms/step - loss: 2.9918 - accuracy: 0.5910
Epoch 35/100
20/20 [==============================] - 0s 22ms/step - loss: 2.8407 - accuracy: 0.6280
Epoch 36/100
20/20 [==============================] - 0s 24ms/step - loss: 2.6882 - accuracy: 0.6438
Epoch 37/100
20/20 [==============================] - 0s 23ms/step - loss: 2.5275 - accuracy: 0.6728
Epoch 38/100
20/20 [==============================] - 0s 23ms/step - loss: 2.5365 - accuracy: 0.6939
Epoch 39/100
20/20 [==============================] - 0s 23ms/step - loss: 2.1263 - accuracy: 0.7045
Epoch 40/100
20/20 [==============================] - 0s 24ms/step - loss: 2.5610 - accuracy: 0.7203
Epoch 41/100
20/20 [==============================] - 0s 22ms/step - loss: 2.1402 - accuracy: 0.7414
Epoch 42/100
20/20 [==============================] - 0s 23ms/step - loss: 2.1953 - accuracy: 0.7520
Epoch 43/100
20/20 [==============================] - 0s 23ms/step - loss: 2.0280 - accuracy: 0.8127
Epoch 44/100
20/20 [==============================] - 0s 24ms/step - loss: 1.7919 - accuracy: 0.8707
Epoch 45/100
20/20 [==============================] - 0s 22ms/step - loss: 1.9407 - accuracy: 0.8364
Epoch 46/100
20/20 [==============================] - 0s 23ms/step - loss: 1.8282 - accuracy: 0.8602
Epoch 47/100
20/20 [==============================] - 0s 22ms/step - loss: 2.5649 - accuracy: 0.8285
Epoch 48/100
20/20 [==============================] - 0s 22ms/step - loss: 2.1973 - accuracy: 0.8496
Epoch 49/100
20/20 [==============================] - 0s 24ms/step - loss: 1.7562 - accuracy: 0.8707
Epoch 50/100
20/20 [==============================] - 0s 22ms/step - loss: 1.8785 - accuracy: 0.8654
Epoch 51/100
20/20 [==============================] - 0s 24ms/step - loss: 2.2490 - accuracy: 0.8496
Epoch 52/100
20/20 [==============================] - 0s 23ms/step - loss: 1.6284 - accuracy: 0.8945
Epoch 53/100
20/20 [==============================] - 0s 24ms/step - loss: 1.8458 - accuracy: 0.8839
Epoch 54/100
20/20 [==============================] - 0s 24ms/step - loss: 1.6240 - accuracy: 0.8971
Epoch 55/100
20/20 [==============================] - 0s 24ms/step - loss: 1.8156 - accuracy: 0.8918
Epoch 56/100
20/20 [==============================] - 0s 22ms/step - loss: 1.7938 - accuracy: 0.8945
Epoch 57/100
20/20 [==============================] - 0s 23ms/step - loss: 1.6943 - accuracy: 0.8997
Epoch 58/100
20/20 [==============================] - 0s 23ms/step - loss: 1.7234 - accuracy: 0.8945
Epoch 59/100
20/20 [==============================] - 0s 23ms/step - loss: 1.6719 - accuracy: 0.8997
Epoch 60/100
20/20 [==============================] - 0s 23ms/step - loss: 1.8248 - accuracy: 0.8918
Epoch 61/100
20/20 [==============================] - 0s 23ms/step - loss: 1.3472 - accuracy: 0.9208
Epoch 62/100
20/20 [==============================] - 0s 22ms/step - loss: 1.5877 - accuracy: 0.9050
Epoch 63/100
20/20 [==============================] - 0s 23ms/step - loss: 1.7987 - accuracy: 0.8918
Epoch 64/100
```

```
20/20 [==============================] - 0s 24ms/step - loss: 1.9122 - accuracy: 0.8865
Epoch 65/100
20/20 [==============================] - 0s 23ms/step - loss: 1.2683 - accuracy: 0.9261
Epoch 66/100
20/20 [==============================] - 0s 23ms/step - loss: 1.7730 - accuracy: 0.8918
Epoch 67/100
20/20 [==============================] - 0s 23ms/step - loss: 1.7684 - accuracy: 0.8918
Epoch 68/100
20/20 [==============================] - 0s 24ms/step - loss: 1.6352 - accuracy: 0.9024
Epoch 69/100
20/20 [==============================] - 0s 23ms/step - loss: 1.3376 - accuracy: 0.9182
Epoch 70/100
20/20 [==============================] - 0s 22ms/step - loss: 1.7574 - accuracy: 0.8892
Epoch 71/100
20/20 [==============================] - 0s 24ms/step - loss: 1.4977 - accuracy: 0.9103
Epoch 72/100
20/20 [==============================] - 0s 23ms/step - loss: 1.8384 - accuracy: 0.8892
Epoch 73/100
20/20 [==============================] - 1s 25ms/step - loss: 1.6291 - accuracy: 0.8971
Epoch 74/100
20/20 [==============================] - 0s 23ms/step - loss: 1.7076 - accuracy: 0.8945
Epoch 75/100
20/20 [==============================] - 0s 24ms/step - loss: 1.5358 - accuracy: 0.9050
Epoch 76/100
20/20 [==============================] - 0s 22ms/step - loss: 1.4022 - accuracy: 0.9156
Epoch 77/100
20/20 [==============================] - 0s 24ms/step - loss: 1.8258 - accuracy: 0.8865
Epoch 78/100
20/20 [==============================] - 0s 22ms/step - loss: 2.2090 - accuracy: 0.8628
Epoch 79/100
20/20 [==============================] - 0s 23ms/step - loss: 1.6100 - accuracy: 0.9024
Epoch 80/100
20/20 [==============================] - 0s 22ms/step - loss: 1.3937 - accuracy: 0.9156
Epoch 81/100
20/20 [==============================] - 0s 23ms/step - loss: 2.2816 - accuracy: 0.8602
Epoch 82/100
20/20 [==============================] - 0s 23ms/step - loss: 1.6016 - accuracy: 0.9024
Epoch 83/100
20/20 [==============================] - 0s 23ms/step - loss: 1.3953 - accuracy: 0.9103
Epoch 84/100
20/20 [==============================] - 0s 23ms/step - loss: 1.3937 - accuracy: 0.9129
Epoch 85/100
20/20 [==============================] - 0s 22ms/step - loss: 1.7286 - accuracy: 0.8945
Epoch 86/100
20/20 [==============================] - 0s 25ms/step - loss: 1.3450 - accuracy: 0.9182
Epoch 87/100
20/20 [==============================] - 0s 23ms/step - loss: 1.3879 - accuracy: 0.9129
Epoch 88/100
20/20 [==============================] - 0s 23ms/step - loss: 1.4729 - accuracy: 0.9077
Epoch 89/100
20/20 [==============================] - 0s 24ms/step - loss: 1.4292 - accuracy: 0.9103
Epoch 90/100
20/20 [==============================] - 0s 23ms/step - loss: 1.3850 - accuracy: 0.9129
Epoch 91/100
20/20 [==============================] - 0s 22ms/step - loss: 1.5953 - accuracy: 0.9024
Epoch 92/100
20/20 [==============================] - 0s 23ms/step - loss: 1.1692 - accuracy: 0.9288
Epoch 93/100
20/20 [==============================] - 0s 23ms/step - loss: 1.7618 - accuracy: 0.8918
Epoch 94/100
20/20 [==============================] - 0s 23ms/step - loss: 1.5936 - accuracy: 0.8997
Epoch 95/100
20/20 [==============================] - 0s 23ms/step - loss: 1.6762 - accuracy: 0.8971
Epoch 96/100
20/20 [==============================] - 0s 22ms/step - loss: 1.8476 - accuracy: 0.8839
Epoch 97/100
20/20 [==============================] - 0s 23ms/step - loss: 1.5928 - accuracy: 0.8997
Epoch 98/100
20/20 [==============================] - 0s 22ms/step - loss: 1.5507 - accuracy: 0.9050
Epoch 99/100
20/20 [==============================] - 0s 24ms/step - loss: 1.3345 - accuracy: 0.9182
Epoch 100/100
```

```
20/20 [==============================] - 0s 23ms/step - loss: 1.7231 - accuracy: 0.8918
```

In [ ]:

```
# generate a sequence of characters with a language model
def generate_seq(model, mapping, seq_length, seed_text, n_chars):
    in_text = seed_text
    # generate a fixed number of characters
    for _ in range(n_chars):
        # encode the characters as integers
        encoded = [mapping[char] for char in in_text]
        # truncate sequences to a fixed length
        encoded = pad_sequences([encoded], maxlen=seq_length, truncating='pre')
        # one hot encode
        encoded = to_categorical(encoded, num_classes=len(mapping))
        # predict character
        yhat = np.argmax(model.predict(encoded), axis=-1)
        # reverse map integer to character
        out_char = ''
        for char, index in mapping.items():
            if index == yhat:
                out_char = char
                break
        # append to input
        in_text += char
    return in_text

# load the model
model = load_model('model.h5')
# load the mapping
mapping = load(open('mapping.pkl', 'rb'))
```

In [ ]:

```
# test start of rhyme
print(generate_seq(model, mapping, 10, 'Sing a son', 20))
# test mid-line
print(generate_seq(model, mapping, 10, 'king was i', 20))
# test not in original
print(generate_seq(model, mapping, 10, 'hello worl', 20))
```

```
Sing a song of sixpence,A pock
king was in his counting house
hello worle.Ta kkig  baiain aa
```

2: Write a function to split the text corpus file...

In [ ]:

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42
)

# define model
model = Sequential()
model.add(LSTM(150, input_shape=(X.shape[1], X.shape[2])))
model.add(Dense(vocab_size, activation='softmax'))
model.add(Dropout(0.1, input_shape=(vocab_size,)))


print(model.summary())
# compile model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
# fit model
history=model.fit(X_train, y_train, epochs=100, steps_per_epoch=20, validation_data=(X_t
est, y_test))
```

```
Model: "sequential_10"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
```

```
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_10 (LSTM)              (None, 150)               113400

 dense_14 (Dense)            (None, 38)                5738

 dropout_9 (Dropout)         (None, 38)                0

=================================================================
Total params: 119,138
Trainable params: 119,138
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/100
20/20 [==============================] - 3s 58ms/step - loss: 4.8696 - accuracy: 0.0681 -
val_loss: 3.3313 - val_accuracy: 0.1491
Epoch 2/100
20/20 [==============================] - 0s 25ms/step - loss: 4.6403 - accuracy: 0.1541 -
val_loss: 3.2132 - val_accuracy: 0.1579
Epoch 3/100
20/20 [==============================] - 0s 24ms/step - loss: 4.3410 - accuracy: 0.1541 -
val_loss: 3.2584 - val_accuracy: 0.1579
Epoch 4/100
20/20 [==============================] - 1s 26ms/step - loss: 4.5828 - accuracy: 0.1541 -
val_loss: 3.2536 - val_accuracy: 0.1491
Epoch 5/100
20/20 [==============================] - 1s 26ms/step - loss: 4.2775 - accuracy: 0.1254 -
val_loss: 3.2463 - val_accuracy: 0.1667
Epoch 6/100
20/20 [==============================] - 1s 25ms/step - loss: 3.6711 - accuracy: 0.2007 -
val_loss: 3.2488 - val_accuracy: 0.1491
Epoch 7/100
20/20 [==============================] - 0s 24ms/step - loss: 4.5480 - accuracy: 0.1434 -
val_loss: 3.3250 - val_accuracy: 0.1579
Epoch 8/100
20/20 [==============================] - 0s 24ms/step - loss: 4.2995 - accuracy: 0.1935 -
val_loss: 3.2634 - val_accuracy: 0.1667
Epoch 9/100
20/20 [==============================] - 1s 25ms/step - loss: 3.9770 - accuracy: 0.1971 -
val_loss: 3.3805 - val_accuracy: 0.1053
Epoch 10/100
20/20 [==============================] - 0s 25ms/step - loss: 4.2511 - accuracy: 0.1756 -
val_loss: 3.3000 - val_accuracy: 0.1491
Epoch 11/100
20/20 [==============================] - 1s 26ms/step - loss: 4.0386 - accuracy: 0.2294 -
val_loss: 3.4007 - val_accuracy: 0.1491
Epoch 12/100
20/20 [==============================] - 0s 25ms/step - loss: 3.8545 - accuracy: 0.2115 -
val_loss: 3.4970 - val_accuracy: 0.1140
Epoch 13/100
20/20 [==============================] - 1s 25ms/step - loss: 3.9079 - accuracy: 0.2186 -
val_loss: 3.4873 - val_accuracy: 0.1491
Epoch 14/100
20/20 [==============================] - 0s 25ms/step - loss: 3.7011 - accuracy: 0.2186 -
val_loss: 3.4795 - val_accuracy: 0.0877
Epoch 15/100
20/20 [==============================] - 1s 25ms/step - loss: 3.4229 - accuracy: 0.2652 -
val_loss: 3.5910 - val_accuracy: 0.1228
Epoch 16/100
20/20 [==============================] - 1s 26ms/step - loss: 3.9932 - accuracy: 0.2760 -
val_loss: 3.6192 - val_accuracy: 0.1404
Epoch 17/100
20/20 [==============================] - 1s 25ms/step - loss: 3.7135 - accuracy: 0.2509 -
val_loss: 3.6227 - val_accuracy: 0.1316
Epoch 18/100
20/20 [==============================] - 0s 24ms/step - loss: 3.3204 - accuracy: 0.3118 -
val_loss: 3.6799 - val_accuracy: 0.1316
Epoch 19/100
20/20 [==============================] - 1s 26ms/step - loss: 3.5336 - accuracy: 0.3741 -
val_loss: 3.5928 - val_accuracy: 0.1491
Epoch 20/100
20/20 [==============================] - 0s 25ms/step - loss: 3.7338 - accuracy: 0.3620 -
```

```
20/20 [==============================] - 0s 25ms/step - loss: 3.7338 - accuracy: 0.3620 -
val_loss: 3.6860 - val_accuracy: 0.1579
Epoch 21/100
20/20 [==============================] - 0s 25ms/step - loss: 3.0701 - accuracy: 0.4050 -
val_loss: 3.7599 - val_accuracy: 0.1404
Epoch 22/100
20/20 [==============================] - 1s 25ms/step - loss: 3.1472 - accuracy: 0.4014 -
val_loss: 3.7801 - val_accuracy: 0.1404
Epoch 23/100
20/20 [==============================] - 1s 25ms/step - loss: 3.4598 - accuracy: 0.4588 -
val_loss: 3.7873 - val_accuracy: 0.1667
Epoch 24/100
20/20 [==============================] - 1s 28ms/step - loss: 2.6421 - accuracy: 0.5412 -
val_loss: 4.0515 - val_accuracy: 0.1316
Epoch 25/100
20/20 [==============================] - 1s 26ms/step - loss: 2.6494 - accuracy: 0.5412 -
val_loss: 4.0119 - val_accuracy: 0.1140
Epoch 26/100
20/20 [==============================] - 1s 25ms/step - loss: 3.1870 - accuracy: 0.4695 -
val_loss: 4.0605 - val_accuracy: 0.1579
Epoch 27/100
20/20 [==============================] - 0s 25ms/step - loss: 3.4723 - accuracy: 0.4982 -
val_loss: 4.0656 - val_accuracy: 0.1579
Epoch 28/100
20/20 [==============================] - 0s 25ms/step - loss: 2.8921 - accuracy: 0.5448 -
val_loss: 3.9610 - val_accuracy: 0.1228
Epoch 29/100
20/20 [==============================] - 1s 26ms/step - loss: 2.3404 - accuracy: 0.6487 -
val_loss: 4.2434 - val_accuracy: 0.1228
Epoch 30/100
20/20 [==============================] - 1s 25ms/step - loss: 1.9558 - accuracy: 0.6344 -
val_loss: 4.2347 - val_accuracy: 0.1053
Epoch 31/100
20/20 [==============================] - 1s 27ms/step - loss: 2.2062 - accuracy: 0.7348 -
val_loss: 4.3512 - val_accuracy: 0.1316
Epoch 32/100
20/20 [==============================] - 1s 25ms/step - loss: 2.6343 - accuracy: 0.6810 -
val_loss: 4.3053 - val_accuracy: 0.1404
Epoch 33/100
20/20 [==============================] - 1s 26ms/step - loss: 2.3403 - accuracy: 0.7491 -
val_loss: 4.4757 - val_accuracy: 0.1228
Epoch 34/100
20/20 [==============================] - 0s 25ms/step - loss: 2.3622 - accuracy: 0.7204 -
val_loss: 4.4459 - val_accuracy: 0.1316
Epoch 35/100
20/20 [==============================] - 0s 25ms/step - loss: 2.0759 - accuracy: 0.7921 -
val_loss: 4.4489 - val_accuracy: 0.1140
Epoch 36/100
20/20 [==============================] - 0s 23ms/step - loss: 1.9383 - accuracy: 0.8280 -
val_loss: 4.5823 - val_accuracy: 0.1228
Epoch 37/100
20/20 [==============================] - 0s 25ms/step - loss: 2.3762 - accuracy: 0.7957 -
val_loss: 4.7483 - val_accuracy: 0.1053
Epoch 38/100
20/20 [==============================] - 0s 25ms/step - loss: 2.0129 - accuracy: 0.8345 -
val_loss: 4.6807 - val_accuracy: 0.1404
Epoch 39/100
20/20 [==============================] - 1s 26ms/step - loss: 2.0921 - accuracy: 0.8530 -
val_loss: 4.8073 - val_accuracy: 0.1228
Epoch 40/100
20/20 [==============================] - 1s 25ms/step - loss: 1.7267 - accuracy: 0.8853 -
val_loss: 4.8304 - val_accuracy: 0.0877
Epoch 41/100
20/20 [==============================] - 1s 25ms/step - loss: 1.7973 - accuracy: 0.8602 -
val_loss: 4.9012 - val_accuracy: 0.1053
Epoch 42/100
20/20 [==============================] - 0s 25ms/step - loss: 1.8630 - accuracy: 0.8853 -
val_loss: 4.9536 - val_accuracy: 0.0877
Epoch 43/100
20/20 [==============================] - 1s 26ms/step - loss: 1.1629 - accuracy: 0.9211 -
val_loss: 4.9562 - val_accuracy: 0.1316
Epoch 44/100
20/20 [==============================] - 1s 28ms/step - loss: 1.8695 - accuracy: 0.8746 -
```

```
20/20 [==============================] - 1s 28ms/step - loss: 1.8695 - accuracy: 0.8746 -
val_loss: 5.1130 - val_accuracy: 0.1228
Epoch 45/100
20/20 [==============================] - 1s 26ms/step - loss: 2.3336 - accuracy: 0.8459 -
val_loss: 5.0246 - val_accuracy: 0.1053
Epoch 46/100
20/20 [==============================] - 1s 25ms/step - loss: 1.8478 - accuracy: 0.8961 -
val_loss: 5.1904 - val_accuracy: 0.0877
Epoch 47/100
20/20 [==============================] - 1s 25ms/step - loss: 1.7162 - accuracy: 0.8925 -
val_loss: 5.2746 - val_accuracy: 0.1053
Epoch 48/100
20/20 [==============================] - 0s 25ms/step - loss: 1.5150 - accuracy: 0.9140 -
val_loss: 5.3200 - val_accuracy: 0.0965
Epoch 49/100
20/20 [==============================] - 1s 26ms/step - loss: 1.9023 - accuracy: 0.8853 -
val_loss: 5.3739 - val_accuracy: 0.1053
Epoch 50/100
20/20 [==============================] - 0s 25ms/step - loss: 1.8058 - accuracy: 0.8889 -
val_loss: 5.3525 - val_accuracy: 0.1140
Epoch 51/100
20/20 [==============================] - 1s 26ms/step - loss: 1.6556 - accuracy: 0.9032 -
val_loss: 5.4868 - val_accuracy: 0.1053
Epoch 52/100
20/20 [==============================] - 1s 25ms/step - loss: 1.6551 - accuracy: 0.8961 -
val_loss: 5.4667 - val_accuracy: 0.1053
Epoch 53/100
20/20 [==============================] - 1s 26ms/step - loss: 1.8669 - accuracy: 0.8889 -
val_loss: 5.5048 - val_accuracy: 0.1053
Epoch 54/100
20/20 [==============================] - 1s 25ms/step - loss: 1.6300 - accuracy: 0.9032 -
val_loss: 5.5350 - val_accuracy: 0.0965
Epoch 55/100
20/20 [==============================] - 1s 27ms/step - loss: 1.2756 - accuracy: 0.9247 -
val_loss: 5.5782 - val_accuracy: 0.1140
Epoch 56/100
20/20 [==============================] - 1s 26ms/step - loss: 1.5580 - accuracy: 0.9068 -
val_loss: 5.6316 - val_accuracy: 0.1140
Epoch 57/100
20/20 [==============================] - 1s 25ms/step - loss: 2.0837 - accuracy: 0.8741 -
val_loss: 5.6552 - val_accuracy: 0.0965
Epoch 58/100
20/20 [==============================] - 0s 25ms/step - loss: 2.3682 - accuracy: 0.8530 -
val_loss: 5.6414 - val_accuracy: 0.1140
Epoch 59/100
20/20 [==============================] - 1s 26ms/step - loss: 1.4377 - accuracy: 0.9104 -
val_loss: 5.7222 - val_accuracy: 0.1053
Epoch 60/100
20/20 [==============================] - 0s 24ms/step - loss: 1.4344 - accuracy: 0.9104 -
val_loss: 5.7265 - val_accuracy: 0.0965
Epoch 61/100
20/20 [==============================] - 0s 25ms/step - loss: 1.9540 - accuracy: 0.8781 -
val_loss: 5.8394 - val_accuracy: 0.1053
Epoch 62/100
20/20 [==============================] - 1s 26ms/step - loss: 1.2039 - accuracy: 0.9283 -
val_loss: 5.7857 - val_accuracy: 0.0877
Epoch 63/100
20/20 [==============================] - 0s 25ms/step - loss: 2.1164 - accuracy: 0.8710 -
val_loss: 5.8353 - val_accuracy: 0.1053
Epoch 64/100
20/20 [==============================] - 1s 27ms/step - loss: 1.4841 - accuracy: 0.9104 -
val_loss: 5.8332 - val_accuracy: 0.0965
Epoch 65/100
20/20 [==============================] - 1s 26ms/step - loss: 1.5440 - accuracy: 0.9032 -
val_loss: 5.8786 - val_accuracy: 0.0965
Epoch 66/100
20/20 [==============================] - 0s 24ms/step - loss: 1.5960 - accuracy: 0.9032 -
val_loss: 5.8509 - val_accuracy: 0.0965
Epoch 67/100
20/20 [==============================] - 0s 25ms/step - loss: 1.9369 - accuracy: 0.8817 -
val_loss: 5.9178 - val_accuracy: 0.1053
Epoch 68/100
20/20 [==============================] - 0s 24ms/step - loss: 1.4160 - accuracy: 0.9140 -
```

```
20/20 [==============================] - 0s 24ms/step - loss: 1.4160 - accuracy: 0.9140 -
val_loss: 5.9355 - val_accuracy: 0.0965
Epoch 69/100
20/20 [==============================] - 0s 24ms/step - loss: 1.8211 - accuracy: 0.8853 -
val_loss: 5.9586 - val_accuracy: 0.0965
Epoch 70/100
20/20 [==============================] - 1s 26ms/step - loss: 1.4765 - accuracy: 0.9104 -
val_loss: 5.9764 - val_accuracy: 0.1140
Epoch 71/100
20/20 [==============================] - 1s 26ms/step - loss: 1.2729 - accuracy: 0.8996 -
val_loss: 5.9621 - val_accuracy: 0.0702
Epoch 72/100
20/20 [==============================] - 1s 25ms/step - loss: 2.0259 - accuracy: 0.8065 -
val_loss: 5.6535 - val_accuracy: 0.1228
Epoch 73/100
20/20 [==============================] - 1s 25ms/step - loss: 1.8550 - accuracy: 0.7634 -
val_loss: 5.2384 - val_accuracy: 0.0965
Epoch 74/100
20/20 [==============================] - 1s 25ms/step - loss: 2.2148 - accuracy: 0.7849 -
val_loss: 5.6286 - val_accuracy: 0.0965
Epoch 75/100
20/20 [==============================] - 1s 26ms/step - loss: 1.1407 - accuracy: 0.9283 -
val_loss: 5.6868 - val_accuracy: 0.0965
Epoch 76/100
20/20 [==============================] - 0s 24ms/step - loss: 1.7821 - accuracy: 0.8849 -
val_loss: 5.6303 - val_accuracy: 0.1053
Epoch 77/100
20/20 [==============================] - 1s 27ms/step - loss: 1.5652 - accuracy: 0.9032 -
val_loss: 5.8255 - val_accuracy: 0.1053
Epoch 78/100
20/20 [==============================] - 0s 24ms/step - loss: 2.0620 - accuracy: 0.8746 -
val_loss: 5.9252 - val_accuracy: 0.1228
Epoch 79/100
20/20 [==============================] - 1s 26ms/step - loss: 1.4770 - accuracy: 0.9104 -
val_loss: 5.8695 - val_accuracy: 0.1053
Epoch 80/100
20/20 [==============================] - 0s 24ms/step - loss: 1.0072 - accuracy: 0.9391 -
val_loss: 5.9466 - val_accuracy: 0.1053
Epoch 81/100
20/20 [==============================] - 1s 26ms/step - loss: 1.6396 - accuracy: 0.8996 -
val_loss: 5.9881 - val_accuracy: 0.1053
Epoch 82/100
20/20 [==============================] - 1s 25ms/step - loss: 1.9256 - accuracy: 0.8817 -
val_loss: 6.0246 - val_accuracy: 0.1053
Epoch 83/100
20/20 [==============================] - 1s 25ms/step - loss: 1.8081 - accuracy: 0.8889 -
val_loss: 6.0665 - val_accuracy: 0.1053
Epoch 84/100
20/20 [==============================] - 1s 27ms/step - loss: 1.3465 - accuracy: 0.9176 -
val_loss: 6.0816 - val_accuracy: 0.1053
Epoch 85/100
20/20 [==============================] - 1s 26ms/step - loss: 2.2689 - accuracy: 0.8602 -
val_loss: 6.1057 - val_accuracy: 0.1053
Epoch 86/100
20/20 [==============================] - 1s 25ms/step - loss: 1.2301 - accuracy: 0.9247 -
val_loss: 6.1343 - val_accuracy: 0.1053
Epoch 87/100
20/20 [==============================] - 1s 26ms/step - loss: 1.5737 - accuracy: 0.9032 -
val_loss: 6.1578 - val_accuracy: 0.1053
Epoch 88/100
20/20 [==============================] - 1s 27ms/step - loss: 1.5745 - accuracy: 0.9032 -
val_loss: 6.1869 - val_accuracy: 0.1053
Epoch 89/100
20/20 [==============================] - 1s 26ms/step - loss: 1.4002 - accuracy: 0.9140 -
val_loss: 6.2114 - val_accuracy: 0.1053
Epoch 90/100
20/20 [==============================] - 1s 25ms/step - loss: 1.9191 - accuracy: 0.8817 -
val_loss: 6.2259 - val_accuracy: 0.1053
Epoch 91/100
20/20 [==============================] - 1s 26ms/step - loss: 1.0530 - accuracy: 0.9355 -
val_loss: 6.2455 - val_accuracy: 0.1053
Epoch 92/100
20/20 [==============================] - 1s 26ms/step - loss: 2.0332 - accuracy: 0.8746 -
```

```
20/20 [==============================] - 1s 26ms/step - loss: 2.0532 - accuracy: 0.8746 -
val_loss: 6.2576 - val_accuracy: 0.1053
Epoch 93/100
20/20 [==============================] - 1s 25ms/step - loss: 1.3407 - accuracy: 0.9176 -
val_loss: 6.2798 - val_accuracy: 0.1053
Epoch 94/100
20/20 [==============================] - 1s 27ms/step - loss: 2.0904 - accuracy: 0.8710 -
val_loss: 6.2967 - val_accuracy: 0.1053
Epoch 95/100
20/20 [==============================] - 1s 25ms/step - loss: 1.1126 - accuracy: 0.9317 -
val_loss: 6.3115 - val_accuracy: 0.1053
Epoch 96/100
WARNING:tensorflow:Your input ran out of data; interrupting training. Make sure that your
dataset or generator can generate at least `steps_per_epoch * epochs` batches (in this ca
se, 2000 batches). You may need to use the repeat() function when building your dataset.
20/20 [==============================] - 0s 7ms/step - loss: 1.1126 - accuracy: 0.9317 -
val_loss: 6.3115 - val_accuracy: 0.1053
```

In [ ]:

```python
# generate a sequence of characters with a language model
def generate_seq(model, mapping, seq_length, seed_text, n_chars):
    in_text = seed_text
    # generate a fixed number of characters
    for _ in range(n_chars):
        # encode the characters as integers
        encoded = [mapping[char] for char in in_text]
        # truncate sequences to a fixed length
        encoded = pad_sequences([encoded], maxlen=seq_length, truncating='pre')
        # one hot encode
        encoded = to_categorical(encoded, num_classes=len(mapping))
        # predict character
        yhat = np.argmax(model.predict(encoded), axis=-1)
        # reverse map integer to character
        out_char = ''
        for char, index in mapping.items():
            if index == yhat:
                out_char = char
                break
        # append to input
        in_text += char
    return in_text

# load the model
model = load_model('model.h5')
# load the mapping
mapping = load(open('mapping.pkl', 'rb'))
```

In [ ]:

```python
# test start of rhyme
print(generate_seq(model, mapping, 10, 'Sing a son', 20))
# test mid-line
print(generate_seq(model, mapping, 10, 'king was i', 20))
# test not in original
print(generate_seq(model, mapping, 10, 'hello worl', 20))
```

```
Sing a song of sixpence,A pock
king was in his counting house
hello worle.Ta kkig  baiain aa
```

3: Write a summary (methods and results)...

There are at least 5 notable adjustments that I made to our model to optimize accuracy and tune the model; the following are descriptions of ways that I adjusted my model:

1. **First decreased sequence length from 10 to 5, then increased to 15 and found best results.**
2. **Also decreased the number of layers in my LSTM from 75 to 50 but again found. better results when I increased the number of layers to 150.**

3. I made various model.add calls to try different layer types, namely ConvLSTM2D and SimpleRNN, but did not find any that made improvements.
4. Used dropout as a method of regularization.
5. Adjusted Epoch length from 12 down to 5, but again I saw improvements in my model when I increased it up to 20.

Adjusting the size of the LSTM layers and the epoch length were the two biggest impacts for boosting my accuracy. I also did notice that when I raised my sequence length of my chars, that while accuracy did increase the variance in accuracy also increased as well. My intuition behind why the variance also increased when I increased the sequence length is because the text corpus is so short so taking big sequences made less room for a poorly predicted sequence. Overall, I was not able to beat the accuracy of the original model (which was around 98-99%) but I was able to slowly boost the accuracy of this unqiue model.

4: EXTRA CREDIT: Try a different source text. Train a word-level model...

In [ ]:

```
# The text corpus I chose is the lyrics of "Feel Good Inc" by the Gorillaz

s = "City's breaking down on a camel's back\
They just have to go, 'cause they don't know wack\
So while you fill the streets, it's appealing to see\
You won't get out the county, 'cause you're bad and free\
You got a new horizon, it's ephemeral style\
A melancholy town where we never smile\
And all I wanna hear is the message beep\
My dreams, they got a kissing\
'Cause I don't get sleep, no\
Windmill, windmill for the land\
Turn forever hand in hand\
Take it all in on your stride\
It is ticking, falling down\
Love forever, love is freely\
Turned forever, you and me\
Windmill, windmill for the land\
Is everybody in?"

with open('rhymes.txt','w') as f:
  f.write(s)
```

In [ ]:

```
#load doc into memory
def load_doc(filename):
    # open the file as read only
    file = open(filename, 'r')
    # read all text
    text = file.read()
    # close the file
    file.close()
    return text
# save tokens to file, one dialog per line
def save_doc(lines, filename):
    data = '\n'.join(lines)
    file = open(filename, 'w')
    file.write(data)
    file.close()
```

In [ ]:

```
#load text
raw_text = load_doc('rhymes.txt')
print(raw_text)

# clean
tokens = raw_text.split()
```

```
raw_text = ' '.join(tokens)

# organize into sequences of characters
length = 10
sequences = list()
for i in range(length, len(raw_text)):
    # select sequence of tokens
    seq = raw_text[i-length:i+1]
    # store
    sequences.append(seq)
print('Total Sequences: %d' % len(sequences))
```

```
City's breaking down on a camel's backThey just have to go, 'cause they don't know wackSo
while you fill the streets, it's appealing to seeYou won't get out the county, 'cause you
're bad and freeYou got a new horizon, it's ephemeral styleA melancholy town where we nev
er smileAnd all I wanna hear is the message beepMy dreams, they got a kissing'Cause I don
't get sleep, noWindmill, windmill for the landTurn forever hand in handTake it all in on
your strideIt is ticking, falling downLove forever, love is freelyTurned forever, you and
meWindmill, windmill for the landIs everybody in?
Total Sequences: 576
```

In [ ]:

```
# save sequences to file
out_filename = 'char_sequences.txt'
save_doc(sequences, out_filename)
```

In [ ]:

```
# load doc into memory
def load_doc(filename):
    # open the file as read only
    file = open(filename, 'r')
    # read all text
    text = file.read()
    # close the file
    file.close()
    return text
```

In [ ]:

```
# load

in_filename = 'char_sequences.txt'
raw_text = load_doc(in_filename)
lines = raw_text.split('\n')
```

In [ ]:

```
# integer encode sequences of characters
chars = sorted(list(set(raw_text)))
mapping = dict((c, i) for i, c in enumerate(chars))
sequences = list()
for line in lines:
    # integer encode line
    encoded_seq = [mapping[char] for char in line]
    # store
    sequences.append(encoded_seq)

# vocabulary size
vocab_size = len(mapping)
print('Vocabulary Size: %d' % vocab_size)

# separate into input and output
sequences = array(sequences)
X, y = sequences[:,:-1], sequences[:,-1]
sequences = [to_categorical(x, num_classes=vocab_size) for x in X]
X = array(sequences)
y = to_categorical(y, num_classes=vocab_size)
```

```
Vocabulary Size: 38
```

In [ ]:

```
# define model
model = Sequential()
model.add(LSTM(75, input_shape=(X.shape[1], X.shape[2])))
model.add(Dense(vocab_size, activation='softmax'))
print(model.summary())
# compile model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
# fit model
history=model.fit(X, y, epochs=100)
```

```
Model: "sequential_15"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 lstm_15 (LSTM)              (None, 75)                34200     
                                                                 
 dense_19 (Dense)            (None, 38)                2888      
                                                                 
=================================================================
Total params: 37,088
Trainable params: 37,088
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/100
18/18 [==============================] - 2s 9ms/step - loss: 3.5848 - accuracy: 0.1302
Epoch 2/100
18/18 [==============================] - 0s 10ms/step - loss: 3.2835 - accuracy: 0.1771
Epoch 3/100
18/18 [==============================] - 0s 8ms/step - loss: 3.0903 - accuracy: 0.1667
Epoch 4/100
18/18 [==============================] - 0s 9ms/step - loss: 3.0527 - accuracy: 0.1667
Epoch 5/100
18/18 [==============================] - 0s 9ms/step - loss: 3.0202 - accuracy: 0.1667
Epoch 6/100
18/18 [==============================] - 0s 10ms/step - loss: 3.0003 - accuracy: 0.1667
Epoch 7/100
18/18 [==============================] - 0s 9ms/step - loss: 2.9790 - accuracy: 0.1927
Epoch 8/100
18/18 [==============================] - 0s 10ms/step - loss: 2.9565 - accuracy: 0.1927
Epoch 9/100
18/18 [==============================] - 0s 10ms/step - loss: 2.9273 - accuracy: 0.1997
Epoch 10/100
18/18 [==============================] - 0s 9ms/step - loss: 2.8949 - accuracy: 0.1997
Epoch 11/100
18/18 [==============================] - 0s 9ms/step - loss: 2.8633 - accuracy: 0.2118
Epoch 12/100
18/18 [==============================] - 0s 10ms/step - loss: 2.8319 - accuracy: 0.2240
Epoch 13/100
18/18 [==============================] - 0s 8ms/step - loss: 2.7952 - accuracy: 0.2240
Epoch 14/100
18/18 [==============================] - 0s 9ms/step - loss: 2.7768 - accuracy: 0.2344
Epoch 15/100
18/18 [==============================] - 0s 9ms/step - loss: 2.7366 - accuracy: 0.2413
Epoch 16/100
18/18 [==============================] - 0s 9ms/step - loss: 2.6841 - accuracy: 0.2674
Epoch 17/100
18/18 [==============================] - 0s 9ms/step - loss: 2.6508 - accuracy: 0.2552
Epoch 18/100
18/18 [==============================] - 0s 10ms/step - loss: 2.6141 - accuracy: 0.2830
Epoch 19/100
18/18 [==============================] - 0s 10ms/step - loss: 2.5597 - accuracy: 0.3056
Epoch 20/100
18/18 [==============================] - 0s 9ms/step - loss: 2.5318 - accuracy: 0.2882
Epoch 21/100
18/18 [==============================] - 0s 10ms/step - loss: 2.4817 - accuracy: 0.3003
Epoch 22/100
18/18 [==============================] - 0s 9ms/step - loss: 2.4551 - accuracy: 0.3160
Epoch 23/100
18/18 [==============================] - 0s 9ms/step - loss: 2.3931 - accuracy: 0.3229
```

```
18/18 [==============================] - 0s 9ms/step - loss: 2.3951 - accuracy: 0.3229
Epoch 24/100
18/18 [==============================] - 0s 9ms/step - loss: 2.3583 - accuracy: 0.3229
Epoch 25/100
18/18 [==============================] - 0s 10ms/step - loss: 2.3359 - accuracy: 0.3472
Epoch 26/100
18/18 [==============================] - 0s 9ms/step - loss: 2.2672 - accuracy: 0.3559
Epoch 27/100
18/18 [==============================] - 0s 10ms/step - loss: 2.2246 - accuracy: 0.3767
Epoch 28/100
18/18 [==============================] - 0s 9ms/step - loss: 2.1925 - accuracy: 0.3594
Epoch 29/100
18/18 [==============================] - 0s 9ms/step - loss: 2.1424 - accuracy: 0.4045
Epoch 30/100
18/18 [==============================] - 0s 8ms/step - loss: 2.1100 - accuracy: 0.3785
Epoch 31/100
18/18 [==============================] - 0s 9ms/step - loss: 2.0549 - accuracy: 0.4132
Epoch 32/100
18/18 [==============================] - 0s 10ms/step - loss: 2.0323 - accuracy: 0.4201
Epoch 33/100
18/18 [==============================] - 0s 9ms/step - loss: 1.9855 - accuracy: 0.4080
Epoch 34/100
18/18 [==============================] - 0s 10ms/step - loss: 1.9299 - accuracy: 0.4479
Epoch 35/100
18/18 [==============================] - 0s 9ms/step - loss: 1.8803 - accuracy: 0.4479
Epoch 36/100
18/18 [==============================] - 0s 10ms/step - loss: 1.8580 - accuracy: 0.4618
Epoch 37/100
18/18 [==============================] - 0s 9ms/step - loss: 1.8381 - accuracy: 0.4670
Epoch 38/100
18/18 [==============================] - 0s 9ms/step - loss: 1.7948 - accuracy: 0.4931
Epoch 39/100
18/18 [==============================] - 0s 9ms/step - loss: 1.7385 - accuracy: 0.5035
Epoch 40/100
18/18 [==============================] - 0s 10ms/step - loss: 1.6806 - accuracy: 0.5208
Epoch 41/100
18/18 [==============================] - 0s 9ms/step - loss: 1.6329 - accuracy: 0.5556
Epoch 42/100
18/18 [==============================] - 0s 9ms/step - loss: 1.5953 - accuracy: 0.5660
Epoch 43/100
18/18 [==============================] - 0s 9ms/step - loss: 1.5434 - accuracy: 0.5764
Epoch 44/100
18/18 [==============================] - 0s 9ms/step - loss: 1.4999 - accuracy: 0.6024
Epoch 45/100
18/18 [==============================] - 0s 11ms/step - loss: 1.4667 - accuracy: 0.6146
Epoch 46/100
18/18 [==============================] - 0s 9ms/step - loss: 1.4146 - accuracy: 0.6458
Epoch 47/100
18/18 [==============================] - 0s 9ms/step - loss: 1.3693 - accuracy: 0.6424
Epoch 48/100
18/18 [==============================] - 0s 11ms/step - loss: 1.3490 - accuracy: 0.6562
Epoch 49/100
18/18 [==============================] - 0s 9ms/step - loss: 1.3085 - accuracy: 0.6788
Epoch 50/100
18/18 [==============================] - 0s 9ms/step - loss: 1.2580 - accuracy: 0.6875
Epoch 51/100
18/18 [==============================] - 0s 9ms/step - loss: 1.2036 - accuracy: 0.7118
Epoch 52/100
18/18 [==============================] - 0s 10ms/step - loss: 1.1843 - accuracy: 0.7240
Epoch 53/100
18/18 [==============================] - 0s 9ms/step - loss: 1.1262 - accuracy: 0.7535
Epoch 54/100
18/18 [==============================] - 0s 9ms/step - loss: 1.0944 - accuracy: 0.7656
Epoch 55/100
18/18 [==============================] - 0s 9ms/step - loss: 1.0584 - accuracy: 0.7656
Epoch 56/100
18/18 [==============================] - 0s 9ms/step - loss: 1.0088 - accuracy: 0.7847
Epoch 57/100
18/18 [==============================] - 0s 9ms/step - loss: 0.9811 - accuracy: 0.8160
Epoch 58/100
18/18 [==============================] - 0s 9ms/step - loss: 0.9555 - accuracy: 0.7934
Epoch 59/100
18/18 [==============================] - 0s 9ms/step - loss: 0.9202 - accuracy: 0.8212
```

```
18/18 [==============================] - 0s 9ms/step - loss: 0.9202 - accuracy: 0.8212
Epoch 60/100
18/18 [==============================] - 0s 11ms/step - loss: 0.8685 - accuracy: 0.8403
Epoch 61/100
18/18 [==============================] - 0s 9ms/step - loss: 0.8375 - accuracy: 0.8524
Epoch 62/100
18/18 [==============================] - 0s 9ms/step - loss: 0.7967 - accuracy: 0.8698
Epoch 63/100
18/18 [==============================] - 0s 10ms/step - loss: 0.7652 - accuracy: 0.8559
Epoch 64/100
18/18 [==============================] - 0s 9ms/step - loss: 0.7372 - accuracy: 0.8802
Epoch 65/100
18/18 [==============================] - 0s 9ms/step - loss: 0.7132 - accuracy: 0.8837
Epoch 66/100
18/18 [==============================] - 0s 10ms/step - loss: 0.6819 - accuracy: 0.8872
Epoch 67/100
18/18 [==============================] - 0s 10ms/step - loss: 0.6397 - accuracy: 0.9045
Epoch 68/100
18/18 [==============================] - 0s 9ms/step - loss: 0.6248 - accuracy: 0.9097
Epoch 69/100
18/18 [==============================] - 0s 8ms/step - loss: 0.5835 - accuracy: 0.9201
Epoch 70/100
18/18 [==============================] - 0s 9ms/step - loss: 0.5530 - accuracy: 0.9236
Epoch 71/100
18/18 [==============================] - 0s 9ms/step - loss: 0.5337 - accuracy: 0.9323
Epoch 72/100
18/18 [==============================] - 0s 10ms/step - loss: 0.5130 - accuracy: 0.9410
Epoch 73/100
18/18 [==============================] - 0s 9ms/step - loss: 0.4854 - accuracy: 0.9462
Epoch 74/100
18/18 [==============================] - 0s 10ms/step - loss: 0.4658 - accuracy: 0.9566
Epoch 75/100
18/18 [==============================] - 0s 10ms/step - loss: 0.4450 - accuracy: 0.9583
Epoch 76/100
18/18 [==============================] - 0s 9ms/step - loss: 0.4256 - accuracy: 0.9601
Epoch 77/100
18/18 [==============================] - 0s 9ms/step - loss: 0.4014 - accuracy: 0.9618
Epoch 78/100
18/18 [==============================] - 0s 9ms/step - loss: 0.3813 - accuracy: 0.9618
Epoch 79/100
18/18 [==============================] - 0s 9ms/step - loss: 0.3575 - accuracy: 0.9705
Epoch 80/100
18/18 [==============================] - 0s 9ms/step - loss: 0.3382 - accuracy: 0.9740
Epoch 81/100
18/18 [==============================] - 0s 10ms/step - loss: 0.3280 - accuracy: 0.9774
Epoch 82/100
18/18 [==============================] - 0s 9ms/step - loss: 0.3106 - accuracy: 0.9792
Epoch 83/100
18/18 [==============================] - 0s 10ms/step - loss: 0.2954 - accuracy: 0.9826
Epoch 84/100
18/18 [==============================] - 0s 10ms/step - loss: 0.2851 - accuracy: 0.9861
Epoch 85/100
18/18 [==============================] - 0s 9ms/step - loss: 0.2695 - accuracy: 0.9878
Epoch 86/100
18/18 [==============================] - 0s 11ms/step - loss: 0.2566 - accuracy: 0.9931
Epoch 87/100
18/18 [==============================] - 0s 9ms/step - loss: 0.2428 - accuracy: 0.9931
Epoch 88/100
18/18 [==============================] - 0s 9ms/step - loss: 0.2340 - accuracy: 0.9896
Epoch 89/100
18/18 [==============================] - 0s 10ms/step - loss: 0.2251 - accuracy: 0.9913
Epoch 90/100
18/18 [==============================] - 0s 9ms/step - loss: 0.2133 - accuracy: 0.9913
Epoch 91/100
18/18 [==============================] - 0s 10ms/step - loss: 0.2033 - accuracy: 0.9913
Epoch 92/100
18/18 [==============================] - 0s 9ms/step - loss: 0.1958 - accuracy: 0.9931
Epoch 93/100
18/18 [==============================] - 0s 9ms/step - loss: 0.1841 - accuracy: 0.9931
Epoch 94/100
18/18 [==============================] - 0s 9ms/step - loss: 0.1793 - accuracy: 0.9913
Epoch 95/100
18/18 [==============================] - 0s 9ms/step - loss: 0.1736 - accuracy: 0.9896
```

```
Epoch 96/100
18/18 [==============================] - 0s 10ms/step - loss: 0.1725 - accuracy: 0.9931
Epoch 97/100
18/18 [==============================] - 0s 10ms/step - loss: 0.1651 - accuracy: 0.9931
Epoch 98/100
18/18 [==============================] - 0s 9ms/step - loss: 0.1565 - accuracy: 0.9948
Epoch 99/100
18/18 [==============================] - 0s 9ms/step - loss: 0.1464 - accuracy: 0.9948
Epoch 100/100
18/18 [==============================] - 0s 10ms/step - loss: 0.1406 - accuracy: 0.9931
```

In [ ]:

```
# save the model to file
model.save('model.h5')
# save the mapping
dump(mapping, open('mapping.pkl', 'wb'))
```

In [ ]:

```
# generate a sequence of characters with a language model
def generate_seq(model, mapping, seq_length, seed_text, n_chars):
    in_text = seed_text
    # generate a fixed number of characters
    for _ in range(n_chars):
        # encode the characters as integers
        encoded = [mapping[char] for char in in_text]
        # truncate sequences to a fixed length
        encoded = pad_sequences([encoded], maxlen=seq_length, truncating='pre')
        # one hot encode
        encoded = to_categorical(encoded, num_classes=len(mapping))
        # predict character
        yhat = np.argmax(model.predict(encoded), axis=-1)
        # reverse map integer to character
        out_char = ''
        for char, index in mapping.items():
            if index == yhat:
                out_char = char
                break
        # append to input
        in_text += char
    return in_text

# load the model
model = load_model('model.h5')
# load the mapping
mapping = load(open('mapping.pkl', 'rb'))
```

In [ ]:

```
# test start of rhyme
print(generate_seq(model, mapping, 10, 'windmill f', 20))
# test mid-line
print(generate_seq(model, mapping, 10, 'holy town ', 20))
# test not in original
print(generate_seq(model, mapping, 10, 'hello worl', 20))
```

```
windmill for the landTurn fore
holy town where we never smile
hello worla thw fehereor ytive
```

I chose to use our model on the lyrics from one of my favorite songs called "Feel good inc" by the Gorillaz. After uploading the text and tweaking my model to work properly on this new text corpus, I was able to get excellent results. I closely followed the original template given to us for this lab because it yeilded an overhwlemingly more accurate result than any of the tuning that I conducted myself in questions 1-3. Ultimately, the model was able to predict the next lyrics in the song with outstanding accuracy.

In [ ]: