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:

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

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-whe
Requirement already satisfied: tensorflow in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.7/dis
Requirement already satisfied: numpy>=1.20 in /usr/local/lib/python3.7/dist-pack
Requirement already satisfied: setuptools in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.7/dist-pack
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.7/dis
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.7/d
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/loca
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.7/dist
Requirement already satisfied: flatbuffers<2,>=1.12 in /usr/local/lib/python3.7/
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python
```

```
Requirement already satisfied: h5py>=2.9.0 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: tensorboard<2.10,>=2.9 in /usr/local/lib/python3.
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: gast<=0.4.0,>=0.2.1 in /usr/local/lib/python3.7/di
Requirement already satisfied: protobuf<3.20,>=3.9.2 in /usr/local/lib/python3.7/
Requirement already satisfied: keras-preprocessing>=1.1.1 in /usr/local/lib/pytho
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.7/di
Requirement already satisfied: tensorflow-estimator<2.10.0,>=2.9.0rc0 in /usr/loc
Requirement already satisfied: packaging in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: keras<2.10.0,>=2.9.0rc0 in /usr/local/lib/python3
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.7/dis
Requirement already satisfied: cached-property in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: google-auth-oauthlib<0.5,>=0.4.1 in /usr/local/lib
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.7/di
Requirement already satisfied: tensorboard-plugin-wit>=1.6.0 in /usr/local/lib/py
Requirement already satisfied: tensorboard-data-server<0.7.0,>=0.6.0 in /usr/loca
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: google-auth<3,>=1.6.3 in /usr/local/lib/python3.7/
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: cachetools<6.0,>=2.0.0 in /usr/local/lib/python3.7
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.7/
Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/local/lib/python3
Requirement already satisfied: importlib-metadata>=4.4 in /usr/local/lib/python3.
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: pyasn1<0.5.0,>=0.4.6 in /usr/local/lib/python3.7/d
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-pack
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/lo
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dis
Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /usr/local/lib/python3
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-whee
Requirement already satisfied: keras in /usr/local/lib/python3.7/dist-packages (2
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-whee
Requirement already satisfied: h5py in /usr/local/lib/python3.7/dist-packages (3.
Requirement already satisfied: numpy>=1.14.5 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: cached-property in /usr/local/lib/python3.7/dist-p
```

```
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.

```
#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()


#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))

    ng of sixpence,A pocket full of rye.Four and twenty blackbirds,Baked in a pie.Whe
    uences: 384


# 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.

```
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


# 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.

```
# 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

```
from keras.backend import dropout
from keras.layers.core.embedding import Embedding
# define model
model = Sequential()
```

```
#model.add(Embedding(input_dim= 38, output_dim= 38)
model.add(LSTM(210, input_shape=(X.shape[1], X.shape[2]), dropout = 0.2))
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=110)
```

```
Epoch 82/110
12/12 [==============================] - 0s 26ms/step - loss: 0.4774 - accuracy:
Epoch 83/110
12/12 [==============================] - 0s 28ms/step - loss: 0.4923 - accuracy:
Epoch 84/110
12/12 [==============================] - 0s 30ms/step - loss: 0.4132 - accuracy:
Epoch 85/110
12/12 [==============================] - 0s 27ms/step - loss: 0.3869 - accuracy:
Epoch 86/110
12/12 [==============================] - 0s 28ms/step - loss: 0.4885 - accuracy:
Epoch 87/110
12/12 [==============================] - 0s 28ms/step - loss: 0.3707 - accuracy:
Epoch 88/110
12/12 [==============================] - 0s 27ms/step - loss: 0.3397 - accuracy:
Epoch 89/110
12/12 [==============================] - 0s 29ms/step - loss: 0.3704 - accuracy:
Epoch 90/110
12/12 [==============================] - 0s 28ms/step - loss: 0.3633 - accuracy:
Epoch 91/110
12/12 [==============================] - 0s 26ms/step - loss: 0.3553 - accuracy:
Epoch 92/110
12/12 [==============================] - 0s 27ms/step - loss: 0.3599 - accuracy:
Epoch 93/110
12/12 [==============================] - 0s 27ms/step - loss: 0.3668 - accuracy:
Epoch 94/110
12/12 [==============================] - 0s 28ms/step - loss: 0.3062 - accuracy:
Epoch 95/110
12/12 [==============================] - 0s 28ms/step - loss: 0.3589 - accuracy:
Epoch 96/110
12/12 [==============================] - 0s 28ms/step - loss: 0.3437 - accuracy:
Epoch 97/110
12/12 [==============================] - 0s 28ms/step - loss: 0.3554 - accuracy:
Epoch 98/110
12/12 [==============================] - 0s 29ms/step - loss: 0.3034 - accuracy:
Epoch 99/110
12/12 [==============================] - 0s 28ms/step - loss: 0.3211 - accuracy:
Epoch 100/110
12/12 [==============================] - 0s 28ms/step - loss: 0.2843 - accuracy:
Epoch 101/110
12/12 [==============================] - 0s 30ms/step - loss: 0.3472 - accuracy:
Epoch 102/110
12/12 [==============================] - 0s 27ms/step - loss: 0.3107 - accuracy:
Epoch 103/110
12/12 [==============================] - 0s 30ms/step - loss: 0.2891 - accuracy:
Epoch 104/110
12/12 [==============================] - 0s 29ms/step - loss: 0.3221 - accuracy:
Epoch 105/110
```

```
12/12 [==============================] - 0s 27ms/step - loss: 0.3098 - accuracy:
Epoch 106/110
12/12 [==============================] - 0s 28ms/step - loss: 0.3018 - accuracy:
Epoch 107/110
12/12 [==============================] - 0s 28ms/step - loss: 0.2955 - accuracy:
Epoch 108/110
12/12 [==============================] - 0s 28ms/step - loss: 0.2917 - accuracy:
Epoch 109/110
12/12 [==============================] - 0s 28ms/step - loss: 0.2758 - accuracy:
Epoch 110/110
12/12 [==============================] - 0s 29ms/step - loss: 0.2182 - accuracy:
```

```
# 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.

```
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.

```
# 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))
```

```
1/1 [==============================] - 0s 490ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
Sing a song of sixpence,A pock
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 23ms/step
```

```
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 23ms/step
king was in his counting house
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 18ms/step
```

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. I modified the model in the codes above

```
#2
model = Sequential()
model.add(LSTM(76, 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=110, validation_split=0.1)
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_1 (LSTM)               (None, 76)                34960

 dense_1 (Dense)             (None, 38)                2926

=================================================================
Total params: 37,886
Trainable params: 37,886
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/110
11/11 [==============================] - 3s 70ms/step - loss: 3.6166 - accuracy:
Epoch 2/110
11/11 [==============================] - 0s 12ms/step - loss: 3.5364 - accuracy:
Epoch 3/110
11/11 [==============================] - 0s 11ms/step - loss: 3.3236 - accuracy:
Epoch 4/110
11/11 [==============================] - 0s 11ms/step - loss: 3.1429 - accuracy:
Epoch 5/110
11/11 [==============================] - 0s 10ms/step - loss: 3.0799 - accuracy:
Epoch 6/110
11/11 [==============================] - 0s 11ms/step - loss: 3.0346 - accuracy:
Epoch 7/110
11/11 [==============================] - 0s 12ms/step - loss: 3.0210 - accuracy:
Epoch 8/110
```

```
11/11 [==============================] - 0s 12ms/step - loss: 3.0030 - accuracy:
Epoch 9/110
11/11 [==============================] - 0s 11ms/step - loss: 2.9849 - accuracy:
Epoch 10/110
11/11 [==============================] - 0s 11ms/step - loss: 2.9664 - accuracy:
Epoch 11/110
11/11 [==============================] - 0s 11ms/step - loss: 2.9504 - accuracy:
Epoch 12/110
11/11 [==============================] - 0s 11ms/step - loss: 2.9302 - accuracy:
Epoch 13/110
11/11 [==============================] - 0s 12ms/step - loss: 2.9078 - accuracy:
Epoch 14/110
11/11 [==============================] - 0s 11ms/step - loss: 2.8929 - accuracy:
Epoch 15/110
11/11 [==============================] - 0s 12ms/step - loss: 2.8714 - accuracy:
Epoch 16/110
11/11 [==============================] - 0s 12ms/step - loss: 2.8450 - accuracy:
Epoch 17/110
11/11 [==============================] - 0s 11ms/step - loss: 2.8001 - accuracy:
Epoch 18/110
11/11 [==============================] - 0s 11ms/step - loss: 2.7699 - accuracy:
Epoch 19/110
11/11 [==============================] - 0s 11ms/step - loss: 2.7562 - accuracy:
Epoch 20/110
11/11 [==============================] - 0s 11ms/step - loss: 2.7047 - accuracy:
Epoch 21/110
11/11 [==============================] - 0s 11ms/step - loss: 2.6679 - accuracy:
Epoch 22/110
11/11 [==============================] - 0s 11ms/step - loss: 2.6359 - accuracy:
```

3. Firstly I increased the epochs to 110, and the accruacy went up, but the prediction result is not good. So I changed the number of memory cells to 210, and added the dropout rate equaling to 0.2. It gives me a better result. However, when I changed the sequence length from 10 to 20, the accuracy goes up to almost 1, but the prediction result is not good. So I changed it back to sequence length of 10. Overall, I think the reason why the the third prediction result (Hello world) is not good because our trainset is small, and it's not a modern text, it's a ancient poem. So the model have never seen the modern English in the training, that's why it couldn't give a better prediction.

```
#4 New Text, and a word_level model
s='Two roads diverged in a yellow wood,\
And sorry I could not travel both \
And be one traveler, long I stood \
And looked down one as far as I could \
To where it bent in the undergrowth \
Then took the other, as just as fair,\
And having perhaps the better claim,\
Because it was grassy and wanted wear;\
Though as for that the passing there \
Had worn them really about the same,\
```

```
And both that morning equally lay \
In leaves no step had trodden black.\
Oh, I kept the first for another day \
Yet knowing how way leads on to way \
I doubted if I should ever come back \
I shall be telling this with a sigh \
Somewhere ages and ages hence \
Two roads diverged in a wood, and I \
I took the one less traveled by \
And that has made all the difference.'

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


#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()

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

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


# organize into sequences of characters
length = 4
sequences = list()
for i in range(length, len(raw_text.split())):
    # select sequence of tokens
    seq = ' '.join(raw_text.split()[i-length:i+1])
    # store
    sequences.append(seq)
print('Total Sequences: %d' % len(sequences))

# save sequences to file
out_filename = 'char_sequences.txt'
```

```
save_doc(sequences, out_filename)

# 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

# load

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

words = sorted(set(raw_text.split()))
mapping = dict((c, i) for i, c in enumerate(words))
print(mapping)
sequences = list()

for line in lines:
    # integer encode line
    encoded_seq = [mapping[word] for word in line.split()]
    # 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)
```

```
Two roads diverged in a yellow wood,And sorry I could not travel both And be one
Total Sequences: 139
{'And': 0, 'Because': 1, 'Had': 2, 'I': 3, 'In': 4, 'Oh': 5, 'Somewhere': 6, 'Th
Vocabulary Size: 97
```

```
# define model
model = Sequential()
#model.add(Embedding(input_dim= 38, output_dim= 38)
model.add(LSTM(210, input_shape=(X.shape[1], X.shape[2]), dropout = 0.2))
```

```
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=110)

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

```
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_7 (LSTM)               (None, 210)               258720

 dense_7 (Dense)             (None, 97)                20467

=================================================================
Total params: 279,187
Trainable params: 279,187
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/110
5/5 [==============================] - 2s 17ms/step - loss: 4.5769 - accuracy: 0
Epoch 2/110
5/5 [==============================] - 0s 18ms/step - loss: 4.5497 - accuracy: 0
Epoch 3/110
5/5 [==============================] - 0s 18ms/step - loss: 4.5266 - accuracy: 0
Epoch 4/110
5/5 [==============================] - 0s 17ms/step - loss: 4.5048 - accuracy: 0
Epoch 5/110
5/5 [==============================] - 0s 18ms/step - loss: 4.4808 - accuracy: 0
Epoch 6/110
5/5 [==============================] - 0s 16ms/step - loss: 4.4556 - accuracy: 0
Epoch 7/110
5/5 [==============================] - 0s 18ms/step - loss: 4.4204 - accuracy: 0
Epoch 8/110
5/5 [==============================] - 0s 20ms/step - loss: 4.3836 - accuracy: 0
Epoch 9/110
5/5 [==============================] - 0s 17ms/step - loss: 4.3240 - accuracy: 0
Epoch 10/110
5/5 [==============================] - 0s 17ms/step - loss: 4.2370 - accuracy: 0
Epoch 11/110
5/5 [==============================] - 0s 19ms/step - loss: 4.1150 - accuracy: 0
Epoch 12/110
5/5 [==============================] - 0s 17ms/step - loss: 4.0023 - accuracy: 0
Epoch 13/110
5/5 [==============================] - 0s 18ms/step - loss: 3.9528 - accuracy: 0
Epoch 14/110
5/5 [==============================] - 0s 19ms/step - loss: 3.8257 - accuracy: 0
Epoch 15/110
```

```
5/5 [==============================] - 0s 18ms/step - loss: 3.7256 - accuracy: 0
Epoch 16/110
5/5 [==============================] - 0s 20ms/step - loss: 3.5778 - accuracy: 0
Epoch 17/110
5/5 [==============================] - 0s 19ms/step - loss: 3.4483 - accuracy: 0
Epoch 18/110
5/5 [==============================] - 0s 17ms/step - loss: 3.2610 - accuracy: 0
Epoch 19/110
5/5 [==============================] - 0s 19ms/step - loss: 3.0554 - accuracy: 0
Epoch 20/110
5/5 [==============================] - 0s 16ms/step - loss: 2.9139 - accuracy: 0
Epoch 21/110
5/5 [==============================] - 0s 18ms/step - loss: 2.7599 - accuracy: 0
Epoch 22/110
5/5 [==============================] - 0s 18ms/step - loss: 2.5528 - accuracy: 0
```

```
def generate_seq_word(model, mapping, seq_length, seed_text, n_words):
    in_text = seed_text
    # generate a fixed number of characters
    for _ in range(n_words):
      encoded = [mapping[word] for word in in_text.split()]
        # 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_word = ''
      for word, index in mapping.items():
        if index == yhat:
          out_word = word
          break
        # append to input
      in_text = in_text + " " + word
    return in_text


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


print(generate_seq_word(model, mapping, 4, 'And sorry I could', 10))

print(generate_seq_word(model, mapping, 4, 'the first for another', 5))
```

```
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 75ms/step
```

```
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 46ms/step
1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 95ms/step
1/1 [==============================] - 0s 57ms/step
1/1 [==============================] - 0s 18ms/step
And sorry I could not travel both And be one traveler long I stood
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 22ms/step
the first for another day Yet knowing how way
```