Text Generation Using LSTMs

Source code: https://github.com/adibyte95/Text-Generation-using-LSTM/blob/master/text_generation_prog.ipynb

Dataset: https://github.com/adibyte95/Text-Generation-using-LSTM/blob/master/wonderland.txt (original dataset)
https://github.com/xinyili2022/Deep-Learning-Text-generation-LSTM/blob/main/wonderland_simplied.txt (simplied datset)

My Code:
https://github.com/xinyili2022/Deep-Learning-Text-generation-LSTM/blob/main/(Simplied%20dataset)Text_generation_using_LSTM.ipynb (simplied version)

Presenter: Xinyi Li

Dataset

Original dataset

```
Lewis Carroll

THE MILLENNIUM FULCRUM EDITION 3.0




CHAPTER I. Down the Rabbit-Hole

Alice was beginning to get very tired of sitting by her sister on the
bank, and of having nothing to do: once or twice she had peeped into the
book her sister was reading, but it had no pictures or conversations in
it, ‘and what is the use of a book,’ thought Alice ‘without pictures or
conversations?’
```

Simpified dataset: deleted some chapters from the original dataset

Goal

Use LSTM to learn the sequences of characters from dataset.

1. Generate text for a random seed(number)

2. Generate characters for a input text

Step 1: Data Preparation

```
filename = "wonderland.txt"
raw_text = open(filename).read()
raw_text = raw_text.lower()
```

Original code

```
filename = "wonderland.txt"
raw_text = open(filename, 'r', encoding='latin-1').read()
#raw_text = open(filename).read()
raw_text = raw_text.lower() #vocabulary reduction
```

Modified code

ALICE, Alice, alice are the same vocabularies

Step 1: Data Preparation

```
chars = sorted(list(set(raw_text)))
char_to_int = dict((c, i) for i, c in enumerate(chars))
int_to_char = dict((i, c) for i, c in enumerate(chars))
```

```
print(chars)
print(char_to_int)
print(int_to_char)
```

To sort the list of unique characters:

Print (int_to_char):
{0: '\n', 1: ' ', 2: '!', 3: '(', 4: ')', 5: '*', 6: ',', 7: '-', 8: '.', 9: '0', 10: '3', 11: ':', 12: ';', 13: '?', 14: '[', 15: ']', 16: '_', 17: 'a', 18: 'b', 19: 'c', 20: 'd', 21: 'e', 22: 'f', 23: 'g', 24: 'h', 25: 'i', 26: 'j', 27: 'k', 28: 'l', 29: 'm', 30: 'n', 31: 'o', 32: 'p', 33: 'q', 34: 'r', 35: 's', 36: 't', 37: 'u', 38: 'v', 39: 'w', 40: 'x', 41: 'y', 42: 'z', 43: '\x91', 44: '\x92', 45: '\x93', 46: '\x94'}

Step 1: Data Preparation

```
seq_length = 100
dataX, dataY = [], []
```

```
for i in range(0, len(raw_text) - seq_length, 1):
    seq_in = raw_text[i:i + seq_length]
    seq_out = raw_text[i + seq_length]
    dataX.append([char_to_int[char] for char in seq_in])
    dataY.append(char_to_int[seq_out])
```

Original code

Don’t understand? Starting from the simple version

Step 1: Data Preparation

```
seq_in = raw_text[1:1 + 50]
print("seq_in:",seq_in)
seq_out = raw_text[51]
print("seq_out:",seq_out)
```

```
seq_in: ** start of this project gutenberg ebook alice’s a
seq_out: d
```

```
dataX=[]
print(seq_in)
dataX.append([char_to_int[char] for char in seq_in])
print(dataX)
```

```
** start of this project gutenberg ebook alice’s a
[[5, 5, 1, 35, 36, 17, 34, 36, 1, 31, 22, 1, 36, 24, 25, 35, 1, 32, 34, 31, 26, 21, 19, 36, 1, 23, 37, 36, 21, 30, 18, 21, 34,
23, 1, 21, 18, 31, 31, 27, 1, 17, 28, 25, 19, 21, 44, 35, 1, 17]]
```

```
dataY = []
print(seq_out)
dataY.append(char_to_int[seq_out])
print(dataY)
```

```
d
[20]
```

Step 1: Data Preparation

```
seq_length = 100
dataX, dataY = [], []
```

```
for i in range(0, len(raw_text) - seq_length, 1):
    seq_in = raw_text[i:i + seq_length]
    seq_out = raw_text[i + seq_length]
    dataX.append([char_to_int[char] for char in seq_in])
    dataY.append(char_to_int[seq_out])
```

Understand:
aaaaaaaaaaaaaaaaaaaaaaaaaa(100)aaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaa(100)aaaaaaaaaaaaaa
……
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa(100)a
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa(100)

Step 1: Data Preparation

```
# reshape X to be [samples, time steps, features]
X = numpy.reshape(dataX, (n_patterns, seq_length, 1))
# normalize
X = X / float(n_vocab)
# one hot encode the output variable
y = np_utils.to_categorical(dataY)
```

Original code

```
from sklearn.preprocessing import OneHotEncoder

encoder = OneHotEncoder(sparse=False, categories='auto')
y = encoder.fit_transform(np.array(dataY).reshape(-1, 1))
```

Modified code

Step 2: LSTM Modelization

```
# Define the LSTM model
model = Sequential()
model.add(LSTM(256, input_shape=(X.shape[1], X.shape[2]), return_sequences=True))
model.add(Dropout(0.8))

model.add(LSTM(256))
model.add(Dropout(0.8))

model.add(Dense(len(unique_chars), activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam')
```

```
# Train the model (you can adjust epochs and batch_size)
model.fit(X, y, epochs=10, batch_size=2000, verbose=1)
```

Step3: Text Generation

```
#generating the text
# pick a random seed
random = np.random.randint(0, len(dataX)-1)
pattern = dataX[random]
print("\"", ''.join([ints_to_chars[value] for value in pattern]), "\"")
```

```
" ere me?’

‘well, perhaps not,’ said alice in a soothing tone: ‘don’t be angry
about it. and yet i wi "
```

Set a random seed number to get a text from the text dataset.

Run the code form multiple times would getting various results.

```
#generating the text
# pick a random seed
start = np.random.randint(0, len(dataX)-1)
pattern = dataX[start]
print ("Seed:")
print ("\"", ''.join([int_to_char[value] for value in pattern]), "\"")
```

```
Seed:
" only knew how to begin.’
for, you see, so many out-of-the-way things had happened lately,
that alice "
```

Step3: Text Generation

```
# generate characters
for i in range(1000):
        x = numpy.reshape(pattern, (1, len(pattern), 1))
        x = x / float(n_vocab)
        prediction = model.predict(x, verbose=0)
        index = numpy.argmax(prediction)
        result = int_to_char[index]
        seq_in = [int_to_char[value] for value in pattern]
        sys.stdout.write(result)
        pattern.append(index)
        pattern = pattern[1:len(pattern)]
print ("\nDone.")
```

Original code

```
#generate based on the input text
def generate_text(seed_text, n_chars=100):
    seed_text=seed_text.lower()
    generated_text = seed_text

    for i in range(n_chars):
        x_pred = np.array([char_to_int[c] for c in seed_text])
        x_pred = np.reshape(x_pred, (1, len(x_pred), 1)) / float(len(chars))

        prediction = model.predict(x_pred, verbose=0)
        index = np.argmax(prediction)
        char_out = int_to_char[index]

        generated_text += char_out
        seed_text = seed_text[1:] + char_out


    return generated_text
```

Modified code

Step3: Text Generation

```
‘what is a "
 larter ’ said the ming. ‘i dan to the thing the was to the thing the was to the thing the was a little said the was oot a lit
tle sabbit with the was oot a little sabbit with the was oot a little sabbit with the was oot a little sabbit with the was oot
a little sabbit with the was oot a little sabbit with the was oot a little sabbit with the was oot a little sabbit with the wa
s oot a little sabbit with the was oot a little sabbit with the was oot a little sabbit with the was oot a little sabbit with
the was oot a little sabbit with the was oot a little sabbit with the was oot a little sabbit with the was oot a little sabbit
with the was oot a little sabbit with the was oot a little sabbit with the was oot a little sabbit with the was oot a little s
abbit with the was oot a little sabbit with the was oot a little sabbit with the was oot a little sabbit with the was oot a li
ttle sabbit with the was oot a little sabbit with the was oot a little sabbit with the was oot a little sabbit with
Done.
```

```
print(generate_text("Alice was beginning",200))
```

```
alice was beginning
```

Simplified Dataset

```
alice was beginning to the cane the taid the taid the taid the taid the taid the taid the taid the taid the taid the taid the
taid the taid the taid the taid the taid the taid the taid the taid the taid the taid the tai
```

Original Dataset