**An AI Writer—Using RNN to Generate Poetries and Lyrics**

**Topic**

In our textbook, chapter8, it introduces how to implement character-level LSTM text generation in English. However, many combinations of English letters are not meaningful. In contrast, every Chinese character has its own meaning and we can always get meaningful combinations of Chinese characters by training Chinese texts. In this project, I want to generate Chinese ancient poetries and Chinese lyrics. Since these two kinds of texts are different from each other, I built two neural networks with different data preprocessing method and different structures. Then I trained them on my gaming laptop to generate texts. My GPU is GTX 970M with 6.00 GB. After that, I integrated these two trained models into one GUI to generate both Chinese ancient poetries and lyrics.

**Generating Chinese Ancient poetry**

1. **Dataset**

<https://github.com/jackeyGao/chinese-poetry>

I downloaded the data and used the code below to extract the data I want. Considering my limited GPU performance, I only extracted those poetries with five to seven Chinese characters in each sentence from the whole large dataset using the code below. Based on this subset, I can also generate poetries with five to seven Chinese characters in each sentence. After building my own dataset, I still need to transfer every poetry from Chinese characters into a vector or we should say a tensor. The detailed code can be found on my Github.

<https://github.com/wangshibo1993/An-AI-Writer-Using-RNN-to-Generate-Poetries-and-Lyrics-Course-Project>

paths = glob.glob('./chinese-poetry-master/json/poet.\*.json')

poets = []

for path in paths:

data = open(path, 'r', encoding='utf-8').read()

data = json.loads(data)

#print(data)

for item in data:

content = ''.join(item['paragraphs'])

if len(content) >= 24 and len(content) <= 32:

content = SnowNLP(content)

content=content.han

if len(content)%4==0:

poets.append('[' + content + ']')

print('We have %d Chinese ancient poets' % len(poets), poets[0], poets[-1])

We have 104147 Chinese ancient poets [欲出未出光辣达，千山万山如火发。须臾走向天上来，逐却残星赶却月。] [书劒催人不暂闲，洛阳羁旅复秦关。容颜岁岁愁边改，乡国时时梦里还。]

In my previously submitted “Project Assignment 4”, I did not set a validation set. However, in this final report, I add a validation set to monitor the training process.

1. **LSTM Model**

My code references this work [2]. The model is built on a linear stack of layers with the sequential model. There are two hidden layers in total, which are LSTM layers. And compared to reference 2 and my previous version in “Project Assignment 4”, I also add Dropout in each layer. The whole model is built as:

cell = tf.nn.rnn\_cell.MultiRNNCell(

[tf.nn.rnn\_cell.DropoutWrapper(tf.nn.rnn\_cell.LSTMCell(hidden\_size), input\_keep\_prob=ikeep\_prob, output\_keep\_prob=ikeep\_prob )for i in range(num\_layer)],

state\_is\_tuple=True)

1. **Shape of Tensor**

X\_training is (256, ?), in which “?” needs to be assigned in the training process. It should be the maximum length of a vector transferred from a poetry. For example, if a poetry has 4 sentences, each of which has 7 Chinese characters, the total length of the transferred vector is 7\*4+4+2-1=33. Here we include some characters “,”, “.”, “[” and “]”.

After an embedding layer, the input data is transferred into (256, ?, 256). We can regard this step as one-hot-encoding and the embedded layer is also included in tf.trainable\_variables(), which means that it is trained during the training process.

There are two LSTM layers, each of which has 256 nodes. Thus, the output of the first and second hidden layer are both (256, ?, 256). And in the last output layer, we use softmax and the final output tensor is (256, ?, 8011), where 8011 is the total amount of all Chinese characters included in my dataset. However, if we use a validation set, this dimension in training data set is not 8011 anymore but depends on the validation set we choose.

1. **Hyperparameters & Training and Testing Performance**

In this project, there are many hyperparameters to tune. However, we found that the validation loss is always convergent to 6.0 and increases again, which means overfitting. The following four figures shown below present training and validation loss with different hyperparameters. We really have tried many modifications, from changing the neuron size to decrease training step length, and the lower bound of validation loss still exists. For example, if we increase LSTM layers or decrease embedding\_size of the embedding layer, the validation loss would become convergent after about 20-25 epochs, but still convergent to 6.0. It seems that we have no better choice. However, this project is to generate texts, which cannot be evaluated just by loss or accuracy but also based on human perception. And if we choose the valley of the validation loss, which means the epoch in which the lowest validation loss exists, we found that sometimes our model would generate some messy format. We suppose that at this moment our model has not remembered enough patterns from these poetries. Thus, we need to make the model in some degree overfitting. And the more Chinese characters’ combinations our model can remember, the more its generation looks like real poetry. Finally, after many experiments, we choose the batch size, embedding\_size, and neuron size all as 256 and also we set 2 LSTM layers, each with dropout as 0.8. The training epoch is set as 50.

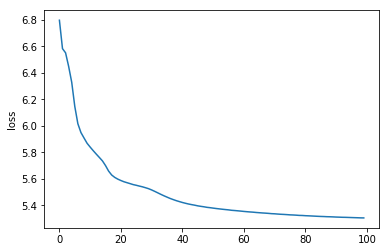


Figure 1. Training loss with eopchs. Neuron size and Embedding\_size are both 128.

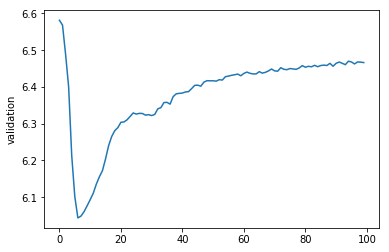


Figure 2. Validation loss with eopchs. Neuron size and Embedding\_size are both 128.

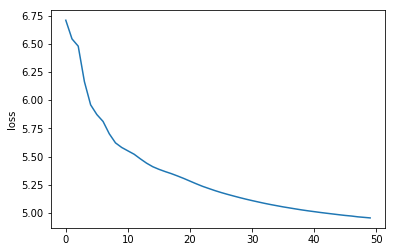


Figure 3. Training loss with eopchs. Neuron size and Embedding\_size are both 256.

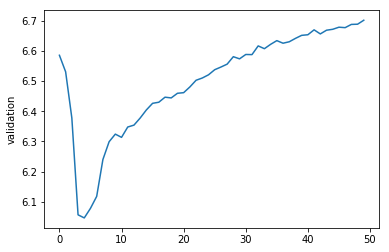


Figure 4. Validation loss with eopchs. Neuron size and Embedding\_size are both 128.

1. **Annotated Code**

The annotated code is shown on Github.

<https://github.com/wangshibo1993/An-AI-Writer-Using-RNN-to-Generate-Poetries-and-Lyrics-Course-Project>

1. **Instruction on how to use the trained model and the GUI to generate poetry**

**6.1 Install Dependencies**

Python 3

Tkinter

Keras

Tensorow

Numpy,Sicpy

**6.2 Execution**

Please first unzip the model “model\_epoch50\_2lstm\_1dense\_seq50\_phrase\_based\_ best.zip” in “demo” file and then run demo.py file and you can play around with the demo. Please attention that we have two patterns of generating poetry. If you do not input anything and generate directly, the model will generate a poetry randomly. And if you input something, and choose a generating method, our model will generate an acrostic poetry or a poetry with the specific topic that you just input before. And there is a low possibility that the program may have a KeyError. If you meet this problem, just restart the demo. The reason why we have this problem is that the training epoch is not enough. However, this is really a trade-off because if we train more epochs, the overfitting will be very severe and generating texts just becomes reciting the full text. Some generated examples are shown below:

离愁满面日相留，落日闲高晓景融。最爱一池风雨急，杜鹃千里杏花流。

边塞南边欲黯回，白羊归去半归回。日光万里无尘土，只在春山半落梅

长歌相背识联鸿，江浙春空满帝宫。朝雨便登双白鹤，郢花真属显曹公

天台云室遶山关，下水云生复浦环。无限目光留不得，双天雪作玉溪闲。

石帆南浦竞人行，翅薄寒荷白发生。近有叶来溪上过，白苹弄影照沧浪。

**6.3 Code**

The GUI code is shown on Github.

**6.4 Video**

Attached in my Github.

**Generating Chinese Lyrics**

1. **Dataset**

<https://github.com/Encaik/TongJi>

We download the data and use the “data\_processing.ipynb”, which is upload to the Github, to extract and clean the data that I want.

In this part, I also add a validation set to monitor the training process. In the first topic, “Generating Chinese ancient Poetry”, I used Tensorflow. However, I found Keras is much easier and more user-friendly. Thus, in this topic, I choose Keras to build my neural network.

1. **LSTM Model**

Just as the previous topic, here we also use LSTM layers. My code references an example in the official document of Keras [3]. Our model is built on a linear stack of layers with the sequential model. However, since lyrics are not as rhyming as poetries, we need to use a little more complicated network. Thus, we choose bidirectional LSTM and also add an embedding layer before these three LSTM layers, which is much easier to set in Keras than in Tensorflow.

1. **Shape of Tensor**

It is very convenient for Keras to output the shape of the tensor in each layer, as is shown below in Table 1.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

x\_train (InputLayer) (None, 3) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

embedding\_1 (Embedding) (None, 3, 256) 873472

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

bidirectional\_1 (Bidirectional) (None, 3, 1024) 3149824

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

bidirectional\_2 (Bidirectional) (None, 3, 1024) 6295552

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

bidirectional\_3 (Bidirectional) (None, 512) 2623488

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Dense\_1 (Dense) (None, 3412) 1750356

=================================================================

Total params: 14,692,692

Trainable params: 14,692,692

Non-trainable params: 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Table 1. Model Summary

1. **Hyperparameters & Training and Testing Performance**

In this project, there are many hyperparameters to tune. However, just as in the previous topic, the validation loss here is also tending to convergent to a point which is between 4.3 and 4.5, then it increases again, which means overfitting. To get the lowest validation loss valley, we set our hyperparameters as below:

SEQ\_LENGTH = 3

EMBEDDING\_DIM = 512

EMBEDDING\_DIM\_2 = 512

EMBEDDING\_DIM\_3 = 256

BATCH\_SIZE = 1024

EPOCHS = 50

The reason why we choose a very short SEQ\_LENGTH which is just 3 is that in our training lyrics, there are many short Chinese characters’ combinations which can constitute a longer meaningful sentence. And we could also see that the valley is around 40 epochs but we choose 50 epochs. Here we make our network a little overfitting but not very severe because the validation accuracy is still in the rising edge from 40 to 50 epochs. Therefore the network can remember many useful Chinese characters’ combinations but avoiding reciting the full text. We use Tensorboard to monitor the training process. The results of the optimized network are shown in the following four figures, from Figure 5 to Figure 8.

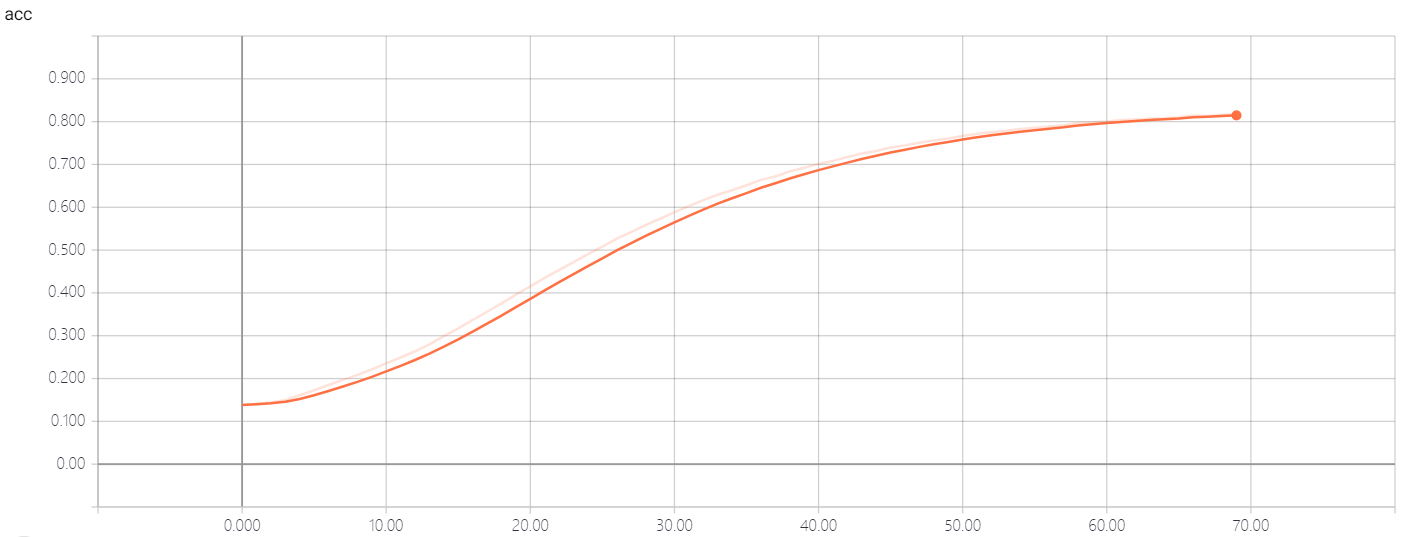


Figure 5. Training accuracy with epochs.

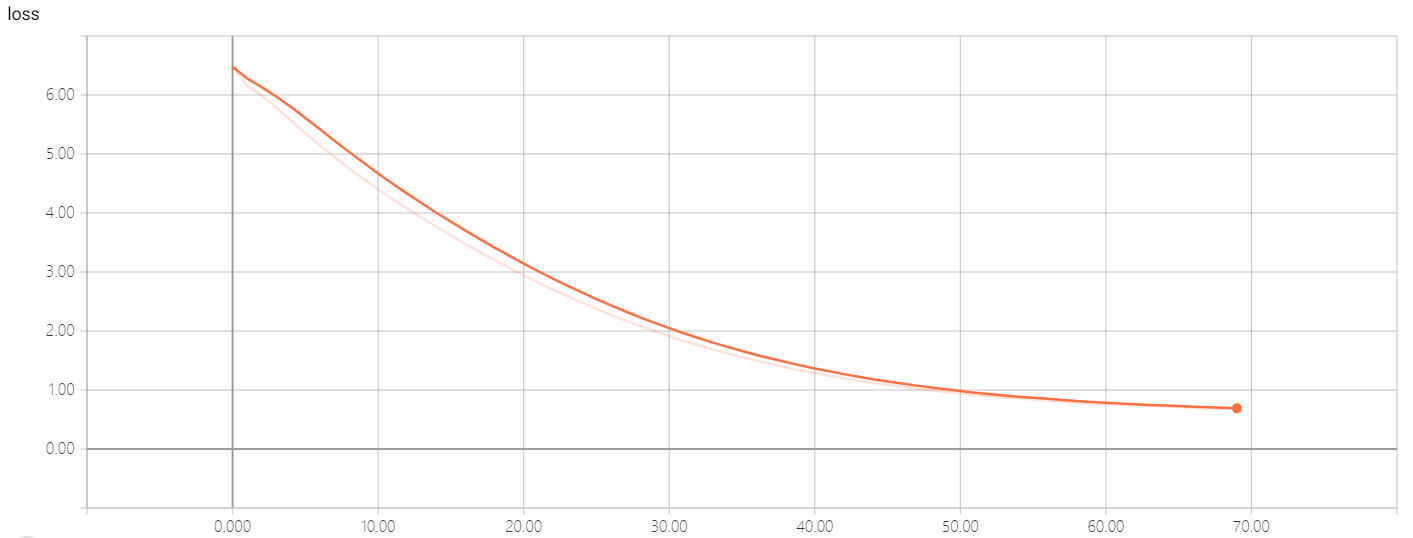


Figure 6. Training loss with epochs.

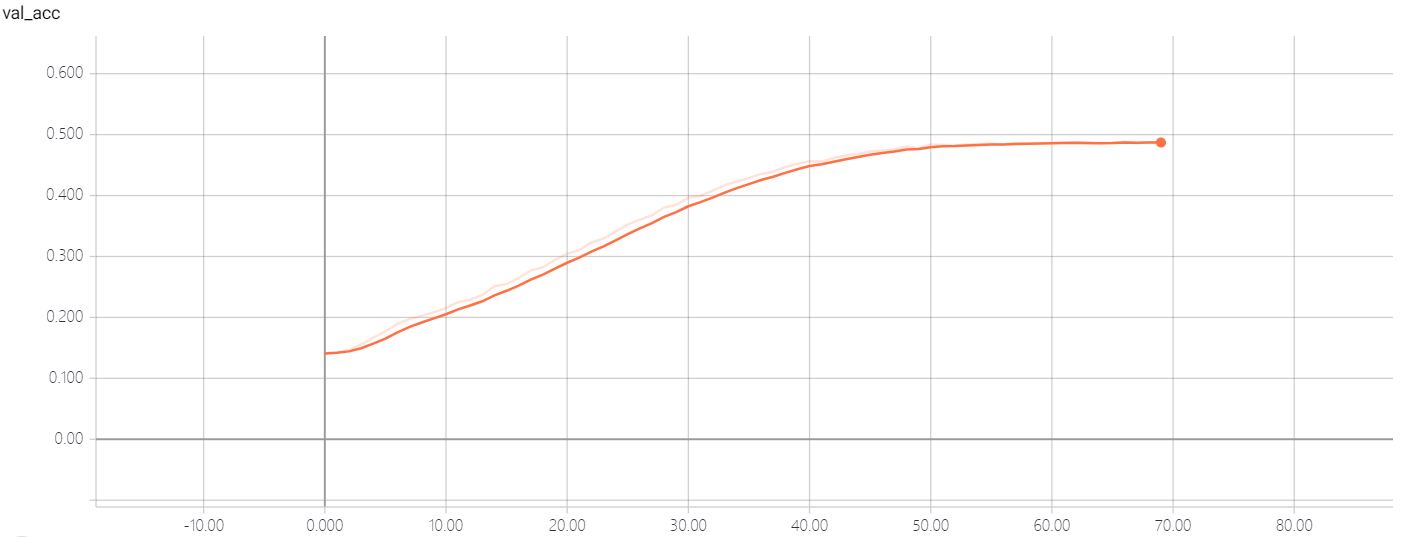


Figure 7. Validation accuracy with epochs.

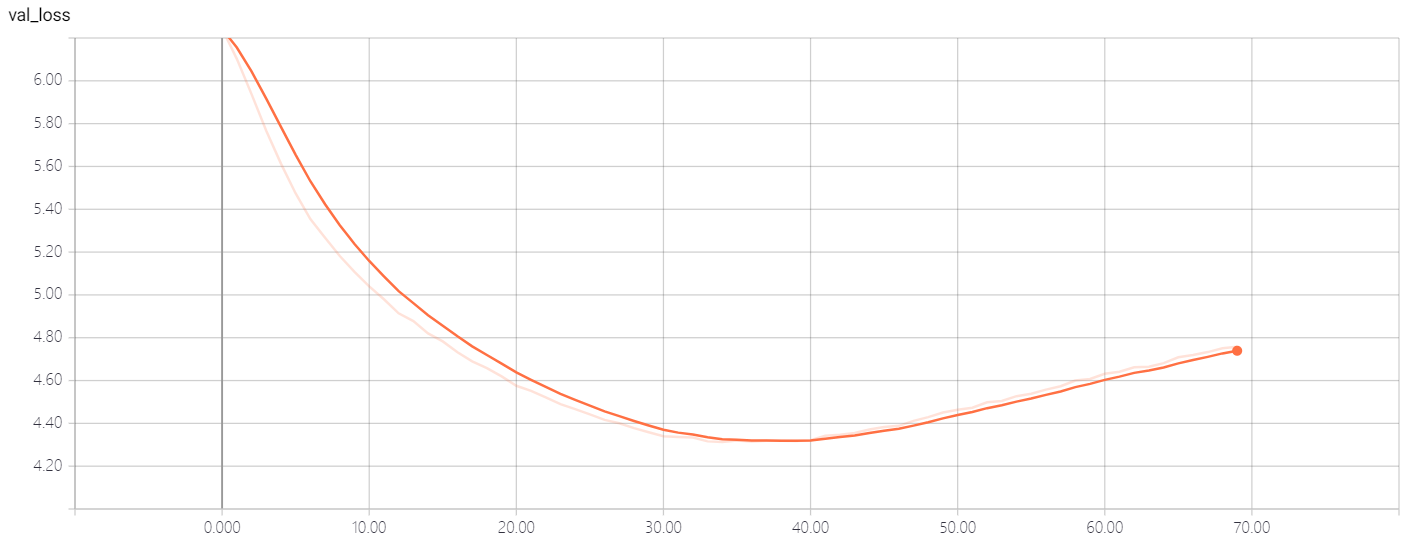


Figure 8. Validation loss with epochs.

And Figure 9 and 10 show that a two-layer model becomes overfitting much quicker than our optimized three-layer model. This also supports our choice of hyperparameters.

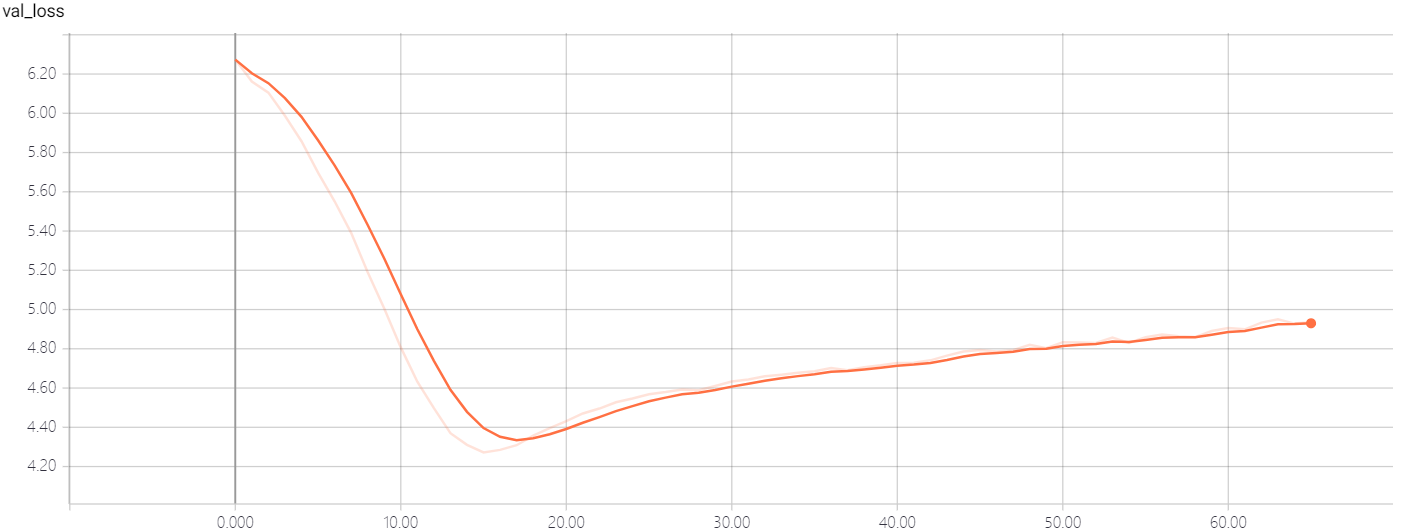


Figure 9. Validation loss of the model with two LSTM layers with epochs.

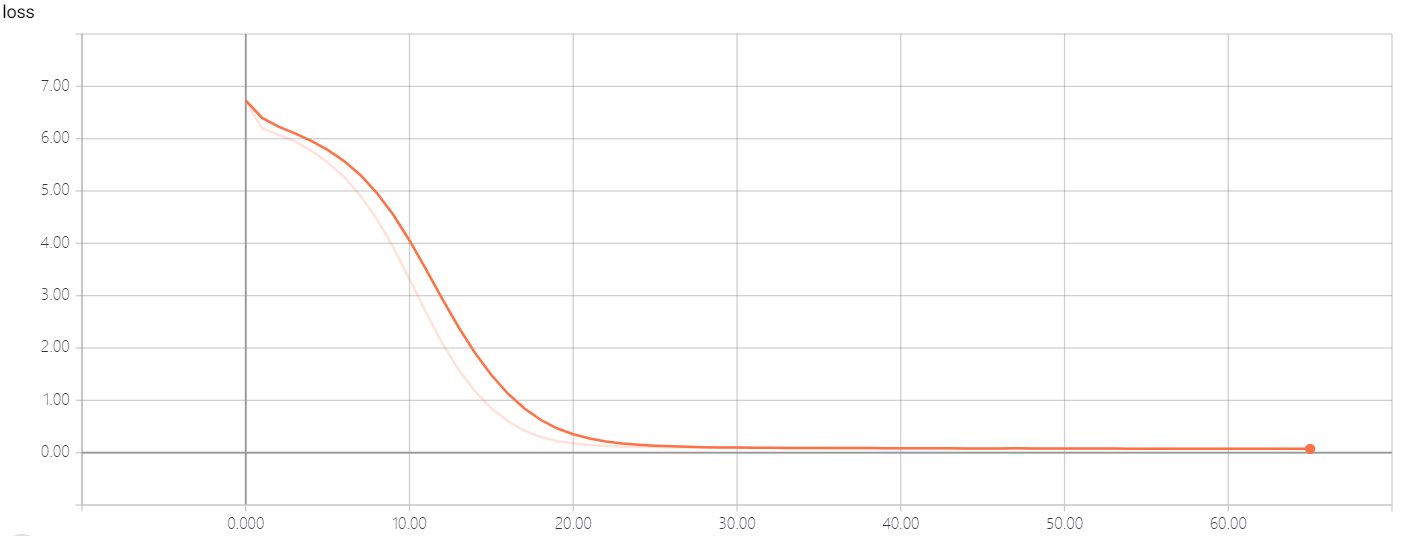


Figure 10. Training loss of the model with two LSTM layers (512, 1024) with epochs.

And also, just as our textbook [1] says, with higher diversity, the generated text becomes more interesting, surprising and even creative and it can avoid extremely repetitive and predictable local structures. Thus, in our generating function, we set the diversity as 1.0.

preds = np.asarray(preds).astype("float64")

preds = np.log(preds + 1e-10) / diversity

1. **Annotated Code**

The annotated code is shown on Github.

1. **Instruction on how to use the trained model and the GUI to generate poetry**

**6.1 Install Dependencies**

Python 3

Tkinter

Keras

Tensorow

Numpy,Sicpy

**6.2 Execution**

Run demo.py file and you can play around with the demo. If you do not input anything and generate directly, the model will generate a lyric randomly. And if you input something, our model will generate a lyric with the specific topic that you just input before. And there is a low possibility that the program may have a KeyError. If you meet this problem, just restart the demo. Some generated examples are shown below:

1. 八荒

俯瞰饮倾响

情话的脂垢

功过有几许见

那个鞠躬尽瘁的托孤之臣

原本该料事如神

谋定乾坤

曾檀板击节奏炽血

聚与合未书写

摧心化骨

都随它

待后人来寻

征魂归家无言

正是山明呀

遮蔽眼神

背转身

察觉不出天际正情愿

我得三世方写

故事渐渐爬亮双亲

山水幻能散

兰亭袅袅入凡尘

行书以鉴当琼月

锋寄骨弦上

无情清浊

棍下丈量就算像心的姑娘

是否就算是拥有春天

我又为你在乎

若贻笑旧流放

还记得吗

1. 边塞

相逢相知本无意

锋如虹

此霜渐怒浪透与夕阳低意远空残云飞雪一季

我依然眉眼也罢

枝上的碧桃芽

艳红开遍多热闹

旖旎风光满城绕

人来人往轮回飘萍

楚色云淡流霞冷

雷霆岁月

现在已经回家

夕阳拂去归路

掠惊鸿留在脆弱新桑

风卷战歌喝情长

许是为它而酿

轻轻地唱

月亮照在陇上

我的姑娘啊

霜雪飘零之身相逢于落寞

九皐唳鹤来收尽的那份乐负各自说不识得愁

儿女传人

换一场嚣云转绵

何必较自由

最坦荡的

**6.3 Code**

The GUI code is shown on Github.

**6.4 Video**

Attached in my Github.

**Reference**

1. Textbook
2. <https://github.com/sherjilozair/char-rnn-tensorflow>
3. <https://github.com/keras-team/keras/blob/master/examples/lstm_text_generation.py>