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.

2. 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)
```

3. Shape of Tensor

Figure A illustrates the neural network model. X_training is (256, ?), in which the first dimension "256" is the batch size and the second dimension "?" 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 transfers each character's id into the range of [-1, 1] uniformly. Each Chinese character is translated into a 256-dimension vector where 256 is the embedding size. This embedding 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 hidden size. Thus, the output of the first hidden layer is (256, ?, 256) and of the second hidden layer is (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 the training data set is not 8011 anymore but depends on the validation set we choose. The loss function used here is sequence_loss.

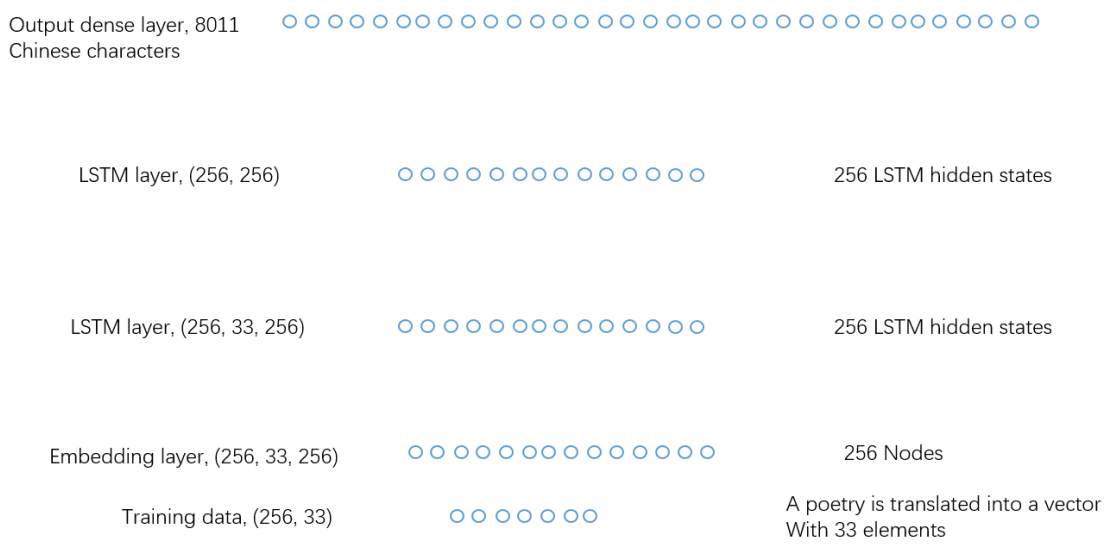

Figure A

4. 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.

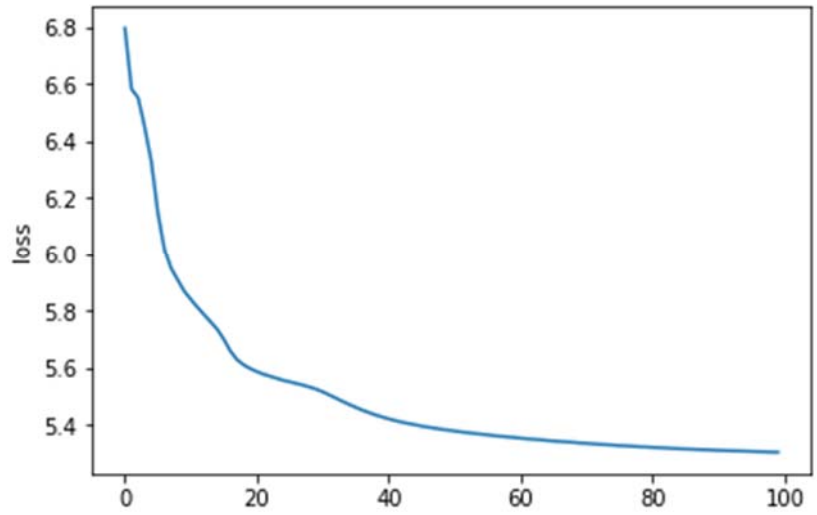

Figure 1. Training loss with eopchs. Neuron size and Embedding_size are both 128.

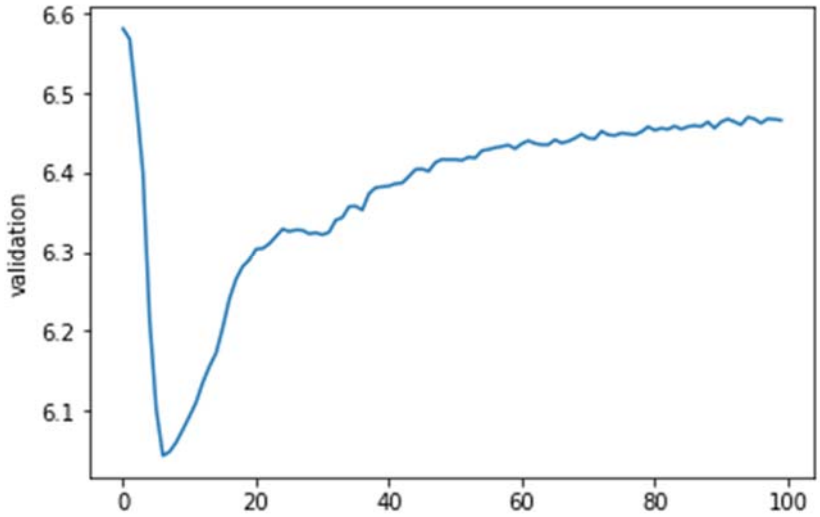

Figure 2. Validation loss with eopchs. Neuron size and Embedding_size are both 128.

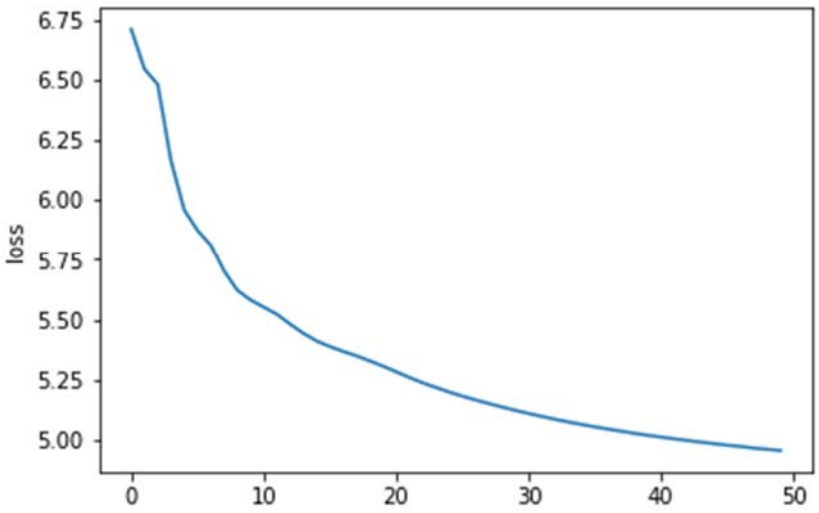

Figure 3. Training loss with eopchs. Neuron size and Embedding_size are both 256.

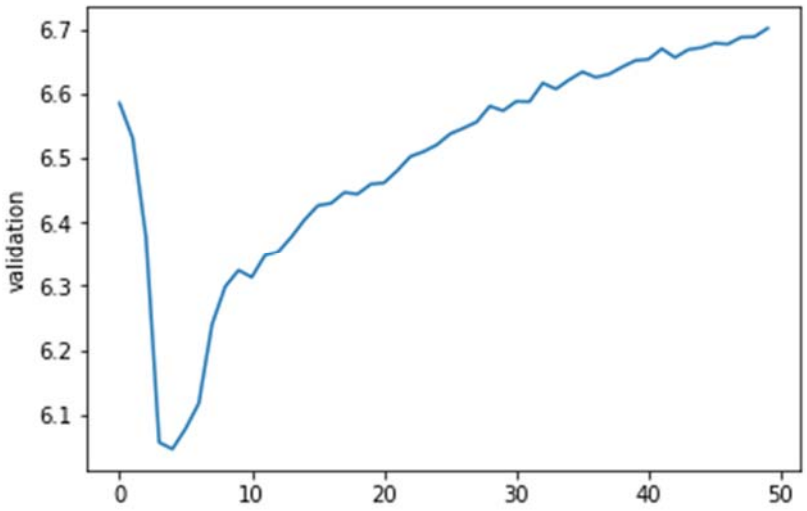

Figure 4. Validation loss with eopchs. Neuron size and Embedding_size are both 128.

5. 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

6. 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

https://github.com/wangshibo1993/An-AI-Writer-Using-RNN-to-Generate-Poetries-and-Lyrics-Course-Project/blob/master/Apr%2016%202019%2011_11%20AM.webm

7. Using Genetic Algorithm as A Systematic AutoML Method

In this part, I will use genetic algorithm to help us tune the parameters of our neural network instead of just manual tuning of these parameters. Genetic algorithm is a heuristic search, which is an optimization method based on the natural selection process. They are widely used to find optimization problems for approximate optimal solutions in large parameter spaces. The process of species evolution (the solution in the example) is imitated, relying on biologically inspired parts, such as crossovers. In addition, since it does not consider auxiliary information (such as derivatives), it can be used for discrete and continuous optimization.

For genetic algorithms, two prerequisites must be met, a) the solution represents or defines a chromosome, and b) the fitness function to evaluate the resulting solution. In our case, I want to find the best hidden size combination of two LSTM layers. Thus, the binary array which is the genetic representation of the solution is shown in Figure 5. I use 8 bits to present the hidden size of each layer and the value is between 0 and 255. I also provide two options for fitness function and I will compare these two options in the following.

The three basic operations that make up the genetic algorithm are as follows:

1. Select: It defines a solution that is reserved for further replication.
2. Crossover: It describes how to create a new solution from an existing solution.
3. Mutation: Its purpose is to introduce diversity and novelty into the solution pool through random exchange or shutdown solutions.

The basic structure is shown in Figure 6. Our genetic algorithm is realized with “deap” package in python. The code is attached in GitHub and you can find it in “/generate_Chinese_poetry” directory.

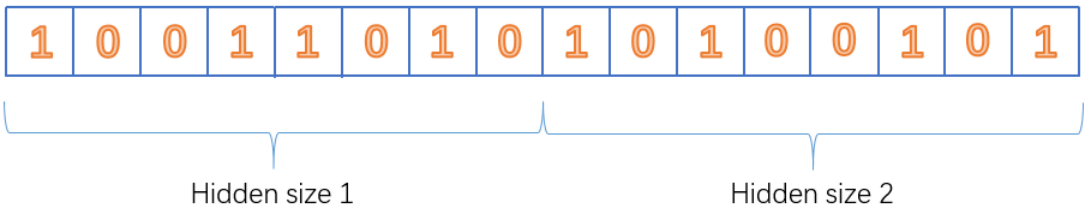

Figure 5. Genetic representation of a solution

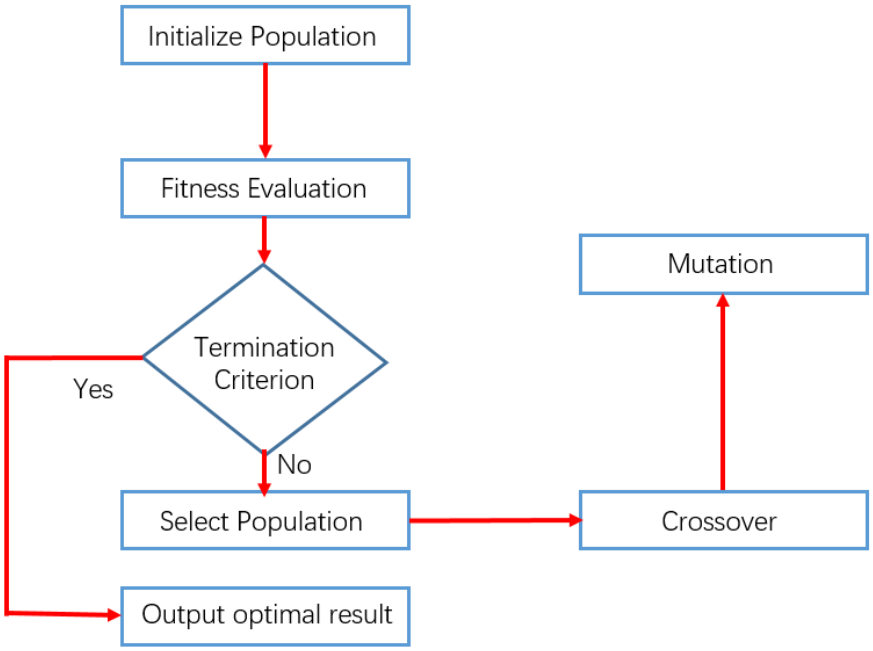

Figure 6. Basic structure of Genetic Algorithm

Here I would like to have a discussion about how to choose the fitness function(s). As is mentioned before, during the training process I split the dataset into two parts, training set, and validation set. In the beginning, I set two fitness functions, loss, and validation loss. I want them to be both minimized. And in each generation, I also let the model output a poem in order to keep tracking of the model's evolution. As is shown in GA_1.ipynb, I finally get the final result as a combination of 25 and 38 for the first and the second LSTM layers' hidden size. And the poem generated from this model seems not as good as the original version which is manually tuned by ourselves. This result is quite counterintuitive. The reason is that the two fitness functions we used at this moment have quite a different tendency with training epochs. The validation loss declined in about first 10 to 15 epochs and increased rapidly, while loss always declined with epochs. Since the genetic algorithm tends to minimize both of them, the optimized result is really in chaos.

As we mentioned before, we need our model in some degree overfitting, so it would not be appropriate to set the validation loss as a fitness function. Then I removed the validation loss and only set the loss as fitness function. The result is shown in GA_2.ipynb. We get the optimized result as 240 and 237 for the first and the second LSTM layers' hidden size. The model is trained for 40 epochs and the output poem is already rhythmic and has a vague topic. Now what I need to do is train a neural network with these optimized parameters with more epochs and the whole dataset. The model is stored in the folder "GA" and some generated poems are shown below:

大漠山扃一日开，小舟飞泊碧瑶台。佳人何日飘为醉，卧待烟阴月上风。

临安略远几时言，萤散峰头数隐路。诸子可怜年下相，三生千里西江去。

楼兰北静入无车，竹粉摇暄尚凭花。金烛忽盐清碧嶂，犹劳藕雨睡晨凉。

石舫俱人对白云，少年修壤病流频。我来坐引人劳说，万象情间箇事愁。

雨散山光水芋礽，聚中时复洗人舟。旧人一曲幷前着，得尽佳心泛泪斜。

独送扁舟五海东，昭关太守百年炎。汉家承学羣恩制，细取深斋寄垒阳。

Based on our results, it seems that using genetic algorithm can really do a good job to find the appropriate parameter combination with much higher efficiency compared with manually parameter tuning. And the result is as good as what we manually tuned. For future work, I suppose we can increase the population and generation number in our genetic algorithm, as well as add more parameters into the genetic representation, for example, batch size and drop out rate, but they will certainly need much more computing resource.

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. A data sample is shown below.

青石板的江南雨巷
烟花时节花开正香
丹青纸上月下荷塘还同当年一样
外婆做的桂花蜜糖
远方飘来阵阵酒香
记忆中的童年时光还在静静流淌
他唱着待月西厢
风流倜傥是岁月贪恋了红妆
又一出地久天长
依旧如今在小城回荡
仿佛听到当初的我们
跟着轻轻地唱
我走在朱雀桥上
抬眼就看见燕子归乡
多年之后又回到最初地方
三月里来油菜花黄
湖面泛着粼粼波光
你画上的戏水鸳鸯从来入对出双
余音绕梁琴声悠扬
烛火的光透过了窗
三三两两戏台之上诉说一段过往
他唱着待月西厢
风流倜傥是岁月贪恋了红妆
又一出地久天长
依旧如今在小城回荡
仿佛听到当初的我们
跟着轻轻地唱
我走在朱雀桥上
抬眼就看见燕子归乡
多年之后又回到最初地方
那一晚华灯初上
是否已将这小城点亮
提笔写一阙满庭芬芳
映着悠久月光
我看着古道斜阳
回首依旧是人海茫茫

看这条乌衣酒巷
一切如常

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.

2. 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. Figure B illustrates the neural network model. There are three bidirectional LSTM layers behind the embedding layer and the dense layer is the last layer. Here we use 'categorical_crossentropy' as loss function.

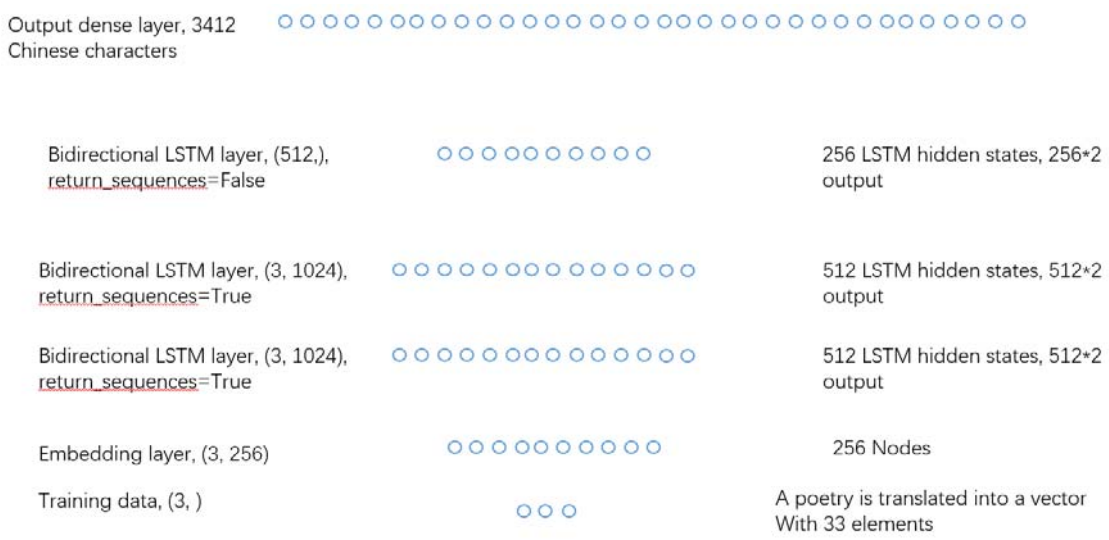

Figure B

3. 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

4. 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.

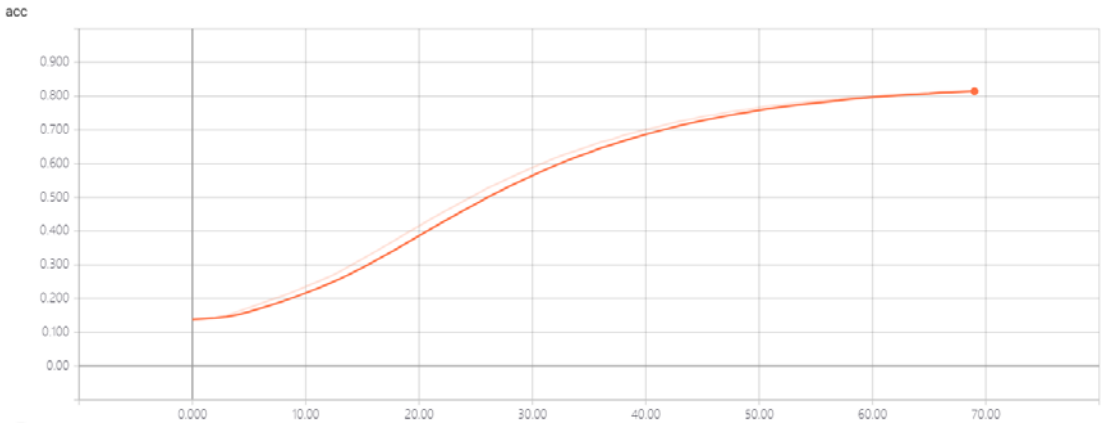

Figure 5. Training accuracy with epochs.

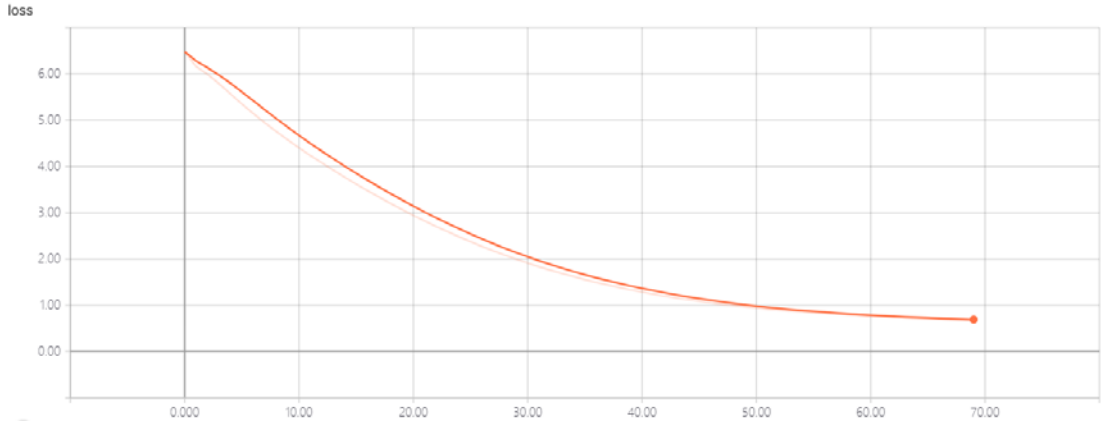

Figure 6. Training loss with epochs.

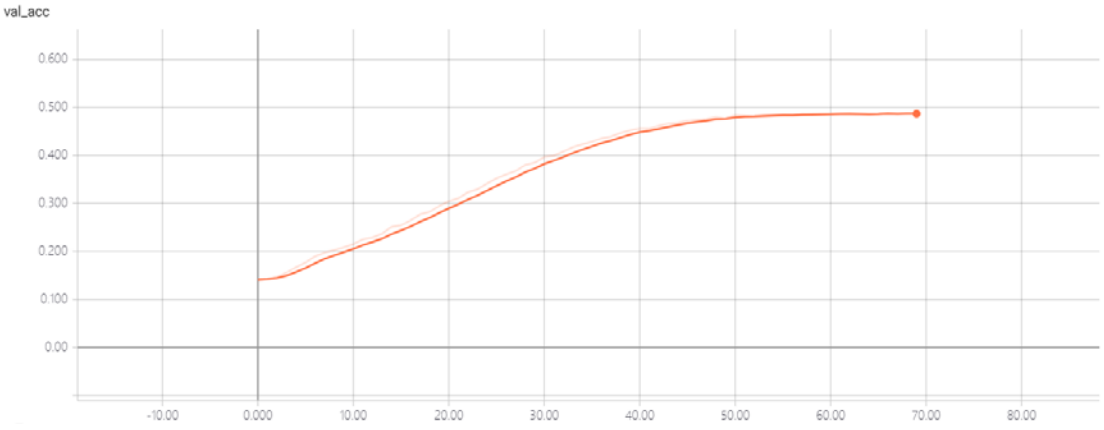

Figure 7. Validation accuracy with epochs.

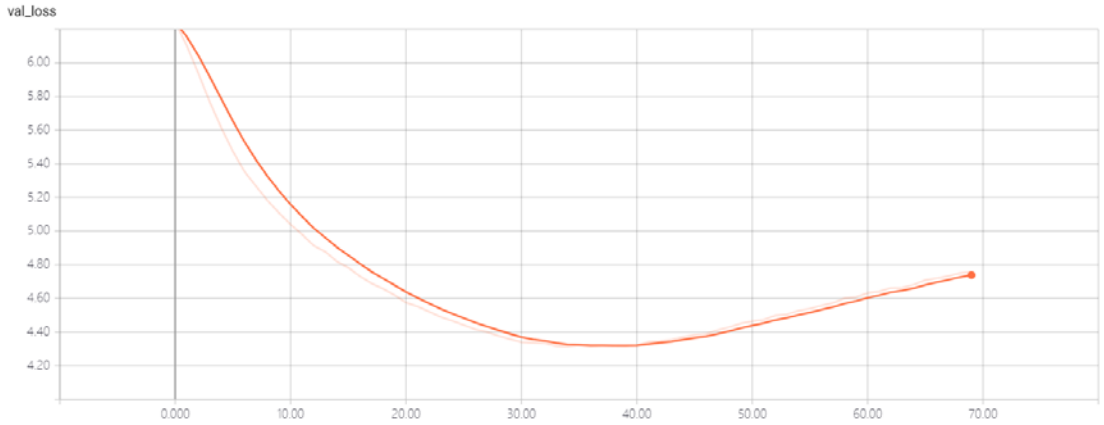

Figure 8. Validation loss with epochs.

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.

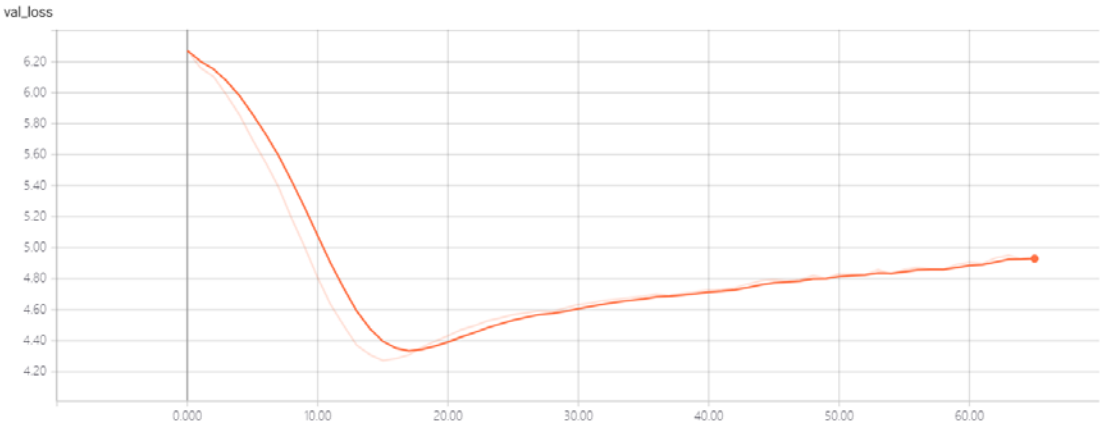

Figure 9. Validation loss of the model with two LSTM layers with epochs.

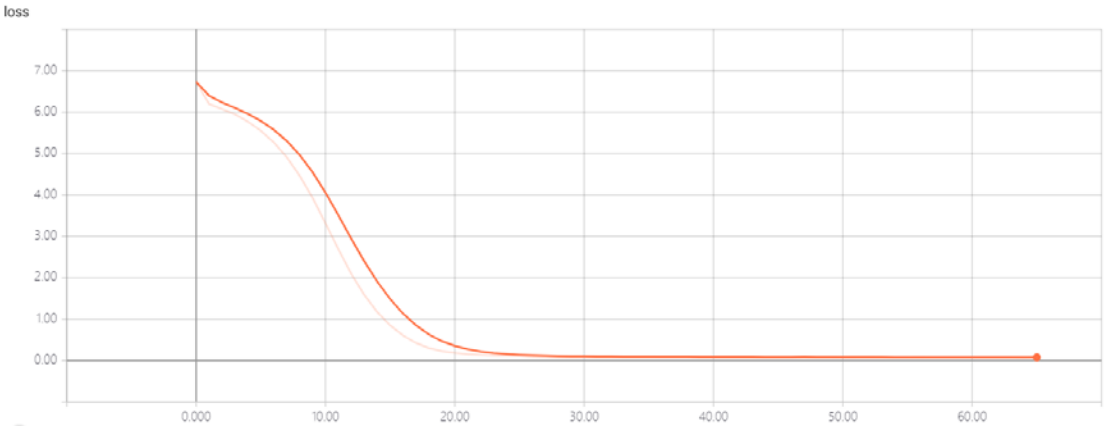

Figure 10. Training loss of the model with two LSTM layers (512, 1024) with epochs.

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
```

5. Annotated Code

The annotated code is shown on Github.

6. 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

2. 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
九皋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

https://github.com/wangshibo1993/An-AI-Writer-Using-RNN-to-Generate-Poetries-and-Lyrics-Course-Project/blob/master/Apr%2016%202019%2011_11%20AM.webm

7. Using Genetic Algorithm as A Systematic AutoML Method

In this part, I still use the genetic algorithm to tune the parameters of our neural network. However, compared with generating poetry, the neural network of generating lyrics has more parameters that can be tuned, which really exceeds my laptop computing capability. Thus, we need to make a compromise, deciding which parameter(s) should be tuned by genetic algorithms and which parameter(s) could be chosen by our experience and fixed during training. Let's first make a conclusion about our parameters:

1. SEQ_LENGTH, which means we use SEQ_LENGTH Chinese characters to determine the next (SEQ_LENGTH+1)th Chinese characters
2. EMBEDDING_DIM, which corresponds to the dimension of the embedding layer before LSTM layer.
3. Three Hidden_size for three LSTM layers

You can see that we have five parameters needed to be tuned. However, during my manually tuning parameters, I have already found some empirical rules about LSTM layer's hidden size. For example, a larger neural network with a bigger hidden size is always good than the one with a smaller hidden size. And for our project, a neural network with three LSTM layers works better than the one with two LSTM layers. Therefore, we just set three LSTM layers with a large enough hidden size for each layer and only use the genetic algorithm to automatically tune SEQ_LENGTH and EMBEDDING_DIM. We set the range of SEQ_LENGTH from 1 to 8 and the range of EMBEDDING_DIM is from 1 to 256. Here we try to use both loss and validation loss as fitness functions and we only train 30 epochs in each generation to limit the validation loss still in the falling edge or just at the beginning of the rising edge, which is easy for the genetic algorithm to optimize in each generation and make both loss and validation loss to be minimized. Our genetic algorithm is also realized with "deap" package in python. The code is attached in GitHub and you can find it in file "GA.ipynb" which is under "/generate_Chinese_lyrics" directory. We got the optimized SEQ_LENGTH as 5 and EMBEDDING_DIM as 169. After that I need to train the

neural network with these optimized parameters with more epochs. The model is stored in the folder “GA” and some generated poems are shown below:

长歌
笔动迷局中个缘分
倾心不豪逃
风声裁扫原
不思着愁字事
雷针观弓疏狂
烟红绾
它号畔红梅去温柔
为你而酿
请记住我
天宝里沉西
为谁布
剑色风起
白衣同梅芬芳
不过轮廓独忍
情三言
侠世已翻过
管你一世的红线
王者将要
声雪炎河走亮
与我以的词容
缠一壶前尘的茶
斟一盏恩怨饮平
平生失家归巢力思欣退假皮
搔酊的刀归
不必今朝与君又香姻缘
被血容变成多情于月
每一片遍告欢堂
濛敌客
降时争

边塞
良海令到他们之尖沧桑却我家阴阳温的新劳
不让何时心求长
当时轮回不相负
羁冥纱树痕
渡儿弯染月如门
这一声过长河
肝画旧骨盘香
滋味小尽一梦恩

何处辜得天涯
江湖名于画
这明月落笔纷飞飒零
桃花瓣年难算
来世一物是谁的脸谱
挥剑冷园炽是江湖有离别
感谢前
过人家分丧
听风雨似罢
天下无疆
一生孑然种英雄
只想看四方
只写一方未减
倾几军中后
天梯传着当时遣
情过所有心不了的风
徒者任君的仙

It seems that the result of the auto-tuned neural network is just as good as what we manually tuned, even better because we find many sentences in the second song "边塞" generated by the new model have clear meaning. I suppose this is because the genetic algorithm gave us a better choice of SEQ_LENGTH than what I manually chose before. A short SEQ_LENGTH can make a partially ordered sequence but a larger SEQ_LENGTH can give us some sentences, maybe short sentences but still longer than Chinese characters' combinations, with clear meanings.

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