**Comparison of Two Chinese Poetry Generators Based on Rule Based Text Generation and RNN Algorithm**

**Name USC email USC ID**

**Jiaqi Gu** [**jiaqigu@usc.edu**](mailto:jiaqigu@usc.edu)

**Nan Zuo** [**nanz@usc.edu**](mailto:nanz@usc.edu)

**Zhiyuan Liu** [**liuzhiyu@usc.edu**](mailto:liuzhiyu@usc.edu)

**Zheng Lou** [**zhenglou@usc.edu**](mailto:zhenglou@usc.edu)

**Division of labor**

Jiaqi Gu:

Nan Zuo:

Zhiyuan Liu: Collect word corpus and raw training materials. Wrote a program to standardize the training data.

Zheng Lou:

**Word count**

**Introduction**

Tang poems and Song poems are two significant parts of China’s cultural. As means of expressing personal emotion, political views, or communication message at festive occasions, Tang poems and Song poems are created at Tang Dynasty and Song Dynasty respectively. Actually, Song poems are more likely to be sung than Tang poems. That is why they are the means of expressing feelings and emotion.

Among many poems styles, five-character quatrain is a very popular one while Song poems have their own style for each tune name. In our paper, only five-character quatrain will be taken into consideration as training data for Tang poems and all popular styles for Song poems because the amount of each Song poems style is small while that of the five-character quatrain style of Tang poems is comparable to all Song poems styles.

Both poems follow specific character pattern, rhyme pattern and tonal pattern. As for character pattern, five-character quatrain have five lines and each line have exact five characters while Song poems have two lines and the number of characters in each line is fixed according the specific tune name. In most cases, both Tang poems and Song poems should rhyme. However, the rhymed lines for Tang poems are the second line and forth line while for Song poems are first line and second line. Moreover, poems should also follow a prescribed tonal pattern. In traditional Chinese, every character has one tone, Ping (level tone) or Ze (downward tone). For Tang poems, only one style, five-character quatrain, has many kinds of tonal pattern while for Song poems, each style, which can be also treated as each tune name, has the fixed tonal pattern.

Therefore, in our paper, given the keywords for each line, Rule Based model is introduced to generate Song poems which has more fixed pattern and Recurrent Neural Network (RNN) model is introduced to generate Tang poems which have more flexible patterns.

We compare the two models by evaluating the quality of generated poems by them based on human evaluation methods.

**Method**

Materials

The purpose of this program experiment is to compare the actual effect of two algorithms, Recurrent Neural Networks(RNN) and Rule Based text generation, in the field of Chinese Poetry Generation. In order to make those two algorithms comparable, we decide to use Chinese Poetry Collection to train both algorithm and input same parameters to run both algorithms.

Since the RNN algorithm cannot learn too many rules in one single layer and it takes a very long time to finish the learning process, we decide to use Tang Poem, a kind of poem which has simple format and each sentence in the poem is strictly in the same length, to train RNN algorithm. Here, we use “QuanTangShi.txt”(Collection of Tang poems, downloaded from the internet) as the training data, which has 30,000 poem in total. And the output of the RNN algorithm will be a Tang poem that strictly follows the format requirement of Tang poem and express the meanings defined by input parameters.

For Rule Based text generation algorithm, considering the rule based nature of this algorithm, it is able to learn and generate the text that have very complicated rules. But unlike the RNN algorithm, which has the ability to improve the quality of the output poem by increase the number of layers it exercised, the rule based algorithm has its own limitation: when generating a word to fill in to a sentence of the poem, the poem format rules and the previously learned probabilities are the only factors that determines which word will be chose. So, considering the advantages and disadvantages of the rule based text generation algorithm, we decide to use Song poem as the training data. Song poem is a kind of poem that has a very complicate format, which is very suitable for Rule Based text generation algorithm. There are 57 kinds of Song poem format and each kind of those format has unique length, rhythm and punctuation. Here, we take “QuanSongCi.txt”(Collection of Song poems, downloaded from the internet) as the raw training data and wrote a program to extract 50 poem of each format to construct the training set.

**Procedure**

Rule-based poem generator

The first step is to collect the training data. We search on the Internet and found a corpus named “QuanSongCi” (the collection of Poem of Song Dynasty). The original version of the “QuanSongCi” is of bad quality because it contains many unrecognized characters and some of the poems’ structures are not qualified. We write a script to filter and screen the file and select about 3000 (545,782 characters) well-formatted poems as our training data. Another training data is about the rhythm. Simply, the last character of current line should have the same or similar pronunciation with the last character of previous line. The rhythmical poem is easy to read and of high aesthetics.

The next step is to train the data. Because the poem generator is rule-based, we need to figured out the rules of the poems. Fortunately, Chinese Ancient Poems are the literature with strict requirement in terms of structure and rhythm. Take Poem of Tang Dynasty as an example, usually it contains 4 lines and each line contains 5 or 7 characters. It’s fixed. While the Poem of Song Dynasty is different and the style is much more free. Usually it contains more than 10 lines and each line is of length ranging from 2 to 9. In this project we will mainly focus on the generation of Poem of Song Dynasty. Besides, each poem has a title. While in terms of Poem of Song Dynasty, the number of title is fixed. There are about 50 well-known title and each title represents a poem structure and rhythm. In other words, different poems sharing the same title will have exactly the same number of lines, the same number of characters of each corresponding line and the same rhythm. Another issue is about the sentence segmentation. Fortunately the sentence structure is fixed given a poem line. For example, if this line contains 7 characters, the first 2 characters form a semantic unit, the middle 2 characters form a semantic unit and the last 3 characters form the last unit. Lines with different length have different segmentation and the lines with the same length share the same segmentation. (Usually, the poem will only contain the semantic unit of length 2 or 3)

Based on these observation, we can train our training data practically. Essentially we use the word2vec method from gensim library. We read the training data line by line, and then segment the lines according to their lengths. We use two list to store the semantic units, one to store the the semantic units of length 2 and another store the semantic unit of length 3. After this job is done, we call the word2vec API and pass these two lists as the parameters. Now the training step is basically done.

In the generation step, we need user to provide us the poem title and the keywords. As mentioned before, the title of the poem will regulate the structure of the poem, and the keywords will regulate the content of the poem. Our idea is really simple. Each single line is generated based on a keyword. So if the user could provide more keywords, then it’s more likely that the poem generated is more accurate. However this raises another problem. As mentioned before, the Poem of Song Dynasty usually have more than 10 lines and it’s not friendly to require user to provide so many keywords. Our solution is the keyword extension. If the keywords are not enough, then we will extend the keywords by selecting the alternative ones from the corpus according to their frequency. This corpus is built in training step and it only involves some trivial statistics work.

When we try to generate each line of the poem with a keyword, The algorithm is like DFS. We will try to insert it into some potential positions. For example, the current line is of length 7 and the keyword is of length 2, then the potential position of this keyword is 0 and 2. We will first try to insert it in position 0 and then we try to fill the remaining positions. If failed, we try to insert the keyword in position 2 and fill the previous gap and next gap. When we need to generate the new semantic units after we settle down the keywords, we use word2vec to select top 100 most related ones as our candidate semantic units and try one by one. When we insert the last semantic unit, we need to pay attention of the rhythm to make sure the rhythm is consistent. While unfortunately, sometimes we can not generate a qualified line because the last semantic unit cannot satisfy the rhythm contains. If this happens, the program will use the “ForceBuild” mode, which means the construction of the line will not consider the rhythm. It’s still a qualified poem but lose a little bit aesthetics.

The flow char is as follows:

Collection training data:

collect the poem data

->

Screen and filter the poem data and only select the well-format ones

->

Collect the rhythm set, build the rhythm dictionary

Training step:

Read the poem data

->

Sentence segmentation and build semantic units list

->

Build word2vec model

->

Count the frequency of the semantic units

Generation step:

User provides the title and keywords

->

If keywords is not enough, then extend

Generate each line using a keyword in DFS way

->

Try to insert the keyword in the potential positions

If failed, then use ForceBuild mode to regenerate line without consider the rhythm

Add punctuation to the lines and return the poem

Planning-based Neural Network

In this section, we talk about our implementation of planning-based RNN poem generator, which adopts the ideas from the paper [Wang et al. 2016].

According to the original paper, the planning-based poem generation includes two stages: poem planning and poem generation. In the first stage, four keywords are extracted (or expanded) from the input text as a guideline for poem generation. In the second stage, the poem generator will generate a quatrain sentence by sentence, and each sentence is generated according to both a corresponding keyword and the preceding sentences it has generated. We adopt the ideas of this paper since the author claims that it can effectively ensure the coherency and semantical consistency of the generated poems, which we deem as important qualities of good poems.

At its core, our implementation is based on the model from that paper, which has modified the attention-based RNN encoder-decoder framework (Bahdanau et al., 2014). To generate a new quatrain sentence, we need a corresponding keyword generated from the poem planning step and all the preceding sentences that we have generated. We first encode the keyword and the preceding text into two sequences of hidden states respectively by using bi-directional Gated Recurrent Unit (GRU). Then we concatenate the last forward state and the first backward state of the keyword as the first attention state, and use the hidden states from the preceding text as the remaining attention states. Finally, a GRU decoder will use those attention states to generate a new sequence as the output.

Nevertheless, our implementation still differs a lot from that of the original paper due to the practical situation we are facing. One of the most prominent modifications we have made is that we encode and decode Chinese poems word by word, whereas most of the relevant papers (including [Wang et al. 2016]) encode Chinese text character by character. We decided to break with tradition because we think a word is the minimal unit of a language that maintains semantical information, and it makes more sense to use words rather than characters to process a language. However, one of the difficulties of doing this is that Chinese is quite different from many other languages in that its word segmentation is much more difficult, and the distinct word set of ancient Chinese poem makes it more challenging to most open source word segmentation libraries, which are based on modern Chinese corpus. Moreover, the number of words in the poem corpus we use can be as large as several millions, and matrix multiplications with that high dimensionality are far beyond our available computation resources.

To address those problems, we decided to use the 49998 most frequent words in our corpus, and restrict the training data to the quatrains that only use those limited words. We think doing this is reasonable because it is sufficient to use the most frequent words to generate poems that modern Chinese can appreciate, while many other words in our corpus are too recondite to most people. On the other hand, we improve the word segmentation of Jieba open source library by enforce segmentation at certain positions of a sentence according to the structural and tonal patterns of classical poems. For example, a sentence with 7 characters will be split into tokens with length 2,2,3 before segmentation. In addition, we also use ShiXueHanYing, a poetic phrase taxonomy as a reference for further segmentation.

Our entire preprocessing steps are as follow. Firstly, we parse the poem corpus used by XingxingZhang’s rnnpg project (<https://github.com/XingxingZhang/rnnpg>) line by line and maintain the 284308 poems with best qualities. Then we use an improved word segmenter to count the words in all those poems and get the most frequent 49998 words as vocabulary. By using those vocabulary words, we selected 8436 quatrains from all the poems, and for each quatrain, we gather all its words except for those in our stopword set as keyword candidates. Then we run TextRank algorithm (Brin and Page, 1998) over those candidate words to rank them. Finally, we use the word ranking we get to extract a single keyword from every sentence in our selected quatrains, so that a training data set can be generated by combining poem sentences, extracted keywords and preceding sentences.

We got 33736 pieces of training data for our attention-based RNN encoder-decoder. We group the training data according to the lengths of sentences and preceding texts, so that every minibatch will contain data with similar lengths during the stochastic gradient descent. To be specific, we used the AdaDelta optimizer in TensorFlow library with learning rate 0.002 and decay rate 0.97. We set the size of each minibatch as 128, and the size of all RNN cells in our model as 512. For word embedding, we trained a word2vec model using Gensim library over the entire corpus as the initial embedding matrix, and set the corresponding tensor as trainable to allow TensorFlow improve it during training. Besides, we also use the labels of our training data to train another RNN model for keyword expansion, in which we use a 4-layer LSTM cell in TensorFlow.