LIGN 167 Final Project Report

Leyi Shang, Yichen Shi, Gaotong Wu

Theme: Chinese Poetry Generation with Deep Learning Approaches

I. Introduction

Chinese poems are of over two thousand years of history, which were written to praise heroic characters, beautiful scenery, love, friendship, etc. Different kinds of Chinese classical poetry follow some specific structural, rhythmic and tonal patterns. For our project, we are generating poems with the form quatrain: four lines, seven characters each line following a particular tone pattern.

This generator allows the users to enter conceptual words and automatically generates a poem according to the given concept. The trained model will first expand or extract keywords from the user input query depending on the input. If the input query contains more than 4 words, the model will extract 4 keywords from the query, whereas if the input query has less than 4 words, the model will expand to 4 keywords.The keywords serve as subtopics for the lines. This process is called **poem planning**. The next process is **poem generation**, which takes the keyword corresponding to that line and the preceding lines as inputs to generate the next line.

We are developing based on open-source codes. The results from the open-source code rise imperfection: the poems generated do not strictly follow the sub-topic of each keyword input and the whole sentence of each line does not make much sense. Hence we modified the model so that it produces more understandable poems. Details of the modification are discussed in the "troubleshooting" section in the "Model" part.

Note: In Chinese, "character" and "word" are two different concepts; a word consists of a number of characters (useful analogy: a word consists of a number of alphabet letters in English). Specifically, sometimes one character already form a word while sometimes we need several characters to form a word. Those characters put together have to make sense to form a word. That is, if you just randomly put a bunch of characters together, they do not necessarily make sense and thus do not form a word. In the figures below, words, each formed by one or more characters, are separated with spaces (for "*hints*" and "*keywords*") in order for non-Chinese speakers to recognize more easily. However, when we write sentences, we do not need the spaces. Therefore, each line of the poems generated does not have spaces.

These backgrounds are helpful for you to understand our troubleshooting parts so make sure that you have a clear mind of the concepts! But you don't need to completely understand Chinese.

II. Model:

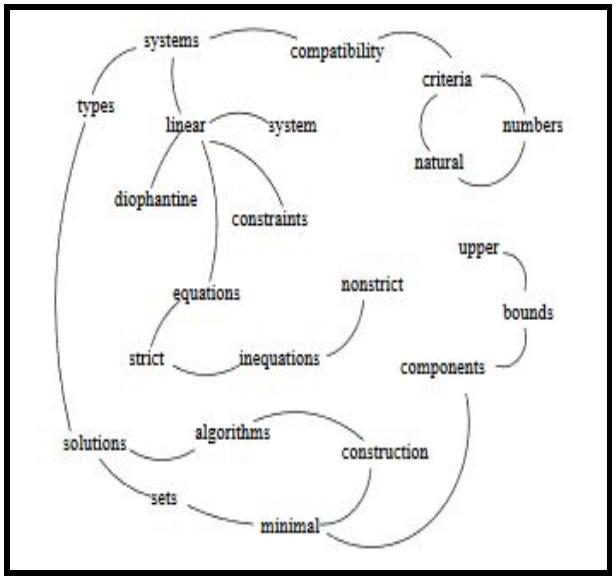
2.1 Poem Planning:

- *Keyword extraction*: we extract or expand four keywords that are most relevant to the input context. The open-source code uses TextRank algorithm, a graph-based ranking algorithm to complete this task.

The figure illustrates the keyword generated when the input query has more than 4 words

The user input five words "*Pine, Spring, River, Eagle, Peach Blossom*" and it extracts to four keywords "Peach Blossom, Spring, Pine, Road".

The figure below is a graph indication of TextRank algorithm: word is represented by a vertex in the graph and edges are added according to their co-occurrence;



$$S(V_i) = (1-d) + d \sum_{V_j \in E(V_i)} \frac{w_{ji}}{\sum_{V_k \in E(V_j)} w_{jk}} S(V_j)$$

The score of each word is calculated by the equation above, where w_{ij} is the weight of the edge between node V_j and V_i, $E(V_i)$ is the set of vertices connected with V_i , and d is a damping factor that usually set to 0.85 and the initial score of S(Vi) is set to 1.0.

- *Keyword expansion*: number of keywords from the input query is less than the number of lines. The open-source code uses Knowledge-based method to complete the task. It uses encyclopedia as extra knowledge sources and the word has to satisfy the following requirements:
- (1) the word is in the window of [-5, 5] around the keyword
- (2) the part-of-speech of the word is an adjective or noun
- (3) the word is covered by the vocabulary of the poem corpus.

The user only gives two input hints "*Spring, Peach Blossom*". The method expands to four keywords "*Peach Blossom, Today, Nobody, Spring*".

2.2 Poem Generation: the open source uses an **encoder-decoder** network model

- **Encoder:**

• a keyword k which has T_k characters, i.e.$k = \{a_1, a_2, \ldots, a_{Tk}\}$,
and the preceding text x which has T_x characters, i.e.$x = \{x_1, x_2, \ldots, x_{Tx}\}$;
• encode k into a sequence of hidden states$[r_1 : r_{Tk}]$, and x into $[h_1 : h_{Tx}]$;
• concatenating the last forward state and the first backward state of $[r_1 : r_{Tk}]$: $r_c = \begin{bmatrix} \overrightarrow{r_{Tk}} \\ \overleftarrow{r_1} \end{bmatrix}$
• Set $h_0 = r_c$, and the sequence of vectors $h = [h_0 : h_{Tx}]$;
• Notes: the first line is actually generated from the first keyword and the length of the preceding text is zero, i.e. Tx = 0, then the vector sequence h only contains one vector, i.e. h = [h0].

- **Decoder:**

- an internal status vector $\overrightarrow{s_t}$, context vector ct and previous output
- $\overrightarrow{v_t}$ is generated that can maximize the probability: $y_t = argmax_y P(y|s_t, c_t, y_{t-1})$
- After each prediction, $\overrightarrow{s_t}$ is updated by: $s_t = f(s_{t-1}, c_{t-1}, y_{t-1})$
- $\overrightarrow{c_t}$ is calculated by: $c_t = \Sigma_{j=0}^{T_h-1} a_{tj} h_j$,where h_j is the j-th hidden state in the encoder's output;
- The weight $a_{tj} = \frac{exp(e_{tj})}{\Sigma_{k=0}^{T_h-1} exp(e_{tk})}$, where: $e_{tj} = v_a^T tanh(W_a s_{t-1} + U_a h_j)$

2.3 Troubleshooting and modification:

Since there is already a usable open-source model, we chose to refine the algorithm based on their approach. We found three problems with the open-source code:

Firstly, the author uses gensim word2vec model to train a model based on poem lines written by ancient Chinese poets. Then he uses this trained model together with randomly initialized vectors as the character embedding. However, the word2vec model trained on poems recognize words rather than characters (Chinese differs from English in that different characters combine to form words and different combinations carry different meanings). Out of the 6000 characters stored in the dictionary, only one character 花 was found in the model. Hence the original word embedding does not encode any contextual meaning of the character and perform poorly.

Secondly, at the start of the poem generation, the decoder only sees the start symbol and one keyword, hence the first sentence generated by the model is always the same. Also, if the keywords are not tightly correlated, the sentences seem to have limited correlations.

Thirdly, the open-source follows a random selection strategy regardless of the probability distribution. So the poems produced are gibberish and do not make sense to human readers. However, when following the original paper's greedy approach and picking out the most likely character all the time, the same character appears again and again.

To address the three issues, we came out with three approaches tackling the problems one by one.

Load Pre-trained Word2Vec Embedding

Though characters are often used differently in modern times compared to ancient times, their original meanings are still rich in some Chinese novels. Hence in order to enrich the context encoded in the embedding and improve the model's performance, we found a word2vec model pre-trained on Chinese novels and parsed feature vectors stored in the model as the new embedding.

The word2vec model we found has a dimension 300 so we adjusted all dimension related constants to 300 accordingly. After the modification 5073 characters in the dictionary have a match in the new embedding. So % of our embedding matrix now encodes rich meaning.

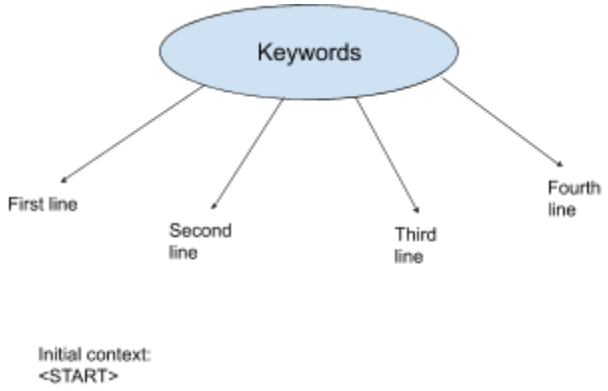
General Keyword As Hint

To make sure the generator knows more about the context, in addition to using 4 keywords corresponding to 4 lines of poems, we generate 5 keywords during the planning stage, randomly select one among them and append that selected keyword to the context vector as a hint. So we assume that the decoder has already generated the hint and is writing following the hint. Then after the whole poem is generated, we remove this hint from the poem.

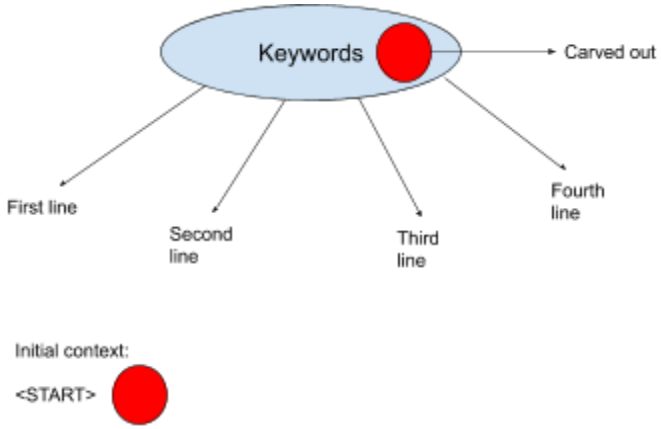
By taking this approach, the first sentence will vary according to the context and the following lines will be more correlated. The resulting effect is that in addition to the four different keywords, we have a general hint affecting the whole poem.

Assuming c_i is the initial context vector, originally c_i = <START>, now c_i = <START><hint>.

Original:



After modification:



Line Generation

A poem has four lines and a line is composed of seven characters. We split the line generation to four parts: first character, second and third character, fourth and fifth character, sixth and seventh character. In Chinese, two characters often form a word, hence generating two characters together may improve the fluency of the poem.

First character

Generating the first character in a line is a great challenge. At this stage, the decoder only reads a start character. Even though there are the context vector and the keyword vector, the model still tends to distribute an overly large probability to certain words in the vocabulary. Therefore selection based on highest probability will result in similar poems produced regardless of the keyword. In order to address this issue, instead of selecting the most possible character, we randomly sampled 10 characters.

1. Compute the cumulative sum array of the probability list (np.cumsum() is used to achieve this)
2. Times the last entry of the cumulative sum array by a random number
3. Iterate through the cumulative sum array and choose the index of the first element with a sum entry greater than the random number
4. Store the character in the character array
5. Compute the score of this character. Assuming we are looking at the j^{th} element in the probability list, its score is calculated by the following formula:
   $score_j$ = -log(prob_list[j])
6. Repeat 10 times and store the characters and scores in the array
7. Iterate through the score array, choose the minimum score and append the character with the minimum score

By doing so, we introduce randomness to the generator model to avoid repeatedly selecting the same character while keeping probability as the dominant factor during selection.

Two characters together as a word

Once the first character is generated, the rest of the line is produced in batches of two. Here we use a simple beam search approach.

1. Select the 10 most likely characters
2. Explore all 10 choices by concatenating them with the context vector, feed them to the decoder, select the most likely character following them separately

3. Set the initial score to 1 for all batches. At each stage, the new score is calculated by timing the original score by the new negative log
4. After exploring all 10 possible choices, assuming we are looking at i^{th} batch, which contains the j^{th} element and k^{th} element in the probability list. i^{th} score is **$score_i$ = (-log(prob_list[j])) * (-log(prob_list[k]))**
5. To avoid picking a used character for the second character, keep retrieving the next most likely character if the character is used or repeats the first character in the batch. Assuming the new selection is the m^{th} element in the probability list, the score is **$score_i$ = (-log(prob_list[j])) * (-log(prob_list[m]))**
6. Retrieve the index of the minimum score in the score array and find the corresponding characters in the character arrays
7. To avoid used character for the first character, keep retrieving the next lowest score character without repetition
8. Append the pair to the context

By doing so, we select the most possible pair out of all 10 batches while minimizing the possibility to repeat.

III. Datasets:

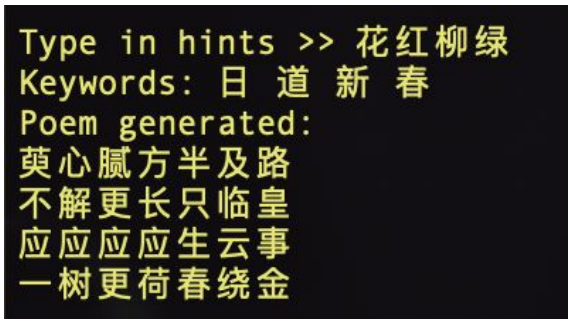
For the dataset, we just directly used what the paper proposed; the original data come from: https://github.com/chinese-poetry/chinese-poetry.
It contains 55,000 poems in Tang Dynasty and 260,000 poems in Song Dynasty. The author of the open source code preprocessed the poems so that the poet names and titles are removed and all of them are combined in one document. All the poems in the training set are first segmented into words using a CRF based word segmentation system. It extracted a sequence of 4 keywords for every quatrain. From the training corpus of poems, it extracted 72,859 keyword sequences, which is used to train the RNN language model for keyword expansion. For knowledge-based expansion, it used Baidu Baike (an equivalence of Wikipedia in China) and Wikipedia as the extra sources of knowledge. After extracting four keywords from the lines of a quatrain, we generate four triples composed of (the keyword, the preceding text, the current line), for every poem.

IV. Results:

The grading for generated poem is based on four aspects:

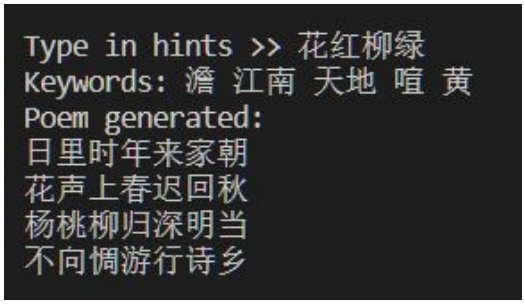
- Poeticness: Does the poem follow the rhyme and tone requirements?
- Fluency: Does the poem read smoothly and fluently?
- Coherence: Is the poem coherent across lines?
- Meaning: Does the poem have a certain meaning and artistic conception?

Since there is no data characteristics in poems to evaluate these aspects by computer, we use human evaluation for grading the generated poems. The two poems in yellow are the poems before modification, and the two poems in white are the ones after modification. We put these poems together to compare the modification result. We built up a google form showing these four poems to gather the score data.
Here is the google form where we collected the score data:
https://forms.gle/eT8T7EgFFPN2DS7s8
Here is the excel form containing summary of responses to our google form survey:
https://docs.google.com/spreadsheets/d/1YEBNhaB3d49hFQ5644HHIXcHXhjo-evRD4jXfsDyevE/edit?usp=sharing



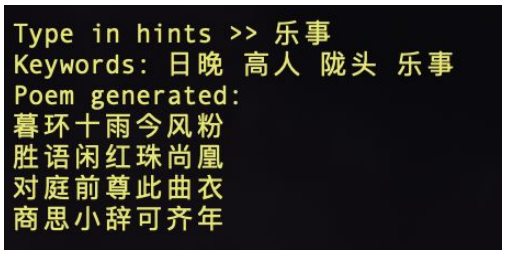
Hints: flower gets red and willow gets green
Keyword: day, road, new, spring
萸心(Cornelian cherry heart)腻(greasy方半及路(near the road)
不解更长(don't know how long)只(only)临皇(near emperor)
应应应应(ying ying ying ying)生云事(make some cloud)
一树(a tree)更荷(more lotus)春(spring)绕(around)金(gold)

Poeticness	Fluency	Coherence	Meaning	Overall
2.429	2.286	2.5	2.5	2.429



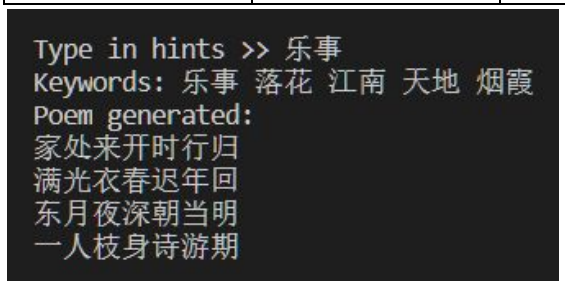
Hints: flower gets red and willow gets green
Keyword: tranquil, Jiangnan(place), world, noise, yellow
In a day of the year I go back home in the morning,
The flower in Spring look like it will turn to Fall very late.
Poplar, peach and willow make the bright way to home,
I travel to countryside without any worries.

Poeticness	Fluency	Coherence	Meaning	Overall
3	3.357	3.357	3.5	3.304



Hints: joy
Keyword: day and night, supreme, Longtou, joy
暮(twilight)环(ring)十雨(ten rain)金风(gold wind)粉(powder)
胜语(victory)闲红(idle red)珠尚凰(pearls like phoenix)
对庭前(Facing the cour)尊(honor)此曲衣(this song and clothes)
商(trade)思(thinks)小辞(take leave)可(can)齐年(be a year)

Poeticness	Fluency	Coherence	Meaning	Overall
2.571	2.857	2.929	3.071	2.857



Hints: joy
Keyword: joy, falling flower, Jiangnan(place), world, haze
I went out from home,
I went back feeling great, just as it is in bright spring.
East moon in dark night, but it will turn bright in the morning,
I will go to poem journey alone again.

Poeticness	Fluency	Coherence	Meaning	Overall
3.429	3.429	3.429	3.571	3.465

V. Conclusion:

From the “Result” section, we conclude that after the modification the generator produces better poems. However, there is still large room for improvement since the poems only reach an average of 3.5.

We plan to further improve the model by adding an embedding layer and fine-tuning the embedding matrix during training. Also, we plan to further improve the planning stage by using alternatives to word2vec.

Reference

Code developed based on:
https://github.com/DevinZ1993/Chinese-Poetry-Generation
https://github.com/Disiok/poetry-seq2seq
(Modified char2vec.py, plan.py and generate.py)

Embedding model downloaded from:
https://github.com/Embedding/Chinese-Word-Vectors

Other references:
https://ai.tencent.com/ailab/media/publications/emnlp-18-poem-final-hanson(poster).pdf
http://www.cse.scu.edu/~mwang2/projects/NLP_ChinesePoetryGenerator_18f.pdf
https://slideplayer.com/slide/14827220/
https://cs.uwaterloo.ca/~mli/Simon_Vera.pdf
https://nlp.stanford.edu/projects/chinese-nlp.shtml