Text Steganography with High Embedding Rate: Using Recurrent Neural Networks to Generate Chinese Classic Poetry

Yubo Luo
Department of Electronic Engineering
Tsinghua University
Tsinghua National Laboratory for Information Science and Technology
Beijing, China 100086
luoyb14@mails.tsinghua.edu.cn

Yongfeng Huang
Department of Electronic Engineering
Tsinghua University
Tsinghua National Laboratory for Information Science and Technology
Beijing, China 100086
yfhuang@tsinghua.edu.cn

ABSTRACT

We propose a novel text steganography method using RNN Encoder-Decoder structure to generate quatrains, one genre of Chinese poetry. Compared to other text-generation based steganography methods which have either very low embedding rate or flaws in the naturalness of generated texts, our method has higher embedding rate and better text quality. In this paper, we use the LSTM Encoder-Decoder model to generate the first line of a quatrain with a keyword and then generate the following lines one by one. RNN has proved effective in generating poetry, but when applied to steganograpy, poetry quality decreases sharply, because of the redundancy we create to hide information. To overcome this problem, we propose a template-constrained generation method and develop a word-choosing approach using inner-word mutual information. Through a series of experiments, it is proven that our approach outperforms other poetry steganography methods in both embedding rate and poetry quality.

KEYWORDS

text steganography; poetry generation; recurrent neural networks

ACM Reference format:
Yubo Luo and Yongfeng Huang. 2017. Text Steganography with High Embedding Rate: Using Recurrent Neural Networks to Generate Chinese Classic Poetry. In *Proceedings of IH&MMSec '17, June 20–22, 2017, Philadelphia, PA, USA,* , 6 pages.
DOI: http://dx.doi.org/10.1145/3082031.3083240

1 INTRODUCTION

Steganography approaches hide the existence of secret information and are usually classified by the type of multimedia carries that they are based on. As text is the most commonly used media carrier, text steganography is of great value. Generally, text steganography is mainly divided into two categories: format-based methods and content-based ones.

Format-based approaches hide secret information by making changes to the format features of text cover. Early works are based on embedding space characters in text [9], altering the space between words [8] or lines [1], etc. Content-based methods are usually based on lexical, syntactic or semantic manipulations. Most early approaches modify the semantic contents of existing texts to hide information, such as using synonym replacement strategy [10], changing the structures of sentences [5], etc. Then, many text-generation-based steganography methods were proposed, such as generating a random sequence [16] or considering context-free grammars [3].

With regard to poetry-based steganography, An information hiding algorithm based on poetry generation was proposed in the literature [18], and Liu et al. [11] proposed a segment-based method. But, they both choose words randomly during the generation process, totally ignoring word collocation and the relationship between lines. Thus, the generated poems lack central ideas and have high possibility of attracting suspicions. Luo et al. [12] proposed an approach based on Markov model. It produces better Ci-poetry but its embedding rate is quite low.

More recently, deep learning methods have emerged as an effective discipline, which considers the poetry generation as a machine translation problem and thus an encoder-decoder model can be used to generate poetry. Zhang and Lapata [19] compressed all previous information into a vector to guide the generation. Wang et al. [13, 14] and Yi et al. [17] use bidirectional recurrent neural network (RNN) with attention mechanism to help capture long time dependency between poetry lines. Wang et al. [15] proposed a planning-based method to ensure the coherence of generated poems.

However, if quatrain generation based on neural networks is applied to steganography directly, the quality of generated poems will decrease sharply, because we have to create redundancy by arranging several candidates for each char-position. This will cause a situation where characters from certain positions which should be regarded as a word cannot be treated as a word. For example, the first two characters of a quatrain line should always be treated as a word. If we happen to choose two characters which never appear within one word, this will definitely jeopardize the quality of generated quatrains.

To handle this issue, we propose a template-constrained generation method. As we know, all quatrains have fix rhythmic pattern, so we can segment all quatrain positions into two-char or three-char prosodic blocks according to rhythmic pattern, then we can obtain a template to guide the generation process. In each block, characters should be regarded as a word rather than separate characters. In order to make sure that randomly selected characters within one block can be treated as a word, we use inner-word mutual information to help us choose candidate characters. In this way, we can ensure that the chosen characters in one block can be treated as a natural word.

The contribution of this paper is two-fold. First, we introduce neural-based poetry generation to steganography, which has much higher embedding rate than previous poetry-based steganography methods. Second, we propose a template-constrained generation method and a word-choosing approach using inner-word mutual information, to alleviate the quality decline caused by information hiding.

2 BACKGROUND

In this section, we first give a brief overview of neural-based poetry generation framework, and then describe how we implement encoder-decoder models to build the poem generator.

2.1 Encoder-Decoder model

Classical Chinese quatrains have strong semantic relevance between adjacent lines. The four-line structure often follows the "beginning, continuation, transition, summary" process [7]. RNN encoder-decoder is well capable of learning the dependency between quatrain lines.

As shown in Figure 1, the entire framework is an attention-based model. Attention mechanism allows a model to automatically find the most relevant parts in the input sequence, then to generate the target sentence [2]. The encoder model first converts the input sequence into hidden states which contain the semantic meaning of each character of the input. Then the decoder model takes hidden states as input to accomplish the generation of target sequence. The task of attention mechanism is to help the decoder find most relevant input by taking into account the decoder's current state and the encoder's hidden states. Thus, the generated target sequence can well focus on specific relevant characters.

The simple RNN model has a severe problem, hard to remember the long time dependency [14]. It is not suitable to deal with quatrains which have four lines. To overcome this issue, we adopt bidirectional Long Short Term Memory (LSTM) model for the encoder and another LSTM for the decoder [13–15, 17]. In this way, the encoder-decoder model can well capture long distance patterns.

2.2 Poem generator

We use attention-based encoder-decoder model to build two modules (word-to-line and line-to-line) to generate a complete poem. A word-to-line module takes a word as input and then ouput a line, but a line-to-line module produces a line according to one line or multi-lines. Yi et al. [17] proposed a three-block system to generate a whole quatrain that was reported to work well in learning the rhyme automatically. However, the structure seems too complicated (four separate models in total). Wang et al. [15] only uses one model for both keyword input and line input, which will not work well when the input is a short word, as most training pairs are line-to-line pairs. Thus, we adopt two separate models, one for word-to-line and the other for line-to-line.

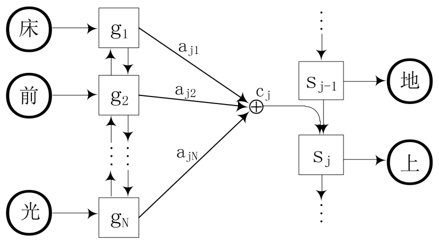

Figure 1: Encoder-Decoder model.

When generating a poem, we first use the word-to-line module to generate the first line of the poem according to the keyword that a user inputs. Next, the line-to-line module takes the first line as input to generate the seconde line, and then to generate the third and forth line according to all previously generated lines.

2.2.1 Line-to-line Module. We first introduce the line-to-line module. We assume that we have already obtained the first line l_1. Then, we will input l_1 to generate the second line l_2. The inputs for generating l_3 and l_4 are $l_1 l_2$ and $l_1 l_2 l_3$, respectively.

Let $X = (x_1, x_2, \dots, x_N)$ be the input line. At time i, the hidden state for encoder is computed as Eq.(1)-Eq.(2):

$$h_i = f_{LSTM}(h_{i-1}, x_i), \quad (1)$$

$$g_i = [h_i; h'_i], \quad (2)$$

where h and h' are the forward and backward hidden states in the encoder, x is the input, and g represents the combined hidden state for the encoder.

As mentioned in Section 2.1, we use an attention-based bidirectional encoder-decoder model to build the line-to-line module. After the hidden state g of the encoder is calculated, it merges with the attention vector a_{ij} into context vector c_j. It is a_{ij} that helps the decoder to find the most relevant parts from the input. The context vector c_j is calculated by:

$$c_j = \sum_{i=1}^{N} \alpha_{ij} g_i, \quad (3)$$

where a_{ij} is the 'attention', that at j_{th} generation step the decoder has, to the i_{th} character of the input. The attention vector α_{ij} is computed by:

$$\alpha_{ij} = \frac{exp(e_{ij})}{\sum_{k=1}^{N} exp(e_{jk})}, \quad (4)$$

$$e_{ij} = v_a^{\top} tanh(W_a \cdot s_{j-1} + U_a \cdot g_i), \quad (5)$$

where s_{j-1} is the hidden state of decoder at the $(j-1)_{th}$ generation step; v_a, W_a and U_a are matrices that will be optimized in the training process [6, 14].

Finally, at the j_{th} generation step, the output y_j and hidden vector s_j are formulated by:

$$s_j = f(s_{j-1}, y_{j-1}, c_j), \quad (6)$$

$$y_j = p(y_{j-1}, s_j, c_j), \quad (7)$$

where, f(·) and p(·) are update functions determined by the model structre [14]. The output vector y_j is what we mainly work on to hide secret information, and we will discuss it detailly in next section.

2.2.2 Word-to-line Module. Basically, the encoder-decoder models for line-to-line and word-to-line are the same. During the training process, the input sequence, no matter it is a word or a line, will be converted into the same vector space. Because current generation systems are all based on characters, keywords are much shorter than lines. Besides, the majority of training data are line-to-line pairs, if we only use one model to deal with both keyword input and line input, it will not work well when the input is a short word [17].

Thus, we train word-to-line pairs in a separate word-to-line module. Of course, if we start the training of word-to-line module from scratch, it will not capture enough semantic relations, duo to the lack of word-to-line pairs. In practice, we start the training of word-to-line module from a pre-trained line-to-line module.

3 STEGANOGRAPHY SCHEME

The steganography part begins when the poem generator produces Y, the vector set of a poem line. Let us denote the line as $Y = (y_1, y_2, \ldots, y_N)$, where N is the character number this line contains. Each vector y_i contains the information about the probability of each character in the vocabulary being the generated one for a certain char-position. Thus, we can hide information by selecting several candidates for each char-position, coding them, and finally choosing one candidate according to the secret message.

3.1 Framework of poetry steganography

There are three steps in poetry steganography, as shown in Figure 2. The first step is to set parameters that regulate poetry generation. The second step is to use the poem generator discussed in section 2.2 to generate poetry lines, and we hide information in step 3, by coding candidates and choosing one character based on secret message.

Different from Ci-poetry which has many tonal patterns, 5-char and 7-char quatrains both only have four common used patterns [7], from which we randomly choose one pattern for generation. *Info* is the secret message that we want to hide in the stego-quatrain. In practice, *Info* will be converted into bit stream to guide character selection. *Key* is the keyword we need to input into the word-to-line model to generate the first line. *Size* is the size of candidate pool, in other words, the maximum number of candidates we can select for coding. Obviously, *Size* is the determinating factor of the embedding rate. The larger *Size* is, the higher the embedding rate will be.

In step 2, poem generator produces a poem line Y which contains N vectors. N is the number of characters contained in one line. y_i contains the information about the probability of each character in the vocabulary being the next generated one. For example, for $y_i = (p_1, p_2, ..., p_{|V|})$, where $|V|$ is the vocabulary size, p_j means the probability of the j_{th} character in the vocabulary being next generated one.

Step 3 is the most critical part of our steganography system. For each y_i, we first filter out all characters that do not meet the

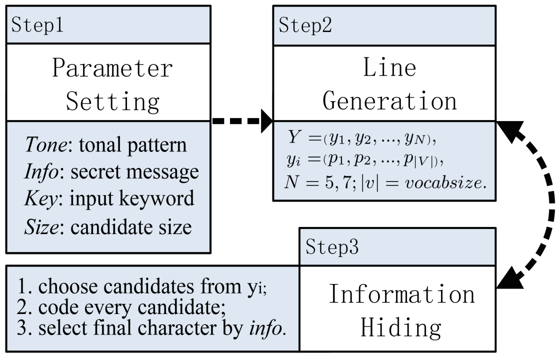

Figure 2: The framework of steganography system.

tone regulation, then select the most probable top-*Size* candidates and finally code these candidates by assigning each one a code (e.g. Huffman code). Next, the character whose code matches the beginning bits of *info* will be the final chosen character. In this way, the matched beginning bits of *info* are hidden in this character.

3.2 Improved char-choosing method

However, there is a severe problem during char-choosing process. If we choose characters for each char-position separately, the generated line is very likely nonsense, especially when *Size* is large. Because, the larger *Size* is, the more likely characters in certain positions can not be treated as a word. Next, we will discuss how to solve this problem.

3.2.1 Templated-constrained generation. When people read a 5-char quatrain line, they will automatically segment the line into the form of *cc*/*ccc*, and *cc*/*cc*/*ccc* for a 7-char quatrain line. In fact, Yu et al. [18] and Wang et al. [11] built their poetry steganography system based on this segmentation method.

Thus, it is crucial that these small blocks (*cc* or *ccc*) should be meaningful when treated as a word. Actually, this is similar to the word collocaiton in English language where collocation are words that like to hang out together. In Chinese, some characters often locate together in the same block, while some do not. Thus, we should try to avoid characters who do not like to hang out together being generated in the same block.

Usually, neural-based poetry generation chooses the character with the highest probability [13, 14] at each char-position, or uses beam search algorithm to help choose characters [15, 17]. But, if we directly choose the candidates from the most probable top-n characters for each char-position, most generated blocks (*cc* and *ccc*) are nonsense when treated as a word. As a result, the quality of generated quatrains will decrease greatly.

To solve this problem, we propose a template-constrained generation approach. For the first char-position of a block, we prepare candidates directly from the most probable top-n characters, then code candidates and finally choose one character based on *info*. We denote the chosen character for this position as c_{pre}. Then, when we come to the second or third char-position of a block (*cc*/*ccc*), we no longer choose candidates only based on their probabilities.

Instead, we reorder y_i by taking into account the relationship between c_{pre} and the potential candidates in y_i. Next, we will introduce how to reorder y_i by using inner-word mutual information.

3.2.2 Inner-word mutual information. Mutual information has been widely used in natural language processing [4, 12]. Luo et al. [12] used mutual information to roughly evaluate the quality of generated poetry, and Church and Hanks [4] used mutual information to estimate word association norms from corpora. Thus, we use mutual information between c_{pre} and characters in y_i to reorder y_i. The higher the mutual information between c_{pre} and a certain character is, the upper position that character will be reordered to.

The problem we need to solve is to make blocks (cc and ccc) more like normal words. So, we use mutual information calculated within a word to help reorder potential candidates, which we call inner-word mutual information, and it is calculated by:

$$I(x, y) = log\frac{p(x, y)}{p(x)p(y)}, \tag{8}$$

where we estimate $p(x)$ and $p(y)$ by counting how many times x and y appear in corpus, and normalizing by N, the size of the corpus. Similarly, $p(x, y)$ is estimated by the number of times that x is followed by y in the cc[1] block, and normalizing by N [4].

In this way, the proposed inner-word mutual information can ensure that generated characters in one block are more look like natural words. Now, we can enjoy the high embedding rate achieved by neural-based methods, but without worrying about the severe decline in the quality of generated poems.

3.3 Steganography example

In this section, we give a simple example to show how information is hidden during the process of poetry generation.

First, we need to determine the parameters mentioned in Figure 2. We take a 5-char quatrain line as the example. The initial parameters are set as follows:

- *Tone*: PPPZZ,PZZPP.ZZPPZ,PPZZP [2];
- *Info*: 0110110001...(converted from 'love');
- *Key*: 春风 (the word input in the word-to-line module);
- *Size*: 4 (the size of candidate pool).

At the beginning, we segment $PPPZZ$ into PP/PZZ which will be used as the template to regulate the char-choosing process. Then, the keyword 春风 is input into the word-to-line module to generate the first line which is denoted as $Y_1 = (y_1, y_2, y_3, y_4, y_5)$. For the first char-position y_1, the tone is P, so we filter out all characters of y_i whose tone is not P and select the most probable top-4 characters as candidates. Huffman coding algorithm is then applied to assign each candidate a code (see Table 1, col 1). As the beginning bits of *Info* are 01, 青 is chosen for this char-position.

For the second char-position y_2, we first filter out characters whose tone is not P, and then we reorder y_2 by calculating the inner-word mutual information between 青 and characters in y_2. In practice, we will not do the calculation between 青 and all characters in y_2, because this may cause us to choose these characters who are irrelevant to the input keyword but only has a high mutual information with 青. To avoid this problem, we actually only do the calculation for the most probable top-M characters, where M is slightly larger[3] than *Size*. After y_2 is reordered, we again use Huffman coding algorithm to assign each candidate a code (see Table 1, col 2) and finally choose 风 according to *Info* whose beginning bits now are 10.

Table 1: Details of char-choosing in steganography

Candidates									
Char1		Char2		Char3		Char4		Char5	
萧	00	葭	00	鸣	00	雨	00	绿	00
青	01	帆	01	迷	01	雪	01	碧	01
一	10	风	10	生	10	草	10	动	10
归	11	鸣	11	烟	11	沙	11	细	11

For the third char-position y_3, as it is the beginning char-position of a new block in the template (cc/ccc), its char-choosing process is exactly the same as y_1. Similarly, y_4 and y_5 follow the same procedure as y_2. Based on *Info*, the final generated stego-line is 青风烟雨碧.

Next, we input the generated line l_1 into the line-to-line module to obtain Y_2. We follow the same procedure as Y_1 to get the second generated line l_2. Then, $l_1 l_2$ is input into the line-to-line module to obtain Y_3, based on which the third line l_3 is generated. Finally, $l_1 l_2 l_3$ is responsible for the generation of the last line l_4.

In fact, we can also hide information during the keyword-choosing process, the same way as char-choosing part. If we suppose that there are m keywords in the keyword list, we can hide extra $log_2 m$ bits of information during the keyword-choosing process.

When the receiver obtains the stego-quatrain, he only needs the trained neural models and the keyword list that the sender uses, to extract the secret information, by following the same char-choosing procedure. The only difference between information hiding and extraction is that the sender chooses characters according to *Info* while the reciever obtains *Info* according to characters.

As shown in Figure 2, there are 4 initial parameters which need to be determined to start the generation process, including *Tone*, *Info*, *Key* and *Size*. For the receiver, *Tone* can be directly told by the received quatrain. *Info* is what the receiver wants to obtain. We suppose that the keyword list has m keywords and *Size* has n possible values, which means that there are $m{\cdot}n$ combinations in total. If the keyword and the value of *Size* that the receiver uses are not the same as the sender's setting, either the extraction process can not be finished, or the extracted message is nonsense. In other words, only the right combination can extract the right message. Thus, the receiver only needs to try out all combinations to extract the hidding information.

If the receiver can not afford too much time complexity, it is recommended to use a small keyword list and set *Size* to a fixed value. In short, m and n are flexible to be adjusted based on the user's own circumstance.

[1]The block of ccc could be segmented into the form of c/cc or cc/c. We count two characters' co-occurrence only when they appear together in cc.
[2]There are two types of tone, P and Z, which are 平 and 仄, respectively.

[3]Empirically, M is set to 1.5 or 2 times of *Size*. M is a pre-determined parameter by both the sender and receiver, so it is fixed during all stego-communications.

4 EXPERIMENTS

4.1 Dataset

There are many classical Chinese poetry corpora on the Internet, but quatrains written in different periods have different styles. So, we only collected quatrains from Tang dynasty. The corpus we used contains 74,474 quatrains in total, and we randomly selected 2,000 quatrains as validation set, and the rest as training set.

For the word-to-line module, we selected the top-1000 most frequently used words as our keyword list, and for each word, we selected 150 lines that contain this word to build word-to-line pairs [17]. For the line-to-line module, we extracted 3 line-to-line pairs from each quatrain. For example, from a quatrain L=(l_1,l_2,l_3,l_4), we can get three pairs, <l_1, l_2>,<$l_1 l_2$, l_3> and <$l_1 l_2 l_3$, l_4> [15].

4.2 Training and decoding

During the training, we selected the top-8000 frequent characters as our vocabulary which is shared by both input and target sides. We built our model based on Keras. The word embedding layer has 512 units, and both the encoder and decoder contain 512 hidden units. The model was trained with Adagrad algorithm, where the minibatch size was set to 128. We selected the final model according to the loss on the validation set [14, 15]. For the line-to-line module, we trained 5-char and 7-char quatrains on one model, but for the word-to-line module, we trained 5-char and 7-char quatrains on two separate models.

In the decoding, we are supposed to input the generated line into the line-to-line model. However, due to the process of information hiding, we may select a character who has low probability. For example, when *Size* is 32, the final chosen character might be the one whose probability only ranks 32_{th}. These low ranking characters actually, to some extent, have deviated from the top ranking character in the meaning. The larger *Size* is, the more likely the final chosen character will deviate from the input keyword in its meaning. In practice, we do not input the original generated line into line-to-line module to avoid the deviation. Instead, for each line l_i, we forge a line l'_i whose characters are all selected from the top-1 candidate. The forged line l'_i will be input into line-to-line module rather than l_i. In this way, the whole quatrain is more likely to reveal the meaning of the input keyword.

5 EVALUATIONS

To evaluate a steganography system based on text generation, there are two important factors: one is the naturalness of generated text; the other is the embedding rate. Next, we evaluate our proposed method by comparison between different poetry steganography algorithms in these two factors.

5.1 Human evaluation

To accurately evaluate the quality of machine generated poetry is a notorious problem. Most researchers rely on human evaluation, in which experts are invited to evaluate generated poems with regard to their fluency, poeticness, meaning, etc. [14, 15, 17, 19]. However, our system is steganography-oriented, and our goal is to generate poems that will not attract suspicions from the third party. Thus, we use the the famous Turing Test to evaluate our model, which is also adopted by [13, 15].

We designed a questionnaire which contains quatrains from four sources. The subject was asked to judge whether these poems were created by machine or written by a poet. According to how many machine-generated poems are identified as human created, we can evaluate the naturalness of poems generated by a specific steganography method.

We chose two poetry steganography methods and human created poems as baselines. All methods based on machine generation utilized the same corpus and parameter setting[4]. We selected eight poems from each source, containing four 5-char quatrains and four 7-char quatrains. So, each questionnaire includes 32 poems.

- Segment-based method [11]
- Markov-based method [12]
- Poet created poems
- Our method

To make the evaluation results more convincing. Subjects are all master or PhD students who are well educated. Moreover, we cleaned [5] returned questionnaires and obtained 20 valid samples out of 33 returned ones. The results are shown in Table 2.

Table 2: Human evaluation

	Identified as H	Identified as M
Segment-based	21.3%	78.7%
Markov-based	33.8%	66.2%
Our method	34.3%	65.7%
Poet	63.1%	36.9%

H: human-created, M: machine-generated

We can clearly see that, our method weakly passed the Turing test whose criterion is to fool people in no less than 30% of asked questions.

5.2 Embedding rate

Embedding rate is highly related to how much information we can hide in a stego-text. Previous methods have either very low embedding rate or flaws in the quality of generated texts. For example, the Markov-based method [12] can produce poems comparable to us in poem quality, but it has low embedding rate. As for the segment-based method [11] which has high embedding rate, its poem quality is far behind us.

The high embedding rate of our approach stems form that it is char-based generation, in which we can hide information at each char-position. Actually, apart from the generation process of the poem body, information can also be hidden when we choose the initial keyword from the keyword list. The keyword list we used in the experiment contains 1,000 words, so the embedding rate can be calculated by:

[4]The size of candidate pool for Markov-based and our neural-based method were both set to 32.

[5]Questionnaires that were finished in less than 2 minutes or more than 30 minutes were excluded. We also did not take into consideration the ones whose accuracy is below 50%.

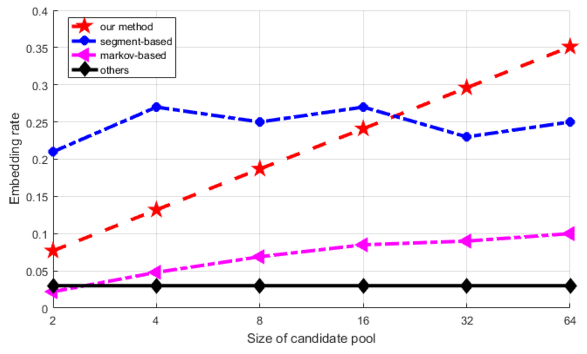

Figure 3: The comparison of embedding rate between different methods.

$$ER = \frac{4N \times log_2 Size + log_2 1000}{4(N+1) \times 16}, \quad (9)$$

where ER is the embedding rate, $Size$ is the size of candidate pool, N is the number of characters contained in one poem line. The comparison of embedding rate between different methods is shown in Figure 3. Obviously, our method outperforms other approaches and the embedding rate can reach about 35.0% (when $Size$=64, N=7).

5.3 Generation example

Finally, in Table 3, we show a 5-char quatrain generated by neural-based steganography. According to its poetic meaning, we name it as 'Returning Night', which is 归夜 in Chinese.

Table 3: A 5-char quatrain generated by neural-based steganography

归夜	Returning Night
风窗烟树中	Out the window stood the tree in mist
柔静雨光斑	Soft and quiet is the shining rain
夜里吹生下	Playing the flute in the night
归人乱浪宽	Traveling back from wide waves

6 CONCLUSION

In this paper we have presented a method that uses neural-based poetry generation to hide information. Attention-based neural model is the state-of-the-art approach in poetry generation. Even though it can produce poems that are coherent and semantically consistent, directly applying it to steganography does not work well. Thus, we propose a template-constrained generation method and use inner-word mutual information to alleviate the quality decline caused by steganography. Experimental results show that our method yields stego-poems with high quality, comparable to the state-of-the-art poetry steganography, but has a much higher embedding rate. In the future, we would like to continually improve the quality of generated poems, and explore poetry steganography using other genres, e.g. English sonnets or even normal-style texts.

7 ACKNOWLEDGEMENT

We thank all reviewers for their constructive reviews. Many thanks to Yixuan Wang for the translation of poem exapmle. This work is supported by the National Natural Science Foundation of China (Grant Nos. U1536207, U1536115 and U1536113).

REFERENCES

[1] A.M. Alattar and O.M. Alattar. 2004. Watermarking electronic text documents containing justified paragraphs and irregular line spacing. *Electronic Imaging 2004. International Society for Optics and Photonics* (2004), 685–695.
[2] D. Bahdanau, K. Cho, and Y. Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473* (2014).
[3] M. Chapman and G. Davida. 1997. Hiding the hidden: A software system for concealing ciphertext as innocuous text. *International Conference on Information and Communications Security. Springer Berlin Heidelberg* (1997), 335–345.
[4] K.W. Church and P. Hanks. 1990. Word association norms, mutual information, and lexicography. *Computational linguistics* 6, 1 (1990), 22–29.
[5] J. Cong, D. Zhang, and M. Pan. 2010. Chinese Text Information Hiding Based on Paraphrasing Technology. *Information Science and Management Engineering (ISME), 2010 International Conference of. IEEE* (2010), 39–42.
[6] S. Ghosh, O. Vinyals, B. Strope, S. Roy, T. Dean, and L. Heck. 2016. Contextual LSTM (CLSTM) models for Large scale NLP tasks. *arXiv preprint arXiv:1602.06291* (2016).
[7] J. He, M. Zhou, and L. Jiang. 2012. Generating Chinese Classical Poems with Statistical Machine Translation Models. *AAAI* (2012).
[8] Y.W. Kim, K.A. Moon, and I.S. Oh. 2003. A Text Watermarking Algorithm based on Word Classification and Inter-word Space Statistics. *ICDAR* (2003), 775–779.
[9] I.S. Lee and W.H. Tsai. 2008. Secret communication through web pages using special space codes in HTML files. *International Journal of Applied Science and Engineering* 6, 2 (2008), 141–149.
[10] Y.L. Liu, X.M. Sun, G. Gan, and W. Hong. 2007. An efficient linguistic steganography for Chinese text. *IEEE International Conference on Multimedia and Expo* (2007), 2094–2097.
[11] Y.C. Liu, J. Wang, Z.B. Wang, Q.F. Qu, and S. Yu. 2016. A technique of high embedding rate text steganography based on whole poetry of song dynasty. *International Conference on Cloud Computing and Security* (2016), 178–189.
[12] Y.B. Luo, Y.F. Huang, F.F. Li, and C.C. Chang. 2016. Text Steganography Based on Ci-poetry Generation Using Markov Chain Model. *KSII Transactions on Internet and Information Systems* 10, 9 (2016), 4568–4584.
[13] Q.X. Wang, T.Y. Luo, and D. Wang. 2016. Can Machine Generate Traditional Chinese Poetry? A Feigenbaum Test. *Advances in Brain Inspired Cognitive Systems: 8th International Conference, BICS 2016* (2016), 34–46.
[14] Q.X. Wang, T.Y. Luo, and D. Wang. 2016. Chinese song iambics generation with neural attention-based model. *arXiv preprint arXiv:1604.06274* (2016).
[15] Z. Wang, W. He, H. Wu, H.Y. Wu, W. Li, H.F. Wang, and E.H. Chen. 2016. Chinese poetry generation with planning based neural network. *arXiv preprint arXiv:1610.09889* (2016).
[16] P. Wayner. 1992. Mimic functionsWayner P. *Cryptologia* 16, 3 (1992), 193–214.
[17] X. Yi, R. Li, and M. Sun. 2016. Generating chinese classical poems with rnn encoder-decoder. *arXiv preprint arXiv:1604.01537* (2016).
[18] Z.S. Yu and L.S. Huang. 2009. High Embedding Ratio Text Steganography by Ci-poetry of the Song Dynasty. *Journal of Chinese Information Processing* 23, 4 (2009), 55–62.
[19] X. Zhang and M. Lapata. 2014. Chinese Poetry Generation with Recurrent Neural Networks. *EMNLP* (2014), 670–680.