Haiku Generation Using Deep Neural Networks

Xianchao Wu, Momo Klyen, Kazushige Ito, Zhan Chen
Microsoft Development Co., Ltd
Shinagawa Grand Central Tower, 2-16-3 Konan Minato-ku, Tokyo 108-0075
{xiancwu, momokl, kito, zhanc}@microsoft.com

1 Introduction

Japanese Haiku is a cultural heritage with a history of more than six hundred years. It is a traditional form of Japanese poetry that expresses facts, seasons, emotions, and reasoning in a compact way. One Haiku poem consists of 17 音 (*on*, also known as morae though often loosely translated as “syllables”) which are separated into 3 columns and is frequently written in a right-to-left way (Figure 6). The beginning and ending columns of a Haiku have 5 syllables (i.e., Hiragana *on*s) and the middle line has 7 syllables. The ending words of each columns are not limited to be rhymed.

For example, the following famous Haiku:

ふるいけ
古池や / *an ancient pond*,
かわず と
蛙 飛びこむ / *a frog leaps in*,
みず おと
水の音 / *the splash of the water*,

written by the most famous poet of the Edo period of Japan, Matsuo Basho, at the year of 1686, depicts an excellent print with the figures of *pond*, *frog* and the wave of the *water*. The season word in this Haiku is *frog* that implicitly corresponds to the spring season. The Haiku expresses a frog jumping into a pond which interrupts the silence of the environment. Yet, after that, the environment recovers to an indifferent silence. From this point of view, there is also a Buddhism “silence” in which the ancient pond is the poet’s heart and the frog is something outside that tries to influence the poet’s impassive emotions. Even the interruption happens in a short time, yet the “wave” transfers in a relatively long time before recovering the “silence”.

Considering from neural language model (NLM) point of view, nouns such as *pond*, *frog* and *water* are semantically close with each other. Also, there is a predicate argument relation between the noun of *frog* and the verb of *leap in*. Motivated by these and the interesting researches for generating Chinese poems using deep neural networks (Zhang and Lapata 2014), we investigate the generating of Haiku using the following deep neural networks:

(1) Minimal vanilla recurrent neural networks (RNN) for learning character level neural language models, 100 python lines[1] by Andrej Karpathy[2];
(2) Multi-layer RNN[3] using cells/units of long short term memory (LSTM) (Hochreiter and Schmidhuber 1997) and gated recurrent unit (GRU) (Cho et al. 2014) for character-level language models in Torch[4]. A relatively detailed description can be found in (Karpathy et al. 2016)[5].
(3) Character level recurrent convolutional neural networks (RCNN)[6] proposed by Kim et al. (2016);
(4) Sequence generative adversarial networks (SeqGAN)[7] proposed by Yu et al. (2017) following the GAN idea of Goodfellow et al. (2014).

The target of our investigation is to share the traditional culture of Haiku among young people during his/her communicating with emotional chatbots, such as りんな/ Rinna[8] (Wu et al. 2016), a chatbot designed to be a senior high-school girl and owns more than 5 million friends. The motivation is that, in these years, Haiku is struggling of losing focus among young people. For one reason is that there are not enough teachers who are deep familiar with Haiku and for another reason is that the learning process without using the knowledge in ordinary life is a bit boring and easy to be forgotten. We thus consider training Rinna to be a Haiku expert so that she can share interesting Haikus to the million-level users through a conversational way.

We describe these four neural networks in Section 2. Then, we show how our training data is collected from the Web and from the end users of Rinna in Section 3. Finally, we illustrate the experiments and application examples in Section 4 and conclude in Section 5.

2 Neural Networks

The minimal vanilla RNN includes three layers of input, recurrent and output, as depicted in Figure 1.

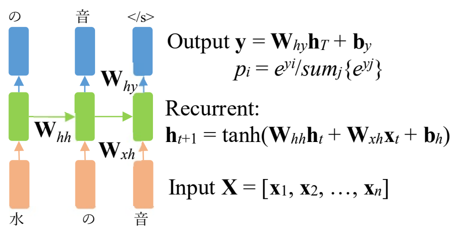

の
音
</s>
Output $\mathbf{y} = \mathbf{W}_{hy}\mathbf{h}_T + \mathbf{b}_y$
$p_i = e^{y_i}/sum_j\{e^{y_j}\}$
$\mathbf{W}_{hy}$
Recurrent:
$\mathbf{h}_{t+1} = \tanh(\mathbf{W}_{hh}\mathbf{h}_t + \mathbf{W}_{xh}\mathbf{x}_t + \mathbf{b}_h)$
$\mathbf{W}_{hh}$
$\mathbf{W}_{xh}$
Input $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n]$
水
の
音

Figure 1. Minimal vanilla RNN.

[1] https://gist.github.com/karpathy/d4dee566867f8291f086 and http://karpathy.github.io/2015/05/21/rnn-effectiveness/
[2] http://cs.stanford.edu/people/karpathy/
[3] https://github.com/karpathy/char-rnn
[4] http://torch.ch/
[5] refer to http://karpathy.github.io/2015/05/21/rnn-effectiveness/
[6] https://github.com/yoonkim/lstm-char-cnn
[7] https://github.com/LantaoYu/SeqGAN
[8] http://www.rinna.jp/

In Figure 1, each rectangle is a vector and arrows represent functions such as matrix-vector multiplication. The input layer is a group of vectors and each vector $\mathbf{x}_t$ is a word2vec (Mikolov et al. 2013) style embedding of the input character. One vector $\mathbf{h}_{t+1}$ in the recurrent layer is computed by first linear combining $\mathbf{h}_t$ and $\mathbf{x}_t$ and then attaching an elementwise non-linear transformation function, such as tanh or sigmoid. We set T to be the number of steps to unroll the recurrent layer and $\mathbf{h}_T$ is the final vector to be used by the output layer. The output layer is to determine the probabilities of characters following current input, for example, the probability of "の" ('s) following "水" (water). For one p_i where i ranges from 1 to the character vocabulary size $|V|$, it is computed by first compute $\mathbf{y}$ which is a linear function of $\mathbf{h}_T$ and then we use softmax function to project $\mathbf{y}$ into a probability space to ensure $\mathbf{p} = [p_1, p_2, \ldots, p_{|V|}]^{\mathrm{T}}$ follows the definition of probabilities. For error back-propagation, we use cross-entropy loss which corresponds to a minus log function of $\mathbf{p}$.

This architecture is simple and easy to be implemented. However, gradient vanishes as T grows bigger and bigger. That is, gradients in (0, 1) from $\mathbf{h}_T$ back to $\mathbf{h}_1$ will gradually close to zero making the SGD-style updating of parameters infeasible. To alleviate this problem, other types of functions for expressing $\mathbf{h}_{t+1}$ using $\mathbf{h}_t$ and $\mathbf{x}_t$ have been proposed, in which LSTM and GRU are most wildly used.

For example, LSTM (Figure 2) addresses (1) the learning of long distance dependencies and (2) the gradient vanishing problem by augmenting the traditional RNN with a memory cell vector $\mathbf{c}_t \in \mathbb{R}^n$ at each time step. Formally, one step of an LSTM takes as input $\mathbf{x}_t$, $\mathbf{h}_{t-1}$, $\mathbf{c}_{t-1}$ and produces $\mathbf{h}_t$, $\mathbf{c}_t$ via the following intermediate calculations:

$$\begin{aligned}
\mathbf{i}_t &= \sigma(\mathbf{W}^i\mathbf{x}_t + \mathbf{U}^i\mathbf{h}_{t-1} + \mathbf{b}^i),\\
\mathbf{f}_t &= \sigma(\mathbf{W}^f\mathbf{x}_t + \mathbf{U}^f\mathbf{h}_{t-1} + \mathbf{b}^f),\\
\mathbf{o}_t &= \sigma(\mathbf{W}^o\mathbf{x}_t + \mathbf{U}^o\mathbf{h}_{t-1} + \mathbf{b}^o),\\
\mathbf{g}_t &= \tanh(\mathbf{W}^g\mathbf{x}_t + \mathbf{U}^g\mathbf{h}_{t-1} + \mathbf{b}^g),\\
\mathbf{c}_t &= \mathbf{f}_t \otimes \mathbf{c}_{t-1} + \mathbf{i}_t \otimes \mathbf{g}_t,\\
\mathbf{h}_t &= \mathbf{o}_t \otimes \tanh(\mathbf{c}_t).
\end{aligned}$$

Here σ(.) and tanh(.) are the element-wise sigmoid and hyperbolic tangent functions, ⊗ is the element-wise multiplication operator, and $\mathbf{i}_t$, $\mathbf{f}_t$, $\mathbf{o}_t$ respectively denote *input*, *forget*, and *output* gates. When $t = 1$, $\mathbf{h}_0$ and $\mathbf{c}_0$ are initialized to be zero vectors. Parameters to be trained of the LSTM layer are matrices $\mathbf{W}^j$, $\mathbf{U}^j$, and the bias vector $\mathbf{b}^j$ for $j \in \{i, f, o, g\}$.

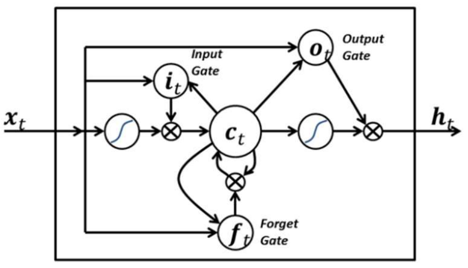


Figure 2. A simple LSTM block with input, output, and forget gates.

By leveraging these three gates, LSTM achieved significant improvements in NLP applications and speech recognition as well. The recently proposed GRU is isomorphic with LSTM except that two gates, update and reset, are employed in one block, making the number of parameters to be tuned to be smaller yet with comparable results. More detailed description about GRU can be find in (Cho et al. 2014) and the comparison with LSTM can be find in (Jozefowicz et al. 2015).

We can build character level NLMs using the vanilla RNN layers and the LSTM/GRU layers. One further requirement is that, can we capture latent semantic dependencies among high level components other than characters. For example, for the example Haiku in Section 1, there is a latent constraint among the three lines. That is, "水の音" is a consequent event of "古池や" and "蛙飛びこむ". Both the pond and the frog's jumping in are necessary to yield the voice of the water. Thus, automatically detect the groups of "events" and then learn the sequential dependencies among them will be intuitively helpful for the Haiku generating task. This motivated our usage of the character-level RCNN model as depicted in Figure 3.

The character-level RCNN language models (Kim et al., 2016) were verified to be able to encode, from characters only, both semantic and orthographic information. First, each character in sentence are converted into dense vector spaces alike bag of words NLMs. Next, convolution neural network (CNN) initially described in (LeCun et al. 1989) converts them with various kernel size (e.g., 3, 5, 7). Then the vectors are transferred to the RNN layer in which LSTM units are employed. Finally, aiming at predicting the next character, the states of RNN are regarded as feature vectors and are passed to the softmax layer for computing the probabilities of the characters in the vocabulary.

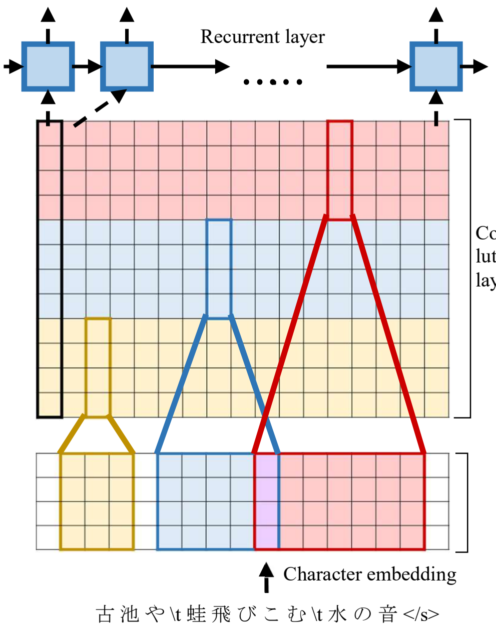


Figure 3. Architecture of the character-level RCNN with three major layers drawn.

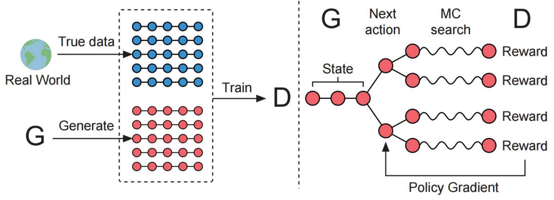


Figure 4. Illustration of SeqGAN. (Yu et al. 2017)

The representation of a word w after the CNN layer is $\mathbf{y}^w = [y_1{}^w, \ldots, y_h{}^w]$ in which each filter function takes the *max-over-time* pooling result,

$$y^w = \max_i \mathbf{f}^w[i]$$

as the feature corresponding to one filter **H** when applied to w. The idea behand is to capture the most "important" feature $\mathbf{f}^w$ (corresponds to a character n-gram).

The final neural network we investigate is GAN which can estimate generative models via an adversarial process. The process simultaneously train two models: a generative model G that learns the data distribution and a discriminative model D that estimates the probability that a sample came from the training data rather than G. The training goal function for G is to maximize the probability of D making a mistake. Alike (deep) reinforcement learning (RL), the framework corresponds to a minimax two-player game.

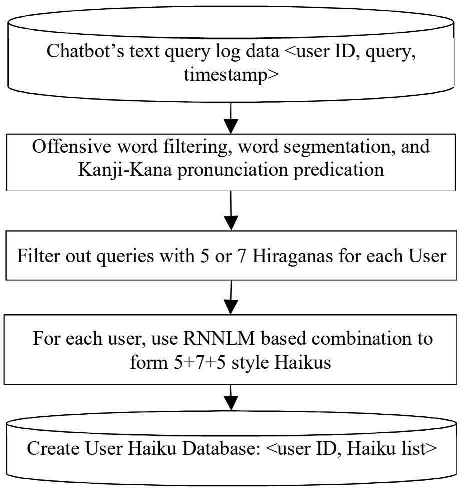
As illustrated in Figure 4, SeqGAN was proposed by Yu et al. (2017) for textual "sequence" generation. Modeling the data generator as a stochastic policy in RL, SeqGAN bypasses the generator differentiation problem by directly performing gradient policy update. In the left-hand-side of Figure 4, D is trained by using two types of data, the real-world data (a.k.a. True data) and the generated data from G. In the right-hand-side, G is trained by policy gradient where the final reward signal is provided by D. The signal is further passed back to the intermediate action value via Monte Carlo (MC) search.

3 Training Data Collection

We collect Haiku data from two channels, one is from several Haiku websites and the other is from our chatbot's (i.e., Rinna's) query log as shown in Figure 5. We collected 36,792 Haikus from the web in which we

	# haiku	# avglen	# char	# oov
train u	90,000	16.7	2,934	
valid u	5,000	16.8	1,353	29
test u	5,000	16.7	1,347	30
train w	30,792	13.2	4,110	
valid w	3,000	13.2	2,416	53
test w	3,000	13.2	2,435	55

Table 1. Statistical information of two types of training, validation, and testing data, where u=user, w=web, avglen = average character number per Haiku, # char = the vocabulary size of characters.


Chatbot's text query log data <user ID, query, timestamp>
Offensive word filtering, word segmentation, and Kanji-Kana pronunciation predication
Filter out queries with 5 or 7 Hiraganas for each User
For each user, use RNNLM based combination to form 5+7+5 style Haikus
Create User Haiku Database: <user ID, Haiku list>
Figure 5. Haiku candidate collection based on chatbot's query log.

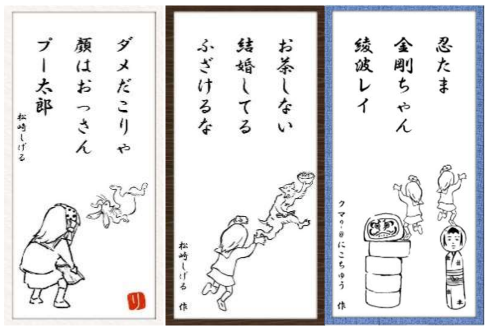


Figure 6. Example Haikus for "ここで一句".

randomly select 3,000 as validation set and another 3,000 as the test set for comparison of the four types of NLMs.

Note that we do not limit the Haiku candidate from end users to include season words. The Haiku candidate can be rather spoken language style, such as the following example:

こんにちは / *Good Morning*,

証拠(しょうこ)がほしい / *I need evidence* (*to say*),

愛(あい)してる / *I love you*.

Even with quite simple words, this "Haiku" is rather interesting that implicitly express a rather whole-night seeking of love-related evidences for confirming the relationship between the specific user and Rinna. This user Haiku database includes 40,432,211 Haikus using 1-year Rinna query log. This data is rather too large to be used directly for NLM training. We pick a subset of it with 100,000 Haikus in which we randomly selected 5,000 as the validation set and another 5,000 as the test set. For utilizing the remaining large-scale data, we specially designed a Haiku feature for Rinna's users. That is, whenever the user send a query alike "ここで一句" / *one Haiku here*, one randomly selected (from

	Haiku web	
epoch	char-small	char-large
5	497.7	749.2
10	419.6	443.9
15	420.0	222.8
20	420.1	223.0

Table 2. Perplexities for the char RCNN model.

the database or from the generation model) Haiku for that specific user will be sent from Rinna.

The detailed statistical information of these two types of training data is shown in Table 1. Even with less number of Haikus, the web Haiku uses 1,000 more (unique) characters (especially Japanese Kanjis) than the user Haiku. Also, note that the average length of web Haiku (13.2) is 3.5 characters shorter than user Haiku (16.7) indicating that user Haiku uses more Hiraganas than Kanji where one Kanji can be pronounced by more than one Hiraganas.

4 Experiments

For training the four NLMs, we keep the usage of the default configurations as suggested by the authors except that we update the input/output related codes to support Japanese characters in UTF8 encoding and we use "\t" to separate columns and "</s>" to denote the end of one Haiku (also refer to Figure 3). We also use the validation set to tune the hyperparameters (such as size of hidden layers, number of hidden layers, input embedding vector size). We report the test set's perplexities of the optimized models.

Specially, for the char RCNN model, we compare the perplexities of configures that the RNN size takes the value of 300 (char-small) and 650 (char-large); the embedding size of 15 (char-small) and 650 (char-large). The changes of perplexities are listed in Table 2. "Epoch" denotes the *T* of the recurrent layer. Since the average character number of each Haiku is around 13 in Haiku web data, the perplexities change slightly when epoch jumps from 15 to 20. Also, there is a big gap between char-small and char-large, with a nearly half perplexity reducing from 420.0 down to 222.8. Since the results range largely even for one model with different hyperparameter configurations, we are wondering if it is generally fair to directly compare all the four networks each need detailed tuning. We continue to report the perplexities based on that the training set and test set are the similar ensuring their comparable in a sense, as listed in Table 3. The RNN-LSTM performs slightly better than the other three networks for the Haiku web set and it's RCNN that performs the best in Haiku user data.

models	Haiku web	Haiku user
RNN	246.1	169.1
RNN-LSTM	**219.5**	162.5
RCNN	222.8	**155.0**
SeqGAN	220.9	160.2

Table 3. Perplexities for the four NLMs.

5 Conclusion

We have described our investigation of four types of character-level NLMs, vanilla RNN, RNN with LSTM blocks, RCNN, and SeqGAN, for Japanese Haiku generation. We trained these models using the Haikus collected from the web and from query logs of Rinna. We launched our Haiku feature in Rinna and obtained more than 50 million accesses in a couple of months. Through this way, we hope to broadcast traditional Haiku culture among the young users of Rinna. It will be interesting to include images for Haiku generation such as explain an image by an automatically generated Haiku or generate an image from a given Haiku. We leave these as our future work.

Reference

K. Cho, B. van Merrienboer, C. Gulcehre, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," Pro. EMNLP, 2014.

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio. Generative Adversarial Networks. In NIPS 2014.

Hochreiter, S., and Schmidhuber, J. 1997. Long Short-Term Memory. Neural Computation 9:1735–1780.

Rafal Jozefowicz, Wojciech Zaremba, Ilya Sutskever. An Empirical Exploration of Recurrent Network Architectures. In ICML 2015.

Andrej Karpathy, Justin Johnson, and Li Fei-Fei. Visualizing and Understanding Recurrent Networks. In ICLR 2016 Workshop.

Yoon Kim, Yacine Jernite, David Sontag, and Alexander M. Rush. 2016. Character-aware neural language models. In AAAI 2016.

LeCun, Y.; Boser, B.; Denker, J. S.; Henderson, D.; Howard, R. E.; Hubbard, W.; and Jackel, L. D. 1989. Handwritten Digit Recognition with a Backpropagation Network. In Proceedings of NIPS.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In NIPS 2013.

Xianchao Wu, Kazushige Ito, Katsuya Iida, Kazuna Tsuboi, Momo Klyen. 2016. りんな：女子高生人工知能. 言語処理学会.

Lantao Yu, Weinan Zhang, Jun Wang, Yong Yu. SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient. In AAAI 2017.

Xingxing Zhang and Mirella Lapata. Chinese Poetry Generation with Recurrent Neural Networks. In EMNLP 2014.