Anthony Bernardi & Anja Castro-Diephouse
April 19th, 2023
CS 4120
Professor Muzny

Generating Haiku Using Natural Language Processing

1 Artist's Statement

Our body of work combines the modern technology of Artificial Intelligence and Haiku. Haiku is a classical form of Japanese poetry composed of three stanzas. The first stanza has 5 syllables, the second has 7, and the third has 5 again. Haiku traditionally use nature as their subject,[1] yet neural networks in many ways are the antithesis of nature. This is what made us so interested in this topic – can something as unnatural and man-made as a neural network create poetry that reflects on the beauty of nature?

Through our work, we have found that it is possible to create incredibly unique haiku using Artificial Intelligence. Because the neural net has access to a vast array of words and phrases, yet lacks explicit instruction in the norms and expectations of conventional language use, it is able to create connections and associations that may not occur to a human poet. This has led to the creation of haiku that are surprising, absurd, humorous, and even beautiful. While these haiku may not be traditional, we believe that this does not diminish their artistic value and in fact brings a unique perspective to the form.

Rather than replacing human creativity, we see our work as a tool for collaboration and inspiration. We hope that our work will contribute to a broader dialogue about the intersection between technology and creativity, and inspire others to engage in their own explorations of this relationship.

2 Introduction

In this paper, we present our approach to generating haiku using neural networks. We used a feedforward neural network and a recurrent neural network that were trained on word embeddings from a corpus of diverse haiku. Our results were generated using two different techniques. The first was a standard sampling method, where the probability distribution produced by the network is sampled to get a likely next word. The second was a greedy algorithm that threw out any words that didn't fit into the stanza until the network provided one that did. Overall, our approach yielded promising results, demonstrating the potential of natural language processing to contribute to poetry and perhaps other fields of art.

3 Data

The dataset we used to train our models was a Kaggle dataset of over 11,000 haiku.[2] These haiku were pulled from the Reddit subreddit, /r/haiku. Almost all of the haiku in the dataset followed the 5-7-5 syllable structure, providing a good opportunity for our models to learn the correct structure by example. However, the haiku covered a wide range of topics, and were not confined to the traditional topic of just nature. Example (1) below provides a few examples from the dataset.

(1) delicate savage / you'll never hold the cinder / but still you will burn $

[1] Academy of American Poets, https://poets.org/glossary/haiku
[2] Bfbarry, https://www.kaggle.com/datasets/bfbarry/haiku-dataset

our destination / the skyline of this city / shining horizon $
a splash and a cry / words pulled from the riverside / dried in the hot sun $
i woke up today / i wanted to write a song / i wrote a haiku $

Before converting these lines into word embeddings, we preprocessed them to create a better context for training. To do this, we whitespace-tokenized the lines and added haiku starting and ending tags, as well as starting and ending tags for each stanza. Each stanza got one start tag and one end tag regardless of the ngram length used by the model, but the number of haiku start and end tokens was always one fewer than the n-gram length. An example of one of the haiku after preprocessing using an n-gram of 5 can be seen below in (2). Note that line breaks were only added for clarity.

(2) <h> <h> <h> <h>
<s> delicate savage </s>
<s> you'll never hold the cinder </s>
<s> but still you will burn </s>
</h> </h> </h> </h>

The vocabulary size of the dataset was only a bit over 14,000 types, case folded but not lemmatized. Due to the relatively small vocabulary, as well as the fact that this model is intended to be trained once and then used to generate text, we did not incorporate unknown tokens for low-frequency words. We wanted the model to be able to use all the words in its input, and did not anticipate the model learning from any new data after its initial training.

Word embeddings were then trained on the processed haiku tokens using the gensim word2vec model, resulting in embeddings of length 200.

In order to write the greedy algorithm to enforce the syllable structure, we also needed a dictionary of English words and their syllable counts. We found this in another Kaggle dataset that included many other features such as the phonetic transcription, the starting vowels, and the ending vowels.[3] Columns other than word and syllable count were removed and the resulting word-syllable pairs were stored in a lookup table.

4 Models

The models we created to generate haiku were two neural language models: a simpler feedforward neural network and a recurrent neural network using long short-term memory cells. The feedforward network was intended to act as a baseline to compare against the LSTM model, which was chosen in the hope that its ability to learn longer-term dependencies would help it replicate the format and structure of the training haikus.

Both models were trained on n-grams, meaning the input data was a sequence of n-1 consecutive words from the training haiku and the correct label was the next word from the text. We experimented with different n-gram sizes and decided on n=7, which gave a large enough window for the models to learn more distant word relationships while still being small enough that each haiku could produce many training data points.

After preprocessing and training the word embeddings, as described above, the tokens from the haiku dataset were broken into n-1-word windows and encoded as integers using the keras Tokenizer. The true label for each of these training examples (the next word in the text)

[3] Jon Schwartz, https://www.kaggle.com/datasets/schwartstack/english-phonetic-and-syllable-count-dictionary

was also encoded as an integer. We did not separate the data into a training set and a validation set, since we were not particularly interested in the models' accuracy at predicting the next word in the text, but rather the quality and artistic merit of the generated haikus.
At this point, the training of the two models diverges, so they will be discussed separately.

4.1 Feedforward model

For the feedforward model, the n-gram training examples had to be converted into word embeddings to be used as input, and the correct labels had to be converted into one-hot vectors to be used in the loss calculation. This was done by means of a data generator, which took in the integer-encoded data and labels and made the conversions. The embeddings for each of the n-1 words in the input window were concatenated together, meaning the input data to the model was a vector of length (n-1)*200. The true label was converted to a one-hot vector with one more dimension than the size of the vocabulary, since the keras Tokenizer starts encoding words from 1 rather than 0. The data generator produced batches of training inputs and true labels with batch size 128, which were the final input given to train the model.

The model consists of two layers: one layer of 1000 hidden units using softmax activation, plus an output layer with 14270 (vocabulary size + 1) units that returns a softmax probability distribution over all known words. The loss function was cross-entropy loss, standard for multi-class outputs like the one used here. The model trained on the data for 3 epochs, each of which took approximately 11 minutes.

4.2 Recurrent model

The LSTM model also used a data generator to produce inputs as word embeddings and labels as one-hot vectors, similar to the feedforward model. However, recurrent neural networks require each input data example to be in the form of a timestep with multiple observations, one label per timestep. In our case, the timesteps are just successive n-gram windows, since if we think of the tokens in a haiku as a sequence of events, then moving to the right in the list of tokens is equivalent to moving forward in time. The main difference from the feedforward model is that the word embeddings are not concatenated together but grouped within a list. Since recurrent networks require inputs of shape (# observations per timestep, # features per observation), the data generator had to produce batches with shape (128, n-1, 200), rather than shape (128, (n-1)*200) like with the feedforward network. The label shape remained the same, with a single one-hot vector for each timestep of n-1 observations.

The LSTM model itself consists of an LSTM layer followed by an output layer to perform the softmax over the vocabulary size and get a probability distribution. The loss function was again cross-entropy loss. The number of hidden units in the LSTM layer varied— we initially used 128 units, as that was the size used in several of the examples we looked at for RNN language models. However, when we experimented with increasing the size to 500 hidden units, the resulting haikus seemed to improve. The examples shown in our presentation and printed below are from the larger model, but examples of haikus generated by the smaller model can be found in the 'generated_haikus.txt' file. The LSTM model also trained on the data for 3 epochs, which took around 5 minutes each for the smaller model and 13 minutes each for the larger model.

5 Results

As previously mentioned, we used two different methods to generate haiku from the models. To

start the generation using the first method, the models were provided a list of the haiku start tokens equal to n - 1 of the n-gram used. Then, the predict function was called for the model which returned a probability distribution of each possible token. This distribution was sampled to produce a new word according to the model's probabilities, and the new word was added to the sentence. Then, the process was repeated until the model generated a haiku ending tag or the haiku reached 30 tokens. A sample of 20 haikus generated by each model using both generation methods can be found in the file 'generated_haikus.txt', included with the submission code, but a few examples are given below to illustrate the patterns in the generated data.

In example (3) below, some outputs from the feedforward neural network can be seen using this method.

(3) (a)
<h> <h> <h> <h> <h> <h>
would i black beautiful heads </s>
you teach seek starts on i were drown crave </s>
rime
<s> </s> </h>

(b)
<h> <h> <h> <h> <h> <h> </s>
<s> i </s> </h>

(c)
<h> <h> <h> <h> <h> <h> </s> </h>

(d)
<h> <h> <h> <h> <h> <h>
strangers throttle icarus turns the </s>
progress than </s>
<s> animals </s>
<s> </s> </s> </h>

These examples demonstrate the volatility of the output of the feedforward model. Perhaps with more compute power, the model would have been better able to pick up on the patterns of the haiku, however, it seems as though the model does not have the long term dependencies needed to fully learn the structure of haiku.

Example (4) below demonstrates the results of this first method on the RNN model. As can be seen, the RNN seems to understand the structure of the haiku a little better, but there are still many instances with fewer stanzas or more stanzas than needed. Similarly, this one does not properly follow the syllable count rules either.

(4) (a)
<h> <h> <h> <h> <h> <h>
<s> maybe left the same spring </s>
<s> this is weird </s>
<s> shooting ballerinas </h>

(b)
<h> <h> <h> <h> <h> <h>
<s> pasty the sun </s>
<s> waiting on the dark </s> </h>

(c)
<h> <h> <h> <h> <h> <h>
<s> leave for a newfound </s> </h>

The second method we used was a greedy algorithm that essentially forced the models to come up with correctly formed haiku. We used primarily the same method as before, but if any selected word would have caused that line to have too many syllables, then we selected again until the model predicted a word that fit into the line. We did this by using the previously mentioned syllable dictionary dataset. If the dataset did not contain one of the predicted words, then we skipped that word as well. Each line began with a stanza begin token, and once the correct syllable length was reached, a stanza end token closed it off, ensuring proper structure. Example (5) below demonstrates some examples of results from the feedforward model using this greedy algorithm. In these examples, the tags have been left out as the greedy algorithm forced them to be correct.

(5) (a)
Nostrils pewter dream
Really ticking message of
Going wishing for is

(b)
Lavender simply
Same music this put away
A the rosary me

(c)
Here fall so horse trees
Yourself love when a our he
Brick band blooming the

(d)
Percolating hot
False the in find will trust a
I tell one I from

Although these examples do follow the stanza rules, they are not necessarily more coherent than before. This can be seen in the many lines that end with stop words such as ‘a’ or ‘the’. In example (6) below, some haiku written by the RNN model using this greedy algorithm are displayed.

(6) (a)
Stars eyes of yellow
Arrived snow falling for snow
Jeopardy like a

(b)
I saw waves hiss the
Metal in the dead earth in
Silent down but come

(c)
Love weighs left at first
Eyes like islands with you and
Edible seven

(d)
These nor things of home
But deceiving like down with
Five one is coming

The haiku generated by this model are the most coherent examples seen so far. While the model does occasionally produce nonsensical results, like ending the haiku with stop words, these successful examples highlight the potential for natural language processing to generate meaningful haiku.

6 Future Experiments

Though the results above are promising, there are several ways we would be interested to explore to potentially improve upon them. The first is in the training data— the Reddit data was well-suited to our purpose in terms of the syllable structure, but the haikus did not always stick to the typical themes of oneness with nature, often including profanity, poems intended as jokes, and other variations. We would like to find some datasets of haiku with more traditional subject matter to hopefully give the model a more accurate impression of haiku as a genre and result in more thematically-appropriate haiku. Additionally, the current dataset is relatively small, with only ~14,000 unique words. A larger dataset could give our model more vocabulary and thereby expand the range of poetry that it can create.

The second improvement is in training. The models had no information about how many syllables are in a given word during training, and given the inconsistency of English spelling versus pronunciation, it is difficult to learn that information accurately from unannotated data.

Ideally, we could find a way to incorporate syllable information into the model's training process, perhaps as an additional dimension on the word embedding and on the label vector. This would hopefully allow the model to learn the syllable rules for itself rather than needing the greedy generation method to enforce it externally. This might prevent the awkward line endings that the greedy method produces when searching for one-syllable words to complete a stanza.

The last improvement is increasing the complexity of the models. As mentioned previously, when we experimented with increasing the number of hidden units in the LSTM model, we found that the results were more fluent and grammatical. We cut the models off at their current sizes due to time and hardware constraints for training, but given more time and compute power, it seems likely that more complex models could capture more nuance and produce more coherent poetry. The feedforward model is very simple, with only two layers, so it would be interesting to see whether a deeper model would be able to capture more grammatical information, even if it could not learn long-term structure. The RNN could add more hidden units, perhaps up to 1000 like the feedforward network, or get deeper, for instance adding a bidirectional layer to capture backward dependencies in the structure as well.

7 Conclusion

In this paper, we have presented our approach to generating haiku using natural language processing techniques. Our experiments demonstrate the potential of neural networks to generate haiku that are interesting and creative. With future additions such as a more expansive and traditional dataset, or incorporating syllable data during training, these models could produce even better haiku. By exploring the intersection of machine learning and poetry, we hope to deepen our understanding of creativity and to inspire new approaches to art.

8 References

Academy of American Poets. (n.d.) Haiku. https://poets.org/glossary/haiku

Bfbarry. (2021). Haiku Dataset. https://www.kaggle.com/datasets/bfbarry/haiku-dataset.

Brownlee, Jason. (2020). How to Develop Word-Based Neural Language Models in Python with Keras. https://machinelearningmastery.com/develop-word-based-neural-language-models-python-keras/

Ethen8181. (n.d.). Keras RNN (Recurrent Neural Network) - Language Model. http://ethen8181.github.io/machine-learning/keras/rnn_language_model_basic_keras.html

Malik, Usman. (n.d.). Solving Sequence Problems with LSTM in Keras. https://stackabuse.com/solving-sequence-problems-with-lstm-in-keras/

Mayank, Mohit. (2020). A practical guide to RNN and LSTM in Keras. https://towardsdatascience.com/a-practical-guide-to-rnn-and-lstm-in-keras-980f176271bc

Schwartz, Jon. (2022). English phonetic and syllable count dictionary. https://www.kaggle.com/datasets/schwartstack/english-phonetic-and-syllable-count-dictionary.