Akira Nair Christine Jeong Sedong Hwang

POE-M: A RNN Language Model Generating Haikus

Introduction

While AI-powered chat-bots such as Chat-GPT are highly capable of generating structured responses to user queries, we are interested in exploring how a model can incorporate more creativity and artistic significance into its responses. We want to challenge ourselves to create an NLP model that is able to learn different writing styles and literary genres. To begin, we wanted to generate poems with a simple structure: haikus (poems with three lines with 5-7-5 syllable counts).

Methodology

Dataset

The dataset is based on a collection of haiku poems (n = 15218) scraped from reddit, and can be accessed at huanggab/reddit_haiku. The data includes an index for each poem and the complete poem as a string with a slash symbol indicating new lines in the haiku.

Model

In our ablation study, we train three generative RNN models, which all adopt the architecture shown in Figure 2. A text is converted into a vector of integers and passed as a sequence into the input layer, which takes each element and maps it into an embedding space. The embeddings of all the sequence elements are fed through a Long-Short-Term Memory (LSTM) layer. The LSTM is bidirectional, meaning it incorporates information from both the previous element and the element ahead. The output of the LSTM is passed to a hidden layer, followed by a final output layer that predicts probabilities for the next word in the sequence across the vocabulary.

Figure 1 shows the core architecture used in training our RNNs. The input sequences are embedded before being passed through a bidirectional LSTM. The output of the LSTM is passed through two dense layers to predict the next word in the sequence.

Figure 2 shows the number of occurrences of the 20 most common words in the corpus. The corpus consists of all haikus in the dataset. Stop words and punctuation were filtered out after the corpus was tokenized.

Figure 1

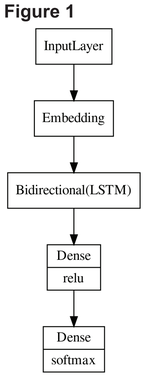
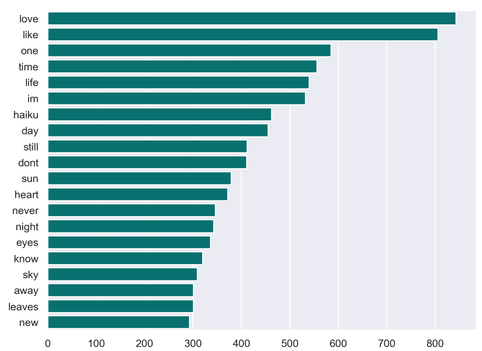


Figure 2



Model 1

The first model uses a naive sequence architecture, where sequences are generated using the whole poem. Each sequence is built by creating all possible context lengths from a text from 1 to n-1, where n is the length of the poem. More formally, the x,y pairs are generated:

$$\{(x, y)|x \subseteq s_1 : i, y = s_{i+1}, i \in \{1, ..., n-1\}\}$$

N-grams are generated one word at a time in the order that is presented in the poem. For example, consider a poem that reads “The boy eats an apple.” The sequences will look something like this:

X		Y
[The]	→	boy
[The, boy]	→	eats
[The, boy, eats]	→	an
[The, bpy, eats, an]	→	apple

The first column corresponds to the words that the model will take in (X), and the second column corresponds to the labels that the model learns to predict. We apply this technique to all the haikus in the dataset. After preparing the training data, we train our LSTM RNN model (Figure 1) using an Adam optimizer with a learning rate of 0.001. After training, we return the model and its history, which is saved as a measure of fit for the model based on the X,Y pairs. We generate poems by taking the model and a user input, which we call a “starter,” that provides the initial context for generating the poem. The starter is then padded to the input size of the model. The model predicts the next word based on the starter and concatenates this word into the growing haiku. The updated sequence is padded then passed through the RNN again, and this process repeats until the model hits the max syllable count for the line of the haiku, which will signal the model to move to the next line.

Model 2

The second model maintains the core functionalities of the first model but differs in how the data is processed by the model. We approach this second model via a line-by-line division of training, where a submodel is trained for each of the three lines of a haiku.

Figure 3

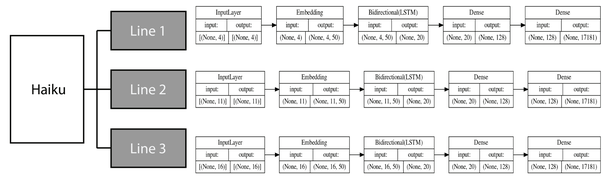


Figure 3 shows the architecture of the second and third models. Each haiku is divided into three lines, which are then passed through the core model architecture outlined in Figure 1.

To prepare our data, we create three sets of sequences for each haiku, where the first sequence corresponds to the first line, the second sequence corresponds to the second line, and the third sequence corresponds to the third line. The first sequence works exactly the same way as it does in the first model where it builds out a sequence in chronological word order. The key difference lies in how the model processes the second and third sequence. When building out the second sequence, the model uses the first sequence as a whole as the input X and the next words of the second line as the label Y. See below for a visualization of this process.

Line 1 of the poem: “The boy eats an apple”
Sequence 1: [The, boy, eats, an, apple]

Line 2 of the poem: “He then found a worm”

Sequence 2:

[Sequence 1]	→	He
[Sequence 1, He]	→	then
[Sequence 1, He, then]	→	found

Similarly, the third model uses a concatenation of lines 1 and 2 along with n-grams of line 3. We hypothesize that the line-by-line model will produce better results because of how the model processes each line; the sequences build on top of each other which means that the model can learn to recognize the context between each sequence.

Model 3

The third model extends the second model. This model improves upon the previous iteration by incorporating new-line tokens. When generating poems with the previous model, we manually input a new line when the syllable count reaches a maximum value (corresponding to the 5-7-5 haiku structure). We realized that this limited the model as it relied on our input to maintain the integrity of the haiku structure. As a result, many of the second model’s generated poems had hanging sentences and phrases. To address this limitation, we built a third model that learns where to insert new lines in a haiku. We achieve this by tokenizing a special newline symbol, adding it to the training corpus, and then training the line-by-line models. In this way, the model does not rely on human input and learns to recognize when it is best to make a new line based on the context words given.

Results

Figure 4 shows the representation of a selection of words in the corpus in the embedding layer of the naive n-gram model. TSNE (t-distributed stochastic neighbor embedding) was applied to the weights from the 50-dimensional embedding layer, to reduce 3000 words in the corpus to 2 components. The 2 components correspond to the x and y axis of the scatterplot.

Figure 5 shows the accuracy and perplexity of the three line-by-line models with and without new-line tokens. Each line model was trained for 200 epochs.

Figure 4

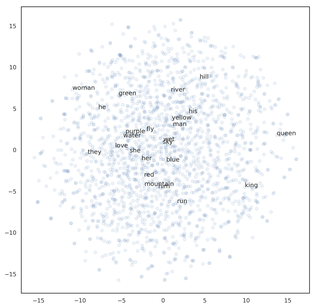
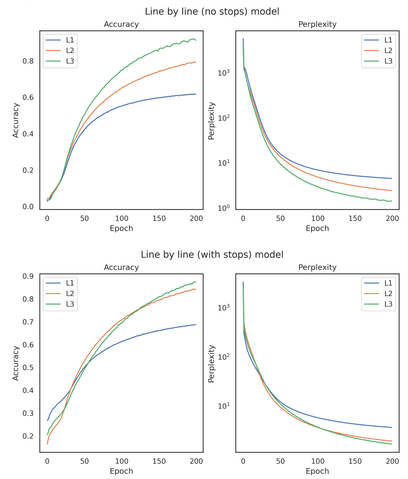


Figure 5



Reflection

Implementation

The biggest advantage of RNN models is that it gives each state the flexibility to choose context from near or far. By feeding our model the context words, our RNN models were able to collect information of words near or far from each other to generate a new poem.

Through our ablation study, we show that by parallelizing our training into three models trained on each line of the haikus, we are able to save training time while maintaining high performance. We also found that our model is also capable of learning the basic structure of a haiku, as shown by introducing stop tokens into the model’s predictions to indicate the end of a line.

LSTM has many advantages: models complex sequential data efficiently, prevents vanishing gradients, and remembers information for a longer time frame. On top of these advantages, we decided to use bidirectional LSTM because it considers the sequential data so that it looks in both directions, providing more context overall. This is where our models are able to gain a sense of the syllabic structure of haiku.

Limitation

Although some words are predicted poetically/within context, not every output haikus have constructive relationships between words and meaning, as shown by our visualizations of word embeddings. The syllable count on each line is not as accurate as we want it to be.

Future Works

If we were to develop or explore other architectures, we would have wanted to experiment with transformers. With transformers, we are able to implement the idea of self-attention where transformers can understand the relationship between sequential elements that are far from each other while paying equal attention to all the elements in the sequence. Additionally, it is more time efficient and works with any kind of sequential data, so we could expand our model to generate poems other than haikus.

We would also want to try other embedding methods that are able to capture semantics when creating vector representations that are fed into the RNN.

Our haikus originate from Reddit, which is not a literarily credible website, and although we tried to preprocess the haikus so that they have maximum 17 words (5 + 7 + 5), not all the haikus in the dataset had 5-7-5 syllables structure, and some even had emojis and strange punctuations. In the future, we could find more credible data that follows the standard haikus better than the dataset we used for this project.

We would like to consider the ethical implications of giving users the opportunity to give context for the haikus and perhaps answer questions, such as “Should we limit the input dictionary so that the users are unable to put inappropriate words?”