Poetry Generator

Term Project

Shahruz Mannan
Computing and Software Systems
Department
University of Washinton - Bothell
Bothell, WA

Abstract

This paper proposes a model for poetry generation using Long short-term memory networks with word2vec embeddings. In this process, a corpus is defined with a poem dataset. Word level tokenization is done for the vocabulary, and sequences of integers are created. Then the hyperparameters are optimized to find the best hyperparameters for the model. The best model is trained with the training data. The trained model takes a string as inputs and starts generating poems. The training metrics achieved good results. However, experimental results show that the generated poetry loses focus on the context if they become too long. Lastly, possible improvements for this model are discussed.

1 Introduction

People have written poems for a long time. In addition, many different poetic forms have been created during this time. One of the most popular poetic forms is sonnets which for example Shakespeare was well admired for. Additionally, one other poetic form which is well known is haiku which is an even older form. Some people have been able to write outstanding poems which are still considered exceptional. Nevertheless, is it possible for a computer to learn and generate new poems? What if an AI could find patterns and create outstanding poetry just like the world-known poets?

In this paper, I am concerned about creating new poetry from previously written poems by others. The poetry generator would take a string as an input which can be just one word, or a sentence and the generator would output a poem that has been generated around the string. Additionally, another goal for this project is to achieve a training accuracy of at least 80%.

This type of problem falls under the Natural language processing (NLP) domain and the subfield would be natural language generation (NLG). Natural language generation is tackling constructing computer systems that can produce text understandable for humans in human languages such as English or Chinese [1]. For generating text, multiple methodologies can be used for this task. These different methods are Markov chains [2], Recurrent neural networks (RNNs) [3], Long short-term memory networks (LSTMs) [4], and Transformers [5]. This paper proposes a model for the poetry generator which is based on LSTMs. Long short-term memory networks were chosen for this project because the networks can learn the context required for processing sequences of data. In addition, LSTMs have produced outstanding results for text generation models [6].

The rest of this paper is structured as follows. In section 2, related works will be discussed. Next, methods and the approach for creating this model will be described in section 3. This section will mention, for example, the data that will be used for this project, the data preprocessing, network architecture, and the training. Section 4 overviews the results. Subsequently, section 5 discusses the results. Lastly, section 6 concludes this paper with future works.

2 Related Work

Automated poetry generation has been gaining attraction in the last decade. In addition, it is not like everybody is trying to implement a similar poetry generator every time. Approaches differ from each other regarding language, poem style, or even just like what kind of approach is followed.

Tosa et al. (2008) introduced a Haiku generator [9]. A haiku is a Japanese poem where the poem's structure is the most important part of the text. Agarwal et al. (2020) presented an acrostic poem generator. An acrostic poem is a poem that contains a hidden message in the text. Usually, the first letters of every row spell out a hidden word [10]. An example of introducing a different concept is from Manurung et al. (2012) where they use genetic algorithms to create significant poetic text [11]. Moreover, Zhang et al. (2014) introduced a poem generator that generates text in another language. It is a Chinese poetry generator based on recurrent neural networks [12].

3 Method & Approach

3.1 Software Applications

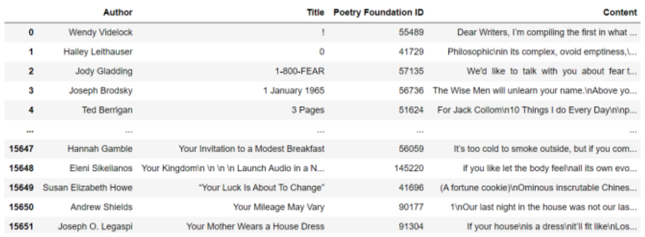
This project will be using the following software applications:

- Jupyter notebook
- Standard AI & ML Python libraries such as Pandas, NumPy, matplotlib, scikit, and TensorFlow Keras
- Some important functions that will be used for this project are Tensorboard, ModelCheckpoint, and GridSearchCV

The usage of the libraries and these functions will be shown more in-depth in the next subsections.

3.2 Data Description

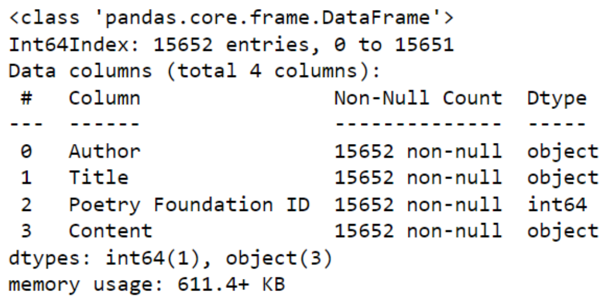
This project will be using a dataset “Complete poetryfoundation.org dataset”. This dataset was created by John Hallman. John Hallman collected most poems found on the poetryfoundation.org website and put them together in Kaggle. Complete poetryfoundation.org dataset consists of 15652 instances which should be enough data for this model to work with. The dataset contains four attributes which are Author, Title, Poetry Foundation ID, and Content. The attributes of the dataset have two different types, numerical and strings. Only the foundation id attribute is numerical, and the rest attributes are string types. For feature selection, only the “Content” attribute is relevant for this project since this is the column that contains the poems.



	Author	Title	Poetry Foundation ID	Content
0	Wendy Videlock	!	55489	Dear Writers, I'm compiling the first in what ...
1	Hailey Leithauser	0	41729	Philosophic\nin its complex, ovoid emptiness,\...
2	Jody Gladding	1-800-FEAR	57135	We'd like to talk with you about fear t...
3	Joseph Brodsky	1 January 1965	56736	The Wise Men will unlearn your name.\nAbove yo...
4	Ted Berrigan	3 Pages	51624	For Jack Collom\n10 Things I do Every Day\n\np...
...	...	...	...	...
15647	Hannah Gamble	Your Invitation to a Modest Breakfast	56059	It's too cold to smoke outside, but if you com...
15648	Eleni Sikelianos	Your Kingdom\n \n \n \n Launch Audio in a N...	145220	if you like let the body feel\nall its own evo...
15649	Susan Elizabeth Howe	"Your Luck Is About To Change"	41696	(A fortune cookie)\nOminous inscrutable Chines...
15650	Andrew Shields	Your Mileage May Vary	90177	1\nOur last night in the house was not our las...
15651	Joseph O. Legaspi	Your Mother Wears a House Dress	91304	If your house\nis a dress\nit'll fit like\nLos...

Table 1: Visual representation of the original dataset

Before processing the data, it was necessary to check if every row had a string value in the “Content” column and no null values.



```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 15652 entries, 0 to 15651
Data columns (total 4 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   Author                 15652 non-null  object
 1   Title                  15652 non-null  object
 2   Poetry Foundation ID   15652 non-null  int64
 3   Content                15652 non-null  object
dtypes: int64(1), object(3)
memory usage: 611.4+ KB
```

Image 1: Information about dataset columns

As we can see from image 1, none of the columns have missing values, thus there was no need to handle data missingness. Data processing will be discussed in the next subsection. More information about this dataset can be found in Appendix (I).

3.3 Pre-processing

Before the data was ready to be used for training, some data processing was necessary. We can see from figure 1, the original dataset had 4 columns from where the “Content” column was only relevant. Thus, after feature selection, the remaining column was the content column. Next, the data had to be checked if it contained duplicate poems. There were altogether 14 poems that showed up in the data at least once. In addition to duplicates, the first poem was not a poem but an introduction message from John Hallman. Once the duplicates and the introduction message were removed, the dataset size was reduced to 15637.

For tokenization which will be discussed more in-depth in the next section, a corpus was needed. The corpus was supposed to be defined from the poems. At first, I tried making the corpus with all the poems. However, that was impossible because the corpus would have way too many words and word2vec transformation needed an excessive amount of resources than what was provided. Therefore, 50 poems were taken from the dataset and all the poems were concatenated into one string with “\n” added in between. In addition, some text cleaning was done while concatenating the poems. For example, few of the poems had double spaces and it would show as '\xa0’ symbol. The poems were concatenated because when defining the corpus, it was easier to just split all the sentences into an array from “\n”. Now every value in the corpus would be a sentence and easier to be tokenized.

3.4 Tokenization

Before the data is ready to be used for training, one very important step needs to be done. To get our computer to understand any text, we need to define the words in a way that our machine can understand. Here is where tokenization comes in.

Tokenization is a concept of separating the text into smaller units, tokens. The tokens can be words, characters, or n-gram characters. The developer can define the size of the token and which characters are the stopping points for a token. For this project, I decided to go with word-level tokenization. Since our model is trying to create poems, the context is important and with word-level tokenization, the context is preserved much better.

TensorFlow Keras has already a prebuilt Tokenizer class, which can be used to tokenize a corpus and create a sequence of integers. Tokenizer also has a parameter called filters which filters defined characters from the texts, thus filtering by us was not that much necessary. Default values for this parameter are special characters such as punctuation, question mark, and exclamation mark.

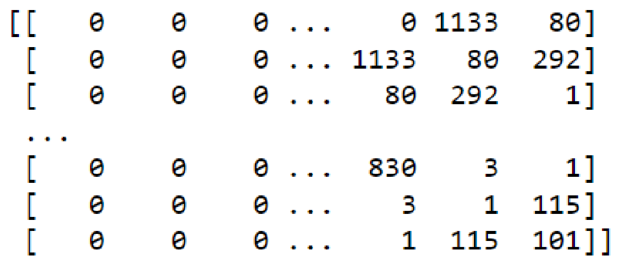
For this project, I used the fit_on_texts method on the corpus which created a vocabulary index based on the word frequency. For example, the sentence “Ball is red” would create a dictionary tokenizer.word_index[“Ball”] = 1 and tokenizer.word_index[“the”] = 2.

The next part is transforming the list of tokens into sequences of integers. Tokenizer has a texts_to_sequences method which takes each word in the text and replaces it with an integer from the word_index dictionary. This method would for example return an array [877, 54, 211, 21, 101]. When the input sequences were done, n_gram sequences were created from each integer array. This means that the previous array would produce

- [877, 54]
- [877, 54, 211]

- [877, 54, 211, 21]
- [877, 54, 211, 21, 101]

Lastly, the arrays were resized to max sequence length and padded the empty values with zeros.



```
[[   0    0    0 ...    0 1133   80]
 [   0    0    0 ... 1133   80  292]
 [   0    0    0 ...   80  292    1]
 ...
 [   0    0    0 ...  830    3    1]
 [   0    0    0 ...    3    1  115]
 [   0    0    0 ...    1  115  101]]
```

Image 2: Visual representation of input sequences

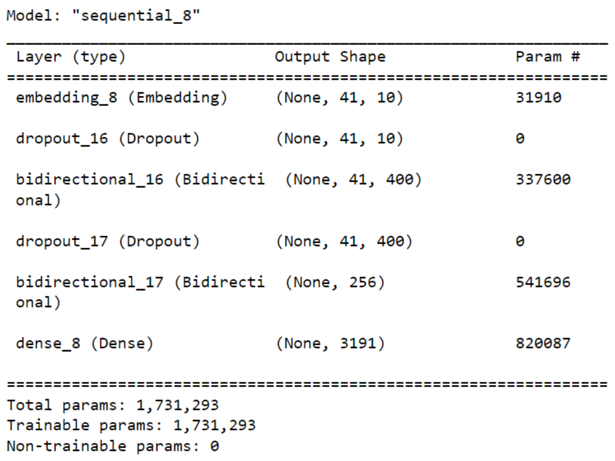
From figure 2, we can see all the input sequences after creating the n_gram sequences and padding the arrays. Now, the data is ready to be used for training. From these sequences, we can get our X and y. The X is every column except the last one and the y is the last column. The final preprocessing step for the data is to do one hot encoding for the y variable. One hot encoding converts categorical data variables into binary arrays. If we have colors red and blue, after one hot encoding red would for example [1,0] and blue would be [0,1] or vice versa. This encoding is to improve predictions [7].

3.3 Network architecture

The first layer in our network is an embedding layer. This layer turns positive integers into dense vectors of fixed size. In our case, the positive integers are the indexes of the tokens. The parameters for this layer are input dimension as the vocabulary size (3526), output dimension (100), and input length which is the max length of an input sequence. The next layer is a dropout layer with a default rate of 0.2. The dropout layer randomly sets inputs to 0 during each step depending on the rate. Dropout helps to prevent overfitting.

After the dropout layer, an LSTM layer is wrapped inside of a bidirectional layer wrapper. Bidirectional long-short term memory(bi-lstm) makes any neural network have the sequence information in both directions This makes bi-lstm a little different from general LSTMs. This layer has two parameters, units as 200 and return_sequences as true. The latter parameter defines whether to return the final output in the output sequence or the full sequence. The default activation function for LSTM layers is tanh.

Next, there is another dropout layer with a default rate of 0.3. The second bidirectional LSTM layer is next with units as 128. Finally, the last layer is a dense layer with the number of nodes as the vocabulary size. The activation function for this output layer is softmax. For compiling the model categorical cross-entropy was used as the loss function and Adam as the optimization algorithm. When using categorical cross-entropy, the targets should be in a categorical form and that is why we did one-hot encoding for y.



```
Model: "sequential_8"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_8 (Embedding)     (None, 41, 10)            31910

 dropout_16 (Dropout)        (None, 41, 10)            0

 bidirectional_16 (Bidirecti  (None, 41, 400)          337600
 onal)

 dropout_17 (Dropout)        (None, 41, 400)           0

 bidirectional_17 (Bidirecti  (None, 256)              541696
 onal)

 dense_8 (Dense)             (None, 3191)              820087

=================================================================
Total params: 1,731,293
Trainable params: 1,731,293
Non-trainable params: 0
```

Image 3: The network architecture

3.4 Hyperparameter optimization

For this project, I decided to do hyperparameter optimization with grid search to optimize the performance. For grid search, GridSearchCV from scikit learn was used. It had an efficient way to add the hyperparameters that wanted to be optimized. The hyperparameters which were selected to be optimized were number of nodes 1, number of nodes 2, dropout rate 1, and dropout rate 2. The first nodes are the nodes in the first LSTM layer and the second nodes are for the latter LSTM layer. The same thing with the dropout rates, the first rate is for the first dropout layer and the second rate for the second dropout layer. In addition, the batch size is also optimized with grid search.

Hyperparameter	Value options
LSTM nodes 1	256, 230, 200
LSTM nodes 2	128, 100, 90
Dropout rate 1	0.2, 0.25, 0.3
Dropout rate 2	0.3, 0.35, 0.37
Batch Size	32, 64, 16

Table 2: Hyperparameters and their options

All these hyperparameters were optimized by themselves because fitting the model otherwise would have been computationally heavy. As we can see from image 2, there were total parameters over 1.5 million. Grid search was done with the epoch size as 50 and the rest of these hyperparameters were kept as the default value as mentioned in the 'Network architecture' subsection. The best hyperparameters will be discussed in the Results section.

3.5 Training

Once, the hyperparameter optimizations were completed, a new model was created with the best hyperparameters.

However, instead of the number of epochs being 50, 150 epochs were used, for the model to have enough time to train. With this, training 2 callbacks were added, Tensorboard and ModelCheckpoint. Tensorboard is a visualization tool provided by Tensorflow which includes metrics summary plots, training graph visualization, and activation histograms. However, for this project, we only pay attention to the metrics summary plots. The modelCheckpoint lets us save the model or the model weights.

Every time the method is run, Tensorboard creates a new folder for every run which enables analyzing the models created at different times simultaneously in the Tensorboard. ModelCheckpoint monitors the training loss and saves when a better loss is found while training. After training the model is completed, the model is loaded from the checkpoint where the model achieved the lowest loss. Now, the model is ready to be tested out.

4 Results

Once every grid search for all the hyperparameters was done, the best hyperparameters were printed out. With those parameters, the new model was built.

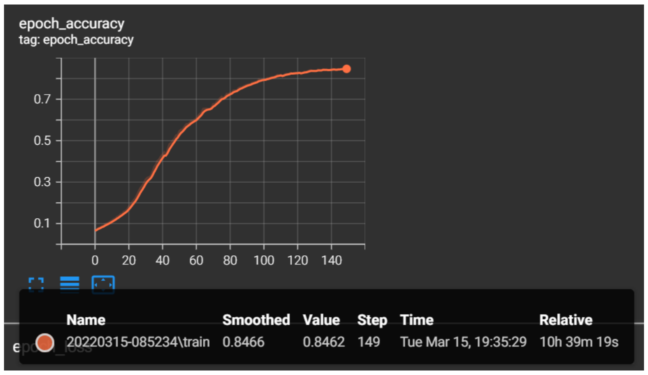
Hyperparameter	Value options
LSTM nodes 1	256, 230, **200**
LSTM nodes 2	**128**, 100, 90
Dropout rate 1	**0.2**, 0.25, 0.3
Dropout rate 2	0.3, **0.35**, 0.37
Batch Size	**32**, 64, 16

Table 4: Hyperparameter with their best values (bold)

After the training was done the best training accuracy and the best loss were printed out.

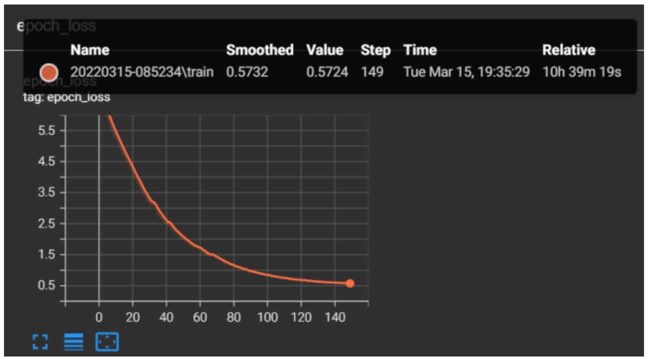
- Accuracy: 0.8485321402549744
- Loss: 0.5662724375724792

In addition to these metrics, Tensorboard logged the accuracy and the loss for every epoch and plotted two charts for both metrics.



Name	Smoothed	Value	Step	Time	Relative
20220315-085234\train	0.8466	0.8462	149	Tue Mar 15, 19:35:29	10h 39m 19s

Figure 1: Model training accuracy plot



Name	Smoothed	Value	Step	Time	Relative
20220315-085234\train	0.5732	0.5724	149	Tue Mar 15, 19:35:29	10h 39m 19s

Figure 2: Model training loss plot

Now that the model training is completed and the correct checkpoint is loaded, it's possible to do inference now. Three inferences were done with 3 different input strings. Their string was tokenized, and the sequences were padded with zeros to the size of the training data input sequences. After preprocessing the input string, the model was able to make a prediction. Next, the index just needed to be found from the index vocabulary. This returned just one word, thus this procedure needed to be run on a loop for the required number of times.

Input	Output
Help me	Help me come up with a strategy to get through this nasty pond the big lounge room not night a full moon covers it
Water	Water surrounds all shapes that enter up with the frozen seed from dazzled snows your daughter man taken to the last of the sea — voice away other than long long wrong shield in dust

The snowy mountains painted the sky	The snowy mountains painted the sky lacy with lightning and the beyond sailboats taken away away out of belief eating the masters of bluff

Table 5: Generated poems

5 Discussion

From figure 1 and figure 2 we can see that accuracy started very low and the loss started very high. Eventually, the model achieved an accuracy of over 80% which was the goal. However, even though the accuracy was decent, the poems are a bit confusing. The context seems to fall off when trying to generate longer poems. In addition, the second poem generated a special character "- "which was not supposed to happen. The Tokenizer should have filtered out all special characters when tokenization on the corpus happened.

Other observations that I noticed were some single words or a sentence are printed out multiple times consecutively. Likewise, when input is just one word, the model sometimes generates the same few lines for those words and has no context at all. There might be multiple reasons why the model did not perform as well as it should have. In the original data, before creating a corpus I noticed that the poems have typos, and some poems have numbers in them. In addition, some words in the poems were not fully on one row but the words were split, and in between, them was a "\n" symbol. From the functional side, the training data had only 50 poems worth of words and for tokenization, we used word-base tokenization which also has its flaws. A major flaw word-base tokenization has is handling without of vocabulary words. These words are countered when doing testing or inference. Because these words are not found in the vocabulary. The model does not know the best next word [8].

6 Conclusion

In this paper, a model for poetry generation was presented based on Long short-term memory networks. The model used other poems as the training data. Before training the model, the data were transformed to word2vec with Keras Tokenizer. After the transformation, a grid search was used to find the optimal hyperparameters of the poetry generator model. Lastly, training performed unexpectedly well, however the poems slightly went out of context when they started to get longer. Nevertheless, the poems were readable and could have deeper meanings to the poems.

For future work, a couple of improvements could be made to this model. The data could be cleaned much better and instead of using just 50 poems for the training data, the number of poems could be increased to enlarge the vocabulary. Instead of just generating modern poems, the model could generate other types of poems such as Japanese haikus or even sonnets.

7 References

[1] Reiter, E., & Dale, R. (1997). Building applied natural language generation systems. Natural Language Engineering, 3(1), 57-87.

[2] Harrison, Brent, Christopher Purdy, and Mark O. Riedl. "Toward automated story generation with markov chain monte carlo methods and deep neural networks." Thirteenth Artificial Intelligence and Interactive Digital Entertainment Conference. 2017.

[3] Tang, Jian, et al. "Context-aware natural language generation with recurrent neural networks." arXiv preprint arXiv:1611.09900 (2016).

[4] Wen, Tsung-Hsien, et al. "Semantically conditioned lstm-based natural language generation for spoken dialogue systems." arXiv preprint arXiv:1508.01745 (2015).

[5] Topal, M. Onat, Anil Bas, and Imke van Heerden. "Exploring transformers in natural language generation: Gpt, bert, and xlnet." arXiv preprint arXiv:2102.08036 (2021).

[6] S. Santhanam, "Context based Text-generation using LSTM networks," ArXiv Preprint ArXiv:2005.00048, 2020

[7] Rodríguez, Pau, et al. "Beyond one-hot encoding: Lower dimensional target embedding." Image and Vision Computing 75 (2018): 21-31.

[8] aravindpai. (2020, May 6). What is tokenization: Tokenization in NLP. Analytics Vidhya. Retrieved March 16, 2022, from https://www.analyticsvidhya.com/blog/2020/05/what-is-tokenization-nlp/

[9] Tosa, Naoko, Hideto Obara, and Michihiko Minoh. "Hitch haiku: An interactive supporting system for composing haiku poem." International Conference on Entertainment Computing. Springer, Berlin, Heidelberg, 2008.

[10] Agarwal, Rajat, and Katharina Kann. "Acrostic poem generation." arXiv preprint arXiv:2010.02239 (2020).

[11] Manurung, Ruli, Graeme Ritchie, and Henry Thompson. "Using genetic algorithms to create meaningful poetic text." Journal of Experimental & Theoretical Artificial Intelligence 24.1 (2012): 43-64.

[12] Zhang, Xingxing, and Mirella Lapata. "Chinese poetry generation with recurrent neural networks." Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP). 2014.

Appendix

(I) John Hallman. Complete poetryfoundation.org dataset, Version 2. Retrieved January 19, 2022, from https://www.kaggle.com/johnhallman/complete-poetryfoundationorg-dataset