# Limerick Generation Using Long Short-Term Memory and Sequence to Sequence Generation

**Nikolai Peralta** NIPERALTA@URSINUS.EDU
**Melissa Meeker** MEMEEKER@URSINUS.EDU
**Berret Babrich** BEBABRICH@URSINUS.EDU
Ursinus College, 601 E Main St., Collegeville, PA 19426

## 1. Introduction

In this paper, we will discuss an application of artificial intelligence to the generation of limerick poetry. Through explorations of Long Short-Term Memory Neural Networks and Sequence to Sequence Learning with Neural Networks, we attempt to create new limericks that learn rhyme schemes and rhythm schemes from a corpus of pre-existing limericks. Our primary goal of this project is to develop a new application of Neural Networks that has previously not been researched in the artificial intelligence community. Throughout this paper, we will discuss previous research with similar goals in the realm of poetry(Section ?), techniques for data collection (Section ?), our methods for limerick generation (Section ?), analysis of the generated limericks (Section ?), and suggestions for future work (Section ?).

## 2. Related Work

Certainly, the area of using artificial intelligence to write poetry has already been explored, so much so that a synthesis of the topic was written (Oliveira, 2013). Researchers at the University of Southern California developed any type of poem desired based on an inputted word, first by choosing related words and then building a poem up from nothing based on rhyme and syllabic structures (Ghazvininejad et al., 2016). There was also a paper which pertained even more specifically to our project idea which focused on generating just limericks alone. A researcher from the University of Leeds took a similar approach to the ones from the researches in California and generated limericks based on an entered keyword (Fleck, 2009). However, these two papers did not use any sort of neural networks in their research. An example of some researchers who did would be two researchers from the University of Coimbra, Portugal. They used a recurrent neural network to generate chines poetry (Xingxing Zhang, 2013). Our project differs in the type of neural network employed.

## 3. Methods

First, we looked for an online database of limericks. The best we could find was FunLimericks.com (FunLimericks). Since they did not make their database available for download, we built a web scraper using python's LXML library. Once the limericks were gathered into a text file, we began the process of converting them into phonemes so that we could study the effect of an LSTM on phonemic representations of words rather than the words themselves. Using keras on tensorflow, and a dictionary from Carnegie Mellon (Mellon) in combination with Python, we translated all 700 of the limericks in to their phonemic representations. In order to significantly cut down on run time, we took the obvious route of using Python's dictionary data structure. However, this meant that any words found in the limericks which were not already in the dictionary could not be converted into phonemes, and due to the informal, often dirty nature of limericks there were several hundred of these occurrences. Swear words, odd contractions, words which are typically British and not American, and simple misspellings all caused issues at this stage. Over a period of time we used various methods to fix up all these "missing" words. Once they were fixed, however, we were still not getting the results we were hoping for. Upon examining our LSTM, we realized that it was learning on a character by character basis, and we were feeding what it thought were words (the phonemes). Thus, one more massaging of our data was performed to transform each phoneme into a corresponding ASCII character. Again, Python was used in combination with a phoneme to ASCII dictionary. One more adjustement which was necessary at this stage was chaning the LSTM character input length from 128 (all possible ASCII chars) to just 85

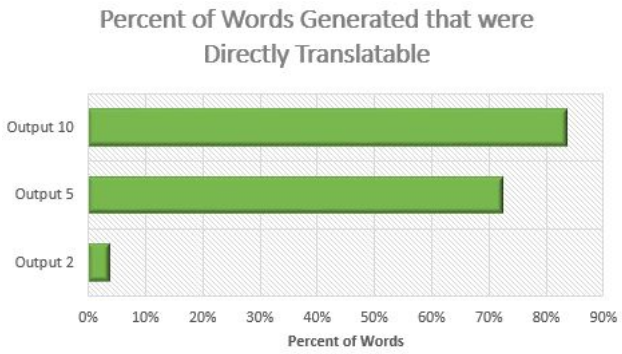
(the number of different basic phonemes).

## 4. Analysis

Output from three different phases of working with the LSTM were analyzed to determine how well the LSTM was able to generate real words from the CMU dictionary, if it learned a rhyming scheme, if it learned line length and stanzas. We also examined the output in the last set of iterations to check for over-fitting.

### 4.1. Ability to Generate Real Words

As the LSTM ran through iterations, there was an increasing trend in the percentage of generated words that existed in the CMU dictionary. After the very first attempt at using the LSTM to generate limericks, only 4% of the generated words were directly translatable. After alterations to parameters of the LSTM and 60 iterations, the percentage of directly translatable words increased to 72%. After our most recent output, which reached 30,673 iterations, the percentage of directly translatable words increased to 84%. This trend shows that a large number of iterations continues to improve the generation of readable words. This helps in generating comprehensible limericks.




### 4.2. Rhyming

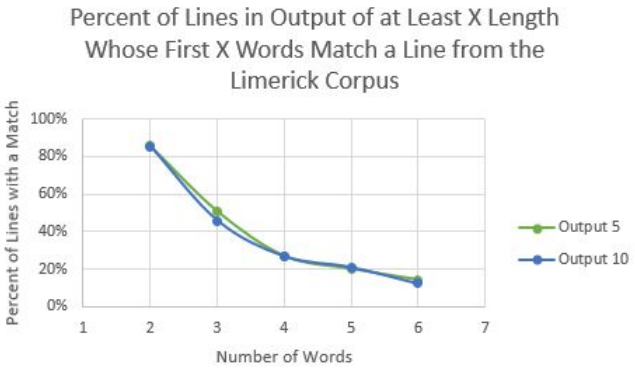
We performed a simple matching test to check for rhyming. Given a valid pair of generated lines, we checked the last word of each line. If the last phoneme of each word matched, we called this a "weak rhyme". If the last two phonemes matched, we called this a "strong rhyme". We ran this test on the last MONEY of output, and found that 6% of the pairs had weak rhymes and 1% had strong rhymes.

### 4.3. Line Length and Stanza Formation

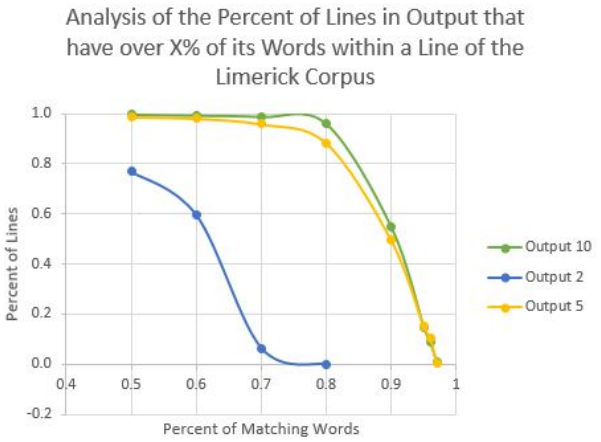
Through observations of the generated output, we were able to examine the length of stanzas that were being created in addition to the change in line lengths. Limericks should be poems of 5 lines and with line lengths of 6 or 8 syllables (the number of syllables may vary poem to poem). In the first output, we observed random break up of lines from one line stanzas to large paragraphs. In addition, the first output included very long lines which indicates too many syllables. In the last set of output that the LSTM generated, we noticed a pattern of more stanzas of appropriate length in addition to lines that averaged 7 to 9 syllables.

### 4.4. Over-Fitting

A primary concern with using the LSTM for over 30,000 iterations is the possibility of over-fitting. One important test that came back negative was determining if any lines of output were directly taken from the original limerick corpus. In all output, this was not the case. To test for more hidden forms of over-fitting, we ran two tests in comparing the second, fifth, and last set of output to the limerick corpus. First we compared each line of output against each line of the limerick corpus to search for occurrences of the first $X$-words in a line to exactly match the first $X$-words of any line in the corpus. This was executed for 2,3,4,5,6 words in the case that line length was at least that many words. This statistic is presented as the percentage of these lines related to the total number of lines in the output set.




The second test for over-fitting was in looking at all the words that each line of the output was comprised of and checking what percentage of the words existed in a single line of the limerick corpus. The largest compared percentage was recorded. We then determined what percentage of lines in the output set had over 50%, 60%, 70%, 80%, 90%, 95%, 96%, and 97% matches. The percentages increase significantly in Output 5 and Output 10. Most lines of output have between 80% to 90% matching words.




## 5. Suggestions for Future Work

The first thing future work would entail would be scraping the internet for more limericks, i.e., input data. That alone would most certainly produce interesting results. Furthermore, we would hope to develop better testing for rhyming. Currently we are only checking for the last word of each line rhyming, but we could also check for slant rhymes, which would entail finding matching phonemes in the middle of the last work of each line. Even further, we would hope to check for syllabic structures within the limericks. Currently, as mentioned, each of the generated lines has 7-9 syllables, perhaps we could study the output and see if we could find any patterns over several lines, or maybe some sort of relationship between the phonemes used and the number of syllables on the surrounding lines. Finally, we would also like to build an actual LSTM generator that loads a model and runs based off of a given input

## 6. Acknowledgements

We would like to thank Dr. Alvin Grissom for his advisement during the process of this project as well as the suggestions of our classmates.

## References

Fleck, Nicola. Computational generation of personalised limericks. *University of Leeds*, pp. 156, 2009. URL https://minerva.leeds.ac.uk/bbcswebdav/orgs/SCH_Computing/FYProj/reports/0809/Fleck.pdf.

FunLimericks. Funlimericks. URL http://www.funlimericks.com/.

Ghazvininejad, Marjan, Shi, Xing, Choi, Yejin, and Knight, Kevin. Generating topical poetry. *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, 2016.

Mellon, Carnegie. The cmu pronouncing dictionary. http://www.speech.cs.cmu.edu/cgi-bin/cmudict.

Oliveira, Hugo Goncalo. A survey on intelligent poetry generation: Languages, features, techniques, reutilisation and evaluation. *University of Coimbra, Portugal*, 2013.

Xingxing Zhang, Mirella Lapata. Chinese poetry generation with recurrent neural networks. *University of Coimbra, Portugal*, 2013.