Generating Rhyming Poetry Using LSTM Recurrent Neural Networks

by

Cole Peterson
B.Sc., University of Victoria, 2016

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

in the Department of Computer Science

Generating Rhyming Poetry Using LSTM Recurrent Neural Networks

by

Cole Peterson
B.Sc., University of Victoria, 2016

Supervisory Committee

Dr. Alona Fyshe, Supervisor
(Department of Computer Science)

Dr. Nishant Mehta, Departmental Member
(Department of Computer Science)

Supervisory Committee

Dr. Alona Fyshe, Supervisor
(Department of Computer Science)

Dr. Nishant Mehta, Departmental Member
(Department of Computer Science)

ABSTRACT

Current approaches to generating rhyming English poetry with a neural network involve constraining output to enforce the condition of rhyme. We investigate whether this approach is necessary, or if recurrent neural networks can learn rhyme patterns on their own. We compile a new dataset of amateur poetry which allows rhyme to be learned without external constraints because of the dataset's size and high frequency of rhymes. We then evaluate models trained on the new dataset using a novel framework that automatically measures the system's knowledge of poetic form and generalizability. We find that our trained model is able to generalize the pattern of rhyme, generate rhymes unseen in the training data, and also that the learned word embeddings for rhyming sets of words are linearly separable. Our model generates a couplet which rhymes 68.15% of the time; this is the first time that a recurrent neural network has been shown to generate rhyming poetry a high percentage of the time. Additionally, we show that crowd-source workers can only distinguish between our generated couplets and couplets from our dataset 63.3% of the time, indicating that our model generates poetry with coherency, semantic meaning, and fluency comparable to couplets written by humans.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Pattern recognition is at the heart of machine learning. Computers are effective at memorizing and regurgitating data, but identifying patterns in the data helps a machine learning model react appropriately to data instances it has not seen before. Thus, identifying patterns is key to the ultimate goal of machine learning: generalization.

Humans often find patterns pleasing. Music is based on periodic signals and harmonic relationships, which on their own can elicit powerful emotions in the listener. Stories often follow a common arc of the “hero’s journey”. There are many patterns in natural languages, from grammatical patterns which structure sentences to rules of spelling. Even the frequency which words occur in natural languages closely adheres to a mathematical pattern called a Zipf distibution [21]. Poetry in particular is filled with obvious patterns like meter and rhyme. Computers have proven an ability to learn patterns, but can they learn and replicate the pattern of rhyme and therefore engage in a creative poetic practice?

The idea that machines could be creative predates the physical realization of a computer. Ada Lovelace (1815-1852) imagined this possibility when “analytical engines” were just a concept, linking ideas of poetics and science[1], and suggesting that computers might be capable of composing music [39]. The use of computers as creative agents is critical in advancing the field of machine learning, as creativity is

[1]Often referred to as “the first computer programmer”, Ada’s educational history reveals why her interests combined poetry with science. Her mother, Lady Byron, a mathematician, asked for a divorce from her father, the poet Lord Byron, when Ada was an infant. Her mother forced Ada into a rigorous mathematical education, steering her from her father’s poetics, but Ada developed her own perspective she termed “poetical science” which combined both a subjective and objective reasoning about the world.

often thought as a cornerstone of intelligence [15], and so must play a role in the realization of general artificial intelligence.

In this thesis, we investigate the ability of recurrent neural networks (RNNs) to learn the pattern of rhyme in poetry and generate original poems. We develop a model which demonstrates an understanding of the pattern of rhyme, by training our RNN to predict the next word in rhyming couplets. Our model does so without any explicit programming for the task of generating rhyming poetry – the model has full agency to use any words of its choosing and violate rhyme but chooses to rhyme 68% of the time. This is the highest reported rhyming rate amongst any published work in this context. We also take a critical eye to the evaluation of poetry generation systems, showing that RNNs have the ability to "catastrophically" overfit and thus require a "plagiarism" test.

Rather than assess whether poetry generated by our system is meaningful or good (which is a tricky, ultimately subjective judgement) we reframe the question as "do the poems generated mimic the training data?" comparing our generated poems to poems in our corpus' held-out test set. We find, through human assessments, that our computer-generated poems are difficult to distinguish from poems in the dataset, as crowd-source workers are only 63.3% accurate in determining a couplet's author as either human or computer.

1.1 Introduction to Recurrent Neural Networks

Recurrent neural networks, particularly their variant LSTMs (Long Short-Term Memory) are the de facto standard technology for natural language generation tasks like machine translation [1, 6]. An RNN is a sequence modeller tasked with predicting a the probability of a token (i.e. unit of language like a word, character, or phoneme) occurring next in the sequence. To predict the next word, an RNN is given the previous token and a hidden state which encodes information about previous tokens, predicts what the next token will be and updates the hidden state. You can use this model to generate new text which resembles the text the model was trained on. In generating a text, the model samples from the probability distribution (selecting a word according to its probability of occurring next) and then feeds the selection back into the model as the token in the next step. A diagram of the basic structure of RNN generation can be seen in Figure 1.1, and more detailed information can be found in Appendix B on page 45 and in Section 2.1 on page 8.

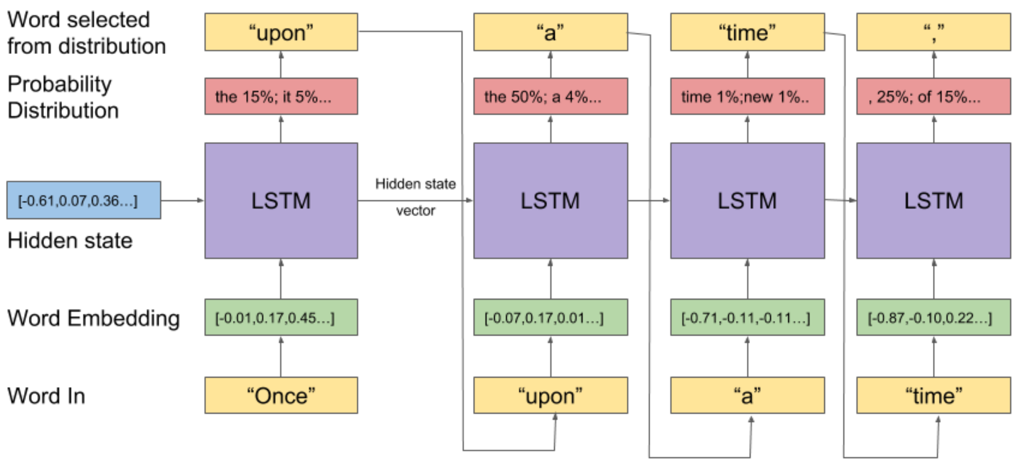


Figure 1.1: Diagram of LSTM and sampling procedure. Every word in the language model is represented by a learned vector called a word embedding (green). The LSTM predicts the probability of the tokens in the vocabulary occurring next (red), given the word embedding (green) and the hidden state (blue, omitted in between timesteps to save on space and so represented solely by an arrow). The goal when training is to minimize the cross entropy between the predicted probability distribution and the actual word which occurs next in the corpus. Once trained, we can use this trained model to generate text by selecting a word from the vocabulary according to the probability distribution of the model (the selected word appears in yellow). The word which is selected from the distribution is fed back into the LSTM at the next timestep.

There are a number of ways to generate text using an RNN. The most simple is shown in Figure 1.1, where at each time-step the next word is sampled according to the RNN's predictions. However, this greedy method of sampling from the model could forgo better outputs. For example, we might want to find the sequence the RNN assigns with the highest probability, or sequence which meets certain conditions (like rhyme). We can accomplish this by turning the problem into a search problem, which involves enumerating multiple possible outputs, and pruning any from the tree which are not of high enough probability or do not meet the desired criteria. This is the approach taken by many neural poetry generation systems [11, 10, 17, 46]. Our investigation examines whether this approach is necessary to generate rhyming poetry, or if poems can be generated in "one pass" without a search/pruning mechanism placed on top of the language model. RNNs are certainly theoretically able to generate rhyming poetry in one-pass, as RNNs are Turing Complete given enough neurons [18] and so can represent any computable function. However, we do not have good characterization of the learning capabilities of recurrent neural networks when trained using gradient descent. It is known that certain classes of problems (such as the parity problem) are not possible to learn using gradient descent on feedforward networks when the dimension is high enough [36]. Is it possible that rhyme-like patterns are difficult for an RNN to learn?

Beyond the scope of poetry generation, the challenge of generating poetry reveals technical challenges which are applicable to sequence modelling in general such as:

Poems contain long-range dependencies and require planning. There is a delay in time between the first rhyming word and the second. Additionally, the network will have to plan to be able to successfully rhyme and maintain coherence, as it is possible to "paint yourself into a corner" and reach a point where there is no word which will satisfy both rhyme and coherence. (An example illustrating "painting yourself into a corner" can be seen in Figure 4.1 on page 21.)

Poems contain both objective and subjective qualities to evaluate. Rhyme and meter have clear definition and so can be evaluated automatically and objectively, but poetry also contains more subtle and subjective patterns like semantic meaning and coherence. Poetry generation will never be "solved" in the way that there will be a perfect poem, but rather serve as a watermark for sequence modelling progress.

Poetry is a limited-resource domain. Relative to corpora built from sources like news reports or consumer reviews, there is not a large volume of readily available poetry. This challenges the sequence modeller to perform well on a low-data domain, which would allow sequence modellers to be used on many other domains.

In order to successfully generate a rhyming poem, a computer must have a mastery of language. In particular, the system must understand the pattern of rhyme: know the sets of words which rhyme with each other, and when rhyme occurs. It must also understand the patterns of language in general: language's sentence structure and content, which makes a poem coherent. Critically, this system must be able to manage coherence and rhyme simultaneously, and carefully plan the words used so as to satisfy both coherence and rhyme. We investigate whether a word-level LSTM model is capable of handling this complex task without any imposed constraints. Success would not only result in original poetry, but also indicate the power of LSTMs to identify classes of pattern and probe the limits of an LSTM's learnability.

In this thesis we train an LSTM on a corpus of rhyming poems, and develop a model which rhymes **68.15%** of the time. This is the first report of a rhyming rate for an unconstrained poetry generation system, and serves as a benchmark for progress. Our model also generates original rhymes unseen in training data, and learns parameters which further indicate a strong understanding and generalization of the pattern of rhyme. Additionally, our computer-generated couplets are not easily distinguishable by humans from couplets taken from our dataset, suggesting that our model generates couplets which are similarly coherent, fluent, and meaningful as couplets in the dataset.

Chapter 2

Overview of Computer Generated Poetry

Over the years there have been many attempts to involve computers in the creation of poetry. The vast majority of writing now uses computers to edit and print text, but we are concerned with works where the computer has direct control over what is written, and the output of the computer system is random. Early poetry generation systems used computers to make decisions based on a template. Possibly the first computer generated poetry system in English is RACTER, which wrote the book "The Policeman's Beard is Half-Constructed" by randomly choosing a word or phrase from a predetermined set of possibilities [5]. The output of the system can be entertaining, but the system itself does not demonstrate an understanding of language or poetry, the set of its possible outputs is relatively small. We aim to give our system as much agency as possible, which requires that the system understands how language and poetry works.

Unlike humans (who are prone to suffer from bouts of writer's block) computers are unafraid to make bold choices of words, sentence structure, or topic, and continue to do so endlessly. These possibly non-sensical texts can be justified by "poetic license", an indulgence by an author for the sake of effect, which breaks from a typical form or rule. From the perspective of "poetic license" a so-called wrong or incorrect choice can be reframed as creative one. Within literary theory, the reader-response school of thinking highlights that readers themselves give meaning to the static texts they consume, often drawing meaning where the author might not have intended it [3]. In the limit case of reader-response-styled thinking, a totally random string of bits could have poetic meaning. However, as Manurung points out in his thesis [27], this is problematic for a scientific assessment of poetry generation systems because if random noise qualifies as poetry, the hypothesis that a poetry generation system actually generates poetry is not falsifiable. Instead, if a poetry generation system is to take “poetic license” and break a rule of spelling, grammar, or form, it must do so knowingly and for a reason. This requires mastery of language, and demonstrable proof that a poetry generation system can “play by the rules” before it uses poetic license.

Constrained forms offer a means of proving a computer can write coherently within the rules of a language and a poetic form. Using a form has the added bonus that forms exist because they make effective poetry, and satisfying a precise form is exactly the type of things computers are good at[1]. To highlight a few poetic forms popular in computer generated poetry and their rules: haikus are made up of three lines and seventeen syllables (5/7/5), limericks five lines with an anapestic meter and the rhyme scheme (AABBA), and the Shakespearean sonnets are made up of fourteen lines of iambic pentameter with the rhyme scheme (ABAB CDCD EFEF GG). Couplets are two rhyming lines, and a quatrain is four rhyming lines (with either an ABAB, AABB, or ABBA rhyme scheme). A sonnet, therefore, is made up of three quatrains and a couplet. Although subject and tone are a key part of a poetic form’s tradition[2], the simplest definition of form distills a poem to its pattern of rhyme and rhythm.

One example of use of form is an English haiku generator, which scrapes text from blogs and then pieces together phrases using the similarity of keywords [43]. Another project uses a stochastic hill climbing algorithm to generate limericks [28]. The model has the ability to add, delete, or change words or phrases in the poem, and evolves the poem, through one of these moves, to better the poem according to a objective function based on phonetic, linguistic, and semantic features. Edits are made until no change can better the poem. Additionally, the COLIBRI poetry generation system uses case-based reasoning to generate poems according to a set of rules [8].

[1]Greene [14] notes that computers have several advantages over human writers: they can store much more perfect information in memory, and if a computer wants to know if there are any words which are five-syllables, start with “p”, and rhyme with early it can look up that information immediately.

[2]Haikus often are about nature and frequently meditate on a moment in time, limericks are often humorous, Shakespearean sonnets are often romantic.

2.1 Neural Poetry Systems

The breakthroughs of deep learning have caused many to wonder if the success could transfer to the generation of poetry. Neural network approaches are among the most sophisticated in computer generated poetry[3], as they have a large possible output space and can demonstrate an understanding (i.e. generalization ability) of language. It is these kind of poetry generation systems which this work concerns.

One fundamental decision made in using a recurrent neural network to generate text is at what level the language will be tokenized. There are many ways to tokenize English including:

Word-level models are a common way to break up language into tokens in language modelling, as words are a natural unit of language. The basis for judging industry-standard tasks like WikiText [32], machine translation [37], are with word-level models. However, there is no exhaustive list of all the words that could appear in a language. There are lots of words which might appear extremely infrequently, like proper nouns, and new words which are used in the language regularly. All word level models have a fixed vocabulary set, and when they encounter an out-of-vocabulary (OOV) word, they must replace it with a token that represents unknown words. Parts of text like numbers or URLs are problematic for a word-level model, as it is not feasible to have a token for every number and every URL. Additionally, word-level models treat each word as independent of one another and fails to see even simple relationships between words like capitalization and pluralization. Words like “apricot”, “apricots”, and “Apricots” would all be independent words to a word-level model.

Character-level models avoid the problem of out-of-vocabulary words by predicting the next character (letters, numbers, space, and punctuation) which occurs in language, spelling out words and sentences one character at a time. With this view of language, the model might be able to pick up on the meaning of prefixes or suffixes of words, whereas the word-level is blind to these kinds of distinctions. However, using a character-level tokenization increases the distance (in timesteps of the RNN) between the the parts of language. This is problematic because learning long-term dependencies is one of the biggest challenges facing recurrent neural networks.

[3]Of course, sophistication of the generation process does not guarantee literary merit.

Phoneme-level models use a phonetic representation of language which represents how a language sounds when spoken aloud. Much like how a word-level model is blind to patterns which occur within the characters of text, any orthographic (i.e. standard written) representation of English is blind to the pronunciation of words as spelling in English is only loosely phonetic[4]. This could be useful in generating rhyming poetry, as it would be more obvious to a phoneme-level model that words like "blue" (phonetically, according to the CMU pronunciation dictionary [24]: `B L UW1`) and "too" (`T UW1`) rhyme, while "slaughter" (`S L AO1 T ER0`) and "laughter" (`L AE1 F T ER0`) do not. However, phonetic models also suffer from the increase in length between dependencies, and transcribing between orthographic and phonetic English is made difficult by homophones and homonyms [17].

One work investigating the ability of LSTMs to learn poetry is by Hopkins and Kiela [17]. They take two approaches to generating poetry. The first is to train a character-level model on a general corpus of poetry (which does not have a cohesive poetic form) and then constrain that model using a probabilistic Weighted Finite State Transducer, searching through for a sequence of characters which is both coherent and meets poetic constraints. The second approach is to model language phonetically, and convert the orthographic text to a sequence of phonemes so that features like rhyme are explicit in the data seen by the model. Challenges with this approach include converting a generated sequence of phonemes back into orthographic representation which can easily be read by humans. Hopkins and Kiela provide examples of poems generated by their phonetic model which adhere to a poetic meter and rhyme but also poems which do not. Unfortunately, they provide no measure of how often rhyming properties occur.

Another state-of-the-art poetry generation system is Hafez [10], which won the 2016 Dartmouth sonnet competition[5]. Hafez accepts a word or phrase as input as basis for its poem and uses the phrase to find related words or phrases through the

[4]Consider that "though" and "sew" rhyme with each other despite not containing a single common letter, yet "rough" and "few" (which have the same suffixes as "though" and "sew") do not rhyme with each other nor do they rhyme with either "though" and "sew". Groups of letters are not necessarily pronounced the same way.

[5]This competition challenges a program to write a poem prompted by a phrase, and the results are judged by humans. Human poets too participate in the contest, and so the contest resembles a Turing Test. Although the poems written by software are convincing, no system has managed to reach indistinguishability from human poets. The contest can be found online at `http://bregman.dartmouth.edu/turingtests/poetix`

People picking up electric chronic.
The balance like a giant tidal wave,
Never ever feeling supersonic,
Or reaching any very shallow grave.

An open space between awaiting speed,
And looking at divine velocity.
A faceless nation under constant need,
Without another curiosity.

Or maybe going through the wave equation.
An ancient engine offers no momentum,
About the power from an old vibration,
And nothing but a little bit of venom.

Surrounded by a sin Omega T,
On the other side of you and me.

Figure 2.1: An example generation of Hafez, seeded with the prompt "wave".

learned word embeddings. It then searches among those related words for pairs of rhymes, and slots those words into the end-rhyme positions in the sonnet. The final step is to use the RNN to search for a fluent path, constrained by a finite state automata (FSA) which only accepts poems meeting rhyme and rhythm constraints. To do this they use a beam search technique. A successful generation can be seen in Figure 2.1.

More recently, work by Lau et al. [23] has demonstrated an ability to learn the rhyming component of their poetry generation system, generating quatrains in English, without the use of a pronunciation dictionary. The system trains on a dataset of sonnets extracted from Project Gutenberg. However, in generation, the system uses a backtracking algorithm to ensure that rhyme and rhythm constraints of form are met, resampling until the constraints are satisfied. Critically, these constraints are judged by components that are learned jointly, rather than explicitly constrained by the programmer using domain knowledge. This would allow for learning poetry even in languages where a pronunciation dictionary is not available. However, at generation time, a backtracking approach is still taken to sampling, the only difference is the condition of rhyme is judged by a learned system, rather than an explicitly coded one.

Outside of English, there have been numerous projects to generate Chinese poetry using neural networks [46, 42, 26, 19, 44]. The approaches in generating Chinese poetry are similar to English, including using a beam search on top of an LSTM to either prune poems which fail to meet required tonal and rhyming constraints [46], or to find a high probability sequence [26]. One difficulty in comparing results across languages, is that the availability of training text varies across languages. For example, work by Yi et al. [44] generates classical Chinese quatrains, and learns poetic form, relationships between words, and scores well in a qualitative human evaluation. However, the system is trained on 398,391 quatrains, which is far larger than any currently available poetic dataset in English. Additionally, lack of open-source code for these papers makes any attempt to replicate results in a new language more difficult. It is unclear how well some of the unique approaches taken to Chinese poetry would translate to English poetry.

Nonetheless, we are unaware of any work in any language which claims to generate rhyming text without using a search technique or adapting a network's structure to the poetic form. We investigate whether a backtracking approach is necessary to generate poetry which meets rhyming constraints, or if LSTMs can learn these patterns well enough to generate rhyming poetry one word at a time.

Chapter 3

Corpora

The selection of corpus used to train a model is critical to the success of a poetry generation system. This is one of the choices a programmer makes which has the largest impact on the output of the system. Once the corpus is determined, the system itself determines what features of the corpus are important and uses those features to generate new poems. The availability of large datasets has been one of the drivers of the deep learning boom [12], and generally deep learning techniques require a large amount of data. This is a challenge for our goal of generating rhyming poetry, as poetry, especially *good* poetry, will always be a low-resource domain. Poems are generally shorter than other texts like news articles or novels, and less are produced. Not that many people are poets, and those who have a knack for verse tend to not write all that much of it. Shakespeare only wrote 154 sonnets.

Without effective transfer learning, a dataset needs to be large enough to not only demonstrate the qualities that make the texts in the corpus poetic but also all information about the world that's relevant in writing a semantically meaningful poem. This kind of corpus is tremendously difficult to assemble, perhaps even impossible. Hopkins and Kiela [17] justify using external constraints on their poetry system because of this – they cannot conceivably assemble a large enough dataset for each individual poetic form, and even if you could the training time to create a whole new model is prohibitive. If you have a target metre and rhyme scheme, you can merely change the settings of your external constraints on the language model rather than assemble a new dataset and train a wholly new model on it.

However, in this work we are concerned with learning poetic form and language jointly, and so require large corpora to train on which have a consistent poetic form. As we also investigate the overlap between general English and poetic English, we also use a dataset which is assembled from Wikipedia articles. Our experimental design for this investigation (described in Section 4.1) requires us to train on models[1] which are interchangeable, which means using the same vocabulary set and tokenization patterns. Quick at-a-glance information about the datasets can be seen in Table 3.1 on page 13. Samples from all datasets can be found in Appendix A on page 42.

Table 3.1: Corpora used in experiments. All datasets are tokenized in the same way as Wiki-Text2, and also use the same dictionary for determining out-of-vocabulary words. All figures are for the full dataset which are then split into 80-10-10 train/test/validation sets. The vocabulary size for the Couplets and Limericks is obviously capped at 33,278 (the vocabulary size of WikiText-2), as only words which are in WikiText-2 can be used. Size on disk is listed uncompressed.

Corpus	Source	Size on Disk	Vocab. Size	OOV words
Penn Treebank	Various: news and technical writing	5.7MB	10,000	4.8%
WikiText-2	Wikipedia articles	13MB	33,278	2.6%
Couplets	Poetry.com user submitted poems	42MB	20,074	3.8%
Limericks	OEDILF: The Omnificent English Dictionary In Limerick Form	14MB	16,337	12.6%

3.1 Penn Treebank Dataset

The Penn Treebank dataset is an English language dataset collected from an array of sources including “IBM computer manuals, nursing notes, Wall Street Journal articles, and transcribed telephone conversations, among others” [38]. It was originally assembled for part-of-speech tagging and skeletal parsing of sentence structure, but has been adopted by language modellers as a source of text, excluding parts-of-speech and parsing information and modelling the words themselves. The language in Penn Treebank is simplified: Penn Treebank removes all punctuation, converts all words to lowercase, and replaces every number with a token representing a number.

[1]A description of the model architecture and training process is described in Section 4.1.1.

3.2 WikiText-2 Dataset

The WikiText-2 dataset [32] was introduced as a replacement for the Penn Treebank dataset. WikiText-2 is two times longer, and has a vocabulary three times larger (10,000 vs. 33,278 words) as Penn Treebank, and retains the original punctuation, case, and numbers which Penn Treebank strips. This makes WikiText more challenging and more realistic, as models must do a better job with rare words and longer dependencies. The text in WikiText is sourced from "Good" and "Featured" articles on Wikipedia. The dataset is tokenized using the Moses tokenizer [22], and all words which have less than three occurrences are considered out of vocabulary and replaced with a `<unk>` token.

3.3 Poetry.com Couplets Dataset

Poetry.com was a website which hosted user-submitted poetry until it was suddenly shutdown in 2018. Before the site was shut down, we scraped it with the goal of creating a large poetic dataset useful in generating poetry for this thesis. As we are interested in only rhyming poetry, all poems from the website were analyzed and filtered for rhyming couplets, and the couplets assembled into a new dataset. A couplet here is defined as two consecutive lines where the last words in each line rhyme. Rhymes were determined using the Python library "Pronouncing" (developed by Allison Parrish[2]), which uses the CMU pronunciation dictionary [24]. In total, 505,076 poems were scraped (428MB on disk, uncompressed), which resulted in 559,681 couplets (8,252,810 words, 41MB on disk, uncompressed). The tokenized couplet dataset (which includes the `<unk>` token for out of vocabulary words, and has spaces between punctuation) is released publicly with this project's code. To avoid issues of copyright other data (such as all of the unfiltered and untokenized poems) are available upon request.

Although this dataset is responsible for some of the advances presented within this thesis, it does have some issues:

- There is no consistency in the meter or length of lines. The dataset is only scanned for end rhymes.

[2]`https://github.com/aparrish/pronouncingpy`

- There are many spelling mistakes in the dataset which result in a higher level of out-of-vocabulary words and provide noise to the learner.
- The poems are of dubious poetic merit. *Anyone* could submit to Poetry.com, and some poems demonstrate not only lack of English skills but lack of thought all together[3].
- The couplets might not actually all rhyme due to ambiguous heteronyms (words which are spelled the same but have different pronunciations and meaning).

3.4 The Limerick Dataset

The Omnificent English Dictionary In Limerick Form[4] is a online project which aims to define all English words using limerick poems. The project is open to contributions to anyone, which are then workshopped by the community before they are accepted on the site. The dictionary is incomplete, and is currently accepting submissions for words beginning with the letters Aa through Go, and expects to be completed from Aa to Zz in September 2076. The OEDILF defines a limerick as a poem which:

1. is five lines long.
2. is based on an anapestic meter (an anapest is a metrical unit where two unstressed syllables are followed by one stressed syllable, "da-da-DAH").
3. has two different rhymes.
4. Lines 1, 2, and 5 have three anapestic feet, and rhyme with each other
5. Lines 3 and 4 have two anapestic feet, and rhyme with each other.

However, they do allow these rules to be broken if for good reason. For example, an entry for brevity is one line shorter than required[5].

[3]This excerpt is presented for evidence of the lack of quality, read at your own peril:
You are cool.
You are like stool.
You are like pool.
You are like wool. *[Author's note: this line does not rhyme with previous; would not be included in couplet dataset.]*

[4]`https://www.oedilf.com/db/Lim.php`

[5]The poem reads:
If the soul of our wit is true brevity,

As the OEDILF is a dictionary, the limericks are on the widest conceivable range of topics and so have a large vocabulary and encode a large amount of world-knowledge. As a result, the language element of the poems would be expected to be difficult to learn. The dataset we use was originally scraped as a part of the PoetRNN project[6]. We note that the dataset is constantly growing, and could be scraped again to make a larger dataset.

3.5 Analysis of Rhyme in Datasets

Some datasets are harder to learn than others. Datasets which are smaller, noisier, and higher dimension are in general more difficult to learn. Within sequence modeling the length of dependencies in the data corresponds with the difficulty of learning. Even with a simple task, there is only a small probability of successfully training a RNN on sequences with a dependency length of 20 using gradient descent[7] [2].

As rhyme is the salient pattern this work is concerned with, we take some measures of rhyme in the corpora, to gauge and explain the difficulty across corpora. Our measures are:

Entropy of the Distribution refers to computing the frequency of all the rhyme pairings (e.g. (see,me):0.76%, (me,see):0.72%, (me,be):0.61%, the three most common rhymes and their percentage of occurrence in the couplet dataset) in the dataset as a distribution and then calculating the entropy of that distribution ($H(X) = -\sum p(X) \log_k p(X)$, where k is the number of distinct rhyming pairs). Higher values denote that the rhymes are more evenly distributed, lower values mean that the rhymes are less evenly distributed. A value of 1 would mean the rhymes are uniformly distributed, 0 would mean that there is only one rhyme pair used.

Lone Rhymes refers to the percentage of rhyme pairs which only appear once in the dataset.

Then limericks are chock-full of levity.
It's no idle boast,
This one's slyer than most.

[6]Code can be found at `https://github.com/sballas8/PoetRNN`

[7]The task is the binary classification of a sequence, but the classification only depends upon the first L elements in the sequence, the remaining T not affecting the classification. By changing T the limits of an RNN's ability to learn sequences with long term dependencies can be investigated.

Distinct Rhymes refers to the total number of different rhyme pairs that appear in the dataset.

Number of Rhymes refers to the total count of rhymes in the dataset.

Words in Rhyming Position Once refers to the count of words which only appear in one rhyming pair, over the total number of words in the dataset (provided explicitly as a fraction of the count of words only in one rhyme pair over the total number of words in the vocabulary).

Average Word Dependency Length [8] refers to the number of tokens in between two rhymes including the token of the second rhyme; similarly,

Average Character Dependency Length refers to the number of characters in between two rhymes.

These can be found in Table 3.2 on page 18. By nearly every measure, the Limerick dataset is harder to learn than the Couplet dataset. There are far fewer rhymes (less data), and also far more different kinds of rhymes (higher dimension). This means that there are far more rhymes which only appear once in the dataset, whereas any model should perform better when shown more examples. We also find that the rhymes in the Limerick dataset are far more evenly distributed, in the Couplet dataset they are less distributed and more repetitive. At a glance, the Couplet dataset reads far more predictably than the Limerick dataset (as most of the topics and themes of the poems are highly cliché).

[8]Both word dependency length and character dependency length are estimates as they are actually a measure of line length. In word dependency length these measures will be off-by-one when a line in a poem ends with punctuation; with character dependency length punctuation will also cause off-by-one or more issues (as ellipses count as more than one character) but additionally it is difficult to identify exactly where a rhyme occurs in the orthographic representation of English.

Table 3.2: Statistical Rhyming Measures of the Corpora. Limerick-5 refers to the dataset described in Section 3.4, whereas Limerick-4 refers to the same dataset but only the first four lines of each poem (as the fifth and final line of a limerick rhymes with the first two lines and so affects statistics, especially dependency length). Couplets refers to the dataset described in Section 3.3. Lower values of Entropy of Distribution, Lone Rhymes, Distinct Rhymes, Words in Rhyming Position Once, Word Dependency Length, and Character Dependency length should in general make the rhyme characteristic easier to learn, while higher values of Number of Rhymes should make the dataset easier to learn. By these measures, the Couplets dataset is expected to be easier to learn than the Limericks dataset, primarily as it has more examples over a smaller set of rhymes. Description of each metric can be found in Section 3.5.

	Limerick-5	Limerick-4	Couplets
Entropy of Distribution	0.9430	0.9523	0.8225
Lone Rhymes	69.6%	72.6%	48.9%
Distinct Rhymes	158,303	91,023	29,995
Number of Rhymes	351,320	175,662	559,681
Words in Rhyming Position Once	$18.6\% = \frac{10131}{54420}$	$53.6\% = \frac{24781}{46202}$	$24.9\% = \frac{4202}{16856}$
Word Dependency Length	12.61 ± 7.60	5.72 ± 1.59	7.54 ± 2.89
Character Dependency Length	70.30 ± 41.53	31.85 ± 7.33	38.04 ± 14.28

Chapter 4

Experiments and Results

Our goal in this thesis is to show that an LSTM language model on its own can sufficiently learn rhyme, and therefore generate rhyming poetry. Unlike related work which uses an explicit structure (like a Finite State Automaton) coupled with a search technique (like beam search) to find a rhyming poem [10, 11, 17], we attempt to generate rhyming poetry one word at a time without any imposed constraint or backtracking. This approach, to train a model without incorporating domain knowledge, is inline with much of the philosophy of deep learning that a large domain of problems can be solved without hand-crafted approaches [12]. By training directly on rhyming training data and generating without any imposed constraints, we probe the limits of LSTMs on their own, to determine where hand-crafting may be necessary.

Additionally, we propose an evaluation criteria which measures:

1. Frequency with which our LSTM models adhere to the poetic form (i.e rhyme, Section 4.2).

2. The extent to which our models *generalize* the pattern of rhyme, generating original rhymes which do not appear in the training data, so as to show there is not a rote memorization of rhyme. We also investigate transitivity as an explanation of the model's abilities to generate rhymes unseen in the training data (Section 4.2.1).

3. The extent to which overfitting (plagiarism) occurs, identifying the expected and actual rates which long n-grams appear in both the training data and in text generated from our model (Section 4.4). This is motivated by our discovery that LSTM models have the ability to catastrophically overfit (Section 4.3).

4. The learned parameters of the system demonstrate the learning of rhyme (Section 4.5.1).

4.1 Learning Rhyming Poetry Experiment

A language model predicts the probability of a word occurring next in language. Recurrent Neural Networks (RNNs) are currently the dominant machine learning model used for language modeling, achieving state-of-the-art results on the major datasets like Penn Treebank [29]. The Long Short-Term Memory (LSTM) network is among the most successful types of RNNs, which was first proposed by Hochreiter and Schmidhuber [16] to help alleviate the vanishing gradient problem which inhibits standard "vanilla" RNNs from learning long-term dependencies. The LSTM was later improved by adding "forget gates" [9], which are typically assumed to exist when referring to LSTMs. The LSTM is one of the most significant advancements in language modeling, as a plain LSTM can achieve state-of-the art performance on Penn Treebank [29]. For a more thorough technical description of LSTMs,"vanilla" RNNs, and the vanishing gradient problem see the Appendix B on page 45. For a high-level overview, see Section 1.1 on page 2.

4.1.1 Training Details

Unless otherwise indicated, all LSTM language models in this thesis are trained to predict the next word in the corpus, conditioned on all previous words. We use an implementation of an LSTM language model provided by Salesforce[1] [31, 30] as the framework of our code. Our models have an embedding layer of size 400, hidden state size of 1150, three layers, and are trained with dropout rate of 0.2 applied to hidden state, and 0.65 applied to embedding layers. Gradients are calculated using the backpropogation-through-time algorithm, which "unrolls" the network over the sequence. In this way, the network resembles a standard "feedforward" network, and the gradients can be calculated in the same way. Once gradients are calculated, we update the weights using stochastic gradient descent with a small amount of weight decay (1.2×10^{-6}). The weight decay acts as a regularizer, which is effectively the same as adding an l2 penalty on weights (and also equivalent to adding a Gaussian prior with mean of zero to the weights) [12]. We clip the weights (to avoid the exploding

[1]Code is available at: `https://github.com/salesforce/awd-lstm-lm`

gradient problem) to a l2-norm of 0.25 (preserving the direction of the gradient), and use an initial learning rate of 30.0. We use the validation set to determine when to stop, choosing the model which incurs the lowest loss on the validation data as our final model. All hyperparameters are based on recommendations for WikiText-2 and we found them to be effective. As initialization of the LSTM's parameters are random, we train our couplet model twice to ensure that the results are repeatable. For the remainder of this Chapter, these are referred to as Coup-1 and Coup-2.

4.2 Rhyme Adherence in "Chaperoned" and "Generative" Testing Methods

When you see all the words of a text,
it is easy to know what comes ____.
But if you are rhyming with orange,
then sorry you're out of the ____.

Figure 4.1: An illustration of a reader's ease to fill in the blanks in rhyming position if the context is correct, but also the impossibility if the set-up does not lend itself to rhyming.

When neural language models are trained, they always predict the next token based on the real previous tokens. This is in contrast to generation time where the model is not fixed to the input data but is generating its own sequence and can conceivably select any word in the vocabulary and therefore veer onto a path where rhyme cannot be easily satisfied while also maintaining coherence. In training, the model is never able to leave the data manifold. It is unclear how this could impact performance at generation time once there is nothing enforcing a pathway to rhyme. For example, within the area of rhyming poetry, rhymes have to be set up to some extent many words before they actually happen. It is possible to "paint yourself into a corner" (as illustrated in the last two lines of Figure 4.1) and have no word which can satisfy both coherence and rhyme. *Rhyming is not done exclusively at the moment it occurs, it takes planning.*

We evaluate rhyme in two ways:

1. In Table 4.1 on page 23 we show results for a **"chaperoned" evaluation**, where we feed the models the Couplet test set and evaluate their performance *only on*

rhyming tokens. This eliminates the need for a model to plan – the rhyme is satisfiable. In particular, we calculate the average loss on rhyming words across various models and also the top 1 accuracy on rhyming words, by comparing the model's most likely next word with the test set's true rhyming word. This top-1 accuracy is the greedy selection from the model. We evaluate under this paradigm using models trained on the Couplet, Limerick, and WikiText datasets. The use of a model trained on WikiText acts as a control, to see how well a model with general English ability performs. A WikiText model might perform acceptably on a rhyming corpus if the rhymes are well "set-up" by the preceding tokens.

2. In Table 4.2 on page 24 we show results for a **"generative" evaluation**, where we generate a large number of couplets from our model to evaluate performance when the model has total freedom to select which words it would like. We generate a "generation set" of couplets from our couplet models (which have an identical word count to the Couplet test set), and evaluate the rhyme performance. The generation set is created by starting with a random initial hidden state, and then selecting the next word according to the model's predicted probabilities (for example, if the model predicted the next word "`the`" with a probability[2] of $\frac{1}{2}$, the word "`the`" would occur next half of the time. This is done in one pass selecting one word at a time with no backtracking.

Results in Tables 4.1 and 4.2 show that the LSTM models trained on the Couplet dataset have learned to exploit the rhyme properties of the text. The fact that top-1 accuracy (in Table 4.1) is this high in a language modeling context is unique, as usually these accuracies would be low. Language's goal of communication is at odds with high top-1 accuracy: if it was possible to predict the next word with high probability in general then not much information would be communicated. Additionally, the high probability of the top-1 word being in the correct rhyming class is reassuring, as it shows that some degree of generalization of the rhyme characteristic is occurring. Within language modeling some "mistakes" are better than others.

We note that although the top 1 accuracy for the Coup-1 model is 52.7%, the mistakes are often (44.7% of the time) in the rhyming class. This is unlikely to occur due to chance, as there are hundreds of different classes of rhyme in English[3].

[2]These probabilities are determined with the softmax function set with a temperature of 1

[3]In Section 4.5.1 we use the 100 most common rhyme classes in our Couplet dataset, which only

Table 4.1: Comparing the impact of learning poetic form patterns to the impact of learning a language. The columns fo the table represent various models. **Coup-1** and **Coup-2** are the same model trained with different random initial conditions trained on the Couplet dataset. **Coup-10%** is a model trained on 10% of the Couplet training data, which makes it approximately the same size as the limerick dataset, and helps gauge the impact of dataset size on rhyming performance. **WikiText** and **Limerick** are models trained on the WikiText and Limerick datasets respectively. The first three rows shows the performance of models on each test set. **Rhyming word loss**: the average cross-entropy loss on words which rhyme with the previous line over the test set of the Couplet dataset. **Top 1 accuracy on rhymes**: the percentage of the time the model's maximum predicted word is the true word. **Top 1 in rhyming class**: when the model's most likely word is not the actual word, the percentage of the time that the model's maximum predicted word rhymes with the previous word. Both top 1 accuracy on rhymes and rhyming class must throw away all cases where the words in the training data are unknown tokens because it is difficult to assess whether an out-of-vocabulary word rhymes. The best results in each row are **bolded**.

Model	Coup-1	Coup-2	Coup-10%	WikiText	Limerick
Couplet test set loss	**3.40**	3.41	3.70	5.93	6.11
WikiText test set loss	7.07	7.09	7.61	**4.19**	8.10
Limerick test set loss	4.91	4.90	5.29	5.67	**3.26**
Rhyming word loss	**2.42**	2.49	3.62	7.27	5.66
Top 1 accuracy on rhymes	**52.7%**	52.0%	38.1%	5.6%	13.9%
Top 1 in rhyming class	**44.7%**	43.1%	28.1%	2.4%	18.0%

The limited rhyming ability of the WikiText model means that a knowledge of English will only get you so far on this dataset, it is not just in the "set-up". However, the high loss values of WikiText and the Couplet dataset on each other's test set indicates limited transfer between the two datasets. This could be expected: there are many differences between the type of English used in the couplets versus on Wikipedia. There are obvious differences, for example: every poem in the Couplet dataset rhymes whereas Wikipedia does not, but also more subtle differences like many of the couplets are written in first person, whereas Wikipedia does not contain many first person sentences.

It is not surprising that all models performed the best on their own test sets (Table 4.1). However, it is interesting to note that the couplet and limerick models performed

covers 1,574 of the 20,074 words in the dataset.

Table 4.2: Results from "generative" assessment of rhyme. **Gen Set 1** and **2** refer to generations produces by couplet models 1 and 2 respectively. **Two Lines** refers to the number of poems in the dataset which are two lines long, like all couplets in the training data by definition. **Rhymes** refers to the frequency that two-lined poems rhyme the first line with the second. **Self rhymes** refers to when the last word in the first line is the same as the last word in the second line. **Original transitive rhymes** refers to the percentage of the time that rhyme pairs occur which are not in the training data, but can be explained due to at least one transitive link between rhyming words. **Fluke rhymes** refer to where a rhyme occurs which is not present in the training data but cannot be explained by a single transitive link (however, it is possibly explained by a longer transitive links).

	Gen Set 1	Gen Set 2	Test Set
Two Lines	99.67%	99.68%	100%
Rhymes	68.15%	66.84%	99.93%
Self Rhymes	9.23%	9.06%	7×10^{-5}%
Original Transitive Rhymes	0.57%	0.53%	1.48%
Fluke Rhymes	0.02%	0.04%	0.35%

significantly better on each other's datasets than the WikiText dataset. This could be due to commonalities in poetic language styles. Also, the couplet models recorded lower loss on the rhyming words than on an average of all words, showing that the rhyming word is a highly predictable part of the sequence, however the opposite was true for the Wikipedia model, which had higher loss on the rhyming word than the rest of the sequence on average.

The model trained on 10% of the data also did not exploit the property of rhyme as well as the models trained on the full dataset, incurring similar loss on rhyming words as over the rest of the sequence (3.70 on the sequence vs. 3.62 on the rhyming words, whereas model trained on full dataset incurs 3.40 and 2.42 respectively). This implies that more data affects the learning of rhyme more than the learning of the rest of the couplet. The model trained on 10% of the data also performs relatively worse than the models trained on the full data on the alternative corpora (Limerick dataset and WikiText dataset).

4.2.1 Recurrent Neural Networks Are Able To Learn Transitive Relationships

As shown in Table 4.2 on page 24, our trained models are able to generate rhymes which do not appear in the training data, but can be explained through understanding of the transitive nature of rhyme. For example, let's examine one couplet that our system generates:

```
Show me the terrain .
Makes me feel like I 'm in this foreign drain .
```

At no point in our dataset do the words `terrain` and `drain` appear as a rhyming pair. However, there is much evidence that the two words rhyme with each other because they are seen to rhyme with other words like `rain` and `again`. We identify these transitive links as follows: For every pair of rhyme (w_1, w_2) which appears in the generation set but does not appear in the training set we look up all words which w_1 rhymes with in the training data and assemble them into a set of words S_1. We do the same thing for words which rhyme with w_2 and assemble them into a set of words S_2. We then look for common words between sets S_1 and S_2 which are a transitive link between w_1 and w_2. There are almost always multiple transitive links between rhyming words which appear in the generation set. An example of all of the transitive links to explain the couplet above is seen in Table 4.3 on page 26.

In our generation set, there are 183 instances of original rhymes like the `terrain`, `drain` example in Figure 4.3 on page 26. These original rhymes occur when there are multiple different transitive links between words, and those links occur frequently. An original rhyme in our dataset has on average **10.19** linking words; the `terrain`, `drain` example has 14 linking words, represented by each row of Table 4.3. Additionally, these transitive links are supported by many rhymes, on average an original rhyme in the generation set is supported through **389.3** rhymes which develop these transitive links. This is evidence that when original rhyme does occur, there must be many examples to demonstrate the transitivity property, this is not learned by one example.

4.3 Overfitting Experiment

The ability of neural networks to generalize so effectively is mysterious. One of the strengths of neural networks is that they can represent a large class of functions, but this also means that they are prone to overfit. One experiment by Zhang et al.

Table 4.3: This table shows an example of the many transitive linking words that result in our model generating an original rhyme in the generation set. A couplet in the generated set rhymes `terrain` with `drain`, even though `terrain` and `drain` never rhyme in the training set. However, the words `terrain` and `drain` share rhyming words in the training set, which helps explain the appearance of this rhyme in our generation set. In each row, which represents a linking word, there can be up to four different appearances of the word in rhyme: corresponding to the linking word appearing with either `terrain` or `drain`, and whether the linking word appears first or second in the rhyme. For example, the word `gain` in the third row rhymes once with `terrain` in the poem:

```
Over huge mountains and rocky terrain
He was travelling , not much gain
```

and so is the reason for the (`terrain`, `gain`) pair. This table shows all of the words which could be used to transitively link `terrain` and `drain` together.

Linking word	Appearances and count in training data			
rain	terrain,rain (1)	rain,terrain (4)	drain,rain (16)	rain,drain (27)
again	again,terrain (2)	terrain,again (2)	drain,again (2)	again,drain (9)
gain	terrain,gain (1)	gain,drain (2)	drain,gain (5)	
pain	terrain,pain (5)	pain,terrain (2)	drain,pain (25)	pain,drain (60)
plane	plane,terrain (1)	drain,plane (1)	plane,drain (3)	
sustain	sustain,terrain (1)	drain,sustain (1)		
insane	terrain,insane (2)	drain,insane (9)	insane,drain (6)	
refrain	terrain,refrain (1)	drain,refrain (4)	refrain,drain (2)	
chain	chain,terrain (1)	terrain,chain (1)	chain,drain (1)	drain,chain (3)
brain	brain,terrain (3)	drain,brain (8)	brain,drain (9)	
explain	explain,terrain (1)	terrain,explain (1)	explain,drain (3)	
plain	plain,terrain (2)	drain,plain (2)		
train	train,terrain (1)	drain,train (4)	train,drain (1)	
reign	reign,terrain (3)	reign,drain (2)		

[45] trained an image classification model on random class labels uncorrelated to the image and found that the network was still able to learn to learn this completely noisy mapping perfectly. In this noisy data set, a photo of a dog might be labeled "cat", "airplane", "compact disk", or any of the 1000 categories of the image classifier (including dog). The ability of a network to learn on this completely noisy dataset is troubling, because if a network can learn arbitrary labelings (without developing compressed representations of the input), why would networks ever generalize?

If this catastrophic overfitting capacity is also present in language modeling, it would present problems for generative purposes. Generalization in a creative domain is equivalent to *originality*. Generative systems which merely replicate training data exactly are not good generative systems. We investigate if it is possible for a language model to catastrophically fit and be able to memorize arbitrary labelings of training data.

4.3.1 Experimental Design

We perform a similar experiment to Zhang et al. [45] but in the language domain. We use the Penn Treebank dataset as its size smaller size results in faster training and limits usage of computational resources. We randomly assign a "true" label for the next word according to the same unigram distribution as the training data. This gives the two data sets the same unigram distribution, and means that the ideal classifier for this corrupted dataset would simply predict this unigram distribution at every timestep, as that is the true generating distribution. We then train the model for very many epochs and observe the loss function over time on both true and corrupted labels.

4.3.2 Results

We find that, like CNN models used in image classification, neural language models also contain the capacity to catastrophically overfit. Figure 4.2 plots the loss curves over epochs of training time for the models trained on the real labels (blue line) and the random labels (orange line). Additionally, we plot the loss of state-of-the-art model on Penn Treebank (green line), as well as the ideal random classifier, which would always predict the unigram distribution of the training data at each timestep (red line) to indicate visually the point at which overfitting is occurring.

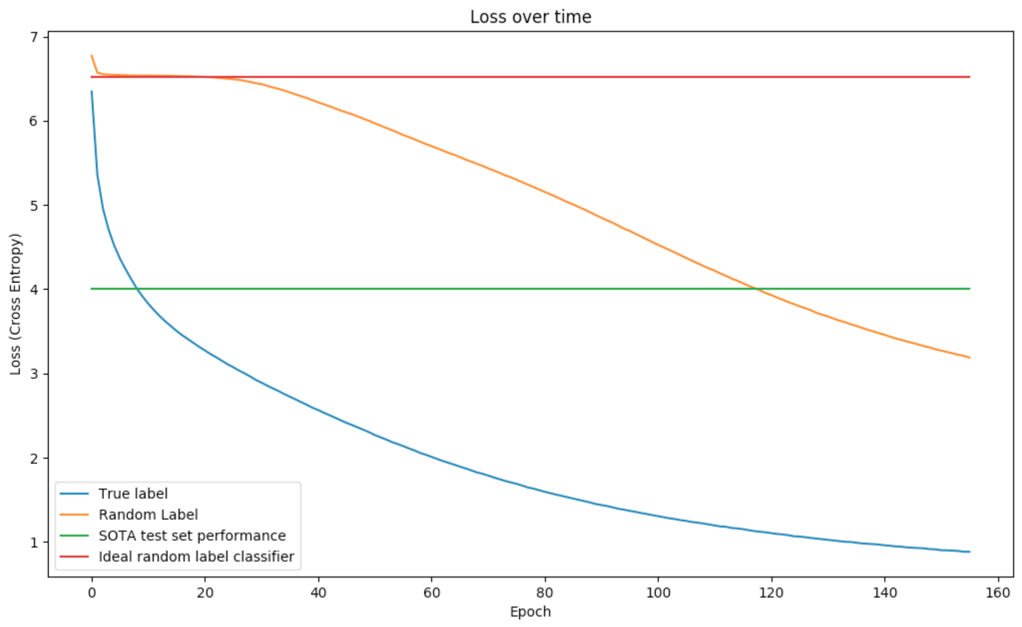


Figure 4.2: Loss of models trained on the true labels (blue line) and randomly corrupted (orange line). Both of these lines decrease below the point of overfitting, which is the green line for the blue line and the red line for the orange line respectively. The ability to progress well beyond the point of overfitting is problematic, as if an arbitrary labeling is learnable, then it's hard to explain a network's generalization abilities.

As seen in Figure 4.2, the cross entropy loss continues to drop lower than what possible – the ideal classifier would always predict the same unigram distribution at each timestep Similarly to what is reported by Zhang et al. [45], it takes longer for the network to fit random labels than it does true labels. Training was ceased before convergence due to computational limitations, however, both loss curves continue to decrease suggesting that even more dramatic overfitting is possible if training was to continue.

4.4 N-Gram Plagiarism Test

As a plagiarism test, we look at n-grams of varying lengths to check if long n-grams which appear in the training set also appear in generations. We find this type of check necessary, as a system which merely memorizes couplets from the training set would score perfectly on the rhyme evaluations, but would not be generating original poetry. Recurrent Neural Networks have a demonstrable ability to catastrophically overfit and memorize an arbitrary labeling of data (as seen in Section 4.3) and so this is a realistic danger. Good validation set loss suggests that there is no catastrophic overfitting occurring, but excessive "plagiarism" could still occur in a model's generations while maintaining good validation set loss. As a second assurance, we examine the set of n-grams present in the training set, test set, and a generation set the same size as the test set. We then examine the count of n-grams that appear in both the training set and either the test set or generation set.

4.4.1 Experimental Design

We record all of the unigrams, bigrams, trigrams etc. to 7-grams which appear in the training set, and then do the same for the test set and generation set. We then compute the intersection of the set of n-grams in the training data with the set of n-grams in the test and generation set. If the set of common n-grams is small relative to the test set, then there is no "plagiarism" occurring, if it is large then there is a chance that plagiarism is occurring. Obviously there is a degree of overlap that is expected (especially with unigrams and bigrams) so we use the test set as an independent control, as it represents the expected overlap in n-grams. Results can be seen in Table 4.4.

4.4.2 Results

Table 4.4: The raw counts for the size of the set resulting from intersecting the set of n-grams in the training data with the set of n-grams in the test set and generated sets. Higher values are bolded.

N-Gram	Test Set	Generated Set
Unigram	12,689	**14,041**
Bigram	**159,513**	125,063
Trigram	**270,203**	170,853
4-gram	**202,239**	80,997
5-gram	**126,967**	19,782
6-gram	**92,201**	3,898
7-gram	**74,304**	899

There are fewer repeated n-grams in the generated set than in the test set, and so there is no significant plagiarism occurring in the model's generated couplets. In fact, these results could suggest underfitting by the model, as "perfect" data (the test set) shows higher values of repeated n-grams.

4.5 Interpretability and Explainability of Poetic and Neural Systems

Intention in Poetry Systems

A poetry generation system, including a person with a pen and paper, does not need to be interpretable. It is possible to enjoy poetry agnostic to the process that produced it, and read any poem at face-value. However, poetic intent can be important to the art, especially within movements like conceptual poets where the "concept of the work supplants the content of the work" and call for poems to be judged on their intents over their realization. Currently, no poetry systems are able to explain their choices in the same way that a human poet would. This limits the kinds of poetic expression a computer can engage in, and so by identifying intents of the a poetry generation system we can increase its merit. We can attribute intent to examining the outputs of the system (e.g. the system almost always completes a rhyme, and therefore intends to do so, as shown is Section 4.2), or by looking inside the system to determine how it works (as described in Section 4.5.1).

Explaining and Interpreting Neural Systems

Neural networks are often thought of black-boxes which are tasked with predicting an output given an input, but can make that prediction however they'd like. The fact that intermediate computations are called "hidden" layers rather than "intermediate" layers or another alternative represents a historical indifference to how neural systems come to their outputs. However, as machines are making more life-or-death decisions (in medical contexts or autonomous vehicles), there is a growing desire to explain how these systems come to an answer. Interpretability of neural networks has become a large enough issue to warrant its own symposium at NIPS [4] which included a debate by eminent researchers on whether "interpretability is necessary in machine learning"[4]. Although interpretability is desirable, requiring it could limit the performance of computer models, and therefore make a system less safe. Birds do not need to understand the intricacies of aerodynamics to fly safely.

However, concerns over uncontrolled algorithms has led to what has been termed a "Right to Explanation" becoming a part of international policy and law. This aims to make the people who implement algorithms more responsible for their decisions, but some fear that the new regulations in the European Union's General Data Protection Regulation (GDPR), could limit the use of black-box machine learning techniques like neural networks without rapid development in interpreting their output [13]. Still, others have argued that GDPR "only mandates that data subjects receive meaningful, but properly limited, information" and more corresponds to what they term a "right to be informed" [41]. Regardless if there is a legal or scientific need for interpretability, there is a growing interest in the area.

Within the field of natural language generation, interpretability often means justifying a network's output by showing that it has learned information not explicitly required for the task. Some studies have focused on visualizing the information inside of a neural network, including work by Karpathy et al. who find and visualize individual cells inside an LSTM network and find cells in a character-level model which activate inside of quotation marks, near the end of lines, and accord among others [20]. Another example is the "unsupervised sentiment neuron", which was a single cell inside an LSTMs hidden state which correlated with the review score when trained to predict the next character of a corpus of Amazon product reviews, despite the review score never provided to the model [35]. These experiments also go to show the extent of understanding of a trained network. However, these interpretable neurons, while exciting, might not actually help to explain a network's output. One study found neurons which are interpretable to be no more important to a network's performance than neurons that are not [33].

Knowing that poetic intent is important to the craft of poetry. We look for interpretable elements of the network that showcase a knowledge of the poetic form that goes beyond the tendency to produce poetry with those properties.

[4]Rich Caruana and Patrice Simard argued for the proposition, Kilian Weinberger and Yann LeCun against. Video of the debate can be found at `https://youtu.be/2hW05ZfsUUo`.

4.5.1 Rhymes are Linearly Separable in Word Embeddings

Here we examine the learned word embeddings for evidence of encoded rhyme information, as the learning of rhyme can not only be demonstrated by the output of the system, but also by how language is represented inside the model.

Experimental Design

We select the 100 most common rhyme classes in the training data, and include every word in those rhyme classes which appear in more than 10 rhymes. This results in a total of 1,574 words[5]. As some words have multiple pronunciations and therefore multiple rhyme classes, we use the primary pronunciation (according to the CMU pronunciation dictionary) to determine rhyme class. We then train a linear support vector machine model to classify the rhyme class given the word's embedding from our Coup-1 model.

Results

The words selected are linearly separable according to their rhyme class. Using leave-one-out cross-validation as assessment (training on all words except one, leaving each word out once), we achieve accuracy of **95.5**% for classifying a word embedding to a rhyme class. As there are 100 different classes, chance would be 1%. This implies that the rhyme information is readily available in the word embeddings, as even a linear model is able to achieve high accuracy. Words which are misclassified[6] are typically from a rhyme class which are small (for example, the only other words in the rhyme class that contains `above` are `love` and `of`) or are frequently occurring words (like `in`, `it`, `or`, `this`, `that` and others).

4.6 Human Evaluation of Our Poetry Generation System

Some elements of poetry like meter and rhyme can be evaluated objectively, while other qualities such as the fluency and semantic sense are difficult to evaluate effectively. Within machine translation, measures like BLEU [34] and METEOR [7] have

[5]The rhyme classes and words are available in Appendix C.
[6]A list of misclassified words can be found in Appendix C.

been developed to help algorithmically assess qualities like fidelity to the untranslated text and fluency of the translated text, so as to limit the need for expensive human evaluations. These measures have been used to evaluate poetry generation [46], but the measures have found to not correlate with human assessments in a generative context [25]. Thus, there is currently a need to involve humans in the evaluation process.

In his seminal paper "Computing Machinery and Intelligence" [40], Alan Turing proposed his "Imitation Game" which many now refer to as the Turing Test. The game has a human judge interview two subjects of two different classes through a text chat (Turing's first example is male/female but the more famous example is human/machine), which are both trying to convince the judge that they are one class (either trying to convince the judge that they are the true male/female or they are human). If the two are indistinguishable, and the judge cannot tell which conversationalist is male/female or human/machine, then Turing posits that the two should be considered equal. If a computer machine can play chess with you, provide advice for you, and converse to the level of a human, the thought experiment concludes that denying the computer the designation of "thinking" or intelligence is anthropocentric.

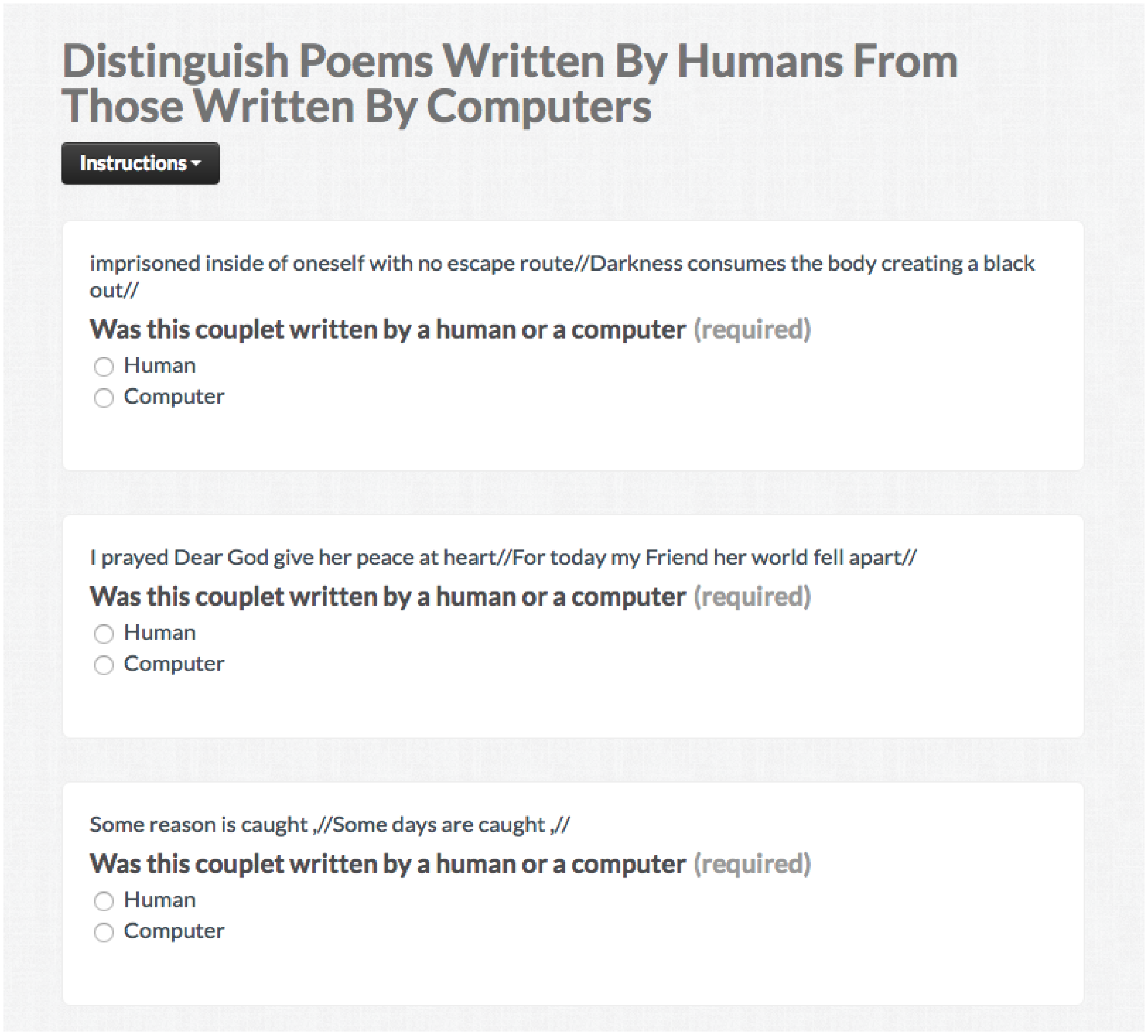
Several papers have used this framework as inspiration to evaluate their poetry generation systems, typically developing an online interface where a visitor guesses whether a poem was written by a "Bot or Not"[7]. The results generally show a large amount of confusion between poems of flesh or silicon, however, the results should interpreted cautiously, as the set-up for the test is riddled with bias. The most glaring problem is that the poems used in the interfaces are selected, without any description of the selection process, to be included in the test. The researcher who selects these poems has incentive to select the most human-like computer poems and the most computer-like human poems, as it reflects best upon their poetry generation system.

Experimental Design

We propose a test inspired by Turing's Imitation Game where humans attempt to distinguish between a randomly sampled generation of our model and a randomly selected human-written couplet from our test set. The questions are sampled uniformly, so as to be representative of a typical poem in the test set and a typical generation

[7]A non-academic example is the site `http://botpoet.com/`, which quizzes users with a curated selection of human and computer generated poetry, even with a "leaderboard" of most human-like and computer-like poems created by both humans and computers.

of our trained poetry generation system. We crowd-source human evaluators through the platform FigureEight to answer the question when presented with a couplet from one of the two datasets: "Was this couplet written by a human or a computer?". An example of the interface can bee seen in Figure 4.3 on page 34.


Distinguish Poems Written By Humans From Those Written By Computers

Instructions

imprisoned inside of oneself with no escape route//Darkness consumes the body creating a black out//

Was this couplet written by a human or a computer (required)

- Human
- Computer

I prayed Dear God give her peace at heart//For today my Friend her world fell apart//

Was this couplet written by a human or a computer (required)

- Human
- Computer

Some reason is caught ,//Some days are caught ,//

Was this couplet written by a human or a computer (required)

- Human
- Computer

Figure 4.3: The interface respondents would have seen in the experiment. Questions were given in sets of five, one of which would be a test question used for education and quality control.

A total of 500 couplets were evaluated with an even split of human-written and computer-written poetry. Our respondents assess each poem five times for a total of 2500 assessments. Additionally, there are 100 couplets which are reserved as test questions. One question in every group of five is a test question. The test questions serve two purposes:

1. **Quality Control**. If a crowd-source-worker drops below a threshold accuracy on test questions, they will no longer be able to continue assessing couplets. We set a threshold of 40% on test questions. If a respondent was unable to maintain at least 40% accuracy (lower than chance) they were not allowed to continue. This prevents a crowd-source-worker from randomly responding.

2. **Feedback**. The respondents are able to see the true answer of test questions, which might help them make more informed decisions about future couplets.

If a respondent believes that the answer to a test question is incorrect, they have the option to *contest* it, and provide a reason for their contention[8]. This feature exists because incorrect responses on test questions can prevent a respondent from continuing the work (and therefore prevent them from being paid).

Instructions were provided to the respondents about how to answer questions, the nature of the experiment, and the potential risks associated with completing the experiment. The University of Victoria's Human Research Ethics Board approved the experiment. Respondents were informed that couplets written by both humans and computers could contain an `[unk]` character. Respondents were not informed that all human-written couplets rhymed.

Results

The FigureEight respondents were able to correctly guess the authorship of a poem 63.3% of the time. This suggests that the generated poems are not only picking up on the dataset's pattern of rhyme, but also other properties of language like semantic meaning and fluency, as respondents cannot easily distinguish between the two datasets. As this is a binary decision, chance is 50%. Table 4.5 shows a confusion matrix for the two classes. Overall, human respondents have better precision on computer-written poetry than human-written poetry (64.7% vs. 61.9%), and were also slightly biased towards assessing poetry as bot-like (50.8% vs. 49.2%).

The accuracy of the respondents (63.3% over all responses) significantly differs from random guessing, meaning the respondents are able to distinguish between hu-

[8]Contested judgments included "sounds like a teenager's writing to me" to the couplet `And I am within its core from all glee//It 's a sight to see//` (actually written by a computer); "This one doesn't make sense, which is why i thought a computer wrote it. " to the couplet `To curb sense gear ,//Without mind fear .//` (actually written by a human).

man and computer written poems. The statistical significance of this is shown by assuming the null hypothesis of 50% accuracy, and calculating, through the binomial test, the probability that the number of correct and incorrect responses are explained by the null hypothesis[9]. The p-value through this test is 6.045×10^{-41}, so it is highly unlikely this distribution of correct/incorrect responses would occur if the true accuracy of the respondents was 50%. Thus, we strongly reject the null hypothesis.

Table 4.5: A confusion matrix for the human-evaluation of couplets written by humans and our Coup-1 model.

	Assessed as bot	Assessed as human	Precision
Written by a bot	815	445	64.7%
Written by a human	478	777	61.9%
Total	1260 (50.8%)	1222 (49.2%)	63.3%

Figure 4.4 on page 37 shows the distribution of respondents assessment over the couplets. There is considerable overlap and confusion between computer-generated couplets and human written couplets. For example, there are several computer-generated poems which were thought by all respondents to be written by a human, and also several human-written poems which were thought by all respondents to be written by a computer. This indicates if we hand-selected poems for a bot-or-not styled evaluation, we could make it relatively easy to show indistinguishability, and so hand-selection should not be used for this type of evaluation metric. The results also indicates that some of the human-written poems used in this dataset are of dubious poetic quality, as many are thought to be written by a computer.

The ideal language model, which perfectly imitates the Couplets dataset, would have identical distribution over human evaluations (the bars in Figure 4.4 would have the same shape). The two distributions would have a cross entropy of zero. Over this evaluation, we find a cross entropy of **2.19**, where cross entropy is defined as: $H(p,q) = -\sum_{x \in X} p(x) \log q(x)$, where p is the distribution of human assessments over human-written poems, q is the distribution of human assessments over computer-written poems, and log is the natural logarithm. This could be used as a benchmark for future evaluation of this style.

Not all respondents are equally skilled at distinguishing human from computer-

[9]Calculated by `scipy.stats.binom_test(815+777,815+777+445+478,p=0.5)`; `817+777 =` number of correct responses, `815+777+445+478 =` total number of responses. See Table 4.5.

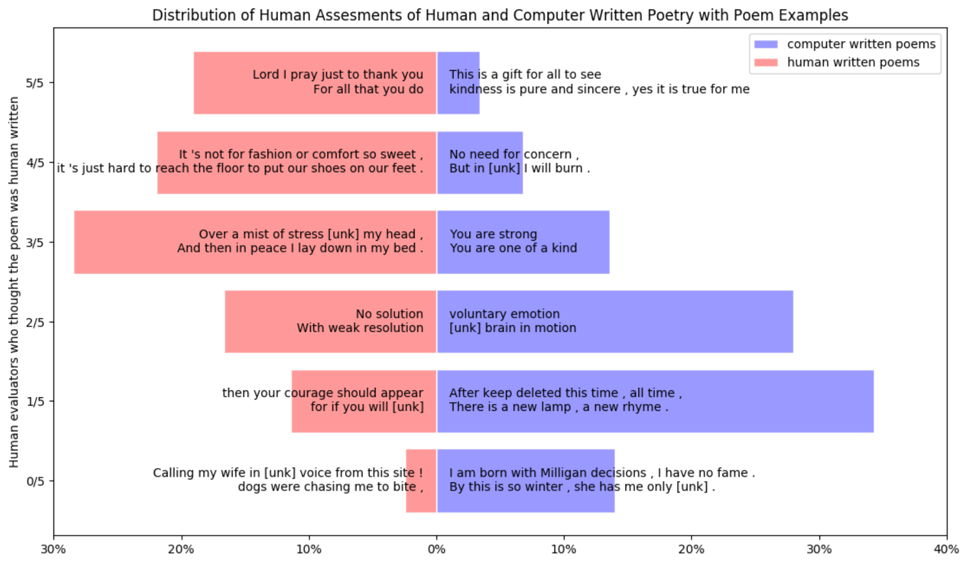


Figure 4.4: As each couplet was assessed five times by crowd-source-workers, there are six possible final assessments for each couplet: a poem can be assessed to be human zero to five times. This is represented by the six red (red: human-written poems) and six blue (blue: computer-written poems) horizontal bars. For example, the top red bar indicates that 18.7% of human-written poems are assessed by five out of five respondents to be human, which is correct. The top blue bar, on the other hand, indicates that 4.0% of computer-written poems were assessed by five out of five respondents to be human, which is incorrect, as they were actually written by a computer. One poem is randomly chosen as a representative of each bar to show the range of quality from both the test set and the computer generated poems, and overlaid on the bar it represents. For example, the couplet `voluntary emotion//[unk] brain in motion` which overlays the blue bar fourth from the top is a computer-written poem which is assessed by 2/5 respondents to be human-written (this means, by the nature of the binary choice, that it was assessed by 3/5 respondents to be computer-written). The distribution shows that in both datasets there is a broad range of poetic quality, and confusion between computer and human-written poetry.

written poems. Figure 4.5 on page 39 shows each respondent's accuracy plotted against the number of couplets they assessed. As expected, there is a range of accuracies across all 46 respondents. The highest accuracy was 90% over ten poems and the lowest accuracy was 40% over five poems. However, this range decreases as the respondents assess more poems. The highest accuracy of a respondent who has assessed more than 100 poems is 75.6% (over 225 poems), and the lowest accuracy of a respondent who has assessed more than 100 poems is 55.9% (over 213 poems). The reduction of the range as more poems are evaluated can be explained by measurements becoming statistically more accurate as the sample size is increased, or also potentially due to a small learning effect as respondents view more poems. There is no respondent who is supremely good at distinguishing the poems over a large number of couplets.

4.7 Conclusion

In this section, we have showed that an LSTM model trained on a corpus of rhyming poetry can generate its own couplets which rhyme 68.15% of the time. This is the first reported demonstration of a LSTM's ability to generate rhyme unrestricted. The new dataset of rhyming couplets (which has many repetitive rhymes) as well as using a word-level model which reduces the dependency length of rhyme, are believed to be the source of this breakthrough. Additionally, the system is able to generalize the concept of rhyme, and produces original rhymes which do not appear in the training data. These original rhymes can be explained by the model using the transitive nature of rhyme, as in examples of original rhymes there are often many transitive links between words observed in the training data. The model's parameters also indicate a strong learning of the concept of rhyme, in particular the learned word embeddings are linearly separable. The pattern of rhyme is deeply ingrained and demonstrated by our learned system. Additionally, a human-based evaluation can only distinguish between human-written and computer-written poems with 63.3% accuracy. This suggests that not only do the couplets replicate the data set's rhyming poetic quality, but also capture a similar amount of semantic meaning, fluency, and other properties of the human-written poems our model was trained on.

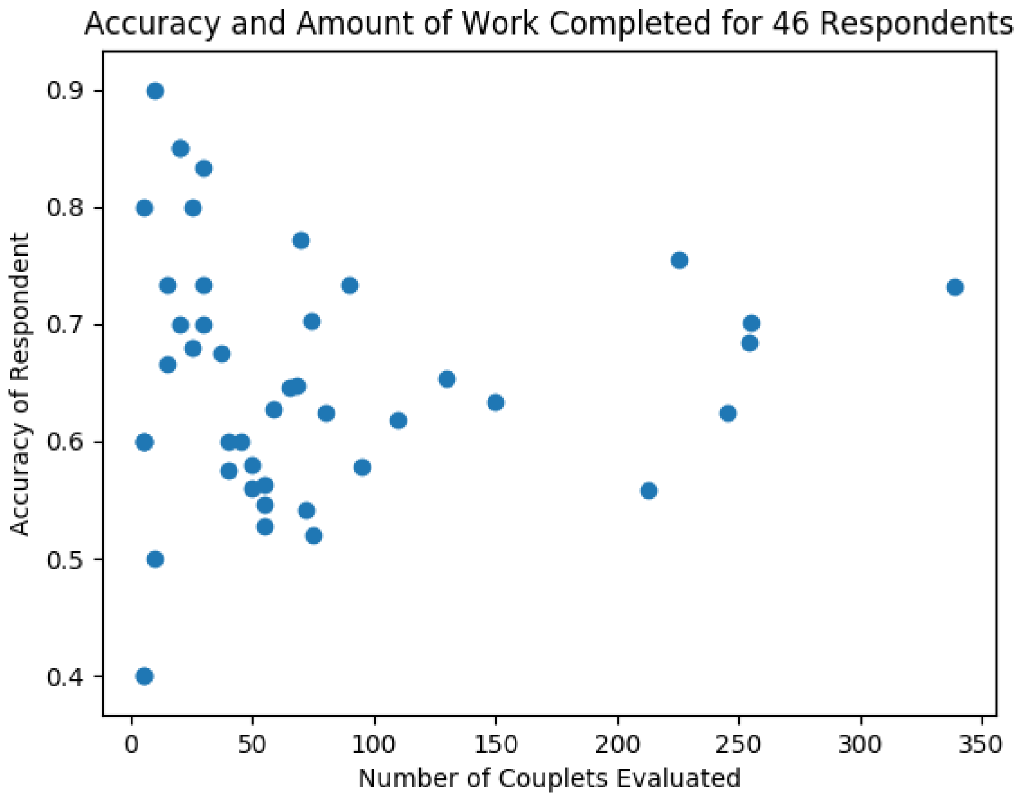


Figure 4.5: The individual accuracy of respondents varies. However, respondents who assess more poems have a smaller range of variation, likely due to measurements becoming more accurate with a larger sample size. There is no respondent who assesses a large amount of poems who shows extremely high accuracy, meaning that the two groups of poems are not easily distinguishable to any respondent.

Chapter 5

Conclusion

The generation of poetry is not a task that can be "solved" by computers in the same way that a game like chess can, because poetry can have many different objectives. We focus primarily on one objective, writing rhyme, as rhyme can be objectively evaluated and because rhyme can be pleasing whether or not the poem makes semantic sense. We use a corpus-based approach to train our models, as it gives the computer maximal agency to generate whatever it would like unconstrained. The corpus based approach also lends itself to testing with the paradigm of direct comparisons between test sets and generation sets of the same size, as corpus-based approaches aim to mimic the training set, and the test set is an independent representative sample of what we aim to generate.

We find that our generated poetry and the collection of amateur rhyming poetry we trained on are not easily distinguishable from one another. Crowd-sourced workers were able to determine the origins of a poem with only 63.3% accuracy. This low accuracy indicates that our model learns a very good representation of what it means to be a poetic couplet, including difficult to measure properties of language such as semantic meaning and fluency as well as more objective properties of language such as rhyme.

Our models succeed at learning rhyme, generating an original couplet which rhymes 68.15% of the time. Additionally, we find that the models generalize the pattern of rhyme, producing original rhymes which can largely be explained by the model understanding rhyme's transitive nature. Evidence of the understanding of rhyme is also demonstrated by the trained model's word embeddings, which are linearly separable on rhyme class. Succeeding at this task means that LSTMs have the ability to:

1. Learn transitive relationships like rhyme
2. Plan ahead to maintain coherence and also rhyme
3. Learn periodic patterns of inconsistent length

This thesis is the first reported LSTM which generates rhyming English poetry unconstrained. The use of a word-level model (which reduces the dependency length between rhymes) and the new dataset (which is larger than existing rhyming poetry datasets, as well as more repetitive) are the main drivers of this breakthrough.

However, large datasets are not always possible. Within the world of generating poetry, there might not be enough poetry written in every language and every poetic form. If computers are to engage in smaller sects of poetry, we need approaches which can work on smaller amounts of data. This work lays the foundations for future work which can succeed with less data, and also write better poems which are even less distinguishable from those written by human poets.

Appendix A

Samples from Corpora

A.1 Sample from the WikiText-2 training set

```
 As with previous <unk> Chronicles games , Valkyria Chronicles III is a
tactical role @-@ playing game where players take control of a military
unit and take part in missions against enemy forces . Stories are told through
comic book @-@ like panels with animated character portraits , with characters
speaking partially through voiced speech bubbles and partially through <unk>
text .

The player progresses through a series of linear missions , gradually unlocked
as maps that can be freely <unk> through and replayed as they are unlocked
. The route to each story location on the map varies depending on an individual
player 's approach : when one option is selected , the other is sealed off
to the player . Outside missions , the player characters rest in a camp
, where units can be customized and character growth occurs . Alongside
the main story missions are character @-@ specific sub missions relating
to different squad members . After the game 's completion , additional episodes
are unlocked , some of them having a higher difficulty than those found
in the rest of the game . There are also love simulation elements related
to the game 's two main <unk> , although they take a very minor role .
```

A.2 Sample from the Poetry.com Couplets training set

Sample from the Poetry.com Couplets training set, as tokenized according to WikiText-2.

```
It ’s the Greens who promote life ,
What ’s the solution to avoid the knife ?
#
whatever it be ,
as long as it sets you free ...
#
New and unfamiliar ground
causing thoughts of fear all around
#
Looking and Seeing much unrest
My Love and Heart is put to test
#
We were raised to make a way
Not raised to take away
#
How could I resist such <unk> ?
When she was to leave with no relation !
#
I always admired even at the cost of being criticized
As I was fond of her and totally <unk>
```

A.3 Sample from the The Omnificent English Dictionary In Limerick Form training set

Sample from the The Omnificent English Dictionary In Limerick Form, as tokenized according to WikiText-2. The first poem defines “alcoholic”, the second “castle”. The first <unk> word is frolic and the second Henry.

```
the man who becomes alcoholic
is not on a permanent <unk> .
so understand please
it 's a painful disease ,
part genetic and part metabolic .
#
it 's in castles that monarchs reside .
thick stone walls make a safe place to hide .
unless you 're , by fate ,
the eighth <unk> 's mate ,
in which case you 'd be safer outside !
```

Appendix B

Definition of an LSTM

B.1 "Vanilla" RNN

In language modelling, the input to a "vanilla" RNN is a word vector (embedding) of size M, and the input vector at time t is represented as $\mathbf{x}^t$. The model (which has a hidden state of size N) is composed of various kinds of weights including:

- Input weights $\mathbf{W} \in \mathbb{R}^{N \times M}$
- Recurrent weights $\mathbf{R} \in \mathbb{R}^{N \times N}$
- Decoding weights $\mathbf{D} \in \mathbb{R}^{N \times \text{size of vocabulary}}$
- Bias weights $\mathbf{b} \in \mathbb{R}^{N}, \mathbf{c} \in \mathbb{R}^{\text{size of vocabulary}}$

And the output of the system, which is a probability distribution over the entire vocabulary ($\mathbf{y}^t$) is calculated as follows:

$$\begin{aligned} \mathbf{h}^t &= \tanh(\mathbf{W}\mathbf{x}^t + \mathbf{R}\mathbf{h}^{t-1} + \mathbf{b}) && \textit{Update of hidden state} \\ \mathbf{y}^t &= \text{softmax}(\mathbf{D}\mathbf{h}^t + \mathbf{c}) && \textit{Decoding of hidden state} \end{aligned}$$

Where the softmax function is defined as follows: $y_k = \frac{\exp(\mathbf{x}_k)}{\sum_j^c \exp(\mathbf{x}_j)}$, and computes the probability for each word k in the vocabulary.

B.2 LSTM

The LSTM is an advancement of the "vanilla" RNN which was proposed to help solve the *vanishing gradient problem* of RNNs. The vanishing gradient problem is that the magnitude of the gradient often decays through time and does not allow a learning signal to propagate through many layers/time-steps and prevents the learning of long-term dependencies. Vanishing gradients can be caused by using activation functions which reduce the gradient[1] or because the weights in the model result in the gradient decaying. In RNNs this is especially problematic because you multiply by the same weight matrix repeatedly, meaning if the spectral radius (the maximum eigenvalue) of the recurrent matrix is < 1 the gradients will vanish. It's cousin, the *exploding gradient problem* where the magnitudes of gradients increase through layers/time can easily be solved by introducing gradient clipping.

Similarly to the "vanilla" RNN, the LSTM takes in a word vector (embedding) of size M at time t, $\mathbf{x}^t$, which it uses to update its hidden state $\mathbf{h}^t$ and is then decoded for a prediction of the word to occur next as a probability distribution over the entire vocabulary $\mathbf{y}^t$. The LSTM has a hidden state size of N, and there are V words in the vocabulary. Then we get the following weights for an LSTM layer:

- Input weights: $\mathbf{W}_z$, $\mathbf{W}_i$, $\mathbf{W}_f$, $\mathbf{W}_o \in \mathbb{R}^{N\times M}$
- Recurrent weights: $\mathbf{R}_z$, $\mathbf{R}_i$, $\mathbf{R}_f$, $\mathbf{R}_o \in \mathbb{R}^{N\times N}$
- Peephole weights: $\mathbf{p}_i$, $\mathbf{p}_f$, $\mathbf{p}_o \in \mathbb{R}^{N}$
- Bias weights: $\mathbf{b}_z$, $\mathbf{b}_i$, $\mathbf{b}_f$, $\mathbf{b}_o \in \mathbb{R}^{N}$ and $\mathbf{b}_D \in \mathbb{R}^{V}$
- Decoding weights $\mathbf{D} \in \mathbb{R}^{V\times N}$

Then the vector formulas for a vanilla LSTM layer forward pass can be written as ($\odot$ denotes point-wise multiplication):

[1]For example, the maximum of derivative the sigmoid function is $\frac{1}{4}$, and so decays the gradient by at least a quarter at each layer when the chain rule is applied to compute the gradient

$$
\begin{aligned}
\mathbf{z}^t &= g(\mathbf{W}_z\mathbf{x}^t + \mathbf{R}_z\mathbf{h}^{t-1} + \mathbf{b}_z) && \textit{block input}\\
\mathbf{i}^t &= \sigma(\mathbf{W}_i\mathbf{x}^t + \mathbf{R}_i\mathbf{h}^{t-1} + \mathbf{p}_i \odot \mathbf{c}^{t-1} + \mathbf{b}_i) && \textit{the input gate}\\
\mathbf{f}^t &= \sigma(\mathbf{W}_f\mathbf{x}^t + \mathbf{R}_f\mathbf{h}^{t-1} + \mathbf{p}_f \odot \mathbf{c}^{t-1} + \mathbf{b}_f) && \textit{the forget gate}\\
\mathbf{c}^t &= \mathbf{z}^t \odot \mathbf{i}^t + \mathbf{c}^{t-1} \odot \mathbf{f}^t && \textit{cell state}\\
\mathbf{o}^t &= \sigma(\mathbf{W}_o\mathbf{x}^t + \mathbf{R}_o\mathbf{h}^{t-1} + \mathbf{p}_o \odot \mathbf{c}^t + \mathbf{b}_o) && \textit{the output gate}\\
\mathbf{h}^t &= h(\mathbf{c}^t) \odot \mathbf{o}^t && \textit{the block output}\\
\mathbf{y}^t &= \mathrm{softmax}(\mathbf{D}\mathbf{h}^t + \mathbf{b}_D) && \textit{prediction over vocabulary}
\end{aligned}
$$

Where σ, g and h are point-wise non-linear activation functions. Typically, the *logistic sigmoid function* ($\sigma(x) = \frac{1}{1+e^{-x}}$) is used as gate activation function because of its range from zero to one, and the *hyperbolic tangent function* ($g(x) = h(x) = \tanh(x)$) is used as the activation for the block input and output.

Appendix C

Rhyme Classes and Words

The following words were used in the experiment described in Section 4.5.1.

Table C.1: The rhyming classes used in the experiment in Section 4.5.1

agree	b	be	bee	d
debris	decree	degree	disagree	e
fee	flea	flee	free	glee
guarantee	he	key	knee	me
pea	plea	sea	see	she
tea	thee	three	tree	we
wee	ye			
blew	blue	clue	crew	cue
debut	do	drew	due	ensue
few	flew	flu	glue	grew
hue	knew	lieu	new	pew
pursue	queue	renew	review	screw
shoe	subdue	sue	taboo	tattoo
threw	through	to	too	true
two	u	view	who	withdrew
woo	you	zoo		
array	astray	away	bay	bouquet
cafe	clay	convey	day	decay
delay	dismay	disobey	display	everyday

gay	gray	grey	halfway	hay
lay	may	obey	okay	pay
play	portray	pray	prey	ray
replay	say	stay	stray	sway
they	today	way	weigh	
bite	bright	delight	despite	excite
fight	flight	height	ignite	invite
knight	light	might	night	overnight
plight	polite	quite	recite	reunite
right	rite	sight	site	slight
spite	sprite	tight	tonight	upright
white	write			
apply	awry	buy	by	bye
comply	cry	defy	deny	die
dry	dye	eye	fly	goodbye
guy	hi	high	i	lie
my	nearby	rely	reply	shy
sky	spy	supply	thigh	thy
tie	try	why		
ago	below	bestow	blow	flow
glow	go	grow	know	low
no	oh	owe	pro	row
show	slow	snow	so	though
throw	toe	tow		
affair	air	aware	bare	bear
care	chair	compare	declare	despair
fair	fare	flare	hair	hare
heir	lair	mare	millionaire	pair
pear	prayer	prepare	rare	repair
scare	share	spare	square	stare
tear	their	there	unaware	unfair
wear	where			
attain	brain	campaign	cane	chain
cocaine	complain	contain	crane	disdain
domain	drain	entertain	explain	gain

grain	insane	lane	main	maintain
mane	mundane	obtain	pain	plain
plane	rain	refrain	regain	reign
remain	retain	slain	stain	strain
sustain	terrain	train	vain	vein
ace	base	case	chase	disgrace
displace	embrace	erase	face	grace
interlace	pace	place	race	replace
space	trace	vase		
apart	art	cart	chart	dart
depart	heart	part	restart	smart
start	tart			
anymore	ashore	before	bore	core
door	explore	floor	for	fore
four	ignore	lore	more	or
ore	pore	pour	restore	score
shore	sore	store	tore	war
wore	your			
ahead	bed	bread	bred	dead
fed	fled	head	instead	lead
led	overhead	read	red	said
shed	spread	stead	thread	wed
abide	applied	aside	beside	bride
coincide	cried	decide	denied	died
divide	dried	eyed	fried	guide
hide	inside	outside	pride	provide
replied	reside	ride	side	slide
tide	tied	tried	wide	worldwide
all	ball	brawl	call	fall
hall	haul	mall	small	stall
tall	wall			
begun	bun	done	fun	gun
hun	none	one	pun	run
son	spun	sun	ton	won
aligned	assigned	behind	bind	blind

combined	confined	declined	defined	designed
find	grind	hind	inclined	intertwined
kind	mankind	mind	refined	remind
resigned	signed	wind		
advise	applies	arise	cries	demise
denies	dies	disguise	eyes	flies
fries	guise	guys	highs	lies
prize	replies	revise	rise	size
skies	supplies	surprise	thighs	ties
tries	wise			
around	bound	crowned	drowned	found
ground	mound	pound	profound	round
sound	surround	wound		
aim	became	blame	came	claim
fame	flame	frame	game	lame
name	proclaim	same	shame	tame
along	belong	lifelong	long	prolong
song	strong	wrong		
ate	await	bait	create	date
debate	eight	fate	gait	gate
great	hate	innate	late	mate
plate	rate	relate	slate	state
straight	trait	wait	weight	
alone	atone	blown	bone	clone
cyclone	drone	flown	grown	known
loan	lone	overthrown	own	phone
prone	shown	stone	throne	thrown
tone	unknown	zone		
appear	beer	career	clear	dear
deer	disappear	fear	frontier	gear
mere	near	peer	pier	reappear
rear	sear	severe	sheer	sincere
spear	sphere	steer	unclear	year
band	banned	bland	brand	command
demand	expand	grand	hand	land

planned	sand	stand	understand	withstand
beat	compete	complete	concrete	deceit
defeat	discreet	eat	elite	feat
feet	fleet	heat	incomplete	meat
meet	repeat	retreat	seat	sheet
street	sweet	treat	wheat	
arrest	behest	best	blessed	breast
chest	confessed	crest	depressed	digest
distressed	dressed	expressed	guessed	guest
impressed	invest	nest	obsessed	pest
possessed	pressed	quest	request	rest
stressed	suggest	suppressed	test	unrest
vest	west			
bold	cold	controlled	fold	gold
hold	mold	mould	old	rolled
sold	told	unfold		
bowl	coal	control	goal	hole
mole	patrol	pole	role	roll
scroll	sole	soul	stole	toll
whole				
above	love	of		
amassed	blast	cast	fast	last
mast	passed	past	vast	
amend	ascend	attend	bend	blend
comprehend	contend	defend	depend	descend
end	extend	fend	friend	intend
lend	pretend	send	spend	suspend
tend	transcend	trend		
appeal	conceal	deal	feel	heal
heel	ideal	keel	meal	ordeal
real	reel	reveal	seal	steal
steel	surreal	wheel	zeal	
bell	cell	dwell	farewell	fell
hell	quell	sell	shell	smell

spell	swell	tell	well	
accommodation	admiration	adoration	anticipation	application
appreciation	aspiration	association	calculation	celebration
civilization	combination	communication	concentration	condemnation
confrontation	consideration	contemplation	continuation	conversation
creation	dedication	degradation	destination	determination
devastation	discrimination	domination	duration	education
elevation	examination	expectation	explanation	exploration
fascination	formation	foundation	frustration	generation
graduation	humiliation	imagination	imitation	incarnation
inclination	indication	infatuation	information	innovation
inspiration	interpretation	invitation	irritation	isolation
justification	liberation	limitation	location	manifestation
manipulation	medication	meditation	motivation	nation
obligation	observation	occupation	orientation	ovation
participation	population	preparation	presentation	preservation
proclamation	qualification	realization	reconciliation	recreation
relation	relaxation	reputation	reservation	restoration
revelation	salvation	segregation	sensation	separation
situation	starvation	station	transformation	vacation
vibration				
been	begin	bin	chin	fin
pin	sin	skin	spin	thin
tin	twin	violin	win	within
about	bout	devout	doubt	drought
grout	out	scout	shout	spout
stout	throughout	without		
add	bad	clad	dad	glad
had	lad	mad	pad	sad
knife	life	wife		
align	benign	decline	define	design
divine	fine	incline	intertwine	line
mine	nine	pine	refine	resign
shine	shrine	sign	spine	vine

wine				
again	den	hen	men	pen
ten	then	when		
brake	break	cake	fake	lake
make	mistake	quake	rake	retake
sake	shake	snake	stake	take
wake				
aisle	file	isle	mile	pile
smile	style	tile	while	worthwhile
are	bar	bizarre	car	far
guitar	jar	par	scar	star
superstar	tar			
appease	bees	breeze	cheese	degrees
disease	ease	fleas	flees	freeze
keys	knees	pleas	please	seas
sees	seize	these	trees	
bet	cigarette	debt	forget	fret
get	jet	let	met	net
pet	regret	reset	set	threat
upset	wet	yet		
climb	crime	lime	prime	rhyme
sublime	time			
brown	clown	crown	down	drown
gown	town			
appears	cheers	disappears	ears	fears
peers	reappears	years		
agreed	bleed	breed	deed	exceed
feed	freed	heed	indeed	need
plead	proceed	seed	speed	succeed
weed				
alive	arrive	dive	drive	five
live	revive	strive	survive	thrive
attack	back	black	crack	hack
jack	lack	pack	rack	stack

track				
bought	caught	dot	got	hot
knot	lot	not	plot	pot
rot	shot	slot	spot	trot
between	clean	fifteen	green	intervene
keen	lean	machine	mean	obscene
queen	routine	scene	screen	seen
serene	sixteen	spleen	teen	thirteen
unseen				
craze	days	gaze	haze	lays
maze	pays	phase	phrase	plays
praise	prays	raise	rays	stays
ways				
bill	drill	fill	fulfill	hill
ill	kill	mill	nil	pill
skill	spill	still	till	until
uphill	will			
beam	cream	deem	downstream	dream
esteem	extreme	redeem	scheme	scream
seam	seem	steam	stream	supreme
team	theme	upstream		
asleep	cheap	deep	keep	leap
sheep	sleep	steep	sweep	
beak	bleak	cheek	creek	critique
freak	leak	peak	seek	sneak
speak	streak	technique	unique	weak
week				
adjust	bust	crust	disgust	distrust
dust	gust	just	lust	must
rust	thrust	trust	unjust	
could	good	hood	misunderstood	should
stood	understood	wood	would	
advance	chance	dance	enhance	glance
lance	romance	stance	trance	

clutch	much	such	touch	
bring	king	ring	sing	sling
spring	string	swing	thing	wing
ban	began	can	clan	fan
man	pan	plan	ran	scan
span	tan	than	van	
acquire	attire	choir	desire	entire
fire	higher	hire	require	sire
tire	wire			
arose	blows	chose	compose	expose
flows	froze	goes	grows	impose
knows	lows	nose	pose	propose
prose	rose	rows	shows	slows
suppose	those	toes		
afraid	aid	betrayed	blade	conveyed
crusade	delayed	dismayed	displayed	fade
grade	grenade	invade	laid	made
maid	paid	parade	persuade	played
portrayed	prayed	raid	shade	stayed
trade				
affairs	bears	cares	prayers	shares
stairs	tears	theirs		
annoy	boy	coy	decoy	deploy
destroy	employ	enjoy	joy	ploy
toy				
admit	bit	commit	fit	grit
hit	it	kit	lit	permit
pit	quit	sit	slit	split
submit	unfit	wit	writ	
age	cage	engage	page	rage
sage	stage	wage		
brings	clings	kings	rings	sings
springs	stings	strings	swings	things
wings				

allow	bow	brow	cow	how
now	thou	vow		
cool	fool	pool	rule	school
stool	tool			
another	brother	mother	other	
arc	bark	dark	embark	mark
park	remark	shark	spark	stark
beams	dreams	extremes	schemes	screams
seems	streams	teams	themes	
burn	concern	earn	learn	return
stern	turn	urn		
advice	device	dice	entice	ice
lice	mice	nice	precise	price
rice	slice	twice	vice	
chess	confess	distress	dress	express
guess	impress	less	mess	possess
press	stress	success	suppress	yes
dismiss	hiss	kiss	miss	this
dawn	drawn	gone	lawn	pawn
withdrawn				
bail	detail	fail	gale	hail
jail	mail	male	nail	pale
prevail	rail	sail	sale	scale
stale	tail	tale	trail	veil
whale				
afternoon	balloon	cartoon	cocoon	dune
immune	monsoon	moon	noon	soon
spoon	tune			
brink	drink	ink	link	pink
sink	sync	think		
absurd	bird	blurred	curd	heard
herd	occurred	stirred	third	unheard

word				
bluff	enough	puff	rough	stuff
tough				
breath	death			
flower	hour	our	power	shower
tower				
cure	endure	impure	lure	mature
obscure	pure	secure	sure	unsure
book	brook	cook	hook	look
shook	took			
brought	distraught	fought	fraught	ought
sought	taught	thought	wrought	
allowed	aloud	bowed	cloud	crowd
loud	proud			
birth	earth	girth	worth	
cope	hope	pope	rope	scope
slope	soap			
at	bat	cat	chat	fat
flat	hat	mat	pat	rat
sat	that			
bent	cent	consent	dent	descent
discontent	event	extent	intent	lament
lent	meant	percent	prevent	rent
represent	resent	scent	sent	spent
tent	vent	went		
in				
choice	voice			
blues	choose	clues	confuse	fuse
lose	news	refuse	reviews	shoes
views				

The following words were misclassified:

Table C.2: Words which are misclassified when training

above	add	affairs	again	all	allow
another	at	been	bought	bow	bowed
chat	choice	clings	clutch	death	den
down	entire	five	get	girth	hen
i	in	it	just	let	life
love	lows	man	me	meant	men
my	no	not	of	oh	or
ought	our	pen	persuade	prayed	prevent
rat	rolled	sat	she	sit	slows
stairs	tear	tears	than	that	their
these	they	things	this	those	thou
thy	voice	we	without	your	

Bibliography

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[2] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, March 1994.

[3] Michael Benton. Reader-response criticism. *International companion encyclopedia of childrenís literature*, pages 112–128, 2004.

[4] Rich Caruana, William Herlands, Patrice Simard, Andrew Gordon Wilson, and Jason Yosinski. Proceedings of nips 2017 symposium on interpretable machine learning. *arXiv preprint arXiv:1711.09889*, 2017.

[5] Bill Chamberlain. *The Policeman's Beard Is Half-Constructed.* UbuWeb, Warner Books, 1984.

[6] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[7] Michael Denkowski and Alon Lavie. Meteor universal: Language specific translation evaluation for any target language. In *Proceedings of the ninth workshop on statistical machine translation*, pages 376–380, 2014.

[8] Belén Díaz-Agudo, Pablo Gervás, and Pedro A González-Calero. Poetry generation in colibri. In *European Conference on Case-Based Reasoning*, pages 73–87. Springer, 2002.

[9] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. 1999.

[10] Marjan Ghazvininejad, Xing Shi, Yejin Choi, and Kevin Knight. Generating topical poetry. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1183–1191, 2016.

[11] Marjan Ghazvininejad, Xing Shi, Jay Priyadarshi, and Kevin Knight. Hafez: an interactive poetry generation system. *Proceedings of ACL 2017, System Demonstrations*, pages 43–48, 2017.

[12] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT Press, 2016.

[13] Bryce Goodman and Seth Flaxman. Eu regulations on algorithmic decision-making and a ”right to explanation”. 38, 06 2016.

[14] Erica Greene, Tugba Bodrumlu, and Kevin Knight. Automatic analysis of rhythmic poetry with applications to generation and translation. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, EMNLP ’10, pages 524–533, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.

[15] Patricia A Haensly and Cecil R Reynolds. Creativity and intelligence. In *Handbook of creativity*, pages 111–132. Springer, 1989.

[16] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[17] Jack Hopkins and Douwe Kiela. Automatically generating rhythmic verse with neural networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 168–178, 2017.

[18] Heikki Hyötyniemi. Turing machines are recurrent neural networks. *Proceedings of step*, 96, 1996.

[19] Long Jiang and Ming Zhou. Generating chinese couplets using a statistical mt approach. In *Proceedings of the 22Nd International Conference on Computational*

Linguistics - Volume 1, COLING '08, pages 377–384, Stroudsburg, PA, USA, 2008. Association for Computational Linguistics.

[20] Andrej Karpathy, Justin Johnson, and Li Fei-Fei. Visualizing and understanding recurrent networks. *arXiv preprint arXiv:1506.02078*, 2015.

[21] Zipf George Kingsley. Selective studies and the principle of relative frequency in language, 1932.

[22] Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, et al. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th annual meeting of the ACL on interactive poster and demonstration sessions*, pages 177–180. Association for Computational Linguistics, 2007.

[23] Jey Han Lau, Trevor Cohn, Timothy Baldwin, Julian Brooke, and Adam Hammond. Deep-speare: A joint neural model of poetic language, meter and rhyme. *arXiv preprint arXiv:1807.03491*, 2018.

[24] Kevin Lenzo. The cmu pronouncing dictionary, 2007.

[25] Chia-Wei Liu, Ryan Lowe, Iulian Serban, Michael Noseworthy, Laurent Charlin, and Joelle Pineau. How NOT to evaluate your dialogue system: An empirical study of unsupervised evaluation metrics for dialogue response generation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016*, pages 2122–2132, 2016.

[26] Dayiheng Liu, Quan Guo, Wubo Li, and Jiancheng Lv. A multi-modal chinese poetry generation model. *arXiv preprint arXiv:1806.09792*, 2018.

[27] Hisar Manurung. An evolutionary algorithm approach to poetry generation. 2004.

[28] Hisar Manurung, Graeme Ritchie, and Henry Thompson. Towards a computational model of poetry generation. Technical report, The University of Edinburgh, 2000.

[29] Gábor Melis, Chris Dyer, and Phil Blunsom. On the state of the art of evaluation in neural language models. *arXiv preprint arXiv:1707.05589*, 2017.

[30] Stephen Merity, Nitish Shirish Keskar, and Richard Socher. Regularizing and Optimizing LSTM Language Models. *arXiv preprint arXiv:1708.02182*, 2017.

[31] Stephen Merity, Nitish Shirish Keskar, and Richard Socher. An Analysis of Neural Language Modeling at Multiple Scales. *arXiv preprint arXiv:1803.08240*, 2018.

[32] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

[33] Ari S Morcos, David GT Barrett, Neil C Rabinowitz, and Matthew Botvinick. On the importance of single directions for generalization. *arXiv preprint arXiv:1803.06959*, 2018.

[34] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 311–318. Association for Computational Linguistics, 2002.

[35] Alec Radford, Rafal Jozefowicz, and Ilya Sutskever. Learning to generate reviews and discovering sentiment. *arXiv preprint arXiv:1704.01444*, 2017.

[36] Shai Shalev-Shwartz, Ohad Shamir, and Shaked Shammah. Failures of gradient-based deep learning. *arXiv preprint arXiv:1703.07950*, 2017.

[37] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 3104–3112. Curran Associates, Inc., 2014.

[38] Ann Taylor, Mitchell Marcus, and Beatrice Santorini. The penn treebank: an overview. In *Treebanks*, pages 5–22. Springer, 2003.

[39] Betty Alexandra Toole. Ada byron, lady lovelace, an analyst and metaphysician. *IEEE Annals of the History of Computing*, 18(3):4–12, 1996.

[40] Alan Turing. Computing Machinery and Intelligence. *Mind*, 59(236):433, 1950.

[41] Sandra Wachter, Brent Mittelstadt, and Luciano Floridi. Why a right to explanation of automated decision-making does not exist in the general data protection regulation. *International Data Privacy Law*, 7(2):76–99, 2017.

[42] Xiaoyu Wang, Xian Zhong, and Lin Li. Generating chinese classical poems based on images. In *Proceedings of the International MultiConference of Engineers and Computer Scientists*, volume 1, 2018.

[43] M Tsan Wong, A Hon Wai Chun, Qing Li, SY Chen, and Anping Xu. Automatic haiku generation using vsm. In *WSEAS International Conference. Proceedings. Mathematics and Computers in Science and Engineering*, number 7. World Scientific and Engineering Academy and Society, 2008.

[44] Xiaoyuan Yi, Ruoyu Li, and Maosong Sun. Generating chinese classical poems with rnn encoder-decoder. In *Chinese Computational Linguistics and Natural Language Processing Based on Naturally Annotated Big Data*, pages 211–223. Springer, 2017.

[45] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*, 2016.

[46] Xingxing Zhang and Mirella Lapata. Chinese poetry generation with recurrent neural networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 670–680, 2014.