Deep Learning-based Poetry Generation Given Visual Input

Paper type: Technical Paper

Malte Loller-Andersen and Björn Gambäck
Department of Computer Science
Norwegian University of Science and Technology
7491 Trondheim, Norway
malte.loller-andersen@knowit.no gamback@ntnu.no

Abstract

The paper describes the implementation and evaluation of a system able to generate poetry satisfying rhythmical and rhyming constraints from an input image. The poetry generation system consists of a Convolutional Neural Network for image object classification, a module for finding related words and rhyme words, and a Long Short-Term Memory (LSTM) Neural Network trained on a song lyrics data set compiled specifically for this work. In total, 153 stanzas were generated and evaluated in two different experiments. The results indicate that the deep learning based system is capable of generating subjectively poetic, grammatically correct and meaningful poetry, but not on a consistent basis.

1 Introduction

Computational linguistic creativity involves theoretical study of language as well as developing computer algorithms to process and generate language. It uses a varied arsenal of machine learning, artificial intelligence and statistics to generate, predict and extract the meaning of texts, such as story narratives, jokes, analogies, word associations, and poetry. For language generation, poetry is one of the more interesting and complex challenges, since its value depends on both form and content. In this paper, a state-of-the-art poetry generator is designed and implemented. It takes an image as input and uses Inception (Szegedy et al. 2016), a pre-trained convolutional neural network (CNN) image classifier, to find objects in the image. The poetry generator then returns a short stanza based on the objects, by combining tree search with a Long-Short Term Memory (LSTM) recurrent neural network (RNN) trained on a custom made data set built from scratch from $200,000+$ songs. Instead of using rule-based and fill-in methods, the system actively predicts outcomes in a creative way. Strophically correct poems are guaranteed by combining tree search with deep learning, searching for optimal paths with suitable rhyme words.

Poetry has various different forms and is a very subjective genre, therefore a concrete definition of poetry can be hard to nail down. The definition used in this paper is:

Poem. *A set of lines satisfying rhyming and rhythmic constraints. The text generated consists of one stanza containing Y lines. The lines have to be a length of X syllables, and a line must rhyme with another line.*

By using this definition, it is clear what kind of poetry is being generated, and conclusions can be based on this. Furthermore, a stanza is a group of lines separated from others in a poem, often to shift between action, moods and thoughts. The terms stanza and poetry will be used interchangeably about the poetry generated by this system.

The next section describes the state-of-the-art in poetry generation, focusing on corpus-based methods and deep learning approaches. The data set gathering and the preprocessing of the songs are outlined in Section 3. Section 4 talks about the design and implementation of the poetry generation system. Section 5 shows the results gathered from the system, which are then discussed and evaluated in Section 6, showing that participants in a survey were able to identify a system generated poem vs a human generated poem with a 76.5% success ratio, with the system being able to generate stanzas that were perceived as aesthetically good, but not by everyone and not consistently. Finally, the conclusions are drawn and possible future work is discussed.

2 Related Work

Deep learning has proven very successful in image recognition where convolutional neural networks have heavily dominated in recent years. Most of latest efforts in poetry generation also use deep learning, although the research field itself started developing already in the 1990s, with multiple different methods being tried out. The high complexity of creative language creates substantial challenges for poetry generation, but even though the task is complex, many interesting systems have been developed. Gervás (2002) roughly divided poetry generation into four types of approaches: *template-based* (e.g., PoeTryMe by Oliveira, 2012 and Netzer et al.'s 2009 Haiku generator), *generate and test* (e.g., WASP by Gervás, 2000 and Tra-La-Lyrics by Oliveira, Cardoso, and Pereira, 2007) , *evolutionary* (e.g., POEVOLVE by Levy, 2001 and McGonagall by Manurung, 2004), and *Case-Based Reasoning* approaches (e.g., ASPERA by Gervás, 2001 and COLIBRI by Díaz-Agudo, Gervás, and González-Calero, 2002). Oliveira (2017) updates and extends Gervás' classification, while Lamb, Brown, and Clarke (2017) introduce a slightly different taxonomy. However, the focus in recent years can really be said to have shifted to two types of approaches, corpus-based and deep learning-based methods.

Corpus-based methods aim to find other poems and use their structure to create new poems. They often use multiple corpora and substitute words based on their part-of-speech (POS) tags and relevance. Colton, Goodwin, and Veale (2012) presented Full-FACE Poetry Generation, a corpus-based system which uses templates to construct poems according to constraints on rhyme, metre, stress, sentiment, word frequency, and word similarity. The system creates an aesthetic, a mood of the day, by analyzing newspapers articles, and then searches for an instantiation of a template maximizing the aesthetic.

Toivanen et al. (2012) introduced a system using two corpora, a grammar corpus and a poetry corpus, in order to provide semantic content for new poems and to generate a specific grammatical and poetic structure. The system starts by choosing a topic, specified by a single word. Topic associated words are then extracted from a background graph, a network of associations between words based on term co-occurrence. A desired length text is then randomly selected from the grammar corpus, analyzed and POS-tagged, and each word is substituted by words associated to the topic. Toivonen et al. (2013) extend this model and show how simple methods can build surprisingly good poetry.

Zhang and Lapata (2014) made one of the earliest attempts at generating poetry using *deep learning*. The poem is composed by user interaction, with the user providing different keywords, that has to be words appearing in the *ShiXueHanYing* poetic phrase taxonomy. The generator creates the first line of the poem based on the keywords and then expands the keywords into a set of related phrases that are ranked using multiple character-level neural networks. The system is very complicated and computationally heavy, using a CNN and two RNNs, but yields respectable results.

Wang, Luo, and Wang (2016) proposed an architecture using an attention-based recurrent neural network, which accepts a set of keywords as the theme and generates poems by looking at each keyword during the generation. The input sequence is converted by a bi-directional GRU (gated recurrent unit) encoder to a sequence of hidden states. These hidden states are then used to regulate a decoder that generates the poem character by character. At each time step, the prediction for the next character is based on the current status of the decoder and all the hidden states of the encoder.

Wang et al. (2016) used a planning-based recurrent neural network, inspired by the observation that a human poet often makes an outline before writing a poem. The system takes a user's input which can be either a word, a sentence or a whole document, and generates the poem in two stages: First, in the *poem planning stage* the input query is transformed into as many keywords as there are lines in the poem, using TextRank (Mihalcea and Tarau 2004) to evaluate the importance of words. However, if the user's input query is too short, keyword expansion is done using both an RNN language model and a knowledge-based method, where the latter aims to cover words on topics that are not in the training data. Then in the *poem generation stage*, the system takes all previous generated text and the keyword belonging to a given line as input, and generates the poem sequentially line by line. The generator uses the same encoder-decoder structure with GRUs as in (Wang, Luo, and Wang 2016), but slightly modified to support multiple sequences as input.

Ghazvininejad et al. (2016) proposed Hafez, a system that creates iambic pentameter poetry given a user-supplied topic. It starts by selecting a large vocabulary, and computes stress patterns for each word based on CMUdict,[1] an open-source machine-readable pronunciation dictionary for North American English that contains over $120,000$ words. A large set of words related to the user topic are retrieved using the continuous-bag-of-words model of *word2vec* (Mikolov et al. 2013). Next, rhyme words are found and put at the end of each line. Also using CMUdict, the system tries to find rhyming words with related words. However, as a fall-back for rare topics, fixed pairs of often used words are added. A Finite-state-acceptor (FSA) is built, with a path for every conceivable sequence of vocabulary words that obeys formal rhythm constraints. A path through the FSA is selected using a RNN for scoring the final outcome. The RNN uses a two layer recurrent neural network with long short-term memory, trained on a corpus of $94,882$ English songs.

Most similar to the present work is the recent effort by Xu et al. (2018) to use an encoder-decoder model to generate Chinese poetry. They utilised the poem data set of Zhang and Lapata (2014) together with images collected from the Internet that depicted key words contained in the poems. In the encoder part, a CNN extracts visual features from the input images, while semantic features from previously generated lines of the poem are built by a bi-directional Gated Recurrent Unit (GRU). The decoder then consists of another GRU that generates a new line of the poem (Chinese) character by character, using the visual and semantic features together with the keywords.

3 Data set

Optimally, a data set should consist of data as close as possible to the desired results of the system, since the goal of training computer models is to mimic the data set as closely as possible. However, collecting a data set consisting of poetry is very inconsistent, meaning that every poet writes differently and uses different words and expressions. The result is that the available sample of reasonably consistent poetry is rather small. In comparison, hundreds of thousands of song lyrics — essentially rhythmic poems — are readily available. Hence, those in the field of poetry generation commonly use song lyrics rather than poetry as their data sets. Therefore, just like Ghazvininejad et al. (2016), song lyrics were chosen as a base for the data gathered for this project, since other data sets of the right size and content are not available due to various copyright protections. For the same reasons, the collected data set cannot be distributed further.

The data set was collected from `www.mldb.org`, an online song lyrics database. A Python script was written to connect to their site, sort through the HTML files of the site and find the song text, artist and album. Beautiful Soup[2] handles the HTML file by creating a parse tree, making it easy to navigate and handle the data provided in the HTML

[1] `www.speech.cs.cmu.edu/cgi-bin/cmudict/`

[2] `www.crummy.com/software/BeautifulSoup/`

file. A total of $206,150$ songs were collected, with a vocabulary of $391,363$ unique words and $46,346,930$ tokens in total. This dataset was filtered to remove songs that contained less than 95% of English words, as well as non-lyric content such as escape characters (e.g., newline, backslash and quote characters), custom messages by the sender (e.g., "Submitted by x" and "Thanks to y"), name of artists, verse and chorus notation, and notation of physical movements. After filtering, $80,608$ songs remained with a vocabulary of $91,097$ unique tokens. This is still too large and sparse a dataset for efficiently training a classifier, so all songs containing words that are not in the top $30,000$ vocabulary were also removed, leaving $40,685$ songs ($8,712,213$ tokens) with a final vocabulary size of $27,601$.

4 System Design and Implementation

The system consists of several stages. First, after an image is obtained it is run through Inception for classification. The output from Inception is the five top scoring classes. If these are scored lower than a set threshold, the result is discarded; otherwise they are used to find related words using ConceptNet by utilising ConceptNet's related words feature to return a scored list of related concepts, and then using the core of ConceptNet to find edges for the concepts, with the concept at the other end of the edge being saved and scored. When all related words have been found, rhyme words are explored, with the aim that both rhyming words should be related to the picture.

The next stage is to find the optimal path through a tree structure to construct a sentence. The path starts from the start of the sentence and ends on the rhyme word at the given line. Any number of lines can be generated. When a line is generated, an attempt to check the grammatical structure is made, but its use is limited. Figure 1 shows all steps from an input image to the generated poetry.

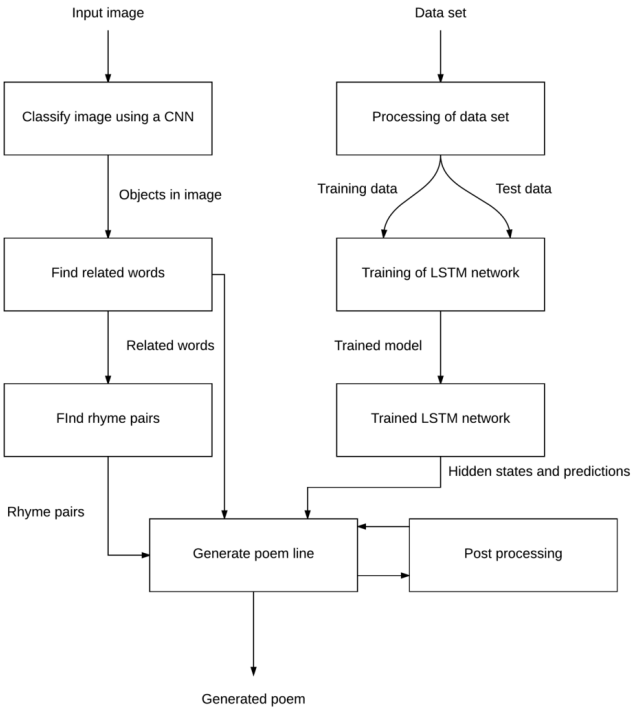


Figure 1: A high level overview of the system

Finding rhyming words from images

Looking at the components of the architecture in more detail, Inception-v3 (Szegedy et al. 2016) is the **object recognition** part, so the images used in the experiments must contain at least one object recognizable by Inception-v3, which is a 42 layer convolutional neural network available through TensorFlow in both Python and C++. Inception-v3 is pretrained on the 2012 ImageNet Large Visual Recognition Challenge (ILVRC), where the task is to classify a picture into one of 1,000 classes. The classes have no particular themes, however, 400 of them are animals, while the rest are quite diverse and can be anything from *CD player* to *castle*.

The keywords returned by the CNN are used to gather a large set of **related words**, based on the related words feature in ConceptNet (Speer and Havasi 2012), which returns a number of words along with a similarity score between 0 and 1. The next step looks at the edges of the concepts belonging to the keywords: each edge has another concept end point, and these are retrieved along with a score, which is between 1 and 12, but normalized to $[0, 1]$. The scores from related words and nearby concepts are treated equally. If the system has few high scoring related concepts, the system will prune the highest scores and retrieve concepts related to them. This repeats until the system has at least 200 related words. All related words need to be in the vocabulary. The more related words that are gathered, the more options the system has to chose from. At 200, or more, words the system seemingly has no problems finding applicable related words. The risk of finding no appropriate related words, or using poorly scored related words is that the poem will not be perceived as relevant to the image given as input.

When the related words are found, the system looks for **rhyme pairs**. This is done using CMUdict, which finds the pronunciation of a given word. For instance, the word *dictionary* is returned as *D IH1 K SH AH0 N EH2 R IY0*, where the numbers are the syllable stress and the letters the pronunciation. Two rhyming rules are used to find the rhyming words. The first checks if the endings of the words are pronounced the same. The length of the rhyme does not matter, though it must be at least one syllable long. The second rule states that the consonant sounds preceding the rhyme must be different. A consequence of this is that slant rhymes are not allowed in the system. Therefore, mostly perfect rhymes are present in the rhymes. This is a design choice, because slant rhymes such as *milk—talk* do not sound particularly good. However, if the system cannot find any suitable rhyming words, the second rhyming rule is discarded, and only the word endings are checked. Ideally, both rhyme words should be related to the initial image, but if there are no related words rhyming with each other, the system looks for other words rhyming on a related word in the vocabulary. The highest scoring related words and their rhyming words are preferred.

Predicting word sequences

The Long Short-Term Memory (LSTM) network then has the task of predicting the next word in a given sequence. It takes a word and the previous hidden state as an input, before producing an array of the scores for each other word in the vocabulary. The hidden state is then updated taking the whole sequence of previous words into consideration. The vocabulary is embedded into a dense Vector Space Model representation before being fed further into the network. The embedding matrix is initialized randomly, but as the training goes on, the model learns to differentiate between words by looking at the data set. To save memory when training, batches of the data set are converted into tensors that TensorFlow can use to train the model.

When training a network, several different parameters have to be set. Following Zaremba, Sutskever, and Vinyals (2014), the most important variables experimented with were: batch size, learning rate and learning rate decay, probability of keeping a neuron (dropout; Srivastava et al., 2014), number of steps unrolled in the LSTM layers, and parameter initialisation. The learning rate starts at 1.0, but after 14 epochs it is decreased by a factor of 1.15 after each epoch. The batch size is 20, and the parameters are uniformly initialized in the range $[-0.04, 0.04]$. The dropout rate is 65% for non-recurrent units. The number of steps unrolled in the LSTM layers is 35. The training time for the best performing network was 34 hours and 41 minutes on a NVIDIA GeForce GTX 970 with 4 GB memory.

Several different architectures of the LSTM network were tried, with the model with the lowest word perplexity score during training, and therefore best performing, having four layers: one input layer to represent the words coming in from the data set, followed by two hidden layers with the size of 1100 cells, and one softmax layer to give the predictions for the next words. The core of the network are the LSTM cells found in the two hidden layers that compute the possible values for the next predictions.

The loss function the network tries to minimize is the average negative log probability of the target word: $-\frac{1}{N}\sum_{i=1}^{N}\ln(p_{\text{target}_i})$ where N is the total training set size, *target* is the target word, and i is the word being looked at. The log loss is the cross entropy between the distribution of the true labels and the predictions given by the network. Therefore, by minimizing the log loss, the weights are optimized to produce a precise distribution of true labels. This is equivalent to maximizing the product of the predicted probability under the model. Per-word perplexity then measures how well the LSTM network performs during training:

$$e^{-\frac{1}{N}\sum_{i=1}^{N}\ln(p_{\text{target}_i})} = e^{loss} \tag{1}$$

A gradient descent optimizer is used along with backpropagation to minimize the loss function during training. The backpropagation algorithm uses the error values found by the loss function, *L*, to calculate the gradient of the loss function with respect to the weights, *w*, in the network, $\frac{\partial L}{\partial w}$.

To optimise the architecture with respect to word perplexity, the number of layers, hidden units, and epochs were varied. Zaremba, Sutskever, and Vinyals (2014) achieved a perplexity of 68.7 on the Penn Tree Bank data set, with a vocabulary of only $10,000$ words. Usually when training on larger vocabularies, the network should be more confused, so perplexity should increase. Here the vocabulary size is $27,601$, but the tested networks still achieved remarkably low perplexity: with only one hidden layer and 1100 hidden units, the LSTM's perplexity after 53 epochs was 65.0. With two hidden layers and 1100 units, perplexity dropped further, to 36.7 after 53 epochs. The reason for the low perplexity is the data set: song lyrics tend to use a small variety of words and often repeat lines. The network tries to mimic this behaviour. The result is that the predictions of rare words are often 0.0, with major implications for system performance.

Generating poetry

The poetry generation takes the related words, rhyme pairs and a trained LSTM network as input. It creates a tree structure to find the highest scoring paths through the tree, applies part-of-speech tagging to the paths, and returns the most optimal path that fits the grammar constraints. The stanzas are chosen to be 4 lines long, with each line being 8 syllables long, and the rhyming scheme being *AABB*. One stanza is generated per image. First, a tree is generated for each line in the stanza. One node symbolizes one word and its syllables, and each edge is the score between one node and another, as provided by the trained LSTM network. The root of the tree is an empty string with the initialized hidden state. Based on the root and the hidden state, the system takes the top 30 words predicted and sets them as the root's children. For these, the system generates 30 predictions. This process is done when the syllable constraints are fulfilled.

However, this tree is too large to be effective and the search takes too long. In the worst case scenario, when all words are monosyllabic, a tree finding the path to an eight syllable length phrase will consist of 30^8 nodes. Due to various calculations in TensorFlow, only about five nodes can be looked at per second, so searching through that many nodes is unacceptable. The solution to this is two-folded: First, a depth first search is performed while deleting all child nodes that have already been looked at. Second, the tree is pruned based on the score of a line or syllabic constraints.

The depth first search takes the syllable constraints into account. When the search reaches a leaf node, the system generates a score of how good the line is, which is the sum of each prediction for each word. The line can be positioned anywhere in the stanza, depending on the rhyming scheme chosen for the stanza. A tuple is generated that contains the rhyme words for two different lines, and scored based on how relevant the words are to the image. The tree search then goes back to look at the leaf node's parent, sets the current node to be looked at as one of the siblings of this node, and calculates the next predictions for this new node. This is done 30 times, once for each of the 30 top predictions.

Pruning of the tree is done in two ways: Based on the score, or based on different syllabic constraints. A node is skipped if does not fulfil the following constraint:

$$node_{\text{score}} > \frac{S_{\text{node}}}{S_{\text{tot}}} \times \frac{top\_score[-1]}{2} \tag{2}$$

where S_{node} is the syllable count for a given node, S_{tot} the total number of syllables needed to finish the line, and $top\_score[-1]$ the worst top score achieved at this point.

The score of a given line is calculated in two parts, with one coming from the words generated in the sentence and the other being the prediction of the rhyming word. These values are continuously normalised against other values in the top score list, so that the line score and the rhyme word prediction are evaluated as equal parts — otherwise, the line score would dominate, since it is the sum of multiple predictions, while the rhyme word prediction is only one.

The system tends to repeat words, since the network is trained on song lyrics that often are repetitive. Therefore words already used in a line receive lower scores, and are avoided if possible. Furthermore, the system prefers to use easy and safe words. Because of this, a higher score is given to the related words. This forces the system to explore new words and hence enhances performance. However, completely avoiding repetition of words also has a disadvantage, since repetition is an important poetic feature.

A problem the system cannot handle very well is the transition between the generated line and the rhyme word. POS-tagging is added to address this problem. The tagger first tags the line generated, and then the rhyme word separately. However, tagging a word without any context is not possible, so the 1M word POS-tagged Brown corpus is used to determine the most frequent tag of the rhyme word. By fixing the rhyme word at the end of the line, and POS-tagging again, this time the line and the rhyme word together, the rhyme word gets another POS-tag. If this tag matches the most frequent one in the Brown corpus, the line is accepted. Clearly, it is not always enough to check that the POS-tag matches the most frequent tag; however, it guides the system to generate grammatically better poems.

5 Experiments and Results

Two different experiments were conducted to test the system. In the first, human subjects selected any image of their choosing, preferably one they had taken themselves. This image was then run through the system, and the subject evaluated the generated poems according to three criteria described below. To avoid testing the system on images and to ensure using the best generated poetry, the participants were first asked to find at least three images of their choosing. Each image was then run through the system and the stanzas retrieved, which the participants were asked to rate.

In the second experiment, the subjects were tested to see if they could differentiate between poetry generated by the system and poetry written by a human. Here, a participant was shown an image and the corresponding human generated and machine generated stanzas, and asked which stanza was the human generated one.

Finally, the participants were asked about their overall thoughts of the system. 46 persons participated in the experiments, with no requirement that they should have any prior experience of writing or evaluating poetry. A total of 153 stanzas were rated on the three criteria, and a total of 38 evaluations were done to decide if the poetry was from a human or a computer.

The sun is in my big raincoats
I don't know what to do scapegoats
I'm raining and it looks like rain
There's so much for me to abstain

Score: [3.0, 2.7, 2.0]

Figure 2: "Red Umbrella" by DLG Images (CC BY 2.0)
www.flickr.com/photos/131260238C%40N08/16722739971/.

I don't know why it feels like crabs
That make me want to look at cabs
So come on lets get out of zoo
And dive into my big canoe

Score: [3.0, 2.5, 1.0]

Figure 3: "Jellyfish" by Jennifer C (CC BY 2.0)
www.flickr.com/photos/29638108%40N06/34440131755/.

I want to be with your tent group
In the shape of your hand and troop
My life is an operation
So come on lets go now station

Score: [2.5, 3.0, 2.5]

Figure 4: "KFOR 12 training", The U.S. Army (CC BY 2.0)
www.flickr.com/photos/soldiersmediacenter/3953834518/.

Grammaticality, poeticness and meaningfulness

State-of-art poetry generation commonly use either automatic or human-based evaluation methods. The automatic approaches, such as BLEU (Papineni et al. 2002), originate in machine translation research and assume that human written references are available, which is not the case for the present work. Another problem with these evaluation methods is that they have been found to have little coherence with human evaluation (Liu et al. 2016).

Hence we will here focus on human-based evaluation, in particular along the lines of Manurung's (2004) criteria *grammaticality* (lexical and syntactic coherence; in essence ruling out random sequences of words), *meaningfulness* (convey a conceptual message which is meaningful under some interpretation; here it will also include topic coherence with the visual input), and *poeticness* (phonetic features such as rhymes and rhythmic patterns). Each of the dimensions is scored on a scale of 1–3. A slightly different version of these criteria has also been proposed: *fluency*, *meaning*, *poeticness* and *coherence* (Yan et al. 2013; He, Zhou, and Jiang 2012). However, no in-depth explanation of these criteria has been made and different definitions of the four metrics have been given, so this work will use the metrics of Manurung (2004) and the goal of the first experiment was to evaluate each poem on those criteria. The average score and standard deviation of all poems evaluated were 2.614 ± 0.519 for poeticness, 2.381 ± 0.470 for grammaticality, and 2.050 ± 0.712 for meaningfulness, with median scores of 2.8, 2.5, and 2.0, respectively.

Hole in the back of my bottle
I don't know what it's like throttle
It makes me feel this way grape wine
And when I'm with you flask align

Score: [2.8, 2.5, 2.5]

Figure 5: "NAPA 2012" by cdorobek (CC BY 2.0)
www.flickr.com/photos/cdorobek/8093546797/.

I don't know what to do cute bear
The way you look at me when fair
And I'm so in love with teddy
You're just a part of already

Score: [2.8, 2.7, 3.0]

Figure 6: Picture by Yumeng Sun, used with permission

Five randomly selected poems are shown in Figures 2–6 to display a variety of different scored poems. The input image is to the left while the generated poem related to the image is on the right, followed by the score for the poem, in the order of poeticness, grammaticality, meaningfulness.[3] As can be seen, poeticness scores higher than grammaticality and meaningfulness. The reason for this is the guarantee of the lines being eight syllables long, and almost every line rhymes. When the system cannot find suitable rhyme words, sub-optimal rhyme words are chosen.The grammaticality score is a bit lower, and the meaningfulness is slightly above the "partially meaningful" level. Both of these are influenced by the system predicting line ending rhyme words with a 0 probability. This happens when the LSTM network dislikes all the rhyming words found, scoring them all as 0.

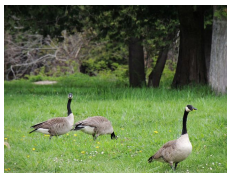
It is also interesting to compare the results of poems with two or more lines where the rhyme word is predicted by a non-zero number against poems containing zero or one such lines, since when the rhyme word is predicted by a zero value, the POS-tagging and the word relevance decide the rhyme word and the corresponding line. There can be two reasons for a zero score: that the network has been trained on the word but does not consider it a good fit, or that the training data contained too few occurrences of the rhyming word so that the network is uncertain about its fitness. 33 of the 153 stanzas contain two or more lines where the rhyme word was predicted by a non-zero value. Looking at the scores for those only, it is clear that when it is known that the system has an opinion of the rhyming word, performance increases (note that the opinion not necessarily has to be good). The average poeticness goes from 2.614 to 2.793, grammaticality increases from 2.381 to 2.691, and poeticness from 2.050 to 2.464. However, the Pearson Correlation Coefficient reveals that even though there technically is a positive correlation between the number of non-zero predictions for the rhyme word and survey scores, the relationship

[3]Unless stated otherwise, images are used under Creative Commons Attribution 2.0 license (CC BY 2.0):
creativecommons.org/licenses/by/2.0/legalcode

It's been a long time and I've bees
But I don't want to get disease
I'm all out of love, oh so sweet
Just give me one more chance to meet

I am flying in the warm air
The other bees are very fair
It is a beautiful sunday
I am leaving later today

Figure 7: "bee" by David Elliott (CC BY 2.0)
www.flickr.com/photos/drelliott0net/15105557167/



I want to see swans in the sky
You and me, we are in for fly
If there's a lake out there for goose
There is no limit for excuse

Eating dinner like a wild goose
Food is something that I can't lose
Although cooking is really tough
Nothing better can make me bluff

Figure 8: "Canada Geese" by Kevin M. Klerks (CC BY 2.0)
www.flickr.com/photos/ledicarus/34618458382/.

is weak: the correlation score for poeticness was 0.1658, 0.3040 for grammaticality, and 0.2674 for meaningfulness.

Human- and machine generated poetry experiment

The second experiment was done to see if the participants could differentiate between the machine generated poetry and human written poetry. The process was as follows: After 30 stanzas had been evaluated, the four highest rated poems were selected. The persons who had chosen those images were asked to write a poem with the same syllabic and rhyming constraints, i.e., each sentence had to have eight syllables, and the two first and two last lines should rhyme.

For all these four images, the computer generated stanza and human evaluated stanza are shown in Figures 7–10. The top stanza next to each image is the computer generated one, and the stanza in italics is the human written. The question asked to the participants of the experiment was: "Which stanza is human generated?" 38 persons evaluated the four items, giving a total of 152 evaluations. The participants chose the correct option in total 117 times, while the wrong option was chosen 35 times, so the participants had a 76.5% success ratio. Figure 8 was the easiest for the participants to single out, with only 5 of 38 participants answering wrongly. For Figures 7, 9, and 10, respectively 10, 11 and 9 participants erroneously picked the machine generated stanza.

The last part of the experiment was to get the participant sentiments. The question asked to the participants was: "After seeing the results of the images that you chose, and the poetry of the four images of experiment two, do you have any final thoughts?" 29 of the 46 participants used the word "fun" when evaluating the poetry, while 20 participants used the word "random", and a few used the word "bad". Other notions such as "interesting system" and "the poems feel alive" were also common. Seven participants mentioned "personification" as a *core value* of the generation system.

Broomstick on the back of my broom
Move on like dead man can assume
I don't know what to do and sweep
It makes me think that I'm asleep

Checking the status of my broom
I need the broom to sweep the room
I have no choice to keep waiting
It's hard for me to stop hating

Figure 9: "broom" by danjo paluska (CC BY 2.0)
www.flickr.com/photos/sixmilliondollardan/3074916976/.

It's been a long time since the kick
It makes me want to hold you stick
And I know communication
I've got so much combination

Tapping on my keyboard at night
Hoping it will not cause a fight
You are my only listener
Don't let me be your prisoner

Figure 10: Picture by Yumeng Sun, used with permission

6 Discussion

The average score of 2.614 in the 'poeticness' category tells that people overall found the stanzas in between *partially poetic* and *poetic* in terms of rhythm and rhyming. However, the standard deviation is quite high (0.519), either representing a big spread of the sentiment of the participants, or a big spread in the quality of the poems generated. The average score of 2.381 for 'grammaticality' indicates that the language is closer to being *partially grammatically correct* than *grammatically correct*. The standard deviation of 0.470 indicates that the spread is smaller than for poeticness, but it is still quite large. The average score of 2.050 for the 'meaningfulness' category tells that the system is on average evaluated as *partially meaningful*, although the standard deviation of 0.712 indicates that meaningfulness is the most inconsistent property of the poems generated by the system.

Several effects of using song lyrics as training data for the LSTM network are reflected in the poetry, including: extensive use of the first person pronoun (*I*) as well as of the word *love* and terms related to it, repetition of some phrases (e.g., "I don't know", "I want to", "It's been a long time since", "So come on"), and that a large part of the vocabulary is predicted with zero probability at any given point in time. As a result, and as can be seen in the examples above, a large fraction of the generated stanzas include at least one form of *I*: 87% had at least one first person pronoun, 42% had two, and 12% of the poems had three. The extensive use of pronouns can lead to personification (non-human objects taking on human characteristics), with the poems often gaining human traits even though there are no humans in the image. Furthermore, *love* appears $10,671$ times in the dataset and is found in 7% of the songs, leading to it appearing in 11% of the poems generated, which makes the poetry love based.

The system has a couple of limitations. The first one is Inception: if it misclassifies the input image, the system will perform poorly, because all related words and rhyme words will become related to the wrong class. Another limitation is that only the top-1 class is used to find related words. The reason to only use the top-1 result is that other results are often not related and might be wrong, therefore hurting the performance of the system. This means that only one object in the image is used. Using object recognition instead of classification might yield better results, since this would make it possible to identify multiple objects in the image. This could make the generated poems more dynamic as the poem can choose from a broader set of related words, and mention multiple objects in the image.

7 Conclusion and future work

A system was designed and implemented to take an image as input and generate poetry related to the image. The system includes a CNN for object classification, a module to find related words and rhyme pairs, and an LSTM network to score a tree search where nodes are representing the words being generated. 153 stanzas were generated and experiments were conducted with volunteering participants to evaluate the quality. The results of the system were varied, with a big standard deviation on the three criteria it was evaluated on. The system was not able to consistently generate stanzas that are perceived by everyone to be aesthetically good. The best results were achieved when the LSTM network could predict the rhyme word with non-zero prediction scores, however, only a weak correlation was found between the evaluation and the stanzas containing non-zero predictions.

A data set of more than $200,000$ songs was gathered and pre-processed to train the LSTM model. Various difficulties arose when gathering the data, the biggest being the variety of random content in the song lyrics. Upon closer inspection of the data set some content that is not song lyrics are still present, however, the fraction of this content does not have any noticeable impact on the system. Using song lyrics is interesting due to elements of personification emerging and the grammar in a line is usually good until the rhyme word appears, while the system has trouble predicting rhyme words. Overall the system will probably never write any poetic masterpiece, but the evaluations made by the participants indicate that some generated stanzas were subjectively enjoyable. This suggests that the implementation could be useful as a foundation for other poetry generation systems.

One option for trying to enhance the performance of the predictions is changing the LSTM network to a Sequence to Sequence Model (Cho et al. 2014). Several newer poetry generation systems such as Wang et al. (2016) use this approach, and report good results. Another possible change to the system is to train a separate model for fetching related words. One popular model for doing this is to train a word2vec model. Implementing this into the system could enhance performance by finding better related words and better rhyming words. Training the LSTM network on poetry instead of song lyrics is also an interesting variation to test. Poetry has different properties than song lyrics, such as using a bigger variety of words more often.

References

Cho, K.; Van Merriënboer, B.; Gulcehre, C.; Bahdanau, D.; Bougares, F.; Schwenk, H.; and Bengio, Y. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

Colton, S.; Goodwin, J.; and Veale, T. 2012. Full-FACE poetry generation. In *Proceedings of the 3rd International Conference on Computational Creativity*, 95–102.

Díaz-Agudo, B.; Gervás, P.; and González-Calero, P. A. 2002. Poetry generation in COLIBRI. In *European Conference on Case-Based Reasoning*, 73–87. Springer.

Gervás, P. 2000. WASP: Evaluation of different strategies for the automatic generation of Spanish verse. In *Proceedings of the AISB-00 symposium on creative & cultural aspects of AI*, 93–100.

Gervás, P. 2001. An expert system for the composition of formal Spanish poetry. *Knowledge-Based Systems* 14(3):181–188.

Gervás, P. 2002. Exploring quantitative evaluations of the creativity of automatic poets. In *Workshop on Creative Systems, Approaches to Creativity in Artificial Intelligence and Cognitive Science*.

Ghazvininejad, M.; Shi, X.; Choi, Y.; and Knight, K. 2016. Generating topical poetry. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, 1183–1191.

He, J.; Zhou, M.; and Jiang, L. 2012. Generating Chinese classical poems with statistical machine translation models. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence*, 1650–1656.

Lamb, C.; Brown, D. G.; and Clarke, C. L. 2017. A taxonomy of generative poetry techniques. *Journal of Mathematics and the Arts* 11(3):159–179.

Levy, R. P. 2001. A computational model of poetic creativity with neural network as measure of adaptive fitness. In *Proceedings of the ICCBR-01 Workshop on Creative Systems*.

Liu, C.-W.; Lowe, R.; Serban, I. V.; Noseworthy, M.; Charlin, L.; and Pineau, J. 2016. How NOT to evaluate your dialogue system: An empirical study of unsupervised evaluation metrics for dialogue response generation. *arXiv preprint arXiv:1603.08023*.

Manurung, H. 2004. *An evolutionary algorithm approach to poetry generation*. Ph.D. Dissertation, School of Informatics, University of Edinburgh.

Mihalcea, R., and Tarau, P. 2004. TextRank: Bringing order into texts. In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*. ACL.

Mikolov, T.; Chen, K.; Corrado, G.; and Dean, J. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Netzer, Y.; Gabay, D.; Goldberg, Y.; and Elhadad, M. 2009. Gaiku: Generating haiku with word associations norms. In *Proceedings of the Workshop on Computational Approaches to Linguistic Creativity*, 32–39. ACL.

Oliveira, H. G.; Cardoso, F. A.; and Pereira, F. C. 2007. Exploring different strategies for the automatic generation of song lyrics with tra-la-lyrics. In *Proceedings of 13th Portuguese Conference on Artificial Intelligence, EPIA*, 57–68.

Oliveira, H. G. 2012. PoeTryMe: a versatile platform for poetry generation. *Computational Creativity, Concept Invention, and General Intelligence* 1:21.

Oliveira, H. G. 2017. A survey on intelligent poetry generation: Languages, features, techniques, reutilisation and evaluation. In *Proceedings of the 10th International Conference on Natural Language Generation*, 11–20. Santiago de Compostela, Spain: ACL.

Papineni, K.; Roukos, S.; Ward, T.; and Zhu, W.-J. 2002. BLEU: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, 311–318.

Speer, R., and Havasi, C. 2012. Representing general relational knowledge in ConceptNet 5. *Proceedings of LREC*.

Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; and Salakhutdinov, R. 2014. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research 15* 1929–1958.

Szegedy, C.; Vanhoucke, V.; Ioffe, S.; Shlens, J.; and Wojna, Z. 2016. Rethinking the Inception architecture for computer vision. In *The IEEE Conference on Computer Vision and Pattern Recognition*, 2818–2826.

Toivanen, J.; Toivonen, H.; Valitutti, A.; and Gross, O. 2012. Corpus-based generation of content and form in poetry. In *Proceedings of the 3rd International Conference on Computational Creativity*, 175–179.

Toivonen, H.; Gross, O.; Toivanen, J. M.; and Valitutti, A. 2013. On creative uses of word associations. In *Synergies of Soft Computing and Statistics for Intelligent Data Analysis*. Springer. 17–24.

Wang, Z.; He, W.; Wu, H.; Wu, H.; Li, W.; Wang, H.; and Chen, E. 2016. Chinese poetry generation with planning based neural network. *arXiv preprint arXiv:1610.09889*.

Wang, Q.; Luo, T.; and Wang, D. 2016. Can machine generate traditional Chinese poetry? A Feigenbaum test. In *Advances in Brain Inspired Cognitive Systems: 8th International Conference*, 34–46. Beijing, China: Springer.

Xu, L.; Jiang, L.; Qin, C.; Wang, Z.; and Du, D. 2018. How images inspire poems: Generating classical Chinese poetry from images with memory networks. In *Proceedings of the 32nd AAAI Conference on Artifical Intelligence*, (in press).

Yan, R.; Jiang, H.; Lapata, M.; Lin, S.-D.; Lv, X.; and Li, X. 2013. i, Poet: Automatic Chinese poetry composition through a generative summarization framework under constrained optimization. In *Proc. of the 23rd International Joint Conference on Artificial Intelligence*, 2197–2203.

Zaremba, W.; Sutskever, I.; and Vinyals, O. 2014. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*.

Zhang, X., and Lapata, M. 2014. Chinese poetry generation with recurrent neural networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, 670–680.