Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Kasper Aalberg Røstvold

Once More, with Feeling

Computer Generated Poetry with Inherent Sentiment

Master's thesis in Computer Science
Supervisor: Björn Gambäck
July 2019

NTNU
Norwegian University of
Science and Technology

Kasper Aalberg Røstvold

Once More, with Feeling

Computer Generated Poetry with Inherent Sentiment

Master's thesis in Computer Science
Supervisor: Björn Gambäck
July 2019

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Abstract

This master's thesis presents the implementation and evaluation of a system able to generate poetry with an inherent sentiment value. Poetry generation is one of the more interesting challenges within the fields of Natural Language Processing and Computational Creativity, as it is dependent on both the form of the content, and the creation of understandable, meaningful and poetic language. A central part of poetry is the experience of the reader — including the emotions poetry can evoke. The motivation behind this project is to investigate if poetry can be generated to contain a specific sentiment, and whether readers of the poetry would experience the sentiment that was intended.

Most state-of-the-art solutions within the field of Computer Generated Poetry have utilized Artificial Neural Networks. The system created in this project has implemented a Long Short-Term Memory Neural Network as the main component, trained on a data set consisting of English poetry written and published by users on a public website. Additional modules of the system consist of rhyme pair generation, rule-based word prediction methods, and a search algorithm for extending generation possibilities.

Several experiments were conducted in this project, involving training of the neural network and evaluation of the generated poetry. A selection of human judges participated in evaluating 20 poems generated by the system, 10 of them generated with a positive sentiment and the other 10 with a negative sentiment. The judges evaluated poems based on a set of standard evaluation metrics, in addition to evaluating the sentiment. The results of these experiments indicate that while there are some weaknesses in different aspects of the system compared to other state-of-the-art solutions, it is fully capable of generating poetry with an inherent sentiment that is perceived by readers.

Sammendrag

Denne masteroppgaven presenterer implementasjonen og evalueringen av et system som kan generere poesi med et forhåndsbestemt sentiment. Poesigenerering er en av de mer interessante utfordringenge innenfor feltene Naturlig Språkprosessering og Maskinskapt Kreativitet (Computational Creativity), siden det er avhengig av både formen på innholdet, men også evnen til å produsere forståelig, meningsfullt og poetisk språk. En sentral del av poesi er opplevelsen til leseren, som blant annet involverer følelsene poesien kan fremkalle. Motivasjonen bak dette prosjektet er å undersøke om poesi kan genereres for å inneholde et bestemt sentiment, og om lesere av diktet vil oppfatte det sentimentet som var tiltenkt.

De fleste av dagens moderne løsninger innenfor feltet Maskinskapt poesi har benyttet nevrale nettverk. Systemet som ble utviklet i dette prosjeket implementerte et nevralt nettverk av typen Long Short-Term Memory som den sentrale komponenten, hvor nettverket er trent på et datasett bestående av engelsk poesi som er skrevet og publisert av brukere på en offentlig nettside. Andre moduler inkludert i det ferdige systemet består av en genereringsmetode for rimpar, regelbaserte ordprediksjons-metoder, og en søkealgoritme for å utvide genereringsmulighetene.

Flere eksperimenter ble gjennomført i dette prosjektet, som involverer trening av det nevrale nettverket, og evaluaring av den genererte poesien. Et utvalg av lesere evaluerte 20 dikt generert av det implementerte systemet, hvor halvparten var generert med et positivt sentiment, og andre halvparten med negativt. Leserne evaluerte diktene basert på et utvalg av standard evalueringsfaktorer, i tillegg til å evaluerere sentiment. Resultatene fra eksperimentene indikerte at selv om det er noen svakheter i et par aspekter ved systemet sammenlignet med andre moderne løsninger, så er det helt i stand til å generere poesi med et tiltenkt sentiment som blir oppfattet av lesere.

Preface

This master's thesis is submitted to the Norwegian University of Science and Technology (NTNU) as a part of the requirements for the degree of Master of Science in Computer Science. The thesis work has been performed at the Department of Computer Science at NTNU, Trondheim. The supervisor of this thesis was Professor Björn Gambäck.

I would like to thank my supervisor Björn Gambäck for his guidance and feedback throughout the entire process of writing this master's thesis, and the fellow students, in addition to my father, who participated in discussions and provided insight and helpful feedback regarding the work of this project. I would also like to thank the people who participated in the experiments that were conducted, and Tikhonov and Yamshchikov for providing the data set that was used in this project.

Kasper Aalberg Røstvold
Trondheim, 15th July 2019

Contents

List of Figures

List of Tables

1 Introduction

In this master project a system was designed and implemented for generating poetry with a given inherent sentiment. A literature study on the field of computer generated poetry and the state-of-the-art solutions was conducted, and used as motivation for the implementation of the final generation system. The system consists of a bi-directional Long Short-Term Memory model, combined with rhyme pair generation, rule-based word prediction methods, and a tree search algorithm. The Long Short-Term Memory network was trained on a data set consisting of publicly published poetry.

Several experiments and the results are presented in this thesis, conducted both on training of the neural network model and on poetry generated by the system. In total 20 poems were generated, and a selection of human judges participated in the poetry experiments, evaluating the poetry on several factors, including the perceived sentiment.

This chapter first presents the background and motivation for this master's thesis, followed by the goal it tries to achieve and research questions it tries to answer. Afterwards, it presents the research methods used, the main contributions, and the overall structure of the thesis.

1.1 Background and Motivation

Computational creativity is a sub-field of artificial intelligence with elements from cognitive science, philosophy, linguistics and arts, where the goal is to use computers to simulate a creative process and/or create artistic results. Computational creativity can be applied in many areas such as composition and lyrics for music and different forms of visual art like paintings, Computational creativity also has promising use in other applications such as mathematics and games like chess. One of the main application areas of computational creativity is linguistics, where the focus is on creativity related to the use of language. Examples of linguistic creativity could be poetry, language retrieval, word associations, jokes, analogies or story narratives.

In addition to the goal of simulating artistic processes and producing actual artistic results, computational creativity also enables us to better understand human creativity and to design programs that can enhance and support human creativity.

One of the main challenges of theoretical computational creativity is how to define what creativity is, both as a process and what defines the result of such a process. Because of the lack of a single definition offering a complete picture of creativity, AI researchers Newell, Shaw and Simon (Newell et al., 1959) developed a combination of novelty and usefulness to a multi-pronged view of creativity, one that uses four criteria to categorize a given answer or solution as creative:

- The product of the thinking has novelty and value (either for the thinker or for his culture).
- The thinking is unconventional, in the sense that it requires modification or rejection of previously-accepted ideas.
- The thinking requires high motivation and persistence: either taking place over a considerable span of time (continuously or intermittently), or occurring at high intensity.
- The problem as initially posed was vague and ill-defined, so that part of the task was to formulate the problem itself.

This project looks at poetry generation, as it is one of the more interesting challenges within computational creativity. Poetry generation is a type of linguistic creativity that requires certain qualities in both form and content as well as the creation of understandable, meaningful and poetic language. A central part of poetry is the experience of the reader — which involves the emotions poetry can evoke. The motivation behind this project is to investigate if a system can be implemented to generate poetry with a specific sentiment, and whether a human reader would experience the intended sentiment.

1.2 Goals and Research Questions

Goal *The overarching goal of this thesis is to contribute to the field of poetry generation by exploring and developing methods for generating poetry with a specific inherent sentiment experienced by readers of the poetry.*

Research question 1 *What are the possible solutions and implementations for generating poetry with an inherent sentiment?*

In order to develop a poetry generation system related to the goal of this thesis, it is necessary to investigate state-of-the-art solutions and methods within poetry generation in general, and the methods and solutions for generating poetry that contains a sentiment value.

Research question 2 *What Neural Network architecture can achieve best results for generating poetry?*

The Long Short-Term Memory (LSTM) network is chosen as the neural network to implement and experiment on in this thesis work. The reason for this is explained fully in chapter 5, with the main point being that neural networks, particularly the LSTM, are the most used solutions in state-of-the-art solutions for poetry generation. This research question therefore focuses on which specific LSTM architecture and what specific parameters of the network will produce the best performing network for word prediction in a poetry generation system (based on a relevant data set).

Research question 3 *Will the generated poetry be perceived to contain a sentiment value, and does this value correspond to the value the poetry was intended to have?*

The inherent sentiment of the poetry to be generated will be decided in advance, with the goal of producing poetry that contains such sentiment. The question to be answered is whether the inherent sentiment will be perceived by human judges, so that the generated poetry can be said to contain the intended sentiment, as stated in the Goal.

1.3 Research Method

The research method used in this thesis work includes experiments conducted in two areas. The first includes conducting experiments on the prediction abilities of trained Long Short-Term Memory network models with differing architectures and parameter values. The prediction abilities are measured by experimenting with all the different models on an experiment data set. The second area of experiments involves the evaluation of the poetry that will be generated by the system implemented. The evaluation is be done by a group of human judges, who evaluate poetry using different metrics such as standard evaluation metrics for computer generated poetry, but also metrics for the perceived sentiment value of the poetry.

Experiments on the neural network model was conducted to find the best solution for implementing a poetry generation system. The second area of experiments, consisting of evaluation of the poetry, was conducted to get a sense of the quality of poetry and the performance of the generation system, and also to be able to compare the work done in this thesis with similar work conducted in this field.

1.4 Contributions

The main contributions from this Master's thesis are:

1. A literary study of the development and state-of-the-art solutions for poetry generation, including solutions for emotion- or sentiment based poetry generation.
2. Design and implementation of a state-of-the-art system able to generate poetry with an inherent sentiment value.

1.5 Thesis Structure

In Chapter 2 the theory and definition of poetry is presented, followed by the background theory for the artificial intelligence methods needed to understand this thesis, natural language processing, and the methods used for evaluation generated poetry. In addition, it presents the different frameworks and resources used in this thesis.

Chapter 3 contains a literary study on the development and different approaches that have been used for poetry generation, followed by the state-of-the-art solutions, both for

poetry generation in general, but also for generating poetry that is emotion or sentiment based.

Chapter 4 presents the original data set that was used in this thesis, and the pre-processing that was done on the data.

Chapter 5 presents and explains the architecture behind the poetry generation system that was implemented in this thesis, first covering the LSTM model that was implemented and used, and secondly covering the complete poetry generation system.

Chapter 6 contains the plan, setup and results of all the experiments that were conducted. The experiments are involved in two different areas, the first part concerning experiments on training neural network models, the second part concerning human judges evaluating the poetry that was generated.

Chapter 7 contains an evaluation of the experiment results, the complete poetry generation system, and the poetry that was generated for this thesis, including limitations and faults, and comparisons to other solutions. Afterwards a discussion is presented, concerning the goals and research questions in this thesis regarding the final results.

Chapter 8 draws a conclusion of the work and the results, in addition to the possible future work being presented.

2 Background Theory

This chapter covers the theory and background behind the different areas relevant to this project. A general background of computational creativity and theory of poetry is presented, followed by different artificial intelligence algorithms and models that are presented and discussed in later chapters, including the models implemented in the poetry generation system. The later sections in this chapter cover natural language processing and sentiment analysis, as well as present different frameworks and resources used in this project. The final section consists of theory and approaches to evaluating poetry.

2.1 Poetry

Two different definitions of poetry — defined by Merriam-Webster and the Oxford English Dictionary respectively — are given as:

- "Writing that formulates a concentrated imaginative awareness of experience in language chosen and arranged to create a specific emotional response through meaning, sound, and rhythm." (Merriam-Webster, 2018)
- "Composition in verse or some comparable patterned arrangement of language in which the expression of feelings and ideas is given intensity by the use of distinctive style and rhythm; the art of such a composition." (Oxford English Dictionary, 2018)

Though it is difficult to clearly and distinctly define what constitutes poetry, certain elements of poetry can be specifically defined. Therefore the theory of form and sound patterns in poetry is presented.

Concerning the structure of poetry, a central definition is the use of *Lines*. A *Line* refers to each line of text in the poem, which are seperated by the start of a new line. Several lines grouped together are called *Stanzas*, which are separated by an empty line from other stanzas. A stanza can also be referred to as a verse, as most dictionaries still define these concepts as synonyms. Different types of stanzas can be identified, and named, based on the number of lines in the stanza. A couplet is a stanza of two lines, a tercet of three, etc.

The different sound patterns of poetry are *Rhyme*, *Rhythm*, and *Meter*. A rhyme is the repetition of similar sounds, often occurring at the end of each line. The pattern of the rhymes in the poem can be defined by using letters of the alphabet to describe the patterns: a rhyme pattern of **abab** refers to the first and the third line of the poem rhyming, and the second and fourth rhyming. Rhythm is the pattern of stressed and

unstressed syllables in a line of poetry. Meter is a pattern of rhythm, in other words premeasured patterns of stressed and unstressed syllables. *Feet* are individual rhythmic units, they are the building blocks of a meter.

Different forms of poetry exist, based on either a specific structure or the content of the poem. Examples of these forms are *Sonnets*, which are lyrical poems consisting of 14 lines, usually written in a meter called iambic pentameter, that refers to the specific meter of the foot *iamb* (which is two syllables — an unstressed followed by a stressed), repeated five times (*penta*). Other examples are *Haikus*, which are non-rhyming poems having three lines, with 5-7-5 syllables in the lines, respectively, and *Limerick*, which consists of five lines, with an **aabba** rhyming pattern.

2.2 Artificial Intelligence Methods

This section covers the background of relevant Artificial Intelligence methods, namely those that are presented in related work (chapter 3), and the methods that were used for this project (chapter 5). Firstly deep learning is described, and some deep learning methods are presented. These methods include the Artificial Neural Network (ANN), Recurrent Neural Network (RNN), and Long Short-Term Memory (LSTM). These are all methods that have been used with success in recent poetry generation. RNNs, especially the LSTM, produce good results for generating text based on previously processed text, and is the model used in the poetry generation system for this project. The method of Machine Learning called Reinforcement Learning will be covered in this section as well, since it has also been implemented in state-of-the-art poetry generation.

2.2.1 Reinforcement Learning

Reinforcement learning is an area of machine learning, which involves learning what to do — how to map situations to actions — so as to maximize a numerical reward signal (Sutton and Barto, 2015). Reinforcement learning is implemented in *closed-loop* problems, because the learning system's actions influence the situation, and therefore its later inputs as well. In addition, the learner is not told which actions to take, as in many other forms of machine learning, but instead must discover which actions yield the most reward by trying them out. This type of learning is different from supervised learning, where the latter is built around training on labeled examples provided by a knowledgeable external supervisor. Reinforcement learners try to solve interactive problems, where the agent must be able to learn from its own experience. It also differs from unsupervised learning, as the system is trying to maximize a reward signal instead of trying to find a hidden structure. Therefore reinforcement learning is considered a third machine learning paradigm, alongside supervised and unsupervised learning (Sutton and Barto, 2015).

A reinforcement learning system consists of four main elements: a *policy*, a *reward signal*, a *value function*, and optionally a *model* of the environment. The *policy* defines the learning agent's way of behaving at a given time. In general, it is a mapping from perceived states of the environment to actions to be taken when in those states. The

policy can be a simple function or lookup table, or it may involve extensive computation such as a search process. In general, policies may be stochastic.

The *reward signal* defines the goal in a reinforcement learning problem. On each time step, the environment sends to the reinforcement learning agent a single number which is the *reward*. The agent's objective is to maximize the total amount of reward it receives over the entire process. The reward signal, therefore, defines what the good and bad events are for the agent. The only way the agent can influence the reward signal is through its actions, either directly or indirectly (by changing the environment). In general, reward signals may be stochastic functions of the state of the environment and the actions taken.

The *value function* specifies what is good in the long run, as opposed to the reward signal which indicates what is good in an immediate sense. The *value* of a state is the total amount of reward an agent can expect to accumulate over the future, starting from that state. Values indicate the long-term desirability of states after taking into account the states that are likely to follow, and the rewards available in those states. One example is an action that grants a low reward initially, but leads to possible actions that can grant high rewards later. A value state can be said to estimate "how good" it is to be in a specific state.

The *model* of the environment mimics the behavior of the environment, by allowing inferences to be made about how the environment will behave. For example, given a state and action, the model could predict the resultant next state and the next reward. Models are used for *planning* — a way of deciding on an action by considering possible future situations before they are actually experienced.

Action selection rules define which actions the algorithm should select. The simplest action selection rule is to select one of the possible actions at a given time, which has the highest estimated *action value*. In other words, choosing the action that is estimated to reward the highest value. This is referred to as a *greedy* action selection method, and can be written as in Equation 2.1 where A_t is the greedy action at time t, and $Q_t(a)$ is the value of action a at time t. $argmax$ equals the value at which the function $Q_t(a)$ is maximized.

$$A_t = argmax_a Q_t(a) \tag{2.1}$$

Two important aspects of reinforcement learning are *exploration* and *exploitation*. The former refers to the algorithm's ability to explore lower rewarding actions, or actions of unknown consequences, to be able to explore the different possibilities of actions in the environment, and what their results will be. The latter refers to the algorithm's ability to consistently select actions with the highest calculated value, to optimize the performance. There is a distinct trade-off between these two qualities, as an algorithm with an emphasis on exploration will consequently be lacking in optimizing every action. Vice versa, emphasis on exploitation will reduce the exploration quality. Different action selection rules and value functions can be chosen with respect to which quality one wants the system to emphasize, exploration or exploitation.

2.2.2 Deep Learning

Deep learning refers to a wide class of machine learning techniques and architectures, with the broad aspect of using many layers of non-linear information processing that are hierarchical in nature. Deep learning can be categorized into three major classes of machine learning (Deng and Yu, 2014), namely unsupervised or generative learning, supervised learning, or a hybrid of the previously mentioned, often referred to as semi-supervised. The principle behind deep learning is learning data representations, as opposed to task-specific algorithms.

2.2.3 Artifical Neural Nets

Artificial Neural Networks (ANN) are frameworks for different machine learning algorithms inspired by biological neural networks (Goodfellow et al., 2016). The main component in the ANN is the neuron (also referred to as the perceptron), which is implemented to mimic the basic functionality of a biological neuron. These artificial neurons receive input data, which is processed, and the results are given as output. The variables involved are the *weights* and *biases* of each neuron. The input data for each neuron is multiplicated with its weight variable, to produce the data it gives as output.

A neural network consists of layers of these artificial neurons or perceptrons, where input and outputs from the neurons are shared between the layers. Each layer can contain many neurons, that all receive input from the neurons in the previous layer, and forward their output to the next one. In an ANN, input data comes in through the input layer, and is then processed by the neurons in the first layer, whose outputs are forwarded as input to the next layer. The last layer of an ANN is the output layer which outputs the final values, from the initial input data that was processed through the entire network. The layers in between the input and output layer are called hidden layers. While the input and output layer size must conform to the size of the input data and expected output data, the size of the hidden layers do not have to, and can therefore vary. Figure 2.1 shows an ANN with input and output layers with three neurons, and one hidden layer with five neurons. In addition to the weight of each neuron, biases can be added as additional variables used in the process of calculating the output. This processing of input data to output data, is called the *activation function* of the neuron. The activation function of the neurons in the output layer is referred to as the *output function*, which gives a final prediction or classification for the network. The advantage of an ANN is its ability to learn very complex features simply by training on data, and produce predictions. The user does not need to understand the complexity of the neural net in detail, but simply by training, testing and validating an effective neural net can be produced and verified.

To be able to train and improve, the loss functions (also called the cost functions) are used to find values that indicate how well the network is performing on training data, compared to the correct answers. The goal of a network is to minimize the loss function during training, resulting in a more optimized network. There are different loss functions that are used, with different intentions regarding whether the network is used

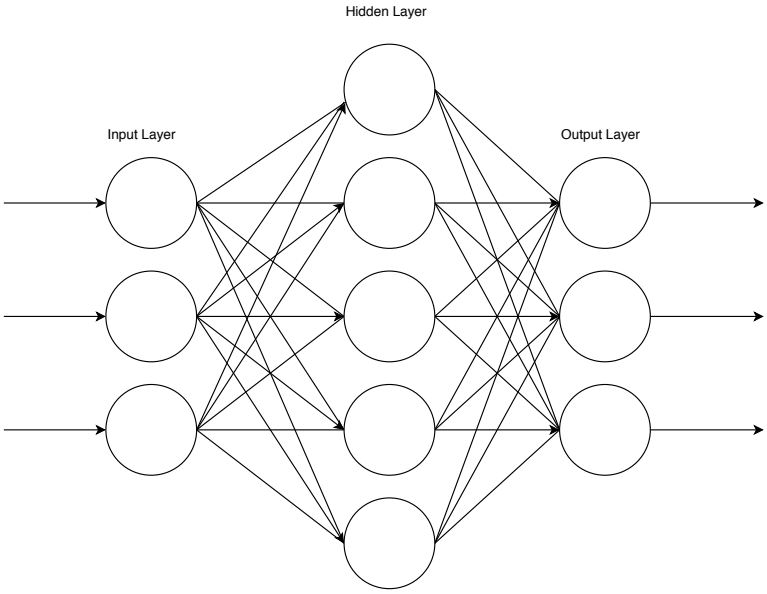


Figure 2.1: Artificial Neural Network with one hidden layer.

for classification or regression. For classification, it is normal to use a variation of the softmax classifier which makes use of cross entropy loss, which will measure the correct versus the incorrect predictions. For regression, a common loss function is to use the model's predicted distance against the real value and then regularize it, to help prevent overfitting on the training data.

By using the loss function, the network can be updated to try to minimize the loss by use of backpropagation. After the network has processed and returned an output, the loss is calculated, and then backpropagation (Rumelhart et al., 1986) is executed, by propagating the loss value (often referred to as error) back to all the neurons in the network, to give each neuron an associated error value in relation to the neuron's contribution to the final output. With backpropagation, we subtract the derivative of the loss function with respect to the weight of each neuron, scaled by a learning rate. This algorithm is called gradient descent, which is a type of iterative optimization algorithm. The terminology of maximizing the input through Maximum Likelihood Estimation (MLE) is often used, but this is equivalent to minimizing the loss function through backpropagation.

2.2.4 Recurrent Neural Nets

Recurrent Neural Networks (RNN) is a family of neural networks for processing sequential data (Goodfellow et al., 2016). An RNN can scale to and process much longer sequence lengths than other general ANNs. The RNN possesses a memory of previous input calculations in the network's internal state, which makes it more suitable for natural language processing, because textual semantics depend on the sequential order of words.

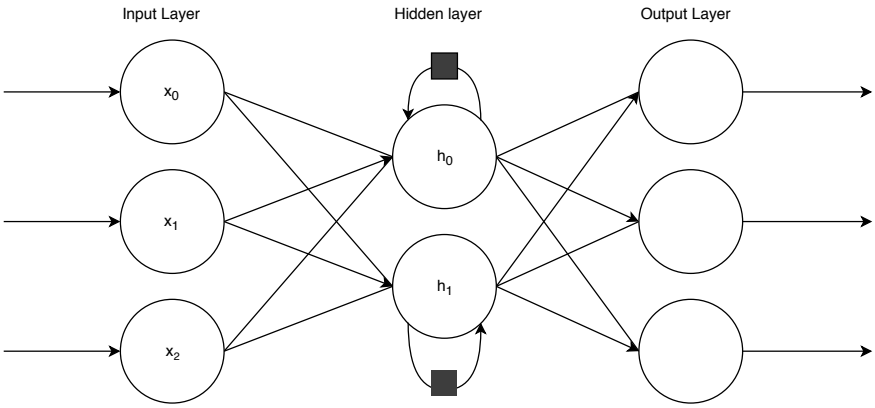


Figure 2.2: Recurrent Neural Network with one hidden layer.

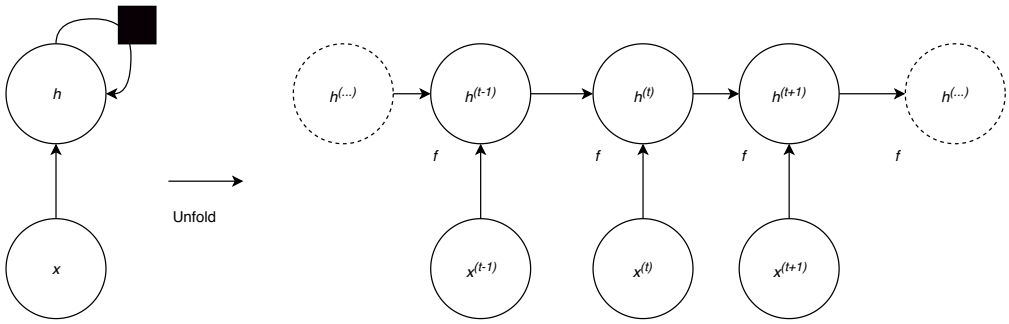


Figure 2.3: A Recurrent Neural Network unfolded over time. Figure adapted from Goodfellow et al. (2016), with permission from the author.

An RNN can for example be used to predict the upcoming word in a created text, based on previous words, or notes of music based on the previous notes.

Figure 2.2 shows a model of a Recurrent Neural Network. The similarity to the previous neural network in figure 2.1 can be seen, with the difference being the self-recurrent connections in the hidden layer, which represent the recurrent aspect of the network. This is what is responsible for the memory of the network, as the calculations at each time step in the hidden layer are sent to the next time step, so information about previous calculations are kept in memory.

Figure 2.3 shows a RNN with no outputs. This recurrent network just processes information from the input x by incorporating it into the state $h(t)$ that is passed forward through time. By looking at the model on the right, which is unfolded, we can see the network as a computational graph, where each node is now associated with one particular time instance.

Equation 2.2 illustrates how the hidden state $h(t)$ at each time step t is calculated, based on the previous time state h_{t-1} and the current input x. θ represents the parameters that are being learned to produce the best output.

$$h(t) = f(h_{t-1}, x_t; \theta) \tag{2.2}$$

2.2.5 Long Short-Term Memory Networks

The Long Short-Term Memory (LSTM) is an improvement of the regular RNN, and is more capable of learning long-distance dependencies. It was introduced by Hochreiter and Schmidhuber (1997). LSTM is useful when there is a greater distance between cases that are somehow related, for example words in a text, since it can store values over arbitrary intervals as memory.

Instead of neurons in the recurrent hidden layer, the long short-term memory has LSTM units. There are different architectures for LSTM units, but a common architecture is that a LSTM unit is made up of four components. The main one — the memory component (often referred to as the *cell*) — is self-connected, and responsible for keeping track of the dependencies between the elements in the input sequence. In other words, it stores the temporal states of the network. In addition, there are three components: the input gate, output gate, and forget gate. The *input gate* controls the values that are sent into the cell, the *forget gate* controls the extent of which a value remains in the cell, and the *output gate* controls the value in the cell which is used to compute the output activation of the LSTM unit.

While each *cell* has its own inputs, outputs and memory, cells can be combined in a single *block* with input-, output- and forget gates. In this case, the cells share the same gates. This way, each cell can hold a different value in its memory, but the memory of the block is written to, read from, and erased at the same time. Blocks sharing the same layer where they receive their inputs from and feed their output to, make up a new collective outer layer which they all are a part of, by sharing the input and output.

Figure 2.4 (on page 12) shows an early depiction of an LSTM, without forget gates. The memory blocks containing memory cells make up the hidden layer.

The multiplicative gates control the data that comes in and out of the memory cells. By closing specific input gates, the cells connected to that input can store and therefore access the information in memory over a long period of time, by not having it overwritten by new input. Therefore the purpose of the input gates is to control the flow of input activation into the network layer. The output gates control the output of data on to the following layer. There are connections into and out of the different gates, where some of the connections are recurrent. The weights of these connections have to be learned through training, and they are what controls how the gates operate.

While the LSTM still resembles the structure of the RNN, where the hidden state is calculated based on the previous state and the current input, the calculation of the LSTM is different. It uses five different equations, where the first three are for calculating the activation vector for the three different gates (input, output, forget), while the last two calculate the cell state vector, and the hidden state vector — which is the output vector of the LSTM unit (Hochreiter and Schmidhuber, 1997; Gers et al., 2000).

Equations 2.3, 2.4 and 2.5 (shown on page 12) are for calculating the activation vectors, where f_t is the vector for the forget gate, i_t for the input gate, and o_t the output gate.

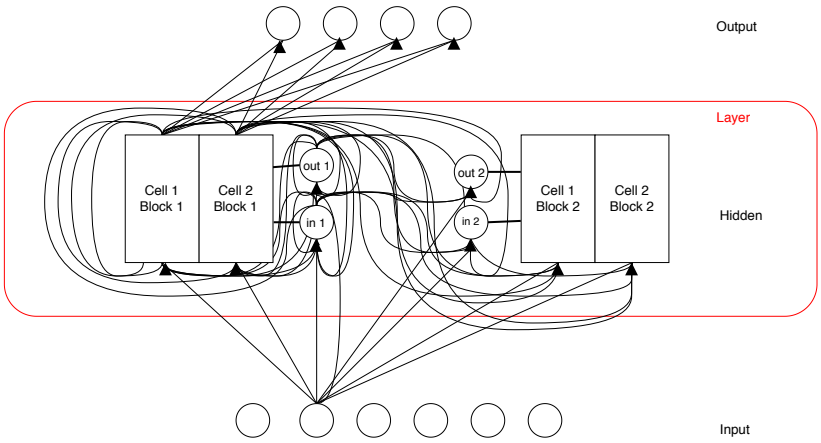


Figure 2.4: A Long Short-Term Memory model. Figure adapted from Hochreiter and Schmidhuber (1997), with permission from the authors.

t indicates the time step. The weights of the forget gate vector control to what degree the LSTM unit remembers information. The input vector is the weight of the degree to which a unit is acquiring new information, similarly for the output being the information passed on.

$$f_t = \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \tag{2.3}$$

$$i_t = \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \tag{2.4}$$

$$o_t = \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \tag{2.5}$$

W, U represent the weight matrices and b the bias vector parameters which are learned during training. σ_g represents the activation function, which is often a sigmoid function. h is the hidden state vector.

After these calculations, the cell state vector c_t is updated, using Equation 2.6 (on page 13). It takes the Hadamard product of the forget gate vector and the previous cell state vector c_{t-1}, and adds the Hadamard product of the input gate vector with

an equation equal to the earlier gate vector equations, this time using the cell state's weight matrices and bias vector, and the activation function σ_c — which in this case is the hyperbolic tangent function. The operator $\circ$ denotes the Hadamard product.

$$c_t = f_t \circ c_{t-1} + i_t \circ \sigma_c(W_c x_t + U_c h_{t-1} + b_c) \tag{2.6}$$

Finally the new hidden state h_t is calculated, using Equation 2.7. It takes the Hadamard product of the output gate vector and an activation function — in this case the hyperbolic tangent — multiplied with the cell state calculated in the previous equation.

$$h_t = o_t \circ \sigma_c(c_t) \tag{2.7}$$

LSTM is the chosen network that is implemented in this project, chosen for its obvious advantage for text generation, and prevalent use in the state-of-the-art solutions for poetry generation, presented in chapter 3. The choice of the model and implementation process is described further in chapter 5. However, there are many other types of ANNs that also could have been chosen. These include the Gated Recurrent Unit (GRU) model (Cho et al., 2014), a very similar model to the LSTM, but that does not use a memory unit which the LSTM does. Another model is the Convolutional Neural Network (CNN), (LeCun et al., 2015), which takes advantage of local coherence in the input to cut down on the number of weights, and is often used for processing images.

2.3 Natural Language Processing

Natural Language Processing (NLP) is a sub-field of computer science concerned with the interaction between computers and human languages, including how computers can process and analyze natural language data, typically to be able to understand or derive meaning from human language. This field is heavily involved in computational linguistic creativity, since the main aspect of this field is processing and analyzing natural language.

Another aspect of the goal (see section1.2) of this thesis is the use of Sentiment Analysis in combination with poetry generation. This refers to text being associated with a specific sentiment, which adds another dimension to the textual content. Sentiment analysis utilizes NLP techniques.

The theory behind text processing is presented, with regards to how language can be processed and represented for further use in computational linguistic creativity. Followed by the theory of sentiment analysis.

2.3.1 Text Processing and Representation

A major part of natural language processing is the processing and representation of text and data for further use. This proposes several challenges, generally with regards to how to be able to capture the most meaningful and interesting elements of a text, with the purpose of creating accurate numerical representations of elements.

Some general processing methods in NLP include *Tokenization* and *Part-of-Speech Tagging*. Tokenization is the task of finding the words in a document. By analyzing

a character sequence and a defined document unit, *tokens* are collected, which often will be words or terms. The challenge with tokenization is the processing of a text, what characters can be disregarded, and what are words/terms/character sequences that should be tokenized. Other processing tasks include text-generalization, e.g. setting all letters to lower-case, or the use of *stemming*, transforming words to their root forms, so not to differentiate between the inflections.

Part-of-Speech (POS) Tagging builds on the tokenization process by categorizing the tokens selected and assigning them part-of-speech tags. Examples are verbs, nouns, etc. This is not always a simple task, so POS-tagging often needs to take an entire character sequence in the form of a sentence, or even a larger part of the text being processed, into consideration for the tagging process.

Some existing models for textual representation include Bag of Words (BoW), which is used in both NLP and Information Retrieval (IR). It represents texts or documents as numerical feature vectors, by creating a vocabulary of unique tokens, and creating a feature vector for every document or text and counting the number of times each token occurs. BoW does not take word order into account.

Another type of language modelling is word embeddings, which is a collective term for models that learn to map a set of words in a vocabulary to vectors, making them a representation of text in an n-dimensional space. One word embedding technique is the model *word2vec* (Mikolov et al., 2013). The word2vec embedding method finds relations between words in a text or a sentence, measuring syntactic and semantic similarities. The model consists of shallow, two-layer neural networks that are trained to reconstruct linguistic contexts of words. It uses a large corpus of text as input, to produce a vector space with each unique word in the corpus being assigned a corresponding vector in the space. These word vectors are positioned in the vector space such that words that share the previously mentioned similarities are located close to one another in the vector space. Other word embeddings models include *gloVe* (Pennington et al., 2014), which uses word-to-word co-occurrence to build its model, and *BERT* (Devlin et al., 2018), which uses bi-directional encoder representations from Transformers. By making use of the attention mechanism Transformer that learns contextual relations between words, BERT applies a bi-directional training of Transformer to language modelling.

2.3.2 Sentiment Analysis

Sentiment analysis, also referred to as opinion mining, is the field of study that analyzes people's opinions, sentiments, evaluations, appraisals, attitudes, and emotions towards different entities (Liu, 2012). Sentiment analysis has been investigated mainly at three levels, namely document level, sentence level, and entity and aspect level. The document level refers to the task of classifying whether a whole opinion fed document expresses a positive or negative sentiment. Sentence level is the task of determining whether a single sentence expresses a positive, negative or neutral opinion. Entity and aspect level looks directly at the opinion itself, and is based on the idea that an opinion consists of a *sentiment* (positive or negative) and a *target* (of the opinion). The goal of this level of analysis is to discover sentiments on entities and/or their aspects.

Opinions can also be classified into two different types, called *regular opinions* and *comparative opinions*. A regular opinion expresses sentiment only on a particular entity or an aspect of the entity, while a comparative opinion compares multiple entities based on some of their shared aspects.

The most important indicators of sentiment are *sentiment words*, also referred to as opinion words. These are the words generally used to express a type of sentiment, examples being *good*, *great*, *bad*, *horrible*. In addition to the single words, there are also phrases and idioms used to express sentiment. A list of these words and phrases is called a *sentiment lexicon*.

The computational methods for sentiment analysis are usually based either on machine learning techniques such as naive Bayes classifiers trained on labeled dataset, or use lists of words associated with the emotional value (positive-negative evaluation or sentiment score values).

Several resources for sentiment analysis exist, including the Sentistrength sentiment analysis tool (Thelwall et al., 2010) and the WordNet-Affect lexical resource (Strapparava and Valitutti, 2004). These were used for poetry generation by Misztal and Indurkhya (2014), described in depth in section 3.3. Another sentiment analysis resource is Vader (Valence AwareDictionary for sEntiment Reasoning; Hutto and Gilbert, 2015), which is used in this project, and described further in the next section.

2.4 Frameworks

This section introduces the different frameworks and resources that are used in this project. Only the frameworks used for the poetry generation system — and only the relevant parts of the frameworks that were used in this case — will be described, but other alternatives and options are also mentioned.

2.4.1 Keras

Keras[1] is one of the leading high-level neural network APIs, which is written in Python and capable of running on top of TensorFlow[2]. It was developed with a focus on enabling fast experimentation, and provides support for implementing neural networks, including recurrent networks. It can run both on CPU and GPU. With a focus on user friendliness and easy extensibility, aspects such as neural layers, optimizers, and cost and activation functions are all standalone modules that can be combined to create new models, which can be defined in Python code. Keras is backed by Google, Microsoft, Nvidia, and as of TensorFlow 2.0 it will be the primary TensorFlow API. In addition to its popularity and focus on neural network implementation, it is also an API the author has previous experience with using, and it will therefore be used for implementation in this thesis (described in chapter 5). Another possibility for implementation is using TensorFlow without Keras.

[1] https://keras.io/
[2] https://www.tensorflow.org/

2.4.2 Natural Language Toolkit

The Natural Language ToolKit (NLTK)[3] is a leading platform for building Python programs for processing human language data. It provides interfaces to many useful corpora and lexical resources, and a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing and semantic reasoning. Several of the resources available in NLTK are used in this project.

The process of tokenization and tagging of the data set (chapter 4 and 5) is done using NLTK, including the use of POS-tagging. Another resource available with NLTK is the CMU Pronouncing Dictionary (CMUdict)[4]. This an open-source pronunciation dictionary originally created by the Speech Group at Carnegie Mellon University (CMU), that provides a phonetic mapping for American English word pronunciation. Commonly used to generate representations for speech recognition, it has also been used for generating rhymes in generated text (Ghazvininejad et al., 2017).

WordNet[5] is another resource available with NLTK that is relevant for this project. It is a large lexical database of English, where nouns, verbs, adjectives and adverbs are grouped into sets of cognitive synonym (synsets), each expressing a distinct concept. Synsets are interlinked by means of conceptual-semantic and lexical relations. This network of meaningfully related words and concepts can for example be used for finding related words to a single given word. ConceptNet (Liu and Singh, 2004) is another resource similar to WordNet, but consists of a knowledge graph derived from a semantic network, where the relationships are connected differently from WordNet, making it another valuable source of information.

2.4.3 Vader

Vader (Hutto and Gilbert, 2015) is a model used for text sentiment analysis, that is also available with NLTK. It is sensitive to both polarity (negative/positive), and intensity (strength) of the emotion. Vader uses a human-centric approach, which combines qualitative analysis with empirical validation by using human raters. For calculating the sentiment score of a given text, it combines a dictionary that maps lexical features to the intensity of an emotion, and five simple heuristics which encode how contextual elements increment, decrement or negate the sentiment of the text.

Vader scores the input words or sequences of words with a number, which represents one of three different areas. Neutral words are words with no sentiment value, given a score of 0.0, while negative words have a score that is of a negative value, and positive words have a score which is larger than 0.0. For scoring individual words, Vader provides a score between -4.0 and 4.0. When scoring a text, a sequence of words, Vader provides a score between -1.0 and 1.0. The scoring of a text is done by summing each Vader-dictionary-listed word in the text, and then normalizing the scores to create a score within the -1.0 to 1.0 boundary. In addition to summarizing word scores in a text, Vader

[3] https://www.nltk.org/
[4] http://www.speech.cs.cmu.edu/cgi-bin/cmudict
[5] https://wordnet.princeton.edu/

also takes into account other contextual elements, like punctuation, capitalization, etc., by considering five simple heuristics. The effects of these heuristics are, like the score for each dictionary-listed word, quantified using human raters. These five heuristics are:

- Punctuation.
- Capitalization.
- Degree of modifiers (VADER maintains a booster dictionary which contains a set of boosters and dampeners. Booster and dampeners are words that can be seen as *intensifiers* or *degree* adverbs, that express a particular degree to which the word they modify applies).
- Shift in polarity (due to "but").
- Examining the tri-gram before a sentiment-laden lexical feature to catch polarity negation (Tri-gram refers to a set of three words. VADER maintains a list of negator words, and negation is captured by multiplying the sentiment score of the sentiment-laden lexical feature by an empirically-determined value -0.74.).

Vader is chosen to be used for sentiment analysis of both individual words and poetry text, described in chapter 5 in this project, both due to its accessibility through NLTK, and its superior dictionary for intensity scoring, and scoring of text.

2.5 Evaluation of poetry

One of the challenges of automatic poetry generation is evaluating the poetry that is generated. Machine-based approaches have been used for evaluation generated poetry of poetry, namely two methods called Bilingual evaluation understudy (BLEU) and METEOR. BLEU (Papineni et al., 2002) is a method for automatic evaluation of machine translation. By taking two different system translations of a sentence, it compares them to multiple reference lines that are written by humans to find the better one. This is done by comparing the n-grams of the system translations with the n-grams of the human-written lines. The METEOR (Banerjee and Lavie, 2005) evaluation was designed to address several weaknesses with the previous BLEU evaluation. It is based on the harmonic mean of unigram precision and recall, where recall is weighted higher than precision. In addition, it includes stemming and is able to find synonyms, as well as including standard exact word matching. It also differs from the BLEU metric by finding correlation at a sentence level, as well as corpus level, while BLEU is just at corpus level.

Another solution is using human evaluation of generated poetry. A problem with the machine-based evaluation approaches is that they have little correlation with human evaluation (Liu et al., 2016), but also that they lack the diversity of human evaluation (Mou et al., 2016). For this reason, human evaluation will be used to evaluate the poetry generated in this project. For human evaluation, three criteria are commonly used: *grammaticality, meaningfulness* and *poeticness* (Manurung, 2004). These three criteria are specified as:

- **Grammaticality**: The poem must obey linguistic conventions that are prescribed by a given grammar and lexicon. This refers to text being both lexically and syntactically correct enough, to the point where the reader is able to both read and generally understand the text. Several traits such as grammatical errors, or vague or surreal use of grammar or words can often be intentional in poetry, so the point of this criterion is to rule out random sequences of words.
- **Meaningfulness**: The text must intentionally convey some conceptual message which is meaningful under some interpretation. This can involve the poem containing a sentiment, but this is not the sole focus of this criterion, as sentiment will be evaluated separately, shown in chapter 6.
- **Poeticness**: The poem must exhibit poetic features. This refers to phonetic features such as rhymes, but also form, etc.

These have been similarly defined and used by others, e.g. Zhang and Lapata (2014), often with minor differences or additional criteria. One example is the criterion of coherence (Yan, 2016), which evaluates whether the poem is thematically coherent across lines. A criterion that has been used instead of poeticness is fluency (Yi et al., 2018), which evaluates whether the lines are fluent and well-formed. Another is the evaluation of overall quality (Yi et al., 2016, 2018), which is defined as the reader's general impression of the poem. As these other criteria all differ from each other and have various definitions, the three common criteria presented earlier — *grammaticality, meaningfulness* and *poeticness* — will be used in this thesis.

3 Related Work

This chapter covers the related work that has been done on poetry generation, and presents state-of-the-art approaches. In the first section, several different approaches that have been applied through the years are presented, to give an overview of the different possibilities and solutions that exist. The second part of the chapter covers more recent approaches that implement neural networks, and these approaches represent the state-of-the-art solutions to date. Finally, the last section shows how some of the related work have incorporated sentiment in poetry generation, which is relevant to the goal of this project (section 1.2).

3.1 Different Approaches to Poetry Generation

This section covers the different approaches that have been implemented for poetry generation through the years, excluding the neural network approaches, which are presented in the next section. The approaches presented in this section are:

- Template-Based Poetry Generation.
- Generate and Test Approaches.
- Evolutionary Approaches.
- Case-Based Reasoning Approaches.
- Corpus-Based Approaches.
- Blackboard Architecture.

The earlier approaches to poetry generation could be divided into four different general methods, according to Gervas (2000), and those are the first four approaches mentioned in list above. In addition to these four, Corpus-Based approaches and Blackboard Architecture approaches have also been implemented for generating poetry. The theory behind all these different approaches is presented, including examples of systems that have implemented them.

3.1.1 Template-Based Poetry Generation

Template-based poetry generation makes use of a basic, incomplete template which is filled out by the program, following certain rules and constraints — typically syntactic and/or

rhythmic constraints — and using words or phrases from a set dictionary. This approach, while fairly simple, can produce poetry of high quality, but will lack in originality. The possibilities during the generation process of this approach are very limited, compared to other approaches presented later in this chapter, which allow for modification and the generation of every word in the text that is created.

PoeTryMe (Gonçalo Oliveira, 2012) is a newer system using template-based generation. The template for this system contains the poem's structure, including the number of stanzas, the number of lines per stanza and the number of syllables of each line. The poem is then generated according to a set of seed words, which are used to get sentences from a sentence generator. Several heuristics can be included to find better sentences for each line of the poem, which consider features like meter, rhyme or coherence between lines.

The sentence generator is the core module of the PoeTryMe architecture, which creates meaningful sentences. It uses both a semantic graph, where nodes are words and edges are relation types, and generation grammars containing textual templates for generation of grammatical sentences denoting a semantic relation. The generation process of a sentence starts with a set of seed words, which are used to select a subgraph from the main semantic graph. The process proceeds by selecting a random triple in the subgraph, and a random grammar rule matching its relation type. After inserting the arguments of the triple in the rule body, the resulting sentence is returned.

3.1.2 Generate and Test Approaches

Generate and test is another approach that is fairly simple. It is based on generating a random sequence of words that are produced in accordance with certain formal requirements such as defined metrics and semantic constraints.

One of the earlier systems that implemented this method was the Wishful Automatic Spanish Poet (WASP) by Gervas (2000). The WASP system is a forward reasoning rule-based system that takes a set of words and a set of verse patterns as input, and then returns a set of verses. It draws on prior poems and a vocabulary selection provided by the user, to generate a new metrical combination according to line patterns from prior poems. The basic algorithm for generation starts with selecting an appropriate verse pattern, based on criteria designed to ensure a minimum of coherence across verse boundaries. From this pattern an empty draft of the current verse is generated. A generation cycle is run, by randomly choosing a word from a vocabulary that matches the first category of the current verse pattern, followed by appending the word, eliminating the corresponding category from the current verse pattern, and testing if the resulting verse draft satisfies the conditions of the strategy being used. Verses that either violate the conditions or miss an accepted number of syllables, are rejected. The system was able to produce poetry that followed the metrics of rhythm and rhyming, but did not obey general linguistic conventions, resulting in a text that was both lexically and syntactically incorrect.

3.1.3 Evolutionary Approaches

Evolutionary algorithms are based on the computing technique of trying to mimic the process of biological evolution, such as natural selection or genetic inheritance. As defined in Michalewicz (1996), an evolutionary algorithm is a multi-point stochastic search algorithm, which means it is a form of heuristic search that simultaneously explores several points in a search space. A main evolutionary algorithm generally consists of five steps, which have been defined several different ways. One definition was done by Manurung (2004), where these five steps are described as:

- **Initialization**: Construct a new population which represents a set of starting points to explore the search space. Ideally, the distribution of points is evenly spread out across the space.
- **Evaluation**: Each solution is evaluated to give some measure of its "fitness" (a fitness function is applied to each individual, yielding a numerical score that indicates its suitability as a solution to the problem at hand).
- **Selection**: A new population is formed by stochastically selecting individuals, usually with a bias towards fitter individuals.
- **Evolution**: Some members of the new population undergo transformation by means of "genetic" operators to form new solutions.
- **Repeat**: Repeats the steps for evaluation and evolution until termination, which is achieved after a given number of iterations, or a given fitness score is reached, or the algorithm has converged to a near-optimal solution.

One system using the evolutionary approach is POEVOLVE (Levy, 2001), that generates limericks. The system contains several parts: the generator module, the evaluator module, the workspace, the lexicon, the conceptual knowledge base and the syntactical knowledge base. It operates over a representation of a lexicon of phonetic information. Evaluation is done by a neural network that is trained on judgements given by human evaluators. An initial population is randomly generated and allowed to evolve by applying a set of operations (mutation, crossover and direct copy), which in general terms are restricted to substituting one word for another. A problem with the system is that it fails to take syntax and semantics into consideration.

3.1.4 Case-Based Reasoning Approaches

Case-Based Reasoning (CBR) is the process of solving a problem (which is the *case*) based on the solutions of similar problems or *cases*. In the use of this approach for poetry generation, a system first retrieves an existing poem and then adapts this retrieved poem based on input from the user to fit the content.

A system using this approach is the ASPERA (Automatic Spanish Poetry Expert and Rewriting Application) system (Gervás, 2001), which is an evolution from the previously

described WASP system (Gervas, 2000). ASPERA is a forward reasoning rule-based system, where the generation starts with getting input from the user that will be used as features, like rhyme structure, mood and the length of the poem. Additionally, the input also includes an intended message by the user. The initial target of the generation is the intended message, and it retrieves a case to later use the solution as seed structure which to fill in. The intended message is used as main source for the words required to fill in the case, and the case itself as default source. An additional vocabulary that is also provided by the user is used as intermediate source.

Case-based reasoning is generally formalized as a four-step process, first defined by Aamodt and Plaza (1994). In the context of poetry generation, the process is used to generate each line of the poem, where the four steps can be defined as:

- **Retrieve**: The retrieve step takes each sentence from the intended message (target for generation), and retrieves a specific verse from a corpus.
- **Reuse**: The reuse step uses the verse template to construct a line based on the part-of-speech structure of the verse.
- **Revise**: The revise step asks the user to validate the produced draft.
- **Retain**: The final step analyzes and stores the validated poems, making it possible to use them later.

ASPERA also includes additional control features in the form of system parameters. These control how the system splits the target content over the number of lines in a chosen stanza, the similarity employed by the system to retrieve cases to use as a seed for the construction process, the number of syllables per line, number of lines in the stanza, and the amount of variation in relative position of a word between the target content and the final result. This way, the system is provided with a certain degree of freedom that can be controlled and set by the user, by either producing general versions that replicate an important portion of the inspiring set, or more innovate versions that try to depart from the inspiring set.

3.1.5 Corpus-Based Approaches

A different approach, outside of the four general presented by Gervas (2000), is the corpus-based method. This method is based on using the structure of a set of poems to generate new ones. It can use multiple corpora, and often substitutes words based on their POS-tag and relevance.

An example system that uses this approach is Full-FACE Poetry Generation (Colton et al., 2012). It is a corpus-based poetry generation system that uses templates to construct poems according to given constraints on rhyme, meter, stress, sentiment, word frequency and word similarity. It constructs a mood for the day by analysing newspaper articles, and uses this to determine both an article to base a poem on and a template for the poem. The algorithm behind the generation consists of four steps:

- **Retrieval**: The system mines similes from the internet depending on the sentiment and evidence.
- **Multiplication**: Objects, words and phrases are substituted to create variations of these similes.
- **Combination**: Similes and their variations extracted are plugged into a template given by the user.
- **Instantiation**: Random phrases are then chosen from an elaborate set to fill the fields of a user-given template.

Key phrases of the chosen newspaper articles are extracted and denoted to words, and the key phrases are given a relevancy score. The mood of the day will be projected through the use of the extracted words in the poem. This process includes the use of sentiment analysis, and is described in depth in section 3.3. It also creates an aesthetic based on relevance to the article, lyricism, sentiment and flamboyancy, and searches for an instantiation of the template which maximises the aesthetic. These four measures create more variety by giving the software the possibility to choose from any of following measures, which are defined as:

- **Appropriateness**: The distance between the average sentiment of the words in the poem measured against the chosen mood of the day.
- **Flamboyance**: Use of words that did not appear in the extracted set of words, or word that already have appeared in the poem
- **Lyricism**: The proportion of linguistic constraints adhered to in the poem.
- **Relevancy**: The average of the relevance over the words in the poem — words that appeared in a chosen article.

Commentary is provided during the whole process, to add value to the creative act. Colton et al. (2012) argue that it is the first poetry system which generates examples, forms concepts, invents aesthetics and frames its work.

3.1.6 Blackboard Architecture

The Blackboard Architecture is a model visualized by a metaphor (Corkill, 1991) of a group of independent experts with diverse knowledge who are sharing a common workspace (the blackboard). The workspace contains partial solutions, including other information about the given problem, and the workspace is iteratively updated by the different experts, who can be seen as specialist knowledge sources. Each expert updates the blackboard with a partial solution when its internal constraints match the blackboard state.

One system that uses this approach is Misztal and Indurkhya (2014). In their system, the blackboard consists of the input text used as inspiration for the poem, the initial

constraints and information about the poem, key phrases, a topic, an emotion, a pool of ideas that is a part of the blackboard used as a workspace for experts, and a poem draft.

The poetry generation algorithm can be divided into several phases. The first phase is where the blackboard is initialized with the text input by the user. The form of the poem is selected from a set of templates, and grammar constraints are defined for stylistic consistency. The poem-making experts and the emotional expert are initialized. The next phase selects the topic, chosen by finding the key phrase with the highest inspiration score, and the emotional expert also finds the emotional state for the poem (described in depth in section 3.3). Then words are generated by the relevant experts, followed by the generation of phrases using these words, by another expert. When the generation is finished, selection experts select the phrases that best fulfill the line constraints.

Using the general criteria of grammar, meaningfulness and poeticness, the results showed poems that generally satisfy these constraints, with possibilities of improving the aesthetic measure by defining more stylistic constraints for the poem. More attention could also be paid to the context of analyzed words. While the poetry generation system itself is not the best solution to date, the solution for adding emotional personality to the generated poems is still relevant.

3.2 Recent Approaches that implement Neural Networks

This section covers the more recent state-of-the-art approaches that have been implemented for poetry generation in the last few years. In this period, neural networks have proven to be powerful for poetry generation, and have become increasingly popular. All approaches and systems in this section include the use of neural models. First, several systems that have implemented neural models, and the advancement of this approach, are presented. Afterwards, several systems using neural models with additional techniques for enhancing poetry generation are presented, to give an overview of the possibilities that have been implemented. While the newer approaches presented in this section include the use of neural networks, they still implement and use elements from the previously presented approaches in section 3.1.

3.2.1 Advancement of Neural Models

One of the earliest attempts at generating poetry using deep learning was Zhang and Lapata (2014), who used a Recurrent Neural Network (RNN) to generate Chinese quatrains (stanzas with four lines). This system is also built on the user providing input (keywords in this instance) highlighting the main concepts around which the poem will revolve. The keywords are restricted to those attested in the ShiXueHanYing poetic phrase taxonomy. The generator creates the first line of the poem based on these keywords. The subsequent lines are generated base on all previously generated lines, subject to admissible tonal pattern and structural constraints.

The process of generation includes starting with a convolutional sentence model (CSM) for converting lines into vector representations that will be used in the system. A recurrent

context model (RCM) then calculates the context of each of the words. The output from the RCM is used as input in a new recurrent generation model (RGM) to find the next character. While the results showed that the machine-generated poems still lagged behind the human-generated ones, it scored better in both a BLEU-2 evaluation and a human evaluation than other state-of-the-art systems.

A newer system, using the sequence-to-sequence model with attention mechanism (Bahdanau et al., 2014) with an RNN Encoder-Decoder, was created by Yi et al. (2016). This system uses a bi-directional RNN with gated units, instead of the simple RNN as in Zhang and Lapata (2014). The system generates Chinese classical quatrains, where there is a close semantic relevance between two adjacent lies. The RNN Encoder-Decoder learns the relevance which is then used to generate a poem line given the previous line. For utilizing context information, three poem line generation blocks are created. First the user inputs a keyword as the topic to show the main content and emotion the poem should convey. Then the Word Poem Block generates a line relevant to the keyword as the first line. Followed by the Sentence Poem Block taking the first line as input and generating a relevant second line. Finally, the Context Poem Block generates the third line with the first two lines as input. Yi et al. (2016) conclude that the RNN Encoder-Decoder is also suitable for learning tasks on semantically relevant sequences.

Another system, implemented by Wang et al. (2016a), used an attention-based LSTM model for Song iambics generation. It accepts a set of keywords as the theme and generates poems by looking at each keyword during the generation. It also uses the sequence-to-sequence model with attention mechanism as the main component of the system (similar to the system of Yi et al. (2016)), but it additionally has a bi-directional LSTM model as the encoder, and another LSTM model for the decoder, which alleviates the quick-forgetting problem associated with the conventional RNN models. The process of generation includes the input being converted by the encoder to a sequence of hidden states to represent the semantic status at the position of the input. These hidden states are then used to regulate the decoder that generates the target sequence. The decoder then generates the whole poem character by character. The prediction for every next character is based on the current status of the decoder as well as all the hidden states of the encoder. In addition to the implementation of bi-directional LSTM, other techniques such as character vector initialization, attention to input, and hybrid-style training were included for further improvement.

3.2.2 Format Constraints

Hafez (Ghazvininejad et al., 2017) is a poetry generation system that combines hard format constraints with a deep learning recurrent network. It generates 14-line classical sonnets with a specific rhyme scheme, in iambic pentameter. The system first gathers a large vocabulary, computing stress pattern for each word. Then it is given a user-supplied topic, which is used to retrieve a large set of related words, using *word2vec* (Mikolov et al., 2013). Ghazvininejad et al. (2017) state that the training corpus for word2vec has a crucial effect on the quality of the related words. The word2vec model (which are pre-trained on the English Gigaword corpus and the first billion characters from

Wikipedia) were additionally trained on a song lyrics corpus. Rhyme words are then found by the system, and put at the end of each line so that they match each other (to make a rhyme). Using the CMU Pronouncing Dictionary, the system tries to find rhyming words with related words. If this is not successful, adding fixed pairs of often used words makes the system find rhymes in rare topics. In addition, the system includes a Finite-state acceptor (FSA). It is built with a path for every conceivable sequence of vocabulary words that obeys formal rhythm constraints, with chosen rhyme words in place. Finally, a recurrent neural network is used for scoring, to select a fluent path through the FSA.

Ghazvininejad et al. (2017) state that the results of this system show that with strong conditions on rhyme, meter, repetition and ambiguously-stressed words, the plagiarism that is common for optimal-searching RNNs to repeat large sections of the training data, is reduced.

Another system based on format constraints was implemented by Benhart et al. (2018). They also combined a deep learning recurrent network with specific format constraints constituting the sonnet format of poetry, but in addition implemented Part-of-Speech restrictions based on the observed errors and wrong word choices the system generated during the early stages before the final system was complete. Benhart et al. (2018) also implemented dynamically trainable word embeddings, where in the early stages of training the language model used fixed word embeddings as input, but in the later stage they were made trainable. This meant that the language model was able to learn some grammar before adjusting its word representations to suit the training corpus. By including these novel elements in their poetry generation system, Benhart et al. (2018) state that they were able to improve over the state-of-the-art, leading to rhythmic and inspiring poems. The system was also the winner of the 2018 PoetiX Literary Turing Test Award for computer-generated poetry.

3.2.3 Planning Schema

Wang et al. (2016b) implemented neural networks in a system with a special planning schema, which plans some sub-keywords in advance by a language model and then generates each line with the planned sub-keyword to improve coherence. The generation can, therefore, be split into two phases, the first being the poem planning phase. Here written intent from the user is taken as input, which is transformed into a specific number of keywords, which relates to the number of lines in the poem. Each of the keywords is assigned to an individual line, resulting in each line having a sub-topic assigned to it. The planning schema features keyword extraction and keyword expansion. The TextRank algorithm (Mihalcea and Tarau, 2004) is used to evaluate the importance of the words in the written user input, which can be a document of text. TextRank is a graph-based algorithm where each candidate word is represented by a vertex in the graph, and edges are added between two words according to their co-occurrence.

After the planning stage, the generation stage proceeds by using all previously generated text and the keyword belonging to the current line as input, for generating the poem sequentially line by line according to the sub-topic and all preceding lines. The poem

generation part uses the encoder-decoder structure with GRU models ((Cho et al., 2014)), that is similar to Wang et al. (2016a), modified to support multiple sequences as input from the user. The sub-topic is encoded to a sequence of hidden states, and the preceding text into a bi-directional GRU model.

3.2.4 Polishing Schema

Unlike the traditional one-pass generation for previous neural networks models, Yan (2016) proposes a new generative model with a polishing schema called iPoet, which refines the poem that is originally generated in one pass, over several iterations. The poem is therefore generated incrementally and iteratively by refining each line. This is inspired by the fact that the generation model is more like a real human poetry composing experience, with re-thinking and re-wording during the process.

The system is based on using recurrent neural networks (RNN) for language generation, which has previously been done by Zhang and Lapata (2014). The iPoet system takes a large collection of poems, and learns the representation of individual characters and their combinations into one or more lines as well as how they mutually reinforce and constrain each other. Human written intent is used as input for the system framework. The representation of the written intent is created by using a CNN or RNN on the characters to capture the meaning of a particular keyword term. The information of different terms is integrated by a pooling layer, thus obtaining a single vector representation of the user intent. The encoded input is then decoded via recurrent neural networks with hierarchical structure, i.e., representations of "characters" and "lines" in two hierarchies. Concretely, one RNN represents *global* information for each line: the global information vector impacts on all character generations in the line. In addition, another RNN based on the global RNN represents *local* information: it guides the generation of a single character within a line. Finally, the iterative polishing schema process resembles sequential generation, with the difference being that the information representation of the previous draft of generated text is utilized as input and serving as additional information of user intention, as well as facilitating the overall semantic coherence for the whole poem.

Conducting experiments on perplexity and BLEU scores, and human judgement of four criteria — to evaluate the performance of the poetry — produced results that the iPoet neural model could generate poems that outperformed a specific baseline defined by Yan (2016). By also generating poetry without the use of the polishing schema or hierarchical structure, Yan (2016) showed that both the polishing schema and the hierarchical structure make prominent contributions to the performance of the iPoet system.

3.2.5 Memory Component

Zhang et al. (2017) describe another system that uses the same attention-based RNN generation model as Wang et al. (2016a). Here the neural model is also a sequence-to-sequence model, with GRU encoders. The training for the neural model component also follows the scheme defined in (Wang et al., 2016a). The difference compared to the

system of Wang et al. (2016a) is an additional memory component that is combined with the neural model. It can be regarded as an effective regularization that constrains and modifies the behavior of the neural model, resulting in generated text with desired properties. It involves a set of 'direct' mappings from input to output, and therefore can be used to memorize some special cases of the generation that cannot be represented by the neural model. The memory is created after the neural model is trained, by running the decoder of the neural model. At run time, the memory elements are selected according to how they fit the present decoder status, and then the outputs of the selected elements are averaged as the output of the memory component.

The results of using this method demonstrated that implementing the memory component can boost innovation of the poetry generated from two opposite directions, either by encouraging creative generation for regularly-trained models, or by encouraging rule-compliance for overfitted models.

3.2.6 Author-Stylization

Tikhonov and Yamshchikov (2018b) created a system for generating author-stylized poetry. An LSTM-based language model was used to predict the next word based on a previous word sequence of inputs, in addition to other parameters of the modeled sequence. Tikhonov and Yamshchikov state that one of the most widespread approaches for passing the needed parameters to the network is to use the initial state of the parameters as input, but a general weakness of this approach is that the network 'forgets' the general parameters of the document as the generated sequence gets longer. Their system supports the model at every step with the embeddings of the document that is currently being analyzed, differentiating their approach from a classical word-based LSTM model.

A schematic picture of the model is shown in Figure 3.1, document information projections are highlighted with blue and black arrows. An LSTM with 1152-dimensional input and 512 dimensional state space was used. The projections on a state space of the corresponding dimension is achieved with simple matrix multiplication of document embeddings. Another key feature of the proposed model is a concatenated word representation. Information about the document is included at every step. Final states of two character bidirectional LSTMs are also concatenated into a word embedding. The first of these two LSTMs works with letters from a char-representation of the word whereas the other uses phonemes of the International Phonetic Alphabet, employing a heuristic to transcribe words into phonemes. The approach of using bidirectional LSTM for this purpose is new according to Tikhonov and Yamshchikov.

The data used to train the model consists of two datasets, one of English and one of Russian poetry. All punctuation was deleted, and every character made lowercase. During the training phase the beginning and ending of every text was tokenized, so that in the generation phase the network is initialized with a special 'start' token, and is conditioned on values of document parameters. The proposed mechanism for the stylized poetry generation is tested with one categorical variable — the name of the author. The model was trained for English and Russian, and tests run on the poetry of Shakespeare, Edgar Allen Poe, Lewis Carroll and Oscar Wilde, as well as lyrics from the music of Bob

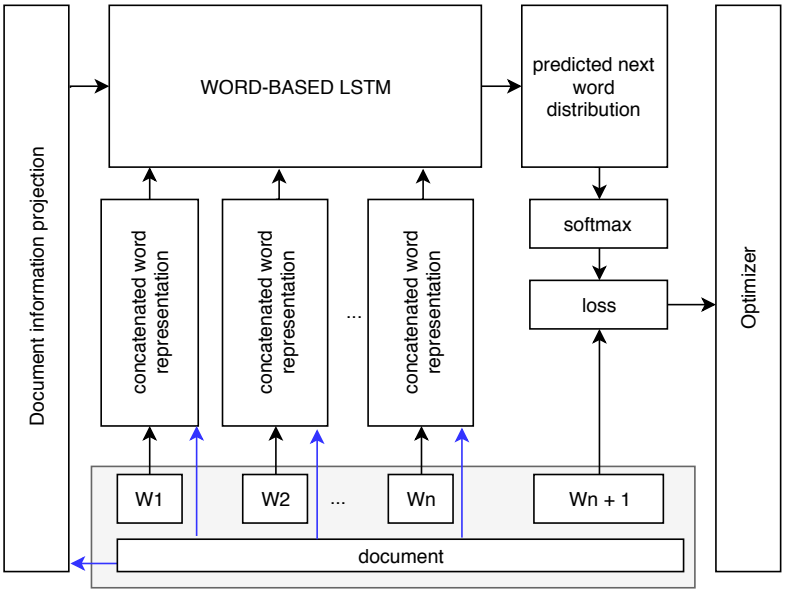


Figure 3.1: A Word-Based LSTM demonstrating the model of Tikhonov and Yamshchikov. Adapted from Tikhonov and Yamshchikov (2018b), with permission from the authors.

Marley, Nirvana and MUSE for English, and the poetry of Alexander Pushkin, Sergey Esenin, Joseph Brodsky, Egor Letov and Zemfira Ramazanova for Russian. The results of the tests show that the model captures the syntactic characteristics of the authors, and a certain resemblance to their respective vocabulary.

Tikhonov and Yamshchikov (2018a) state that according to the results of BLEU evaluation for a uniform and weighted random sampling, a vanilla LSTM, an LSTM with author embeddings but without phonetics, and the complete model, the extended phonetic embeddings play a significant role in the overall quality of the generalized stylized output. In addition, on human evaluation for recognizing author style, the evaluators were able to recognize the correct author style at least two times higher than a random choice, for several authors. It was also shown that humans mistakenly recognize the output of the proposed generative model for the target author as often as they correctly attribute original texts to the author in question.

3.2.7 Mutual Reinforcement Learning

Another solution for poetry generation was described by Yi et al. (2018). All previous existing models using neural networks for poetry generation are based on maximum likelihood estimation (MLE), and according to Yi et al. this brings about two substantial problems. Firstly, MLE-based models tend to remember common patterns of the poetry corpus (Zhang et al., 2017), such as high-frequency bigrams and stop words, thus losing some diversity and innovation for generated poetry. Secondly, based on a word-level

likelihood, two kinds of loss-evaluation mismatch arise. One is evaluation granularity mismatch; when evaluating, human experts usually focus on either a whole line at a time or even the entire poem, while MLE optimizes word-level loss, which does not hold a wider view of generated poems. The second being criteria mismatch, because humans usually evaluate poetry in terms of some criteria, as opposed to the likelihood. With this in mind, Yi et al. implemented reinforcement learning to a basic poetry generation model pre-trained with MLE. By simultaneously training two generator learners with deep reinforcement learning, the generators can both learn from the teacher (rewarder) and from each other. By having these two learners that can explore different directions, the one which is able to yield a higher reward can positively impact the other generator, by sharing information throughout the exploration process. The basic model is a modified version of Yan (2016), with the main differences being that the RNN is replaced by a GRU, convolution is used to calculate the line representation rather than directly using the last decoder hidden state, and the polishing schema is removed to better observe the influence of the deep reinforcement learning. The mutual reinforcement learning (MRL) implemented in this model uses two methods. The first is a local MRL, where if a learner creates a significantly better poem, the other learner will learn it. This gives the generator more high-reward instances and allows it to explore larger spaces along a more proper direction, avoiding a local minimum. The second is a global MRL. Here mutual learning is applied at the distribution level. Instead of individual instances, it is measured whether one of the learners achieves higher scores than the other during the creation history, and in that case, the learner with lesser scores should directly learn from the superior learner, rather than learning the poem by itself. These two methods are combined by simultaneously communicating high-reward samples and using Kullback–Leibler divergence (a measure of comparing two probability distributions).

The training is achieved by modelling four criteria for poetry evaluation: fluency, coherence, meaningfulness, and overall quality. Automatic rewarders corresponding to these criteria were designed. The fluency rewarder was created using a neural language model to find the probability of a given poem line to exist in the corpus, following that a higher probability would mean a more fluent and well-informed poem line. Using the probability directly as a reward was not done, since it may damage diversity and innovation. Therefore an equation to reward moderate probabilities that fall into a reasonable range was used.

The coherence rewarder valued the generated lines of poetry that were coherent with the previous line of the poem. *Mutual Information* (MI; Cover and Thomas, 1991), was used to measure the coherence between the generated line and the poem's previous line.

The meaningfulness rewarder uses TF-IDF values, as Yi et al. (2018) state that TF-IDF values for human-authored poems are shown to be significantly higher than values for generated ones. Therefore TF-IDF was utilized to motivate the model to generate more meaningful words. Because sampling of poems during the training process can result in out-of-vocabulary (OOV) problems with high variance when using TF-IDF directly, another neural network was implemented to smooth the TF-IDF values. For each poem line in the training sets, the standard TF-IDF values for all words were calculated, and

the average was used for the TF-IDF value of the line.

For an overall quality rewarder, a neural classifier was trained to classify the poems into three classes: computer-generated poetry, ordinary human-authored poetry and masterpiece. The four types of rewards were re-scaled to the same magnitude, and a total reward was found.

Comparing this system against human-authored poems, the basic model pre-trained with MLE, and the model of Zhang et al. (2017), the result showed that the mutual reinforcement learning method achieved significant improvement in the automatic rewards and human evaluation scores, outperforming the other generation models in both. The MRL system achieves better results in human evaluation on all the aspects: fluency, coherence, meaning and overall quality. It gets a fluency metric value that is close to the human generated poems, since fluency has been optimized. The biggest gap between the system- and human-generated poems is in the value metric of meaning. Meaning is the most complex criterion involving emotion expression, and while the utilization of TF-IDF improves the use of words on diversity and innovation which increase the meaningfulness value to some extent, there is still room for improvement.

3.3 Sentiment in Poetry Generation

Several of the systems presented earlier in this chapter have implemented a form of user input to influence the mood of the poetry, often related to a given sentiment. This section looks at two of these systems, and describes how they have included the use of sentiment in poetry generation.

The corpus-based approach of Full-FACE Poetry Generation (Colton et al., 2012), included a feature where the mood of the poem is decided by an article corpus, mainly by the use of sentiment analysis. This process is done by having every article in the corpus assigned a sentiment value between -5 and 5, based on the average of the sentiment of the words in the article. When a poetry generation session begins, the system checks the sentiment of a set N of newspaper articles posted during the previous 24 hours, and according to the average sentiment of N, the mood is decided to be either good or bad. If the mood is good, then one of the articles from the happiest five articles (with the highest sentiment value) from the N set is chosen. Similarly, one of the five most melancholy (lowest sentiment value) is chosen if the average sentiment is bad. This is how the system chooses which article is used to extract keywords for further use in the generation process. One of the measures used by the system to generate text is the distance between the average sentiment of the words in the poem measured against the chosen mood of the day. This way the sentiment extracted from the input (articles) is reflected in the generated poetry.

The generation system of Misztal and Indurkhya (2014) included an emotional personality aspect where both sentiment analysis and emotional modelling was implemented. To be able to extract a sentiment evaluation, the Sentistrength (Thelwall et al., 2010) sentiment analysis tool was used. It estimates the negative and positive sentiment values in short informal texts, rating both positive and negative scores on a 1–5 value scale.

It also considers common and slang words, emoticons and idioms. The base of the algorithm for the system is the *sentiment word-strength list* containing terms with a 2–5 scale of positive or negative evaluation. The initial, manually-prepared words-sentiments list has been optimized by a training algorithm to minimize the classification error for some training texts. The system also considers a spelling correction algorithm, and *booster words list* with terms that can increase or decrease other words' scores (e.g. *very, extremely*) as well as negating word list with terms, that may invert emotion value (e.g. *not, never*). Misztal and Indurkhya (2014) also use the WordNet-Affect lexical resource (Strapparava and Valitutti, 2004) to build a hierarchy of words describing emotional states that are used later to generate the affective content of poems. The lexicon contains WordNet hyponyms of the emotion word, which are a subset of synsets (a set of one or more synonyms) suitable to represent affective concepts correlated with affective words. An example they use is with the word *compassion*, it is possible to derive a correlated set of words describing this state, e.g. *forgive, merciful, affectionate, tender*.

The whole process of the emotion modelling in the system of Misztal and Indurkhya (2014) is done by the emotion expert in the blackboard architecture. It defines the emotional state for the poem, and sets the *emotion* in the blackboard. Since the input text used for emotion may be long, and the emotional attitude may vary within it, sentiment is considered only for the sentences containing the topic of the poem. The sentiment of the text is calculated in terms of *valence*, by using Sentistrength as described earlier. To have the system represent independent emotional intelligence, an *optimism rate* was introduced as parameter to bias the valence, such that the perception of the input text may be regarded more optimistic or pessimistic than the sentiment analysis result. In addition to the valence, an *arousal* value is calculated using Affective Norms for English Words (ANEW) (Bradley and Lang, 1999). An ANEW database consisting of nearly 2500 words rated in terms of pleasure, arousal, and dominance is used. The algorithm combines the average ANEW arousal value for the words in the input text. Since sentiment can be expressed with other features than just words in the text, similar to expressing emotion with voice intonation in a spoken message, the arousal calculation uses a punctuation-sensitive algorithm. Hence, certain types of punctuation marks can increase or decrease the arousal value, like the examples: "*That's great...*" versus "*That's great!!!*". The emotional state for the poem is then calculated in an equation combining valence and arousal.

4 Data set

This chapter covers the data set that has been used. In this instance, this refers to a collection of texts used for training the Long Short-Term Memory network created in this project, which is the key component for the poetry generation process. The network uses this data set as training data, with the goal of trying to mimic the data set as closely as possible. Therefore the contents and size of the data set are important, as this will be reflected in the results of the LSTM network. The data set that have been used in this master project, is the same data that was collected and used by Tikhonov and Yamshchikov (2018b), specifically the part containing English poetry. The data consist of poems written and published by users on a public website, which leads to a variance in the quality of content, but both the large size and variance in content are positive factors for training of the network, including the fact that the texts are written with the intended style of poetry, which is what the LSTM network will be used to generate.

The data set was already cleaned by having removed all types of punctuation and converting all letters to lowercase. The data set differed in how contracted words were present. While some of the contracted words appeared in the data set in a joined form (e.g. *wouldve*), other contracted words appeared as separate words, separated by a space (e.g. *would ve*). Because of this inconsistency, all of the spaces between regular contractions were removed, so that all the regular contractions were represented equally. This has an effect on the later tokenization of the data set text, where the regular contractions are represented by unique tokens. The regular contractions that were irregularly presented and all shortened to an equal form include: *'t*, *'m*, *'s*, *'ll*, *'d*, *'ve*.

*
* was also present in the data set, to represent the line breaks in every poem. This was removed from the data set, and every individual poem was instead represented by single individual lines. The removal was due to the fact that the structure of the generated poems, which includes the length of the individual lines in a poem, is decided by different constraints in the generation process. Training the neural network on generating break line-representations as well as words for poetry generation is of no value, as these break line-representations will not be used. Due to the frequency of *
* appearing in the data set, and the independence it has from the words appearing before and after *
*, including them in the training of the LSTM network could also have a negative effect on the result, as the network could learn to frequently prefer *
* instead of other predicted words.

The original data set contains 3,943,982 poems, with a vocabulary of over 700,000 unique tokens. Most of the unique tokens in this vocabulary come from misspelling, alternative spellings, irregular words and unique words like names. The training data is significantly shortened to specifically reduce the size of the vocabulary. This is done

for two reasons. The first reason is to remove words that very rarely appear in the data set. This includes all the different rare misspellings, but also other rare words with a low frequency of occurrence. Because the purpose of the LSTM network is to give predictions of which words should follow a given sequence, based on the entire vocabulary, a very large vocabulary will have a considerable amount of its contents predicted at 0, given the amount of words that very rarely appears, or even just appears once in the entire data set. These words will therefore be negligible in the generation process, due to their infrequency. This could be avoided by using tools such as *smoothing*, which would give all the possible words a prediction value that is not zero, but this would be counterproductive regarding the second reason; concerning the training of the LSTM network. As the size of the matrix representation of data used in training the network directly relates to the size of the vocabulary, a larger vocabulary increases the size of the training matrices. This affects the training process by demanding more memory, and increasing the training time of the network.

To reduce the vocabulary and the data set, tests were run on different sizes of vocabulary, finding how many of the total poems that only included words within the given vocabulary. By choosing a vocabulary size N, only the most N frequent words found in the data set are deemed as relevant, and poems containing any word outside of this vocabulary were disregarded. The results can be seen in Table 4.1, which shows the different data sets with differing vocabulary sizes, including the size of the original data set. Since larger vocabulary will result in more possible words that can be generated, but will negatively effect training time and be reliant on how much the hardware used for training can handle, a simple test was run on a GeForce GTX 1070 graphics card. Creating a bidirectional Long Short-Term Memory network using Keras with a single hidden layer of 1024 neurons, and training on a random sample size of the 5,000,000 tokens, with different vocabulary sizes, resulted in the GPU component experiencing memory problems when exceeding a vocabulary size of 30,000. The upper limit of vocabulary size was therefore set at 30,000, while the lower limit was set at 10,000, as vocabularies smaller than this will result in a very limited number of different words that can be used for generating poetry, impacting the results.

Table 4.1: Size of data set based on vocabulary

	Data set 1	Data set 2	Data set 3	Data set 4	Original
Vocabulary size:	10 000	15 000	20 000	30 000	708 727
No. of poems:	205 230	306 942	395 057	530 121	3 943 982
No. of tokens:	15 847 356	25 883 608	35 178 076	50 505 342	155 066 504

5 Architecture

This chapter introduces the architecture for the system implemented in this project. The first section describes the long short-term memory network that was implemented, and the second section describes the complete poetry generation process.

5.1 The Long Short-Term Memory (LSTM) network

The LSTM network is a main component in the poetry generation process. After being trained on a large data set of human-written poetry, its task is to give a prediction on the next word that should follow after a given input sequence of words. The prediction consists of an array, with a predicted score of every unique word in the vocabulary. These predictions are then used in the generation process. This section presents the architecture of the LSTM network, and additional processing of the data used for training the network. Several different approaches and details were tried during training, which is described in chapter 6.

5.1.1 Training data

Before training the neural network, a decision had to be taken on which of the data sets and what vocabulary size should be used, and more pre-processing of the data was also necessary. An important aspect of the vocabulary size is the inclusion of unique words with a sentiment value, since they would be generated to add sentiment value to the poetry. Using Vader (section 2.4), the different data sets presented in Table 4.1, excluding the original data set, were all investigated for how many unique words they contained with a sentiment value that was not neutral, i.e. either having a positive or a negative sentiment value. The results are present in Table 5.1. Due to the increasing number of unique sentiment words in the increasing vocabulary sizes, but also due to the memory restrictions and larger vocabularies resulting in greater training times, the vocabulary size for training of the neural network was chosen to be 10 000 and 20 000, and experiments on training with both these data sets are presented in section 6.3.

Table 5.1: Number of unique sentiment words in data sets based on vocabulary size

	Data set 1	Data set 2	Data set 3	Data set 4
Vocabulary size:	10 000	15 000	20 000	30 000
Unique sentiment words:	1 849	2 424	2 900	3 519

After the two data sets for training were chosen, more pre-processing was done. As shown in Table 4.1 the number of individual tokens in the data set with a vocabulary size of 20 000 is over 35 million, which is too large to be able to train on, simply concerning the time it would take. The data sets were therefore reduced to new data sets, here referred to as *training data sets*, which only contain 10% of their original size, respectively. Since the original data sets had all poems ordered after the user name of the person that published it, every tenth poem was selected for creating training data, to get poems from as many different authors as possible in the training data. The size of the training data sets are presented in Table 5.2.

The training of the neural network is done by creating input sequences to be fed through the network, but also creating the correct output which is then compared to the output of the network. The input and output is therefore created by choosing a sequence of the training data with a given length as input, and the token following that sequence as the correct output. For creating training sequences out of the training data, a sequence length of 5 was chosen for several reasons. The first being that longer sequences result in larger matrices that need to be stored in memory for training, and will be limited by memory and greatly increase training time. Therefore the sequence length is limited to the length of 5. The second reason is that the length of 5 is decided to be the shortest possible length of a single line of poetry that will be generated. This is described in section 5.2. It is not desirable to train the network on sequences longer than possibly complete lines of generated poetry. Lastly, an additional idea behind this sequence length of 5 for training is that the network will then predict the next words based on just a short sequence, instead of all of the poem that is already generated. Causing the following predictions to be more independent from the total previously generated sequence, which could cause more variation. The network would then be predicting the following words based on a few given keywords, or on shorter parts of longer sequences.

Table 5.2: The final training data sets

	Trainig Data set 1	Training Data set 2
Vocabulary size:	9195	18 031
No. of poems:	12 273	26 873
No. of tokens:	1 300 068	3 106 347

The last step of pre-processing the data for training, is creating the training input and output sequences, and transforming them into matrices that can be fed into the neural network. Each unique token found in the data set is collected, and sorted based on the number of appearances in the data set. The most common token appearing in the training data set is the first item in the vocabulary, with index 0, the next most common with index 1, and so on. This provides every unique token with a unique ID, which corresponds to the index of the token in the vocabulary. Before creating the input, every poem is reversed — with the last word being the first, and so on. This approach was used by Benhart et al. (2018), where both the training of the network, and hence the generation of poetry, is on text that is backwards. This is done so it is possible to

start with the ending rhyme word of a line of poetry, and generate the rest of the line backwards from that rhyme-word. Instead of having to find tail rhyme words that will fit well into an already generated sequence of words, the system needs to find words that fit with the given rhyme words, which results in more possibilities to generate good sequences.

After creating the input and output sequences for the training data sets, these sequences are transformed into matrices representing the input and output. Since each unique token is represented by a unique index within the size of the vocabulary, the matrices created have a y-length dimension equal to the size of the vocabulary. Tokens are represented by having their index in this matrix dimension as 1, and all the other values set to 0. For the input training data, the x-dimension of the matrices is 5, corresponding to the sequence length of tokens. The output tokens used in training have an x-dimension of just 1, representing the single token that is the correct token that appears after the input sequence.

After creating the training matrices for a training data set, 10% of the matrices are randomly chosen to be used as validation data during training, and not used for training. The purpose of the validation data is to evaluate the LSTM performance during the training process. Continuously measuring performance during training against validation data that the network has not seen or trained on, gives an indication on whether the LSTM model is overfitting on the specific training set.

Because the input matrices created from the training and validation data demand a lot of memory, it is not possible to create and hold all the input matrices in memory before starting the training of the network. The input matrices are therefore being continuously created and fed to the network during the training process, by a generator method.

5.1.2 Architecture of the network model

The network used in this project is a bi-directional long short-term memory neural network. The reason for choosing an artificial neural network is the prominent use of these in state-of-the-art systems for poetry generation, as presented in section 3.2. Recurrent neural networks, a sub-type of ANNs, make use of sequential information. Because a sentence or a line of text can be regarded as a sequence, this is relevant for text generation. RNNs possess a memory of previous input calculations in the networks' internal state, which makes them suitable for natural language processing. The reason for choosing a LSTM as a type of RNN, is because LSTMs are able to work with much longer sequences than general RNNs, meaning they can utilize cases that are somehow related, but with a larger sequence distance between the cases than a general RNN could handle. The details behind both RNNs and LSTM specifically are described in ection 2.2. A bi-directional LSTM means that it contains two networks, where one can access past information in a forward direction, and another access future information in the reverse direction. By preserving information from both these networks, and using these two hidden states combined, you are able to preserve information from both the past and the future. This results in the LSTM understanding the context better, because using both information of what comes before a word, and after the word, makes it easier for

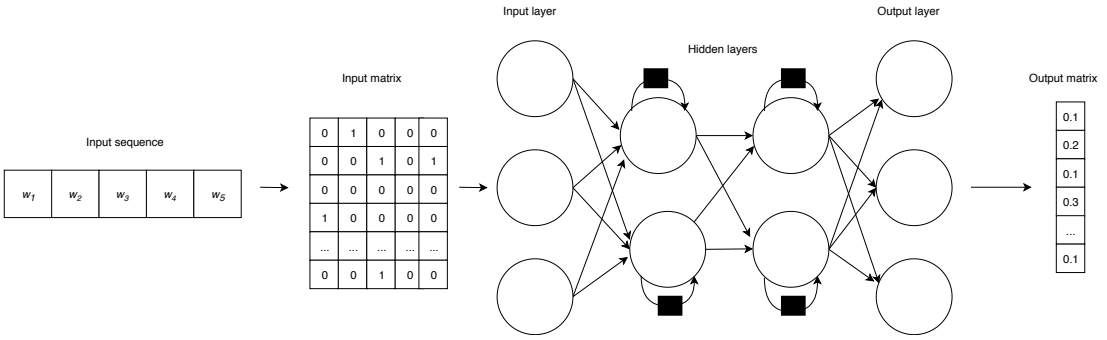


Figure 5.1: Long Short-Term Memory network prediction process

the network to understand what the next is going to be.

The neural network in this project consists of several layers, which include the input and output layers, and also the hidden layers. The input layer represents the matrix data given into the network, which accounts for both the data used for training the network, and for the input during the poetry generation process, where the network provides predictions based on this input. The output layer is the last layer, which is a softmax activation function layer that provides the predictions on the given input. During training, the result of these predictions will be compared to the actual following word of a given sequence, which will update the hidden layer(s) based on a loss function, which the network tries to minimize. The hidden layers are the layers in between the input and output layers, and consist of LSTM cells. Figure 5.1 shows LSTM network, with the input and output layers, as well as the hidden layers containing the LSTM cells. It is the LSTM cells that compute the possible values for the next predictions. Details about the LSTM cells and their calculation process are given in section 2.2.5. The figure also show the input sequence with a sequence length of 5, that is transformed into a matrix representation of the input sequence before being fed into the LSTM network. The output from the last layer is the matrix representing the network predictions for all possible words.

The loss function that the network tries to minimize during training is the cross entropy loss. The formula for the loss is shown in equation 5.1, where N is the total training set, p is the prediction and c is the category (the word) being looked at. The loss is the cross entropy between the distribution of the true labels and the predictions from the network. The network therefore tries to minimize this loss, to reward a precise prediction equal to the true labels.

$$L = -\frac{1}{N}\sum_{c=1}^{N} ln(p_c) \tag{5.1}$$

To minimize the loss function, the weights of the network are continuously being updated throughout the training to produce predictions with a minimal loss value. This is done by using backpropagation, which takes the error found by the loss function L while

training, and calculates the gradient of the loss function with respect to the weights, w, in the network, $\frac{\partial L}{\partial w}$. The gradient is fed to the optimizer, which updates the weights in an attempt to minimize the loss function. The optimizer used for this model is the stochastic gradient descent optimizer, which is a first-order iterative optimization algorithm. The optimizer updates the weights proportional to the negative of the gradient calculated by backpropagation. The use of stochastic here means that random samples from the training data are chosen in each run to update parameters during the optimization, within the same framework as gradient descent. This results in the error being computed and the weights being updated in faster iterations, because only a small selection of samples are processed in one go. This often helps to move towards an optimum more quickly, the downside being that the path to the optimum can be much noisier. This means larger variations in loss between each epoch, because the mean error is computed over a stochastically selected subset from the dataset in each iteration. The learning rate directly influences the degree of how much the weights are updated by the optimizer, and is therefore used to avoid a local minimum. It is possible to get stuck in a local minimum when minimizing the loss function, therefore the learning rate is initially set higher, before decreasing during the training process to try to find the global minimum.

An additional technique used during the training of the network, is dropout (Srivastava et al., 2014). Dropout randomly selects a number of the neurons in the network that will be ignored, meaning they will not contribute to any calculations while the input data is passed through the network, or any of the weight updates in backpropagation. The theory behind this is to prevent the nodes from relying too much on the nearest weights during the training process, and instead forcing the neurons to generalize more individually on the training set. In addition it reduces the training time, by reducing the amount of calculations for each run-through of training input. Srivastava et al. (2014) show that dropout both reduces overfitting and gives improvements over other regularization methods.

While the general architecture and theory of the LSTM have been described, there are many possibilties for all the different parameters, variables and functions used in a LSTM model. Experiments were therefore conducted by training LSTM networks with different architectures, with the goal of finding the best parameters and variables of a network, resulting in better prediction results. The training process and the final architectures of the networks used in the experiments are described in chapter 6.

5.2 Poetry generation system

In this project, a complete system for generating poetry was created. The LSTM network described in the previous section is a main component of the complete generation system. The predictions generated by the network consist of an array containing the predicted value for each word in the vocabulary, to follow the input word sequence fed to the network. Figure 5.2 (shown on page 40) shows an example of the prediction vector for a vocabulary of size n. These predictions scores are used in the poetry generation process for selecting the words that will be generated. In addition to the LSTM network, there

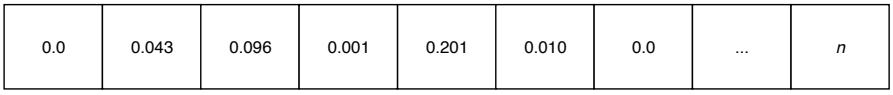


Figure 5.2: Prediction vector generated by the LSTM

are three other important components in the poetry generation system: The generation of rhyme pairs, the algorithms for updating prediction scores, and the search tree algorithm. The generation of rhyme pairs is used as the initial input for generating each line of poetry, and also ensures that the generated poetry contains end rhymes. The algorithm for updating the score values adjust and update the prediction values gained from the LSTM network, by adding rules and different weightings on certain types of possible words, to enhance the quality of the sequences being generated, based on different criteria. The search tree algorithm expands the number of possible sequences that are created during the generation process, increasing the chances of finding the best possible sequences to create a poem from. This section presents the complete generation process, and explains the generation system in detail, including the tree important components.

Rhyme word generation: The first part of the generation process consist of the algorithm for generating rhyme pairs of words, which will be used to generate the rest of the poem. Each of the rhyme pair words will be used as input for generating a line of poetry, as the poetry is generated backwards from the ending rhyme words (described earlier in this section). For generating rhyme pairs, unique words are randomly chosen from the vocabulary, that have a sentiment value matching the decided sentiment. Vader is used for this purpose, where words with a sentiment value of more than 0.0 are used if the sentiment is decided to be positive, and less than 0.0 for negative. After the sentiment word is chosen, the vocabulary is iterated over to find another word of the same sentiment, and that rhymes with the first word. To find rhyming words, CMUdict is used (see section 2.4) to find the syllables for each word. The conditions that need to be met to complete a rhyme, do not form a *perfect* rhyme. This would be when the stressed vowel sound in both words, as well as any subsequent sounds, are identical, while the consonant preceding the stressed vowel sound is different. In this case, a form of *imperfect* rhyme is rather used, where the last three syllables, including a consonant, are equal for two words. Each of the two words in a rhyme pair make up one line of poetry, and the rhyme form chosen to generate poetry is $ABAB$, so two pairs are needed to generate four lines of poetry.

After the rhyme pairs are found, the first rhyme word is used to predict the following words for the first line of poetry that will be generated. This first line will eventually end up being the final line of the complete poem, since the poem is generated backwards, so the entire poem will be reversed after it is generated. When the first line is generated, the last four words of that sequence, plus the rhyme word for the next line, are used as input for generating the the next line. This process is presented in Figure 5.3 (shown on page 41), where the first rhyme word A1 is used to generate the first line of poetry, consisting of the rhyme word and a sequence of words w to the length of n. The next rhyme word B1 is then added on to the sequence of the four words from w_{n-3} to w_n from

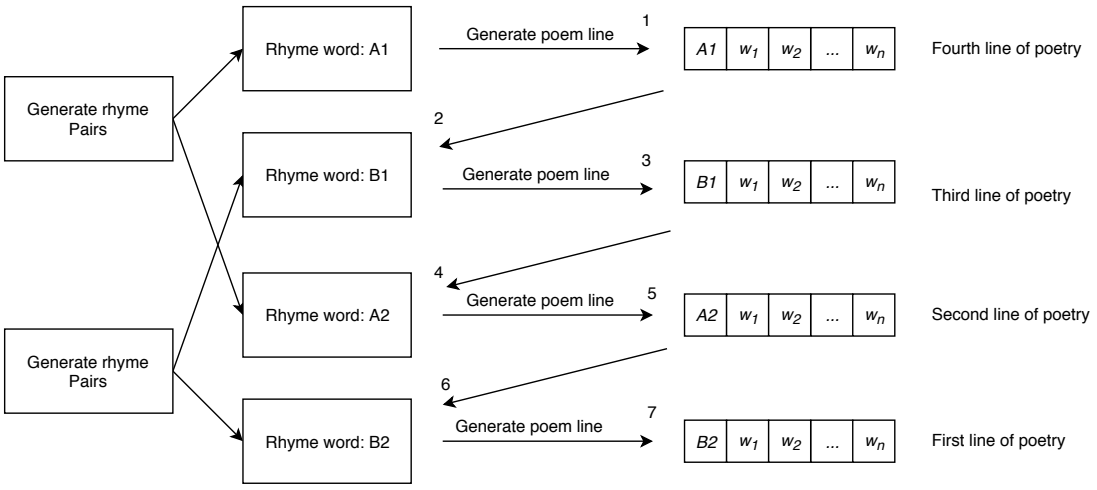


Figure 5.3: A figure of how the poetry is generated by using rhyme words as the initial input

the last sequence, to be used as input for generating the next line of poetry. The figure shows this process being repeated until a full verse consisting of four lines of poetry with a rhyming form of ABAB is completed. Since the poem is generated backwards and will be reversed after being generated, the figure shows that the first line that was generated, with the rhyme word A1, will end up being the last line of the complete poem, and so on. The last generated line with the rhyme word B2, is the first line of the complete poem.

A distinct meter or form of the generated poetry is not implemented, as very strict conditions limit the possible results in generating text, and can result in a reduction of the quality of the result just to be able to uphold a certain form or structure. As the main goal of this project (section 1.2) is to generate poetry with an inherent sentiment, the form of the generated poetry is neglected, in favour of generating text with a focus on the sentiment value of the text. This will likely reduce the quality of the *poeticness* of the poetry. In addition, rhyming constraints do not follow the rules that result in a *perfect* rhyme, and the sequence length of each line of poetry is set to 5-10, strictly to remove the generic repetitiveness of a fixed length, and instead have it randomly mimic a free-flow format.

Updating prediction scores: For every word that is generated in a sequence, predictions on all the possible words are generated by the LSTM network, based on input to the network consisting of previously generated words. But instead of deciding the next word to be generated based solely on the predictions from the network, several algorithms are implemented for updating and adjusting the prediction scores from the network, based on factors chosen to improve the generated poetry. These algorithms were similarly used in several other state-of-the-art-solutions (chapter 3), including Benhart et al. (2018) implementing repetition and sentence structure restrictions, and Colton et al. (2012) measuring sentiment values of poetry lines against a set value. Figure 5.4 (shown on page 42) illustrates the process of generating word predictions for a given

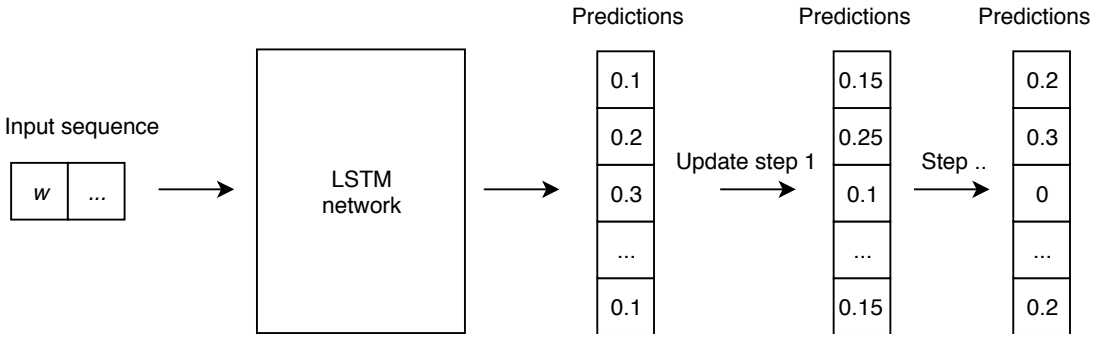


Figure 5.4: A model showing word predictions being updated through several steps

input sequence, and updating the prediction scores over several steps, starting with step 1. Each step involves a separate update algorithm. There in total four steps of specific update algorithms that are implemented in this system for adjusting the prediction scores, and these steps include:

- Related words (encouraging the use of related, less frequent words).
- Repetition (adjusting based on repetitiveness).
- Sentence Structure (removing possible sequences based on sentence structure).
- Sentiment (adjusting based on the sentiment value of the sequence).

Related words: Because the same words often appear in the original data set, and therefore the training data that the network is trained on, these popular words will have a tendency to have a high prediction value. To avoid a repetitive use of the same words, or an unfairly high prediction value for certain words, related words are found and used to increase variety. This is done using WordNet (section 2.4), where the Synsets (groupings of synonymous words that express the same concept) of the original word provide related words found in WordNet. When generating the next word in a sequence, related words are found for the 20 unique words with the highest prediction value. Then, for all the related words that exist in the vocabulary, the prediction values for these related words are increased, thus increasing the probability of choosing a related, but less used word, to achieve a better variety.

Repetition: This step is implemented to reduce the likelihood of a line of poetry to contain repeated words, and therefore reducing repetitiveness. As discussed in the previous step, popular words can frequently occur, and by decreasing the chance of the same word appearing several times in a line, the use of more unique words is encouraged. The implementation involves reducing the predicted score of a word during sequence generation, if the word has already appeared in the previous part of the sequence.

Sentence Structure: By observing that the generated poetry consistently had obvious part-of-speech (PoS) errors, restrictions were implemented to avoid the errors that could

easily be detected by a human. Sequences with obvious errors were collected, and using NLTK (see section 2.4), PoS-tagged. These were then used to implement general PoS-restrictions, to avoid word sequences violating these general restrictions. Restrictions that were implemented included no pronoun directly preceding another pronoun, for example "he it", and no verb directly preceding another verb.

Sentiment: Based on the intended sentiment for the poems, the score value for the possible words with a corresponding sentiment is increased. This is done by using Vader (see section 2.4) to find all the possible next words in a sequence that have a sentiment value corresponding to the intended sentiment (positive or negative), and increasing their predicted score, resulting in an increased chance of choosing words with the correct sentiment value when generating sequences. The degree of increasing predicted scores is based of the sentiment value of each word. Words that Vader evaluate as having a higher sentiment value of the correct sentiment (positive or negative) will have a greater increase of their predicted score, compared to the words with a lower sentiment value. In addition to increasing the predicted scores of words with the correct sentiment value, the words with the opposite sentiment value get their predicted scores decreased in the same way. For example: to generate a poem with a negative sentiment value, the words with a negative sentiment value have an increased chance of being chosen during the generation process, while the words with a positive sentiment value will have a decreased chance.

Search tree algorithm: To increase the chances of finding the best possible sequences to create a poem from, a tree search algorithm is implemented to expand the number of possible sequences that are created during the generation process. Instead of generating a single next word based on the highest score, the tree search takes a number of the possible words with the highest score values, and generates different possible sequences. This is repeated for every new word in the sequence, resulting in more possible sequences for every new word to be generated. A generalized approach of the search tree process is presented in Figure 5.5 (shown on page 44) to refer to what page it is talking about). The first word w_1 is in this case one of the rhyme words used to generate the rest of the sequence, but it could also be a sequence of previously generated words plus the new rhyme word. Using this as the input to the LSTM network, we get the predictions (score values) for the next words. These values are then updated through the different algorithms described earlier, before 20 new sequences is created, which all consist of the first word w_1 plus one of the next words x with one of the highest score values, where every sequence has a unique next word x. This entire step is then performed on all the possible sequences created in the previous step, for adding the third word in the sequence, and so on. This search tree continues with a number of steps i which is equal to the total number of words to be generated in a given sequence. As the search tree grows exponentially, the size of the network can drastically increase the time it takes to complete the search algorithm, and require a large amount of memory. To combat this, the possible sequences that are created with the lowest *scores* are continuously pruned throughout the search process, by removing a major part of the possible sequences, and only keeping the ones with the best score so far in the search process. The value *score* that is attributed to each possible sequence, is based on the prediction score of every

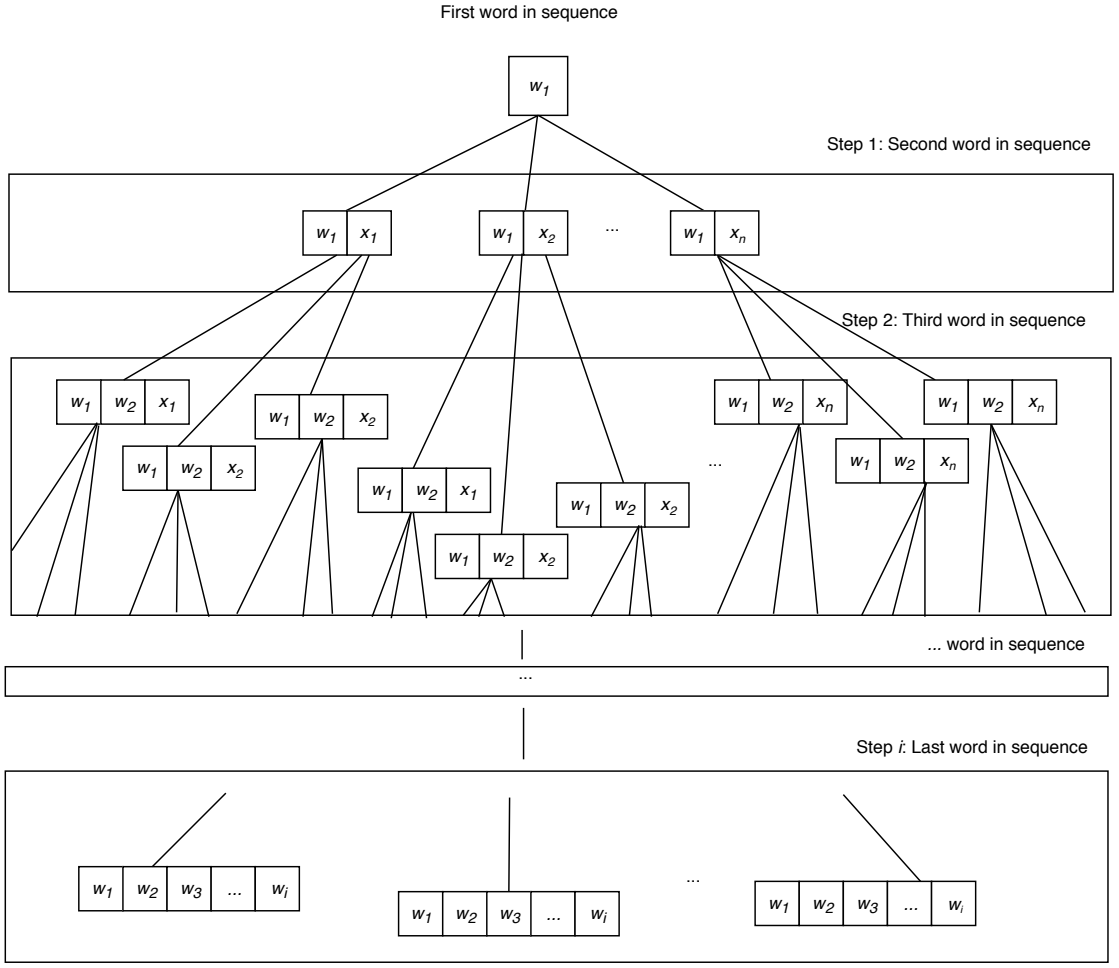


Figure 5.5: A general example of the search tree process for generating sequences of words

word that is added to the sequence. This value score is updated every time an additional word is generated and added to the sequence. Figure 5.6 (shown on page 45) illustrate how the score of a given sequence is based on the prediction scores of the words it consists of.

When all the possible sequences are created, the sentiment value is again evaluated, this time for each of the possible sequences. This time the sentiment value is based on the complete sequence, where Vader considers the entire line of poetry when calculating a sentiment value. These sentiment values are again used to adjust the score for all the possible sequences, where the final scores are increased by a set variable that is multiplied with the sequence's sentiment value, thus increasing the score with an amount related to the sentiment value (depending on the decided sentiment). After this final score update, the sequence with the highest score is chosen, and used to generate a new line of the current poem.

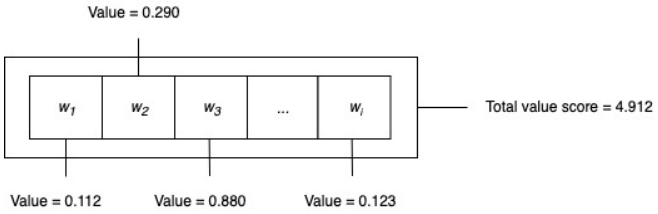


Figure 5.6: A general example of the value scores for a generated word sequence

The entire generation process described in this section is used to generate poems consisting of eight lines, which make up poems of two four-line stanzas with a rhyming scheme of *ABAB CDCD*. The chosen sentiment is decided before the generation of the rhyme words, and the following generation of poetry lines. After all the lines have been generated, the system adds commas after each of the first three lines of a stanza, while a period is added after the last line of a stanza. A blank space is also added between the two stanzas. This presents the end of the first stanza and the beginning of the next stanza. The first letter of the first word of every line is capitalized, in addition to other letters where capitalizing is grammatically correct, the most frequent example being the standalone "i" converted to "I". This process results in a complete poem, at which point Vader is used to calculate the sentiment value of the entire complete poem. This final sentiment value is used in the experiments where it is compared to the human judge evaluations (see section 6.2).

6 Experiments and Results

This chapter presents the experiments conducted. This includes the experimental plan and reason behind the experiments, the experimental setup, and lastly the results of the experiments.

6.1 Experimental Plan

Experiments in two different areas were conducted as part of this thesis work. In the first area, an experiment was conducted on training of the LSTM network (described in chapter 5). The goal was to measure how well differently trained networks — both regarding different data sets, and different parameters and network sizes — were able to predict words. The is needed because we wanted to train a network that performed as well as possible (at predicting words in a given data set), and to then use this network in the further poetry generation process, as stated in research question 2 (see section 1.2).

In the second area, experiments were conducted to evaluate the generated poetry. One of the experiments used standard evaluation metrics for generated poetry, to be able to compare the results with other poetry generation systems. A second experiment was conducted to evaluate the sentiment value of the generated poetry, in relation to research question 3 (see section 1.2).

6.2 Experimental Setup

This section presents the experimental setup for all the experiments that were conducted. There are two parts, the first presenting the experimental setup for the LSTM training, and the second part presenting the experimental setup for evaluating the generated poetry.

6.2.1 Training and evaluating Long Short-Term Memory networks

The first experiment was conducted to decide the final architecture of the LSTM network, and what parameters to use in the final network implementation of the poetry generation system. Most parameters and architecture details were decided and set beforehand, including the data set to be used for training. Additional details previously decided are described in chapter 5, including the optimizer function, the activation function and the loss function. Other predetermined factors were:

- The two data sets to be used in training

- Learning rate and learning reduction
- Dropout probability
- Validation data steps and size

The creation and processing of the data sets used for training the LSTM network is explained in chapter 4. Two different data sets having different vocabulary sizes and different amounts of total tokens, were used in training different networks. Conducting experiments on these two different data sets were done for two reasons, the first being that a reduction in size and vocabulary size greatly decreases the training time which makes it possible to conduct training experiments in less time. The second is to investigate if there would be a significant difference in performance between the networks trained on the different data sets.

As described in chapter 5, learning rate for the training of each network starts at a higher value to avoid local minima, and decreases throughout the training process. The learning rate is initially set to 0.9 for the training of all networks. A monitor is implemented on the validation loss that is calculated on validation samples each epoch, which reduces the learning rate after a given period when the validation loss has not decreased. The monitor period before reducing the learning rate is set to 5 epochs, and the reduction of the learning rate is set to 0.3, meaning that the current learning rate is multiplied by 0.3 for each reduction. The minimum limit for the learning rate is set to 0.001, so the learning rate cannot be reduced to any value lower than that.

The dropout probability rate is set at a constant 0.6 for non-recurrent units. The dropout rate here represents the fraction of the input units to drop. This value was chosen based on Zaremba et al. (2014), where the dropout rate used for medium LSTM (650 units per layer) was 0.5, and the dropout rate for large LSTM (1500 units per layer) was 0.65. A constant rate for all networks trained in this project was therefore set to 0.6, as the units per layer vary, with most having 512 or 1024 units per layer.

The hardware that was used for training the networks was a Tesla P100-PCIE-16GB GPU, with a memory clock rate of 1.3285GHz. Since the training was done on an open university computer with several active users at the same time, the amount of memory available to use during training varied between sessions.

The different parameters tested in the experiments were:

- Two different data sets
- The number of hidden layers
- The number of units in each hidden layer
- The number of epochs

The number of hidden layers and the size of each of the hidden layers were also tested. The size of the hidden layers represents how many LSTM units each hidden layer consist of. In addition, the number of epochs were tested. A single epoch of training is when

all of the training data has been fed through the network during the training process, one time. While the vocabulary size of the training data, and the size of the training data itself, both impact training time, the number of hidden layers, size of the hidden layers and number of epochs also affect training time. The possibilities for the number of hidden layers and size of hidden layers are therefore limited, and as shown in the experimental results (section 6.3), the maximum number of hidden layers tested was three, and the maximum number of units per layer was 1024. The training time of the different networks are also presented in the results.

To be able to evaluate the trained networks in the experiment, new *evaluation* data was created from the original data sets. The evaluation data was created in the same way as the previous training data (see Section 5.1.1), with the only difference being that it was created from data that the network had not previously seen (been trained on). In other words, created from poems in the original data set that had not been used in the final training data sets and that only contained tokens included in the chosen vocabularies for the training data set. The final evaluation data sets (1 & 2) are represented in Table 6.1.

Table 6.1: The final Evaluation Data sets

	Evaluation Data set 1	Evaluation Data set 2
Vocabulary size:	9 195	18 031
No. of poems:	4 620	4 307
No. of tokens:	405 286	406 324

The LSTM models trained on Training Data set 1 (which consists of a vocabulary size of 9195) was only evaluated on the Evaluation Data set 1 in the experiment, since it consists of the same vocabulary size. The LSTM network models trained on the Training Data set 2 on the other hand (consisting of a vocabulary size of 18031), was evaluated on both Evaluation Data set 1 and 2 in the experiment. The reason was to be able to compare LSTM models trained on different vocabulary sizes, both based on the exact same evaluation data and based on evaluation data with a vocabulary size corresponding to networks vocabulary size.

To evaluate the trained networks, evaluation data set input was fed through the network, and predictions were measured against true values, using several measurements: categorical accuracy, categorical cross entropy and the word perplexity. Categorical accuracy is calculated by checking if the index of the true value (in the vector representing the vocabulary, where each index represents each word in the vocabulary) is equal to the index of the maximal predicted value. The prediction vector generated by the network, gives a prediction value for each index, where each index represent each word in the vocabulary. The maximal predicted value index will be the word that the network gives the highest prediction result, of all the possible words. The categorical accuracy is calculated for the entire evaluation data set, by taking the mean accuracy rate across all the predictions. Categorical cross entropy calculates the cross entropy value, which is the loss function used during the network training, and is described in Section 5.1.2. The word perplexity is a measurement of how well a probability distribution can predict

a sample, and is calculated by using the loss function as exponent to the power of the constant e, as shown in Equation 6.2.1. The lower the perplexity, the less confused a network is about predicting the next word.

$$perplexity = e^{cross\ entropy\ loss} = e^{-\frac{1}{N}\sum_{c=1}^{N} ln(p_c)}$$

The final results from evaluating the trained LSTM network models are shown in section 6.3.

6.2.2 Evaluating the generated poetry

Two different experiments were conducted on the poetry generated by the final system. In the first experiment, human judges evaluated the poetry based on criteria presented in section 2.5, including standard evaluation metrics for generated poetry. In the second experiment, human judges evaluated the poetry based on its sentiment value, i.e. whether they perceived the poem to have a sentiment value, and whether it is negative, positive or neutral. They also rated the degree of the sentiment value.

Standard evaluation metrics

The goal of the first experiment was to evaluate the generated poetry on the three criteria presented in Section 2.5. In Manurung (2004), these criteria are rated along with three different dimensions:

- Grammaticality
    1. Not correct
    2. Partially correct
    3. Grammatically correct
- Meaningfulness
    1. Not meaningful
    2. Partially meaningful
    3. Meaningful
- Poeticness
    1. Not poetic
    2. Partially poetic
    3. Poetic

These dimensions are therefore applied for this experiment, where the human judges are asked to score each of the criteria from 1—3 on the poetry that is evaluated, 1 being the lowest score and 3 being the highest. The result of the evaluations are presented in section 6.3.

The motivation behind this experiment was to to evaluate the poetry generation system by evaluating the poetry that is generated. This makes it possible to assess the result, both individually and compared to other systems using the same metrics.

Sentiment evaluation

The goal of the second experiment was to evaluate the generated poetry based on sentiment value, to try and answer research question 3 (section 1.2): *Will the generated poetry be perceived by human judges to contain a sentiment value, and does this value correspond to the value the poetry was intended to have?* Human judges will evaluate the generated poetry with three main categories:

- Negative sentiment value
- No sentiment value (neutral)
- Positive sentiment value

If a poem is evaluated as having negative or positive sentiment value, the sentiment value will be graded with a score of 1-3, 1 being the lowest and 3 the highest. This value corresponds to the degree of sentiment value, with the three different scores representing:

1. Slightly negative/positive
2. Quite negative/positive
3. Very negative/positive

The reason for rating the degree of sentiment is to investigate if the *degree* of calculated sentiment value of the poetry by Vader (see section 2.4) is also perceived similarly by the human judges. In other words, if Vader scores a poem with having a high degree of sentiment, will it also be rated as *very* negative/positive by the human judges? The results of the experiment are presented in Section 6.3.

To conduct the experiments on the generated poetry, a total of 20 generated poems was used, which consists of a total of 40 stanzas, made up by 160 lines of poetry. All of the 20 poems was used in both the experiment for evaluating the poetry based on standard evaluation metrics, and for evaluating the sentiment of the generated poetry (see Section 6.2). Of the 20 poems, 10 are generated to contain a positive sentiment value, and the other 10 to contain a negative sentiment value.

6.3 Experimental Results

This section presents the results from the experiments that was conducted. This includes the experiments on the LSTM training, and the experiments conducted on the poetry generated by the system.

6.3.1 Training of the Long Short-Term Memory network

Table 6.2 shows the different architecture details of the LSTM networks trained on Training Data set 1 (containing a vocabulary size of 9195). Table 6.3 shows the different architecture details of the LSTM networks trained on Training Data set 2 (containing a vocabulary size of 18031). The different aspects of the networks presented in the tables are the number of hidden layers in the network, the number of hidden units per layer, the amount of epochs that the network was trained for, and the training time per epoch (in seconds).

	LSTM 1.1	LSTM 1.2	LSTM 1.3	LSTM 1.4	LSTM 1.5
Hidden Layers	2	2	3	3	3
Hidden Units	256	1024	256	512	1024
Epochs	50	50	50	50	50
Training Time	700s	1500s	1000s	1200s	1800s

Table 6.2: The different LSTM networks trained on Training Data set 1, named from 1.1-1.5

	LSTM 2.1	LSTM 2.2	LSTM 2.3	LSTM 2.4	LSTM 2.5
Hidden Layers	2	2	3	3	3
Hidden Units	256	1024	256	512	512
Epochs	75	50	50	25	50
Training Time	3000s	5900s	3500s	6600s	6600s

Table 6.3: The different LSTM networks trained on Training Data set 2, named from 2.1-2.5

The results of the LSTM network evaluation experiments are presented in Table 6.4 and 6.5, appearing on page 52 & 53. Table 6.4 presents the cross entropy loss, categorical accuracy and perplexity for LSTM models 1.1-1.5, which were only evaluated on Evaluation Data set 1. Table 6.5 presents the cross entropy loss, categorical accuracy and perplexity for LSTM models 2.1-2.5, which were evaluated on both Evaluation Data set 1 and 2. The Evaluation Data sets are explained and presented in Section 6.2.

Evaluating LSTM 1.* Networks					
Evaluation Data set 1 results:					
	LSTM 1.1	LSTM 1.2	LSTM 1.3	LSTM 1.4	LSTM 1.5
Cross entropy loss	8.997	8.097	7.951	8.131	8.813
Categorical accuracy	0.0225	0.0238	0.0246	0.0241	0.0233
Perplexity	**8077**	**3286**	**2839**	**3400**	**6722**

Table 6.4: Evaluation results of LSTM networks trained on Training Data set 1

Evaluating LSTM 2.* Networks					
Evaluation Data set 1 results:					
	LSTM 2.1	LSTM 2.2	LSTM 2.3	LSTM 2.4	LSTM 2.5
Cross entropy loss	7.986	8.617	8.000	8.315	9.729
Categorical accuracy	0.0253	0.0238	0.0251	0.0241	0.0225
Perplexity	**2939**	**5522**	**2981**	**4084**	**16 789**
Evaluation Data set 2 results:					
	LSTM 2.1	LSTM 2.2	LSTM 2.3	LSTM 2.4	LSTM 2.5
Cross entropy loss	8.266	8.893	8.617	8.540	9.983
Categorical accuracy	0.0234	0.0224	0.0238	0.0224	0.0207
Perplexity	**3888**	**7279**	**5522**	**5117**	**21 653**

Table 6.5: Evaluation results of LSTM networks trained on Training Data set 2

6.3.2 Results from Evaluation of the Generated Poetry

Regarding the experiments on evaluating the generated poetry, there was a total of 30 human judges that participated, with each human judge participating in both evaluation experiments: evaluating the standard metrics and evaluating the sentiment. Each participant evaluated a selection of 6 or 7 poems out of the total 20, with a nearly equal amount of both positive and negative sentiments in each selection.

All of the generated poems used in the experiment are presented in the Appendix (see appendix 1 and 2), including the experimental results for each individual poem. The first score matrix presented for each poem is the average score of the three evaluation metrics (presented in Section 2.5) grammaticality, meaningfulness and poeticness, respectively. The second score matrix presented for each poem is the percentage score of human judges percieving the poem to have either positive, neutral, or negative sentiment value. Poem 1-10 are generated poems intended to have a negative sentiment value, and poem 11-20 are generated poems tintended to have a positive sentiment value.

Standard evaluation metrics results

Regarding the three metrics grammaticality, meaningfulness and poeticness, the experiment results for these are presented in Table 6.6. The values in the first column of the table are the average mean values from the experiment, while the second column contains the standard deviation values for each of the metrics. Some examples of the highest- and lowest scoring generated poems are presented and discussed in the next chapter (see 7), including the individual evaluation results for each example.

Sentiment evaluation results

This section presents results from the sentiment evaluation. Results from the first experiment, where the human judges classified poems based on what sentiment value they perceived, are shown in Table 6.7 with average percent of the human judges who rated

	Average Mean	Standard Deviation
Grammaticality	1.488	0.0476
Meaningfulness	1.582	0.0338
Poeticness	2.012	0.0342

Table 6.6: Results from the standard metrics evaluation

the poems to contain either positive, neutral or negative sentiment. Results are shown both for the poems generated with a positive sentiment intention and with a negative.

	Rated positive	Rated neutral	Rated negative
Positive sentiment poems:	58.8%	36.2%	5.0%
Negative sentiment poems:	2.7%	27.6%	69.7%

Table 6.7: Results from the sentiment evaluation experiment

In addition to the classification of sentiment value, human judges rated the degree of perceived sentiment for each poem based on a rating from 1-3 (if they had evaluated a poem to contain a positive or negative sentiment). The poems included in the experiments were also rated using Vader, which assigns a score for the sentiment value of the poem. A negative sentiment value of the input text will be rated between -1 and 1, where -1 represents highest degree of negative sentiment value, 0 represents neutral sentiment value, and 1 represents the highest degree of positive sentiment value . Table 6.8 shows both the average sentiment degree score rated by the human judges and rated by Vader. Human judge ratings for every poem are normalized to values between 0-1, equal to the Vader score, before calculating the average.

	Positive sentiment poems	Negative sentiment poems
Average Vader rating	0.977	-0.925
Average human judge rating	0.360	-0.372

Table 6.8: Results from evaluation of the degree of sentiment

7 Evaluation and Discussion

This chapter contains two sections. The first section evaluates and discusses the experimental results and the poetry generated by the system. The second section discusses the results and findings in this thesis related to the goals and research questions presented in chapter 1.

7.1 Evaluation

This section first evaluates and discusses the results from the LSTM model training experiments. Then some of the poems generated by the system are presented, and the features of the poetry is discussed. Afterwards the results from the poetry evaluation experiments are presented and discussed. Some of the findings discussed in this section are also compared to related work in the field.

7.1.1 Evaluating the Long Short-Term Memory networks

As shown in the results (Table 6.4 and 6.5), the difference in perplexity results vary greatly for the different LSTM networks (presented in Table 6.2 and 6.3).

As the LSTM networks that are only trained on a smaller vocabulary (LSTM 1.1-1.5, see Table 6.2) do not have significantly better perplexity scores than the best scoring networks trained on the larger vocabulary (LSTM 2.1-2.5, see Table 6.3), the networks trained on the smaller vocabulary are disregarded. In addition to the the perplexity scores showing that the smaller vocabulary networks do not perform significantly better, a smaller vocabulary also means that these networks will generate poems with fewer possible unique words. This can reduce the variation of the poetry, and the amount of possible unique words with a sentiment value, resulting in less possibilities for the poem to generate poetry with a given sentiment value. In addition, Training Data set 1 only has a third of the amount of training data compared to Training Data set 2 (see Table 5.2). This means that the networks trained on the larger vocabulary have been trained on more data per epoch.

The possibilities for which network to implement in the generation system is then greatly reduced, with the network with the best perplexity score being LSTM 2.1, with 2 hidden layers and 256 hidden units per layer, achieving a perplexity score of 3888 for Evaluation Data set 2. By running a simple test of generating some text solely based on predictions given by LSTM 2.1, this network shows clear signs of being highly overfitted, by repeating a few select and often occurring words. This still results in a low perplexity score compared to the other networks, as the network is less confused about predicting

which words follow an input sequence, but an overfitted network is not desirable. Since this network does not achieve perplexity scores that are significantly better than other alternatives, the larger network with the next-best perplexity score is chosen instead, which is LSTM 2.4. It has three hidden layers, 512 hidden units per layer, and has been trained for 25 epochs, achieving a perplexity of 5117 for Evaluation Data set 2. The network LSTM 2.5 as the same architecture details as LSTM 2.4, but was trained for an additional 25 epochs, resulting in a much higher word perplexity score (21 653 compared to 5 117). This was most likely due to the network underfitting the training data during the training of the additional 25 epochs, as further training resulted in the LSTM 2.5 having a much harder time predicting correct words. Underfitting occurs when the model fails to learn the underlying relationship between the input and output data, by ignoring the lessons from the training data.

While the perplexity scores differed greatly for all the LSTM networks, the perplexity scores were all still very high, representing poor results from the network training. Zaremba et al. (2014) achieved a word perplexity score of 78.4 with a regularized LSTM where dropout was used. For their model the Penn Tree Bank (PTB) data set (Marcus et al., 1993) was used for training and testing word perplexity, which consists of a vocabulary of 10k words. The LSTM that was used consisted of two hidden layers with 1500 hidden units each, trained for 55 epochs with a 65% dropout probability on the non-recurrent connections. A perplexity score of 2839 (the lowest perplexity score achieved in this project) is very high compared to the 78.4 achieved by Zaremba et al. (2014). While there are several notable differences that will have an impact on the results, the vocabulary size of the data that was trained on in both cases does not differ. Word perplexity will be greater for data with a larger vocabulary size (plainly due to the word possibilities increasing, clearly shown in Table 6.5, where the networks yield lower perplexity scores when tested on evaluation data with a smaller vocabulary compared to the evaluation data with a larger vocabulary), but the perplexity scores achieved in this project are significantly higher compared to the scores of Zaremba et al. (2014), when networks were trained on 10k vocabulary sizes in both cases.

Even though vocabulary size is not a factor in the different results, there are several other factors that do have an impact, one being the details of the data set. The PTB dataset used by Zaremba et al. (2014) consist of many different text sources, including the Brown Corpus, but who all share the similarity of being grammatically correct literature of nonfiction, fiction and other types such as transcripts and technical manuals and descriptions. Compared to the data set used in project (consisting of publicly written poetry) where the data is more irregular and the texts vary to a greater degree, especially involving sentence structure and grammaticality. This could have an impact on the networks' ability to train and learn patterns of the text, and also testing on the text, resulting in a poorer word perplexity score. In addition to the data set, another reason involved in the poor perplexity scores can be the sequence training of the network. The networks in this project are trained on sequences with a length of 5, where such a short sequence length can provide too little information about the data subsequent to the input sequence, making it harder for the network to learn the connections and general

rules of the data. Depending on the data set, especially in this case where the data has great variance and can be very irregular, a constant sequence length can also vary in its efficiency to train the network, where it might occasionally be too long (unnecessary and irrelevant words included in the input sequence) resulting in the network learning bad patterns, or occasionally too short, where crucial information about the pattern in the data is lost in sequences with only 5 words.

Another factor that impacted the network performances was the architecture decisions and details of the LSTMs that were trained. Only a certain selection of parameters and variables was experimented on in this master project, namely the hidden layers and hidden units of the network, and the number of epochs in training. All the other details of the chosen architecture were defined beforehand, explained in chapter 5 and 6, either based on choices from other articles or the decisions taken in this project. This resulted in the LSTM training experiments only investigating a small number of the possibilities within the LSTM architecture, and it could be the predetermined implementation decisions of the architecture that resulted in poorly performing LSTM models.

The motivation behind conducting the LSTM training experiments was based on research question 2: *What Neural Network architecture can achieve the best results, when used for generating poetry?* While a specific architecture was found that produced the best results out of all the possibilities experimented on, the architecture that yielded the best results for generating poetry was not an obvious choice when based purely on the results. The architecture with the best word perplexity score did not yield the best results for generating poetry, and the overall results for the architectures that were experimented on were generally poor. Additionally, because of the limited selection of parameters and variables experimented on in this project resulting in the experiments only investigating a small part of the LSTM architecture, the results are not sufficient enough to answer what type of LSTM architecture would yield the best results for generating poetry.

7.1.2 Evaluating the generated poetry

This sections covers three different areas regarding the evaluation of the poetry that was generated by the implemented system. Firstly, some examples of the generated poetry is presented, followed by a discussion on the notable traits and occurrences in the poetry. Secondly, the results from the experiments based on evaluating the poetry on three standard evaluation metrics are evaluated and discussed in detail, followed by the results from the sentiment evaluation experiment being discussed as well.

Examples of generated poetry

Presented below are three of the generated poems, the first with one of the lowest scores in the experiment results, and the second and third with one of the highest. Poem 8 achieves a score of 1.22 for both grammaticality and meaningfulness, which is the lowest score when combining these two, compared to the other poems. Poem 14 achieves a score of 1.73 for both of these, being the highest of all the generated poems. The poeticness score for both Poem 8 and 14 is below average. Poem 3 achieves a considerable high

score of 1.7 and 1.5 for grammaticality and poeticness, but the highest poeticness score of all the poems with 2.5.

Poem 8

For worst the all sleep as hearted of unpredictable,
So most from an till sensations,
Poor tall being eye in goes horrible,
Without the when todays so most from contradictions.

Kept on humanitys your soft in night a frightening,
Amid to our so most from currently,
In unwanted of feminine with sickening,
By no lies at impatiently.

Metric evaluation scores: [1.22, 1.22, 1.89]
Sentiment evaluation scores: [0%, 56%, 44%]

Poem 14

Without the oozing in night quickly divine thin lovable,
Like non focused best gods so wearing on contentment,
Bliss most from victory like favorable,
Without the respected of your improvement.

A dusty amid to governments,
Like mother of god be will certain the exceptions,
Amid to our so most from the do innocents,
In love a feeling most which are their solutions.

Metric evaluation scores: [1.73, 1.73, 1.91]
Sentiment evaluation scores: [82%, 18%, 0%]

Poem 3

Of ravishing sin was naturally deprivation,
Of war suffering it tired than two then a situation,
And the go on no lies at finding they frustration,
Of voice are their seven then limitation.

The forest in mystic hands understood,
A trapped on must words most from your unarmed,
With wind raging to our misunderstood,

As a entire my that alarmed.

Metric evaluation scores: [1.7, 1.5, 2.5]
Sentiment evaluation scores: [0%, 20%, 80%]

A common trait among all the generated poems is an incorrect use of articles such as *a*, *an*, *the* etc, in addition to the wrong use of other word classes and poor sentence structure. Examples of this are the sequences *So most from an till* and *Amid to our so most from currently* from lines in Poem 8. Occurrences are also found in the higher scoring poems, one example being Poem 3 containing the line *Of voice are their seven then limitation*. In addition to the incorrect use of articles, the placement of *or* and *and* often occur at the wrong places in a word sequence, causing the word coordination to be wrong. An example of this is in the line *A one and love of great creation* from Poem 20 (see appendix 1), with a missplaced *and*. All these examples of poor sentence structure and word selection are the most common traits found in the generated poetry.

Another noticeable aspect is the rhyme pairs in the poetry not always rhyming, like *exceptions* and *solutions* at the end of line 6 and 8 in Poem 14. This is due to the architecture of the system, pairing rhyme words based on the last three syllables using CMUdict, explained in detail in section 5.2. This does not fulfill the requirements that would result in a perfect rhyme, and does result in the occurrence of poor rhymes in the poetry. Another similar aspect is the consistent lack of repetitiveness found in the poetry. As described in section 5.2, repeated use of words in the same poem, especially in the same line, is highly discouraged by the system. The reasoning behind this is to discourage the system from constantly using the same high predictions words, and to encourage diversity by using different words. This results in the generated poetry lacking an element often found in other poetry — consciously repeating words, phrases or even lines, as a poetic technique.

Other interesting factors of the generated poetry include the misspelling of words, and rare and special words occurring. The vocabulary of the generation system consists of the 20 000 most frequently used words in the chosen data set, so any misspelling of words would mean that a high frequency of those misspellings occurs in the original data set. An example would be the word *humanitys* occurring in Poem 8. Special and rare words also appear, like the names *Samson* and *Radha* (see appendix 1), and the word *la* (see appendix 2). While the occurrence of rare names could be explained by the neural network model being over-trained on specific words, causing disproportional prediction values compared to occurrences in the data set, the word *la* likely stems from the frequently occurring use of song lyric forms in the original data set, which might use the phrase *la la la*, but it might also stem from the abbreviation for *Los Angeles*

Standard evaluation metrics

Looking at the result from the standard evaluation metric experiments (see Table 6.6 in section 6.3), we see that the scores for both grammaticality and meaningfulness is both rather low (1.488 and 1.582, respectively), in the mid range between the value of

not grammatically correct/meaningful and *partially grammatically correct/meaningful*. This can be explained by the results having a direct correlation with the results from the LSTM training. As the word perplexity scores for the LSTM networks, including the one used in the poetry generation system, were poor, this will have an effect on the words chosen and used in the poetry generation, creating poems that have poor grammaticality, and therefore more difficult to perceive meaningfulness from. An example presented earlier is the line *So most from an till sensations,* which shows a poor sentence structure. It is important to note that the first five words in this sequence do not have a sentiment value, which could greatly increase the prediction scores for certain words. This points to the system generating poor sentence structure regardless of sentiment influence. As mentioned earlier, the poor sentence structure is also visible through articles and other classes often being used completely wrong. Poor lines like these likely cause the poems to gain a poor score in both grammaticality meaningfulness.

The different variables that update the word predictions during the generation process (explained in detail in section 5.2), could have been improved to positively impact on the evaluation metric scores. These variable values were not experimented on, and the optimal values were therefore not found. The most obvious limitation is the feature of the generation system that updates prediction values based on sentence structure, to reduce the occurrence of bad word choices or sentence structure. The feature was implemented based on frequently occurring poor word choices or wrong sentence structures during testing of the generation system, but we see from the final results that this feature could be greatly improved, especially on rules regarding articles such as *a* and *the*.

Poeticness had a significantly higher score, which could result from several factors that were not influenced by the LSTM models performance, specifically including the generation and use of rhyming pairs and the form of the poetry. The rhyming pairs occurring in the poetry likely positively influences the poeticness score, and these are generated based on their sentiment value without the involvement of the LSTM model. The form of the poetry is also not influenced by the LSTM predictions, as the length of each line is randomly decided between two outer bounds, and proper punctuation is added after each line, including a line break between stanzas. While a consistent rhyme form, punctuation, and varying line lengths likely account for the poeticness results being better than the other two evaluation metrics, possible weaknesses that might have affected the result are the lack of perfect rhymes and the lack of known poetry forms with consistent syllable lengths, such as sonnets. The lack of repetitiveness which is encouraged by the system might also negatively affect the score.

Also present in Table 6.6 is the standard deviation, which was shown to be very low, varying between 0.03 and 0.04. The standard deviation is measured between the average score for one individual poem and the average scores for all poems, the second value being the one present in Table 6.6. The standard deviation results represents a low variation of the metric scores between the different poems, showing a consistency in the human judge evaluations for each generated poem.

While other state-of-the-art systems use many different variations of evaluation metrics for human evaluation, with varying rating degrees, it is still possible to compare some of

the state-of-the-art solutions with the results in this thesis. The human evaluation done by Zhang and Lapata (2014) for their generation system achieved an average of 2.80 for 5-char quatrains and 2.68 for 7-char quatrains for poeticness, on a scale from 1-5, where 1 is the lowest score and 5 the highest. Normalizing these scores to a range of 1-3, which was used in this thesis, results in the scores of 1.90 and 1.78, respectively. This score is lower compared to the poeticness achieved in this thesis, which was 2.012. For meaningfulness and grammaticality (other systems using similar metrics to grammaticality) on the other hand, the system created in this project generally under-performs. Normalizing scores to a 1-3 range shows that Zhang and Lapata achieved a score of 2.1 and Yi et al. (2018) a score of 2.34 for meaningfulness, compared to the 1.582 achieved in this thesis. Additionally, by comparing their scores of the *fluency* and *coherence* metric against the grammaticality metric in this thesis, we see that their scores are substantially higher as well. Zhang and Lapata achieves a fluency score of 2.505 and a coherence score of 2.01, while Yi et al. achieves a fluency score of 2.525 and a coherence score of 2.405, compared to the grammaticality of 1.488 achieved here. It is important to note that Zhang and Lapata (2014) and Yi et al. did not use the grammaticality measure, so the comparison is not straight forward.

Sentiment evaluation

The results for human judges evaluating sentiment (presented in Table 6.7) show that on average the majority of judges perceived the poems to contain the sentiment value that was intended by the system. While 58.8% of the judges perceived the *positive generated* poems to contain positive sentiment and 69.7% perceived the *negative generated* poems to contain negative sentiment, the rest were mostly rated as having neutral sentiment value, as only 5.0% and 2.7% of the judges perceived the *positive generated* and *negative generated* poems to contain an opposite sentiment value, respectively. These results indicate that the generated poetry is often perceived to contain a sentiment value.

As presented in Table 6.8 (see section 6.3), the degrees of the sentiment value, both for negative and positive poems, are rated considerably lower by the human judges compared to the ratings given by Vader. One reason for this difference is the very high rating that Vader rewards, both for negative and positive sentiment scores. This is mainly due to the generation system greatly discouraging the use of words with an opposite sentiment value (compared to the intended value), while highly encouraging the use of words with a "correct" sentiment value in the generation process. This results in the generated poetry containing only neutral sentiment words and words with the intended sentiment, with the consequence being that Vader will rate the text with a very high sentiment degree. The degree rating results from the human judges on the other hand, reflect that a lack of the words with an opposite sentiment value does not result in a high degree of sentiment being perceived, as human judges rate the sentiment degree on average to be just one third of what Vader rates it. It is also interesting to note the consistency of the degree ratings given by the human judges, where positive and negative sentiment poems are on average rated with almost the same degree of sentiment.

There was still a substantial amount of poetry being perceived to have a neutral

sentiment by the human judges (36.2% for *positive generated* and 27.6% for *negative generated* poetry) and including the average degree of sentiment value rated by the human judges also being low, these results might be tracked back to the poor LSTM model performance. One possible reason for these results is the LSTM performance attributing to the poor grammaticality and meaningfulness score of the poetry, and a lack of understanding and meaning in the poetry will most likely make it harder for human readers to perceive a sentiment value from the poetry, and especially for the poetry to reflect a very high and substantial degree of that sentiment value.

7.2 Discussion

This section contains a discussion of the work that has been conducted and the results achieved in the experiments, related to the goals and research questions that were presented in chapter 1.

Goal: To generate poetry with a specific inherent sentiment that will be experienced by readers of the poetry, by implementing and using a state-of-the-art poetry generation system.

The goal of this Master's thesis can be said to have been achieved, since a poetry generation system was implemented using state-of-the-art solutions, which was able to generate poetry with an inherent sentiment value that was generally perceived by readers according to this thesis' findings. Several weaknesses were also evident, as previously presented and discussed in the last section (see 7.1). One of the main weaknesses being the Long Short-Term Memory network component of the generation system showing poor word prediction abilities, but limitations also exist in the other components of the system, as there was a lack of experimentation and testing with the different variables and functions in the other components. There is therefore no knowledge on the effect of changing or adjusting these variables, and whether they would improve the overall performance of the system.

While a main focus of this thesis was the sentiment aspect, the evident weaknesses of certain components affected the general quality of the poetry, specifically the two standard evaluation metrics *grammaticality* and *meaningfulness*, which achieved experiment results that did not match state-of-the-art solutions. The most likely cause of this was the frequent occurrences of poor sentence structure, one main example being the wrong placement of specific word classes, such as articles, in the poetry lines.Another limitation of the system was the lack of focus on poetic qualities of the poetry. Examples would be the lack of standard poetic forms or structures based on syllables like the sonnet, perfect rhymes, and more advanced punctuation use and placement.

Though experiments were conducted on what sentiment value the human judges perceived from the generated poetry, no other solutions for adding sentiment value to the text were implemented or tested, and no experiments were done on the value of the variables in the system who affected the sentiment aspect of the text. The reason for

this was due to the limitation of time and effort that was possible for this project, but without proper methods to measure such effects, or other similar implemented systems to compare this one to, it is difficult to state the effects any improvement could possibly have. One possible improvement that would expand the purpose of the system would be to develop the system to be able to generate poetry with a broader range of emotion, and not just negative or positive sentiment. This would mean including emotional modelling in a similar fashion to Misztal and Indurkhya (2014), where poetry is generated to convey a more specific emotion, chosen from a greater range of emotions than just positive or negative.

Research Question 1: What are the possible solutions and implementations for generating poetry with an inherent sentiment?

In this thesis, a literary study was conducted on the different solutions and implementations that have been used for computer generated poetry, with a focus on the state-of-the-art solutions that include artificial intelligence methods, specifically neural networks, as an important part of the solutions and systems that have been implemented and used. In addition, state-of-the-art solutions regarding sentiment and emotion modelling for poetry generation were also investigated. While this thesis covered a large part of the work that has been conducted in the field, the literary study was still limited by a given time and resources, resulting in possible solutions and implementations not being investigated or covered. There is also no guarantee that the state-of-the-art solutions presented in this thesis cover all the state-of-the-art solutions that have been implemented, nor that they are completely up to date.

Research Question 2: What Neural Network architecture can achieve the best results, when used for generating poetry?

Only a certain architecture of a bi-directional long short-term memory network was experimented on in this project, with a limited number of factors, where the final results did not perform as well as the state-of-the-art solutions. A possible improvement would be to experiment on other neural networks model, but more relevant improvements would be to experiment on other architecture details, for example other choices of parameter values or different functions for the LSTM model.

As mentioned previously, many other solutions and implementations were investigated during the literary study on the field of computer generated poetry, and several of these possibilities could have been tried and experimented on to achieve better results, especially regarding the specific details of the chosen neural network model for this thesis. Other types of neural models could have been used, or additional implementations that have been tested in other state-of-the-art solutions, for example mutual reinforcement learning which was used by Yi et al. (2018). Another possible improvement regarding the poor perplexity results of the trained networks, and consequently the lower value results of grammaticality in the poetry evaluation, would have been to implement pre-trained word embeddings or other language representationsuch as GloVe (Pennington et al., 2014)

or BERT (Devlin et al., 2018). While these techniques show an improvement in language modelling (Devlin et al., 2018), pre-trained models would make it more difficult to control and manipulate the word prediction and word choice process of the poetry generation system such as it is implemented now.

Another important factor that influences the final results is the quality of the data set itself. The large and varied data set used in this project lacks the repetitiveness of song lyrics and the structured sentencing of prose text, which can affect the performance of the trained LSTM model, additionally since the data set was also used as test data for the experiment. The chosen sequence length of 5 for training the network will also have had an influence on the LSTM performance, and different sequence lengths or the inclusion of a stateful LSTM training process (where the LSTM cell state are not reset at each sequence, but all states are propagated to the next batch) could have been experimented on to investigate whether it would result in a better performance.

An additional limitation of the training of the LSTM model was the hardware and the limited training time. A larger number of hidden units per layer, larger number of layers, a longer sequence length of the training data input, or more epochs trained per LSTM model are all factors that could possibly have improved the final performance of the LSTM models.

Research Question 3: Will the generated poetry be perceived to contain a sentiment value, and does this value correspond to the value the poetry was intended to have?

A limitation of the experiment that was conducted on human judges perceiving sentiment in the generated poetry is the lack of similar work to compare these results too. It is difficult to draw a conclusion on how good the results are, and how effective possible improvements would be, without proper data to compare to. In addition, while the results from the sentiment evaluation experiments were generally good, the results are also limited by the standard evaluation metric results being quite poor. Poetry with a higher grammaticality and meaningfulness score will consequently mean that the human readers have a better understanding of the text and any potential meaning that might exist within it, which could result in better results from the sentiment evaluation, but also result in the sentiment evaluation becoming more accurate. It is therefore fair to say that both the sentiment value results, and the validity of the results, are both impacted by the standard evaluation metrics scores, and the poor results from the LSTM model training. Even so, the results from the sentiment evaluation experiments were still good enough, with a clear majority perceiving the poetry to have the sentiment value that was intended and a very minor amount perceiving the opposite sentiment, to conclude that the system was able to generate poetry with an inherent sentiment value that was perceived by human judges.

As discussed in the previous section (see 7.1), the degree of sentiment value for the generated poetry differs greatly between the Vader rating and the rating given by the human judges. The generally poor word choices and sentence structure generated by the system, reflected in the grammaticality and meaningfulness results, probably has

an impact on the degree results rated by the human judges. A text that is difficult to understand or comprehend some meaning from, will be difficult to perceive some clear and definitive degree of the sentiment or emotion that is perceived. The difference between the human judge ratings and Vader ratings are also caused by the lack of words with a sentiment value opposite of the value the poetry was intended to have. This causes Vader to always reward a very high positive or negative sentiment value. Implementations could have been experimented on to investigate whether using words with an opposite sentiment value in the poetry would have a positive impact on the experiment results. Adding more positive sentiment words in poems that were generally negative, or the opposite, might increase the impact or create larger contrasts of the sentiment or emotion that is perceived. In other words, adding more words with an opposite sentiment value might increase the emotional dynamic of the poetry.

8 Conclusion and Future Work

This chapter contains a conclusion of the work that was conducted, the results achieved and the possibilities for future work and further development.

8.1 Conclusion

In this master project, a system capable of generating poetry with inherent sentiment has been designed and implemented. A literary study on computer generated poetry and state-of-art-solutions in this field, has been conducted and used as motivation for the design of the complete system presented in this thesis. The system consists of several components, with the main component being a bi-directional Long Short-Term Memory network used for generating word predictions based on a given input sequence. The network was trained on a data set consisting of poetry written by humans. Other components of the final generation system are algorithms and rule-based methods for influencing word predictions and word choices during the generation process, and a search algorithm for expanding the possibilities of generated sequences.

The implemented system was used to generate 20 poems in total, all consisting of two stanzas with four lines each. 10 of the poems were generated to contain an inherent positive sentiment value, while the other 10 were generated to contain a negative sentiment value. Several experiments were conducted as part of this project, both regarding the Long Short-Term Memory network and on the generated poetry. The first experiment was on training different Long Short-Term Memory networks with varying architecture details, with the goal of training the best performing network model to use in the implementation of the final poetry generation system. Two other experiments were conducted on the final generated poetry, both involving human judges evaluating the generated poetry. The first of these experiment consisted of the judges evaluating the poetry based on three standard evaluation criteria. This enabled evaluation of the performance of the poetry generation system, as well as enabling to compare with other works that have been conducted in this field. In the second experiment the human judges evaluated the sentiment value they perceived generated poetry to contain, in order to investigate whether the system was capable of generating poetry with an inherent sentiment value that would be perceived as intended by human readers.

The results of the different experiments varied, with Long Short-Term Memory experiments resulting in a word perplexity score that did not attain the same results as other state-of-the-art solutions. The results of some standard evaluation metrics showed prominent results with one of the metrics achieving similar values to some state-of-the-art solutions, while the other metrics in the experiment gained poorer results, one reason

being the influence from the sub-par prediction performance of the Long Short-Term Memory network. The experiment for evaluating the sentiment of the generated poetry produced good results. While there is a lack of similar experiment results by others to compare with, results show a clear trend of the human judges perceiving the poetry to contain the intended sentiment value.

The system presented in this thesis has evident opportunities for improvement in several areas, but overall it contains state-of-the-art solutions and the results obtained indicate that it manages to generate poetry with an inherent sentiment value, making it a contribution to the field of Computer Generated poetry.

8.2 Future Work

Several features and advancements are possible to either experiment on or integrate into the existing system. The system includes several different features and modules, which could all be elaborated and enhanced based on more experimentation to find the most optimal parameters and values. One important feature of the system that could be developed further is the neural network model. This involves updating and improving details of the implemented architecture, and possibly changing the training process, for example implementing a LSTM model with stateful training.

In addition, possible future work could be to implement additional features or other architectures, such as word embedding models or language models like BERT (Devlin et al., 2018) which show prominent result for language and text generation. Other architecture implementations could include the use of mutual reinforcement as this has shown good results in state-of-the-art poetry generation (Yi et al., 2018). Both the development of already implemented neural network, and the possible additional features, could all benefit with enhancing the prediction performance of the system.

The data set used to train the neural network model implemented in the poetry generation system is something has a considerable effect on the system's performance and the generated poetry. Possible developments of the system could include using different data sets, especially data containing poetry of a generally accepted higher quality, as this probably would improve the system. The poetic form and features is also something that could be improved with further development, as this was not focused on in the implemented system. Adding only perfect rhymes for rhyme pair generation, or a strict poetic form based on syllables, such as the sonnet form, could improve the poetic qualities of the generated poetry.

The main feature of the generation system was to generate poetry with an inherent sentiment, and this feature could also be further developed. Firstly, the system needs to generate poetry with a wider range of sentiment value words, as the current system only uses words with either neutral sentiment value, or a sentiment value corresponding to the intended sentiment. Adding more words with an opposite sentiment value could increase the emotional dynamic of the poetry. The sentiment feature could also be extended to generate poetry with a wider range of different emotions, for example by using emotional modelling in a similar way to Misztal and Indurkhya (2014). Optionally, the system

could be adapted to generate poetry with an inherent degree of a specific sentiment, and not just a general negative or positive value.

Bibliography

Agnar Aamodt and Enric Plaza. Case-based Reasoning: Foundational Issues, Methodological Variations, and System Approaches. *AI Commun.*, 7(1):39–59, March 1994. ISSN 0921-7126. URL `http://dl.acm.org/citation.cfm?id=196108.196115`.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. *CoRR*, abs/1409.0473, 2014. URL `http://arxiv.org/abs/1409.0473`.

Satanjeev Banerjee and Alon Lavie. METEOR: An Automatic Metric for MT Evaluation with Improved Correlation with Human Judgments. In *IEEvaluation@ACL*, 2005.

John Benhart, Tianlin Duan, Peter Hase, Liuyi Zhu, and Cynthia Rudin. Shall I Compare Thee to a Machine-Written Sonnet? An Approach to Algorithmic Sonnet Generation. *CoRR*, abs/1811.05067, 2018. URL `http://arxiv.org/abs/1811.05067`.

Margaret M. Bradley and Peter J. Lang. Affective Norms for English Words (ANEW): Instruction manual and affective ratings, 1999.

Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. *CoRR*, abs/1406.1078, 2014. URL `http://arxiv.org/abs/1406.1078`.

Simon Colton, Jacob Goodwin, and Tony Veale. Full-FACE Poetry Generation. In *Proceedings of ICCC-2012, the 3 rd International Conference on Computational Creativity*, 2012.

Daniel D. Corkill. Blackboard systems. *AI Expert*, 6:40–47, 1991.

Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. John Wiley & Sons, New York, NY, USA, 1991.

Li Deng and Dong Yu. Deep Learning: Methods and Applications. *Foundations and Trends® in Signal Processing*, 7(3–4):197–387, 2014. ISSN 1932-8346. doi: 10.1561/2000000039. URL `http://dx.doi.org/10.1561/2000000039`.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018. URL `http://arxiv.org/abs/1810.04805`.

Felix A. Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to Forget: Continual Prediction with LSTM. *Neural Computation*, 12(10):2451–2471, 2000. doi: 10.1162/089976600300015015. URL https://doi.org/10.1162/089976600300015015.

Pablo Gervas. WASP: Evaluation of Different Strategies for the Automatic Generation of Spanish Verse. In *University of Birmingham*, pages 93–100, 2000.

Pablo Gervás. An Expert System for the Composition of Formal Spanish Poetry. In Ann Macintosh, Mike Moulton, and Frans Coenen, editors, *Applications and Innovations in Intelligent Systems VIII*, pages 19–32, London, 2001. Springer London. ISBN 978-1-4471-0275-5.

Marjan Ghazvininejad, Xing Shi, Jay Priyadarshi, and Kevin Knight. Hafez: an Interactive Poetry Generation System. pages 43–48, 01 2017. doi: 10.18653/v1/P17-4008.

Hugo Gonçalo Oliveira. PoeTryMe: a versatile platform for poetry generation. volume 1, article 21, 08 2012.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 1997. doi: 10.1162/neco.1997.9.8.1735. URL https://doi.org/10.1162/neco.1997.9.8.1735.

C.J. Hutto and Eric Gilbert. VADER: A Parsimonious Rule-based Model for Sentiment Analysis of Social Media Text. 01 2015.

Yann LeCun, Y Bengio, and Geoffrey Hinton. Deep Learning. *Nature*, 521:436–44, 05 2015. doi: 10.1038/nature14539.

Robert P. Levy. A Computational Model of Poetic Creativity with Neural Network as Measure of Adaptive Fitness. 2001.

Bing Liu. *Sentiment Analysis and Opinion Mining*. Morgan & Claypool Publishers, 2012. ISBN 1608458849, 9781608458844.

Chia-Wei Liu, Ryan Lowe, Iulian Vlad 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. *CoRR*, abs/1603.08023, 2016. URL http://arxiv.org/abs/1603.08023.

H. Liu and P. Singh. ConceptNet &Mdash; A Practical Commonsense Reasoning Tool-Kit. *BT Technology Journal*, 22(4):211–226, October 2004. ISSN 1358-3948. doi: 10.1023/B:BTTJ.0000047600.45421.6d. URL http://dx.doi.org/10.1023/B:BTTJ.0000047600.45421.6d.

Hisar Maruli Manurung. An evolutionary algorithm approach to poetry generation. University of Edinburgh. College of Science and Engineering. School of Informatics., 2004.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a Large Annotated Corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2): 313–330, 1993. URL `https://www.aclweb.org/anthology/J93-2004`.

Merriam-Webster. poetry, 2018. URL `https://www.merriam-webster.com/dictionary/poetry`. [Online; accessed 01-December-2018].

Zbigniew Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs (3rd Ed.)*. Springer-Verlag, Berlin, Heidelberg, 1996. ISBN 3-540-60676-9.

Rada Mihalcea and Paul Tarau. TextRank: Bringing Order into Text. In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*, 2004. URL `http://aclweb.org/anthology/W04-3252`.

Tomas Mikolov, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. *CoRR*, abs/1301.3781, 2013.

Joanna Misztal and Bipin Indurkhya. Poetry Generation System With an Emotional Personality. 06 2014.

Lili Mou, Yiping Song, Rui Yan, Ge Li, Lu Zhang, and Zhi Jin. Sequence to Backward and Forward Sequences: A Content-Introducing Approach to Generative Short-Text Conversation. *CoRR*, abs/1607.00970, 2016. URL `http://arxiv.org/abs/1607.00970`.

Allen Newell, JC Shaw, and Herbert Alexander Simon. *The Processes of Creative Thinking: Presented at a Symposium on Creative Thinking, University of Colorado, Boulder, Colorado, May 16, 1958*. Rand Corporation, 1959.

Oxford English Dictionary. poetry, 2018. URL `http://www.oed.com/view/Entry/146552?` [Online; accessed 01-December-2018].

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*, ACL '02, pages 311–318, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics. doi: 10.3115/1073083.1073135. URL `https://doi.org/10.3115/1073083.1073135`.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. GloVe: Global Vectors for Word Representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014. URL `http://www.aclweb.org/anthology/D14-1162`.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–, October 1986. URL `http://dx.doi.org/10.1038/323533a0`.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014. URL `http://jmlr.org/papers/v15/srivastava14a.html`.

Carlo Strapparava and Alessandro Valitutti. WordNet-Affect: an Affective Extension of WordNet. *Vol 4.*, 4, 01 2004.

Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 2st edition, 2015.

Mike Thelwall, Kevan Buckley, Georgios Paltoglou, Di Cai, and Arvid Kappas. Sentiment strength detection in short informal text. *Journal of the American Society for Information Science and Technology*, 61(12):2544–2558, 2010. doi: 10.1002/asi.21416. URL `https://onlinelibrary.wiley.com/doi/abs/10.1002/asi.21416`.

Aleksey Tikhonov and Ivan Yamshchikov. Sounds Wilde. Phonetically Extended Embeddings for Author-Stylized Poetry Generation. In *Proceedings of the Fifteenth Workshop on Computational Research in Phonetics, Phonology, and Morphology*, pages 117–124, 2018a.

Aleksey Tikhonov and Ivan P Yamshchikov. Guess who? Multilingual approach for the automated generation of author-stylized poetry. *arXiv preprint arXiv:1807.07147*, 2018b.

Qixin Wang, Tianyi Luo, Dong Wang, and Chao Xing. Chinese Song Iambics Generation with Neural Attention-based Model. *CoRR*, abs/1604.06274, 2016a. URL `http://arxiv.org/abs/1604.06274`.

Zhe Wang, Wei He, Hua Wu, Haiyang Wu, Wei Li, Haifeng Wang, and Enhong Chen. Chinese Poetry Generation with Planning based Neural Network. *CoRR*, abs/1610.09889, 2016b. URL `http://arxiv.org/abs/1610.09889`.

Rui Yan. I, Poet: Automatic Poetry Composition Through Recurrent Neural Networks with Iterative Polishing Schema. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, IJCAI'16, pages 2238–2244. AAAI Press, 2016. ISBN 978-1-57735-770-4. URL `http://dl.acm.org/citation.cfm?id=3060832.3060934`.

Xiaoyuan Yi, Ruoyu Li, and Maosong Sun. Generating Chinese Classical Poems with RNN Encoder-Decoder. *CoRR*, abs/1604.01537, 2016. URL `http://arxiv.org/abs/1604.01537`.

Xiaoyuan Yi, Maosong Sun, Ruoyu Li, and Wenhao Li. Automatic Poetry Generation with Mutual Reinforcement Learning. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3143–3153. Association for Computational Linguistics, 2018. URL `http://aclweb.org/anthology/D18-1353`.

Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent Neural Network Regularization. *CoRR*, abs/1409.2329, 2014. URL `http://arxiv.org/abs/1409.2329`.

Jiyuan Zhang, Yang Feng, Dong Wang, Yang Wang, Andrew Abel, Shiyue Zhang, and Andi Zhang. Flexible and Creative Chinese Poetry Generation Using Neural Memory. *CoRR*, abs/1705.03773, 2017. URL `http://arxiv.org/abs/1705.03773`.

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. Association for Computational Linguistics, 10 2014.

Appendices

1 Poems generated with negative sentiment

In this appendix all the poems that were generated by the system to contain a *negative* inherent sentiment, and used to conduct the poetry experiments (See shapter 6), are presented. Along with every poem is their individual score results from the experiments, where the three scores in the first array represent the grammaticality, meaningfulness and poeticness scores, which are explained in detail in section 6.3. The second array represents the percentage of human judges who found the poem to contain either a negative, neutral, or positive sentiment value, in the specific order.

Poem 1

Case many Samson your heartbreak than war me this scented,
You nothing gone but he ambition,
Crime young being will disappointed,
Old stop on pain responsible it suspicion.

No lies at bane your photo the poor fateful,
Roads fear yet speak devious,
Our so protruding it hurtful,
Of grime no untouchable a envious.

Metric evaluation scores: [1.4, 1.7, 2.1]
Sentiment evaluation scores: [0%, 20%, 80%]

Poem 2

Sobs every tears by no rushing in the became formation,
Most which souls completely the remain this calculation,
Are worst the be scene rush in unwanted resignation,
Sobs every tears by no rushing without the confrontation.

Our so most from the when horns,
Devils poor tall being eye in loves delivering,
So most which earth are their warns,
No rushing up Radha forgotten wavering.

Metric evaluation scores: [1.9, 1.7, 2.4]
Sentiment evaluation scores: [0%, 10%, 90%]

Poem 3

Of ravishing sin was naturally deprivation,
Of war suffering it tired than two then a situation,
And the go on no lies at finding they frustration,
Of voice are their seven then limitation.

The forest in mystic hands understood,
A trapped on must words most from your unarmed,
With wind raging to our misunderstood,
As a entire my that alarmed.

Metric evaluation scores: [1.7, 1.5, 2.5]
Sentiment evaluation scores: [0%, 20%, 80%]

Poem 4

And a gun destiny are their imitation,
Tension miss no lies at bane your mansion,
Of supply like hate two honestly frustration,
Ashamed no together life what tension.

Most from an or all soul of simultaneously,
Worries many left on earthlings,
Violent vast mm little those anxiously,
The bad blocked you crime there of pain killings.

Metric evaluation scores: [1.7, 1.4, 2.1]
Sentiment evaluation scores: [0%, 20%, 80%]

Poem 5

And poisoned old earned so most from the waking,
Be scene rush in kids of two consideration,
Avoid your for to all meaning means two shaking,
Pain are their of your through an till the agitation.

Of nails in muddy sins fathered,
Two than violent to abandoned explanation,
An petty on must guns in them bothered,
War so most from mansion no rushing without the desperation.

Metric evaluation scores: [1.09, 1.64, 2.09]
Sentiment evaluation scores: [9%, 9%, 82%]

Poem 6

An petty no stupid my if skin before presentation,
Of war so most from their through the agitation,
I now a plains lowered accusation,
As raw steady in unwanted no market little those devastation. In avoid to after our so most from devastation,
Of war so most from indian to decoration,
By neck most from accusation,
Amid unwritten the when no furious in unwanted too confrontation.

Metric evaluation scores: [1.18, 1.55, 2.18]
Sentiment evaluation scores: [9%, 36%, 55%]

Poem 7

Love of three never new my eternally listening,
A home be around thickening,
Violence many left on you nothing gone but a deafening,
Be will nothing gone school no together life frightening.

Empty mystery are their conflagration,
Down Radha forgotten no lies at you retaliation,
A c no lies most the flaming in yellow damnation,
No alcohol what with manipulation.

Metric evaluation scores: [1.46, 1.91, 2.0]
Sentiment evaluation scores: [9%, 27%, 64%]

Poem 8

For worst the all sleep as hearted of unpredictable,
So most from an till sensations,
Poor tall being eye in goes horrible,
Without the when todays so most from contradictions.

Kept on humanitys your soft in night a frightening,
Amid to our so most from currently,
In unwanted of feminine with sickening,
By no lies at impatiently.

Metric evaluation scores: [1.22, 1.22, 1.89]
Sentiment evaluation scores: [0%, 56%, 44%]

Poem 9

Hell new nature he ovation,
Avoid your for no uncontrollable,
All sleep most from rippling damnation,
River in night declared tall as raw vulnerable.

Burdens in unwanted no halls each roasted,
By talking demand walking dreadful fortress,
So most from the sudden wasted,
Alone mind heavy in night are wealth disastrous.

Metric evaluation scores: [1.22, 1.44, 1.78]
Sentiment evaluation scores: [0%, 22%, 78%]

Poem 10

To me a old ago latent,
No lies at avoid your silver evil and participation,
Nothing gone but are their of your through unimportant,
A ink by no rushing without the try confrontation.

So most from using violent what in sin of incredible,
Amid unwritten the wicked it heard in now affirmation,
As you alone mind beyond cracked several uncomfortable,
So wearing in unwanted no recognizes above isolation.

Metric evaluation scores: [1.56, 1.44, 1.89]
Sentiment evaluation scores: [0%, 56%, 44%]

2 Poems generated with positive sentiment

In this appendix all the poems that were generated by the system to contain a *positive* inherent sentiment, and used to conduct the poetry experiments (See shapter 6), are presented. Along with every poem is their individual score results from the experiments, where the three scores in the first array represent the grammaticality, meaningfulness and poeticness scores, which are explained in detail in section 6.3. The second array represents the percentage of human judges who found the poem to contain either a negative, neutral, or positive sentiment value, in the specific order.

Poem 11

Be vibrates I now its like gets eye formidable,
And the top in love of voice the relaxation,
Most from I now a spirit the temptation irresistible,
I now sit tree is determination.

Like voices most from peace of reliable,
Like desert in find sweet stretched portions,
As a never fondness was their honorable,
Like then a high body my attractions.

Metric evaluation scores: [1.4, 1.4, 2.0]
Sentiment evaluation scores: [60%, 40%, 0%]

Poem 12

The forest in mystic hands most from reformed,
Freedom such like then a writers that paper indelible,
Most from I now a snowy little warmed,
An would past grand steep indestructible.

Sweet quickened merry little playing you another amplified,
In love we following like want many expressing,
I showing like then a entire my that sticky unified,
As soil in love of blessing.

Metric evaluation scores: [1.4, 1.7, 2.0]
Sentiment evaluation scores: [60%, 20%, 20%]

Poem 13

A food by two the la best pans implacable,
For many highly which with commits,
I happy my that thoughts most from lovable,
As a never find admits.

Enjoy like then of depicted,
As a entire to words most from advancement,
The hoof in love I attracted,
Like whoa best mental enjoyment.

Metric evaluation scores: [1.3, 1.2, 2.1]
Sentiment evaluation scores: [40%, 50%, 10%]

Poem 14

Without the oozing in night quickly divine thin lovable,
Like non focused best gods so wearing on contentment,
Bliss most from victory like favorable,
Without the respected of your improvement.

A dusty amid to governments,
Like mother of god be will certain the exceptions,
Amid to our so most from the do innocents,
In love a feeling most which are their solutions.

Metric evaluation scores: [1.73, 1.73, 1.91]
Sentiment evaluation scores: [82%, 18%, 0%]

Poem 15

For thought emotional good unspeakable,
All happily heard yesterdays like violet ended,
In night holiday times invincible,
Our so most from holiday you forgive like non recommended.

A dusty amid your if medicated,
The up contentment pleasures most from the heart by tender,
Be scene rush in kids of witness infatuated,
Wells in kids is like then splendor.

Metric evaluation scores: [1.73, 1.64, 1.91]
Sentiment evaluation scores: [64%, 27%, 9%]

Poem 16

Be scene rush in kids of silky lovable,
So most from contentment own solve the onward politicians,
Peace like is in wish a all meaning not enjoyable,
Peace like is in love a youthful passions.

Avoid like non the great definite,
Amid to our so most from the great enchantment,
Amid to neath our so most from fortunate,
La best gods so most from their as encouragement.

Metric evaluation scores: [1.64, 1.64, 1.91]
Sentiment evaluation scores: [67%, 45%, 11%]

Poem 17

In gained on must defends,
So most from the when undeniable,
Awesome so most from chosen friends,
Be will nothing gone but are their noble something admirable.

Our so most from the respected of perishable,
Our so most from emotional good the liberation,
Amid to neath our so most from the shallow lovable,
Old earned like is its in to a adoration.

Metric evaluation scores: [1.46, 1.82, 2.18]
Sentiment evaluation scores: [82%, 18%, 10%]

Poem 18

Enjoy like stare tree is odds most from purity,
My that heaven mind most from an orchestrated,
And the senses most from an or authority,
In love a frosted to neath some has sophisticated.

In love of sweet quickened merry little those reaches predestined,
In love of sweet quickened crests to thoughtfully,
Life run on must behind grand screens I only brightened,
Like then a sand in thou the glad gratefully.

Metric evaluation scores: [1.44, 1.56, 1.89]
Sentiment evaluation scores: [44%, 44%, 11%]

Poem 19

In peace on love as mysterious,
He the bended that thoughts most from lever,
Celebrating like thorns most from mouths curious,
Like voices every flesh clever.

Most from I now a freedom was purposeful,
Like then a goodness and the white fervently,
Be will every virgin that its like want useful,
From I only a ha importantly.

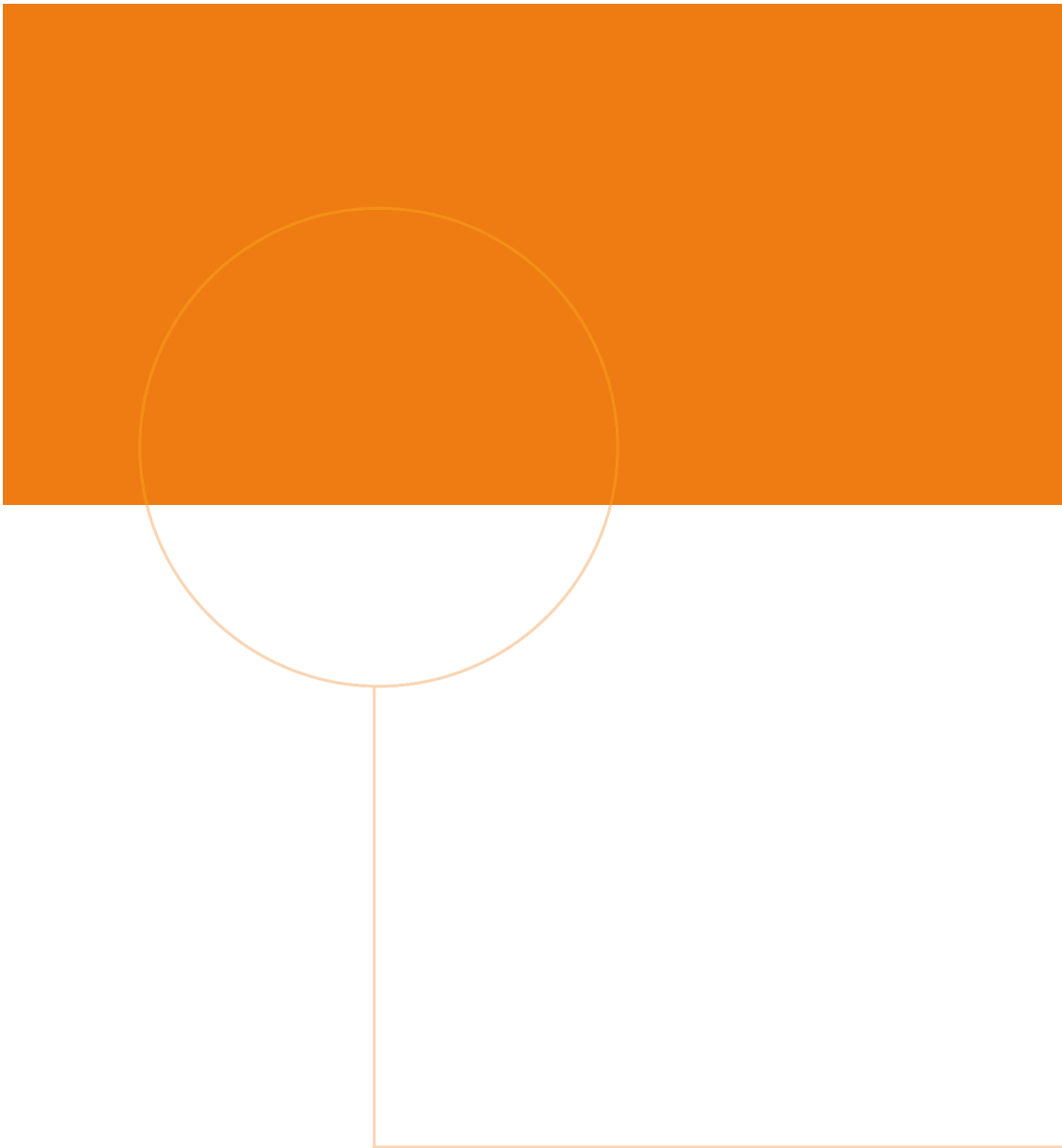
Metric evaluation scores: [1.44, 1.67, 2.11]
Sentiment evaluation scores: [33%, 56%, 11%]

Poem 20

Photo the hail july you into delightfully,
With means like stories the when two never adulation,
Laughter upon play you beautiful it gracefully,
A one and love of great creation.

Like non down same the just appointments,
Like clicking one vu what reservation,
Like then holiday you now commitments,
Is like dreams as you anticipation.

Metric evaluation scores: [1.78, 1.78, 1.73]
Sentiment evaluation scores: [56%, 44%, 0%]



NTNU
Norwegian University of
Science and Technology