Tampere University

Anh Nguyen

AUTOMATIC TEXT GENERATION USING SUPERVISED PRE-TRAINING AND REINFORCEMENT LEARNING BASED ADAPTATION

Bachelor's Thesis
Science and Engineering
Okko Räsänen
Joni Pajarinen
February 25th 2021

ABSTRACT

Anh Nguyen: Text Generation using LSTM and Reinforcement Learning
Bachelor of Science Thesis
Tampere University
International Degree of Science and Engineering
February 25th 2021

Text generation tasks are becoming more and more prominent in applications such as machine translation, image captioning, dialogue systems, etc. While text generation systems often require an extremely large amount of data, the lack of data can be compensated by using certain machine learning algorithms. This Bachelor’s thesis introduces an approach of building a text generation system that utilizes reinforcement learning to control its output. The thesis first discusses training a small text dataset using supervised learning. The thesis then discusses the application of a deep reinforcement learning algorithm to make the generated text adapt to specific criteria. The result shows that the system adapts well to different conditions. The system adapts well to reward functions that depend on temporal modelling of the data, as well as validity of each individual word produced. However, the system was unable to improve the text’s grammaticality or diversity.

Keywords: deep learning, LSTM, RNN, reinforcement learning, deep Q-learning, Keras, TensorFlow, text generation

PREFACE

I would like to thank my thesis supervisors, assistant professors Okko Räsänen and Joni Pajarinen, for suggesting the topic and guiding me through my research progress and writing process. I would like to also thank Mrs. Laeticia Petit for pushing and helping me keep track of my progress and providing useful resources for tasks such as information retrieval or thesis writing guide.

Tampere, Finland, 25th February 2021

Anh Nguyen

CONTENTS

LIST OF FIGURES

LIST OF SYMBOLS AND ABBREVIATIONS

BERT	Bidirectional Encoder Representations from Transformers
CNN	Convolutional Neural Networks
DQN	Deep Q-Learning
JSD	Jensen-Shannon divergence
KLD	Kullback-Leibler divergence
LSTM	Long Short-Term Memory
NLTK	Natural Language Toolkit
RL	Reinforcement Learning
RNN	Recurrent Neural Networks
SGD	Stochastic Gradient Descent
SOTA	State Of The Art

.

1. INTRODUCTION

Natural Language Processing (NLP) is a subfield of artificial intelligence, which defines a set of tools to process natural language data such as text or speech. Useful applications of NLP include machine translation, which has the ability to translate from a language to another, speech recognition, sentiment analysis, which analyses human's emotions based on their input text, and text generation.

This research will be focusing on text generation application of NLP. In particular, the system designed and implemented will be able to generate useful text based on its model trained on an existing text database. Since most NLP applications rely heavily on the use of machine learning algorithms, the aforementioned system will also be utilizing machine learning algorithms, particularly deep learning models. Deep learning is a branch of machine learning which uses neural networks to solve problems such as feature extraction or classification. Moreover, besides building a deep learning model, the system also implements reinforcement learning algorithms in order to improve the quality of the text generated. Reinforcement learning is also a subfield of machine learning whose principle is to have a software-defined agent taking a set of actions, known as policy, in an environment so as to maximize a specific reward schema. Finally, the whole system is evaluated using a custom evaluation function which measures the accuracy of the text generated.

Chapter 2 presents the history of text generation and some of the popular text generation models, as well as key concepts or theoretical background of deep learning, reinforcement learning and its algorithms. Chapter 3 explains the implementation of the whole system, from data gathering, data preprocessing, to building the deep learning model and the reinforcement learning architecture needed to train the text database. Chapter 4 discusses the result obtained from chapter 3 and evaluates the system's quality, along with data visualization for better insights. Last but not least, chapter 5 concludes the research with generalized claims and findings from chapter 3 and 4.

2. BACKGROUND

This chapter explains key concepts of the research: deep learning as well as reinforcement learning concepts. In terms of deep learning, the structure of the Recurrent Neural Network (RNN), particularly Long Short-term Memory RNN [2], will be elaborated. Deep learning frameworks used in the research will also be mentioned and explained briefly. As regards reinforcement learning, key definitions such as policy, environment and reinforcement learning algorithms are prerequisites to understanding: i) how reinforcement learning works and ii) the mechanics of the solution model presented in later chapters.

2.1 Text generation

Text generation is one of many tasks in the field of NLP. One of the earlier models for text generation was the Seq2Seq [10] language model, which uses two RNNs to predict the next text sequence from a previous one. However, there were certain limitations to the quality of the text produced by a Seq2Seq: the generated text is often nonsense and incorrectly spelled. A more efficient model later introduced was the Word2Vec model [11], which treats sequences of words as vectors. The aforementioned models served as a basis for most State Of The Art (SOTA) text generation models nowadays [7]. One of the most notable text generation models is GPT-3 [8], which has approximately 175 billion parameters, 10 times more than a regular model. GPT-3 has the ability to produce newspaper articles that are nearly indistinguishable when compared to human written articles [8]. Another popular text generation model worth mentioning is the Bidirectional Encoder Representations from Transformers, also known as BERT [17], which was developed by Google in 2018. BERT achieved SOTA performances on tasks such as questions answering as well as language inference (language inference means determining if a "hypothesis" is true or false) [17].

2.2 Deep learning

Deep learning is a subset of machine learning and one of its algorithms. Applications of deep learning include machine translation, self-driving cars, digital marketing, and so on. At the most fundamental level, deep learning models are artificial neural networks with multiple layers. While traditional neural networks often consist of 2-3 layers, the number of layers in a deep learning model can go up to 150.

An artificial neural network consists of an input layer, a number of hidden layers and an output layer. Each layer's output (excluding output layer's) is fed through an activation function. An activation function is defined as a weighted function of an input neuron and the bias term which has the purpose of eliminating unnecessary neurons in a neural network layer [9]. Activation functions can be either linear or non-linear. The activation functions used in the supervised model which will be defined in Section 3.2 are ReLU and Softmax. Both ReLU and Softmax are non-linear activation functions and can be defined as Equation 2.1 and 2.2 respectively:

$$f(x) = max(0, x) \tag{2.1}$$

$$f(x_i) = \frac{exp\,(x_i)}{\sum_j exp\,(x_j)} \tag{2.2}$$

In terms of performance, loss function is used to evaluate a model, as well as the error between the actual label y and predicted label $\hat{y}$. The loss function which will be used in the model defined in Section 3.2 is the categorical crossentropy loss function, which can be expressed as Equation 2.3:

$$L(y, \hat{y}) = \sum_{j=0}^{M} \sum_{i=0}^{N} (y_{ij} * log(\widehat{y_{ij}})) \tag{2.3}$$

where $\hat{y}$ is a one-hot encoded prediction vector and y is the actual ground truth distribution. One-hot encoding is a process where categorical data is transformed into a binary label matrix. The purpose of one-hot encoding is to convert categorical data into numeric data, making it easier for machine learning algorithms to process.

To train a model, we use optimizers. Optimizers are algorithms or methods used to update a model's weight or learning rate in order to reduce a model's loss. The optimizer which will be used in the model defined in Section 3.2 is the Adaptive Moment Estimation (ADAM) [12] optimizer. ADAM is an optimization algorithm which is considered an upgraded version of the Stochastic Gradient Descent (SGD) algorithm. ADAM is effective and often outperforms other optimization algorithms in deep learning applications [12].

2.3 Recurrent neural network and Long short-term memory RNN

A recurrent neural network (RNN) is a type of artificial neural networks that has a more complex structure than traditional neural networks. It is often used for natural language processing problems such as speech recognition, text classification, and so on. In an RNN architecture, neurons that are connected to each other form a loop, which means that the RNN utilizes both feedforward and feedback structure of a neural network. Hence, unlike traditional feed-forward neural networks, recurrent neural networks store their states after processing a sequence of input and thus use their internal states to process future inputs [5].

The supervised model which will be defined in Section 3.2 uses a special type of RNN: Long short-term memory RNNs (LSTM) [2]. Therefore, understanding how standard RNNs work is the prerequisite to understanding the structure of LSTMs. The architecture of an RNN is dynamic, thus, it keeps track of an internal state within each step of the network. The principle of each hidden layer in RNN is that after getting put into the activation function, the outputs will be saved in “context cells”, which will be fed back to the corresponding hidden neuron of the previous layer.

In principle, having loops inside an RNN means that it has access to the previous state of the model. For example, when training a character-based text generation model, having access to previous output characters increases meaningfulness of the generated text. However, there are limitations to standard recurrent neural networks. In terms of solving problems that require learning long-term temporal dependencies, such as text generation where the gap between the context and the output is considerably large, recurrent neural networks are proved to be incapable [18]. While training a neural network, gradient descent is used to optimize the network’s parameters. As the gradients get backpropagated across multiple timesteps, they explode, causing a problem called vanishing gradient. Vanishing gradient problem makes the process of optimizing the network parameters more difficult and sometimes impossible when gradients are close towards zero.

Thus, a new RNN structure was made to solve the vanishing gradient problem, as well as provide the capability to learn long-term dependencies. Long Short Term Memory recurrent neural networks (LSTM) were introduced by Hochreiter and Schmidhuber in 1997 [2]. In principle, LSTMs are able to solve vanishing gradients by giving access to the forget gate's activation, thus having more control of the network's gradients at each time step.

In terms of RNN and LSTM's structures, based on Figure 2-1 and 2-3, it can be seen that LSTMs have a more complex structure than standard RNNs.

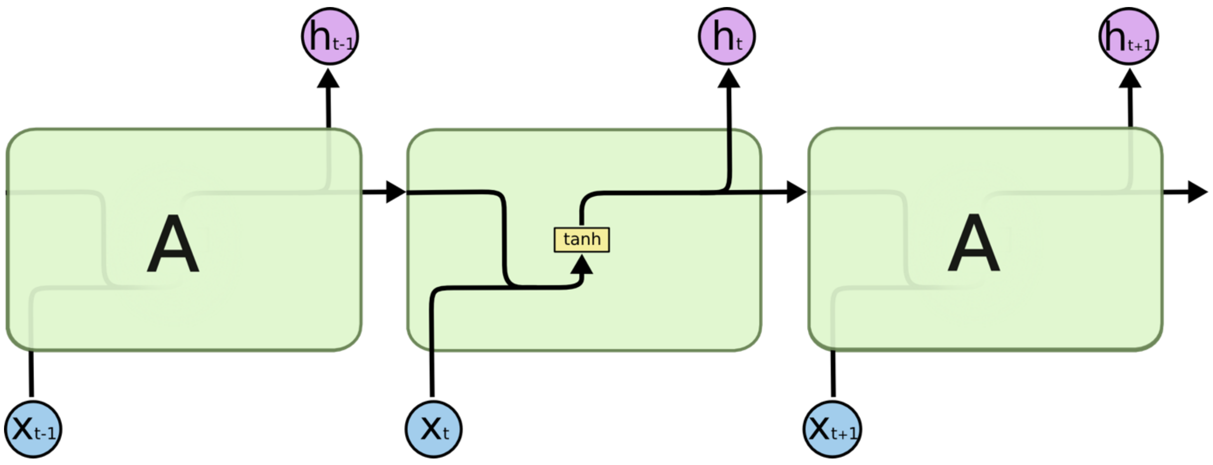


Figure 2-1 *Repeating structure of a standard RNN*

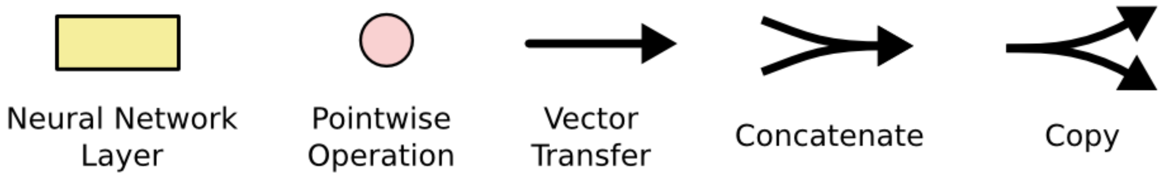


Figure 2-2 *Notation*

As regards recurrent neural networks, their structures are relatively simple. According to Figure 2-1, their repeating module consists of a single layer, usually a tanh layer, which maps the output to the range from -1 to 1. This helps to control the amount of new information that the network can absorb.

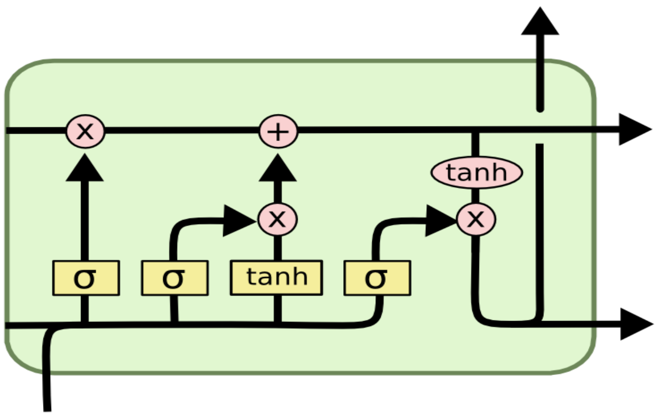


Figure 2-3 *Structure of an LSTM cell*

On the other hand, an LSTM cell contains many more operations than a standard RNN cell, which enables long-term dependencies. There are two parts in an LSTM cell: cell state and activation gates. The cell state is the layer controlling the flow of information within the cell, which can be seen as the line in Figure 2-4.

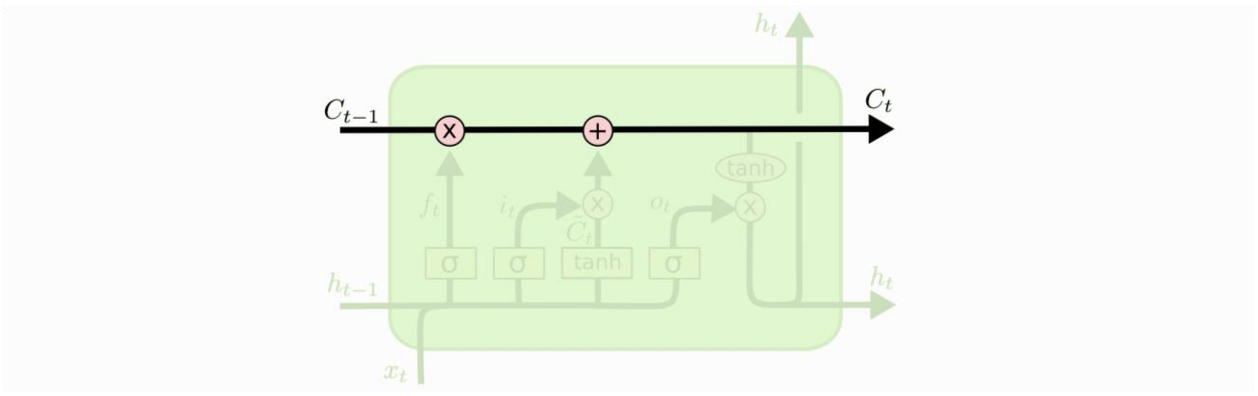


Figure 2-4 *The cell state in an LSTM cell*

An LSTM cell consists of three gates. The first gate is considered the “forget gate”, which determines how much previous information is kept when outputting new information. A sigmoid layer maps the input to the range from 0 to 1, thus calculates the forget rate of previous information according to Equation 2.4 [2]:

$$f_t = \sigma(W_t.[h_{t-1}, x_t] + b_f, \qquad (2.4)$$

where σ is the sigmoid function, $[h_{t-1}, x_t]$ are the inputs and W_t, b_f are the parameters. The output of this gate is a scalar ranging from 0 to 1, where 1 means keeping all the information and 0 means keeping none of the previous information [2].

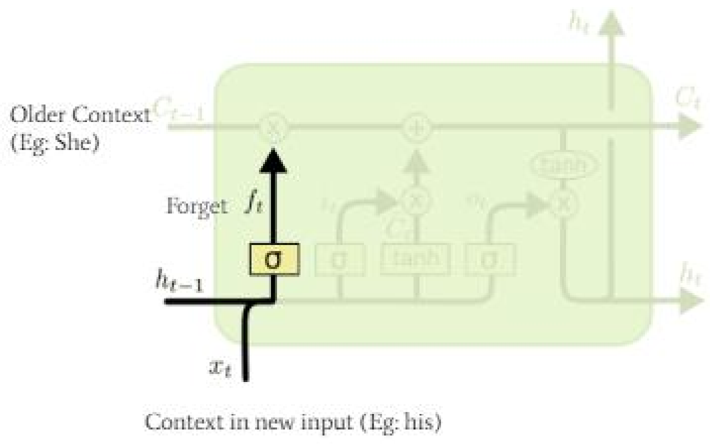


Figure 2-5 *Structure of the forget gate in an LSTM cell*

After passing the information through the “forget gate”, new information needs to be processed. This new information processing gate consists of two parts. Firstly, a sigmoid layer is needed in order to determine the amount of information that will be updated. Secondly, a tanh layer will be used to create a candidate vector $\tilde{C}_t$. Generally, the gate equations can be expressed in Equation 2.5 and 2.6 [2]:

$$i_t = \sigma(W_i.[h_{t-1}, x_t] + b_i \tag{2.5}$$

$$\tilde{C}_t = tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \tag{2.6}$$

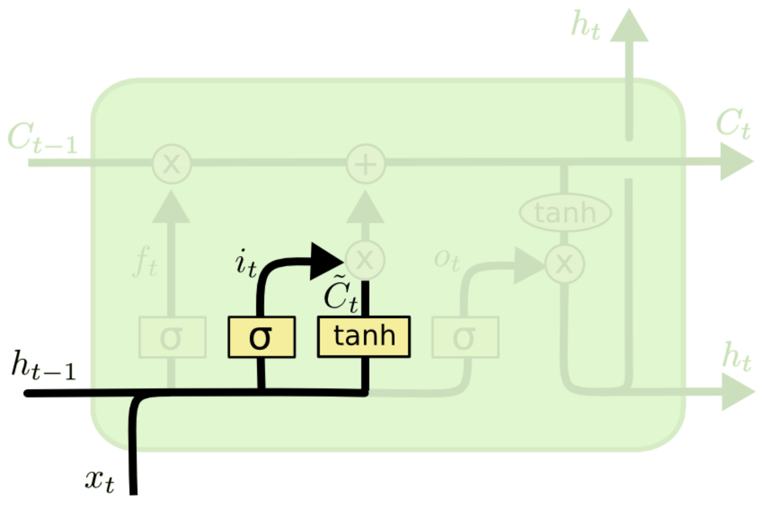


Figure 2-6 *Structure of the "new information" gate in an LSTM cell*

Last but not least, an output gate determines the output of the cell state. The structure of the output gate is relatively similar to that of the aforementioned second gate, with a sigmoid activation layer and a tanh layer to control which part of information is kept. However, the cell state C_{t-1} is fed through the tanh layer instead of the output of the sigmoid layer, which is eventually pointwise multiplied by the gate's sigmoid layer to get the output h_t [2]:

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \tag{2.7}$$

$$h_t = o_t * tanh(C_t) \tag{2.8}$$

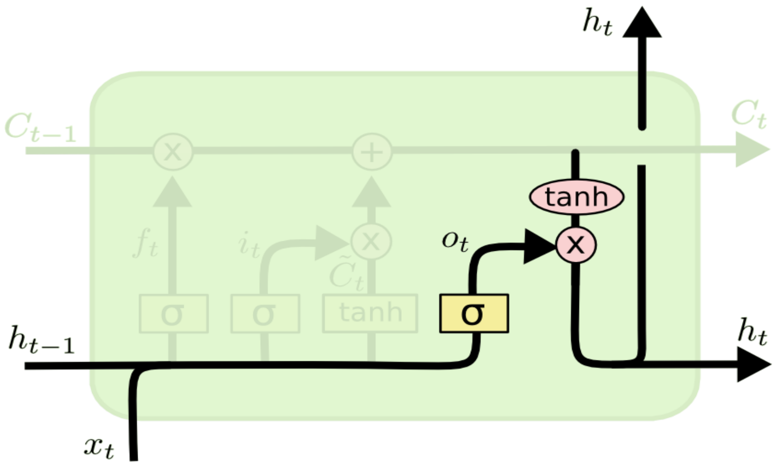


Figure 2-7 *Structure of the output gate in an LSTM cell*

2.4 Reinforcement Learning

Reinforcement learning is a subset of machine learning, one of three machine learning problems, including supervised and unsupervised learning. The objective of reinforcement learning is to utilize an agent in a specific environment and use it to optimize a certain cumulative reward by learning a good strategy to perform actions.

Understanding key concepts of reinforcement learning is the prerequisite to understanding how the system this research will propose can be improved using virtual feedback. In a reinforcement learning problem, there is an environment, which can be defined by an arbitrary model. An agent which acts on an environment has information about its state and an array of actions it can take. Based on the agent's chosen action, it receives a scalar reward from the environment known as feedback. However, the agent may fully understand the environment, or does not have any information about the environment model at all. Hence, it is essential that the agent balances between exploration and exploitation.

Policies can be developed with respect to an agent and the environment it acts on. A policy $\pi(s)$ can be defined as a set of actions for an agent to take in order to maximize the feedback reward [6]. Mathematically, a deterministic policy can be expressed as a function of state s which outputs action a:

$$\pi(s) = a \tag{2.9}$$

In the case of stochastic policy:

$$\pi(a|s) = P_\pi[A = a \mid S = s] \tag{2.10}$$

A value function is used to assess the current state – to see if the current state is good or not. Basically, the value function predicts future reward and is expressed by Equation 2.11:

$$v_\pi(s) = E_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \mid S = s], \tag{2.11}$$

where $v_\pi(s)$ is the value function, E_π is the environment in which the policy π is used and R_t is the reward the agent receives at timestep t. This is used to select a suitable action for the agent to take next.

In terms of reinforcement learning agents, there are numerous types of agents: policy-based, value-based, actor critic, model-free and model-based. A model is defined as a prediction of the environment, which can be split into two functions, P and R. P is a probability function of a next state given the current state and action, whereas R is a probability of a reward given the current state and action:

$$P_{ss'}^a = P[S_{t+1} = s' \mid S_t = s, A_t = a] \tag{2.12}$$

$$R_s^a = E[R_{t+1} \mid S_t = s, A_t = a] \tag{2.13}$$

In general, reinforcement learning can be seen as a method of trial and error. There are two phases of this method of learning: exploration and exploitation. Combining these two phases will help agents to find a better policy to maximize the obtained reward. Particularly, exploration means obtaining information from the environment; it is usually done when an agent has no prior or little information about the environment. On the other hand, exploitation means taking advantage of known information in order to optimize the reward. These two phases are critical in the task of discovering the optimal policy.

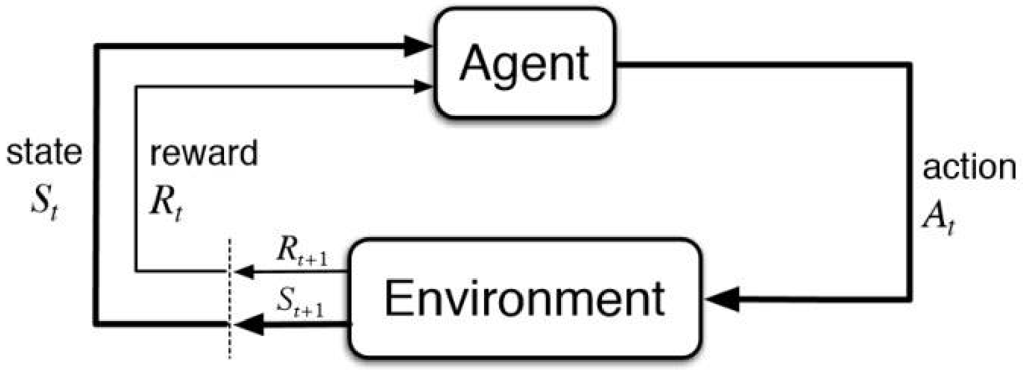


Figure 2-8 *Key concepts of Reinforcement Learning (Agent performs actions based on its internal state and the observed environment. Environment then processes the action, updates the environment state accordingly and provides a reward to the agent)*

2.4.1 Deep Q-Learning

There are two types of reinforcement learning algorithms: model-free and model-based. A model-free algorithm learns solely from rewarded actions, meaning that unrewarded experiences have no impact to the agent's learning process. On the other hand, a model-based algorithm takes into account both rewarded and unrewarded experiences, which is proved to be more efficient most of the time [14]. One of the most widely applied RL algorithms is Q-learning. Q-learning is a model-free RL algorithm, whose main objective is to estimate Q-values so as to form an optimal policy [15]. The learning method of Q-learning utilizes the concept of temporal differences. Temporal difference (TD) is defined as a form of learning in which the agent tries to learn different actions for a specific state and evaluate the consequences of each action in order to choose the best action for each state. To evaluate the value of each action, the concept of Q-value was introduced. A Q-value $Q^{\pi}(s,a)$ is the expected discounted value that an agent following a policy π can be rewarded at state s if it takes action a. Mathematically, the Q-value can be updated using Equation 2.14:

$$Q^{\pi}(s,a) = r(s,a) + \gamma \max_{a} Q(s',a), \tag{2.14}$$

where $r(\boldsymbol{s},\boldsymbol{a})$ is the immediate reward that the agent receives after taking action a at state s, γ is the discount factor and $\max_{a} Q(s',a)$ is the maximum reward that the agent can receive by taking action a at next state s'. The discount factor γ determines the importance of future rewards, in which case $\max_{a} Q(s',a)$**,** to the current state s.

In order to use Q-learning in more complex tasks such as NLP or dealing with high dimensional data, the Deep Q-learning algorithm was introduced, which applies Q-learning to neural networks [16]. While traditional Q-learning algorithms normally used low dimensional state spaces and handcrafted data, Deep Q-learning has been proved to work well with high dimensional sensory inputs. [16]. During training, the environment's state is processed by the model, producing a Q-value for each action that can be taken. The current state s, next state s', action a and reward $r(s,a)$ from each timestep are stored in a replay buffer. During the algorithm training process, minibatches of the replay buffer are sampled, then Q-learning updates are applied to the sampled experience. This is called 'experience replay', which has some advantages over traditional Q-learning algorithm. First of all, the former is more data efficient, since each experience is used in multiple updates. Second, that the experiences are randomly sampled, the variance of the Q-learning updates is reduced. Generally, the DQN algorithm can be written as in Algorithm 1:

Algorithm 1 Deep Q-Learning algorithm

```
Initialize replay memory D to capacity N
Initialize Q-table Q with random weights
for episode = 1, E do
    Initialize state S1
    for timestep t = 1, T do
        Select action a based on probability $\epsilon$ or select a = $max_a Q(S1', a)$
        Execute step function for action a and evaluate reward r
        Store state S1, next state S2, reward r, action a in D
        Sample random minibatch of transitions (S1, S2, r, a) from D
        Update Q using Equation 2.11
        Perform Gradient Descent step
    end for
end for
```

3. IMPLEMENTATION

This chapter will explain the procedure of building a text generation model from a database and apply reinforcement learning algorithms onto it. Generally, the process can be outlined as the following: i) Building a DNN-based text generator using a large text database) and ii) Adapting the model to another dataset in order to produce specific type of text using reinforcement learning-based model adaptation.

3.1 Important frameworks

The programming language used in the research was Python. The frameworks used in order to train our text generation model are Tensorflow and Keras. Tensorflow is a framework providing tools for machine learning applications [3], while Keras is a high-level API for building neural networks, which runs on top of Tensorflow [4]. Keras was used for the creation of the LSTM model for generating text. Additionally, Keras also provided helper built-in functions for data preprocessing such as to_categorical function to one-hot encode the data labels, or functions to configure the text generation model.

On building a reinforcement learning system to improve text generation quality, Keras was used to implement the DQN algorithm.

3.2 Dataset and pre-processing

Before constructing an LSTM model using Keras, a text database must be prepared and processed. For this research, two novels' text from Project Gutenberg were used: 'Alice's Adventures in Wonderland' by Lewis Carroll and 'Pride and Prejudice' by Jane Austen. The text can be found on the respective website, collected into a text file for Python to easily read and process.

In terms of data preprocessing, after being read by Python, the text string will be filtered by omitting all punctuations and symbols so that only alphanumerical characters are chosen. Hence, there were a total of 39 unique characters in the text database.

Furthermore, the text was transformed into lowercase. To fit the model onto the processed text database, data must be split into a training set and a test set. Our proposed LSTM model takes a sequence of text as an input. Hence, for each training and test set, the input data (known as x) are a sequence of 40 characters, whereas their labels (known as y) will be the next character of the following sequence. However, since neural networks are unable to process characters or strings as inputs, characters were one-hot encoded.

3.3 Defining the supervised model

Our text generation model first consists of two LSTM layers of 128 hidden units each. The output of the LSTM layers will be put into a fully connected layer with 50 neurons, with the ReLU activation function. Finally, the activated neurons from the fully connected layer will be put into a classification layer which has the number of neurons equal to the number of distinct characters found in the text database. The activation function used in the classification layer was softmax**.** However, instead of choosing the highest probability class as the output character, temperature-based sampling was used to increase the variability of the outputted text. In terms of optimization methods, categorical crossentropy loss function was used to evaluate the model, along with the ADAM optimizer algorithm to update the model's weights.

Finally, the model was fit onto the training dataset and trained for 200 epochs with batch size of 128. Batch size is defined as the number of samples that are used in the process of gradient calculation simultaneously, which can then be used as a basis for one weights update. On the other hand, the number of epochs is defined as a number of times the training dataset is passed through the model.

3.4 Adapting the model using Reinforcement Learning

This section discusses how a supervised LSTM model trained on a relatively large text dataset, was adapted to generate text that satisfies different criteria using reinforcement learning algorithms.

3.4.1 Deep Q-learning

For this experiment, the derivation of our Deep Q-Learning algorithm was based on the algorithm described in Algorithm 1. However, the action selection method during each timestep was not $\epsilon - greedy$, but instead to use temperature-based sampling from the output of our supervised model. In terms of hyperparameter, the number of episodes of training and the number of timesteps in each episode were 100 and 1000 respectively. The interval between each experience replay process was 40, which means that the algorithm performed Q-learning updates every 40 timesteps. The DQN algorithm for the experiment can be described in Algorithm 2:

Algorithm 2 Deep Q-Learning to improve text generation from supervised model

```
Initialize replay memory D to capacity N
Initialize Q-table Q with random weights
Initialize reference model
for episode = 1,100 do
    Initialize state S1
    for timestep t = 1,1000 do
       Select action a = argmax(model.predict(S1))
       Execute step function for action a and evaluate reward r
       Store state S1, next state S1', reward r, action a in D
       if t divisible by 40 do
           Sample random minibatch of transitions (S1, S1', r, a) from D
           Update Q using Equation 2.10
           Perform Gradient Descent step
           update model weights
      end if
    end for
end for
```

3.4.2 Experimental setup

To test the efficiency of the described reinforcement learning algorithm, several experiments with different reward functions were conducted, along with different values of discount factor gamma.

The model whose weights were updated was taken directly from the supervised model which was introduced and trained in Section 3.3.

In terms of the reinforcement learning training process, weights of all the supervised model's layers were updated after each training episode, which consists of 1000 timesteps, corresponding to 1000 actions produced. The Huber loss was used to perform gradient descent on the model, whereas the ADAM optimizer was used to update the model's weights.

Before each training process, the reset function was called in order to determine the initial state of the environment. For each reward condition, the agent was trained for 100 episodes. This section mathematically defines the reward functions used in the experiments.

3.4.2.1 Simple reward function

For the simplest reward function, the agent learned to only generate one specific character. Assume the character that would be generated is 'c', which corresponds to action number 12 in the present character encoding scheme, the reward can be defined as Equation 3.1:

$$R_s^a = \begin{cases} 5 \ if \ a = 12 \\ 0 \ if \ a \neq 12' \end{cases} \tag{3.1}$$

where R_s^a is the reward at state s for action a.The aim of this reward function was to test whether the proposed Algorithm 2 works fundamentally. This reward function also served as a basis for more complex reward functions defined in Section 3.4.2.2 and 3.4.2.3.

3.4.2.2 Negative distance reward function

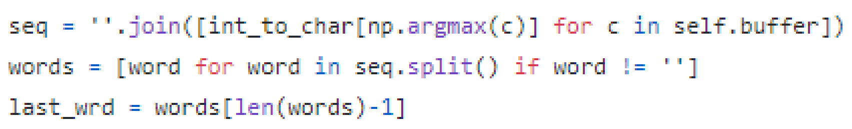
Like the condition presented in Section 3.4.2.1, the reward function can be defined as the negative distance between the last two generated character in the environment's state. The 'distance' is defined as the difference between the mapped integer values of the characters. The equation for this reward function can be expressed as Equation 3.2:

$$R_s^{a_t} = -(abs(a_t - a_{t-1})), \tag{3.2}$$

where $R_s^{a_t}$ is the reward at state s for action a at timestep t and a_t and a_{t-1} are actions at timestep t and $t-1$ respectively. It can be easily seen that the agent's goal for this reward function is similar to that of Section 3.4.2.1. Specifically, this reward function implies that, in order to maximize the reward, the agent learns to output the same character every timestep. Hence, the highest reward obtained would be 0. This reward function was implemented to evaluate the ability of the RL agent to process reward conditions that depend on temporal structure of the produced character strings.

3.4.2.3 Producing meaningful English words

The main objective of this experiment was to improve the quality of text generation obtained from Section 3.3. Hence, in order to test the text generation quality, the agent was rewarded for producing meaningful English words. The evaluated word was determined by first converting the environment's state into a string (or sentence), then extracting the last word from the converted string.



```
seq = ''.join([int_to_char[np.argmax(c)] for c in self.buffer])
words = [word for word in seq.split() if word != '']
last_wrd = words[len(words)-1]
```

Figure 3-1 *The last outputted word was defined in terms of whitespaces then verified by NLTK-based critic whether it was an English word or not*

In terms of reward evaluation, the agent was rewarded if the last produced word was a valid English word. Particularly, the NLTK framework was used to check for valid English words. NLTK is a Python framework mainly used to build applications in order to work with human language data [13]. For this experiment, the environment rewarded the agent if the last produced word belongs in the NLTK *words* database. The reward function can be defined mathematically in Equation 3.3:

$$R_s^a = \begin{cases} 5\ if\ last\ word\ \in NLTK\ words\ database \\ 0, if\ otherwise \end{cases}, \qquad (3.3)$$

where R_s^a is the reward at state s for action a.

4. RESULTS AND DISCUSSION

A sample of text generated from the supervised model defined in Section 3.1 can be quoted: '*in the refuse, she was a time to see her at the convenient any of yourself and a most agreeably, that it was now as she could not have and the match for them, when i have not at all i hade not the bread teasures about it it is well. i wish yoo, you won an all this with, said the evently and then alice waited a confidence and after sitthing about the professions, and all the mock turtle settled a short, and i am sure you will not have the seeing on the subject and these informed that it was a very strong of the subject, in a sort of the whole party, whose elizabeth had been greated all consideration. i am sure, said alice i had all that she much as sear the probation of the subject of the family is the rest of the subject*'. This will be used for comparison to the experiments' outcome in order to evaluate the quality of reinforcement learning algorithm introduced in Section 3.3.1.

For each reward function defined in Section 3.3.2, general observations after training will be discussed, followed by a reward graph throughout episodes trained and the text generated after training is completed.

4.1 Simple reward function

The first reward function is to reward the agent when it produces just a specific character such as the letter 'a'. Particularly, during the training process, each time the agent, in which case the model, produces the character 'a', a reward value of 10 is given to the system, and -1 if the model outputs another character. After 100 training episodes, the training reward is described as the following:

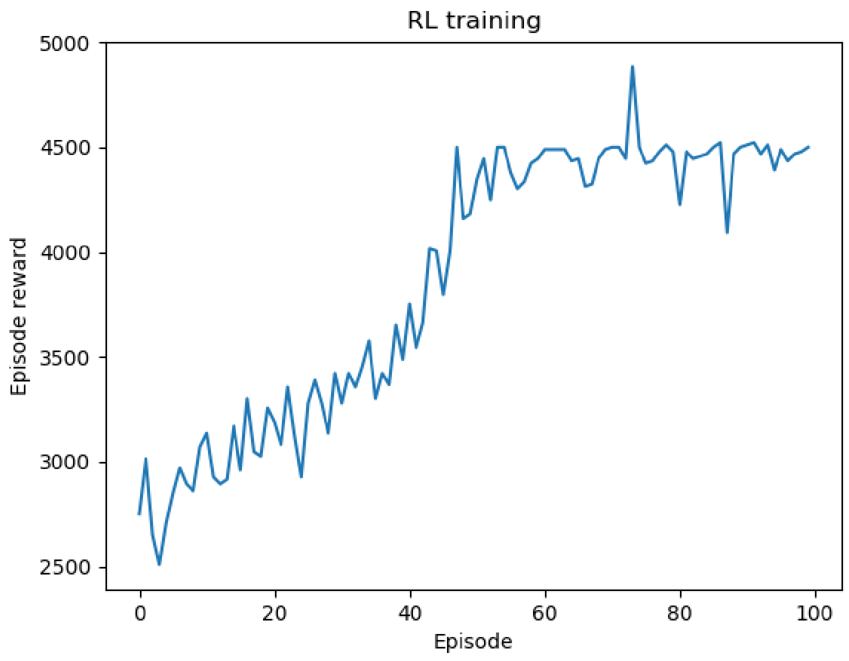


***Figure 4-1** Task's rewards after 100 training episodes (x-axis is the episodes trained, y-axis is the total reward of each episode trained)*

Generally, there is a significant increase in training reward during the training process, from 2750 to a maximum of 5000, which means that all the characters outputted in a training episode is the letter 'a'. In terms of the training time, it took a small amount of time to train the agent since this is only a simple reward function.

4.2 Negative distance reward function

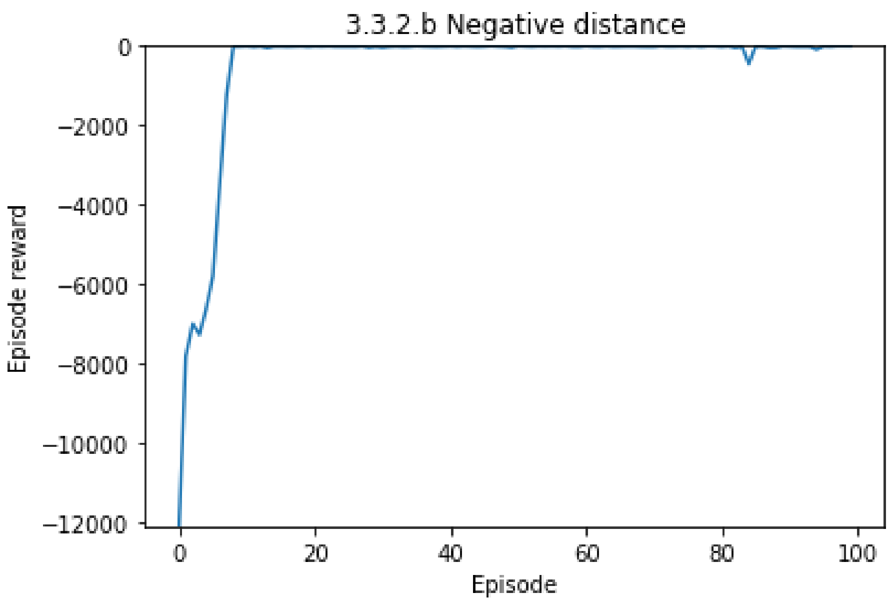


***Figure 4-2** Negative distance reward of 100 episodes (x-axis is the episodes trained, while y-axis is the negative distance between the last two outputted characters, which was mathematically defined in Section 3.4.2.2)*

Results for the experiment using the second reward function, which was mathematically defined in Section 3.4.2.2, can be found in Figure 4-3. Overall, the total reward at the beginning of the training process was relatively low, as the agent was still producing text similar to our text database. However, after approximately 10 episodes, the reward skyrocketed to 0, which means that the agent learned to maximise the environment's reward well, as seen in Figure 4-3. Hence, this experiment proves that the agent adapts well to reward functions that depend on temporal modelling of the data.

4.3 Producing meaningful English words (a more complex reward function)

The starting reward of the agent is already relatively high, since the supervised model is considered sufficiently well-trained. It can be seen from Figure 4-4 that the episode reward throughout the training process fluctuates around 5000 which is the maximum reward. The outputted text after the training process, despite producing correct English words, is rather repetitive and nonsense. A sample of the generated text after training is '*to the the the the tabte on the the the tabble the the the the the tan tabby the tabmy tabt the the the the tabby tabuts and the the the the thost to the tabby the the the the tabbed'*. However, the experiment tested the feasibility of the RL algorithm by tackling one sub-problem, which is checking and improving the validity of individual words. This approach came at the cost of reducing the quality of other aspects such as grammaticality or diversity of words.

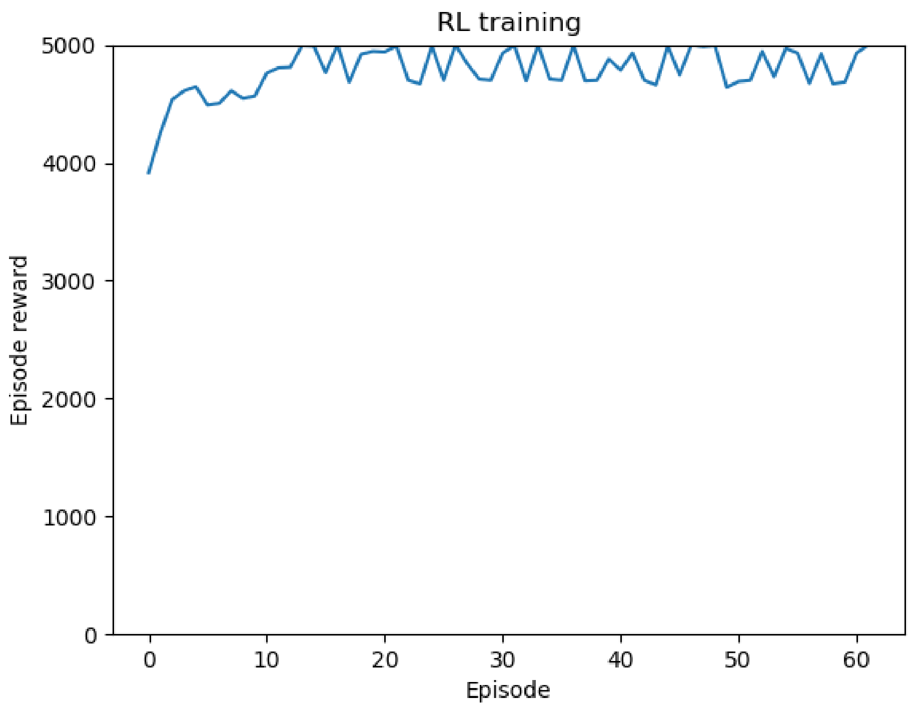


Figure 4-3 *Rewards after 60 episodes for NLTK validity reward function (x-axis is the episodes trained, while y-axis is the reward of each episode trained). For this reward function, only 60 episodes were trained due to computational limitations.*

Other minor reward criteria were introduced into the reward function. For example, the RL agent can learn not to produce the same words twice or not to produce the word ‘the’. Their reward graphs are similar to Figure 4-4; thus, the agent adapted well to both reward functions.

5. CONCLUSION

The present study introduced and evaluated a text generation system which utilized both supervised and reinforcement learning. Results obtained from Chapter 4 shows that the LSTM model defined in Chapter 3 adapts well to different RL reward functions, generating text that satisfies specific criteria. However, that the episode reward went up during the training process did not imply that the quality of the generated text improved. Particularly, for the reward function defined in Section 3.4.2.3, the outputted text after training, despite most words are English, was far from meaningful. This was because the RL agent approached the problem by evaluating and received rewards one word at a time, which came at the cost of reducing the quality of the generated text overall.

The main challenge during the study was to design a suitable reward function for the RL agent. Hopefully, in the future, we can design an optimal reward function for the RL agent in different tasks, or design a loss function that can compensate for the quality reduction of the generated text as a whole, while the supervised model still adapts well to reward criteria.

REFERENCES

[1] Stephenson, I. *Understanding Deep Learning*. [online] methods.blog. Available at: https://methodsblog.com/2019/11/13/understanding-deep-learning/, 2019. [Accessed 14 Mar. 2020].

[2] Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural Computation*, *9*(8), pp.1735–1780, 1997. (See https://colah.github.io/posts/2015-08-Understanding-LSTMs/ for simple explanation).

[3] TensorFlow. (n.d.). *TensorFlow*. [online] Available at: https://tensorflow.org.

[4] Keras.io. (2019). *Home - Keras Documentation*. [online] Available at: https://keras.io.

[5] Schuster, M. and Paliwal, K.K. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, *45*(11), pp.2673–2681, 1997.

[6] Silver, D. (n.d.). *Lecture 1: Introduction to Reinforcement Learning Lecture 1: Introduction to Reinforcement Learning*. [online] Available at: https://www.davidsilver.uk/wp-content/uploads/2020/03/intro_RL.pdf [Accessed 12 Jan. 2021].

[7] Montesinos, D.M. Modern Methods for Text Generation. *arXiv preprint arXiv:2009.04968, 2020*.

[8] Brown, T.B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A. and Agarwal, S.. Language models are few-shot learners. *Advances in Neural Information Processing Systems 33 (NeurIPS), 2020*.

[9] Nwankpa, C., Ijomah, W., Gachagan, A. and Marshall, S. Activation functions: Comparison of trends in practice and research for deep learning. *2nd International Conference on Computational Sciences and Technologies (INCCST 20), 2020.*

[10] Sutskever, I., Vinyals, O. and Le, Q.V. Sequence to sequence learning with neural networks. *Advances in Neural Information Processing Systems, 3104–3112, 2014*.

[11] Mikolov, T., Chen, K., Corrado, G. and Dean, J. Efficient estimation of word representations in vector space. *ICLR, 2013*.

[12] Kingma, D.P. and Ba, J. Adam: A method for stochastic optimization. *ICLR, 2015.*

[13] Nltk.org. *Natural Language Toolkit — NLTK 3.4.4 documentation*. [online] Available at: https://www.nltk.org/. 2019 [Accessed 27 Jul. 2020]

[14] Pong, V., Gu, S., Dalal, M. and Levine, S.. Temporal difference models: Model-free deep rl for model-based control. *ICLR, 2018.*

[15] Watkins, C.J. and Dayan, P. Q-learning. *Machine learning*, *8*(3–4), pp.279–292, 1992.

[16] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D. and Riedmiller, M. Playing Atari with deep reinforcement learning. *NIPS Deep Learning Workshop, 2013.*

[17] Devlin, J., Chang, M.W., Lee, K. and Toutanova, K.. Bert: Pre-training of deep bidirectional transformers for language understanding. *NAACL, 2018.*

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