Anh Nguyen

Text generation using lstm and reinforcement learning

Bachelor's Thesis

|  |  |  |
| --- | --- | --- |
| Science and Engineering  Okko Rasanen  Joni Pajarinen  April 13th 2020 |  |  |

ABSTRACT

Anh Nguyen: Text Generation using LSTM and Reinforcement Learning

Bachelor of Science Thesis

Tampere University

International Degree of Science and Engineering

December 27th 2020

Text generation tasks are becoming more and more prominent in applications such as machine translation, image captioning, dialogue system, etc. Unfortunately, to be able to generate meaningful outputs, text generation systems often require an extremely large amount of data. This Bachelor’s thesis introduces an approach of building a text generation application that compensates for the lack of data. The thesis first discusses training a relatively small text dataset using supervised learning. The system is then fine-tuned by applying specific reinforcement learning techniques, in order to make the system generate text that meets specific criteria. Finally, the quality of text generated after reinforcement learning is compared with that of a text generation system using only supervised learning.

Keywords: deep learning, lstm, rnn, reinforcement learning, deep q-learning, keras, tensorflow, text generation

The originality of this thesis has been checked using the Turnitin OriginalityCheck service.

Preface

I would like to thank my thesis supervisors, assistant professor Okko Rasanen 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, 13th April 2020

Anh Nguyen

Contents

[1. Introduction 1](#_Toc59532053)

[2. BACKGROUND 3](#_Toc59532054)

[2.1 Deep learning 3](#_Toc59532055)

[2.2 Recurrent neural network and Long short-term memory RNN 4](#_Toc59532056)

[2.3 Reinforcement Learning 9](#_Toc59532057)

[2.4 Important frameworks 11](#_Toc59532058)

[3. IMPLEMENTATION 12](#_Toc59532059)

[3.1 Dataset and pre-processing 12](#_Toc59532060)

[3.2 Defining the supervised model 13](#_Toc59532061)

[3.2.1 Generating text 15](#_Toc59532062)

[3.3 Improving the model using Reinforcement Learning 15](#_Toc59532063)

[3.3.1 Deep Q-learning 16](#_Toc59532064)

[3.3.2 Experimental setup 17](#_Toc59532065)

[3.3.3 Jensen – Shannon divergence 19](#_Toc59532066)

[4. RESULTS AND DISCUSSION 20](#_Toc59532067)

[5. CONCLUSION 23](#_Toc59532068)

[REFERENCES 24](#_Toc59532069)

List of Figures

[Figure 2‑1 Repeating structure of an RNN 5](#_Toc60670666)

[Figure 2‑2 Notation 5](#_Toc60670667)

[Figure 2‑3 Structure of an LSTM cell 6](#_Toc60670668)

[Figure 2‑4 Structure of the forget gate 6](#_Toc60670670)

[Figure 2‑5 Structure of the "new information" gate 7](#_Toc60670673)

[Figure 2‑6 Structure of the output gate 8](#_Toc60670674)

[Figure 2‑7 Machine Learning Paradigms 9](#_Toc60670675)

[Figure 2‑8 Key concepts of Reinforcement Learning 10](#_Toc60670676)

[Figure 3‑1 LSTM model for text generation using Keras 15](#_Toc60670677)

[Figure 3‑2 Extracting the last word for reward evaluation 18](#_Toc60670678)

[Figure 4‑1 Generated text from supervised model 20](#_Toc60670679)

[Figure 4‑2 Task's rewards after 100 training episodes 21](#_Toc60670680)

[Figure 4‑3 Negative distance reward of 100 episodes 21](#_Toc60670681)

[Figure 4‑4 Rewards after 100 episodes for NLTK validity reward function 22](#_Toc60670682)

[Figure 4‑5 Outputted text after training 22](#_Toc60670683)

[Figure 4‑6 Outputted text after training for task 1.1 23](#_Toc60670684)

List of Symbols and abbreviations

RNN Recurrent Neural Networks

LSTM Long Short-Term Memory

RL Reinforcement Learning

CNN Convolutional Neural Networks

NLTK Natural Language Toolkit

SGD Stochastic Gradient Descent

KLD Kullback-Leibler divergence

JSD Jensen-Shannon divergence

DQN Deep Q-Learning

.

# Introduction

The ability for machines and computers to understand and use human’s languages has been one of the most popular topics in the field of artificial intelligence. Interactions between machines and humans have been extremely difficult, since machine code (or language) is completely incomprehensible. Hence, the field natural language processing (NLP) is derived, with the aim to help computers interact with humans and vice versa. There are useful applications of NLP, including 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 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 key concepts or theoretical background of deep learning, reinforcement learning and its algorithms. Moreover, a popular deep learning architecture is also discussed, followed by a brief explanation of the frameworks and libraries used in this research. 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.

# 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, 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.

## Deep learning

Deep learning is precisely a subset of machine learning and one of its algorithms. It replicates the human brain to perform tasks such as data processing, pattern recognition and information interpretation. The most fundamental level of deep learning models are neural networks with multiple layers [1]. However, the number of layers in a traditional network is around 2-3 layers while that of deep learning models can be up to 150 layers. Applications of deep learning include machine translation, self-driving cars, digital marketing, and so on. However, the purpose of this research is towards the natural language processing application of deep learning, specifically generating text from an existing database and thus improving the model.

Implementation of deep learning models can be done using various frameworks and libraries such as Tensorflow, Keras API, Pytorch, and so on. The libraries which are used in this research are Tensorflow and Keras. Tensorflow is an open-source end-to-end framework which supports building machine learning or state of the art machine learning models [3]. On the other hand, Keras is a high-level API for building neural networks, which runs on top of Tensorflow [4].

## 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. 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].

Understanding how RNNs work is the prerequisite to understanding the structure of Long short-term memory RNNs (LSTM). The architecture of an RNN is dynamic, thus, it keeps track of an internal state within each step of the network. The most well-known RNN architecture is called “Elman Network” which consists of three layers. 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.

Theoretically, 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 (or sequence) increases the literacy (or meaningfulness) of the generated text. However, there are limitations to 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(ref). Moreover, while training the neural network, gradient descent is often used to optimize the network’s parameters. As the structure of an RNN becomes more complex, gradients of the network’s output with respect to the network’s parameters will at some time becomes significantly small, thus causing a problem called vanishing gradient problem. This makes the process of optimizing the network parameters more difficult and eventually impossible.

Fortunately, 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 networks (LSTM) was introduced by Hochreiter and Schmidhuber in 1997. Theoretically, 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, it can be easily seen that LSTMs have a more unique and complex structure.

A close up of a sign

Description automatically generated

Figure 2‑1 Repeating structure of an RNN

A picture containing object, clock

Description automatically generated

Figure 2‑2 Notation

As regards recurrent neural networks, their structures are relatively simple. 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.

A drawing of a cartoon character

Description automatically generated

Figure 2‑3 Structure of an LSTM cell

On the other hand, an LSTM has a much more complex structure, which allows long-term dependencies and lets information through in a cleverer way. There are two parts in an LSTM cell: the cell state and activation gates. The cell state is the topmost layer, controlling the flow of information within the cell, which can be seen as the top line in the above figure.

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 2.1 [2]:

, (2.1)

where is the sigmoid function, are the inputs and are the parameters. The output of this gate is from 0 to 1, where 1 means keeping all the information and 0 means keeping none of the previous information [2].

A picture containing clock

Description automatically generated

Figure 2‑4 Structure of the forget gate

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 . Generally, the gate equations can be expressed in Equation 2.2 and 2.3 [2]:

(2.2)

(2.3)

A picture containing clock

Description automatically generated

Figure 2‑5 Structure of the "new information" gate

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 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 [2]:

(2.4)

(2.5)

A picture containing clock, meter

Description automatically generated

Figure 2‑6 Structure of the output gate

## 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 research model 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, an array of action it can take, then 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 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 which outputs action a:

(2.6)

In the case of stochastic policy:

(2.7)

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 2.8:

, (2.8)

where is the value function, is the environment in which the policy is used and is the reward. 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, and . is a probability function of a next state given the current state and action, whereas is a probability of a reward given the current state and action:

(2.9)

(2.10)

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.

A close up of a logo

Description automatically generated

Figure 2‑7 Machine Learning Paradigms

A close up of a logo

Description automatically generated

Figure 2‑8 Key concepts of Reinforcement Learning

### 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] (Pong et al. 2018, 1802.09081). One of the most widely applied RL algorithms is the Q-learning. Q-learning is a model-free RL algorithm, whose main objective is to estimate the 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 is the expected discounted value that an agent following a policy can be rewarded at state if it takes action . Mathematically, the Q-value can be updated using Equation 2.10:

, (2.11)

where is the immediate reward that the agent receives after taking action at state , is the discount factor and is the maximum reward that the agent can receive by taking action at next state . The discount factor determines the importance of future rewards, in which case **,** to the current state .

In order to use Q-learning in more complex tasks such as NLP or dealing with high dimensional data, Volodymyr et al. (2013) introduced the Deep Q-Learning algorithm, which applies Q-learning to neural networks. While traditional Q-learning algorithms normally used low dimensional state spaces and handcrafted data, it is proved that the Deep Q-Learning algorithm works well with high dimensional sensory inputs. [16]. Deep learning models used in DQN are Convolutional Neural Networks (CNN). 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 , next state , action and reward 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 or select 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 2.10  Perform Gradient Descent step  **end for**  **end for** |

## Important frameworks

The programming language used in the research is Python. Apart from essential libraries for data processing and helper functions, there are two frameworks that are considered vital for creating a text generation model: 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 [4]. Keras will be used for the creation of the LSTM model for generating text. Additionally, keras also provides 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.

# 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) Finding a text database, ii) Building a deep learning model to study the database, iii) Applying deep reinforcement learning algorithm using Keras.

## Dataset and pre-processing

Before constructing an LSTM model using Keras, a text database must be prepared and processed. For this research, novels’ text from Project Gutenberg will be used. 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 values are chosen. Furthermore, the text will be transformed into lowercase. Two python dictionaries are created for the purpose of simplifying character referencing, one as a mapping from character to integer or index and vice versa. In order to fit the model onto the processed text database, data must be split into a training set and a test set (validation set). Keras’ LSTM model takes a sequence of text as an input. Hence, for each training and test set, the input data (known as x) will be a sequence of 40 characters, whereas their labels (known as y) will be the next character of the following sequence. However, since Keras’ neural networks are unable to process characters or strings as inputs, characters must be converted into numbers (or indexes), which can be done by utilizing the previous aforementioned python dictionary mapping character to integer. Subsequently, the training data can be one-hot encoded using Keras’ to\_categorical built-in function. One-hot encoding is a process where categorical data is transformed into a binary label matrix which will make machine learning algorithms, especially neural networks, perform better.

(Write more on one hot encode)

## Defining the supervised model

In terms of creating the text generation model, Keras’ Sequential model is used as a baseline, followed by 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. An activation function (or transfer 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, however, in this fully connected layer, the ReLU function is non-linear and can be represented by Equation 3.1:

(3.1)

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. As regards activation function, the softmax function is another non-linear function which calculates the probability distribution of each class; in this case, probability of each alphanumeric character. Softmax can be expressed as Equation 3.2:

(3.2)

This means that the probability of each character ranges from 0 to 1, thus, the sum of those probabilities will always be 1. Hence, the output character can be chosen by picking the class with the highest probability. That is why softmax is considered perfect for classification [8].

Overall, the structure of our text generation model is relatively small and simple, since only a computationally efficient yet straightforward model is needed for this task. However, to be able to prepare the model for training, it needs to be compiled. In Keras, compiling the model means setting up configurations for the model so as to optimize its training process, which in this case, defining the loss function and optimizer. In terms of loss function, since the network is a multi-class model, categorical crossentropy is chosen. Mathematically, categorical crossentropy loss is the difference between the distribution of the predicted output and that of the actual output (label) [10]. It can be expressed as Equation 3.3:

(3.3)

where is a one-hot encoded prediction vector and is the actual ground truth distribution. Hence, this loss is suitable for model evaluation when the model output is one-hot encoded.

On the other hand, another important configuration to consider is the model’s optimizer, in which case Adam optimizer is used. Optimizers in a Keras model are algorithms or methods used to update the weights or learning rate of the model so as to minimize the loss function [11]. Particularly, Adaptive Moment Estimation (Adam) is an optimization algorithm which is considered an upgrade version of the Stochastic Gradient Descent (SGD) algorithm. It is effective and often outperforms other algorithms in deep learning applications [12]. Keras sets default parameters for its optimizers, however, in this case, the learning rate was specifically set to 0.0001.

Training the dataset can take a relatively large amount of time. Hence, it is extremely unnecessary to train the model from scratch more than once. However, Keras has callbacks which helps to save and load the weights of a model after training so that they can be reused at any time. Callbacks are a set of built-in functions that can be used to monitor the training process. For instance, ModelCheckpoint saves the weight into a file that can later be loaded into an existing model, or ReduceLROnPlateau readjusts the learning rate if a target property of the model does not improve within a specified number of epochs. In this research, only ModelCheckpoint is used. Finally, the model is fit onto the training dataset and trained for 200 epochs (iterations) with batch size of 128. Batch size is defined as a hyperparameter specifying the interval after which the model’s parameters are updated. For example, a batch size of 128 means that after training each 128 data samples, the model will update its parameters.

### Generating text

After the training process, an evaluation method needs to be derived in order to assess the efficiency of the model. However, diversity is needed so as to keep the output creative and not monotone – generating the same output for the same input text. Hence, a sample function is implemented with the purpose of sample an index from the one-hot encoded predicted vector, based on the “temperature” or level of diversity specified when called. This temperature is a float parameter ranging from 0 to 1, with 0 having no diversity and 1 meaning complete randomized output. The principle of the function can be seen below

A screenshot of a cell phone

Description automatically generated

Figure 3‑1 LSTM model for text generation using Keras

## Improving the model using Reinforcement Learning

With a relatively small dataset, it is understandable that the model itself is unable to generate perfect English text. Therefore, to improve the model in order to generate more outputs that meet different specific criteria, one way is to use reinforcement learning algorithms. This section will explain the used reinforcement learning algorithm, along with its implementation using the Keras framework.

### Deep Q-learning

For this experiment, the derivation of our Deep Q-Learning algorithm is based on the algorithm described in Algorithm 1. However, the action selection method during each timestep is not , but instead to choose the maximum probability action 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 are 100 and 1000 respectively. The interval between each experience replay process is 40, which means that the algorithm performs 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** |

### Experimental setup

To test the efficiency of the described reinforcement learning model, several experiments with different reward functions are 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 3.1. The properties of the RL environment were defined in the Environment class. The observation for each environment was the one-hot encoded sequence of 40 characters from the dataset; on the other hand, the action was a discrete number from 0 to 38, corresponding to 39 possible characters from the dataset. These environments also had a step function which was used to evaluate the reward. Different reward conditions were implemented based on their corresponding task.

In terms of the reinforcement learning training process, weights of all the supervised model’s layers will be updated after each training episode, which consists of 1000 timesteps, corresponding to 1000 actions being 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.

a. 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, the reward can be defined as Equation 3.5:

, (3.5)

where is the reward at state for action .

b. Negative distance reward function:

Like part a, 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 3.6:

, (3.6)

where is the reward at state for action at timestep and and are actions at timestep and respectively. It can be easily seen that the agent’s goal for this reward function is the similar to that of Section 3.3.2.a. 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. However, this function was implemented to evaluate the ability of the RL agent to process reward conditions defined in a more complex manner.

c. Producing meaningful English words:

The main objective of this experiment was to improve the quality of text generation obtained from Section 3.2. Hence, it was essential that the RL agent get rewarded if it produced a meaningful English word. 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.

A picture containing text

Description automatically generated

Figure 3‑2 Extracting the last word for reward evaluation (self.buffer is the environment’s state)

In terms of reward evaluation, the agent got 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 [12]. 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 3.7:

, (3.7)

where is the reward at state for action .

### Jensen – Shannon divergence

Since our supervised model from part 3.1 is sufficiently well-trained, it is good to use it for reference during RL training in order to prevent the agent to learn completely new and ineffective words. To keep the RL training similar to the original supervised model, the Jensen-Shannon divergence can be added to the loss term before model optimization. Jensen – Shannon divergence is based on the Kullback – Leibler divergence, which is often used in machine learning optimization algorithms. However, the former has a few differences, adding symmetricity as well as both lower and upper bound, making the value always finite.

The Kullback – Leibler divergence, also known as the relative entropy, measures the similarity between two probability distributions. It is defined in Equation X: assume and are two discrete probability distributions on the same probability space , and is the KLD of the two probability distributions P and Q.

However, since the KLD is asymmetric and not bounded [13], several symmetrical alternatives of the KLD were introduced, notably the Jensen-Shannon divergence. The JSD can be defined as the

# RESULTS AND DISCUSSION

The text generated from the supervised model defined in Section 3.1 is demonstrated in Figure 4.1. 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

Text, letter

Description automatically generated

Figure 4‑1 Generated text from supervised model

For each reward function defined in Section 3.3.2, general observations after training will be discussed, followed by a reward graph throughout 100 episodes and the text generated after training is completed.

a. Simple reward functions:

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:

Chart

Description automatically generated

Figure 4‑2 Task's rewards after 100 training episodes

Generally, there is a significant increase in training reward after the training process, from 1000 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.

b. Negative distance reward function:

Chart, histogram

Description automatically generated

Figure 4‑3 Negative distance reward of 100 episodes

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 simple reward functions defined in a more complex manner.

c. Producing meaningful English words (more complex reward function):

Since the supervised model is trained for only a limited number of epochs, it is inevitable that the quality of the outputted text is not perfect, in terms of English validity. Hence, it is essential that the reinforcement learning agent, in which case our supervised model, learns to improve its text quality. Thus, the desired environment gives a reward of 10 if the last outputted word in the environment’s state is a valid English word. The environment’s state is defined as a sequence of one-hot encoded 40 characters that is subsequently put into the supervised model in order to output the next action. On the other hand, the NLTK framework is used to check if a word is valid or not. The reward trend after 100 training episodes is depicted in Figure 10:

Chart

Description automatically generated

Figure 4‑4 Rewards after 100 episodes for NLTK validity reward function

The starting reward of the agent is already relatively high, since the supervised model is considered sufficiently well-trained. It can be easily seen from Figure 10 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, which can be seen in Figure 11:

Chart

Description automatically generated

Figure 4‑5 Outputted text after training

Other minor reward criteria were introduced into the reward function. For example, the RL agent can learn not to produce the same words twice (1.1) or not to produce the word ‘the’ (1.2). Their reward graphs are similar to Figure 10; thus, the agent seems to adapt well into more complex reward criteria, which can be seen in Figure 12 and Figure 13 (Figure 11 and 12 shows the text output for both criteria above).

Chart

Description automatically generated

Figure 4‑6 Outputted text after training for task 1.1 (figure to be added)

# CONCLUSION

# REFERENCES

1. Stephenson, I. (2019). *Understanding Deep Learning*. [online] methods.blog. Available at: https://methodsblog.com/2019/11/13/understanding-deep-learning/ [Accessed 14 Mar. 2020].
2. Hochreiter, S. and Schmidhuber, J. (1997). Flat Minima. *Neural Computation*, [online] 9(1), pp.1–42. Available at: <https://www.bioinf.jku.at/publications/older/2604.pdf>. (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. Guide to Writing a Thesis in Technical Fields: Instructions for Master of Science and Bachelor of Science Theses, Tampere University, Tampere, 2019. Available: POP > Study info > Master's thesis > MSc % thesis guidelines
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. Nwankpa, C., Ijomah, W., Gachagan, A. and Marshall, S. (2018). *Activation Functions: Comparison of trends in Practice and Research for Deep Learning*. [online] arXiv.org. Available at: https://arxiv.org/abs/1811.03378.
8. ‌
9. Choi, D., Shallue, C.J., Nado, Z., Lee, J., Maddison, C.J. and Dahl, G.E. (2019). *On Empirical Comparisons of Optimizers for Deep Learning*. [online] arXiv.org. Available at: https://arxiv.org/abs/1910.05446.
10. Kingma, D.P. and Ba, J. (2014). *Adam: A Method for Stochastic Optimization*. [online] arXiv.org. Available at: <https://arxiv.org/abs/1412.6980>.
11. Nltk.org. (2009). *Natural Language Toolkit — NLTK 3.4.4 documentation*. [online] Available at: <https://www.nltk.org/>. [Accessed 27Jul. 2020]
12. Pong, V., Gu, S., Dalal, M. and Levine, S. (2020). Temporal Difference Models: Model-Free Deep RL for Model-Based Control. *arXiv:1802.09081 [cs]*. [online] Available at: https://arxiv.org/abs/1802.09081.
13. Watkins, C.J.C.H., Dayan, P. *Q*-learning. *Mach Learn* **8,**279–292 (1992). https://doi.org/10.1007/BF00992698
14. Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D. and Riedmiller, M. (2013). *Playing Atari with Deep Reinforcement Learning*. [online] arXiv.org. Available at: https://arxiv.org/abs/1312.5602.