School of Computing
FACULTY OF ENGINEERING AND PHYSICAL SCIENCE

UNIVERSITY OF LEEDS

Program Synthesis Using a Long Short Term Memory (LSTM) Recurrent Neural Network

Kieron Sean Hushon

Submitted in accordance with the requirements for the degree of Computer Science

2019/20

40 credits

The candidate confirms that the following have been submitted.

Items	Format	Recipient(s) and Date
Source Code	Github repository	Supervisor, Assessor (29/04/20)
Report	PDF Report	Minerva (29/04/20)

Type of project: Exploratory Software

The candidate confirms that the work submitted is their own and the appropriate credit has been given where reference has been made to the work of others.

I understand that failure to attribute material which is obtained from another source may be considered as plagiarism.

KHushon

(Signature of Student) ______________________

Summary

This project aims to deliver a Long Short Term Memory (LSTM) Recurrent Neural Network, which, given sufficient training data, is capable of generating its own compilable code for the simple task of drawing a square and saving the image in png file format.

Acknowledgements

I would like to thank Dr Andy Bulpitt for his help, support and encouragement during this project.

Contents

Chapter 1

Introduction

1.1 Introduction

Neural networks have many different definitions depending on the specific type of network being referred. Neural networks are currently being used for many different tasks such as speech recognition[4], text classification[16], stock market predictions[8] and cancer classification[21]. With the increasing amount of data that neural networks can take advantage of, it is only over the past decade that the inherit multidisciplinary of machine learning has been recognised[19]. However, the challenge of program synthesis is unlike others such as object recognition and speech translation, since its abstract nature and demand for rigor make it difficult even for human minds to attempt.[15]

1.2 Aim

The aim of this project is to produce a Long Short Term Memory Recurrent Neural Network (LSTM), that makes use of a high amount of automated training data - that can write its own code which produces a square.

1.3 Objectives

The primary objectives are as follows:

- Acquire/produce enough training data for the network
- Develop a testing framework that allows for the manipulation of multiple hyperparameters in the network
- Network creates compliable python script
- Output of python script is a saved png file of a square

1.4 Deliverables

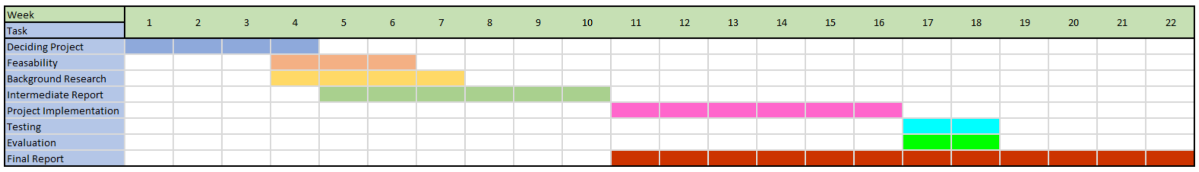
- An automated script that generates many python scripts of square generation
- A python file with the framework for training an LSTM
- An optimal model for program synthesis
- A python file that manipulates the saved model for output the desired output
- A report containing:
  - Sufficient background research

- Design and implementation of the proposed solution
- Evaluation and review of findings

1.5 Timeline

The first 4 weeks will be used to determine the project definition – also in week 4, A check on the feasibility of the project will be conducted to ensure that the scope of the project is viable. Then, the weeks 4 – 7 will be spent conducting background research, this will aid in finding the most efficient way of solving the problem – and give expanded knowledge on the different types of networks/solutions to similar project scopes as this one.

Next, the intermediate report will be completed and handed in during the Christmas break. After the break, the implementation of the project will commence alongside the implementation of the final report. Testing and evaluation of the project will commence after a software solution has been acquired, and the rest of the time available will be spent finalising and polishing the report. Figure 1.1 shows a visual representation of this plan in the form of a Gantt chart.



Week / Task	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
Deciding Project	🟦	🟦	🟦	🟦																		
Feasability				🟧	🟧	🟧																
Background Research				🟨	🟨	🟨	🟨															
Intermediate Report					🟩	🟩	🟩	🟩	🟩	🟩												
Project Implementation											🟪	🟪	🟪	🟪	🟪	🟪						
Testing																	🟦	🟦				
Evaluation																	🟩	🟩				
Final Report											🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥

Figure 1.1: Gantt chart of proposed plan

1.6 Methodology

1.6.1 Agile Scrum

During this project, an agile scrum methodology will be adapted and implemented. This will involve breaking the project down into smaller parts (sprints), and reviewing the work completed within these sprints within a given time period.

1.6.2 Version Control

A version table will be utilised for the writing of the report. This will be a table consisting of previous versions of the report, and a description of what was implemented to the report for each row of the table. A Github repository will be used for the version control of the software.

1.6.3 Pep8

Throughout the software development of the project, the pep8 style guide will be used as for the code writing process. This will increase the readability and clarity of the code. [28]

Chapter 2

Background Research

This chapter discusses the research that was conducted in order to gain understanding the problem scope and deduce an appropriate solution.

2.1 Relevance to Degree

The implementation of this project requires background knowledge of previously studied modules

2.1.1 COMP3611 Machine Learning

This module taught information on varying machine learning techniques, most importantly neural networks and their applications.

2.1.2 COMP1911 Professional Computing

This module taught the foundations of ethics and morals in the world of computing. This knowledge will be applied in the ethical reflection section.

2.2 Neural Networks

Neural networks first sparked interest after the introduction of simplified neurons[19] in 1943. These neurons were presented as models of biological neurons that could perform computational tasks.[17] The neuron has a set of inputs x which are multiplied by their

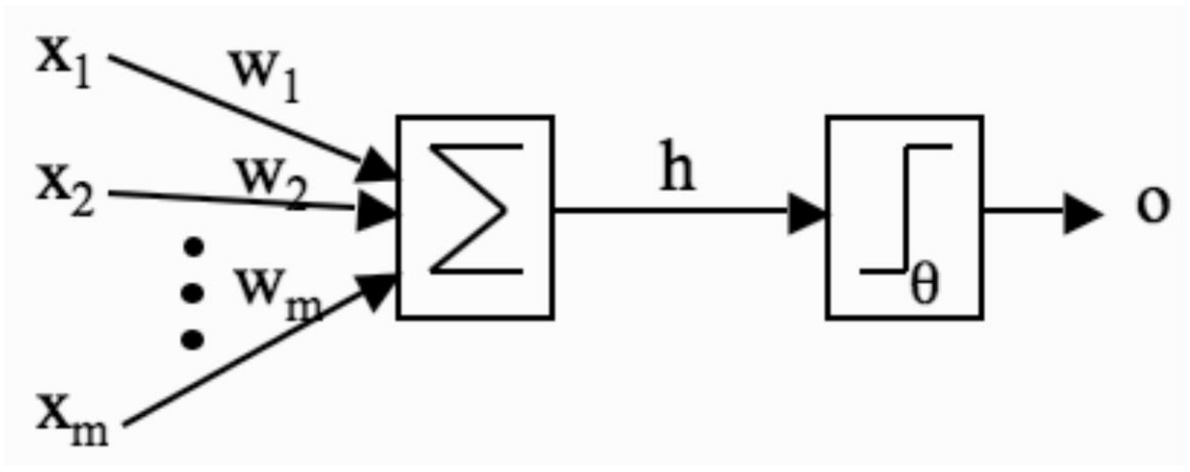


Figure 2.1: McCulloh and Pitts' mathematical model of a neuron[19]

corresponding weight w, the neuron then sums the values of all the inputs multiplied by the weights – if this sum of these multiplications h is greater than the threshold Θ, then the neuron activates. We can write the equation of this neuron as follows:

$$\sum_{i=1}^{m} w_i x_i = h \tag{2.1}$$

2.2.1 Multi-Layer Perceptron

Expanding on the singular neuron model, the Multi-layer perceptron (MLP) can be used for problems that are not linearly separable. MLP's consists of at least 3 layers, the input layer, a hidden layer and an output layer.

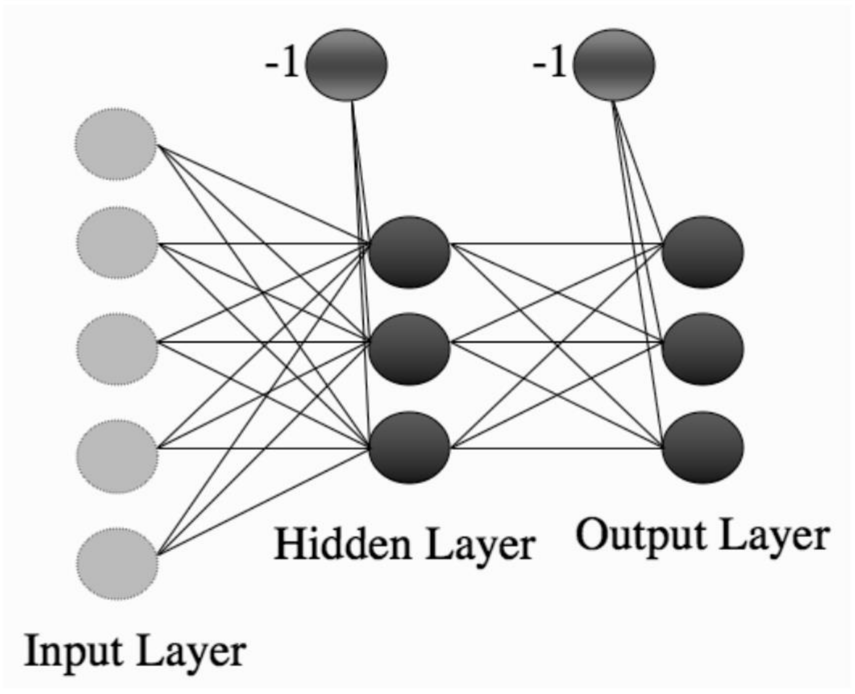


Figure 2.2: The multi-layer perceptron[19]

2.2.1.1 Bias Input

From figure 2.2 we see that there are extra inputs given of -1. This is called a bias input. The bias input is of fixed value thus giving every hidden layer and the output layer an extra input of fixed value.

2.2.1.2 Forward Pass

The forward pass on an MLP is working out the outputs based on the given inputs. This is the same as working out the output on a single neuron, but we need to continue the process for each set of neurons, layer by layer.

2.2.1.3 Backwards Pass

The backward pass on an MLP is where we update the weights in the network based on the error. This can be tricky to do as we need to know which weights to update. To solve this, we use a technique called the back propagation of error.

2.2.2 Regression

Regression is a supervised learning method; this means that the algorithm generalises to respond correctly to all possible inputs[19]. We want to use regression when we are trying to figure out a function based on some given results i.e. when we want to make predictions with a list of continuous numerical values. This is the category our problem falls under.

2.2.3 Classification

Classification is different to regression as it revolves around taking input vectors and deciding which of N specified classes they belong to [19].

2.2.4 Backpropagation

In an multi-layer perceptron, we can use the backpropagation of error to help the network learn and minimise error. The backpropagation algorithm uses a function called stochastic gradient descent. This function calculates the derivative of the error. This derivative can then be used to change the weights between the neurons. A commonly used error function is the mean squared error function. This calculates the difference between the output y and the target t, squares them, and adds them all together:

$$\frac{1}{2}\sum_{k=1}^{N}(y_k - t_k)^2 = E(w) \tag{2.2}$$

We can then take in the input from the hidden layers neurons and the second layer weights:

$$\frac{1}{2}\sum_{k=1}^{N}[g(\sum_{j=0}^{M} w_j k a_j) - t_k]^2 \tag{2.3}$$

This function can then be simplified by focusing on the perceptron and indexing the input nodes by i and the output nodes by k:

$$\frac{1}{2}\sum_{k=1}^{N}[g(\sum_{i=0}^{L} w_i k x_i) - t_k]^2 \tag{2.4}$$

A gradient descent algorithm can now be applied to adjust the weights $w_i k$ for fixed values of i and k in the negative direction of $E(w)$

$$\frac{\delta E}{\delta w_i k} = \frac{\delta}{\delta w_i k}(E(w)) \tag{2.5}$$

$$\frac{\delta E}{\delta w_i k} = \frac{\delta}{\delta w_i k}(\frac{1}{2}\sum_{k=1}^{N}(y_k - t_k)^2) \tag{2.6}$$

$$\frac{\delta E}{\delta w_i k} = \frac{1}{2}\sum_{k=1}^{N} 2(y_k - t_k)\frac{\delta}{\delta w_i k}(y_k - \sum_{i=0}^{L} w_i k x_i) \tag{2.7}$$

Then we derive again to form:

$$\frac{\delta E}{\delta w_i k} = \sum_{k=1}^{N} (t_k - y_k)(-x_i) \tag{2.8}$$

2.2.5 Updating The Weights

Each weight is updated by taking the change in the weight $\Delta w_i k = -(y_k - t_k) \times x_i$ and adding this to the itself:

$$w_i k \leftarrow w_i k + \eta(t_k - y_k)x_i \tag{2.9}$$

However, since we are following gradient descent, the negative of this function is required for the weight update rule

$$w_i k \leftarrow w_i k + \eta(t_k - y_k)x_i \tag{2.10}$$

Where:

- η = The Learning rate (a defined value that controls how much the weights should change by)

2.2.6 Overfitting

A common problem with neural networks is the overfitting of the data. This is when the network is performing very highly on the training data but fails when it comes to the test data. The reason for this is that the network loses its ability to generalise.

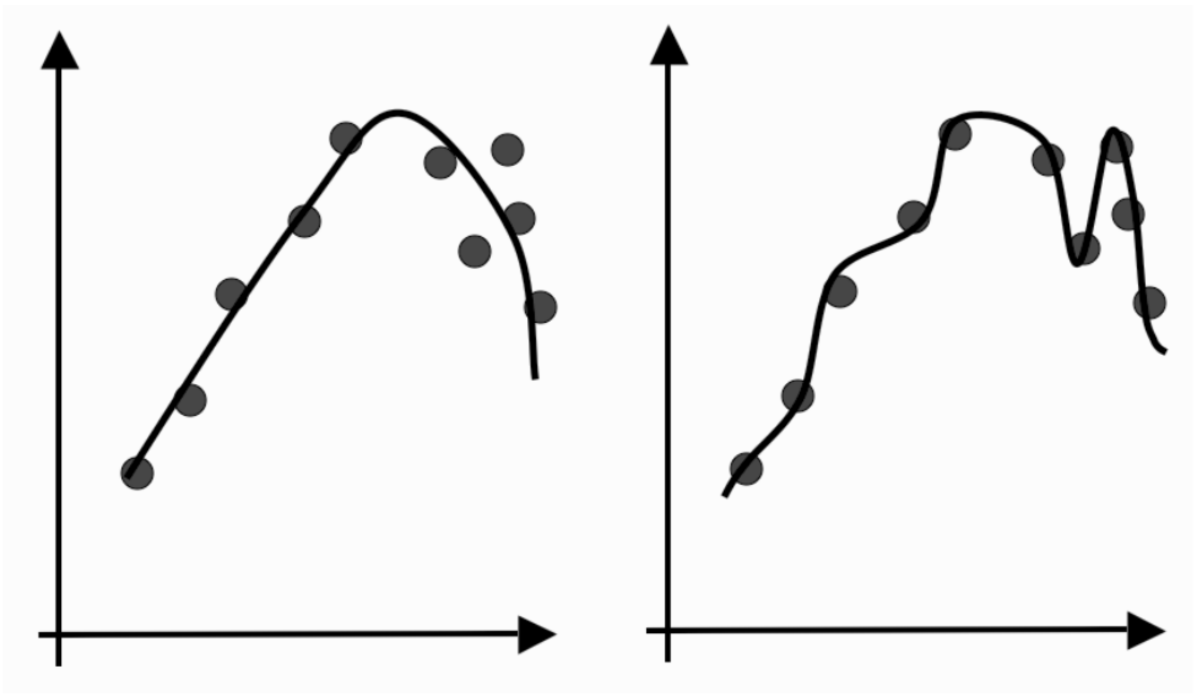


Figure 2.3: Example of overfitting [19]

In figure 2.3 we can see that the graph on the left is generalising well i.e. there is a rough estimate of what the underlying general function is and looks like it can predict new values. On the right hand side however, the network has been trained for too long and is starting to lose the ability to generalise, this is evident from the fact that the estimated curve is too ‘perfect’, and is going through every point (even the noise components). In general, we want to stop the training when the network produces something like the left figure, otherwise the network may overfit and become like that on the right. To combat the occurrence of overfitting, we use three sets of data – the training data, the testing data and the validation data. According to Srivastava, Hinton, Krizhevsky, Sutskever and Salakhutdinov[26] overfitting is a serious problem for neural networks.

2.2.7 Activation Function

The activation function of a node in a neural network determines the output of the node based on given inputs. Activation functions can be used to 'squash' input into a range of smaller values.

2.2.8 Dropout Rate

Adding a dropout rate to the neural network can help to address the problem of overfitting [26]. The idea of the dropout is to set a percentage of units to be randomly ignored (dropped) during training. This prevents units from co-adapting too much[26]. Figure 2.4 shows an example of a simple feed forward neural network with some of the units dropped out.

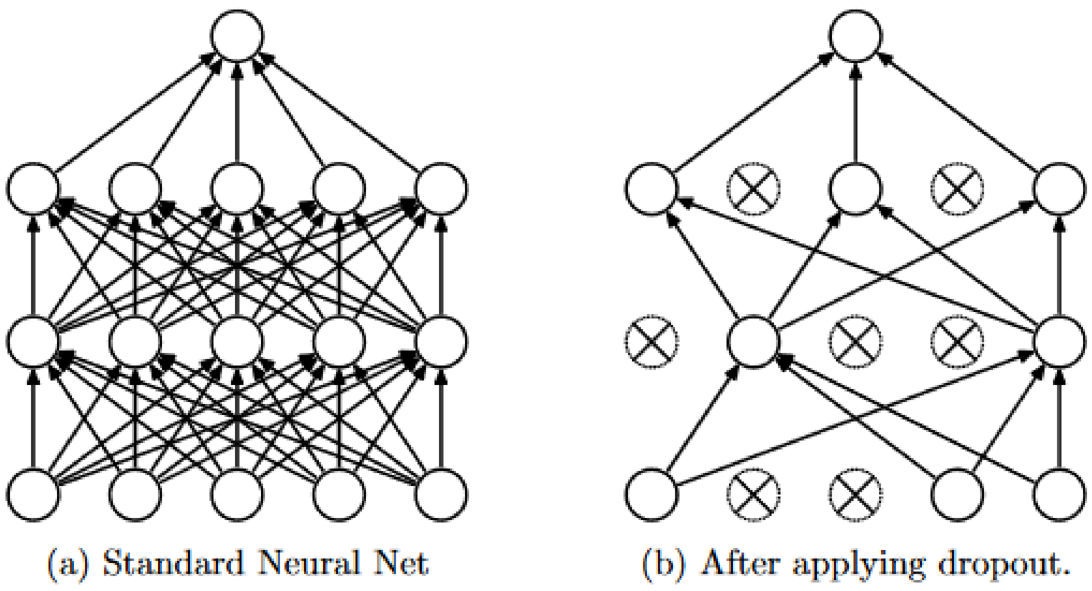


Figure 2.4: **(a)**: shows a feed forward neural network with 2 hidden layers. **(b)**: shows an example of the neural network seen in **(a)** that has been thinned via the application of a dropout function. [26]

2.3 Recurrent Neural Networks

Recurrent neural networks differ from the previously discussed conventional feedfoward neural networks, in the sense that, they not only operate on the inputs from the previous layer, but also keep a track of an internal state of what the network has already processed. This state space allows for the learning of sequentially extended dependencies over unspecified intervals

[7]. However, standard recurrent neural networks cannot bridge more than 5-19 time steps [9], due to the fact that back propagated error signals either grow or shrink every time step, thus, the error either becomes too large or too small. [6] A solution to this problem is a gradient based method called long short-term memory (LSTM) [13].

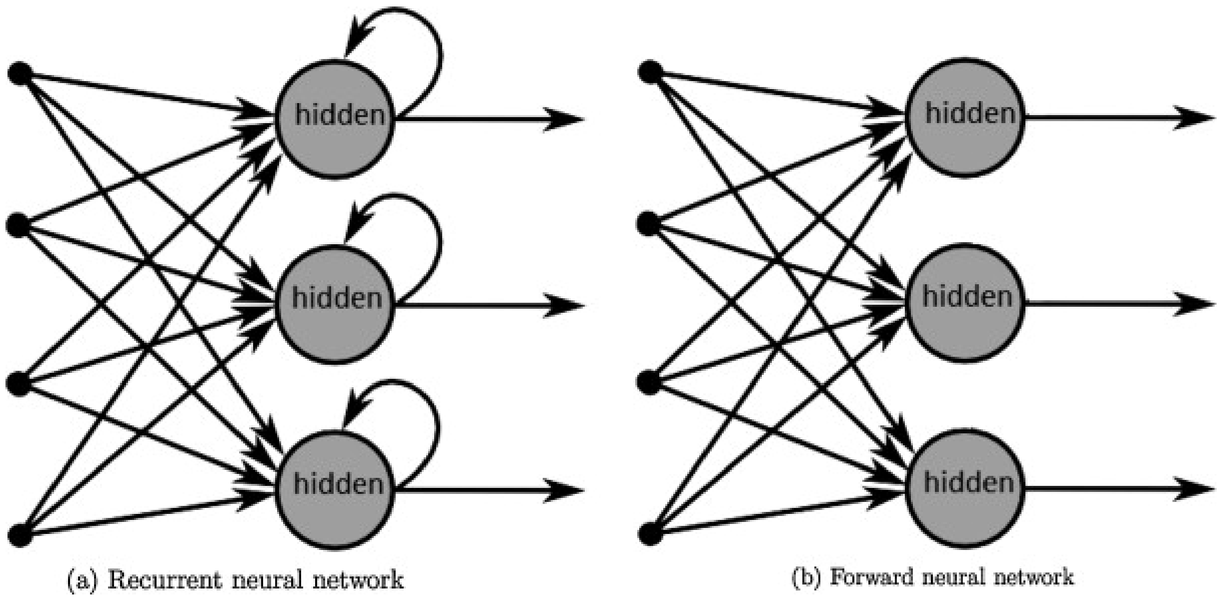


Figure 2.5: Recurrent Neural Network vs Feedfoward Neural Network [11]

2.3.1 Long Short Term Memory

Long Short-Term Memory Recurrent Neural Networks (LSTM-RNN) are one of the most powerful dynamic classifiers publicly known. They are a common Recurrent Neural Network architecture that can remember values for long or short periods of time [22]. Figure 2.6 shows an inside look of an LSTM cell, a detailed decription of this cell is as follows:

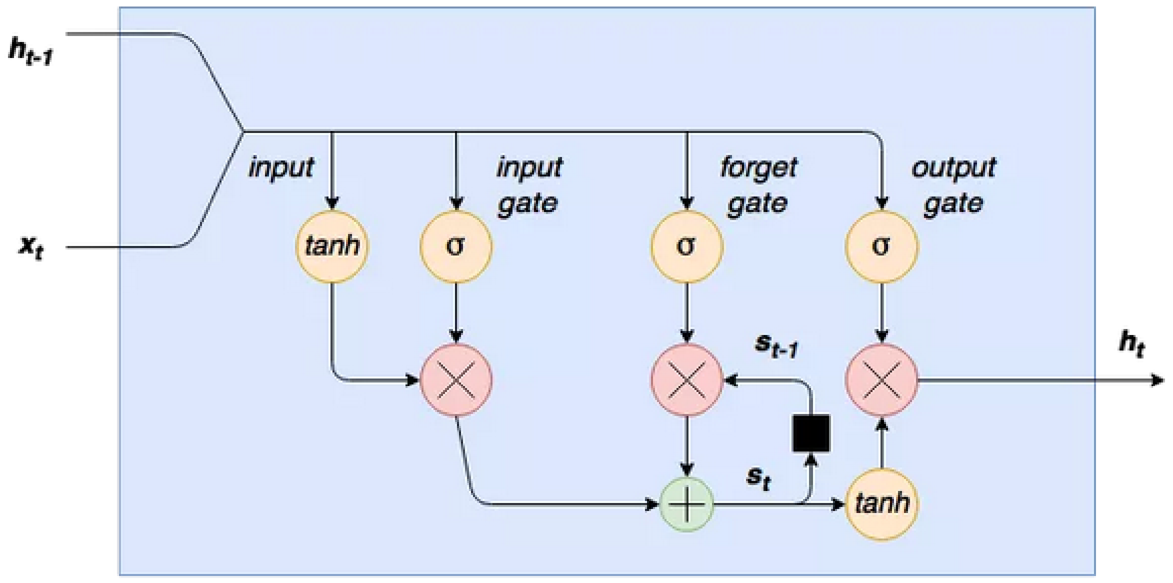


Figure 2.6: Inside look of an LSTM cell [3]

Initially the new word x_t is concatenated with the previous output from the cell h_{t-1}. This combined input is then 'squashed' by the *tanh* layer. Then, the input is passed through an input gate, which is a layer of sigmoid activated nodes, which aims to eliminate any anomalies within the input vector. Next, the data flows through the forget gate loop. LSTMs keep track of an internal state variable which is lagged one time step behind (i.e. added to the input data). This allows for a layer of recurrence and reduces the risk of gradients vanishing. Finally, the data is squashed by a *tanh* function that is passed through an output gate, which determines the values that are allowed to be outputted from the cell h_t. The mathematics behind the cell are as follows:

Input

$$g = tanh(b^g + x_t U^g + h_{t-1} V^g) \tag{2.11}$$

Where:

- x_t = The current input
- U^g = The weights for the current input
- $h_t - 1$ = The previous cells output
- V^g = The weights for the previous cells output
- b^g = The bias input weight

The input is passed to the *tanh* function to squash the input to be in the range $-1 <= input <= 1$. The values that are raised to the exponent g are not raised as powers. This exponent represents that these values are the **input** weights and bias values. (not input gate, forget gate or output gate values.). The initial input is also passed through the input gate, which performs the sigmoid function on the input.

Input Gate

$$i = \sigma(b^i + x_t U^i + h_{t-1} V^i) \tag{2.12}$$

Where:

- $\sigma(z) = \frac{1}{1+e^- z}$

The output of the input g and the output of the input gate i are then multiplied together given as:

$$g \circ i \tag{2.13}$$

Where:

- $\circ$ represents element-wide multiplication

The output of the forget gate is defined as:

$$f = \sigma(b^f + x_t U^f + h_{t-1} V^f) \tag{2.14}$$

The forget gate output f is then multiplied by the previous state s_{t-1}:

$$s_{t-1} \circ f \tag{2.15}$$

Where:

- $s_t = s_{t-1} \circ f + g \circ i =$ the output of the forget gate / state loop

The output of the output gate can be expressed as:

$$o = \sigma(b^o + x_t U^o + h_{t-1} V^o) \tag{2.16}$$

This output is then multiplied by the output of the *tanh* of *st* to give the final output of the cell:

$$h_t = tanh(s_t) \circ o \tag{2.17}$$

2.4 Generative Adversarial Networks

Generative Adversarial Networks were invented by Ian Goodfellow and his team in 2014. [10] The idea of this class of machine learning is that two neural networks contest with each other in a game theory style min-max game. The two networks are a generator and discriminator. The generator generates new data points, and the discriminator classifies this generated data. We want to train the discriminator D to be able to classify if data is from the data set, or generated from the generator G. At the same time as this, we want to train the generator G to minimise the value of $log(1 - D(G(z)))$ i.e. we want G to train such that output on G is classified as part of the data set when passed to D. This can be thought of as a min-max game of two players and can be formularised in the equation:

$$min(G)max(D)V(D,G) = \mathbb{E}_{xpdata(x)}[logD(x)] + \mathbb{E}_{zpz(z)}[log(1 - D(G(z)))] \tag{2.18}$$

Where:

- V: The value function
- G: A differential function represented by a multi layer perceptron
- D: A differential function represented by a multi later perceptron
- p_g: The distribution of G
- x: training data
- $p_z(z)$: input noise variable
- $G(z : \Theta_g)$: Multi layer perceptron
  - Θ: Parameters for generator G
- $D(x; \Theta_d)$ Multi layer perceptron
  - $D(x)$: The output of $D(x; \Theta_d)$ which represents the probability that x came from the initial data set rather than from the output of $G(z : \Theta_g)$

2.4.1 Relational Memory Based Generator

Generative Adversarial Networks (GANs) have achieved great success at generating realistic images, however, text generation remains a challenge [20]. Currently, the dominant GAN architectures for text generation are built using LSTM [18] [30] However, Weili Nie, Nina Narodytska and Ankit B. Patel believe the LTSM may be at fault for the challenges of text generation after some experimental observations [20] Relational memory [24] is offered as an alternative to LTSM's. The idea of relational memory is to consider a memory matrix and allow for interactions between the matrix using a self-attention mechanism [29]. Relational memory was empirically found to perform better than LTSM in language modelling [24] and the use of multiple memory slots can be used to increase the power of the generator that is generating text [20].

2.5 Choice of Neural Network

A long short term memory recurrent neural network will be implemented for the completion of this project (section 2.3.1). Despite the advantages of Generative Adversarial Networks (section 2.4) and the Relational Memory Based Generator (section 2.4.1)), analysis of the increased effectiveness of these architectures versus the increased complexity and time to implement them was deemed futile.

2.6 One-Hot Encoding

Neural networks have limited applicability to categorical values due to their continuous nature [12]. This means that integer representation for categorical values is not the best solution; one-hot encoding is offered as a better alternative. One-hot encoding is a vector representation where all the elements in a row are 0, except one, which has the value of 1. Since the LSTM will need to categorise each character given, a one-hot encoded vectorisation method on the characters can be performed. Where the column that the 1 appears in each row of the vector represents a specific character.

2.7 Turing Machines

The idea of Turing Machines was invented by Alan Turing in 1936 [27]. A Turing Machine is a theoretical model of a computer hardware, which manipulates symbols on a strip of tape according to some rules. A Turing machine can do everything a computer can do [25]. i.e. there exists a Turing Machine that can simulate any computer algorithm.
A formal definition of a Turing machine is as follows: [14]

$$M = (Q, \Gamma, b, \Sigma, \delta, q_0, F) \tag{2.19}$$

Where:

- Q is a finite, non-empty set of states $= \{A, B, C, HALT\}$
- Γ is a finite, non empty set of tape symbols $= \{0, 1\}$

- b is an element of Γ which is the blank symbol that can occur infinitely on the tape: $b \in \Gamma = 0$
- Σ is a subset of Γ excluding b which is the set of symbols allowed to appear in the initial tape: $\Sigma \subseteq \Gamma = \{1\}$
- q_0 is an element of Q which is the initial state: $q_0 = A$
- F is a subset if Q which is the set that contains the state that M will halt on: $F \subseteq \mathrm{Q} = \{\mathrm{HALT}\}$
- δ is the transition function. The machine will halt if δ is not defined on the current state and current tape symbol: $\delta : (Q\ F) \times \Gamma \nrightarrow Q \times \Gamma \times \{L, R\}$
  - Where L is the left shift and R is the right shift

2.7.1 The Halting Problem

Since the desired output for this project is a compilable script of code, the halting problem shall be taken into consideration. The halting problem is as follows: Given an explanation of a program, decide whether the program finishes running or continues to run, and will thus run forever. This is corresponding to the problem of deciding, given a program and an input, whether the program will ultimately halt when run with that input, or will run forever [23]. This problem was proved to be computationally unsolvable by Alan Turing [27] the proof by contradiction can be summarised as follows:

- Define a new Turing Machine T_M that takes the input $< M >$.
  - If T_M accepts $< M, < M >>$ then loop
  - If T_M rejects $< M, < M >>$ then halt
- Now Consider T_M with input $< T_M >$
  - If it halts, then T_M rejects $< T_M, < T_M >>$
    - which implies it cannot halt.
  - If it loops, then T_M accepts $< T_M, < T_M >>$
    - which implies it must halt

We have a contradiction, thus, the problem is unsolvable.

2.8 Development Resources

2.8.1 Python 3

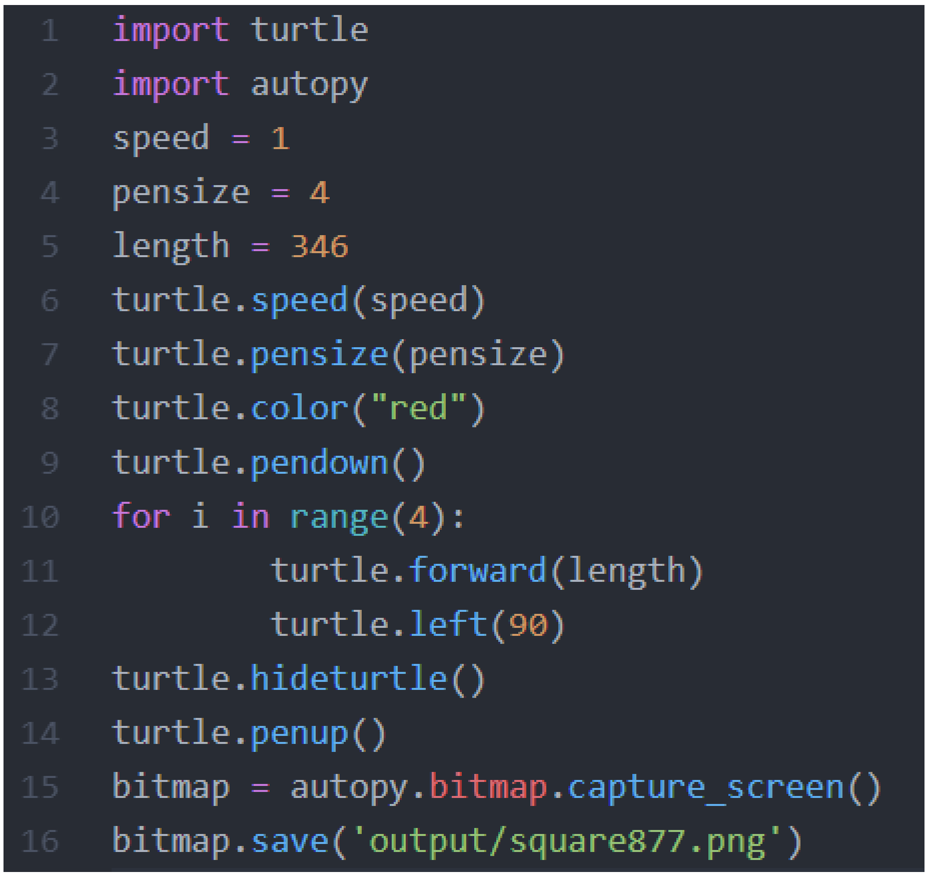
Python 3 was considered to be more intuitive for this project than Python 2, thanks to integer division output defaulting to float values and Unicode being the default storing of Strings.

2.8.2 Python Autopy

The Autopy framework will be used to screenshot the drawn square, and to save the image as a Portable Network Graphics (PNG) file type.

2.8.3 Python Turtle

The python framework turtle graphics[1] is a simple graphics framework that is based on the Logo programming language that was created by Wally Feurzeig, Seymour Papert, and Cynthia Solomon in 1967[5]. The framework uses turtle graphics which allows for a simple way to draw shapes. See figure 2.7



```python
import turtle
import autopy
speed = 1
pensize = 4
length = 346
turtle.speed(speed)
turtle.pensize(pensize)
turtle.color("red")
turtle.pendown()
for i in range(4):
        turtle.forward(length)
        turtle.left(90)
turtle.hideturtle()
turtle.penup()
bitmap = autopy.bitmap.capture_screen()
bitmap.save('output/square877.png')
```

Figure 2.7: An example of the desired output of the LSTM

2.8.4 Keras

The Keras API will be used to implement the LSTM. Keras minimises the number of user actions required and allows for me to be more productive due to the ease of use of changing variables and trying new ideas. Keras also provides clear feedback on user errors.

2.8.5 Google Colaboratory

Due to the computational cost of training neural networks, the code to train the LSTM will be executed online via Google Colaboratory. Google Colaboratory allows for the use of external

GPUs to execute code which are vastly faster than the hardware available to me. This will save time and allow for more imaginative and extensive training of the network

2.8.6 TensorBoard

TensorBoard is a visualisation toolkit which enables the tracking and visualisation of data such as loss and accuracy. [2] This will be used during the testing stage to view the performance of the trained models.

Chapter 3

Design and Generation of Training Data

LSTMs cannot train on character values, and instead requires this data to be converted. This chapter will discuss the process of generating training data in string format, and parsing this data into vectors such that the data is in a readable format for the LSTM to train on.

3.1 Generating Training Data

Before the implementation of the neural network, the generation of appropriate data to train and test on is needed. Firstly, A python script was created that draws a square (figure 2.7. then, each line of this script is parsed into string format, ending each line with a new line. This string can then be written to a `.txt` file.

3.1.1 Ensuring Randomness Within Training Data

Using the turtle library to output an image of a square, some of the lines of code to output a square must be the same across all files. So, to ensure that the multiple examples of training data used are dissimilar, some of the lines of code must be different in the training data. To achieve this, the implementation of a random function for some of the numerical values is required; such as speed of the turtle, the size of the pen, the length of each side of the square and the colour of the pen.

3.1.2 Generating Multiple Scripts

A python script was created and executed to produce 1024 differing versions of the desired output. This output was saved to a text file and will proceed to be manipulated into readable data for the LSTM.

3.2 Reading The Input File

The generated input file was read and assigned to a variable as a single string. This string is then manipulated to convert all characters to lower case. This act guarantees a minimal number of distinct characters in the input data. This will reduce the computational resources required and time required for the network to train, as the vocabulary of characters that it trains on has been reduced.

3.3 Distinct Characters

The model needs to have a vocabulary of all characters that are found within the input file. This ensures that the model knows which characters can be generated. It also eliminates the possibility of generating any character outside of the input file. Figure 3.1 shows that the input data is first passed through the set function to reduce the string into a set of distinct

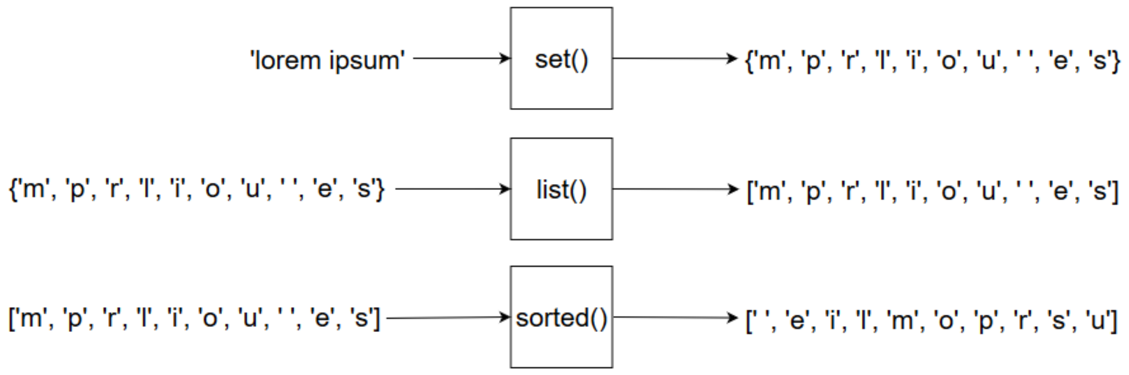

'lorem ipsum'
set()
{'m', 'p', 'r', 'l', 'i', 'o', 'u', ' ', 'e', 's'}
{'m', 'p', 'r', 'l', 'i', 'o', 'u', ' ', 'e', 's'}
list()
['m', 'p', 'r', 'l', 'i', 'o', 'u', ' ', 'e', 's']
['m', 'p', 'r', 'l', 'i', 'o', 'u', ' ', 'e', 's']
sorted()
[' ', 'e', 'i', 'l', 'm', 'o', 'p', 'r', 's', 'u']

Figure 3.1: Diagram of the manipulation of an input string into a sorted list of distinct characters

characters. Next the set is converted into a list, this is necessary as the data is required to be in the list format for section **4.3.1**. Finally, the list is sorted and ordered: symbolically, numerically, alphabetically. This requirement will be explained in section **4.3.1**

3.3.1 Character Index

The sorted list of distinct characters needs to be indexed to find the location of each character. This can be achieved using a dictionary see figure 3.2. The dictionary will contain each character, followed by an integer value of their position in the sorted list. The list was initially sorted so that the numerical values for each character match the order of the sorted list. This will be useful when we are converting from the vectorised data back into characters.

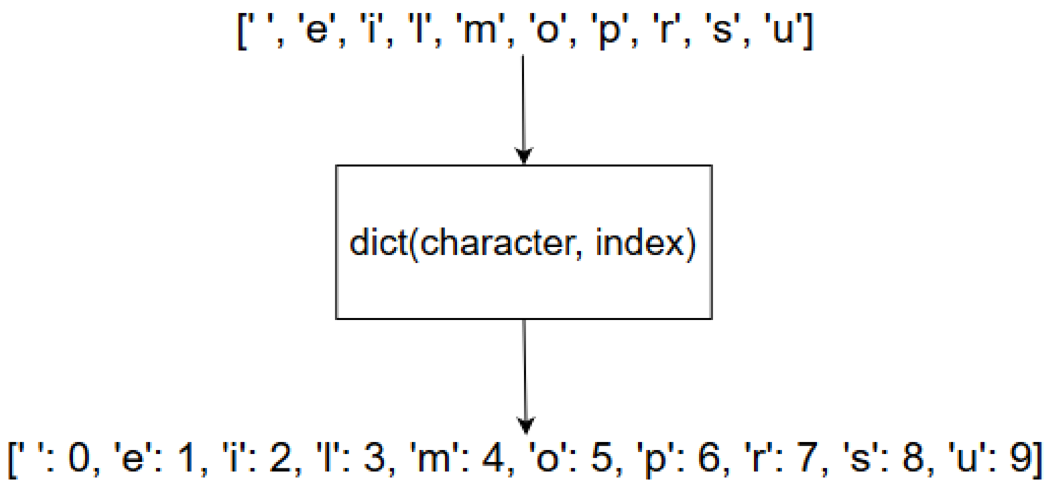

[' ', 'e', 'i', 'l', 'm', 'o', 'p', 'r', 's', 'u']
dict(character, index)
[' ': 0, 'e': 1, 'i': 2, 'l': 3, 'm': 4, 'o': 5, 'p': 6, 'r': 7, 's': 8, 'u': 9]

Figure 3.2: Shows how the character index is formed from an input list

3.4 Splitting The Input Data

The input data needs to be split into separate strings of a given length. This length can be assigned at run time. The length of these strings will be the amount of characters the LSTM will process and make predictions of the new characters on.

3.4.1 List Of Strings

The data is split into a list of strings* of a given length (sequence length), this list of strings will be the data that will be passed to the LSTM to train. The LSTM will be given one of the strings, and will have to produce the character immediately following the final character of the given string. Figure 3.3 shows a diagram of how the list of strings is populated.

*the list is actually a 2 dimensional character array e.g. `[['a','b','b'],['d','e','f']]` as opposed to `['abc','def']` but for the sake of what is done with this array in section 4.5 it is easier to consider these as strings.

3.4.2 Next Character

For every string stored in the list of strings, the character that immediately followed each of the strings in the original data set is stored in a separate list at the same index. This allows for validation of the LSTM's output, as when the LSTM is required to generate a new character based on a given string, the character stored in the next character list will hold the character that the LSTM should be generating. Figure 3.3 shows a diagram of how the next character list is populated.

3.5 Vectorisation

To be able to pass the data to the model for training, the data must first be converted into a format that the LSTM can accept. The two lists from section **4.4** will be one-hot-encoded, such that each element in each list is represented in a binary vector.

3.5.1 Training Vector

The training vector is initially declared as a 3 dimensional zero vector (all elements have an initial value of 0) of dimensions (elements in the list of strings * sequence length * number of distinct characters). The training vector is populated by iterating over the list of strings, and then iterating over each of the characters within each string. Each character is then passed to the character index dictionary, and the index value for that character is returned. The value of the Training vector at the index [index of the string in the list of strings, index of the character within the current string, character index value] is then set to 1 (True). Figure 3.4 shows a diagram of how the training vector is populated, figure 3.5 shows a visual representation of the training vector.

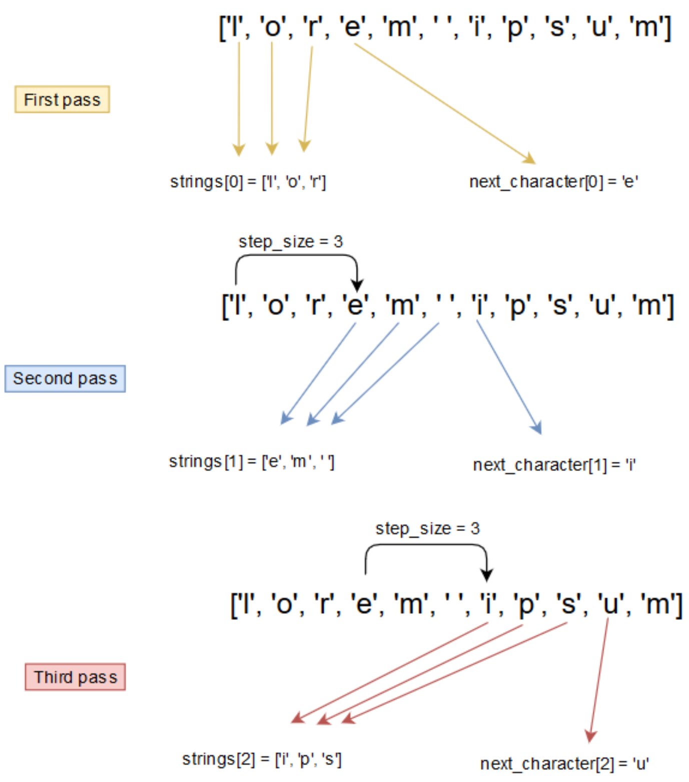

['l', 'o', 'r', 'e', 'm', ' ', 'i', 'p', 's', 'u', 'm']
First pass
strings[0] = ['l', 'o', 'r']
next_character[0] = 'e'
step_size = 3
Second pass
strings[1] = ['e', 'm', ' ']
next_character[1] = 'i'
step_size = 3
Third pass
strings[2] = ['i', 'p', 's']
next_character[2] = 'u'

Figure 3.3: Showing the `split_input_data()` loop with a sequence length of 3 and step size of 3 on some dummy data.

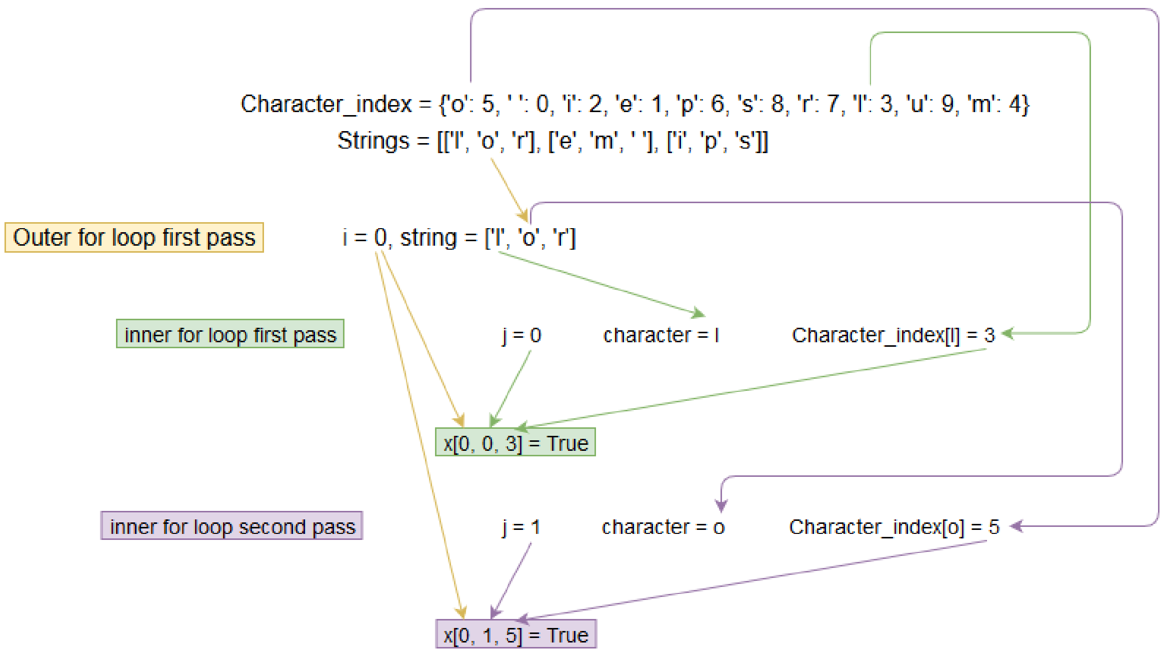

Character_index = {'o': 5, ' ': 0, 'i': 2, 'e': 1, 'p': 6, 's': 8, 'r': 7, 'l': 3, 'u': 9, 'm': 4}
Strings = [['l', 'o', 'r'], ['e', 'm', ' '], ['i', 'p', 's']]
Outer for loop first pass
i = 0, string = ['l', 'o', 'r']
inner for loop first pass
j = 0
character = l
Character_index[l] = 3
x[0, 0, 3] = True
inner for loop second pass
j = 1
character = o
Character_index[o] = 5
x[0, 1, 5] = True

Figure 3.4: Steps to populate the training vector (x)

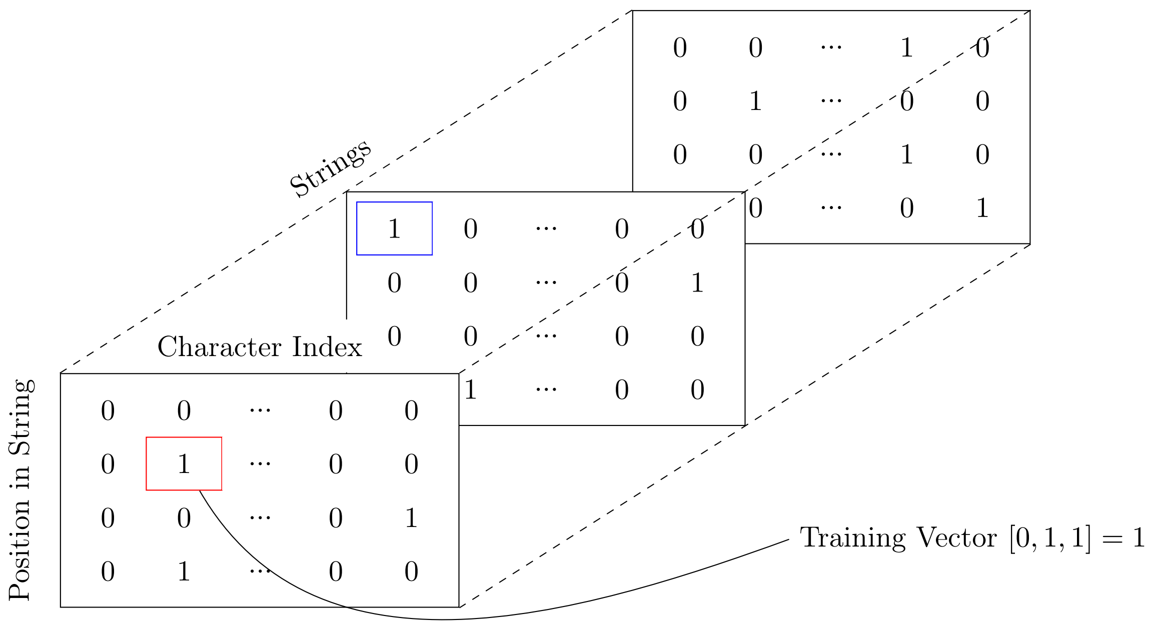


Figure 3.5: Example of the one-hot-encoded training vector. The element highlighted in red indicates that the character in the second position of the first string in the list has an index value of 1 in the character index dictionary. The element highlighted in blue indicates that the first character in the second string in the list has an index value of 0 in the character index dictionary.

3.5.2 Validation Vector

The validation vector is initially declared as a 2 dimensional zero vector of dimensions (elements in the list of strings * elements in the next character list). The validation vector is populated by iterating over each character in the next characters list, passing this character to the character index dictionary, and retrieving the index value for that character. The index in the validation vector [position in the next characters list, index value in the character index dictionary for that character] is set to 1 (True). Figure 3.6 shows a diagram of how the validation vector is populated, figure 3.7 shows a visual representation of the validation vector.

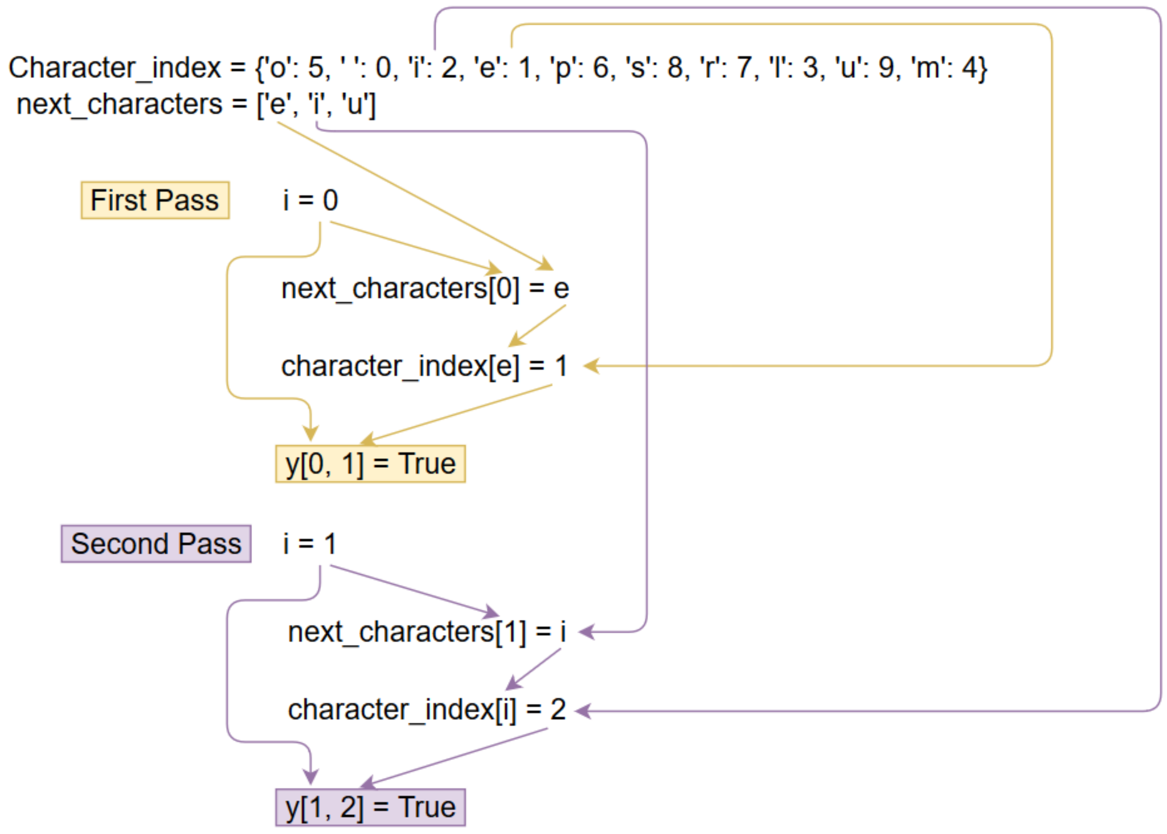


Figure 3.6: Steps to populate the validation vector (y)

$$
\begin{array}{c|ccccc|}
 & \multicolumn{5}{c}{\text{Character Index}} \\
\hline
\multirow{4}{*}{\text{Characters}} & 0 & \boxed{1} & \cdots & 0 & 0 \\
 & 0 & 0 & \cdots & 1 & 0 \\
 & 0 & 0 & \cdots & 0 & 0 \\
 & \boxed{1} & 0 & \cdots & 0 & 0 \\
\hline
\end{array}
$$

Figure 3.7: Visual representation of the one-hot-encoded validation vector. The element highlighted in red indicates that the character in the first element of the next characters list has an index value of 1 in the character index dictionary. The element highlighted in blue indicates that the character in the fourth element of the next characters list has an index value of 0 in the character index dictionary.

3.5.3 Use of The Training and Validation Vectors

The index value for each string in the training vector corresponds to the index value for each character in the validation vector. Hence, we can use these vectors to train and give feedback to the LSTM with the following steps:

1. Pass the two-dimensional cross section of the training vector that represents the first string [0,.,.] to the LSTM.

2. LSTM generates a one-dimensional zero vector. (the size of this vector will be equal to the amount of distinct characters.)
3. LSTM assigns the value 1 to the position in the zero vector that represents the proposed character in the character index dictionary.
4. Check the 1 dimensional cross section of the validation vector at the first character [0,.] for a match.
5. Inform the LSTM of the result of the check.
6. Repeat this process through all elements in each list.

Figure 3.8 displays a visual representation of an example pass of this loop.

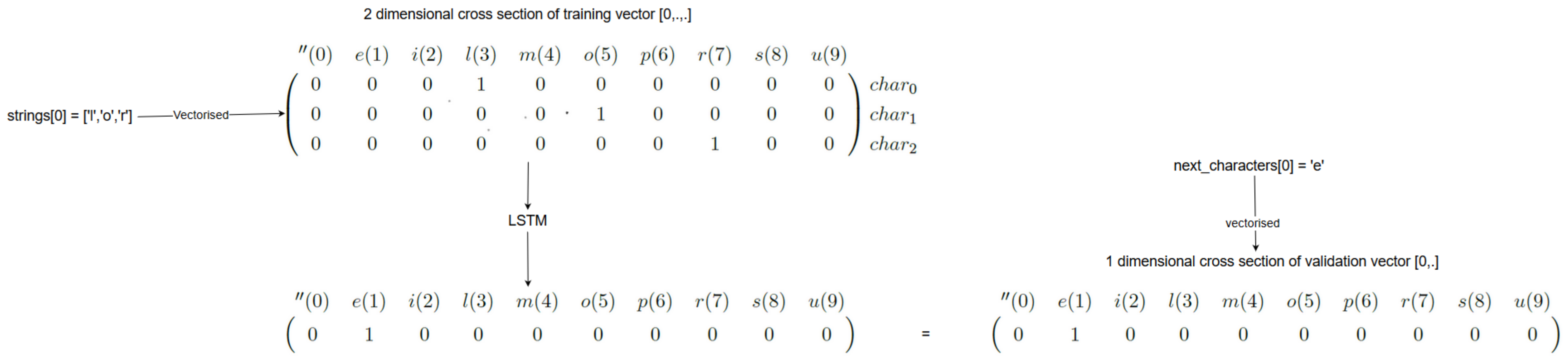

2 dimensional cross section of training vector [0,.,.]
strings[0] = ['l','o','r']
Vectorised
$\begin{array}{cccccccccc|l} "(0) & e(1) & i(2) & l(3) & m(4) & o(5) & p(6) & r(7) & s(8) & u(9) & \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & char_0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & char_1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & char_2 \end{array}$
LSTM
$\begin{array}{cccccccccc} "(0) & e(1) & i(2) & l(3) & m(4) & o(5) & p(6) & r(7) & s(8) & u(9) \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}$
=
next_characters[0] = 'e'
vectorised
1 dimensional cross section of validation vector [0,.]
$\begin{array}{cccccccccc} "(0) & e(1) & i(2) & l(3) & m(4) & o(5) & p(6) & r(7) & s(8) & u(9) \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}$

Figure 3.8: Example of a successful prediction from the LSTM

Chapter 4

Design and Implementation of The LSTM RNN

This chapter will discuss the framework created in order to create and test multiple permutations of models.

4.1 Model Building Framework

Initially, a Keras Sequential object is declared. Sequential is an API provided by Keras that enables the stacking of multiple layers of recurrent neural networks. The sequential API was chosen over the functional API as layers connecting to more than just the previous and next layers was not necessary for this project. Hence, the sequential API was the more intuitive and time efficient choice. The sequential model can now be utilised to add different variations of LSTM layers along with different hyperparameters.

After a model has completed training, it is then saved in a directory with a unique name. The saved models will then be loaded and tested for program synthesis. A more in depth discussion of this can be found in chapter 6.

4.2 Argument Parsing

All hyperparameters that the model can train on can be declared at run time, allowing for the quick change of parameters each time that a new model is trained.

Chapter 5

Testing Models For Program Synthesis

5.1 Reverse Mapping of The Character Index

In section 3.3 the character index dictionary was created to hold an integer value for each character. Now, since the model needs to convert the integer values back into characters for output, a reverse of this dictionary is required.

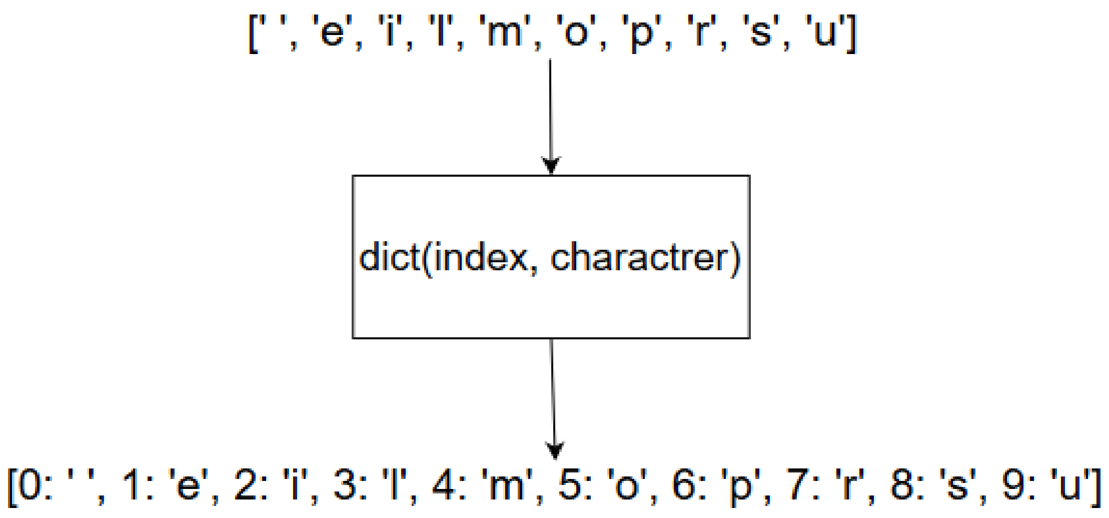

[' ', 'e', 'i', 'l', 'm', 'o', 'p', 'r', 's', 'u']
dict(index, charactrer)
[0: ' ', 1: 'e', 2: 'i', 3: 'l', 4: 'm', 5: 'o', 6: 'p', 7: 'r', 8: 's', 9: 'u']

Figure 5.1: shows how the index character dictionary is populated from an input list

5.2 Generating Output

This section discusses the functions required to extract the output from a saved model.

5.2.1 Seed String

Initially, an input string is required to 'seed' initial data to the model, i.e. give the model an initial string on which to make predictions. This initial string will be random and of the same length as the sequence length that the model was trained on. Figure 5.2 shows the steps involved for generating a random string from the input data set of the given sequence length.

```
current_string = input_data[start_index:  start_index + sequence_length]
```

5.2.2 Prediction Vector

The Initial random string needs to be one-hot-encoded into a zero vector just like in section 3.5. This is because we are once again going to pass data to the model in a readable format. Figure 5.3 shows how this vector is populated from the random string.

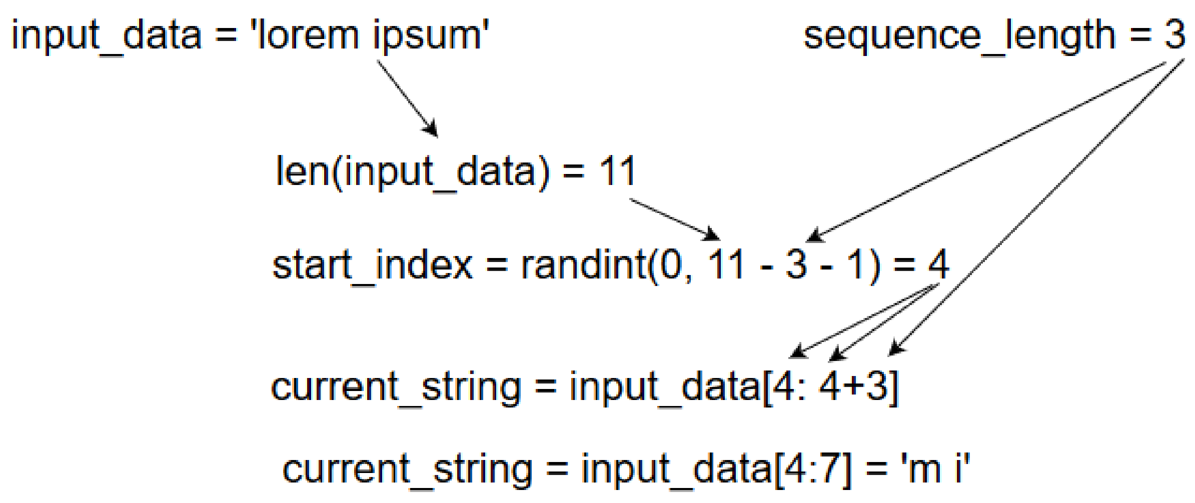

input_data = 'lorem ipsum'
sequence_length = 3
len(input_data) = 11
start_index = randint(0, 11 - 3 - 1) = 4
current_string = input_data[4: 4+3]
current_string = input_data[4:7] = 'm i'

Figure 5.2: Shows how `start_index` and `current_string` are populated

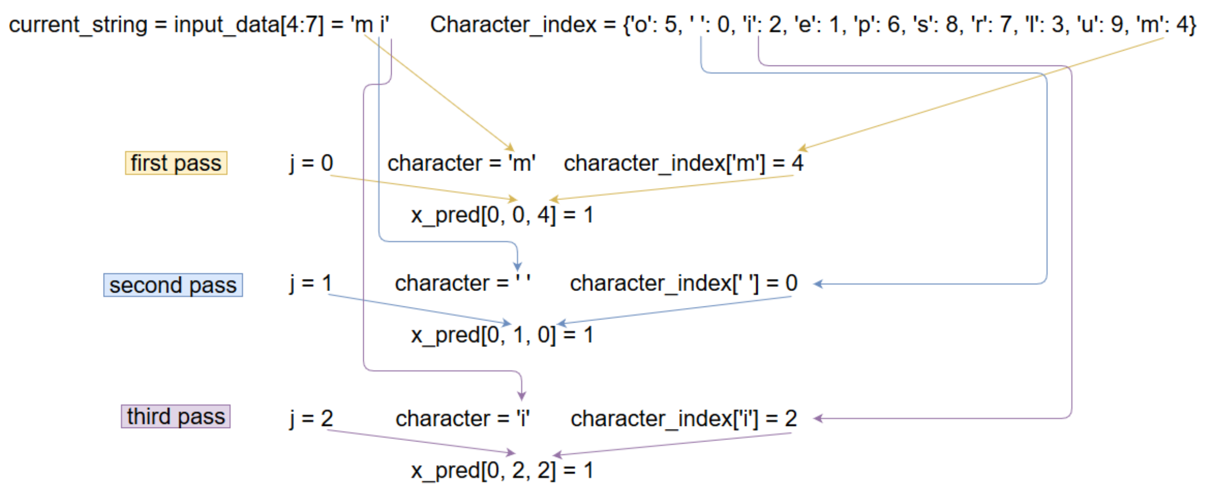


Figure 5.3: Example of the nested for loop populating the prediction vector

5.2.3 Predict Method

The Keras Model API offers a predict function that can be called on the saved model object. This function takes in the prediction vector, and returns a probability array of all the characters in the list of distinct characters (this array is sent for further processing explained in section 5.2.4). Since the list of distinct characters was sorted before creating the character index and index character dictionaries, the indexes of the probability array that the predict function returns corresponds to the indexes in our dictionaries. Hence, the index in the probability array with the maximum value is the index value in the index character dictionary of the character that the model has predicted. We pass this index value to the index character dictionary to retrieve the predicted character. Figure 5.4 shows an updated version of figure 3.8, where a probability array is returned, instead of a one hot vector.

5.2.4 Ensuring Random Output

Randomness of the predictions must be considered, as the model would simply repeat the same output over and over again without an element of randomness. To implement randomness

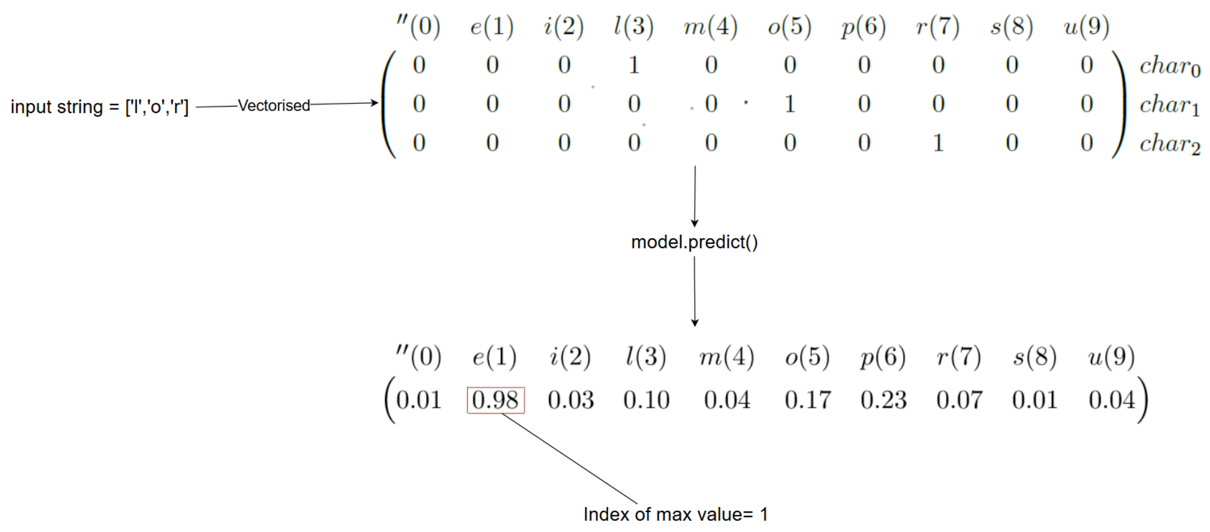

input string = ['l','o','r']
Vectorised
"(0) e(1) i(2) l(3) m(4) o(5) p(6) r(7) s(8) u(9)
0 0 0 1 0 0 0 0 0 0 char0
0 0 0 0 0 1 0 0 0 0 char1
0 0 0 0 0 0 0 1 0 0 char2
model.predict()
"(0) e(1) i(2) l(3) m(4) o(5) p(6) r(7) s(8) u(9)
0.01 0.98 0.03 0.10 0.04 0.17 0.23 0.07 0.01 0.04
Index of max value= 1

Figure 5.4: Example prediction vector

within the prediction array, the values within the array are divided by a given temperature, then passed through a multinomal distribution function. The multinomal distribution function is ran once on the array, i.e. there is one experiment. The position of the maximum value in the array after the multinomal distribution function is applied is our index for the predicted character.

5.2.5 Updating The Seed String

Once the model has predicted the next character and the index character dictionary has returned the character value, the seed string needs to have this character appended. This new string will then be passed to the model on which to make another prediction.
Since the model has been trained to give predictions on strings of a given string length - the act of appending the predicted character increases its length by 1. Therefore, the initial character of the seed string is removed to keep a constant string length of the sequence length that the model trained on. Figure 5.5 shows a diagram of how a seed string of length 3 is updated each iteration of a predicted character from the model.

5.2.6 Output String

Each character that the model predicts is appended to a string. After N predictions, the string will contain N characters.

5.3 Testing

This section will discuss the testing of different models. Test results can be found in Appendix A.

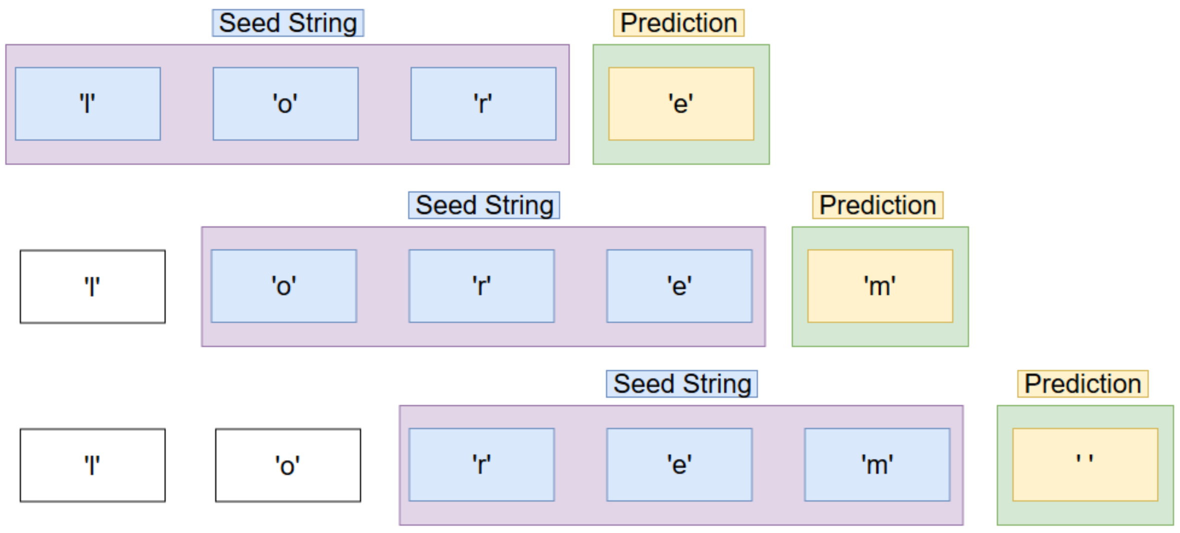


Figure 5.5: Example of updating the seed string

5.3.1 Testing Methodology

Since many permutations of models were tested, a simple table was designed to test each model. The titles of the table 5.1 are as follows:

Model	Legible	Syntax Wise	Compilable	Correct Start Position
dummy model 1	✓	×	×	×
dummy model 2	✓	✓	×	×
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.

Table 5.1: Example of the test table

- Legible: The output of the model is legible i.e. words can be found within the output
- Syntax Wise: The output of the model has signs of basic syntax laws e.g. python indenting, bracket closures etc.
- Compilable: The output of the model has compilable code
- Correct Start Position: The output starts at the correct line of the desired output e.g. `import turtle`

This testing table allowed for a clear visual representation of the performance of the different models. It also allowed for the strongest models to be isolated and tweaked slightly to find the optimal model architecture. If a model scored a ✓in every column of the table, this model was then tested multiple times to ensure that the output was not that of luck.

5.3.2 Correct Start Position

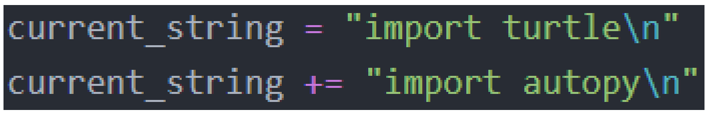
Despite some models scoring a ✓in three out of the four columns in the test table B.2, none of the models achieved a ✓in the final column - 'Correct Start Position' further tests on these models proved that the start position was inconsistent thus not producing the correct output. After spotting this in the testing table B.2 a decision was made to alter the process of obtaining the output. There are two solutions to this problem.

5.3.2.1 Solution 1: Forcing The Seed String

Since every file must start with the two lines of code:

- `import turtle`
- `import autopy`

and since the method of generating the seed string in section 5.2.1 selects a string at a random location within the input data, the decision to remove the randomness of this string was made. The altered solution was to force the input seed string to be equal to the two import statements, and then append the model's output to this string. A code snipped of this is shown in figure 5.6.



```
current_string = "import turtle\n"
current_string += "import autopy\n"
```

Figure 5.6: Code snipped of the forced seed string

5.3.2.1.1 Changing The Sequence Length The length of the seed string is now a fixed length of 28. Hence, the strongest models need to be retrained on a sequence length of 28 in order to make predictions on the forced seed string.

5.3.2.2 Solution 2: Overshooting Output

A different solution is to output more data than is needed, and to then trim this data. The models that achieved a ✓in the 'Compilable' column were outputting the correct code just at the wrong start position. So, if the variable N from section 5.2.6 is increased to a certain value, it is guaranteed that a subscript of the output is the desired output. Some code can then be applied to isolate the desired output.

5.3.2.2.1 Isolating Desired Output The consistency within the desired output can be used to isolate the desired output. For instance, the first line of the desired output will always be: `import turtle`. Hence, this string can be searched for in the output, and any text above this string will be deleted. Furthermore, there are always 16 lines of code in the desired output. This means that once the text above the `import turtle` line have been removed, any output on the 17th line or more will also be removed. This leaves us with the desired output.

5.3.3 Solution 1 vs Solution 2

The two possible solutions to the problem both have benefits and drawbacks. Table 5.2 compares these. Solution 2 was chosen as the optimal solution, due to the difference in output

Solution	Benefits	Drawbacks
Solution 1: Fixed seed	Output is correct length	Some of the output is hard coded
solution 2: Overshoot	All output generated	Unneeded characters produced

Table 5.2: Comparison of benefits and drawbacks of solutions

each time being deemed a key feature of the solution. Also, the waste of computation by printing extra unneeded characters (while being an unwanted feature) is not too bad considering the short length of the desired output itself.

5.3.4 Solution 2 Testing

The models that achieved a ✓in every column in the test table (apart from the correct start position column) (B.2) that were discussed in section 5.3.2 were re-tested with solution 2 5.3.2.2 The test results from these updated models can be found in Appendix C (table). All of the models can be considered a solution to the problem as they all achieve the desired output. Because of this, the model with the highest accuracy and lowest loss was chosen to be the optimal model.

5.4 Optimal model

The optimal model has an accuracy of 0.9673 and a loss of 0.09471. Figure 5.8 shows the output of the model that has been saved to a `.py` file. Figure 5.9 shows the image saved after executing the code in figure 5.8. The hyperparameters and model architecture can be found in appendix B.

5.4.1 Optimal Hyperparameters

Many permutations of Hyperparameters were attempted in the training of the model to find optimal accuracy and loss for this problem. This was made easier thanks to Google Colaboratory's high GPU speeds and Keras' built in activation functions, loss algorithms and optimisers - that can be interchanged via the change of single variables. Here is the structure of the optimal model achieved during testing:

- `sequence_length` = 40
- `step_size` = 3
- 2 LSTM layers
  - Input layer's input shape = (128 x 40 x 45)
  - 128 hidden units in each layer
  - Dropout rate of 20% after each layer
- Batch size of 128
- Output layer of size 45

- activation function = Softmax
- Optimizer = RMSProp
  - Learning rate = 0.01
  - Decay rate = 0.00001
- Loss function = Categorical crossentropy
- Number of epochs = 3
- Temperature = 1.2

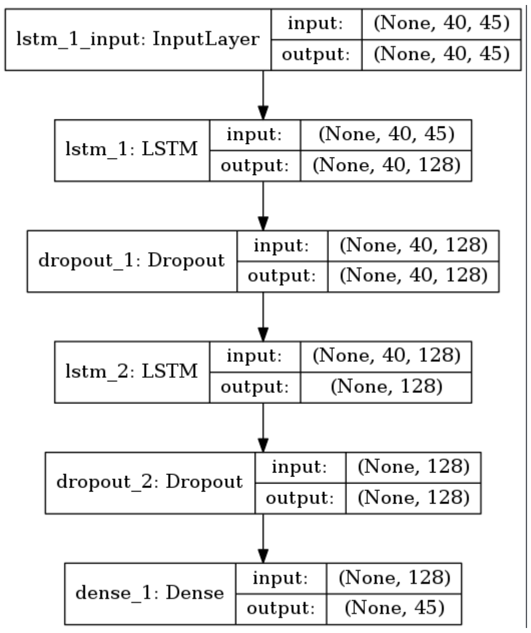
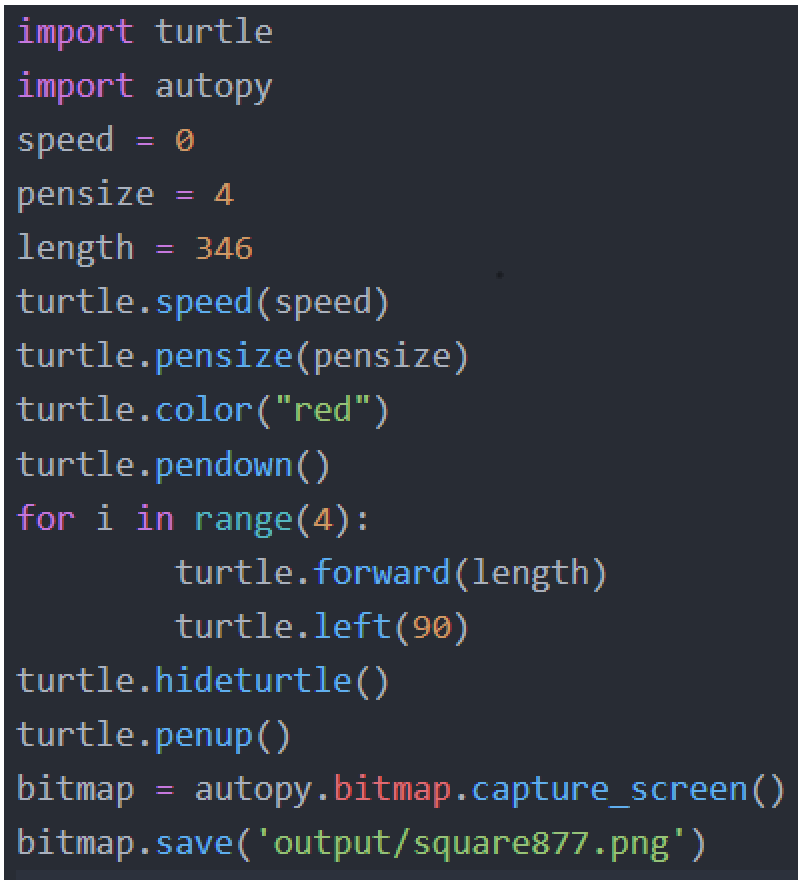


Figure 5.7: Structure of the optimal model



```
import turtle
import autopy
speed = 0
pensize = 4
length = 346
turtle.speed(speed)
turtle.pensize(pensize)
turtle.color("red")
turtle.pendown()
for i in range(4):
        turtle.forward(length)
        turtle.left(90)
turtle.hideturtle()
turtle.penup()
bitmap = autopy.bitmap.capture_screen()
bitmap.save('output/square877.png')
```

Figure 5.8: Optimal model output

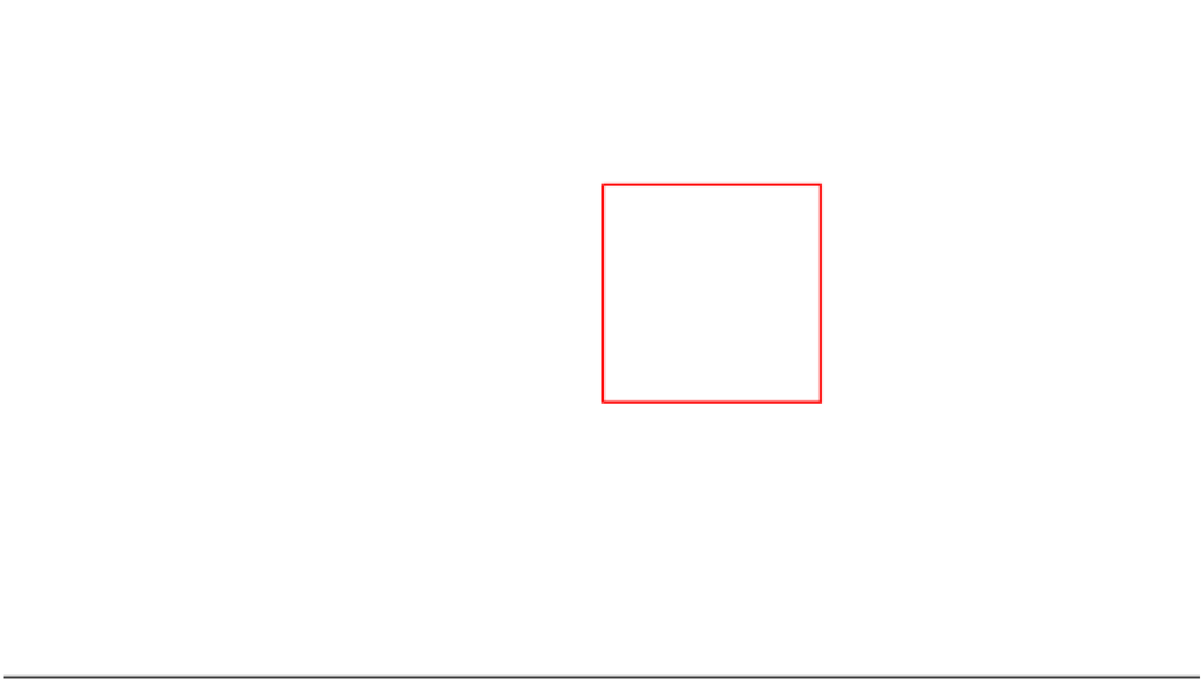


Figure 5.9: Saved image from the code in figure 5.8

Chapter 6

Evaluation

This section discusses the evaluation of myself, the project solution and also contains a section on how to use the developed program.

6.1 Evaluation of Deliverables

In section 1.4 A list of deliverables were written as the goal of this project. In this section I will discuss the effectiveness of each of these points.

6.1.1 Automated Python Script

I am satisfied with the python script that I created to generate multiple versions of ways to draw a square in the turtle graphics framework. I used random functions to ensure dissimilar versions of the file were created. The script can easily generate N amount of differing files.

6.1.2 Vectorising input data

The data generated in the automated python script was successfully vectorised into a one-hot encoded vector. This was achieved thanks to the character index dictionary that allowed for each distinct character to be mapped to an integer value. Then, the use of the index character dictionary allowed for the one-hot encoded vectors to be converted back into characters for output.

6.1.3 Model Training

When running the python file for training a new model, the use of an argument parser allowed for many of the arguments to be amended at run time, or to be left at their default values. This allowed for multiple models to be trained on different hyper parameter setups.

6.1.4 Model Output

The output of the model does fit the description of producing compilable python code to draw a square. However, the output could use some improvements.

- The model has not learnt the start position of the code output, this is forced.
- The length of each side of the square could be produced as too high a number.
- The name of the output file is random but is not guaranteed to be unique.

6.1.5 Report

I am satisfied with my approach to writing this report. The report was broken down into the different chapters, and then each chapter was subsequently broken down into smaller subsections. I implemented an agile approach to the report, by giving myself tasks that were to be completed within a certain time frame (sprints). First, the introduction and background research chapters were finalised before the implementation of any software, as the background research was required before development could proceed. Then, as the software development process began, progress on the report and the software ran in parallel. i.e. I would write sections on the report correlating to what I had implemented in the software. I was happy with this approach as both the report and the software were being worked on at the same time. This meant that I was not left with the entire report to write after the software implementation was completed. But instead, allowed for a more time effective method of continuous progress on the report and software.

6.2 Evaluation of Time management

The Gantt chart shown in figure 1.1 was the initial proposed time schedule for this project. However, some deviations to this plan occurred. For instance, deciding on the specific aim of the project took longer than expected as multiple versions of a problem similar to the final one proposed were discussed. Also, initially a Generative Adversarial Network was the planned neural network type to attempt this problem, rather than the LSTM that was finally implemented.

The use of Google Colabatory vastly improved the time efficiency of this project, and, due to having restricted access to the computational power of the university machines after going home for the Corona virus outbreak (Appendix E), Google Collabatory allowed for fast training of models and the ability to experiment with a greater range of parameters.

6.3 Code Structure

A modularised approach to the structure of the code was implemented for the completion of this project. The solution was split into three python files. A file for the generation of the training data, then a file that takes in the training data - vectorises it and trains the model - then this model is passed to a third python file that produces the output. Figure D.1 shows a visual representation of the layout of the coding structure of the project.

6.4 Self Reflection

I chose this project as I am interested in the possible capabilities of machine learning and neural networks in the near future. I also believed that this project would give a greater understanding of neural networks and the challenges in developing and training a successful model.

I expected many challenges such as; figuring out a way to generate the training data, how to validate the predicted characters that the model produces and how to extract these predictions from the model. I was excited to tackle these challenges and to take on an agile software development methodology to guide and track my development process. The agile methodology allowed for the project to be broken down into small, feasible steps and enabled me to focus on smaller tasks of the broader project and then bringing these together to build the final solution.

I have learnt many things during the process, such as; learning how to appropriately plan a project of this scope and scale, gaining more knowledge in the area of machine learning/neural networks and how to extract relevant information from research papers.

I am very pleased with the software implementation of the project as all of the deliverables discussed in 1.4 were implemented to some degree. However, despite being pleased with the result of the project, there are some things that I would have liked to do differently in hindsight. Firstly, I would have liked to come to a decision on the actual problem to solve at the beginning. The time that was spent theorising different possibilities for a potential project took longer than expected, leaving less time to implement a solution once a problem had been chosen. Also, once the problem had been chosen, multiple possible solutions were then considered until an evaluated decision in these possible solutions was made and an LSTM was deemed the most appropriate solution. If I were to do this project again I would have liked to have made a decision on the best possible solution sooner, by conducting background research in a quicker manner.

6.5 Ethical Issues Addressed

The generated code that is produced by the optimal model cannot be predicted. Therefore, some issues must be addressed. Firstly, the halting problem discussed in 2.7.1 states that a machine cannot determine if a program will halt or not, therefore, the code output may include an infinite loop. However, an infinite loop could be spotted by the user before running the output. Also, if a user was to run an infinite loop, the software would simply run until the process was killed and would not damage any hardware on the users machine. Another ethical issue that had to be taken into consideration was the fact that the output of the model is unknown until it has been produced. This gave me a responsibility of ensuring that no offensive or disrespectful text could be produced in the output.

6.6 Using The Program

This section will discuss the process of running the software in a Linux terminal. The source code required can be found at the github directory in Appendix C

6.6.1 Initial Setup

First, create a python 3 virtual environment:

```
python3 -m venv /path/to/new/virtual/environment
```

Then, to activate the virtual environment run:

```
source /path/to/new/virtual/environment/bin/activate
```

Now that the virtual environment has been activated ensure that pip is updated to the most recent version:

```
pip3 install -upgrade pip
```

Next, install the required packages:

```
pip3 install -r requirements.txt
```

After the packages have been installed the program can be ran. To deactivate the virtual environment simply execute the follwoing in the terminal:

```
deactivate
```

If the program is to be ran again, the virtual environment must be activated.

6.6.2 Generating Training Data

To Generate training data, run the file 'createTrainingData.py' with the command:

```
python3 createTrainingData.py
```

The number of files generated is defaulted at 1024, however this can be changed at the command line. For instance, if 3000 training files are required, the command line input is as follows:

```
python3 createTrainingData.py --num_of_files 3000
```

6.6.3 Training A Model

Once the training data has been generated, the 'train.py' file can be ran to train a model. This file is ran in the terminal with the command:

```
python3 train.py
```

This file has many command line parameters that can be amended to train different variations of models. Use the command:

```
python3 train.py --help
```

to view a list of different parameters that are available.
Once a model has been trained, it will be saved in the 'models' directory. It's name will be the date and time of execution. For example:

```
2020-05-10 10:30:42.3345262.h5
```

6.6.4 Model Output

To retrieve the output of a saved model, run the 'generateCode.py' file with the model name as a command line input:

```
python3 generateCode.py --model modelName.h5
```

This will save the output to a .py file in the output directory.

6.6.5 Running Output

To run the output, run the newly generated file in the output directory. For example:

```
python3 output232.py
```

This will run the program that the LSTM generated and will save the image of a square in the same directory.
Viewing TensorBoard Results Open a Terminal window, and enter the following:

```
tensorboard --logdir path/to/log/directory
```

This will give a link to a locally hosted web address to view the results of the saved logs.

Chapter 7

Conclusion

This chapter discusses a brief conclusion of the project and any future work that is intended.

7.1 Summary

Overall, I am very satisfied with the development and execution of this project. At the beginning of the project I was unsure if I would be able to find a solution to the problem given the amount of background research and time to implement that was required. I was pleased with the coding methodology that I implemented and how the use of command line arguments made for ease of use during the vigorous testing process. The optimal model that is saved can produce compilable code that draws a square and saves the image as a png.

7.2 Future Work

Since a solution to the problem of program synthesis was initially unknown, the decision was made for a very basic python program (draw a square and save the image) to be the desired output. Now that a solution has been found, I would like to experiment with the idea of a neural network for program synthesis on more complex programs. Also, some issues of the project still remain. The first issue is that the start position for the desired output has not been learned by the optimal model, instead, the model outputs more code than necessary, and this code is then trimmed to isolate the desired output. Another potential issue is that the output could have a `length` value that is too large, creating an image of a square that cannot entirely fit on the screen. Finally, the output file that is saved (which contains the model's generated output) will be named `outputN.py` where N is a random number between 1 and 9999. Potentially, this means that an output file of the same name could be produced. Thus leaving the possibility of non-unique file names. I aim to implement solutions to these issues in the future.

References

[1] turtle - turtle graphics.
`https://docs.python.org/3.3/library/turtle.html?highlight=turtle/`, Retrieved 2019-10-04.

[2] Tensorboard. `https://www.tensorflow.org/tensorboard`, Retrieved 2020-02-04.

[3] Keras lstm tutorial – how to easily build a powerful deep learning language model.
`https://adventuresinmachinelearning.com/keras-lstm-tutorial/`, Retrieved 24/10/19.

[4] O. Abdel-Hamid, M. Abdel-Hamid, J. Hui, D. Li, P. Gerald, and Y. Dong. Convolutional neural networks for speech recognition. *ACM TRANSACTIONS ON AUDIO, SPEECH, AND LANGUAGE PROCESSING*, 22(10):1533, 2014.

[5] H. Ablseon, N. Goodman, and L. Rudolph. Logo manual. 1974.

[6] Y. Bengio, P. Simard, and Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE trans, on Neural Networks*, 5(2):66–157, 1994.

[7] M. Bodén. A guide to recurrent neural networks and backpropagation. pages 1–12, 2001.

[8] B. Egeli, M. Ozturan, and B. Badur. Stock market prediction using artificial neural networks. *Proceedings of the 3rd International Conference on Business*, page 1, 2003.

[9] F. Gers, J. Schmidhuber, and F. Cummins. Learning to forget: Continual prediction with lstm. *Neural Computation*, 12(10):2451–2471, 2000.

[10] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. *Proceedings of the International Conference on Neural Information Processing Systems*, 12(10):2672–2680, 2014.

[11] P. Goyal, S. Pandet, and K. Jain. Unfolding recurrent neural networks. *Deep Learning for Natural Language Processing*, 15:119–168, 2018.

[12] C. Guc and F. Berkhahn. Entity embeddings of categorical variables. abs/1604. 06737, 2016.

[13] S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. *A Field Guide to Dynamical Recurrent Neural Networks*, page 15, 2001.

[14] J. Hopcroft and J. Ullman. Intorduction to automata theory, languages and computation 1st edition. page 148, 1979.

[15] N. Kant. Recent advances in neural program synthesis. pages 1–3, 2018.

[16] Y. Kim. Convolutional neural networks for sentence classification. *The 2014 Conference on Empirical Methods In Natural Language Processing*, page 1746, 2014.

[17] B. Kröse and P. van der Smagt. An introduction to neural networks. pages 13–14.

[18] M. Kusner and M. Hernandez-Lobato. Gans for sequences of discrete elements with the gumbel-softmax distribution. *ICLR 2019*, pages 1–7, 2016.

[19] S. Marsland. *Machine Learning, An Algorithmic Perspective SECOND EDITION*. Chapman Hall/CRC, 2014.

[20] W. Nie, N. Narodytska, and A. B. Patel. Relgan: Relational generative adversarial networks for text generation. *ICLR 2019*, pages 1–7, 2019.

[21] M. O'Halloran, B. McGinley, R. C. Conceicao, F. Morgan, E. Jones, and M. Glavin. Spiking neural networks for breast cancerclassification in a dielectrically heteroge-neous breast. *Progress In Electromagnetics Research*, 113:413–414, 2011.

[22] D. Pawade, A. Sakhapara, M. Jain, N. Jain, and K. Gada. Story scrambler -automatic text generation using word level rnn-lstm. *I.J. Information Technology and Computer Science*, 6(10):44–53, 2018.

[23] B. Sadia. Halting problem. *Journal of Problem Solving*, 01:5–6, 2016.

[24] A. Santoro, R. Faulkner, D. Raposo, J. Rae, M. Chrzanowski, T. Weber, D. Wierstra, O. Vinyals, R. Pascanu, and T. Lillicrap. Relational recurrent neural networks. *CoRR*, abs/1806.01822, 2018.

[25] M. Sipser. Introduction to the theory of computation second edition. page 137, 2016.

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

[27] A. Turing. Computable numbers with an application to the entscheidungs problem. *Proceedings of the London Mathematical Society*, v2:230–265, 1936.

[28] G. van Rossum, B. Warsaw, and N. Coghlan. Pep 8 – style guide for python code. `https://www.python.org/dev/peps/pep-0008/`, Retrieved 2019-10-07.

[29] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. pages 5998–6008, 2017.

[30] L. Yu, W. Zhang, and Y. Yu. Seqgan: Sequence generative adversarial nets with policy gradient. *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)*, pages 2852–2858, 2017.

Appendices

Appendix A

External Material

- Github: Repository for the software
- Keras: API for building and training models
- Turtle: Graphics library for the drawing of a square
- Autopy: Python library to screenshot and save image
- Draw.io: Online tool for diagram creations
- Google Colab: Online tool to train models
- TensorBoard: Tool for visualising model data

Appendix B

Test Results

First Testing Phase

Model	Legible	Syntax Wise	Compilable	Correct Start Position
model 1	✓	✓	✓	×
model 2	✓	✓	✓	×
model 3	✓	✓	✓	×
model 4	✓	✓	×	×
model 5	×	×	×	×
model 6	✓	✓	×	×
model 7	✓	✓	×	×
model 8	✓	✓	×	×
model 9	✓	×	×	×
model 10	✓	✓	×	×
model 11	✓	✓	✓	×

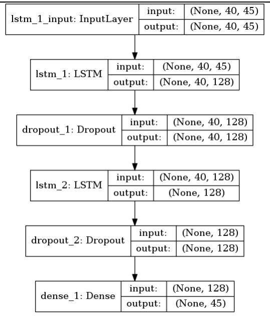
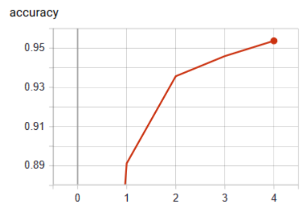
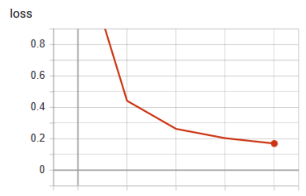
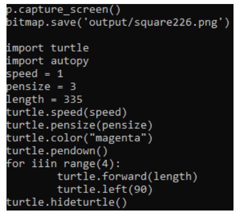
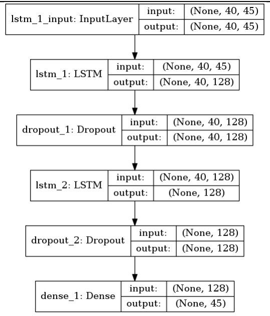
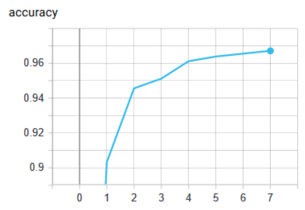
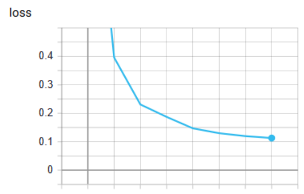
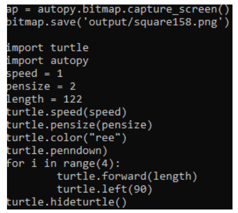
Table B.1: Test Table Phase 1

Second Testing Phase

Model	Legible	Syntax Wise	Compilable	Correct Start Position
model 1	✓	✓	✓	✓
model 2	✓	✓	✓	✓
model 3	✓	✓	✓	✓
model 11	✓	✓	✓	✓

Table B.2: Test Table Phase 2

*Each model implemented a dropout rate of 20% after each layer and used the categorical crossentropy loss function.

Model	Hyper-params	Architecture	Accuracy	Loss	Output
model 1	• LR: 0.01 • batch size: 128 • Units: 128 • sequence length: 40 • optimiser: RMSProp				
model 2	• LR: 0.001 • batch size: 128 • Units: 128 • sequence length: 40 • optimiser: RMSProp				

lstm_1_input: InputLayer | input: (None, 40, 45) | output: (None, 40, 45)
lstm_1: LSTM | input: (None, 40, 45) | output: (None, 40, 128)
dropout_1: Dropout | input: (None, 40, 128) | output: (None, 40, 128)
lstm_2: LSTM | input: (None, 40, 128) | output: (None, 128)
dropout_2: Dropout | input: (None, 128) | output: (None, 128)
dense_1: Dense | input: (None, 128) | output: (None, 45)

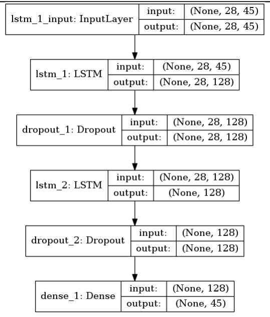
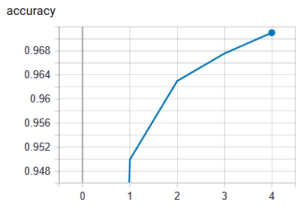
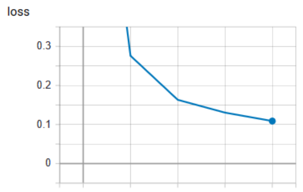
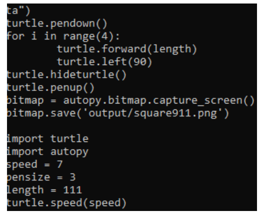
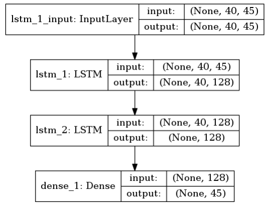
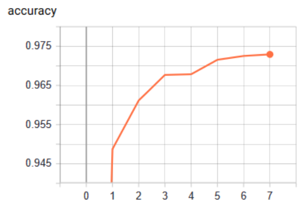
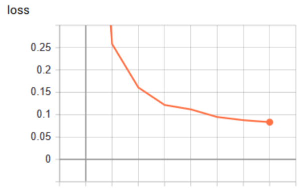
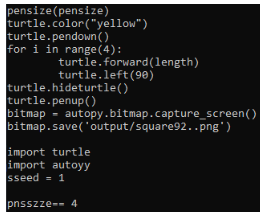
accuracy
0.95
0.93
0.91
0.89
0
1
2
3
4

```
p.capture_screen()
bitmap.save('output/square226.png')

import turtle
import autopy
speed = 1
pensize = 3
length = 335
turtle.speed(speed)
turtle.pensize(pensize)
turtle.color("magenta")
turtle.pendown()
for iiin range(4):
        turtle.forward(length)
        turtle.left(90)
turtle.hideturtle()
```

lstm_1_input: InputLayer | input: (None, 40, 45) | output: (None, 40, 45)
lstm_1: LSTM | input: (None, 40, 45) | output: (None, 40, 128)
dropout_1: Dropout | input: (None, 40, 128) | output: (None, 40, 128)
lstm_2: LSTM | input: (None, 40, 128) | output: (None, 128)
dropout_2: Dropout | input: (None, 128) | output: (None, 128)
dense_1: Dense | input: (None, 128) | output: (None, 45)

accuracy
0.96
0.94
0.92
0.9
0
1
2
3
4
5
6
7

```
ap = autopy.bitmap.capture_screen()
bitmap.save('output/square158.png')

import turtle
import autopy
speed = 1
pensize = 2
length = 122
turtle.speed(speed)
turtle.pensize(pensize)
turtle.color("ree")
turtle.penndown)
for i in range(4):
        turtle.forward(length)
        turtle.left(90)
turtle.hideturtle()
```

model 3	• LR: 0.001 • batch size: 128 • Units: 128 • sequence length: 28 • optimiser: RMSProp				
model 4	• LR: 0.001 • batch size: 128 • Units: 128 • sequence length: 40 • optimiser: Adagrad				

lstm_1_input: InputLayer | input: (None, 28, 45) | output: (None, 28, 45)
lstm_1: LSTM | input: (None, 28, 45) | output: (None, 28, 128)
dropout_1: Dropout | input: (None, 28, 128) | output: (None, 28, 128)
lstm_2: LSTM | input: (None, 28, 128) | output: (None, 128)
dropout_2: Dropout | input: (None, 128) | output: (None, 128)
dense_1: Dense | input: (None, 128) | output: (None, 45)

accuracy
0.968
0.964
0.96
0.956
0.952
0.948
0
1
2
3
4

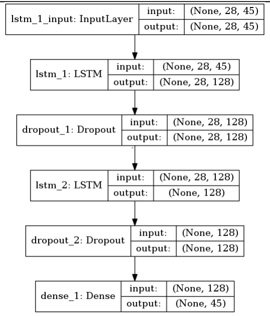
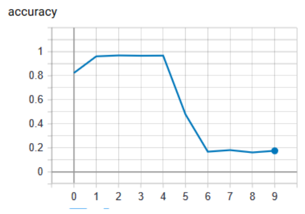
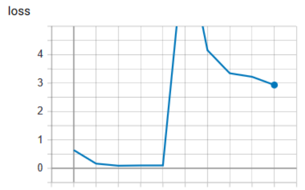
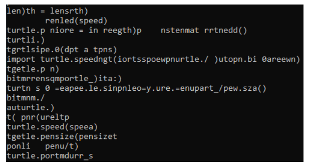
```
ta")
turtle.pendown()
for i in range(4):
        turtle.forward(length)
        turtle.left(90)
turtle.hideturtle()
turtle.penup()
bitmap = autopy.bitmap.capture_screen()
bitmap.save('output/square911.png')

import turtle
import autopy
speed = 7
pensize = 3
length = 111
turtle.speed(speed)
```

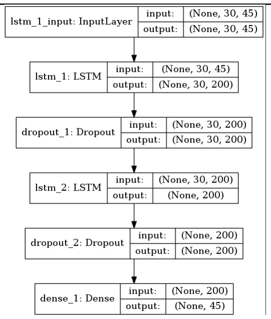
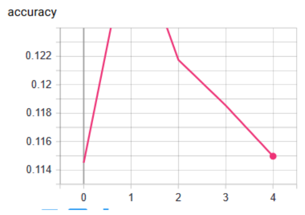
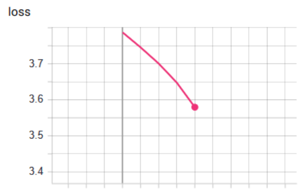
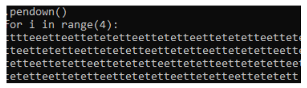
lstm_1_input: InputLayer | input: (None, 40, 45) | output: (None, 40, 45)
lstm_1: LSTM | input: (None, 40, 45) | output: (None, 40, 128)
lstm_2: LSTM | input: (None, 40, 128) | output: (None, 128)
dense_1: Dense | input: (None, 128) | output: (None, 45)

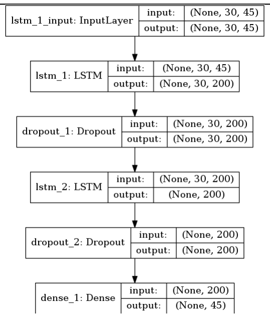
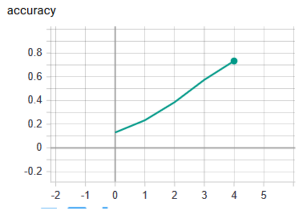
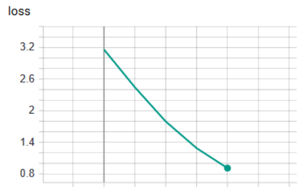
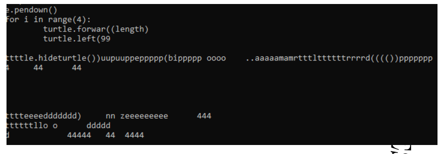
accuracy
0.975
0.965
0.955
0.945
0
1
2
3
4
5
6
7

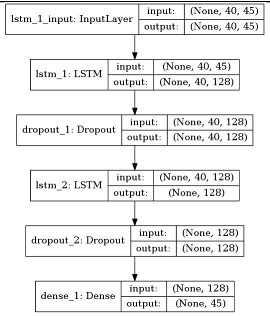
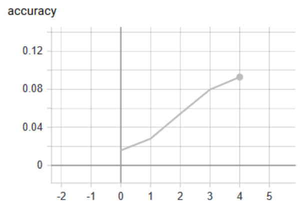
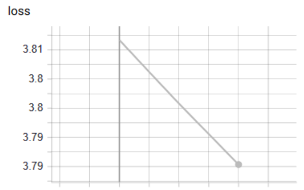
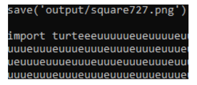
loss
0.25
0.2
0.15
0.1
0.05
0

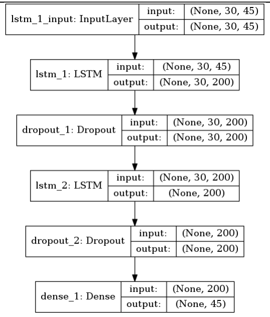
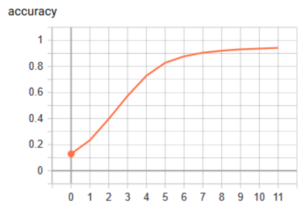
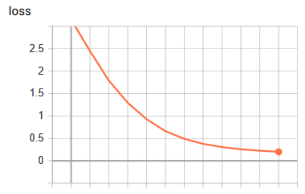
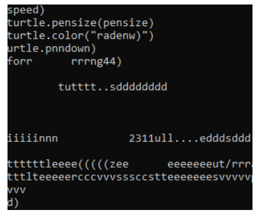
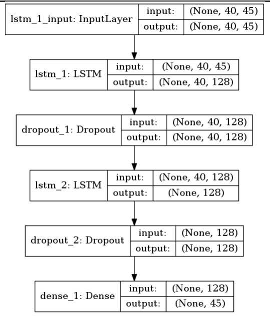
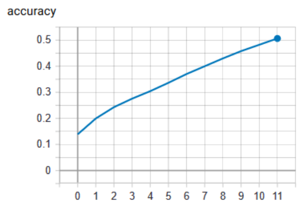
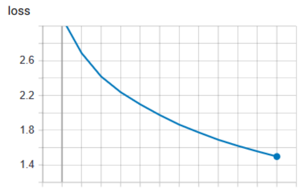
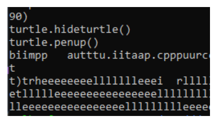
```
pensize(pensize)
turtle.color("yellow")
turtle.pendown()
for i in range(4):
        turtle.forward(length)
        turtle.left(90)
turtle.hideturtle()
turtle.penup()
bitmap = autopy.bitmap.capture_screen()
bitmap.save('output/square92..png')

import turtle
import autoyy
sseed = 1

pnsszze== 4
```

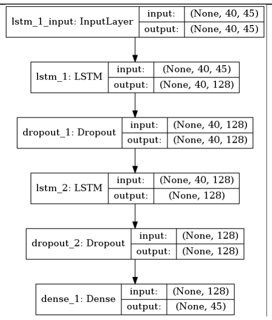
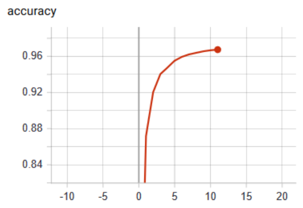
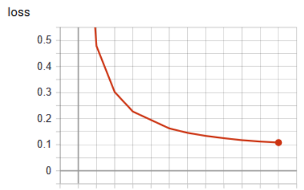
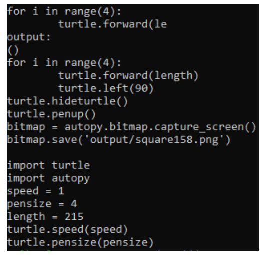
model 5	• LR: 0.0001
	• batch size: 128
	• Units: 128
	• sequence length: 28
	• optimiser: SGD
		 lstm_1_input: InputLayer
		input: (None, 28, 45)
		output: (None, 28, 45)
		lstm_1: LSTM
		input: (None, 28, 45)
		output: (None, 28, 128)
		dropout_1: Dropout
		input: (None, 28, 128)
		output: (None, 28, 128)
		lstm_2: LSTM
		input: (None, 28, 128)
		output: (None, 128)
		dropout_2: Dropout
		input: (None, 128)
		output: (None, 128)
		dense_1: Dense
		input: (None, 128)
		output: (None, 45)
					`len)th = lensrth)`
					`         renled(speed)`
					`turtle.p niore = in reegth)p    nstenmat rrtnedd()`
					`turtli.)`
					`tgrtlsipe.0(dpt a tpns)`
					`import turtle.speedngt(iortsspoewpnurtle./ )utopn.bi 0areewn)`
					`tgetle.p n)`
					`bitmrrensqmportle_)ita:)`
					`turtn s 0 =eapee.le.sinpnleo=y.ure.=enupart_/pew.sza()`
					`bitmnm./`
					`auturtle.)`
					`t( pnr(ureltp`
					`turtle.speed(speea)`
					`tgetle.pensize(pensizet`
					`ponli   penu/t)`
					`turtle.portmdurr_s`

model 6	• LR: 0.001
	• batch size: 248
	• Units: 200
	• sequence length: 30
	• optimiser: Adagrad
		 lstm_1_input: InputLayer
		input: (None, 30, 45)
		output: (None, 30, 45)
		lstm_1: LSTM
		input: (None, 30, 45)
		output: (None, 30, 200)
		dropout_1: Dropout
		input: (None, 30, 200)
		output: (None, 30, 200)
		lstm_2: LSTM
		input: (None, 30, 200)
		output: (None, 200)
		dropout_2: Dropout
		input: (None, 200)
		output: (None, 200)
		dense_1: Dense
		input: (None, 200)
		output: (None, 45)
					`.pendown()`
					`for i in range(4):`
					`ttttteeetteettetetetteettetetteettetetetteettetete`
					`tteettetetteettetetetteettetetteettetetetteettete`
					`tetteettetetteettetetetteettetetteettetetetteet`
					`tetetteettetetteettetetetteettetetteettetetett`

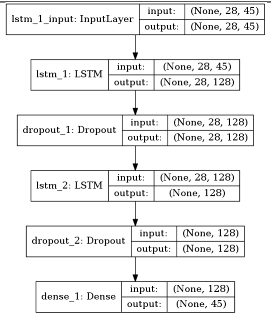
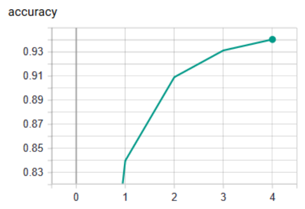
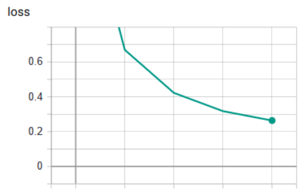
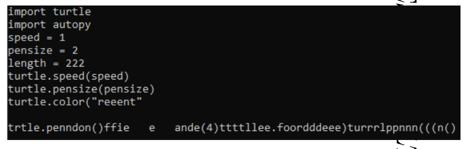
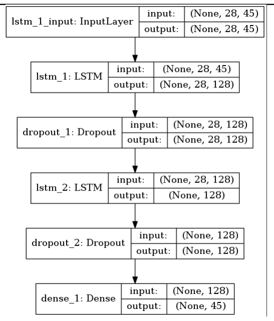
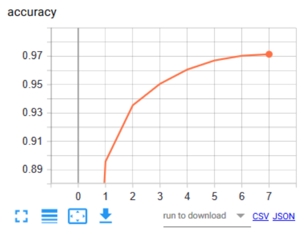
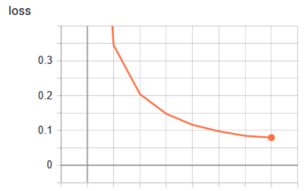
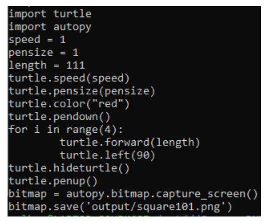
model 7	• LR: 0.001
	• batch size: 248
	• Units: 200
	• sequence length: 30
	• optimiser: RMSProp
		 lstm_1_input: InputLayer input: (None, 30, 45) output: (None, 30, 45)
		lstm_1: LSTM input: (None, 30, 45) output: (None, 30, 200)
		dropout_1: Dropout input: (None, 30, 200) output: (None, 30, 200)
		lstm_2: LSTM input: (None, 30, 200) output: (None, 200)
		dropout_2: Dropout input: (None, 200) output: (None, 200)
		dense_1: Dense input: (None, 200) output: (None, 45)
					`e.pendown()`
					`for i in range(4):`
					`        turtle.forwar((length)`
					`        turtle.left(99`
					`ttttle.hideturtle())uupuuppeppppp(bippppp oooo    ..aaaaamamrtttlttttttrrrrd(((())ppppppp`
					`4     44      44`
					`ttttteeeedddddd)     nn zeeeeeeeee      444`
					`tttttllo o      ddddd`
					`d            44444   44  4444`

model 8	• LR: 0.00005
	• batch size: 128
	• Units: 128
	• sequence length: 30
	• optimiser: SGD
		 lstm_1_input: InputLayer input: (None, 40, 45) output: (None, 40, 45)
		lstm_1: LSTM input: (None, 40, 45) output: (None, 40, 128)
		dropout_1: Dropout input: (None, 40, 128) output: (None, 40, 128)
		lstm_2: LSTM input: (None, 40, 128) output: (None, 128)
		dropout_2: Dropout input: (None, 128) output: (None, 128)
		dense_1: Dense input: (None, 128) output: (None, 45)
					`save('output/square727.png')`
					`import turteeeuuuuueueuuuueu`
					`uuueuuueuuueuuueuuueuuueuuue`
					`ueuuueuuueuuueuuueuuueuuueuu`
					`uuueuuueuuueuuueuuueuuueuuue`

model 9	• LR: 0.001 • batch size: 248 • Units: 200 • sequence length: 30 • optimiser: RMSProp	 lstm_1_input: InputLayer input: (None, 30, 45) output: (None, 30, 45) lstm_1: LSTM input: (None, 30, 45) output: (None, 30, 200) dropout_1: Dropout input: (None, 30, 200) output: (None, 30, 200) lstm_2: LSTM input: (None, 30, 200) output: (None, 200) dropout_2: Dropout input: (None, 200) output: (None, 200) dense_1: Dense input: (None, 200) output: (None, 45)			 speed) turtle.pensize(pensize) turtle.color("radenw)") urtle.pnndown) forr rrrng44) tutttt..sddddddddd iiiiinnn 2311ull....edddsddd ttttttleeee(((((zee eeeeeeeut/rrr tttlteeeeercccvvvsssccstteeeeeeesvvvvv vvv d)
model 10	• LR: 0.001 • batch size: 128 • Units: 128 • sequence length: 40 • optimiser: Adagrad	 lstm_1_input: InputLayer input: (None, 40, 45) output: (None, 40, 45) lstm_1: LSTM input: (None, 40, 45) output: (None, 40, 128) dropout_1: Dropout input: (None, 40, 128) output: (None, 40, 128) lstm_2: LSTM input: (None, 40, 128) output: (None, 128) dropout_2: Dropout input: (None, 128) output: (None, 128) dense_1: Dense input: (None, 128) output: (None, 45)			 90) turtle.hideturtle() turtle.penup() biimpp autttu.iitaap.cpppuurc t t)trheeeeeeeeellllllleeei rllll etllllleeeeeeeeeeeeeeeeellllllll lleeeeeeeeeeeeeeeeelllllllllleeee

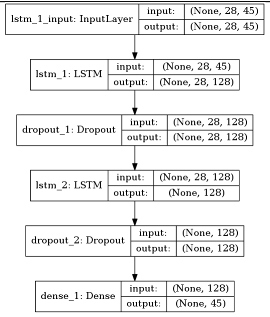
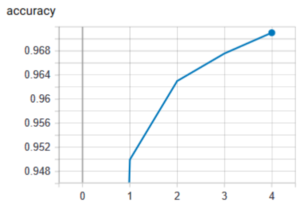
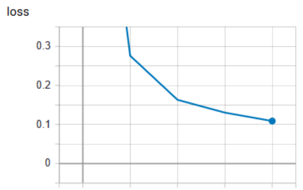
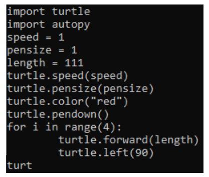
model 11	• LR: 0.01 • batch size: 128 • Units: 128 • sequence length: 40 • optimiser: Adagrad				

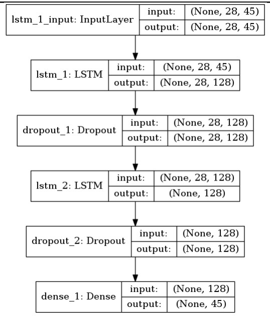
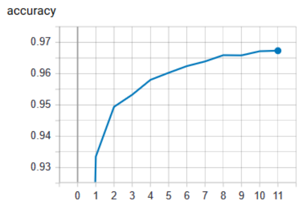
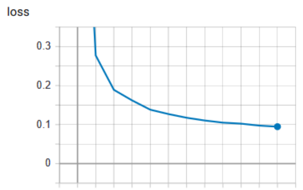
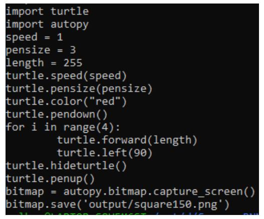
```
for i in range(4):
        turtle.forward(le
output:
()
for i in range(4):
        turtle.forward(length)
        turtle.left(90)
turtle.hideturtle()
turtle.penup()
bitmap = autopy.bitmap.capture_screen()
bitmap.save('output/square158.png')

import turtle
import autopy
speed = 1
pensize = 4
length = 215
turtle.speed(speed)
turtle.pensize(pensize)
```

Model	Hyper-params	Architecture	Accuracy	Loss	Output
model 1	• LR: 0.01 • batch size: 128 • Units: 128 • sequence length: 28 • optimiser: RMSProp	 lstm_1_input: InputLayer input: (None, 28, 45) output: (None, 28, 45) lstm_1: LSTM input: (None, 28, 45) output: (None, 28, 128) dropout_1: Dropout input: (None, 28, 128) output: (None, 28, 128) lstm_2: LSTM input: (None, 28, 128) output: (None, 128) dropout_2: Dropout input: (None, 128) output: (None, 128) dense_1: Dense input: (None, 128) output: (None, 45)			
model 2	• LR: 0.001 • batch size: 128 • Units: 128 • sequence length: 28 • optimiser: RMSProp	 lstm_1_input: InputLayer input: (None, 28, 45) output: (None, 28, 45) lstm_1: LSTM input: (None, 28, 45) output: (None, 28, 128) dropout_1: Dropout input: (None, 28, 128) output: (None, 28, 128) lstm_2: LSTM input: (None, 28, 128) output: (None, 128) dropout_2: Dropout input: (None, 128) output: (None, 128) dense_1: Dense input: (None, 128) output: (None, 45)			

```
import turtle
import autopy
speed = 1
pensize = 2
length = 222
turtle.speed(speed)
turtle.pensize(pensize)
turtle.color("reeent"

trtle.penndon()ffie   e   ande(4)ttttllee.foordddeee)turrrlppnnn(((n()
```

```
import turtle
import autopy
speed = 1
pensize = 1
length = 111
turtle.speed(speed)
turtle.pensize(pensize)
turtle.color("red")
turtle.pendown()
for i in range(4):
        turtle.forward(length)
        turtle.left(90)
turtle.hideturtle()
turtle.penup()
bitmap = autopy.bitmap.capture_screen()
bitmap.save('output/square101.png')
```

model 3	• LR: 0.001
	• batch size: 128
	• Units: 128
	• sequence length: 28
	• optimiser: RMSProp
		 lstm_1_input: InputLayer
		input: (None, 28, 45)
		output: (None, 28, 45)
		lstm_1: LSTM
		input: (None, 28, 45)
		output: (None, 28, 128)
		dropout_1: Dropout
		input: (None, 28, 128)
		output: (None, 28, 128)
		lstm_2: LSTM
		input: (None, 28, 128)
		output: (None, 128)
		dropout_2: Dropout
		input: (None, 128)
		output: (None, 128)
		dense_1: Dense
		input: (None, 128)
		output: (None, 45)
					import turtle
					import autopy
					speed = 1
					pensize = 1
					length = 111
					turtle.speed(speed)
					turtle.pensize(pensize)
					turtle.color("red")
					turtle.pendown()
					for i in range(4):
					        turtle.forward(length)
					        turtle.left(90)
					turt

model 11	• LR: 0.01
	• batch size: 128
	• Units: 128
	• sequence length: 28
	• optimiser: Adagrad
		 lstm_1_input: InputLayer
		input: (None, 28, 45)
		output: (None, 28, 45)
		lstm_1: LSTM
		input: (None, 28, 45)
		output: (None, 28, 128)
		dropout_1: Dropout
		input: (None, 28, 128)
		output: (None, 28, 128)
		lstm_2: LSTM
		input: (None, 28, 128)
		output: (None, 128)
		dropout_2: Dropout
		input: (None, 128)
		output: (None, 128)
		dense_1: Dense
		input: (None, 128)
		output: (None, 45)
					import turtle
					import autopy
					speed = 1
					pensize = 3
					length = 255
					turtle.speed(speed)
					turtle.pensize(pensize)
					turtle.color("red")
					turtle.pendown()
					for i in range(4):
					        turtle.forward(length)
					        turtle.left(90)
					turtle.hideturtle()
					turtle.penup()
					bitmap = autopy.bitmap.capture_screen()
					bitmap.save('output/square150.png')

Appendix C

Github Repository

https://github.com/calkey/LSTM-Program-Synthesis

Appendix D

Code Structure

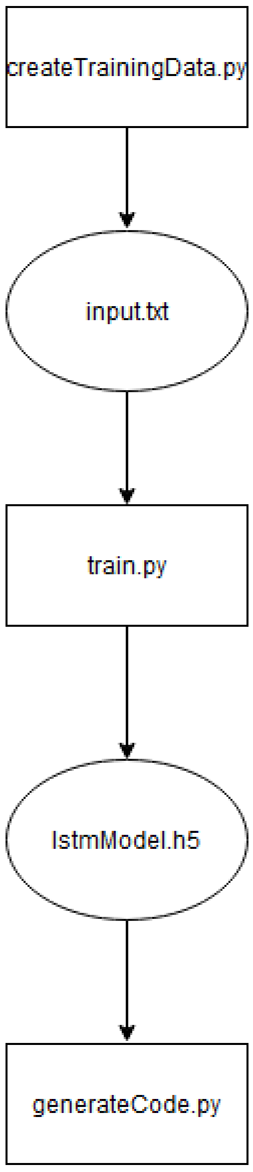


Figure D.1: Code Structure

Appendix E

Impact of COVID-19

The Corona Virus outbreak limited my ability to meet face to face with my supervisor and also prohibited me from utilising the universities computers.