

# School of InfoComm Technology

**Deep Learning Assignment**

Diploma in CSF / FI / IT

Apr 2022 Semester

**ASSIGNMENT 2**

(40% of DL Module)

4th Jul 2022 – 12th Aug 2022

**Submission Deadline:**

**Presentation: 12th Aug 2022 (Week 17),**

**Report: 12th Aug 2022 (Friday), 11:59PM**

|  |  |  |
| --- | --- | --- |
| **Tutorial Group** | **:** | **P02** |
| **Student Name** | **:** | Low Jun Jie, Ryan |
| **Student Number** | **:** | S10222186E |

**Penalty for late submission:**

10% of the marks will be deducted every calendar day after the deadline.

**NO** submission will be accepted after 21st Aug 2022 (Sunday), 11:59PM.

# Overview

The scope of this assignment is to create a classification model to generate English language text characters from a user given input. The model will utilize Recurrent Neural Networks (RNNs) in order to learn and generate characters using a given dataset, which was the first Harry Potter book. This report will elaborate on the steps taken to develop the model and showcase the results and findings of this process.

The main objective is to try and get a reasonably high accuracy for the model through various forms of tweaking and training the model, so that it can generate characters that will form as many coherent words and sentences as possible.

For this project, RNNs were chosen as the main neural network to build the models over other networks such as Convolutional Neural Networks (CNNs). RNNs are a type of neural network that are mainly used for processing sequential or time series data. Examples of sequential data include audio data, which is a natural sequence, or text data, which can be broken down into a sequence of words or characters. RNNs are commonly used in applications that require language translation, Natural Language Processing (NLP), and even speech recognition. They outperform networks such as CNNs in this department as they have a looping mechanism within their hidden layers, which allows them to retain past information and use it for future inputs.

In this case, an RNN model is used to process a large amount of text that will be broken down into sequences of words or characters. As mentioned briefly earlier, this model will use the first Harry Potter book as a form of training, validation, and testing data. The book is in the form of a text file, which will be loaded and processed into usable data for the model.

A single RNN model will be used to generate the text characters. It will start with the standard infrastructure of a neural network as a baseline, using a minimal number of nodes and RNN layers. The baseline model will use SimpleRNN as its RNN layer, which provides the most basic functions of an RNN without much complexity. The model will then slowly be scaled up to a larger network, consisting of more parameters and more complex RNN layers such as Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRU), until the accuracy of the model seems to remain stagnant. When that accuracy point is reached, the hyperparameters of the model will be tweaked. Hyperparameters are basically the “settings” of the model and will be changed accordingly after each model is trained. They are meant for experimenting to observe which changes will result in a better or worse model, where changes are typically applied after observing a trained model’s performance. Hyperparameters can include the number of nodes in a layer, the loss optimizer function for the model, the learning rate of the optimizer, and so on.

After a model is trained, a graph showing the accuracy and loss of the model will be plotted to observe the model’s performance. The model will also generate 400 text characters based on a random sample of text taken from the book, which is used to simulate how it would perform if it were to be used.

The models will be observed according to their performances compared against each other. The model with the best performance will be selected as the model to generate the text characters based on a user input.

# Data Loading and Processing

In order for the model to be able to process the text properly, we need to load and process the data first before feeding it into the model, otherwise there might be errors and performance issues while training. Starting off with the data loading, the text file directory is located, and the file is opened in the Jupyter Notebook by using the Python function open().

Scatter chart

Description automatically generated

Fig 2.1: Data Loading.

The file is opened using ‘rb’, which stands for ‘read binary’. This means that the file is opened in a binary format, consisting of 1s and 0s, so that the machine can read it. In the same line, the file is encoded using UTF-8, which can convert a character into a string of binary and vice versa (binary converted into string). In this case, the binary values in the text file are converted back into characters. The number of unique characters and the characters themselves are then printed out at the bottom.

Next, for the data processing, the raw data is first cleansed before separating them into sequences and sentences, so text such as the headers and footers of the file are removed first. In the code shown below, the page numbers and headings for each page are removed using a ‘for’ loop.

Scatter chart

Description automatically generated

Fig 2.2: Data Processing.

After the ‘for’ loop, the code replaces the “/” character with another letter. This was added because in the text file, it was found that some letters were replaced by the “/” character without making sense in the book’s context, as shown in the images below. This was assumed to be a mistake or an error in the file, so the code was added to amend the errors.

Graphical user interface, text, application

Description automatically generatedGraphical user interface, text

Description automatically generated

Fig 2.3: Example of the Letter “I” replaced by a “/” character .

After the initial cleansing of data was performed, the next step was to obtain sequences of data from the text. This was done by defining the maximum length of the extracted character sequences and the number of characters it would take for another sequence to be built.

Graphical user interface, text, application

Description automatically generated

Fig 2.4: Creating Sequences of Data.

Initially, a sequence of 60 characters will be created, which will be the first 60 characters of the book. A sequence is a string of text from the text data, where each sequence is created after every ‘step’ number of characters, which is an adjustable variable in the code. Each sequence will contain the previous sequence and a ‘step’ number of characters added to the end. There is a sentences list that contains all the sequences that are extracted from the data, which will be used in the one-hot encoding process later on. Code is used to sort and print out each unique character inside the text data and assign the unique characters to an index number.

Next, the one-hot encoding process will be used to encode the characters into binary arrays.

Text, letter

Description automatically generated

Fig 2.5: One-hot Encoding of Data.

Variables x and y are created, where they are arrays consisting of 1s and 0s. Each character in the sentences is then assigned to a binary array of 1s and 0s within these x and y variables. With one-hot encoding, each character will be converted into binary arrays so that the model can process the data. An example of how one-hot encoding works is as shown below.

Diagram

Description automatically generated

Fig 2.6: Example of One-hot Encoding.

# Develop the Sequence Generator Model(s)

## Model 1 – SimpleRNN Model

The first model will be a baseline model using the SimpleRNN layer. It has 1 layer of SimpleRNN with 64 nodes, fed into a dense layer using the softmax activation function, which is used for classifying multiple categories.

Text

Description automatically generated

Fig 3.1.1: Model 1 Configuration.

The optimizer is RMSProp with a learning rate of 0.01. RMSProp is a standard optimizer used for gradient descent. The model is compiled with the RMSProp optimizer and a categorical crossentropy loss function, as this model is categorizing more than 2 classes.

Text

Description automatically generated

Fig 3.1.2: Model 1 Fitting.

The model will be trained with a batch size of 128 and validation split of 0.2 for 20 epochs.

Graphical user interface, text

Description automatically generated

Fig 3.1.3: Model 1 Parameters.

The model has around 11,000 parameters as shown in the model summary.

Graphical user interface

Description automatically generated with medium confidence

Fig 3.1.4: Model 1 Results.

The model peaked at around 0.47 validation accuracy and starts to overfit at around 10 epochs. Code as shown below is used to generate text with the model.

Text

Description automatically generated

Text

Description automatically generated

Fig 3.1.5: Code to Generate 400 Text Characters.

This code allows the model to generate text characters. It generates a random seed of text for the model to process so it can generate text. The randomness of these generations is defined by softmax temperature values, where higher temperature values make the outcome more random. In this case, temperatures of 0.2 and 0.5 were used.

Text

Description automatically generated

Fig 3.1.6: Model 1 Text Generated.

This is the text generated by the model for its last epoch. The text generated at both temperatures do not make much sense in terms of the sentences, and words such as “’harry” were repeated multiple times. The model’s parameters will be changed for better results.

## Model 2 – LSTM Model

The next model will use LSTM instead of SimpleRNN. LSTM stands for Long Short-Term Memory and is a more complex version of SimpleRNN. LSTM can retain input data better than SimpleRNN, as they have a mechanism which allows past information to be reinjected at a later time. LSTM has parameters which allows them to learn, unlearn or retain information. This counters the vanishing gradient problem, where information learnt many steps ago would be lost.

Text

Description automatically generated

Fig 3.2.1: Model 2 Configuration.

For this LSTM model, the same number of nodes (64) will be used as the SimpleRNN model, with no changes to any other hyperparameters like the batch size and validation split.

A picture containing text

Description automatically generated

Fig 3.2.2: Model 2 Parameters.

Despite having the same number of nodes as the SimpleRNN layer, there are 34,000 parameters using LSTM instead of 11,000 with SimpleRNN, showing how much more complex LSTM is compared to SimpleRNN.

Graphical user interface

Description automatically generated with medium confidence

Fig 3.2.3: Model 2 Results.

The validation accuracy is better than the SimpleRNN model, averaging at around 0.54 accuracy. This model barely overfits with a stagnant validation loss, so a more complex model would help show its full potential.

Text

Description automatically generated

Fig 3.2.4: Model 2 Text Generated.

The text generations are better than the previous model, where the words generated here are more complete and less repetitive.

## Model 3 - GRU Model

For the next model, a Gated Recurrent Unit (GRU) layer will be implemented. GRU is similar to LSTM, as they both have a unique mechanism to store and re-use past inputs. Unlike LSTM, GRU has 2 vectors in its layers that can retain information for a long time and decide which information should be passed to the output.

Text

Description automatically generated

Fig 3.3.1: Model 3 Configuration.

This model will use 1 GRU layer consisting of only 64 nodes. The rest of the hyperparameters are unchanged.

A picture containing graphical user interface

Description automatically generated

Fig 3.3.2: Model 3 Parameters.

It is observed that GRU is slightly less complex than LSTM, with around 26,000 parameters as compared to 34,000 using LSTM.

Chart

Description automatically generated with low confidence

Fig 3.3.3: Model 3 Results.

The GRU model performs similarly to the LSTM model. The GRU model overfits slower than the LSTM model but has a lower validation accuracy.

Text, letter

Description automatically generated

Fig 3.3.4: Model 3 Text Generated.

The text generated is also similar to the LSTM model, with somewhat understandable sentences and mostly real words.

The LSTM model had a higher accuracy than both the SimpleRNN and the GRU model, so it will be improved on further.

## Model 4 - Increasing Complexity of LSTM Model

This model will increase the complexity of the LSTM model to try and achieve a better accuracy.

Text

Description automatically generated

Fig 3.4.1: Model 4 Configuration.

An additional LSTM layer consisting of 128 nodes is added. The number of epochs is increased from 20 to 30 to better observe any changes over time. Other than epochs, the rest of the hyperparameters remain unchanged.

To train the model, a “return\_sequences = True” statement is added to the first layer to return the hidden state to keep its output shape the same. For models with multiple RNN layers, this statement must also be present in all layers except the last RNN layer.

Text

Description automatically generated with low confidence

Fig 3.4.2: Model 4 Parameters.

The number of parameters increased from around 34,000 to over 136,000.

**Chart

Description automatically generated**

Fig 3.4.3: Model 4 Results.

This model shows improvement, as it is able to hit 0.55 accuracy and starts to overfit at around 3 epochs.

Text, letter

Description automatically generated

Text

Description automatically generated

Fig 3.4.4: Model 4 Text Generated.

The text in the pink highlighted box is a warning message about a function, but it does not affect any results, so it can be ignored for now.

From the text generation results, it is hard to tell if there is improvement, but the sentences of the 0.2 text generation somewhat made more sense than before.

## Adding Dropout and More Layers for LSTM Model

For this model, more layers are added to increase the complexity of the model for higher accuracy. Dropout is also added to counter overfitting. It randomly drops nodes from the previous layer so that the results will be more random.

Text

Description automatically generated

Fig 3.5.1: Model 5 Configuration.

Another LSTM layer with 256 nodes was added, and a dropout layer with a value of 0.4 is added before the dense layer. The other hyperparameters remain unchanged.

Table

Description automatically generated with medium confidence

Fig 3.5.2: Model 5 Configuration.

The model now has over 530,000 parameters as compared to over 136,000 previously.

Chart, scatter chart

Description automatically generated

Fig 3.5.3: Model 5 Results.

We can see that the model peaked at almost 0.55 accuracy, but for the rest of the epochs the accuracy started dropping and rising. This might be caused by vanishing gradient, where data from previous inputs might be lost or forgotten due to too much data trying to get stored in the layers, hence the large spikes in the validation accuracy and loss.

Text

Description automatically generated

A picture containing text

Description automatically generated

Fig 3.5.4: Model 5 Text Generated.

For the text generation, the text generated was unintelligible, as it was based on the final epoch which performed badly with 0.30 validation accuracy.

## Model 6 – Removing Dropout, Adding Weight Regularization

Weight regularization is added to reduce overfitting and potentially give higher accuracy. Dropout is removed to see if weight regularization will have an effect on accuracy.

Text

Description automatically generated

Fig 3.6.1: Model 6 Configuration.

L2 weight regularization of 1e-4 was added to the final LSTM layer. Nothing else in the model was changed.

Chart

Description automatically generated

Fig 3.6.2: Model 6 Results.

Similar to the model with dropout, the validation accuracy peaks at near 0.55, then starts dropping and fluctuating after 5 epochs.

Text, letter

Description automatically generated

Fig 3.6.3: Model 6 Text Generated.

The text generated was based off the final epoch, so the results are only slightly coherent as the last epoch only had around 0.50 validation accuracy.

## Model 7 – Removed Weight Regularization

This model will remove weight regularization to see if the model’s instability was caused by anything else.

Text

Description automatically generated

Fig 3.7.1: Model 7 Configuration.

The number of parameters is kept the same as no layers were changed. The rest of the hyperparameters are also kept the same.

Graphical user interface, chart

Description automatically generated

Fig 3.7.2: Model 7 Results.

It can be concluded that the extra LSTM layer was the factor that caused the model to be unstable in later epochs. It is suspected that it was hard for the model to perform recurrent functions as there were too many layers or nodes to go through. The overfitting measures will be added and compared later to see which ones would give the best results.

A picture containing calendar

Description automatically generated

Fig 3.7.3: Model 7 Text Generated.

For the text generation, the text is unintelligible as the final epoch had poor accuracy.

## Model 8 - Removing 1 LSTM Layer, Increased Nodes in Remaining LSTM Layers

This model aims to get higher accuracy and a more stable model performance than the previous model by reducing the number of layers.

Text

Description automatically generated

Fig 3.8.1: Model 8 Configuration.

An LSTM layer was removed and the nodes in the remaining LSTM layers were doubled to compensate for fewer parameters. The epochs are increased from 30 to 40 to further observe changes in the model.

Text, table

Description automatically generated with medium confidence

Fig 3.8.2: Model 8 Parameters.

There are now around 500,000 parameters as compared to 536,000 before.

Chart, scatter chart

Description automatically generated

Fig 3.8.3: Model 8 Results.

The validation accuracy of the model peaks above 0.55 before dropping due to vanishing gradient at around 13 epochs. The model can be stopped early to obtain the highest accuracy.

Text

Description automatically generated

Fig 3.8.4: Model 8 Text Generated.

For the text generation, the text is unintelligible as the final epoch had poor accuracy.

## Model 9 – Increased Complexity of Model

This model will have increased complexity to try and reach a higher accuracy.

Text

Description automatically generated

Fig 3.9.1: Model 9 Configuration.

The number of nodes in both layers are now doubled. There are no changes to other hyperparameters except for epochs which is reduced from 40 to 20. It was observed that the accuracy kept dropping after 20 epochs, so this was done to save time.

Text

Description automatically generated with low confidence

Fig 3.9.2: Model 9 Parameters.

The parameters went from around 536,000 before to almost 2 million now.

Graphical user interface, chart

Description automatically generated

Fig 3.9.3: Model 9 Results.

The validation accuracy could only barely peak at 0.55, before gradually dropping due to overfitting which occurred at 2.5 epochs.

Graphical user interface, text

Description automatically generated

Fig 3.9.4: Model 9 Text Generated.

The text is only somewhat intelligible as the final epoch of the model ended at around 0.52 accuracy.

## Model 10 – Reduced Complexity of Model 9

The previous model was too complex, resulting in lower accuracy. This model will be less complex than the previous model.

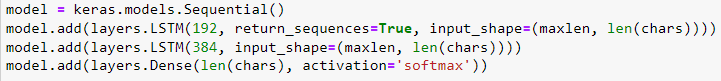


Fig 3.10.1: Model 10 Configuration.

The number of nodes in each LSTM layer were changed to 192 and 384 respectively. The rest of the hyperparameters remain unchanged.

Text

Description automatically generated

Fig 3.10.2: Model 10 Parameters.

The model now has almost 1.1 million parameters as compared to 2 million before.

Graphical user interface

Description automatically generated

Fig 3.10.3: Model 10 Results.

The validation accuracy of the model can reach over 0.55 accuracy before dropping at around 6 epochs due to vanishing gradient. The text generated is unintelligible as the last epoch ended at around 0.45 validation accuracy.

## Model 11 – Adam Optimizer

This model will use the Adam optimizer instead of RMSprop with the same learning rate to see which gives better accuracy. All other hyperparameters are unchanged in this model from Model 10.



Fig 3.11.1: Model 11 Using Adam Optimizer.

Adam is an adaptive optimizer for gradient descent, where it changes the gradient based on the current gradient. It is widely regarded to be a good option for almost any problem because of its adaptiveness and how it performs.

Chart, line chart

Description automatically generated

Fig 3.11.2: Model 11 Results.

The validation accuracy is very similar to the RMSprop model, but the model performs very smoothly over the 20 epochs with no major spikes in accuracy or loss.

Graphical user interface, text, application

Description automatically generated

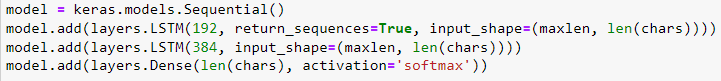
Fig 3.11.3: Model 11 Text Generated.

The text generated is somewhat comprehensible as the last epoch was just below 0.55 accuracy.

## Hyperparameter Tests

As seen earlier from the Adam optimizer model, the better model was undeterminable unless numbers were compared, or a direct graph comparison was involved. Hence, models with different hyperparameter configurations will be directly plotted against each other and compared to see which configuration would give the highest accuracy. The best model in every test will be used to carry out the next test.

The baseline model that will be tested with these different hyperparameters will be Model 10, where its current network is shown below.



Text

Description automatically generated

Fig 3.12.1: Model 10 Network.

Functions will be defined for the testing of these hyperparameters. These functions include creating and running models, model configurations, and so on, which will make changing the hyperparameters easier.

Text

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Text

Description automatically generated Text, letter

Description automatically generated

Text

Description automatically generated

Fig 3.12.2: Functions Defined.

Lastly, as many models are being trained, comparing the validation accuracy and loss of the models would be more accurate than the text generated, so text generation will be omitted until the end of the tests, where one final model will be used.

### Optimizer Tests

First, optimizers will be tested. The optimizers that will be compared are RMSprop, Adam, SGD, and Adagrad, all of which are well-known optimizers that are used for gradient descent. SGD calculates the error and updates the model for each epoch. Adagrad is an improvement of SGD with similar functions, and Adam is a combination of both.

Text, letter

Description automatically generated

Fig 3.12.3: Optimizer Test.

Two lists, accuracy\_measures and loss\_measures will be defined. These lists contain the models’ validation accuracy and loss values so they can be plotted altogether later. The optimizer list is defined with the name of all the optimizers, where the model will be trained with each of those optimizers.

Chart, line chart

Description automatically generated

Fig 3.12.4: Optimizer Test Result.

The models are then plotted after 20 epochs. The model using RMSprop was able to give the highest accuracy, so this model will be used from this point onwards.

### Normal Dropout Test

For this test, different dropout values will be tested. The values will range from 0.1 to 0.5. There will only be one dropout layer just before the final dense layer.

Text

Description automatically generated

Fig 3.12.5: Dropout Test.

Chart, line chart, histogram

Description automatically generated

Fig 3.12.6: Dropout Test Results.

The model with a dropout value of 0.2 gives a much higher accuracy than the other dropout values, so it will be used.

### Weight Regularization Test

For weight regularization, it will only be present in the last LSTM layer. The regularizers tested will be L1, L2, and L1 & L2 combined, using their default values of 0.01.

Text

Description automatically generated

Fig 3.12.7: Weight Regularization Test.

A model with no regularization will also be compared with the other models.

Chart, line chart

Description automatically generated

Fig 3.12.8: Weight Regularization Test Results.

The model without regularization provides a higher accuracy than the models with regularization if stopped early, so it will be used.

### Batch Size and Validation Split Test

For this test, the **epochs will be reduced to 10**, as it was shown in previous models that the accuracy of the models will tend to drop after 10 epochs, so this is done to stop overfitting or the vanishing gradient problem early.

Text, letter

Description automatically generated

Fig 3.12.9: Batch Size and Validation Split Test.

Batch sizes range from 64, 128, 256, and 512. The validation split values range from 0.2, 0.3, and 0.4.

Chart, line chart

Description automatically generated

Fig 3.12.10: Batch Size and Validation Split Test Results.

Looking at the graphs, the orange line seemed to have the highest accuracy of around 0.58. However, another line with the same color performed significantly worse. The model could either be the one with batch size 64 and validation split of 0.3, or a batch size of 512 and validation split of 0.4. To determine the model, the epoch training process was observed to identify which models they were.

Table

Description automatically generated

Fig 3.12.11: Batch Size 512 and Validation Split 0.4 Results.

We can see that the model with batch size of 512 and a validation split of 0.4 had a high peak accuracy of 0.5837, so this model is identified to be the best performing model and will be used in later models.

### Recurrent Dropout Tests

For this test, recurrent dropout will be tested. Recurrent dropout is a form of regularization, where instead of being an external layer like normal dropout, recurrent dropout is present inside the LSTM layer, where it randomly drops one of the recurrent inputs that the LSTM layer sends back to itself.



Fig 3.12.12: Recurrent Dropout in LSTM Layer.

This shows how recurrent dropout is used inside the LSTM layer.

Text, letter

Description automatically generated

Fig 3.12.13: Recurrent Dropout Test.

For models using recurrent dropout, normal dropout will not be present.

Chart

Description automatically generated

Fig 3.12.14: Recurrent Dropout Test Results.

The model with a recurrent dropout of 0.2 will be used as it has the highest accuracy.

#### Recurrent Dropout vs Both Dropouts

This test will compare the model with only the recurrent dropout against a model with both normal and recurrent dropout, to see if which one would have higher accuracy.

Text

Description automatically generated

Fig 3.12.15: Recurrent Dropout vs Both Dropouts Test.

Chart, line chart

Description automatically generated

Fig 3.12.16: Recurrent Dropout vs Both Dropouts Test Results.

The model with both dropouts not only had higher accuracy than the model with only recurrent dropout, but also did not overfit over the 10 epochs. The model can still be trained further and get even higher accuracy, so it will be used as the main model from now on.

### Model 12 - Current State of the Model

This is the current state of the model after going through several hyperparameter tests.

Text

Description automatically generated

Text

Description automatically generated with low confidence

Text

Description automatically generated

Fig 3.12.17: Model 12 Configuration.

Chart, scatter chart

Description automatically generated

Text, letter

Description automatically generated

Fig 3.12.18: Model 12 Results.

The text generated has improvement from before the hyperparameter tests, where there is now slightly more grammatical sense. Complete words and somewhat understandable sentences are formed.

## Model 13 – Bidirectional LSTM Layers

For this model, bidirectional LSTM layers will be used. Bidirectional LSTM layers have the ability to use information from past inputs as well as future inputs. A bidirectional LSTM layer makes the information flow backward, so that the future inputs can be fed along with the past inputs in the layer. This is especially useful for generating sequential text characters, so it will be tested here.

Graphical user interface, text

Description automatically generated

Fig 3.13.1: Model 13 Configuration.

The second LSTM layer is made as the bidirectional layer.

Text

Description automatically generated with medium confidence

Fig 3.13.2: Model 13 Parameters.

The model will have a roughly doubled number of parameters due to the bidirectional layer. As seen in earlier models, having too many parameters can result in a worse model. Hence, the number of parameters will be reduced.

Text

Description automatically generated

Fig 3.13.3: Model 13 New Configuration.

The number of nodes for both layers will be decreased to 128 and 256 respectively.

Table

Description automatically generated

Fig 3.13.4: Model 13 New Parameters.

The number of parameters has decreased to around 910,000, which is closer to the original model with 1 million parameters without bidirectional layers. The rest of the hyperparameters remain unchanged.

Chart

Description automatically generated

Fig 3.13.5: Model 13 Results.

The model has a very similar accuracy, loss, and quality of text generated compared to the model without bidirectional LSTM. These two models will need closer comparison in order to determine which is the better model.

## Models 14 & 15 – With and Without Bidirectional LSTM

The models will be directly plotted against each other for a better observation. It was observed that for both models, they were close to but still not overfitting after 10 epochs. Hence, these models will be trained for 15 epochs, to ensure they are both fully trained without overfitting.

Text

Description automatically generated

Text

Description automatically generated with medium confidence

Fig 3.14.1: Model 14 Configuration.

Model 14 - Bidirectional model’s network as shown above.

Text

Description automatically generated

Table

Description automatically generated

Fig 3.14.2: Model 15 Configuration.

Model 15 - Non-bidirectional model’s network as shown above.

Chart, line chart

Description automatically generated

Fig 3.14.3: Model 14 vs. Model 15 Results.

The model without bidirectional LSTM seems to be the clear winner, with higher validation accuracy and lower validation loss as well. Both models are barely starting to overfit at 12 epochs, so this is a good stopping point for these models.

Text

Description automatically generated

Fig 3.14.4: Model 14 Text Generated.

Text

Description automatically generated

Fig 3.14.5: Model 15 Text Generated.

The model without bidirectional layers seemed to have just slightly better text generation as well, as there looks like less repetition of words in the text generation. Other than that, it is hard to differentiate how much better the model performed through the text generation.

Overall, the model without bidirectional layers, Model 15, seems to be the best model after various test cases and changes, so it will be used as the final model to generate texts.

# Use the developed Model to Generate Texts

The model that will be used to generate texts will be Model 15. This model will generate text based off a custom user input with a maximum length of 60 characters.

Text

Description automatically generated with medium confidence

Fig 4.1: User Input Field.

A variable called ‘text\_input’ will be created with the input function, which allows users to add in a custom input. The maximum character length of the text input will be 60, which also corresponds to the maximum character length of the extracted character sequences used to process the data. If an input less than 60 characters is used, the text input will be padded from the back with empty spaces using the ‘rjust’ function.

Graphical user interface, application, email, website

Description automatically generated

Fig 4.2: User Input Padded to 60 Characters.

The above is an example of a padded text input, where the empty spaces are added at the back in order to make the input length exactly 60 characters.

Text

Description automatically generated

Fig 4.3: Load Model and Generate Text Characters.

The chosen model is then loaded so that it can be used to generate the text characters. After that, there is code to generate text characters, which is the same code used to generate text for the previous models, but with some modifications.

Firstly, there is a function that defines which character would be chosen as the next character in the sequence when the model is generating text. In this function, exponential functions are calculated along with the softmax temperature values to produce the value of the next character generated.

Next, the code to generate a random seed of text for the model is commented out, as it is going to be using the user inputted text instead. The code after that allows the model to generate 400 text characters based on the user input and different softmax temperature values. The temperature values used here are 0.2, 0.4, 0.5, and 1.0. Temperature values of 0.4 and 1.0 were added to observe how the model would perform with different values, as 0.2 and 0.5 have already been observed before.

The text generated will then be printed out as shown below.

Text

Description automatically generated

Text

Description automatically generated

Fig 4.4: Final Model Text Generated.

The text generated with lower temperature values are more predictable and make more sense, while the text generated with the higher temperature values have more randomness and less real words generated. For this model, the text generated at 0.4 temperature seems to make the most sense out of the 4 temperature values, with the least word repetition and somewhat proper sentences.

# Summary

In total, 15 models were developed, trained, and tested, and only 1 was selected as the best model among them to perform text generation. The models had gone through various changes in terms of their network structure, hyperparameters, and layers to get to this final model. For the results of each model, they were plotted in accuracy and loss graphs, and the text generated for each model was shown as well. The models were compared against each other using these visualizations to determine which model would be better to use.

Different RNN layers such as SimpleRNN, LSTM, and GRU were tested and compared against each other to see which could give better results for the model. In the end, LSTM was chosen as the best RNN layer to use. Then, the number of layers and nodes in the models were changed multiple times to see which would give the best results, and which would give poorer ones. The result was a model with 2 layers, having 192 and 384 nodes respectively.

After that, the hyperparameters were extensively tested with, where models were trained with different hyperparameters such as dropouts and batch sizes, before plotting them against each other with different values to see which values would result in the best performing model. The image below shows the hyperparameters that had been used for the final model.

Text

Description automatically generated



Fig 5.1: Final Model Configuration.

Finally, different configurations to the RNN layers such as Bidirectional LSTM layers were tested out and compared against normal LSTM layers. The result was that the model without the bidirectional layers performed better than the model with bidirectional layers, so it was used instead.

In the end, the model was able to end with just over 0.58 validation accuracy, which was a slight improvement from the initial average of 0.54 accuracy in the second model. When tested on the user input, the text generated at a softmax temperature value of 0.4 had the best outcome, as it had the best sentence structure and the least repetitive words appearing among the other temperatures.

For further improvements for this model’s performance, more data can be used to feed into this model. With more data, the model can learn and retain a lot more information as long as it is scaled up to the data size properly and can achieve an even higher accuracy to produce better text results that make more sense. Related data such as more Harry Potter books can be fed into this model, so that it can specialize in generating text related to Harry Potter for a user input. Other general data such as books or novels from a series unrelated to Harry Potter can be fed into the model, which can help the model produce more generalized and possibly more results related to the user input’s context.

Other improvements can include more techniques being experimented on the models, such as batch normalization for RNNs, or even fine-tuning the model’s layers and nodes to get the perfect number of parameters for the highest accuracy.

With more time, all of the forementioned techniques can be used and tested in the future. But for now, with the model already showing improvement in its results and its text generated, this is a good first step in developing a reliable and functional model to generate text characters from a user input.