

**School of InfoComm Technology**

**Deep Learning Assignment**

Diploma in CSF / FI / IT

April 2022 Semester

**ASSIGNMENT 1**

(30% of DL Module)

16th May 2022 – 10th June 2022

**Submission Deadline:**

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

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

|  |  |  |
| --- | --- | --- |
| **Tutorial Group** | **:** | **P02** |
| **Student Name** | **:** | Lim Long Teck |
| **Student Number** | **:** | S10221824G |

**Penalty for late submission:**

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

**NO** submission will be accepted after 19th June 2022 (Sunday), 11:59PM.

Contents

[1. Overview 3](#_Toc111237885)

[2. Data Loading and Processing 4](#_Toc111237886)

[3. Develop the Sequence Generator Models 7](#_Toc111237887)

[4. Generating Text with developed model 34](#_Toc111237888)

[5. Summary 37](#_Toc111237889)

[6. Appendix A 38](#_Toc111237890)

[7. References 39](#_Toc111237891)

# **Overview**

The problem that I would be addressing for this assignment is to implement Recurrent Neural Network (RNN) to create an English language character generator which is capable of building semi-coherent English sentences from scratch. As such, the objective for this assignment is to build a model from scratch such that it is able to successfully predict the next character in the sentence. The approach I would be taking is the 7 steps in the universal workflow of machine learning. Firstly, defining the problem and assembling a dataset. I have identified the problem to be a multiclass, single-label classification. As for assembling a dataset, I would be using the complete version of J.K. Rowling’s famous book, Harry Potter and the Philosophers Stone. I would further elucidate on how I pre-process the book into data in the subsequent paragraphs. Secondly, choosing a measure of success. The metric of success that I would be using is classification accuracy. Its accuracy is judged by the number of correct predictions to the total number of input samples. Thirdly, deciding on an evaluation protocol. I think it would be sufficient to use just the data that have been provided to us, the Harry Potter book. Next, preparing my data. With the Harry Potter book provided, I am supposed to clean the data first before I can start training my models with it. As you can see in Figure 1.1 below, there are unnecessary symbols found in the text file provided to us. An example would be the highlighted symbol, “•”, and the pipe symbol, “|”, both seen in Figure 1.1. Riding the data off unneeded characters would in turn benefit my model as it gets a higher accuracy. In the following paragraphs, I would be explaining on how I managed to clean my text file, also known as the data, off this inessential symbols. For my fifth step, I would have to define the last-layer activation, loss function and optimization configuration to use. The last-layer activation would be softmax with a loss function of “categorical\_crossentropy” as my problem type is multiclass, single label classification. For my optimizer, I would be adopting RMSprop with learning rate of 0.01. During the training process, I would be experimenting with the learning rate, so changes would definitely be made for the learning rate. Sixth, I would have to train the model until it overfits. This suggests that I should not add any layers or hyperparameters to my model that helps with overfitting as it would be going against the machine learning workflow where I should get my model to overfit first. A very clear example of what should not be added to my first model is Dropout layers. Lastly regularizing and tuning the hyperparameters of my model. This is the step whereby I can add dropouts or even add regularizers to my model in hope that the model does not overfit so early. I can also adjust the learning rate of the optimizer, batch size or even increased the number of units per layer. Of course, if I were to make such changes, I would be documenting it in my report and my reasoning behind it.

There are three built-in RNN layers in Keras that I could utilise for my models, SimpleRNN, Gated Recurrent Units (GRU) and Long Short-Term Memory (LSTM). For this assignment, I would mainly be using GRU and LSTM for building my models. When using, SimpleRNN, GRU and LSTM in my practical, I felt that GRU and LSTM gave the best validation accuracy overall.

The reason behind why SimpleRNN might have a lower accuracy than the other two layers, GRU and LSTM, is because there are no gates present in SimpleRNN. For GRU there is an update and rest gate. The gates decide whether to pass the previous output to the next cell or to be neglected. These gates are certainly crucial in the learning process as there may be some data that are redundant and should not be passed on to the next cell or be totally forgotten. Whereas for LSTM, there are three gates, forget, input and output gate. The forget gate decides which information needs attention and which can be ignored similar to the reset gate in GRU. The input gate performs operations to update the cell status, deeming the cell important or not. The input cell is similar to the update cell in GRU. Lastly, the output gate determines the value of the next hidden state as it contains information on previous inputs. With these reasons, I can see why SimpleRNN presented a lower accuracy compared to the other two layers. As such, I would be focusing more on GRU and LSTM when building my models.

GRU is less complex than LSTM due to the difference in gates. However, GRU is preferred for a smaller dataset. As I am unsure if my dataset is considered large or small, I would be experimenting with both layers when building my model.

Text

Description automatically generated

**Figure 1.1 – Unnecessary symbols in data**

# **Data Loading and Processing**

I loaded the data (Harry Potter text file) into the jupyter notebook by typing in the file path of where the text file is located on my laptop. Using the open() method, I managed to load the data into the jupyter notebook where I can start filtering and process the data. Firstly, I convert all the words into lower case. Converting all the words to lower case would help the output of unique characters to be lower. This would in turn relate to having better accuracy from the models that are trained. Next, I put the symbols that I deem as redundant into a list and named it “extra\_punct” as seen in Figure 2.2. I also made another list named “numbers” with numbers one to nine in the list. As seen below in Figure 2.1, there are texts in the data that has the page number of the book displayed. Obviously, these texts are redundant and as such, I had them removed in part of the cleaning process. Next, I would be using a for loop to remove the symbols in the “extra\_punct” list and page numbers. I created an empty string named “harrypotter” and after iterating each word from the data, I would store it inside “harrypotter”. Inside the for loop, I used if-else statements to remove the redundant data. As most of the symbols are attached to the front of the words, I used the following code, as seen in Figure 2.3, to remove these symbols. The line of code basically suggests that I do not add these words into the “harrypotter” string and hence, it would not be used in my training data. The last thing that I removed were the book title. As you can see in Figure 2.1, after the page number, the book title and the author of the book is displayed. As such, together with the page number, as you can see in Figure 2.4, I hardcoded the for loop to remove these redundant texts that may hinder the model’s training process.

After successfully cleaning the data, I then proceeded to process the data. Before I move onto the one-hot encoding process, I would like to present the number of unique characters the harrypotter text have. Before data cleaning was performed, there were a total of 84 unique characters as seen in Figure 2.5. After getting rid of unnecessary characters from my data, I was left with a total of 44 unique characters in my data as seen in Figure 2.6. Now, moving on to the one-hot encoding process, we would encode the characters into binary arrays for both sentences and unique characters. The data type of these arrays would be Boolean and as such, when the number is zero, the value is False. Subsequently, when the number is 1, the value is True.

Referring to Figure 2.7, you can see two variables, “x” and “y”. “x” is the total number of sentences the data has after splitting it up to 60 characters each. Whereas “y” is the total number of unique characters. All the characters are converted into zeros, which also means False. For a quick visualisation, an element of “x” would be the list of “y” repeating itself for 60 times. Moving on with the one-hot encoding process, we would enumerate each sentence after obtaining the variables “x” and “y”. When the character in the sentence matches the element in the unique character list, “y”, we would give the element in the list the value one, thus changing the value to True. The corresponding element in “y” would also be changed to one (True).

This is basically the gist of my data loading and processing process. Next, I would start building my model.

Text

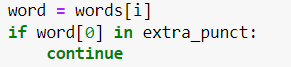
Description automatically generated

**Figure 2.1 – Text displaying page number and book name**

**Chart, scatter chart

Description automatically generated**

**Figure 2.2 – “extra\_punct” and “numbers” lists**

****

**Figure 2.3 – “extra\_punct” and “numbers” lists**

**Text

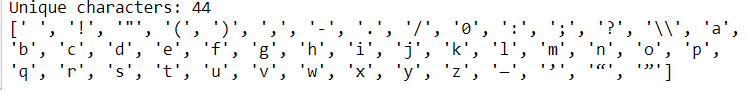
Description automatically generated**

**Figure 2.4 – Hardcoded code to remove redundant texts**

A picture containing text

Description automatically generated

**Figure 2.5 – Number of unique characters before data cleaning**



**Figure 2.6 – Number of unique characters after data cleaning**

**Text

Description automatically generated**

**Figure 2.7 – One-hot encoding process**

# **Develop the Sequence Generator Models**

To recap, I have been tasked to build at least one English language character generator model that is capable of building semi-coherent English sentences that is implemented with RNN. RNN is a type of artificial neural network which uses sequential data or time series data. In the context of the assignment, I would be using sequential data.

For my first model, my main aim was to get the model to overfit following the machine learning workflow. As such, no dropouts and regularizers were added. The length of extracted character sequence is 60 and number of steps taken is 3. I added three GRU layers of 512, 512 and 32 units followed by a dense layer of 44 units with an activation function of softmax as the problem type is multiclass, single-label classification. As mentioned earlier in the overview, I have opted to use RMSprop for my optimizer with a learning rate of 0.01 for the first model. The loss function would be “categorical\_crossentropy” following the problem type of multiclass, single-label classification. I ran the model for 50 epochs with a batch size of 128.

Graphical user interface, chart

Description automatically generated

**Figure 3.1 – First Model**

From Figure 3.1, you can see that the highest validation accuracy the model achieved is 36%. The validation loss of the model started to get noisier around the 30th epoch. Looking at the validation loss of the model, it seems that the model is starts to overfit at around the 40th epoch where the overall loss of the model starts to increase. With a peak validation accuracy of 36%, I expected the predicated text to be gibberish.

**Text

Description automatically generated**

**Figure 3.2 – First model text generation**

Low and behold, the text generation from the first model was atrocious. The text generated are not words at all, as it does not make any sense. I felt that a learning rate of 0.01 might be the reason for the model’s low accuracy as it makes large changes to it weights. I would also increase the batch size so that the model can generalise better, making better predictions.

For my second model, I had the same parameters and layers as my first model. The changes I made were to decrease the learning rate of the model to 0.001 and increase the batch size to 150.

**Chart, line chart

Description automatically generated**

**Figure 3.3 – Second Model**

From Figure 3.3, you can see that my model performed better after making changes to its learning rate and batch size. The highest validation accuracy of the model is 53%. The model overfitted early at around the 10th epoch.

Text

Description automatically generated

**Figure 3.4 – Second model text generation**

Although the model might have a better accuracy, it does not mean that generated text from the model might be better. To my surprise, looking at Figure 3.4, you can see that most of the words generated from the model are actual English words. Although if you read the entire text generated, you would find that the sentence does not make any sense at all. Knowing that altering the learning rate and batch size can affect my model’s accuracy, I decided to change the batch size again for my third model.

Having kept the number of units, layers and learning rate of my model the same, I increased the batch size again to 175. Hoping that my model would get a higher accuracy, I ran the model again for another 50 epochs.

Chart, line chart

Description automatically generated

**Figure 3.5 – Third Model**

As seen in Figure 3.5, the highest accuracy for the third model is 54%, an increase from the previous model. As I did not include any dropouts or regularizers to the model, it is expected that the model overfits early. The third model overfits at around the 8th epoch. The words generated by the model were the same as model 2 with some words not being recognisable and an overall text which is nonsensical.

Text, letter

Description automatically generated

**Figure 3.6 – Third model text generation**

Ignoring the grammatical issues in the texts generated, the words generated seems acceptable as seen in Figure 3.6. However, as the temperature increases, spelling errors become more prevalent.

Earlier, I had mentioned that I would like to experiment with LSTM layers as well as I do not know which layer would be more effective. As such, for my fourth model, I would be swapping the GRU layers with LSTM layers while keeping the rest of my hyperparameters the same. This way, I can then see the difference between LSTM and GRU.

Graphical user interface, chart

Description automatically generated

**Figure 3.7 – Fourth Model**

From Figure 3.7, it seems that the highest accuracy the model attained was 52%, performing worse than the third model with GRU layers. The model overfitted at around the 10th epoch.

Text, letter

Description automatically generated

**Figure 3.8 – Fourth model text generation**

Although there is a higher validation accuracy for the third model with GRU layers, looking at the generated text, there are no noticeable changes. There are still a lot of spelling and grammatical errors.

Having researched on what else I could do to add to the layers to increase the validation accuracy of the model, I came across Bidirectional layers. The bidirectional layers connect two hidden layers of opposite directions to the same output. As such, the output layer can get information from the previous and future states simultaneously. As such, for my fifth model, I would be adding bidirectional to my LSTM layers. Logically, my model should have a better validation accuracy compared to the fourth model as I would not be changing any hyperparameters. For a more detailed explanation on Bidirectional, please refer to [Appendix A](#_Appendix_A).

Graphical user interface

Description automatically generated

**Figure 3.8 – Fifth Model**

From Figure 3.8, the highest validation accuracy the model attained was 53% and the model overfitted at around the 10th epoch as well. It would be safe to assume that with the addition of Bidirectional, the model had a better accuracy, thus performing better.

Text

Description automatically generated

**Figure 3.10 – Fifth model text generation**

From Figure 3.10, it seems that all the generated text with the different temperature all generated the same text after the seed, “said dumbledore,”. The difference in generation of text with different temperature can be seen after the text “said dumbledore,” as different variation of the word, “what”, is generated.

For my next model, I would be switching out the LSTM layers for GRU layers. As such, the model would be built on bidirectional GRU layers. As I am comparing if bidirectional GRU performs better or not, I would be keeping the hyperparameters the same for this model as well.

Graphical user interface, chart

Description automatically generated

**Figure 3.11 – Sixth Model**

As seen in Figure 3.11, the model overfitted at around the 10th epoch as the model’s overall validation loss starts is increasing. The model’s peak accuracy was 54%, suggesting that GRU layers performs better with my current dataset compared to LSTM layers. As such, I would be using bidirectional GRU as my layers.

Text, letter

Description automatically generated

**Figure 3.12 – Sixth model text generation**

There are not a lot of noticeable changes, but the most obvious error would be the text generated with a temperature of 0.2. Nearing the end of the generated code, it seems that the model kept repeating the word "if". More punctuations are also used in the text generated with a temperature of 0.9.

As my model overfits early in the training, for the next model, I decided to add dropout layers between the bidirectional GRU layers. I added two Dropout layers with value of 0.5 hoping that the model would not overfit so early.

Graphical user interface, chart, histogram

Description automatically generated

**Figure 3.13 – Seventh Model**

As seen in Figure 3.13, the model attained a peak validation accuracy of 55%, performing the best amongst the previous model. The model overfitted later at around the 20th epoch. The validation loss does not show a sudden spike as compared to the previous model’s validation loss.

Text

Description automatically generated

**Figure 3.14 – Seventh model text generation**

From Figure 3.14, although there are still grammatical mistakes in the text generated there seems to be lesser misspelt words. It seems that the increase in accuracy resulted in more words that are spelt correctly.

For the next model, I decided to increase the length if extracted character sequence (maxlen) and the steps taken in the data processing. I increased the maxlen to 60 and the steps taken to 5. Keeping the layers, learning rate and batch size the same, I trained my model for 100 epochs.

Graphical user interface, chart, histogram

Description automatically generated

**Figure 3.15 – Eighth Model**

As seen in Figure 3.15, the model performed worse as compared to the previous models. The model attained the highest accuracy of 51%. The model overfits at around the same epoch as the seventh model, at the 20th epoch. Seeing that my model accuracy worsens with the changes made in the data processing process, I decided not to increase the maxlen and steps anymore. Instead, I decreased the steps taken. As such, in the data processing, the maxlen would be reverted to 60 while the value for steps is 2 for the next model.

Text

Description automatically generated **Figure 3.16 – Eighth model generated text**

The text generated with a temperature of 0.5 seems to be the best as it has the least spelling mistakes, around 5 errors found. Grammatical errors are still highly prevalent in the texts generated.

For my next model, after adjusting the data processing values accordingly, I have also adjusted the learning rate and batch size for the new model. I decreased the learning rate to 0.0002 and increased the batch size to 200. I changed the values of the learning rate and batch size as it seemed to help increase my model’s accuracy earlier as seen from my first to third model.

Graphical user interface, chart, line chart

Description automatically generated

**Figure 3.17 – Ninth Model**

From Figure 3.17, the highest accuracy the model achieved was 56.2% and the model overfitted later at around the 30th epoch. The model might have achieved a higher accuracy due to the change in the data processing. As the data is changed such that new sequences are sampled after a smaller number of steps, it helped the model to make more accurate predictions and thus, better accuracy.

Text, letter

Description automatically generated

**Figure 3.18 – Ninth model generated text**

From Figure 3.18, checking the spelling of the words generated, it seems that there were hardly any spelling mistakes. As this model had the best validation accuracy, it seems that the text generated by the model was also the best.

For the next two models, I would be tuning the learning rate to find the best learning rate for my model. The values of the learning rate that I would be testing is 0. 00065 and 0.00055.

Graphical user interface, chart

Description automatically generated

**Figure 3.19 – Tenth Model (learning rate = 0.00065)**

**Chart, line chart

Description automatically generated**

**Figure 3.20 – Eleventh Model (learning rate = 0.00055)**

As seen in Figure 3.19, the tenth model overfitted at around the 30th epoch, while the eleventh model as seen in Figure 3.20 overfitted at around the 20th epoch. There is just a minute difference between the two models’ accuracy. The highest accuracy of the eleventh model is a little better at 56.8%, while the accuracy of the tenth model is 56.6%.

Text, letter

Description automatically generated

**Figure 3.21 – Tenth model generated text**

**Text

Description automatically generated**

**Figure 3.22 – Eleventh model generated text**

As seen in Figure 3.21 and 3.22, from what I can see, there are no differences in the text generated. As such, looking at the validation accuracy of the model, the eleventh model with a learning rate of 0.00055 was better.

As the model with a learning rate of 0.00055 performed better I would be using that as my base model now. My aim now is to cry and find the optimal range for my model. The current model now has 3 bidirectional GRU layers with two 512 units and one 32 units, 2 dropout layers with a value of 0.5 and a final dense layer with 44 units with an activation function of softmax. As seen from Figure 3.20, the eleventh model overfitted very early, as such I would be adding one more dropout layer with value of 0.5 to the model and L2 recurrent regularizers to both the 512 units layers. The value for the recurrent regularizers are 0.001 and 0.0001. The model has a learning rate of 0.00055 and would be trained for a total of 100 epochs with a batch size of 200.

Graphical user interface

Description automatically generated

**Figure 3.23 – Twelfth Model**

As seen from Figure 3.23, the model started to overfit at around the 50th epoch. The model attained a peak accuracy of 56.9%.

Text, letter

Description automatically generated

**Figure 3.24 – Twelfth model generated text**

From Figure 3.24, noticeable differences between the different temperatures is the word after “didn’t”. Only the text with temperature of 0.7 and 0.9 generated different words after the word “didn’t”. Other than that, it seems that the generated text has more spelling errors as compared to the ninth model.

For the next model, I am planning to adjust the regularizers value as to make the model overfit even later. The regularizers value that I would be using are 0.004 and 0.0003. Due to time constraints, I increased the batch size value to 256 as well as each epoch took a very long time to run. Hopefully, increasing the batch size would allow my model to trained finish at a faster rate.

Graphical user interface

Description automatically generated

**Figure 3.25 – Thirteenth Model**

Sadly, as seen in Figure 3.15, my model is underfitted with a peak accuracy of 56.9%. On the bright side, each epoch took about 10 seconds quicker to be completed.

Text

Description automatically generated

**Figure 3.26 – Thirteenth model generated text**

From Figure 3.26, the number of spelling mistakes seems to be lesser than the generated text from the previous model. However, the grammatical errors are still present.

Unfortunately, I did not have enough time to find the right regularizer value to implement into my model so that it would be overfitted at a later epoch. As such, I would be using the twelfth model as seen in Figure 3.23 and train it until it reaches the optimal range which is the 50th epoch.

Graphical user interface

Description automatically generated

**Figure 3.27 – Final Model**

Figure 3.16 displays the training and validation accuracy and loss of my final model. It is trained until its optimal range, and it has an accuracy of 56.7%.

Text, letter

Description automatically generated

**Figure 3.28 – Final model generated text**

Ignoring the grammatical errors, it seems that texts generated by the model are decent. Although there are some spelling mistakes still present. The temperature that generates the most spelling errors is obviously 1.0.

Comparing all the texts generated by the different models, it seems that grammatical errors is always prevalent in the texts. The spelling mistakes seems to go together with the accuracy of the model. As such, higher accuracy results in lesser spelling errors. However, this comparison might be flawed as the seeded text are different.

# **Generating Text with developed model**

Now I would be applying real-life text inputs into my model. I would then analyse the generated texts and comment my feedbacks on it. The first text that I inputted was: “I went oversas to buy multiple items. I did my chores before going overseas.”. As my model is trained with a maxlen of 60, I would have to cut the inputted text until the 60th character before it can generate new texts. Moreover, before inputting the text into the model, the texts would have to be changed to its lower-case form. As such, the text that is inputted into the model is “i went overseas to buy multiple items. i did my chores before”.

Text, letter

Description automatically generated

**Figure 4.1 – Generated Text by Developed Model**

As seen in Figure 4.1, I used softmax temperature to generate 400 characters. The temperature that I have used are 0.2, 0.3, 0.5, 0.7, 0.9 and 1.0. All the texts generated from the different temperature managed to complete the word “befor” to “before”. Since this model is a character generator, the completion of the word “befor” to “before” is a given.

For the text generated with 0.2 temperature, the texts do not make any sense. However, it seems that the word, “them”, is very commonly generated. However, most of the words spelt are correct.

For the text generated with 0.3 temperature, it seems that the text generated uses more punctuations as compared to the texts generated using 0.2 temperature. Although more punctuations are used, mostly full stops, the texts hardly make any sense. Although most of the words are spelt correctly, the texts are still not grammatically correct.

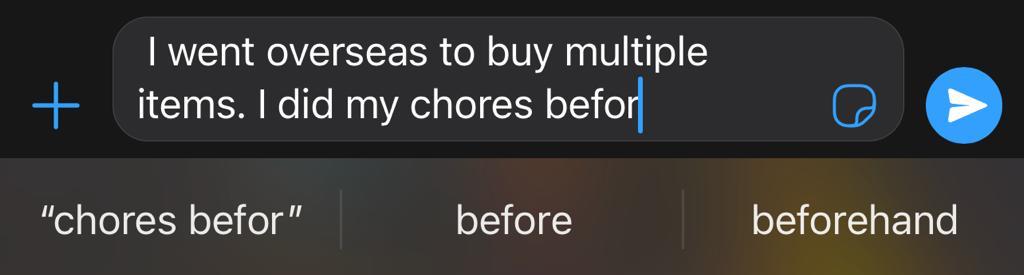
For text generated with 0.5 temperature, it seems that the randomness of the characters generated starts to get more obvious. There are a few words that seems to be misspelt with “dingerous” being the most obvious. It seems that the word the model intended to display is “dangerous” as dingerous is not a normal English word.

For text generated with 0.7 temperature, the model generated more unique punctuations such as the apostrophe (‘) and the quotation mark (“). The words with the apostrophe are actual words such as “can’t” and “couldn’t”. However, the placements of the quotation mark are incorrect. Unfortunately, the overall structure of the text is incorrect and nonsensical as well.

For the text generated with 0.9 temperature, the characters generated are very random. Reading the text generated, there are multiple incorrect words with grammatical errors within the sentence. Although there is a usage of the apostrophe, it was used incorrectly, “an’”.

Lastly, for the text generated with 1.0 temperature. Since the texts generated with 0.9 temperature is already so random, the texts generated with 1.0 temperature is expected to be even more random and nonsensical. As expected, the text generated as seen in Figure 4.1 is nonsensical and there are more misspelt words.

Overall, the texts generated did not make any sense. As the temperature increases, the generated texts start to use more punctuation and more spelling errors starts to occur.



**Figure 4.2 – Predictions of words on the phone**

As seen in Figure 4.2, it seems like the word “befor” is also accurately predicted to be “before” by the phone if the predicative settings in enabled. Now I would be continuously pressing the predicted next words to see what kind of texts get generated.

Text

Description automatically generated

**Figure 4.3 – Generation of text on the phone**

From Figure 4.3, you can see the texts that have been created by continuously pressing the centre word on the keyboard as seen in figure 4.2. The phone also stores words that have been used by the user. For example, the word “lol”, although it is not an English word, it is predicted to be used.

Although this is not a character-by-character text generator, I would like to use this as an example to what my model has generated. It seems that there is a larger variety of the word generated by my model as compared to the phone. Both texts generated by my model and the phone seems are not able to craft grammatically correct sentences.

Although the phone uses a different type of model to do the prediction, I am relieved to know that the text generated by the phone are not grammatically correct as well.

# **Summary**

I felt that my model could have been regularized better by finding the right value so that the optimal range of my model would be at a later epoch. Although the texts generated might be spelt correctly, it seems that throughout the training process, the model was not able to generate a grammatically correct sentence.

In the future, I hope to focus more on the grammar of the texts generated. To do that, more data must be provided. I can pass data that focuses more on the grammar of the sentence of the text for the model to train. Looking at how my model constantly overfits at the early stages of training, I should have used keras early stopping function. I can set the model to monitor its own validation accuracy, and when the validation accuracy constantly decreases for 20 epochs, the model would stop fitting. This way I could have spent more time regularizing my model towards the end of my building process as I need not wait for the model to train finish before I start on my next model. These are the suggestions that I have for my further improvement.

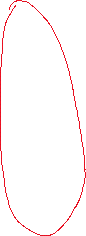
Looking back to my assignment 1, I managed to touch on one of my suggestions for further improvements. That suggestions were to play around more with the model’s hyperparameters. In this assignment, I managed to play around with more hyperparameters and not just the regularizers. This definitely made the building process more enjoyable for me.

All in all, I had a blast building a character-by-character text generator. It was something that I would not have imagined myself building such models, as such this assignment was an entertaining yet tiring one.

# **Appendix A**

Diagram

Description automatically generated



**Figure 6.1 – Bidirectional architecture**

Figure 6.1 shows that overall bidirectional layer architecture. The top layer is the forward propagation while the bottom layer describes the backward propagation process. The input that the layers are trying to learn is “dhaval loves apple it keeps him healthy”.

So, focusing on the cells which is circled, y4, which is an output, us a function of a0, a1, a2 and a3. However, that is only for the forward propagation process. Without Bidirectional, this is what a normal LSTM or GRU layer does. Now, with the addition of Bidirectional, y4 has the influence of the future words as well. Originally, the output y4 is only made up of the word “dhaval loves apple”, however with bidirectional y4 is also made up of the future words, which is “keeps him healthy” (codebasics, 2021).

Looking at this, with the addition of bidirectional, my model should be able to have a greater accuracy.

# **References**

Codebasics. (2021, Feb 28). *Bidirectional RNN | Deep Learning Tutorial 38 (Tensorflow, Keras & Python)* [Video]. YouTube. <https://www.youtube.com/watch?v=atYPhweJ7ao>