****

**School of InfoComm Technology**

Deep Learning Assignment

Diploma in DS / IT Oct 2022 Semester

**ASSIGNMENT 2**

(40% of DL Module)

**Submission Deadline:**

**Presentation: 12th Feb 2023 11:59PM**

**Report: 12th Feb 2023 11:59PM**

|  |  |  |
| --- | --- | --- |
| **Tutorial Group** | **:** | **P01 / P02** |
| **Student Names (Student Number)** | **:** | **Chong Xin Le** |

##### Penalty for late submission:

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

**NO** submission will be accepted after 19 Feb 2023, 11:59PM

Table of Contents

[1. Overview 3](#_Toc127118586)

[2.1 Data Loading 4](#_Toc127118587)

[2.2 Data Processing 4](#_Toc127118588)

[3. Develop the Sequence Generator Model 8](#_Toc127118589)

[4. Use the developed Model to Generate Texts 24](#_Toc127118590)

[4.1 Analyze generated text 24](#_Toc127118591)

[4.2 Real- life applications 25](#_Toc127118592)

[5. Summary 25](#_Toc127118593)

[5.1 Model performance summary 25](#_Toc127118594)

[5.2 Improvements 25](#_Toc127118595)

[6. Reflection 27](#_Toc127118596)

# Overview

The problem of text generation in deep learning arises from the need to automatically generate coherent and meaningful text based on given input data. With the increasing amount of text data generated every day, manual generation of text is becoming impractical and time-consuming. Text generation using deep learning algorithms provides a solution to this problem by automating the process of generating text, saving time and effort. Additionally, the ability to generate text based on input data enables various applications, such as language translation, speech recognition, and content creation, to name a few.

In this paper, a Recurrent Neural Network (RNN) is generated based on a English language character generator. The model will be trained on a complete version of J. K. Rowling's book "Harry Potter and the Philosophers Stone" to generate semi-coherent English sentences character-by-character. The input to the model will be a string of valid characters, and the output will be the next character in the sentence. The model will be trained on multiple such examples to make predictions and generate coherent sentences. The goal is to automate the process of text generation and produce semi-coherent English sentences from scratch.

The Data is first loaded and pre-processed. This involves removing any unnecessary characters, identifying a list of unique characters and punctuations, and preparing the data into training text and labels (X and y) using the ‘sliding window’ method. One-hot encoding is then performed on X and y and they are converted into binary arrays.

Using knowledge learnt during lessons, multiple models using Recurrent Neural Network (RNN) layers such as LSTM or GRU are developed. The model is trained using a baseline model and is improved by increasing its complexity until it overfits. Regularizations techniques such as dropouts, L1 regularization, optimizers and batch size are tuned till it reduces overfitting. The model performance is analysed during the training phase using loss scores and the quality of the generated texts as metrics. The mode performance is further analysed by comparing generated texts to its reference text, which in this case is the “Harry Potter and the Philosophers Stone’ book which was sourced from Kaggle. The BLEU score measures the degree of similarity between the generated texts and the reference text and calculates the precision of the generated text with respect to the number of overlapping n-grams between the two.

In the final step, the developed model is used to generate new texts by recording new input from the user, encoding the input, feeding it into the model, and letting the model generate 400 characters. The generated texts are then analysed to determine if they make sense.

Overall, this project aims to automate the process of text generation using deep learning algorithms. The Harry Potter Book 2 text is used to train a sequence generator model that can generate coherent and meaningful texts.

1. **Data Loading and Processing**

## Data Loading

Text

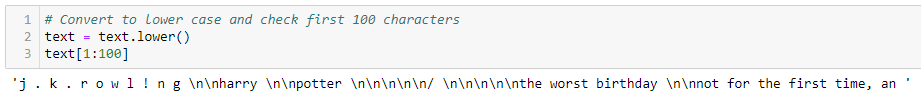
Description automatically generated with low confidence

The code above reads the content of the text file that we are working on. It is opened using the built-in “open” function in Python, and the “r” argument specifies that the file should be opened in “read” mode. The “encoding” argument is set to “UTF-8”, which specifies the encoding format of the text file.

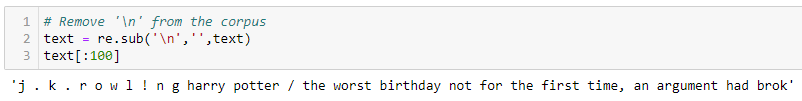
Once the file is opened, the contents of the file are read into a variable called “text” using the “read” method of the file object. The length of the “text” variable is then calculated using the “len” function and printed to the console. From the output we can see that the total number of characters before cleaning is 531,708.

## Data Processing

When doing text generation, it is always useful to convert the text to lower casing. This is because it ensures that all the characters are in a consistent case, making it easier to process the text and make predictions based on the data. In many NLP tasks, the size of the vocabulary can greatly affect the efficiency and accuracy of the model. By converting the text to lower casing, the size of the vocabulary is reduced, as all the characters are in the same case, and this can help simplify the model and reduce the number of parameters that need to be learned. Furthermore, it can improve the generalization ability of the model, as it reduces the dependence on capitalization, which can be language-specific and varies greatly between different texts.



After running, the code for the first 100 characters is displayed. However, the text appears to have multiple “\n” which could have appeared after reading the file, as they were represented as line breaks. These values will be removed since they are irrelevant. The code blow shows that “\n” is being replaced with an empty string.



Often books contain footers at the bottom of the page such as page numbers, chapter titles and other details. This information is not useful to our model and may not bring about a positive impact to the model. This is because footers are usually repetitive and are not relevant to the main context of the text. Removing this can reduce the noise in the data and improve the performance of the text generation model.

Graphical user interface, text, application, email

Description automatically generated

From the first output, we can see that “page | 2 harry potter and the chamber of secrets - j.k. rowling” is the footer of this harry potter book and will appear for every page. The subsequent code is used to remove all occurrences of this specified string pattern from the “text” variable and replace it with an empty string. Lastly, when printing out the same section of the book, the footer is no longer there.

A picture containing application

Description automatically generated

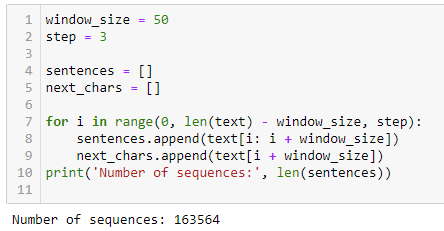
The code above creates a list f all unique characters in the “text” variable and uses the “sort” function to sort the list in alphabetical order. A dictionary is then created to map unique characters to their indices. Lastly, the number of unique characters in the “chars” list is printed out with the list of unique characters. The current clean data has 59 unique characters.

Making use of word cloud, a visualisation displaying the most appeared words from largest to smallest font is created. Here, words such as ron, Harry, said, back, know seems to be the most common words.

These words that appear the most frequent can be an indication of its importance in the text, as words that appear more often are likely to be more central to the text’s meaning and context. In addition, it serves as a starting point for generating new text. The model can use the most frequent words to initiate a sequence of generated text and then continue to generate subsequent words based on the probability distribution of words that typically follow the most frequent word in the text.

Text

Description automatically generated



The code above is used to divide the text into sequcnes, each of which has a length of ‘window size’ and the step between the consecutive sequences is ‘step’

The line window size= 50 sets the size of the text windows that will be extracted from the text corpus, and the line step= 3 sets the step size, which is the number of characters to skip before extracting the next window.

Two lists are created: ‘sentences’ and ’next\_chars’. ‘Sentences’ is a list of substrings of length window size taken from the text, and ‘next\_chars’ is a list of characters immediately following each substring in the ‘sentences’ list.

Lastly, the number of extracted sequences is printed out. The data contains 163,564 sequences.

Text

Description automatically generated

Text

Description automatically generated

The code above is used to prepare the extracted text windows and next characters for use as training examples.

x = np.zeros((len(sentences), window\_size, len(chars)), dtype=bool) creates a 3-dimensional array to store the input data, where each training example is represented as a "window size" x "len(chars)" matrix. The line y = np.zeros((len(sentences), len(chars)), dtype=np.bool\_) creates a 2-dimensional array to store the output data, where each training example is represented as a "len(chars)" x 1 vector.

The outer "for" loop iterates over the "sentences" list, and the inner "for" loop iterates over each character in a sentence. For each character, the code sets the corresponding entry in the "x" array to 1, indicating that the character is present in the sentence. The code also sets the corresponding entry in the "y" array to 1, indicating that the next character is the target character for the training example.

The length of the arrays the first entry of each array is printed out. This is useful for checking that the arrays have been created and populated correctly.

As BLEU scores are used as one of the metric for evaluating the model’s performance, I sourced another harry potter book to use as a reference text. I managed to find the “Harry Potter and the Philosophers Stone” text file for the book through the Kaggle’s website (<https://www.kaggle.com/datasets/balabaskar/harry-potter-books-corpora-part-1-7>)

A picture containing text

Description automatically generated

Similar to how the harry potter book 2 file was open and preprocess, this file has a total of 474,429 characters instead. The code below creates a list ‘reference texts’ and adds the contents of the variable ‘bleu’ to it.

Text

Description automatically generated

# Develop the Sequence Generator Model

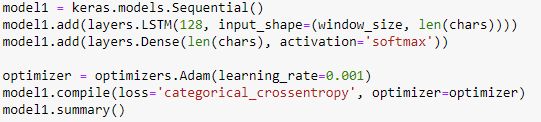
For all my models, I first evaluated the model using BLEU score and loss scores (The higher the BLEU score the better, the lower the loss score the better). This is where I take the generated text with the highest BLEU score and another text with the lowest loss score. However, some models will only have 1 best model due to that model having the highest BLEU score and the lowest loss score. After retrieving the two best texts generated, I compare the models based on my own evaluation as machine evaluated models can sometimes be biased.

For every model, I will show some part of the generated text which is the most useful for explanation since it is too time consuming to read through the texts. For others, I might just make comparison against the loss chart and BLEU score only

For all the models, the text is only generated after every five epochs. This is because generating text at every epoch can be time consuming to run and read through. This by putting the print statement through a loop such that it only prints the generated text every fifth epoch.



**LSTM**



Chart, line chart

Description automatically generated

|  |  |  |
| --- | --- | --- |
|  | Highest BLEU score: | Lowest loss score: |
| Loss score | 1.7778 | 1.5985 |
| Bleu score | 1.5617329148280536e | 1.4488496539373276 |

LSTM produces a high BLEU in return for a low loss score and vice versa. This is not ideal as it is better to have a balance score for both scores than have a very good score for one, and a bad score for the other.

**SimpleRNN**

Chart, line chart

Description automatically generated

|  |  |
| --- | --- |
|  | High BLEU & low loss score |
| Loss score | 1.6881 |
| Bleu score | 1.4837867640225538 |

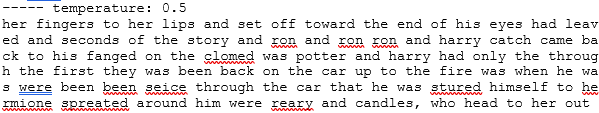
Based off the BLEU score and loss score, SimpleRNN has a much higher loss score for a BLEU score that is not very high as well. As a result, SimpleRNN would not be used.

**GRU**

Chart, line chart

Description automatically generated

|  |  |  |
| --- | --- | --- |
|  | Highest BLEU score: | Lowest loss score: |
| Loss score | 1.5698 | 1.5577 |
| Bleu score | 1.5164169595070107 | 1.4666302765429156 |



**Bidirectional LSTM**

Chart, line chart

Description automatically generated

|  |  |  |
| --- | --- | --- |
|  | Highest BLEU score: | Lowest loss score: |
| Loss score | 1.6260 | 1.5972 |
| Bleu score | 1.5164169595070107 | 1.4837867640225538 |

Text, letter

Description automatically generated

Both GRU and Bidirectional LSTM faired well in terms of BLEU score and loss score. Though both models are not coherent and seem to generate gibberish text, GRU produced a slightly more balanced scores and its sentences made a little more sense compared to bidirectional LSTM.

**Learning rate: 0.01**

**Chart, line chart

Description automatically generated**

Text

Description automatically generated

|  |  |  |
| --- | --- | --- |
|  | Highest BLEU score: | Lowest loss score: |
| Loss score | 1.5920 | 1.5907 |
| Bleu score | 1.46663 | 1.43038 |

A learning rate of 0.01 caused a sudden spike in the loss scores at the 15 epochs. This means that 0.01 is not a suitable value for the model and will not be used.

**Learning rate: 0.0001**

**Chart

Description automatically generated**

Text

Description automatically generated

|  |  |  |
| --- | --- | --- |
|  | Highest BLEU score: | Lowest loss score: |
| Loss score | 2.0748 | 1.46663 |
| Bleu score | 1.46663 | 1.46663 |

Text, letter

Description automatically generated

Original model of 0.001 learning rate:

Text, letter

Description automatically generated

It is apparent that the learning rate of 0.001 is grammatically more correct and produces sentences with lesser spelling errors. Though the sentences may be slightly repetitive, it still makes a better model compared to a learning rate of 0.0001.

From this model onwards, I trimmed the temperature to test for only 0.5 and 1.0. This is because 0.2 has way too many repeated words to be useful to the evaluation of the generated text while a temperature of 1.2 has too many creative words that it is impossible to make out the sentence.

Cutting down on the temperatures that provide us with more useful sentences is easier for us to go through and less time is spent on running the models as well.

A picture containing text

Description automatically generated

**Batch size: 64**

Chart, line chart

Description automatically generated

Text

Description automatically generated

|  |  |  |
| --- | --- | --- |
|  | Highest BLEU score: | Lowest loss score: |
| Loss score | 1.5745 | 1.5553 |
| Bleu score | 1.50036 | 1.43038 |

Text, letter

Description automatically generated

**Batch size: 256**

Chart, line chart

Description automatically generated

Text

Description automatically generated

|  |  |
| --- | --- |
|  | Highest BLEU & lowest loss score: |
| Loss score | 1.5616 |
| Bleu score | 1.466630 |

Minimal to no overfitting for both models, which is a good sign

Text, letter

Description automatically generated

Original model with batch size of 128:

Text, letter

Description automatically generated

Out of all these 3 different batch sizes, batch 256 performed the worst as it is hard to make out its sentences, and they did not make much sense. On the other hand, batch 64 and 128 were more repetive such as batch 64 having multiple ‘car’ words anad128 having multiple ‘ron’ words. However, in terms of context, using the original batch size of 128 seems to be more optimal.

#### Hidden units in layers: 256

#### Chart, line chart Description automatically generated

Text

Description automatically generated

|  |  |
| --- | --- |
|  | Highest BLEU & lowest loss score: |
| Loss score | 1.5448 |
| Bleu score | 1.4488496539373276 |

#### Number of GRU layers: 2, Hidden units: 256, 128

#### 



|  |  |
| --- | --- |
|  | Highest BLEU & lowest loss score: |
| Loss score | 1.4944 |
| Bleu score | 1.430389201127128 |

Tried different hidden layer units and different number of layers. However, this caused the model to overfit as seen from the loss chart. This can happen when we increase the layers and units, the model has more parameters to adjust and can therefore easily fi the idiosyncrasies and noise in the training data. Instead, I proceeded to try out smaller hidden layer units instead.

A model with 2 GRU layer also worked better since for almost the same BLEU score, a model with 2 GRU layer achieved a much lower loss score compared to when there is only one GRU layer.

#### Number of GRU layers: 2, Hidden units: 128, 64

#### Chart, line chart Description automatically generated

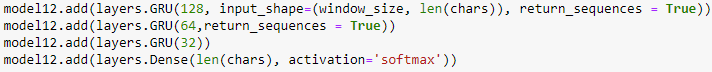


|  |  |  |
| --- | --- | --- |
|  | Highest BLEU score: | Lowest loss score: |
| Loss score | 1.5286 | 1.5235 |
| Bleu score | 1.48378 | 1.39116 |

#### Text, letter Description automatically generated

#### Number of GRU layers: 2, Hidden units: 64,32

#### Chart, line chart Description automatically generated



|  |  |  |
| --- | --- | --- |
|  | Highest BLEU score: | Lowest loss score: |
| Loss score | 1.6871 | 1.5509 |
| Bleu score | 1.48378 | 1.44884 |

#### Text, letter Description automatically generated

Hidden units of 64,32 actually obtained only a few spelling errors which I an improvement compared to other models. However, thesentences are extremely broken and gramatically wrong like ‘into in the for’ being side by side. On the other hand, the context of the text generated with a hidden layer of 128,64 seemed better though the multiple spelling errors. It is more optimal if a model with more coherent sentences is chosen, hence the hidden layer units of 128 and 64 is chosen.

#### Dropout: 0.2, 2 layers

#### Chart Description automatically generated

A picture containing text, indoor, screenshot

Description automatically generated

|  |  |  |
| --- | --- | --- |
|  | Highest BLEU score: | Lowest loss score: |
| Loss score | 1.5369 | 1.5369 |
| Bleu score | 1.46663 | 1.44884 |

#### Text, letter Description automatically generated

Comparing BLEU and loss scores, a dropout of 0.2 has a much higher loss score, for a BLEU score that is not very high. This could be because our initial model does not overfit much and adding dropout may have caused too much information to be lost, in which the model could not learn effectively. As a result, attempts using a smaller dropout and reduce the number of layers has been tested. Recurrent dropouts were also used.

#### Dropout: 0.1, 2 layers

#### Chart Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence

|  |  |  |
| --- | --- | --- |
|  | Highest BLEU score: | Lowest loss score: |
| Loss score | 1.5435 | 1.5111 |
| Bleu score | 1.48378 | 1.46663 |

#### Text Description automatically generated

#### Dropout:0.1, 1 layer

#### Chart Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence

|  |  |  |
| --- | --- | --- |
|  | Highest BLEU score: | Lowest loss score: |
| Loss score | 1.6374 | 1.4787 |
| Bleu score | 1.46663 | 1.41118 |

#### Text, letter Description automatically generated

#### Recurrent dropout: 0.1, 2 layers

#### Chart Description automatically generated

Text, letter

Description automatically generated

|  |  |  |
| --- | --- | --- |
|  | Highest BLEU score: | Lowest loss score: |
| Loss score | 1.6406 | 1.4512 |
| Bleu score | 1.46663 | 1.37023 |

#### Text, letter Description automatically generated

#### Recurrent dropout: 0.1, 1 layer

#### Chart, line chart Description automatically generated

Text

Description automatically generated

|  |  |  |
| --- | --- | --- |
|  | Highest BLEU score: | Lowest loss score: |
| Loss score | 1.6359 | 1.4696 |
| Bleu score | 1.50036 | 1.48378 |

#### Text, letter Description automatically generated

Out of all the other four dropout models, the model above with a current dropout of 0.1 and only one dropout layer appears to have fewer grammatical mistakes and spelling errors. “See it was on the rest of the dark before the breath. you know what way and Hermione” is relatively well structured, and contains no spelling errors, though the sentence still may not make sense.

This also shows that a smaller dropout value was effective to our model compared to using a larger value. Recurrent dropouts also worked better for the model which could be because this is an RNN layer. Furthermore, the use of using normal dropouts may disrupt the flow of information and make the training process less effective as it dropout neurons in random manner compared to recurrent dropouts that ensures the same neurons are dropped out at each training iteration.

**L1 Regularizer: 0.01, 2 layers**

**Chart

Description automatically generated**

Text

Description automatically generated with medium confidence

|  |  |  |
| --- | --- | --- |
|  | Highest BLEU score: | Lowest loss score: |
| Loss score | 3.0676 | 3.0633 |
| Bleu score | 1.51641 | 1.48378 |

**L1 Regularizer: 0.001, 2 layers**

**Chart

Description automatically generated**

Text

Description automatically generated with low confidence

|  |  |
| --- | --- |
|  | Highest BLEU & lowest loss score: |
| Loss score | 3.0157 |
| Bleu score | 1.46663 |

As shown from both loss chart, the validation loss is a straight line, and its loss score is very high after using an L1 regularizer. Its loss score doubles the usual loss score for our model at a value of 3. This typically means that the model is underfitting and an underfitting model has a high validation loss because it is not able to generalize well to new, unseen data. This is because when using L1 regularization, the model is penalized for having large weights. This can lead to a reduction in the complexity of the model and therefore result in underfitting.

**L2 Regularizer: 0.01, 2 layers**

**Chart

Description automatically generated**

Text

Description automatically generated with medium confidence

|  |  |
| --- | --- |
|  | Highest BLEU & lowest loss score: |
| Loss score | 2.1502 |
| Bleu score | 1.51641 |

Text, letter

Description automatically generated

**L2 Regularizer: 0.001**

**Chart

Description automatically generated**

Text

Description automatically generated with medium confidence

|  |  |  |
| --- | --- | --- |
|  | Highest BLEU score: | Lowest loss score: |
| Loss score | 1.9984 | 1.7716 |
| Bleu score | 1.48376 | 1.44884 |

#### Text, letter Description automatically generated

#### As for L2 regularizers, even though there is no overfitting as seen by the loss chart, the text generated is jumbled and contains incoherent string of words with incorrect spellings, grammar, and context. It is very difficult to understand the intended meaning or message being conveyed. Thus, it is clear that having no regularizers is better in terms of model performance and the text generated.

#### Optimizers: RMSprop

#### Chart Description automatically generated

A picture containing text

Description automatically generated

|  |  |
| --- | --- |
|  | Highest BLEU & lowest loss score: |
| Loss score | 1.4964 |
| Bleu score | 1.44884 |

#### Text, letter Description automatically generated

Model with RMSprop optimizer did not overfit and has a decent loss and BLEU score.

#### Original model with optimizer Adam:

#### Text, letter Description automatically generated

The generated text with Adam optimizer is slightly easier to understand due to its relatively coherent sentence structure and fewer spelling errors compared to the generated text with RMSprop. As a result, I went with the Adam optimizer.

**Sampling Methods**

Stochastic sampling

This method generates a probability distribution over the vocabulary and selects the next word based on that probability distribution.

#### 

#### Random sampling

This method selects the next character based on a probability distribution over the characters, which is given by the model. This allows for greater diversity in the generated sequences but can also result in less coherent and less meaningful text.

#### A screenshot of a computer Description automatically generated with medium confidence

#### Greedy Sampling

This method selects the most likely character at each step based on the output of the model. This leads to highly coherent sequences, but with limited diversity.

#### A screenshot of a computer Description automatically generated with medium confidence

#### Different sampling methods will be tried and compared. These sampling methods chosen allows the model to generate more diverse output, which is important in creative writing or when generating text that is meant to be read by humans.

**Random Sampling**

**Chart

Description automatically generated**

Text

Description automatically generated

|  |  |
| --- | --- |
|  | Highest BLEU & lowest loss score: |
| Loss score | 1.4569 |
| Bleu score | 1.44884 |

Text

Description automatically generated

**Greedy Sampling**

**Chart

Description automatically generated**

Text

Description automatically generated

|  |  |
| --- | --- |
|  | Highest BLEU & lowest loss score: |
| Loss score | 1.4651 |
| Bleu score | 1.34830 |

Text, letter

Description automatically generated

With almost no spelling errors, greedy sampling might seem like the best option here but the number of repeated words in a single text is too much. The sentence ‘potter storming the corridor to his potter started to his potter’ is being repeated throughout the whole text which is not ideal. As a result, greedy sampling is not an option.

**Original model with Stochastic sampling:**

Text, letter

Description automatically generated

Overall, between stochastic sampling and random sampling, stochastic sampling was chosen because it is slightly more coherent than random sampling. Stochastic sampling also produced a higher BLEU score with almost the same amount of loss score.

**Temperature: 0.4,0.6,0.8**

**Chart

Description automatically generated**

Text

Description automatically generated

|  |  |
| --- | --- |
|  | Highest BLEU & lowest loss score: |
| Loss score | 1.5926 |
| Bleu score | 1.46663 |

A screenshot of a computer

Description automatically generated with medium confidence

Based on the generated text, it appears that the best text was produced with the temperature of 0.6. A temperature of 0.8 also seems to produce more gibberish with words that do not exist. Although all three generated text contain errors in terms of grammar and spelling, the text generated at temperature 0.6 seems to have a clearer meaning and is easier to understand compared to the temperature at 0.8 and 0.4.

**Final Model (adjusting to temp 0.6)**

A picture containing text

Description automatically generated

**Chart, line chart

Description automatically generated**

Text

Description automatically generated

|  |  |  |
| --- | --- | --- |
|  | Highest BLEU score: | Lowest loss score: |
| Loss score | 1.4840 | 1.4698 |
| Bleu score | 1.50036 | 1.37023 |

Highest BLEU score:

Text

Description automatically generated

Lowest loss score:

Text

Description automatically generated

Our final model has a fair BLEU score and loss score and has almost no overfitting, which is very good. The text generated will not always be the best as the sentences are still a jumbled mix of words and phrases that do not form a meaningful or understandable sentence. However, there are still sections of sentences that make slight sense and the spelling errors have reduced significantly. In terms of context, the sentences still do not have a clear context.

# Use the developed Model to Generate Texts

## 4.1 Analyze generated text

The text input I provided is a sentence describing Harry’s thoughts and actions in front of the Gryffindor common room. It mentions that Harry is thinking about tasks he needs to complete, including homework and an exam, and that he needs to grab a quick bite to eat.

A picture containing text

Description automatically generated

The generated text is generated using our final model which has been mentioned above. The text, however, seems to be completely different from the input text and lacks coherence. The output text seems to be a random combination of words and phrases that don’t form a meaningful sentence. Some of the words and phrases in the generated text, such as ‘malfoy finches’,’counterse’ do not seem to have any connection to the input text. There are also words that were generated which do not exist such as ‘iduries’.

Even with this, the generated text still contains recognizable words and references to the original input. Part of the sentence such as ‘in the car explaining in his wand’ is one of the logical sections of the sentences. Additionally, The amount of spelling errors has also been significantly reduced.

Text

Description automatically generated

Overall, it is important to acknowledge that this model has its limitations of text generation. However, with further improvements and advancements, models like this have the potential to generate high-quality, sophisticated text.

## 4.2 Real- life applications

Text generation models can be applied to many different industrial applications nowadays and one such successful application is in customer support. A text generation tool can provide real-time chatbot support to customers, as well as prepare personalized customer service answers. Such tools can shorten response times and improve customer satisfaction.

Another use case of a text generation model is text summarization. AI text generators can be used to create summaries of longer texts. They offer various possibilities such as creating news latter, summarizing internal company documents, assisting educators in preparing educational material by providing them with summarized content of sources and facilitating the review of literature in research contexts and much more.

Insurance companies have also seen progress in efficiency after adopting a text generation model. Insurance companies evaluate long-written applications in their claims management process to decide whether a case is eligible for the insurance settlement process. (Plagge, Felix, 2022)

A deep learning model called sequence to sequence architecture was implemented to resolve the problem. This is a neural network type commonly used for machine translation, answering questions and many more. As a result of the adoption of this model, summaries of application are generated which makes the decision-making process faster and prevents waste of time.

# Summary

## **Model performance summary**

The best results were obtained using a GRU layer with a learning rate of 0.001 and Adam optimizer. The optimal batch size was found to be 128, with the best number of layers and units being 2 GRU layers with 128 and 64 units respectively. Recurrent dropouts were found to be more effective than regular dropouts. Regularizers did not improve the model performance and were not included in the final model. Stochastic sampling was determined as the best choice among random and greedy sampling methods. Different temperatures were tested, and it was determined that a temperature of 0.6 produced the best quality of generated text.

In terms of final model performance when tested on the input text, the model generated a coherent text, but it does not accurately reflect the content or context of the original input text. The generated text contains some recognizable words and phrases, but the overall meaning is difficult to understand and seems to be gibberish. There is definitely still be a need for further tuning of the model's parameters, such as the number of layers and units, the optimizer, the sampling method, and the temperature, among others, to improve the model's performance.

## 5.2 Improvements

There are several methods out there that can be added to improve the current model. Firstly, the text data may not have been clean and pre-processed thoroughly. This may have left out irrelevant information and noise which has an adverse effect on the model’s performance.

Like every dataset, it will come with its limitation and the harry potter text data could have been too small for the model to learn diverse patterns and generate more accurate results.

In addition, there are many other architectures there were not experimented with. In this paper only 4 different architects were tested, there are many others out there such as bidirectional LSTM or transformer model which could have been compared with the model created. Furthermore, transfer learning, where the use of pre-trained models is fine-tuned for specific tasks were not tested and could have been an effective way to improve the text generation model.

Another approach is the use of Generative Adversarial Networks (GAN). The GAN can learn the underlying patterns in data and generate more deviser and nuanced outputs while traditional text generation models may struggle to capture the complexities and nuances of language, incorporating a GAN can result in more sophisticated and realistic outputs.

Natural Language Processing (NLP) Techniques such as attention mechanism can also be incorporated to the model. These techniques have shown promising results in a variety of NLP tasks and ma help to unlock the full potential of the text generation model.

Aside from adjusting the details of the model, more could have been done towards the model evaluation such that the most optimal model would be chosen. Being the only person on this project accessing the models, it is easy for me to have biased opinions on the generated models. As a result, having a group of people come together to judge the quality and relevance of the generated text can reduce biasness.

In addition to evaluation metrics, there are several techniques and strategies that can be used to improve text generation models such as incorporating additional information such as contextual information or external knowledge sources.

Overall, there is a wide range of techniques and strategies that can be used to improve the performance of and quality of the text generation models. Choosing the right choice of techniques will definitely improve the model’s performance such that I would be able to generate the desired output.

# Reflection

As for problem 1, what really struck me was how important data cleaning and pre-processing can be. This was because initially my team members all had a score of 40-45% accuracy at best even after fine tuning. This clearly means our model does not predict half of the data accurately, which is not good. It took us quite some time to actually figure out what was wrong and after going through a lot of testing and trying, we managed to find a data processing method that lifted all our accuracy scores up. It was surprising as the main issue actually came from the fact that the 5 classes being 1 star, 2 star and so on, seems to be detrimental to our model. Instead after, correcting it to be three classes only, grouping them by positive, neutral, and negative significantly improved our model.

Another issue that intrigued was how everyone fine tune their own models, but Minh was able to produce a model that was suitable for everyone and managed to hit the highest accuracy for the whole team as well. This could be due to the fact that the pre-processing was done on his dataset as testing and then applied to the rest of the team. However, it is still impressive that his model accommodated everyone dataset and performed remarkably well.

Additionally, learning about the use of embedding and pre-train embeddings in a sentimental analysis model was new to me compared to always just fine-tuning the usual hyper parameters like learning rate, batch size and more. Furthermore, before this task, I only knew about Conv1D networks but after this I realized the many different neural networks such as GRU, LSTM, SimpleRNN and even Conv2D.

Lastly, the fact that the model accurately predicted all of our reviews was amazing. With the different types of reviews, we had such as a long form, short and straightforward, and vague but medium length review, the model still managed to classify these reviews into its correct categories.

Before coming to this module, I had never known the existence of text generation models. However, throughout my time I have learnt the various aspects of a text generation model, such as the different sampling methods, the importance of normalizing the predicted probabilities. Additionally, I have also learned about different techniques to improve the performance of a text generation model, such as changing the step size, window size and the different temperatures.

It is intriguing to see how these different techniques can affect the performance of a text generation model and how the results can vary. For example, adding more hidden layers may sometimes improve the performance of the model, while other times it may decrease the performance. This highlights the importance of carefully experimenting with different techniques and selecting the ones that work best for a particular task.

Overall, the text generation model has the potential to be useful in a variety of real-life applications, such as language translation or summarization. By learning about these different techniques and how they can impact the performance of a text generation model, I gained a deeper understanding of the field of deep learning and its applications.