

# School of InfoComm Technology

**Deep Learning Assignment**

Diploma in DS / FI / IT

Oct 2022 Semester

**ASSIGNMENT 2**

(40% of DL Module)

**Submission Deadline:**

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

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

|  |  |  |
| --- | --- | --- |
| **Tutorial Group** | **:** | **T01** |
| **Student Name** | **:** | Tran Huy Minh |
| **Student Number** | **:** | S10223485H |

**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.

Contents

[1. Overview 3](#_Toc127117807)

[2. Data Loading and Processing 4](#_Toc127117808)

[3. Develop the Sequence Generator Models 6](#_Toc127117809)

[4. Use the developed Model to Generate Texts 14](#_Toc127117810)

[5. Summary 16](#_Toc127117811)

[6. Reflection 17](#_Toc127117812)

# Overview

Part 2 of this assignment endeavors to construct a Recurrent Neural Network (RNN) that generates English language characters, with the goal of producing semi-coherent sentences from scratch. To accomplish this task, the training data will consist of J.K. Rowling's classic, "Harry Potter and the Philosopher's Stone”.

The procedure begins with the loading and processing of the data, where the "Harry\_Potter\_Book2.txt" file is opened, read, and the number of characters is counted. After basic cleansing, the unique characters and punctuations are identified, and the data is transformed into training text and labels (X & y) through the "sliding window" method. These data are then one-hot encoded and converted into binary arrays.

The sequence generator model is developed next, where models that incorporates RNN layers, such as LSTM or GRU, are to be built. During the training phase, the data is divided into training and validation samples, and a universal machine learning workflow is employed to optimize the model's accuracy. Starting with a baseline model, the model's complexity is increased until overfitting occurs, and then regularized to prevent overfitting. Model performance is analyzed in terms of accuracy and quality of generated texts, including training and validation accuracy and loss scores.

Finally, the model is put to the test by generating texts using one-hot encoded input recorded from the user. The model produces 400 characters, and the generated texts are analyzed to assess their coherence.

In conclusion, this project aims to develop a character generator capable of producing semi-coherent English sentences using Harry Potter and the Philosopher's Stone as training data. The project involves loading and processing the data, developing an RNN-based sequence generator model, analyzing the model's performance, and finally generating texts.

# Data Loading and Processing

Data loading and preprocessing is a critical component of the process of training a text generation model, as it lays the foundation for the quality and accuracy of the final model. The success of a text generation model is heavily dependent on the quality of the data it is trained on.

During data loading and preprocessing, various tasks are carried out to ensure the data is in a suitable form for the model. These tasks include data cleaning, data standardization, feature engineering, and data preparation.

Data cleaning helps to eliminate inconsistencies, missing values, duplicates, or irrelevant information that may compromise the model's performance. Through data standardization, data is transformed into a consistent format and scale, making it easier for the model to learn and make predictions. Feature engineering is an important step for text data, as it helps to extract meaningful information from the data by converting the text into numerical representations and transforming the text into sequences.

Finally, data preparation transforms the text data into a specific format suitable for training a text generation model, including splitting the text into sequences of characters and transforming the text into numerical arrays. By performing these steps, the data is transformed into a form that can be easily understood and processed by the model, resulting in improved model performance and predictions.

The data loading component begins with opening and reading the "Harry\_Potter\_Book2.txt" file. The corpus of text is loaded into memory and encoded as utf-8, with all characters being transformed to lowercase. The length of the text is then printed to provide an idea of the size of the data being processed which in total is 531,708 characters long.

The data processing component of the code then performs basic cleansing on the text to remove any unnecessary characters that might negatively impact the quality of the data. This is achieved by utilizing the "decontracted" function, which replaces contracted words with their full forms. Additional cleaning is performed by removing specific characters such as brackets, percent signs, and other symbols that have little correlation to the text found in books and novels. It is worth mentioning that every “page” of the dataset mentions the J.K. Rowling, the author’s name, which might create a bias to reproducing her name in text generation and as such, every mention of her name is replaced with a blank string.

The "sliding window" method is used to prepare the data for training, with the window length being set to 60 characters and the step size to 3. This method involves splitting the text into sequences of characters (referred to as sentences) and the next character in the sequence (referred to as next\_chars). These sequences are stored in separate lists.

Finally, the data is transformed into a binary format through one-hot encoding, which is a technique for converting categorical data into a numerical format that can be used for training a model. A list of unique characters in the text is created, and a dictionary is created that maps each unique character to its index in the list. The X and y arrays are then initialized as arrays of zeros with the same shape as the number of sentences and the number of unique characters, respectively. The one-hot encoding is performed by setting the value of the corresponding character index in the X and y arrays to 1.

The result of this code is a set of data that has been transformed into a format suitable for training a predictive model for text generation. A train test split of 80% to 20% is done to be used as validation data to determine the test accuracy of the model.

# Develop the Sequence Generator Models

To build and train the models for this assignment, a baseline model was first established to compare the subsequent model’s performance and complexity. The model's performance was also analyzed during the training phase to identify any trends or patterns that might point to areas for improvement. This included tracking key metrics such as the model's accuracy, loss, and convergence rate, as well as visualizing the performance through graphs and plots. By regularly monitoring the model's performance, any issues were able to be identified and adjustments were made to the model or training process as needed to enhance its performance.

To evaluate the performance of the models, the model’s accuracy, loss, perplexity of text generated, and comprehensiveness of the text will be analyzed.

Perplexity is a measure of the average number of decisions the model needs to make in order to generate the next word in a sequence, and it is often used as a proxy for the model's confidence in its predictions. The lower the perplexity, the more confident the model is in its predictions, and the higher quality the generated text is likely to be.

Comprehensiveness is a subjective measure of the coherence, relevance, and grammatical correctness of the generated text. A model with high comprehensiveness is able to generate coherent and relevant text that follows grammatical rules and conveys meaningful information.

The temperature of the input to a text generation model can affect the generated text by adjusting the model's confidence in its predictions. By adjusting the temperature, you can control the amount of randomness in the generated text and influence its diversity. A lower temperature results in more conservative, predictable outputs, while a higher temperature leads to more diverse, unpredictable outputs.

Below is a table of every recorded model that was built, some of their changes, and their performance at the final epoch. More in-depth explanations below:

|  |  |  |  |
| --- | --- | --- | --- |
| Model | Test/Tune/Changes | Train Accuracy | Test Accuracy |
| 1 | Baseline Model – only embedding layer | 39.76% | 36.96% |
| 2 | Used LSTM layer with 32 units | 44.42% | 45.13% |
| 3 | Used GRU layer with 32 units | 47.01% | 47.55% |
| 4 | Increased GRU neuron units to 128, used RMSprop learning rate of 0.001 | 56.82% | 55.09% |
| 5 | Used LSTM layer with 128 units | 54.40% | 53.16% |
| 6 | Used two GRU layers with 128 units each, increased learning rate to 0.002 | 64.36% | 58.46% |
| 7 | Increased one layer’s units to 256, added dropout of 0.2, increased learning rate to 0.003 | 60.99% | 58.60% |
| 8 | Increased other layer’s units to 160, added another dropout of 0.2, increased learning rate to 0.004, added L2Regularizer of 0.01 | 54.20% | 54.56% |
| 9 | Reduced units back to 128, removed one dropout | 55.04% | 55.27% |
| 10 | Doubled number of units for both layers, increased dropout to 0.3, decreased regularizer to 0.005, used Adam learning rate of 0.001 | 65.19% | 59.51% |
| 11 | Increased regularizer back to 0.01 | 63.69% | 59.11% |
| 12 | Increased dropout to 0.35 and learning rate to 0.0015, 20 epochs | 64.23% | 58.93% |

To fine-tune the model's hyperparameters, a range of different values for each parameter were tested and assessed the effect on the model's performance. By carefully adjusting the hyperparameters, the researcher was able to enhance the model's accuracy in classifying the images and decrease overfitting.

Starting with the first model, the baseline, a single simple embedding layer was added, a softmax activation with 50 classification types, which is the number of unique characters, categorical cross-entropy, RMSprop optimizer function with its default learning rate. It should be noted that a batch size of 128 and 10 epochs were used for most models.

A picture containing text

Description automatically generated

After 10 epochs, the model was able to achieve a training accuracy of 39.76% and a testing accuracy of 36.96% acting as a good baseline for the remaining models. The accuracy and loss chart can be seen below, showing that the model contained a lot of noise and variance as it was unable to keep a consistent accuracy of loss value.

Chart, line chart

Description automatically generatedChart, line chart

Description automatically generated

For my Model 2 and 3, both model’s replaced the embedding layer with either a LSTM layer or a GRU layer with 32 neuron units each. This was done to determine which kind of layer would be more suitable for this problem.

LSTMs are designed to handle the vanishing gradient problem in traditional RNNs by introducing memory cells, input gates, and output gates that allow information to be selectively passed through the network. This helps LSTMs to remember important information from long sequences of data and prevent irrelevant information from affecting the prediction.

GRUs, on the other hand, are a simplified version of LSTMs with two gating mechanisms, a reset gate, and an update gate, that control the flow of information into the network. GRUs are faster and computationally more efficient compared to LSTMs but may not perform as well on complex sequential problems.

A picture containing chart

Description automatically generatedA picture containing text

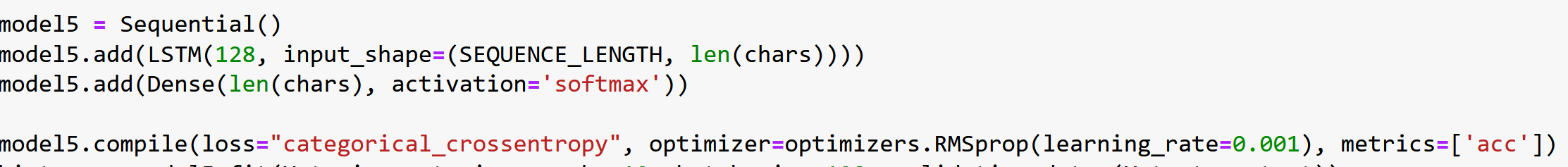
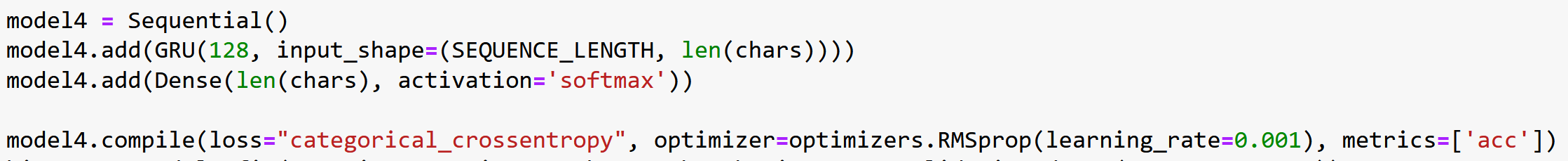
Description automatically generated

Overall, both models shared similar results both in the text generated, loss, and accuracy graphs. The only exception being that the GRU model achieved a higher test accuracy of 47.55% as compared to LSTM’s 45.13%. As both models are still underfitting and thus, a more complex model will be made to scale them up.

Chart

Description automatically generated

Similar to the previous two models, models 4 and 5 share similarities with the exception that one is GRU and the other is LSTM. The models were scaled up by increasing the number of neuron units to 128 from 32 and an RMSprop learning rate of 0.001 was used.



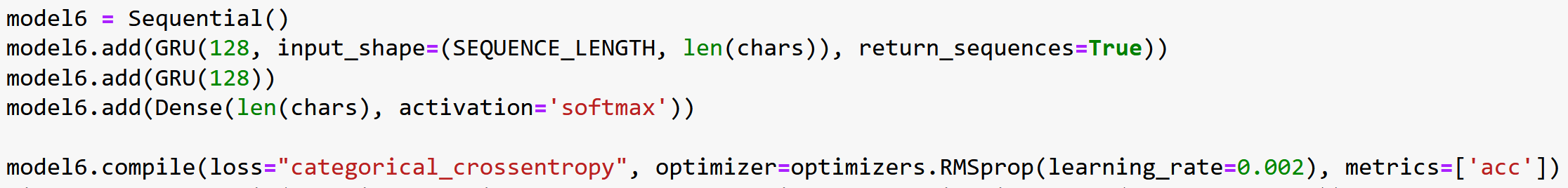
Both models once again shared similar results both in the text generated, loss, and accuracy graphs. The GRU model continues to have a higher test accuracy of 55.09% as compared to the LSTM model of 53.16%. There is a significant improvement to the accuracy due to the more complex neural networks. As both models are no longer underfitting and converge around the 5th or 6th epoch, entering their optimal range. Taking this into consideration, only GRU layers will be used for future models.

Chart, scatter chart

Description automatically generatedChart, line chart

Description automatically generated

Model 6 and 7 essentially added an additional GRU layer with 256 neuron units, a dropout of 0.2 to account for the increased complexity, and an increased learning rate of 0.003.

Text

Description automatically generated with medium confidence

The changes made in model 6 saw an increase in training and testing accuracy of 64.36% and 58.46%. This performance was improved even further in model 7 where its training accuracy decreased to 60.99% and testing accuracy increased to 58.60%. The smaller difference between the training and testing accuracy implicates that the model was able to fit the data better whilst achieving a higher accuracy. Model 7 also converges on the 3rd epoch and remains in its optimal range for the remaining epochs.

Chart, line chart

Description automatically generated

Models 8 and 9 saw an increase in complexity through the addition and removal of another dropout layer, an increased learning rate of 0.004, and an addition of the L2 regularizer of 0.01 for weight decay. These models were built to achieve an underfitting model that can be further scaled up in subsequent models. Despite their lower accuracies, the goal of underfitting was achieved.

Chart, line chart

Description automatically generated

Model 10 attempts to scale up the underfitting model by doubling the number of units for both GRU layers, increasing dropout to 0.3, decreasing the regularizer to 0.005, used Adam learning rate of 0.001.

Text

Description automatically generated

After training the model, it was able to achieve a train and test accuracy of 65.19% and 59.51%. The model converged on its 4th epoch and entered into its optimal range. Thus far, this is the best performing model purely for its high test accuracy.

Chart

Description automatically generated

For models 11 and 12, they attempted to increase model 10’s test accuracy by increasing the regularizer back to 0.01, dropout to 0.35, the learning rate to 0.0015, and the number of epochs to 20.

Text

Description automatically generated Text

Description automatically generated with medium confidenceThough they both saw a decrease in training accuracy signifying a better fitting model, their testing accuracy has not increased either meaning that model 10 still has better performance. Model 12 also converges on the 4th epoch and enters its optimal range for its remaining 16 epochs.

Chart

Description automatically generated

Now looking at some of the text generation for some of the models, the baseline model 1, despite using the lowest temperature, is unable to produce anything cohesive and is also unable to generate any actual words.

A picture containing diagram

Description automatically generated

Model 6 was able to generate text that formed actual comprehensive words for both temperatures of 0.2 and 0.5. Despite that, none of the words formed any coherent sentences. At temperature 1.0 however, it starts to misspell a lot more. The perplexity of the texts also appear to be relatively low which is not a good sign as a higher perplexity would indicate more complex text for more variety.

A picture containing calendar

Description automatically generated

Similarly, model 10 produced text that formed coherent words and occassionally phrases at the temperatures of 0.2 and 0.5. Despite some minor misspelling, the model produced some relatively comprehensive sentences such as ‘they were foul his face was a great still’ that appeared in temperature 0.5.

A picture containing calendar

Description automatically generated

Model 12 also displayed its ability to form real words that make sense with the potential of making up actual sentences. Something notable was the model’s ability to realize that double quotations meant that someone was talking as seen in the ‘”I have to see harry,” said mr. weasley’, which shows its capabilities.

Calendar

Description automatically generated

Overall, out of the twelve models, model 10 has been considered the best due to its higher test accuracy of 59.51% as well as its ability to generate comprehensive words and phrases whilst having a higher perplexity than most other models. As such, this model will be used to generate characters and sentences based on the user’s input.

# Use the developed Model to Generate Texts

To be able to apply the text generation model on a real-life input, the user’s input must first be received and undergo similar preprocessing techniques as the data that the model was trained on.

The input data is first converted to lowercase for normalization, the function for decontracting phrases is then applied and the removal of the following special characters are done, “()\*%/:;\\|•\n”.

After the final character length of the input text is obtained, it is important to either trim off excess characters that exceed the maximum length of 60, or pad the text with blank white spaces to reach 60 characters. This is an important thing to do as the input shape of the text should match the input text of the model, otherwise, the model would output complete gibberish.

The user’s input text is then encoded using one-hot encoding to match the features of the input shape of the model. The encoded input is then fed into the model to generate the next 400 characters. Below is the text generated across four different temperatures.

Text

Description automatically generated

Using the input text, ‘george was nowhere to be found when a student asked him a question,” it can be seen that the input text was trimmed off due to exceeding the 60 character max limit.

With a temperature of 0.2, where the model generates text that it is most confident in, the model was able to generate real words with little to no spelling errors. It is also able to formulate parts of comprehensible sentences such as ‘the dark art looked at his things’.

With a temperature of 0.5, they model begins to see a couple of spelling errors but still overall capable of producing actual words as well as portions of sentences. The higher temperature also allowed the model to generate more unique words such as ‘potions’, which explains its higher perplexity. An example of a comprehensive sentence is ‘harry gathered the first few fire of the end, reached the floor’. This shows the model’s capability of generating text that makes sense.

With the temperature of 1.0 and 1.2 however, the model is unable to produce anything comprehensive and coherent where nothing in the generated text makes sense.

Overall, model 10 is still capable of generating cohesive texts that makes sense at lower temperatures with the user’s input but is unable to produce anything comprehensive at higher temperatures.

# Summary

The purpose of this project was to develop a character-based text generator using Recurrent Neural Network (RNN) technology. The model was trained on the text of J.K. Rowling's "Harry Potter and the Philosopher's Stone". The project involved the loading and preprocessing of the data, development of an RNN-based sequence generator model, analysis of the model's performance, and generation of texts.

Data loading and preprocessing were carried out to ensure the quality of the final model. The data was cleaned, standardized, and transformed into a suitable format for the model through feature engineering and data preparation. The final processed data consisted of 531,708 characters, which were transformed into binary arrays through one-hot encoding. The data was split into a training set of 80% and validation set of 20% to measure the model's test accuracy.

The sequence generator model was built using a baseline model and increasing its complexity until overfitting occurred, then regularizing it to prevent overfitting. The model was optimized using a universal machine learning workflow. The performance of the model was analyzed in terms of accuracy and quality of generated texts. The model was able to generate 400 characters and the generated texts were analyzed for coherence.

Overall, the 10th model was the best model in generating text as it has a high testing accuracy of 59.51%. Using it to generate 400 characters based on the user’s input as a way to apply the model on a real-life text input, it was able to produce coherent words and phrase as well as portions of sentences at lower temperatures but fails at higher temperatures.

To improve the text generation quality, several suggestions have been made. The training dataset could be expanded through data augmentation techniques, such as changing the wording of sentences, adding or removing words, or reordering sentences, which would help the model generalize better and reduce overfitting. The data preprocessing techniques can also be improved, for example by using advanced techniques like Word Embeddings or BERT encoding instead of simple one-hot encoding. The use of a larger training dataset can also lead to a better performance, as the model will be able to learn more patterns and variations in the language. In terms of the model itself, fine-tuning the hyperparameters, such as the number of units in the GRU layers, dropout rate, learning rate, regularization, and temperature, can lead to improved performance. Additionally, incorporating attention mechanisms into the model can help it focus on the most important parts of the input data, leading to better understanding of the input data and improved text generation.

# Reflection

Overall, assignment 2 has been an interesting project when it came to learning how to build recurrent neural networks. For problem 1, we were tasked with gathering review data by scraping Google’s play store for streaming service apps. Utilizing the data, we were performed numerous data preprocessing techniques to prepare it for model training. After training the models, a lot of hyperparameters needed to be tuned to obtain the best performing model. As it was a group project, all of us had different datasets and subsequently, different models. Despite that, my model was surprisingly the best in predicting the sentiments of the reviews for all four datasets.

Similarly for problem 2, where I was tasked with creating a text generation model, I learned a lot about the different preprocessing techniques needed to prepare the data for text generation training. It was also surprising to see that the simple models performed the best when it came to text generation.

A couple of comments about what I found interesting, for problem 1, I created over 200 models utilizing different types of preprocessing techniques in hopes of achieving a higher testing model accuracy. It was really intriguing to see how much more the preprocessing affected the accuracy of the model as compared to the architecture of the model itself. Because of this, I learned a lot about preprocessing and how each technique affects the model’s performance.

As for problem 2, I initially had a hard time getting to model to generate somewhat coherent text based on the user’s input. After realizing that the user input needed to share the exact same character length as the input of the model, I was able to get the model to generate extremely more comprehensive texts when compared to before.