

**School of InfoComm Technology**

**Deep Learning Assignment**

Diploma in CSF / FI / IT

Apr 2022 Semester

**ASSIGNMENT 2**

(40% of DL Module)

4th Jul 2022 – 12th Aug 2022

**Submission Deadline:**

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

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

|  |  |  |
| --- | --- | --- |
| **Tutorial Group** | **:** | **T02** |
| **Student Name** | **:** | **Keene Ng** |
| **Student Number** | **:** | **S1021895A** |

DL Assignment 2 Problem 2

Table of Contents

[1. Overview 2](#_Toc111197338)

[Problem 2](#_Toc111197339)

[Objective 3](#_Toc111197340)

[Approach 3](#_Toc111197341)

[2. Data Loading and Processing 3](#_Toc111197342)

[Data Loading 3](#_Toc111197343)

[Data Processing 3](#_Toc111197344)

[Data Cleaning 3](#_Toc111197345)

[One-hot Encoding 6](#_Toc111197346)

[3. Developing the Sequence Generator Model 6](#_Toc111197347)

[Model Using Softmax Temperature Sampling 6](#_Toc111197348)

[Scaling Up from The Baseline Model 6](#_Toc111197349)

[Regularization of the Model 16](#_Toc111197350)

[Model Using Top-k Sampling 31](#_Toc111197351)

[4. Using the Developed Model to Generate Text 33](#_Toc111197352)

[Applying the Model on a Real-life Text Input 34](#_Toc111197353)

[Analysis of The Generated Text 34](#_Toc111197354)

[5. Summary 35](#_Toc111197355)

[References 36](#_Toc111197356)

# Overview

## Problem

Using the complete version of one of J.K. Rowling’s books from the Harry Potter series, Harry Potter and the Philosophers Stone, an English language character generator was to be created and trained. This was to be done by employing a Recurrent Neural Network and utilizing sampling methods such as greedy sampling, softmax temperature sampling and top-k sampling.

## Objective

The objective of this problem was to train the deep learning model to generate semi-coherent English sentences from scratch, automatically and character-by-character. To achieve this, the model needs to be exposed to a myriad of training examples in order to learn patterns between text inputs and potential character outputs. For example, an acceptable input string is, ‘Kittens are cut’. The next character in the sentence, ‘t’, is the corresponding output that completes the input sentence to become ‘Kittens are cute’. The model would need to be trained on these to be able to generate subsequent characters accurately. Determining which sampling method is best for text generation is also part of the objective.

## Approach

Before building the model, the data needed to be processed by cleaning the text file and then converted into binary arrays. The approach to building the model follows the universal machine learning workflow of first starting with a baseline model, then scaling it up until it overfits, tunning its hyperparameters and finally regularizing the model to overcome overfitting. After developing the model, new sampling methods would then be explored, such as Greedy Sampling, Softmax Temperature Sampling, Beam Sampling or Top-k Sampling methods. Some of these explored sampling methods would then be implemented and used by the trained model to generate texts based on the sampled sequences of the text data from the Harry Potter book provided. These texts of 400 characters each would then be compared with one another to determine the best sampling method. Measurements were thought of to help decide which sampling method generates the highest quality texts. Firstly, the level of repetition of the words created by the characters generated was used. Next, the accuracy of the words generated, whether the words generated are words of the English language is also another measurement. Additionally, the measurements include the quality of structure of the generated texts, and if they follow the English structure of a sentence. This list is not exhaustive and there were many other areas to evaluate the texts generated by the sampling methods.

This report covers the details of how the approach was followed. Further details on how the data was processed, the building of the model and its performance analysis during training and testing are discussed. Moreover, the testing of sampling methods and using the best method used on the model for a real-life text input was included.

# 2. Data Loading and Processing

## Data Loading

To load the dataset of the full text of Harry Potter and the Philosophers Stone, the path of the text file needed to be initialized so that the model can open and read it. In addition, the total number of characters, sequences (sentences) and original list of all unique characters in the corpus from the original text was checked. The count was 474429, 158123 and 59 characters, sequences, and unique characters respectively.

## Data Processing

### Data Cleaning

Data cleaning was done by looking through the list of unique characters and scrolling through the text document to see what needs to be removed or replaced. For example, special characters that were not part of the sentences’ punctuation were removed.

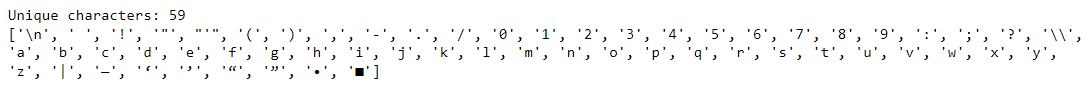


Figure List of Unique Characters in The Original Data Text

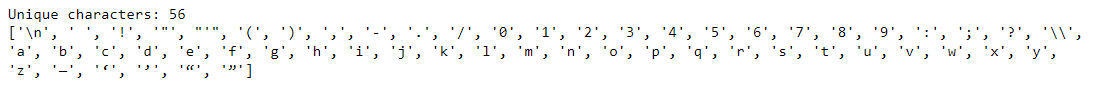


Figure List of Unique Characters in the Data Text After Cleaning

Text

Description automatically generated with medium confidence

Figure English Word Replaced with a Non-alphabetic Character

As seen in Figure 1, there was a special character of a square and a dot, which were removed as shown in Figure 2. Furthermore, when scrolling through the data text file, it was noticed that some of the “I” characters were replaced with the slash character, “/” which can be seen from Figure 3. Hence, I corrected these mistakes by changing the “/” characters to “I” where appropriate.

Graphical user interface

Description automatically generated with medium confidence

Figure Book Page Line

Text, letter

Description automatically generated

Figure Unnecessary Characters in The Original Data File

The page lines in the data text file were also removed such as the example given in Figure 4, because they are not sentences in the book. Not only does it not add value to the model’s training, but also negatively affects it, so they had to be filtered and removed using Microsoft Excel. In addition, characters that were found to not add value to the data were removed like the example in Figure 5 showing 3 “k” characters that were not part of any sentence or paragraph.

Text

Description automatically generated

Figure Sequence Sampling Containing Multiple Blank Lines

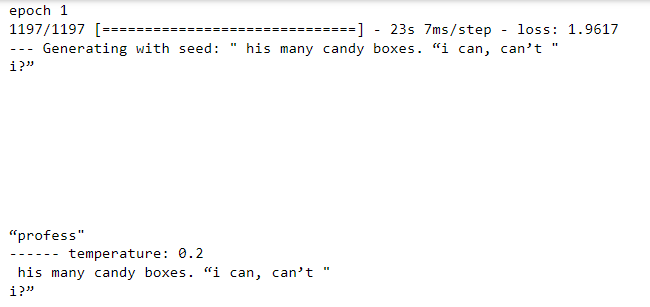


Figure Generation of Many Blank Lines During First Training of Model

All the extra blank lines were removed from the data text. The sequence sampling contained multiple blank lines as shown in Figure 6. Thus, many blank lines were generated as part of the 400 characters during the first training of the model as seen in Figure 7. This was done by opening the data text file in Microsoft Excel, deleting all the blank cells, and shifting the remaining lines up where there were empty cells above.

### One-hot Encoding

In order to perform natural language processing, the text data was converted into numeric form to be suitable to be fed into the recurrent neural network model. To convert the text data into binary arrays, one-hot encoding was performed on induvial characters of each sampled sequence, X, and their respective next character, y.

# 3. Developing the Sequence Generator Model

The development of the sequence generator model followed the approach of the universal machine learning workflow mentioned in the overview. First, it started off from a baseline model. Afterwards, it was scaled up by increasing the number of nodes per layer and adding more Long Short-Term Memory (LSTM) Recurrent Neural Network (RNN) layers until overfitting occurred. After the model overfits, regularization was implemented to the model to overcome overfitting. This was done by adding regularization functions of l1 and l2 and experimenting with dropout in each layer.

## Model Using Softmax Temperature Sampling

### Scaling Up from The Baseline Model

In the beginning, the base model had one LSTM layer with 32 neurons and one dense output layer with the neurons set to the total number of unique characters and the activation function set to softmax.

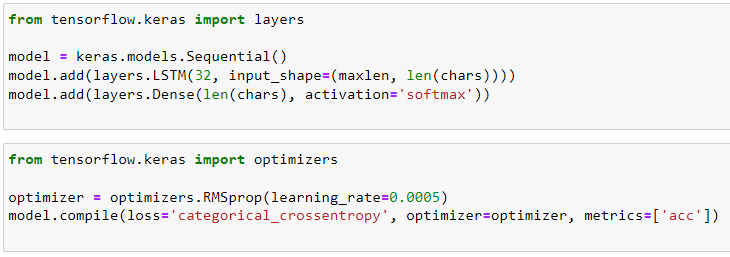


Figure Base Model Code

Furthermore, the model was compiled with the optimizer, RMSprop, with a learning rate of 0.0005 and the loss function, categorical cross entropy.

For this model, softmax temperature sampling method was used to choose the next character to generate. This sampling method controls the amount of randomness of the generated text, the higher the temperature, the more random the generated text becomes.

Generation of texts character by character requires four steps. First, drawing a probability distribution from the model of the next character based on the text presented to it. Next, reweighting the distribution to a specific level of randomness (temperature). Subsequently, taking the next character randomly based on the new distribution, and finally adding the character at the end of the text.

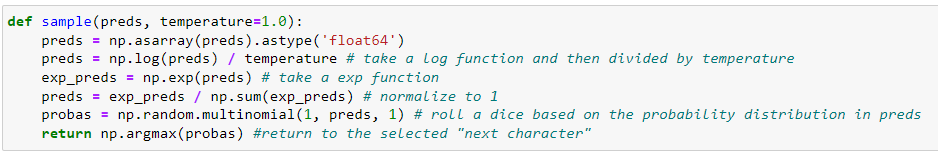


Figure Sampling Function Code

The code shown in Figure 9 is of the sampling function used to reweight the initial probability distribution and select the next character’s index.

Text

Description automatically generated

Figure Code for Model Fitting and Loop to Generate Text Based on Trained Model

After fitting the model and training it, a loop is used to call the sample function to generate the texts, which is seen from Figure 10. Every version of the model trained, texts of 400 characters were generated with four different temperatures of, 0.2, 0.5, 1.0 and 1.2 to analyze how different levels of randomness affects the text generation.

Text, letter

Description automatically generated

Figure Text Generated by the Base Model

Based on the generated text of the base model in Figure 11, the lower the temperature, the more reiteration there is. For example, the words ‘the’, ‘were’ and ‘said’ were generated extremely frequently, however, all the words except two words are legitimate words. On the contrary, higher temperatures result in more artistic and varied texts generated. There are more variations of characters and words generated even though more of the words generated are not actual words due to a higher temperature. The highest temperature, 1.2 mostly generated gibberish strings.

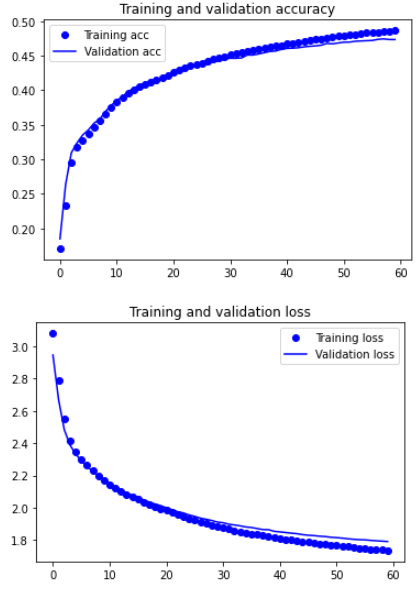


Figure Baseline Model Performance Curve

The validation accuracy of the base model after training was about 0.47 and is almost identical to that of the training accuracy. The model did not overfit after training for 60 epochs shown in Figure 12 and the validation accuracy was about 1.8.

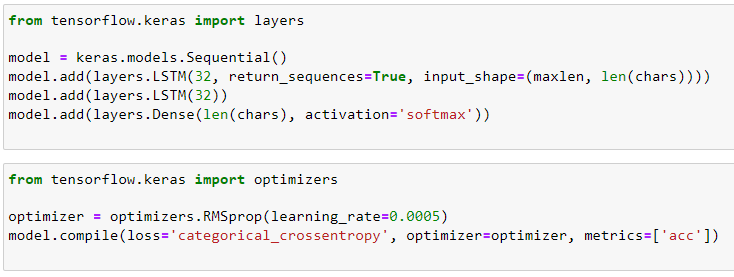


Figure Increasing Number of LSTM Layers of the Base Model

One additional LSTM layer with 32 neurons was added to the baseline model as the first step to scaling up. Previous LSTM layers must set return sequences to ‘True’ in order to add an additional RNN layer.

Text, letter

Description automatically generated

Figure Generated Text After Adding an Additional LSTM layer with 32 Neurons

Overall, the structure and quality of the generated texts were about the same as the base model after the first scaling up.

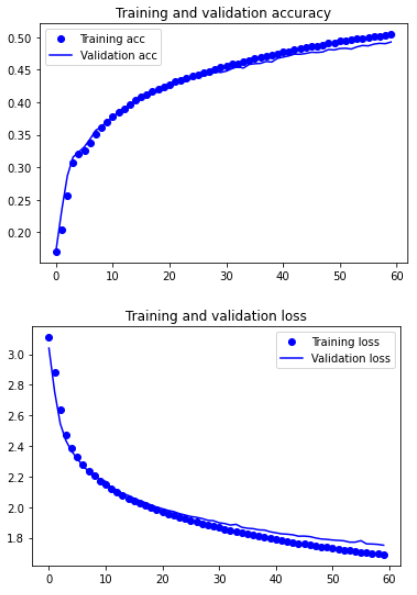


Figure Performance Curve After Adding One LSTM Layer with 32 Neurons

Overfitting did not occur, and the performance of the model increased slightly from about 0.47 to close to 0.5 while the validation loss also decreased insignificantly. The validation accuracy curve is still almost identical to the training accuracy curve.

Graphical user interface, text, application, email

Description automatically generated

Figure Increased Neurons from 32 to 64

To further scale up the model, the neurons of each LSTM layer was increased from 32 to 64.

Text, letter

Description automatically generated

Figure Generated Text After Increasing Neurons from 32 to 64

The generated text in the above Figure 17 became more random where even the text generated using a lower temperature starts to be more random. More non-existent words or words with spelling errors are seen in the text generated with the lowest temperature.

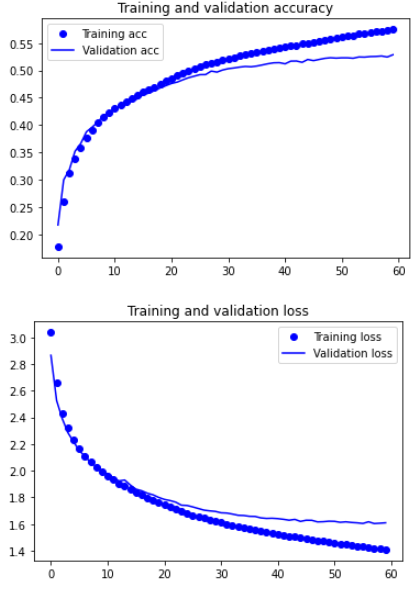


Figure Model Performance Curve After Increasing Neurons from 32 to 64

The model has not overfitted as inferred from Figure 18, however it is close to the turning point before overfitting occurs. The model performance increased where the validation accuracy increased from about 0.5 to about 0.53 and the validation loss decreased from slightly less than 1.8 to about 1.6.

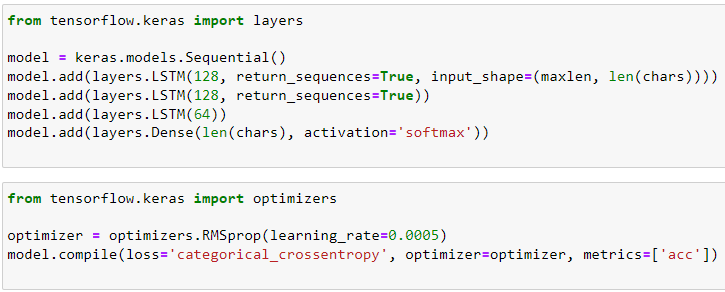


Figure Increased Number of layers to 3 and Increased Neurons

To further encourage overfitting, another LSTM layer was added to the model and the neurons were increased to 128 for two of the layers.

Text, letter

Description automatically generated

Figure Generated Text After Scaling Up a Third Time

The generated texts using a lower temperature became significantly more random and unpredictable, with varied structures and words. On the other hand, the texts generated using higher temperatures are becoming closer to actual English words instead of gibberish.

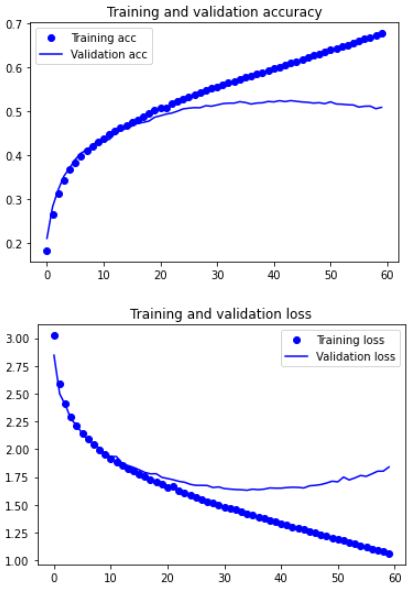


Figure Performance Curve After Third Model Scaling Up

From Figure 21, overfitting is observed to have started after about epoch 35 and the performance decreased due to overfitting.

The following step after scaling up the model until it overfits is regularizing the model.

### Regularization of the Model

The first regularization method used to overcome overfitting was adding Regularizer l2 in the layers.

Graphical user interface, text

Description automatically generated

Figure Adding Regularizer l2(0.001)

The regularizer l2 of Lambda (regularization rate) was set to 0.001 on each layer before the output layer and after the input layer.

Chart, histogram

Description automatically generated

Figure Performance Curve After Implementing l2(0.001)

Not only did adding l2(0.001) did not overcome overfitting, but it also made the model overfit more. The loss increased from previously about 1.875 to nearly 2.25.

Graphical user interface, text

Description automatically generated

Figure Changing Lambda Value of l2 from 0.001 to 0.003

The regularization rate was increased from 0.001 to 0.003 for a larger effect to overcome overfitting.

Chart, histogram

Description automatically generated

Figure Performance Curve Using l2(0.003)

Increasing the value of Lambda to 0.003 still had no impact on overcoming overfitting. Hence it was increased even more to 0.005.

Graphical user interface, text, application

Description automatically generated

Figure Regularization Rate Increased from 0.003 to 0.005

Graphical user interface

Description automatically generated with low confidence

Figure Epochs of Model Using l2(0.005)

By increasing the regularization rate to 0.005, it caused the model to underfit, therefore, it was concluded that l2 regularization was not suitable to use for this model.

L1 regularization was implemented instead of l2 subsequently.

Graphical user interface, text, application, email

Description automatically generated

Figure Implementing l1 Regularization

Knowing that l1 regularization heavily penalizes the model it was only implemented on one of the LSTM layers with a parameter of 0.001.

Text

Description automatically generated with low confidence

Figure Epochs of Model Using l1(0.001)

Seeing as how the model immediately underfitted even when l1 was implemented on one layer, it was decided that it should not be used.

The last regularization technique I used to overcome overfitting is adding dropout layers to the model.

A picture containing table

Description automatically generated

Figure Adding Dropout to the Model

Dropout was implemented to two of the LSTM layers with a dropout rate of 0.5.

Chart

Description automatically generated with low confidence

Figure Performance Curve After Using Dropout(0.5)

Dropout overcame overfitting on the first implementation, where the training and validation curves were extremely close to each other. Based on the results of utilizing dropout, there was still room for the model to be scaled up.

Graphical user interface, text

Description automatically generated

Figure Scaling Model Up Further

The model was further scaled up by adding one more LSTM layer and setting all neurons to 128.

Graphical user interface

Description automatically generated with low confidence

Figure Performance Curve of Model After Scaling Up with Dropout

Based on the results of scaling up further, there was still room for the model to be scaled up.

Text

Description automatically generated

Figure Final Scaling Up with Dropout

In this scaled up model, two layers’ neurons were increased to 256.

Chart

Description automatically generated with medium confidence

Figure Performance Curve of Final Scaling Up

In the final scaling up of the model, the dropout in the LSTM layers still overcame overfitting. Moreover, the performance of the model increased to become the best model. The validation accuracy increased to about 0.56 while the loss decreased to about 1.5 from 1.6. The validation curves are also still very close to the training curves.

Text, letter

Description automatically generatedText, letter

Description automatically generated

Figure Generated Text of Final Version of the Model

Using the final scaled up and regularized model, the generated text using a lower temperature is significantly less random and less predictable. Using higher temperatures, the texts created are also substantially clearer and closer to real words. Based on the text generated by the final and best model, it was determined that using the temperature of 0.5 is the best. It has the most balance between randomness and structure where the words generated are not repetitive, yet a large majority of them are actual English words or words from the book itself.

## Model Using Top-k Sampling

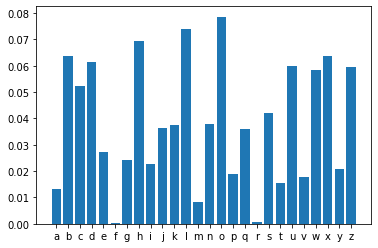


Figure Example Probability Distribution (Karakaya, Sampling in Text Generation, 2021)

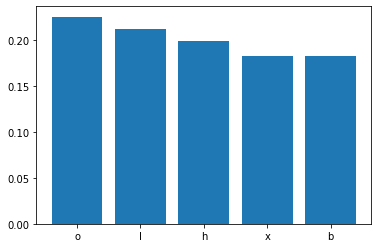


Figure Top k(5) Highest Probability Characters (Karakaya, Sampling in Text Generation, 2021)

Top-k sampling method was implemented on top of Softmax Temperature. This sampling method selects the ‘k’th high probability characters in the probability distribution. On top of this, the probabilities of characters below the ‘k’th chosen character would be changed to zero and the probability distribution would only incorporate the chosen characters. Referencing Figures 37 and 38, the value of k is 5 and the 5 characters with the highest probability are ‘o’, ‘i’, ‘h’, ‘x’ and ‘b’. (Karakaya, Sampling in Text Generation, 2021) The larger the k value, the more random the next character selection is, but it is less so compared to softmax temperature sampling.

Text

Description automatically generated

Figure Top-k Sampling Function (Karakaya, Char Level Text Generation with an LSTM Model.ipynb, 2021)

Text

Description automatically generated

Figure Top-k Text Generation Code (Karakaya, Char Level Text Generation with an LSTM Model.ipynb, 2021)

Figures 40 and 41 show the top-k sampling function and the code used to generate the text using a sample input respectively.

Text

Description automatically generated

Figure Text Generation Using Top-k

The sampling method of the final and best model was changed to Top-k sampling method to generate texts and be compared to that when using softmax temperature sampling method. All the texts generated are more random and the text generated using the k value of 2 is very similar in quality to the text generated using the temperature of 1.0. The rest of the other texts generated using a larger value of k has about the same quality as that using the temperature of 1.2 which is mostly nonsense.

Based on the texts produced by both sampling methods, softmax temperature was determined as the better method in this scenario. This could be mainly because the probability distribution is very broad and biased. Softmax temperature sampling method will hence be used for the real-life application.

# 4. Using the Developed Model to Generate Text

## Applying the Model on a Real-life Text Input

Applying the model on a text input from a user required the use of the input() function. The user would need to enter a sentence of the same character length as the value of maxlen, 60 including spaces and punctuation, which was initialized during the loading of the data. The 60-character user input was then encoded using one-hot encoding and fed into the model to generate a 400-character text based on the input given and the temperature specified.

## Analysis of The Generated Text

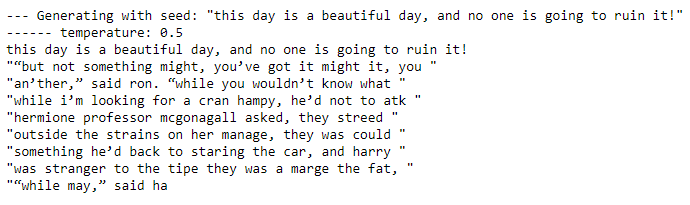


Figure The 400 Characters Generated from the User's Input

Figure 37 displays the 400-character text generated by the model using the 60-character input ‘this day is a beautiful day, and no one is going to ruin it!’. The temperature used was 0.5 which was established as the best temperature from the training of the model to generate text.

Overall, the text that was generated does not make much sense after reading it. The words generated do not follow the structure and rules of the English language, however, majority of the words generated are English words. Some of the words generated are rather creative as they are similar to the British slang which the model frequently encountered in the data text of the Harry Potter book. For example, ‘an’ther’ and ‘cran hampy’. Moreover, the model seemed to learn how to generate text after a dialogue such as ‘“an’ther,” said ron’ for example. Additionally, the use of punctuation, is very inconsistent where the full stop was used only after a sentence while commas were used in place of them. The generation of open and close inverted commas was still too often, highly due to the text data containing a large amount of dialogue, increasing the probability of these characters in the probability distribution of the model.

# 5. Summary

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Development Stages | Model Version | Model Progression | Quality of Text Generated | Highest Validation Accuracy |
| Scaling Up the Model to encourage Overfitting | Base | One LSTM layer with 32 neurons, one dense layer and learning rate of 0.0005 | Increased | 0.47 |
| 1 | Two LSTM layers with 32 neurons each | 0.50 |
| 2 | Two LSTM layers with 64 neurons each | 0.53 |
| 3 | Three LSTM layers with 128, 128 and 64 neurons respectively | 0.535  (Overfitting occurred) |
| Regularization of the Model | 4 | L2(0.001) regularization on two LSTM layers | Decreased | 0.475  (Still overfitting) |
| 5 | L2(0.003) regularization on two LSTM layers | 0.475  (Still overfitting) |
| 6 | L2(0.005) regularization on two LSTM layers | 0.1715 (Underfitted) |
| 7 | L1(0.001) regularization on one LSTM layer | 0.1715 (Underfitted) |
| 8 | Dropout rate of 0.5 implemented on two LSTM layers | Increased | 0.54  (Overcame overfitting) |
| 9 | Four LSTM layers with 128 neurons each and dropout rate of 0.5 implemented on three LSTM layers | 0.55  (Overcame overfitting) |
| 10 | Four LSTM layers with 128 neurons for two, 256 for the other two and dropout rate of 0.5 implemented on three LSTM layers | 0.56  (Overcame overfitting) |

Figure Table Summary of Model's Performance

Overall, the performance of the model was reasonably good. Throughout the process of building the text sequence generator model, the model’s performance generally went up. The validation accuracy of the model increased from the base model to the final scaled up and regularized model, from 0.47 to 0.56 respectively. The quality of the generated text also increased significantly. At the beginning, the generated text using the lowest temperature of 0.2 was exceptionally repetitive and boring. As the temperature increased, the texts became more creative, but the words became more nonsensical especially using the highest temperature of 1.2 where the text was simply strings of random characters. In the end, the words drastically made more sense using highest temperature and were considerably less random and predictable when using the lowest temperature.

The best performance of the model used four LSTM layers and a dense output layer and using the sampling method, softmax temperature. The Recurrent Neural Network layers comprised of an input layer with 128 neurons and the following layers with 256, 256 and 128 respectively.

There are many factors to be considered and changed to further improve the text generation model’s performance. One of the changes is regarding the size of the data set that the model is training on. It should be increased to expose the model to more unique words. This will indeed help tune the model’s probability distribution for each character to be more precise in character sampling to choose the next character for a given input text. Another factor, which in-terms of variety of words, the quality of data that the model is being trained on needs to improve. The data set that was used could have been not varied enough, which could be the reason why top-k sampling method did not work as well as intended. After the analysis of the generated texts, it was noticed that the model generated a lot of open and close inverted commas, which meant that there was a large proportion of the data that was dialogue. This results in a biased probability distribution which will deter the model from improving.

# References

Karakaya, M. (2021, Mar 21). *Char Level Text Generation with an LSTM Model.ipynb.* Retrieved Aug 1, 2022, from Colaboratory: https://colab.research.google.com/drive/1pFL6pHFsG6QAl0th99mIKsPaUJ84w6gN?usp=sharing#scrollTo=2H61W31sPWu\_

Karakaya, M. (2021, Mar 8). *Sampling in Text Generation*. Retrieved Aug 1, 2022, from Medium: https://medium.com/deep-learning-with-keras/sampling-in-text-generation-b2f4825e1dad