A picture containing text, clipart

Description automatically generated

**School of InfoComm Technology**

**Deep Learning Assignment**

Diploma in CSF / FI / IT

Apr 2021 Semester

**ASSIGNMENT 2**

(40% of DL Module)

5th July 2021 – 15th Aug 2021

**Submission Deadline:**

**Presentation: 9th – 13rd Aug 2021 (Week 17),**

**Report: 15th Aug 2021 (Sunday), 11:59PM**

|  |  |  |
| --- | --- | --- |
| **Tutorial Group** | **:** | **P03** |
| **Student Name** | **:** | Tan Zi Xu |
| **Student Number** | **:** | S10185168 |

**Penalty for late submission:**

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

**NO** submission will be accepted after 22nd Aug 2021 (Sunday), 11:59PM.

Contents

[Overview 3](#_Toc79953233)

[Problem 3](#_Toc79953234)

[Objective 3](#_Toc79953235)

[Approach 4](#_Toc79953236)

[Data Loading & Processing 5](#_Toc79953237)

[Loading 5](#_Toc79953238)

[Exploration 6](#_Toc79953239)

[Cleansing 6](#_Toc79953240)

[Transformation 8](#_Toc79953241)

[Develop the Sequence Generator Model 9](#_Toc79953242)

[Base Model 9](#_Toc79953243)

[Scaled Up Model 1 14](#_Toc79953244)

[Scaled Up Model 2 17](#_Toc79953245)

[Regularise Model 20](#_Toc79953246)

[Adam Optimizer Model 22](#_Toc79953247)

[Final Model – Early Stopping 24](#_Toc79953248)

[Developed Model to Generate Text 26](#_Toc79953249)

[Applying model on real life text input 26](#_Toc79953250)

[Generated Text Analysis 27](#_Toc79953251)

[Summary 28](#_Toc79953252)

# Overview

This is an individual report of problem 2 for the Deep Learning module’s assignment 2, this report is the documentation of the progression throughout this project. In this overview section, I will be covering the problem, objective and the approach for this project.

## Problem

My interpretation of the problem in this assignment problem 2 can be sum up as how can we emulate human literature with machines?

By definition, literature is a body of written works. It is traditionally applied as overarching terms for poetry or story telling works and it is deeply rooted in human language as form of lingual art. In Natural Language Processing (NLP), it is a field of study under artificial intelligence that aims to enable machines to understand and process human language. Literature being tied to human language would naturally be part of NLP reach of research as well.

As computational resources had made significant breakthrough in the last 2 decades, progression in deep learning was no longer bottlenecked by the computational limitation. Application of deep learning into the field of NLP has been increasing. Such that, deep learning has been applied to do text classification, text generation and even report summarization task. All of these applications are wonderful, and it has brought great advancement to NLP as a field of study.

We have eventually hit a point where we need to investigate into human literature for further progression in what is possible with NLP. Some such thoughts would be if artificial intelligence is capable to imitating human literature. Before thinking about imitating literature, creating human literature itself requires a level of creativity and ingenuity, it reflects a sense of what kind of person created the literature. For artificial intelligence to imitate something of that complexity is a great challenge for deep learning and NLP. Hence, I think this problem 2 in assignment 2 is how we can emulate human literature with machines.

## Objective

With a problem in sight, the objective would be to create a language model capable of creating semi-coherent English sentences at a character level.

To emulate human literature, language models serve as foundation of this text generation task. A language model will determine a character as output after being fed in inputs to complete a word or add on to the literature it is trying to write.

While it is ambitious to fully imitate a given literature, generating semi-coherent sentence would prove a point in capability of using deep learning for text generation task. In my opinion, semi-coherent sentences should constitute of minimal typos, no repeat words and some sequences of the generated text should have some contextual meaning.

Lastly, it is a character level text generation instead of word base is due to having lesser mapping computation. If the model was word based, the dataset could have its plausible inputs be mapped up to tens of thousands. Whereas character level would limit the mapping to perhaps not more than 100, saving time from computing more inputs.

## Approach

My approach would be to make a language model with a Recurrent Neural Network (RNN) model equipped with Gated Recurrent Unit (GRU) layer. My reasons for choosing RNN is due how suitable the model is for this task. While it is labelled as a character level text generation task, it is a single label, multiclass classification task instead. Because the model is predicting which of the available character should used as part of the generation. As for the GRU layer option, I am familiar on working with GRU layers from problem 1 of assignment 2. I am aware of GRU layers strength in being computationally efficient by being faster than Long Short-Term Memory (LSTM) layers. When searching on about text generation, the usage of LSTM layer was very common, so I thought of trying GRU layer in this task.

During the model development phase, I will be closely following the universal workflow of machine learning. Start with a baseline model, scale it tills overfits and regularize it to reach the ideal model configuration. However, the model development will not be strongly influence by the accuracy performance of the models. Instead, I will also be observing the text generated as part of my decision making in the model development.

For the text generation, my approach is having it generated at midway of the total training epochs as well as at the end of training. Having 2 sets of text generated allows me follow closely on how training epochs might help models generate better text. As for why I am only generating to 2 sets and not more text is due to computational resources limitation. When text is being generated, it is stored on my machine’s memory, and it only has the limited amount of 16 Gigabyte. Which pushes me to minimize how much text I should generate for analysis.

The text generated will generated with softmax temperature, which is controls the greedy sampling and stochastic sampling for choosing the character output. As the task is single label, multiclass classification, the output layer is likely to be a dense layer with set number of neurons representing the amount of possible unique character as output. The dense layer would have softmax activation, which would mean each of the unique character will be assigned a probability of being the output during prediction. At this stage, it is sampling which unique character will be the output based on probability and it can be controlled to be either greedy sampling or stochastic sampling.

Greedy sampling would be choosing characters with the highest probability, and it would often cause word repetition in text generation. As for stochastic sampling, it introduces some randomness and gives unlikely characters a better chance at being the output. This would in turn solve the word repetition issue but also increases typos and create new words that are not actual words. In order to utilize the best of both greedy and stochastic sampling, generating softmax temperature is controlling the right amount of randomness while being greedy. The softmax temperature is typically ranges from 0 to 1, values closer to 0 would be greedier while the other end of the spectrum is stochastic. For text generation, it will be generated at 3 temperatures of 0.2, 0.5 and 0.8. As I understand how the sampling works, I think it is better for me to narrow down the range of temperature my language model is exposed to for me to identify the best text generated.

When the model development phase is over, I will be evaluating the model with text outside from the dataset to see how it performs.

# Data Loading & Processing

Starting with the first step of any deep learning project, it is to extract the data and have it processed for the model development. For this assignment, I was tasked to work with the literature of a classic book by Sir Arthur Conan Doyle, “The Adventure of Sherlock Holmes”, as the dataset. The data will be extracted, go through early exploration, cleansing and transformation in this stage.

## Loading

Text

Description automatically generated

In the data loading, I have used the open function to access the text file, “holmes.txt”, and used the read function to store it in a variable called “raw\_text”. “raw-text” is a string containing the full version of the classic book, “The Adventure of Sherlock Holmes”.

## Exploration

Chart, scatter chart

Description automatically generated

Doing some exploration early to understand the data, I have found the number of characters in the data by using the len() function. The data contains 562439 characters in it, out of that there are 85 unique characters. I have used the set function for the unique character to be identified and stored in a list. The list is also sorted, grouping up punctuations, digits and alphabets respectively in a neat order. Looking at the unique characters, I felt that I would like to retain the whitespace, lower cased alphabets, digits and punctuations in my data. To do that, I will need to do data cleansing and I have prepared a function for it.

## Cleansing

Text

Description automatically generated with medium confidence

To explain what the defined function above does, it first has the text be in lower casing. I would then replace escape sequences with whitespaces. Next, I had set up a list of punctuations I want to keep, as well as the lower-case alphabets and digits from 0 to 9 as strings. I have also prepared an empty list to store the characters I want. The following would be a nested for loop going through each character checking if it is 1 of any punctuations, alphabets or digits I want to keep. If it is to be kept, I will be appended to the empty list from before else, it will be ignored and move on to the next character. This nested for loop will repeating until each character in the text has gone through the loop. With a list of characters that is to be kept, I have used the join function to have it turn backed into a string. However, the string still contains additional whitespaces. Hence, the last line is another join function, but it has the split function on the text to keep only 1 whitespace after each word before returning it as cleansed text.

Text

Description automatically generated

After running through the defined clean function with raw\_text, it had returned the cleansed text and we can see it was successful. The cleansed text is all lowered case, no more escape sequences and no more additional whitespaces between words.

Graphical user interface, text, application, Word

Description automatically generated

Looking through the cleansed text, it is found to be 559572 characters long and the number of unique characters has dropped to 50. With the text cleansed, I need to transform it into data or rather sequences to be fed into the models. In the transformation step, the idea is to have the data split into sequences and each sequence will be accompanied by a character as the possible predicted output. These sequences will be fed for the model to understand the context and it will pick up which character it should use. By understanding the context and which output should be attached to end of the sequence, the prediction process would have model be fed in with a sequence. Based on what it learned from training with multiple sequences, it will come up with a highly like character for the sequence it was just fed with.

## Transformation

Text

Description automatically generated

Referring to the transformation step, there are few lines serving different purposes. Starting with the first line, I wanted each sequence to be 200 characters long. Unlike the reviews from problem 1, the data here is just a long string that I can choose where cut and sequences in the same length. The reason for choosing 200 is that I want to have long sequences to capture enough context for the model to learn from. As for the next variable, this step is used to determine how many sequences I will have after the transformation process. As I want to gather as many sequences for the model as I can, setting step to 1 would ensure I get all the sequences I can get from the data. Following those variables, I created 2 empty lists, one to store the extracted sequences and the other for possible follow up character of the sequences. With components prepared, I used the sliding window technique to extract the sequences, their follow up character and append into their respective list. The sliding window technique basically uses for loop and steps from earlier to determine when it should cut the string as sequence. The number of sequences prepared was 559372. Lastly, these sequences need to be encoded for the model as it works with tensors instead of strings of sequences. I have decided that one hot encoding is suitable at this stage for encoding. As there is only 50 unique characters to map with, label encoding would be too tedious and word embedding might be an overkill. To one hot encode the sequences into tensors, I first need to prepare a dictionary of the unique character mapping. I will then utilize numpy library and some manipulation to have the sequences encoded as arrays or tensors.

# Develop the Sequence Generator Model

With the data loaded in and preprocessed into sequences, I will build RNN models consisting of GRU and dense layers. The models will be fed in sequences as training, text of certain temperature will be generated in the midpoint and end of the training. To enable the model to do prediction, I have prepared a function to do sampling with softmax temperature, where it will choose the character to be the output.

Text

Description automatically generated

The function above works by taking in prediction results and the temperature it should generated at. Based on the temperature, the probability of each possible character will be adjusted, the function will do more probabilistic calculation before returning the output character. This function is a key component for our text generation as it determines the character to be printed and it is specified in the assignment specification to have the model generate 400 characters. I will essentially have models creating text of 600 characters long by understanding one third of the text first and generate the remaining two thirds of it.

As a general reminder, the nature of this task is not having the highest accuracy model but rather have model generate text of quality that I am satisfied with. This is not to say accuracy would be meaningless because is still great indicators of model performance.

All in all, model development will take in the accuracy performance of the model as well as text generated into consideration when making decisions.

## Base Model

As I am following the universal workflow of machine learning, it is best that I start with a simple model as baseline, The RNN model that I will be building would only consist of a GRU layer and dense layer. This type of set up would likely remain the same for remainder of the model development but it will be tuning the number of neurons and optimize other things as well to improve the model.

The GRU layer will serve the purpose of processing sequences, trying to understand it and pass on the outputs to the dense layer. In my opinion, I think a single GRU layer would be enough for the model. Referring to my experience in working with GRU model from before, stacking GRU layer does increase performance through raising the complexity of the model. However, it is far more effective and efficient to expand the layer width by increasing neurons instead. Because stacking layers had doubled the training time but increasing 1 neuron to 500 did not increase the time 500 folds. Weighing in how time taken increase for training via stacking or expanding layers, it is takes lesser time while providing more complexity to the model.

On the other hand, the dense layers serve as an output layer. The number of neurons it should contain should correspond with number of unique characters. Using softmax activation for its probabilistic nature, the predicted output would be reflected with the highest probability at this dense layer.

Table

Description automatically generated

Preparing my baseline model, I named a sequential model as generator, gave it a GRU layer with 5 neurons along with the input shape of sequence length by number of unique character and dense layer of configuration mentioned before.

The GRU layer could have start out with lower neurons than 5 but I felt it is sufficient for the model to be considered relatively simple. The input shape is meant to indicate that the tensors entering are of that shape. The GRU would be accepting tensors of 200 representing the length of each sequence and the 50 as the unique number character is what the follow up character could be.

The dense layers have 50 neurons with softmax activation. The 50 neurons will represent the each of the possible unique character as output. The softmax activation enables single label, multiclass classification since it assigns probability to each possible output.

In the compilation, the loss is categorical cross entropy, the optimizer used is RMSProp and accuracy will be the metric.

RMSProp is one of the many optimizers available in the Tensorflow library and optimizers determine how loss is calculated in training for model. My reasons for choosing RMSProp is duet its popularity as an effective optimizer that performs well from my lecturer and peers in the module alike.

The categorical cross entropy was configured as the loss due to the task being a single label, multiclass classification task by nature. It is picked since it was neither a binary classification nor was the model working with sparse data.

Needless to say, the classification task forces the compilation to have accuracy as the metric. Since the unit of measure for performance of classification prediction uses accuracy instead of mean absolute error for regression task.

Text

Description automatically generated

The code above looks a bit intimidating but a straight forward summary is that the text generating code has been integrated with fitting process. As I wanted text to be generated midway during the training, it is necessary to have both codes to integrate as one.

Running a for loop of 20, my model will be trained 20 epochs but an epoch at a time and it is fundamentally to pause the training after certain rounds. In each epoch, the model will be fitted with the data in batch size of 512 and validation split of 0.2. The reason for choosing high batch size of 512 is to keep with the number sequences there are as well as ensuring the models weight does not get updated too often. By choosing 512 batch sizes to 559372 sequences, the ratio of data being read is about 1:1000 in each sample of batch. Any batch size smaller would take a longer time to train. Having big batch size reduces the amount of time the weights in the model get updated, ensuring the training does not have too much variance in it. The validation split of 0.2 indicates that 20% of the training data will be used for validation. As validation data is needed to observe model performance fairly during training.

The model training history will be recorded into lists epoch by epoch to be use for charting out the model performance. After recording the history, I set a condition where if the model is at the midpoint and end of the training, it would do text generation. It is accomplished by using modulus and ensuring the calculation would be 0 for the condition to trigger.

In the text generation code, it starts by randomly selecting a seed for the model to generate on. The seed could be any sequence from the dataset, and it will start generating text based on temperature specified. Using another for loop on the 3 temperature I want, the seed will become a target sequence. For 400 characters to be generated, the target sequence will be encoded, fed into model for predictions and it will return a set of probabilities that will be fed into the sampling function from before. Retrieving the out character, it will be printed out, attached to target sequence and have this process repeated with the updated sequence till 400 characters are generated. The text generation and printing code utilizes my machine’s system, which in turn cause it to consume its memory to store these generated characters. Hence, the code is too computationally resource costly for more text to be generated and 2 sets per model is probably my limit as well.

Moving on, I will now review the text generated at the 10th and 20th epoch by the baseline model.

Text, letter

Description automatically generated

In the image above, we can see the text generated for temperature 0.2, 0.5 and 0.8 after the 10th epoch. In temperature 0.2, there is an absurd repetition of the word, “the”, in the text generated after the seed. While the model can generate complete words, it is all repetition of what it had generated, providing to contextual meaning to text. This behaviour is also observed in temperature 0.5. Although the text generated that also includes element of randomness enough to create new words while repeating simple word like “the”. As for the text generated in 0.8, it is just gibberish. There is not cohesiveness in text, typos and creation of new words are very common.

Text, letter

Description automatically generated

As for the text generated in the 20th epoch, the model reflected the similar behaviour in all 3 temperatures as it did in the 10th epoch. Closer observation does show that the model new more words to repeat in temperature 0.2 with inclusion of “a” and “and”. Overall, text generated in temperature 0.5 did show slight improvement in lesser typo but still prints repetition and 0.8 text are still gibberish.

Graphical user interface

Description automatically generated with low confidence

For me to make decision towards improving the model, I have used the recorded history of model training and made charts out of it. The charts cover the model’s accuracy, validation accuracy, loss and validation loss. Referring to the charts above, the baseline model achieved 35.41% accuracy and 35.73% validation accuracy. It seems that the model might be underfitted, where it is not performing to the best it could. Observing how the learning curved, it seems that the model’s topology simplicity is preventing the model from doing better. Relating to how it generated text, the constant repetition seems to be caused by the model being too simple as well.

The obvious course of action is to make the model more complex, and I think widening the GRU layer is the right direction. As I believe that widening GRU layers is more effective than stacking more GRU layers, I will be adding more neurons to the existing layer. This will effectively increase the model’s complexity and I expect a better performance out the new model.

## Scaled Up Model 1

Table

Description automatically generated

Referring to the image above, we can see the model’s GRU layer now has neurons that is the same amount of unique character there are. I thought since I am scaling up my model, it would be interesting to see the performance if number of neurons in GRU layer for processing the sequences matches with the number of unique characters. Hence, the model now has 50 neurons in the GRU layers to do the calculation on all the training data while retaining the same configuration for other hyperparameters.

Text, letter

Description automatically generated

In the text generated after the 10th epoch, we see an overall improvement to the quality of text generated. In temperature 0.2, it is no longer repeating a single word but rather the phrase of “he was a man” occasionally. Other than those occurrences of repetitions, there is also slight typos in some words which create new words. For temperature 0.5, there is no repetition, it looks clean with a couple of typos, and it seems cohesive on the surface. Lastly, text in temperature 0.8, the text is no long gibberish. There is a fair amount existing English words and the rest being new words. I think overall there is great improvement in the quality of the text generated but it is still not satisfactory.

Text, letter

Description automatically generated

As for the text generated in the 20th epoch, the model demonstrated great improvement at all 3 temperatures as it did in the 10th epoch. Oddly, the text generated in temperature 0.2 repeated 2 words, “the street” or just the word “street” more often. It is interesting to see text generation with lesser training repeated phrases instead of one or two words. As for the temperature 0.5, I think it is getting closer to a satisfactory level of quality. There is some cohesiveness, no repetition that I am aware of, but it stills has typos and create new words at times. Lastly, the text at temperature 0.8 did not have much change from how it did at the 10th epoch.

Chart

Description automatically generated with medium confidence

Looking at the chart, this model seems like something I am familiar with, it looks better but it suggests the model is still underfitted. The model has achieved 52.91% accuracy and 52.61% validation accuracy. Overall, it is huge improvement in more than performance, the generated text quality was significantly better.

The reason I say the model is still underfitted is due how the trajectory of the model’s learning curve indicates there is more room improvement. The training curves itself looks beautiful on its own but I think the model can reach higher heights. The generated text also speaks for itself that typos can ne reduced or even eliminated. The model is not at stage I am satisfied with.

Hence, I am intending to further scale the model till it overfits. As widening the GRU layer has produced an improvement close 20%, I think I will reuse this approach to scale the model. I also feel that I should increase the number of training epochs to see how the model performance curve out in later epochs.

## Scaled Up Model 2

Table

Description automatically generated

In this iteration of scaled up model, my intention is to purposely overfit the model. This is to allow me to see what are limits of my language model. I have taken the existing number of neurons and 5 folded it, ensuring the model overfits else I would have to do 10 folds. There will be 250 neurons in the GRU layer for this model and no other changes to the model configuration.

Text, letter

Description automatically generated

As I wanted to see how the training will curve out, the training epochs has been increased to 30. The midpoint for text generation will be at the 15th epoch and the last epoch at 30 as well.

Text

Description automatically generated

Looking at the text generated in epoch 15, I think the text were well made. Temperature 0.2 text were somewhat cohesive, no repetition and not any typos that I am aware of. As for temperature 0.5, the text is mostly well made like text in 0.2 but there were words with random letters, creating typos. Lastly, the text generates with temperature 0.8 looks great, I was expecting more randomness, but it looks quite controlled. There is still the distinct randomness in picking wrong characters.

Text, letter

Description automatically generated

Looking at the text generated after the last training epoch, I think it looks worser than the text generated after epoch 15. While the text was relatively similar for temperature 0.2 and 0.5, reading this set of text felt worser. The typo mistakes were also evident in this temperature. As for text in temperature 0.8, I also do not like it very much. Overall, I prefer the text in epoch 15.

Chart

Description automatically generated

Looking at chart, I think I know why text at 30th epoch felt worser than epoch 15. This is because the performance drop after the 15th or 16th epoch. While the accuracy was 69.96% and validation accuracy is 57.34% when the training ended, the validation accuracy peaked 58.79% at the 14th epoch. Since the performance dropped, the over quality of the text felt worse.

Referring to how the performance curved out, it is an obvious sign of overfitting. We can see as training progresses; the model performs worse against data it did not train on by a huge margin. This calls for regularization techniques to be deployed, there is either dropout or regularizers I can use. In my opinion, I think dropout is not suitable for this project. I feel that language model needs to retain as much context as possible, dropping out neuron would lose information that could be crucial. Hence, I will be favouring the use of regularizer instead of dropout. Between regularizers there is l1 and l2, but I will be using l2 regularizer. This is because the regularizing effect is much stronger with l2.

## Regularise Model

Table

Description automatically generated

As I want to regularize my model, I have decided to try using l2 regularizer and it is attached to the kernel in my configuration. The regularization is set to the default value of 0.01. Other than that, there is not other changes to the model.

Text

Description automatically generated

Looking at the text generated across the 3 temperatures, I think there is not any improvement. In fact, the quality across all 3 text looks worser than before regularizing the model. Text in 0.2 feels rigid and some signs of repetition but it is not obvious. Temperature 0.5 text seems to have some funny words and typos and 0.8 was bordering to the gibberish standard. Overall, regularizing seems to have made a negative impact on the text quality in my opinion.

Text

Description automatically generated

Oddly, the text generated here feels very random. Despite having a temperature of 0.2, the text quality had a lot of randomness in it. This observation is also in text of temperature 0.2 and 0.5. I am not liking what regularization has done to the text quality and I am considering taking a step back.

Looking at charts below, accuracy in general has fallen, achieving around 58.22% accuracy and peaking validation accuracy at 50.77% at epoch 24. I felt that regularization was the wrong move and I need to take the step backwards to before regularizing.

Chart

Description automatically generated

Seeing that regularization was not the way to go but feeling I can further improve my model, I think I can try out a different optimizer. In my experience from assignment 1, Adam optimizer has surprisingly brought improvement to model performance. I think I will try using Adam optimizer before early stopping. I doubt changing optimizer would help with reducing overfitting but there is the resort in early stopping and it does not hurt to see if Adam optimizer helps the performance.

## Adam Optimizer Model

Table

Description automatically generated

In this iteration of the model, I have removed the l2 regularizer and replace RMSProp optimizer with Adam optimizer.

Text

Description automatically generated

Looking at the overall text quality across the 3 temperatures, I think the model has regained the quality it should be at. Text from temperature 0.2 and 0.5 were semi-coherent, no repetition and I do not see any typos. I am very satisfied with the text from these 2 temperatures. In my opinion, text from temperature 0.8 is actually almost as good as the other 2 but there are still evident signs of randomness.

Text

Description automatically generated

Like the text generated after epoch 15, the text generated feel exactly the same just slight worse contextually. I am overall satisfied with text at temperature 0.2 and 0.5.

Graphical user interface, chart

Description automatically generated

Looking at the performance curve, I think this might be better than the RMSProp optimizer model. We can see this training curve is smoother and there is little to no variance but it is overfitting. Achieving 70.16% accuracy, peaking higher validation accuracy than the RMSProp model at 79.15% in epoch 14. The overfitting is still an issue but apply regularization seems to bring more negative impact. Hence, I think will be using early stopping get model to its ideal peak performance and stop training there.

## Final Model – Early Stopping

Table

Description automatically generated

From studying the history of the Adam optimizer model, it shows that training till the 8th epoch is the optimal stopping point. In my final model, this RNN model will have 1 GRU layer and 1 dense layer. The GRU layer will have 250 neurons and have input shape specified for the sequence input as tensors at (200, 50). The dense layer representing as the output layer will have 50 neurons represent each of the unique characters in the data with softmax activation. The model will be compiled with Adam optimizer, categorical cross entropy loss and accuracy metric. The fitting will be feeding data in batches of 512 for 8 training epochs and 20% of the data is reserved for validation.

Below we have the image of charts on the model performance as well as text generated after training. Seeing how the best text seems to be generate between temperature between 0.2 to 0.5, I will print text generated at that temperature range to find the best temperature for text generation with my model.

Chart, scatter chart

Description automatically generated

Looking at the model history, the model has achieved the final accuracy of 60.65% and validation accuracy of 58.25%.

Text

Description automatically generated

Analysing the text generated in the image above, the best temperature seems to be 0.4. All the text generated generally fulfilled the semi-coherent to an extent, with some have typos error. What made temperature 0.4 the best is due it having the most sense in contextual meaning in my opinion for this sequence. For evaluation, it is still wiser to observe the quality of text from temperature 0.2 to 0.5.

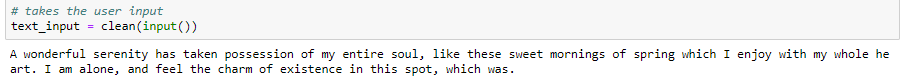


With that I saved the model as h5 file named, “CharGenerator.h5”, and proceed to evaluate the model with real life text inputs.

# Developed Model to Generate Text

In here, I will be testing my final language model against inputs outside of the sequences from training. It will be something that my model has yet to see and generate text off it. The text will be generated at temperature between 0.2 to 0.5, allowing me to analyse what would be the best temperature with the new input.

## Applying model on real life text input



In the image above, we can see the input is key in and it will go through the cleaning function. The input needs to be 200 character long due to the model only accepting sequences of 200 character in length. To assist me preparing the text, I have used a text placeholder from a website that creates generic starting lines for stories. I can set the length to be 200 and there were other styles of writing choose as well. It can be argued if this is a real-life input or not, but stance is as long as it is not a sequence the model has seen before in training, it can be used as a seed for evaluation.

Text

Description automatically generated

Following the input cleaning, I have encoded the input with one hot encoding just like the training data. I have also prepared the sampling method to return a character as prediction for text generation later.

Text

Description automatically generated

In the code above, this is a modified version of the text generating code. It essentially will be doing the same 400-character generation at the temperature range between 0.2 to 0.5. It is modified to take in the input that was cleansed and encoded for the model to do predictions and generate text.

## Generated Text Analysis

Text

Description automatically generated

The image above shows the text generated with new input. My view on the best text generated with that seed is at temperature 0.3.

Temperature 0.2 text had a slight repetition of the word “man”. Contextually, I think text generated at this temperature made no sense at all. It feels like the model was on a loop to reference the “man” repeatedly. This text definitely exhibits a strong presence of greedy sampling by the model.

Temperature at 0.3 had fulfilled the semi-coherent requirement while making some contextual sense. I think the quality of the text is clean, not any typos that I am aware of, and it felt like there is a flow to the text. This is by far the best text generated across in my opinion and the model showed a balance of greedy sampling and stochastic sampling at this temperature.

As for temperature 0.4, the quality of text is almost as good as text from temperature 0.3. However, there are clear signs of randomness which made some weird word or typo. The text could have been the best if not for temperature 0.3 doing better. The model showed slightly more stochastic sampling at this temperature.

Lastly, the text generated at temperature 0.5 is like 0.4. It had demonstrated some semi-cohesiveness, but the presence of randomness was much stronger here. It showed that the model leaned towards to stochastic sampling more.

Overall, I like what my model has produced, I think it had produced some wonderful texts. I am satisfied with the quality of text that my model has produced.

# Summary

For summarization the project progression, I have sticked to the approach I want to take for this assignment as well as followed the universal workflow of machine learning. I had created a base RNN model, improved it over the course of multiple iteration. Each iteration of model had changes done based on my decision to improve the model. The factors that attributed towards my decision making were the accuracy performance of the model as well as the quality of text generated by the model. All of which was a continuous process in improving the language model until I am satisfied with quality of the text model generated. At the end of the model development, I had created a language model that is capable of character level text generation. The final model had achieved 60.65% accuracy and 58.25% validation accuracy after 8 training epochs.

The quality of the text was satisfactory, meeting the requirement or rather the objective by my definition of semi-coherent. Where the model did not exhibit signs of word repetition even at lower temperature, demonstrating understanding of the context in the sequence building new literature on it. There were also minimal typos from adjusting randomness with temperatures. The text also managed to grasp contextual meaning and the product of the language model reflected the characteristic of contextual writing.

Lastly, there are some improvements to this assignment that I can think but I do not know if it stills align with the objective. I was thinking have additional literature from different or same author might enhance the quality of the text. Where having more data allows the model to pick finer details in that style of literature writing. Next, I was thinking of another cleansing method. I think if I knew how use regular expression properly, I could do a more stringent job at cleaning the text. I also felt that I should kept the upper casing of the alphabets into the data. It would certainly have provided more context for model to learn but it slipped of my mind during data cleansing. My last thoughts for improvement would be to try different type of RNN layers such as SimpleRNN, Conv1D, LSTM or even bidirectional layers. I think using different type of layers could yield better performance.