Sentence Autocompletion with GRU-Based RNN

Ilana Zane and William Bidle

January 6, 2023

1 Introduction

In this project, I explore generative text models in the form of charcter-level and word-level models and compare their results. An analysis of the performance of these models is conducted through experimenting with the temperature hyperparameter that is commonly used in Recurrent Neural Networks (RNN). Some of the characteristics of the generated text that are analyzed are the predicted letter frequency, the quality of the words from the generated text, and the TF-IDF score. The two models that are used are composed of GRU Time Distributed layers that are trained on the Alice In Wonderland dataset [1] as well as the New York Times Comments dataset [2].

2 The Data

Throughout this project, I decided to use two very different public datasets, the Alice in Wonderland text [1] as well as the New York Times comments section from April 2017. The project initially started with the intention of only using the NYT dataset, but I found after testing, the generative text lacked a distinct style and was hard to evaluate. This is because the NYT dataset contains comments from people all over the world and therefore lacks a distinctive tone. However, the Alice in Wonderland dataset contains a distinct tone from the author, Lewis Carroll. Therefore it was easier to objectively say whether a generated sentence made sense or not. Each dataset contains hundreds of thousands of unique words that can be used for training.

2.1 Preprocessing the Data

Before training, both datasets had to be cleaned in order to reduce unwanted characters. Punctuation marks and HTML characters that indicated new lines were removed. Once the dataset was cleaned, a tokenizer was created and then fit on the dataset.

```
tokenizer = Tokenizer(char_level=True)
tokenizer.fit_on_texts([allChars])
```

In doing so, all of text was extracted and turned into individual, character level tokens. By changing the *char_level* parameter to *False*. The dataset was able to be tokenized by words and therefore be a word-level model. Once the dataset was split into training and was turned into sequences through a window splitting method. Generally, RNN's cannot learn sequences that are much longer than 100 steps, so I set this as a constraint for splitting our data [3]. The window function takes in the following parameters:

```
n_steps = 100
window_length = n_steps+1
shift = 1
drop_remainder = True
```

Once the sequences were generated based on the window parameters, the now nested dataset was flattened into tensors, shuffled, and a mapping was created to connect the input sequences with their respective target, which was the last character in the sequence. Figure 1 demonstrates this entire process.

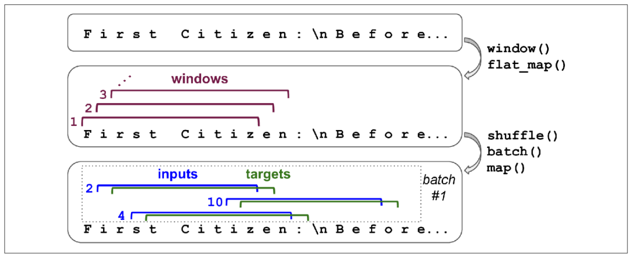

F i r s t C i t i z e n : \n B e f o r e...
window()
flat_map()
windows
3
2
1
F i r s t C i t i z e n : \n B e f o r e...
shuffle()
batch()
map()
inputs
targets
batch #1
2
10
4
F i r s t C i t i z e n : \n B e f o r e...

Figure 1: Preparing the input data[3]

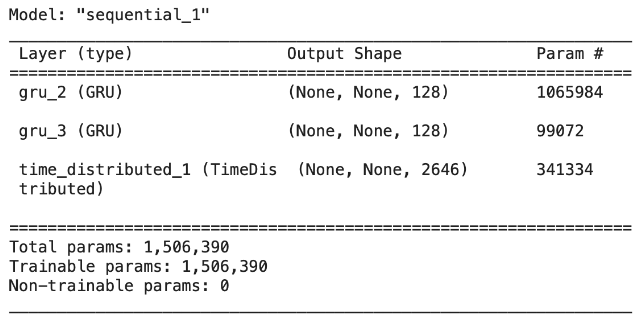
3 Model

GRU cells were used as opposed to LSTM cells in order to reduce the number of training parameters and therefore reduce memory and training time. The sequential model built using Tensorflow [4] contains three layers: two GRU cells and a TimeDistributed Layer.The GRU layers contain the following parameters:

```
return_sequences = True
dropout = 0.2
recurrent_dropout = 0.2
```

For the output of the model, the Dense layer is wrapped in TimeDistributed with a softmax activation function. The model uses sparse categorical crossentropy to measure loss during training and uses adam optimization. Training took place with 20 epochs and 10 steps per epoch.

Figure 2 shows the model that was used for the char-level and word-level networks.



```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 gru_2 (GRU)                 (None, None, 128)         1065984

 gru_3 (GRU)                 (None, None, 128)         99072

 time_distributed_1 (TimeDis  (None, None, 2646)       341334
 tributed)

=================================================================
Total params: 1,506,390
Trainable params: 1,506,390
Non-trainable params: 0
_________________________________________________________________
```

Figure 2: Model [3]

4 Results

Once the character-level model was trained, a prompt of an arbitrary length of characters (around ten characters) was fed into the model as input and the predictions for each character were generated. Fifty generated sentences were produced and put into a text file for later evaluation. Below is an example of generated text:

```
"At it was now the rabbit to her had not
one any the was not"
```

The generated predictions for the word level model followed the same procedure. Below is an example output:

```
"A explanation reaching minded inside bore
vegetable cup treated generally beautify
whisper telescopes lazily shiver"
```

From these results, we can see that at temperature = 0.2 the char-level model produces sentences that have a lot of repetition but there is some semblance of grammar, especially towards the beginning of the sentence. With the same temperature for the word-level model, the generated words are a lot more unique. However, there is no notion of grammar– the sentence seems to be devoid of structure and is just an amalgamation of words.

5 Evaluating Performance

In general, evaluating the quality of a sentence generated from an artificial source is a very complex problem and one that is still being actively researched [sources for this]. Yet, there is still quite a lot of interesting analysis that can be done on generated sentences to test their quality. In order to compare the outputs of the char-level model with the word-level model, I created a function that recursively generated letters and appended them to an ongoing sequence of generated letters. The result was an entire sentence of specified length made up of the generated letters. With this I could compare different characteristics of generated sentences from the two models.

5.1 Temperature

One of the more common parameters that show up in RNN's is temperature. The temperature parameter is important to control the diversity of the generated text– diversity is created by dividing the logits by the temperature. When the temperature is closer to 0, the model will favor high probability characters. When the temperature is closer to 1, all of the characters will have an equal chance of being generated. In the subsequent sections I will analyze the generated text based on different temperatures and determine which temperature objectively creates the best generated sentences.

5.2 Letter Frequency

One of the first things that can be noticed about almost any given text is that it generally follows a certain letter frequency distribution.[1] This means that for a given text, letters will show up some percentage of the time on average. Such a distribution for the two training data sets used in this project can be seen in Figure 3.

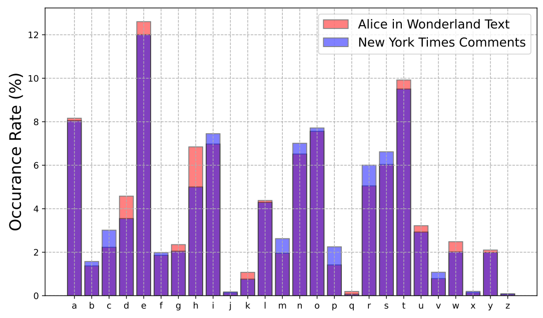


Figure 3: Occurrence rate of letters in the used in [1] and [2]. As an example, for both data sets, the letter *a* has about an 8% occurrence rate, or shows up as 8% of all letters in the document.

This then leads to one of the more simple methods of analyzing the quality of the generated sentences, by comparing the distribution of generated letters to that of the training set as the temperature is varied. Examples of these distributions for the temperatures 0.3, 0.6, and 0.9 can be seen in Figure 4

[1]Obviously texts in different languages will follow a different distribution.

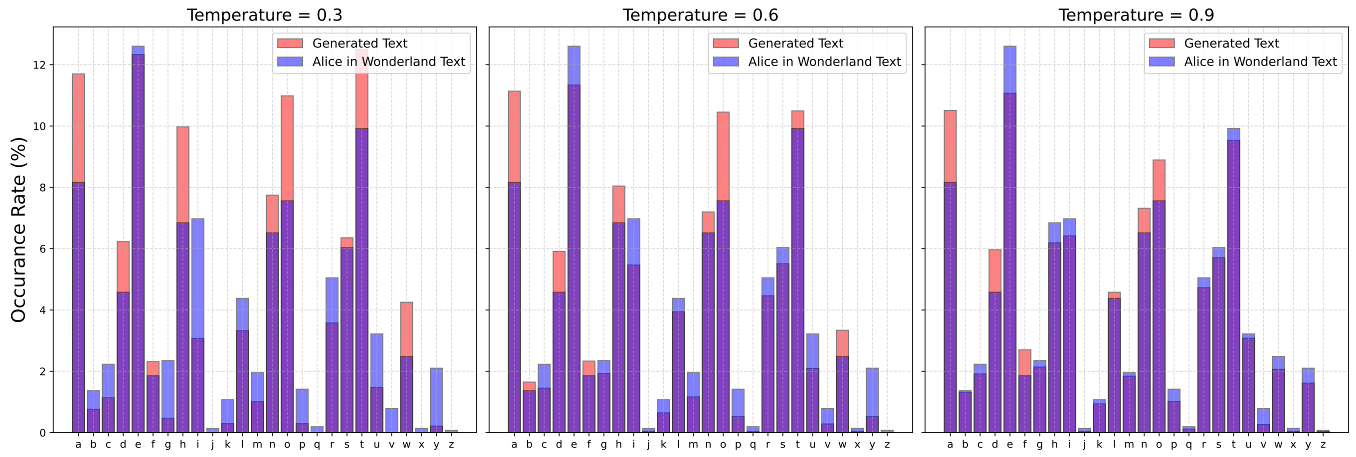


Figure 4: Occurrence rate of letters in the generated sentences (red) at different temperatures compared to that of the Alice in Wonderland training text (blue). The purple region indicates how much the two overlap, indicating the similarity between letter rates - the more the bar is purple, the closer the rate of occurrence for that letter.

As can be seen, the overlapping purple region for each letter increases as the temperature is increased, indicating that the output sentences better represent the training data's letter distribution as the temperature is increased. Taking this further, we can compute a naive *score* for the output based off of how close the average letter rate is to the training. This can be done through the sum of the absolute error between each letter frequency, as given in Equation 1.

$$Error_{LF} = \sum_{x \in \chi} |F_x - f_x| \tag{1}$$

Where the sum takes place over the alphabet under consideration, χ, and F_x and f_x represent the letter frequencies of the training set and model output, respectively. The closer the model output letter frequency distribution resembles the training set, the closer the error will be to zero. Utilizing Equation 1 on the distributions given in Figure 4 for a range of temperatures between 0.1 and 0.9 results in Figure 5 below.

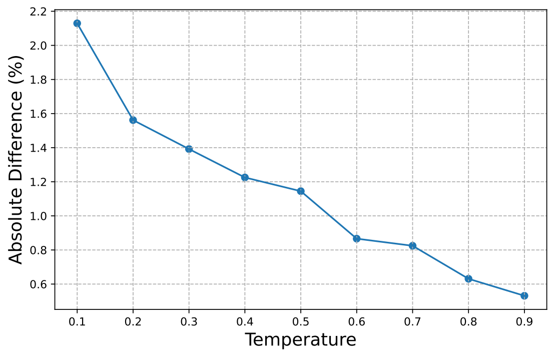


Figure 5: Absolute difference as given by Equation 1 as a function of temperature.

In other words, generated letters at a temperature of 0.1 will occur about 2.1 more/less often than they do in the Alice in Wonderland text, while generated letters at a temperature of 0.9 will occur about 0.5 more/less often. As seen in Figure 5, the deviation of the average letter occurrence rates in the generated sentences from that of the Alice in Wonderland training text tends towards zero as the temperature increases, implying a better performance for higher temperatures. Now of course this method is not perfect, as it could be very easy to fool someone into thinking the model's performance is quite good if it produces a low error. For example, a reference sentence:

```
"Hello there, how are you?"
```

and generated sentence:

```
"Ylue hhtre eow ar,o ehol?"
```

will yield an error of zero when plugged into Equation 1. Clearly the word quality in the example generated sentence is quite bad, and wouldn't fool anyone! Indeed, at a temperature of 0.9, an example of a generated sentence is:

```
"At ot!' quite tirion of notsle op of efe
fain,)that a chat meesn poard offered"
```

Whereas a temperature of 0.1 generates a sentence with almost all real words:

```
"All the was to her herself the was soon
and the was now the was "
```

Therefore it would be helpful to have an additional metric to evaluate the model's performance.

5.3 Word Quality

A second thing that can be done to evaluate the generated sentences is to look at the quality of the generated words. As mentioned in the previous section, it would be quite easy to fool someone into thinking the generated words came from the actual text by looking at the letter frequency alone. The example generated sentence from the previous section contained no real words, and naturally, it might be worthwhile to additionally look into the ratio of real words to fake words. Figure 6 shows the number of real, fake, and repeated words in the generated sentences as a function of temperature.

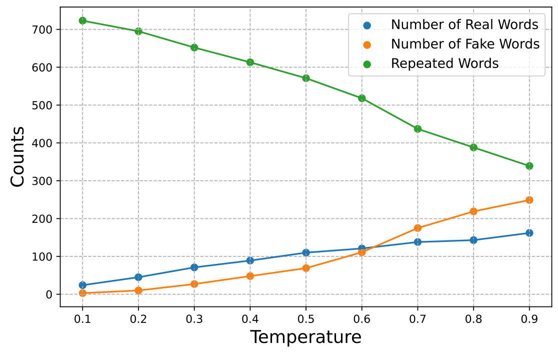


Figure 6: The number of real, fake, and repeated words generated as a function of temperature.

As seen in Figure 6, the number of real words tends to increase while the number of repeated words tends to decrease as the temperature increases, which intuitively makes sense since higher temperatures correspond to more diverse character selection, as the model will have a higher chance of selecting lower probability characters (see Section 5.1). However, there is additionally an increase in the number of fake words generated as the temperature is increased, and at around a temperature of 0.6, the number of fake and real words generated roughly approaches the same value.

Clearly we would like our generated sentences to have a higher amount of real words versus fake words, and it would additionally be good to have the number of repeated words kept to a minimum, as an average English sentence has very few repeated words per sentence. This then leads to the second way in which the error can be calculated, given by Equation 2.

$$Error_{WQ} = \frac{Fake}{Real}\frac{Repeated}{Total} \tag{2}$$

This equation perfectly captures what is desired from a good sentence. If either the number of fake words is high compared to the number of real words or the number of repeated words is high compared to the total number of words, then the error in Equation 2 will be large. Computing this error from the information in Figure 6 yields Figure 7.

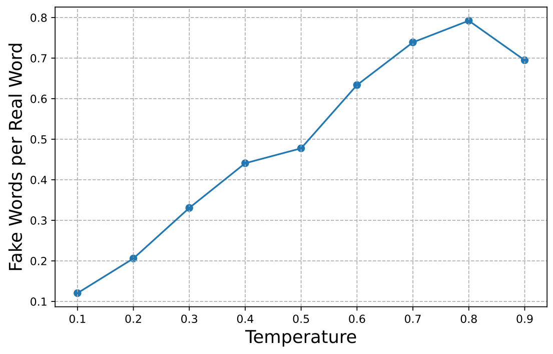


Figure 7: $Error_{WC}$ as given by 2 as a function of temperature.

As seen in Figure 7 the word quality decreases as the temperature increases, opposite to what was seen in Figure 5. This makes sense because we expect the model to take more chances as the temperature is higher, allowing for lower probability letters to be generated regardless of what came before it. Even though the number of repeated words decreases, there is a dramatic rise in the number of fake words ultimately causing the higher temperatures to have a higher error.

5.4 Bringing it Together

By multiplying Equations 1 and 2 together, I can create a way of deducing the performance of the model as a function of temperature.[2] This essentially boils down to combining the results of Figures 5 and 7. The lower the score, the better the performance of the model based on the above metrics. The results using this methodology for the Alice in Wonderland model as well as the New York Times Comments Section Model can be seen in 8.

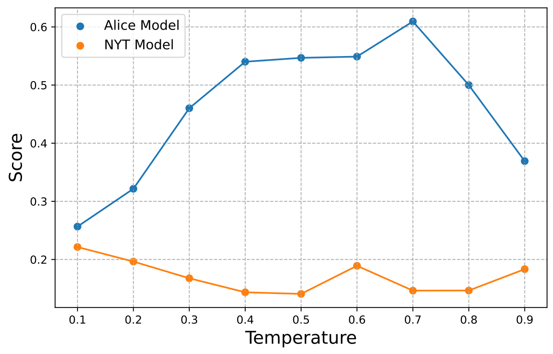


Figure 8: The final model score as a function of temperature. The closer to zero the better the model performs.

[2]The validity of this can be justified from the fact that both of the procedures from Sections 5.2 and 5.3 have clear trends as the temperature changes,and therefore the product will as well.

As seen in Figure 8, a temperature of 0.1 results in the best performance for the character level model trained on the Alice in Wonderland text, whereas a temperature of 0.5 results in the best performance for the character level model trained on the New York Times comments. Now, the same analysis can be done for the word level models, as seen in Figure 9.

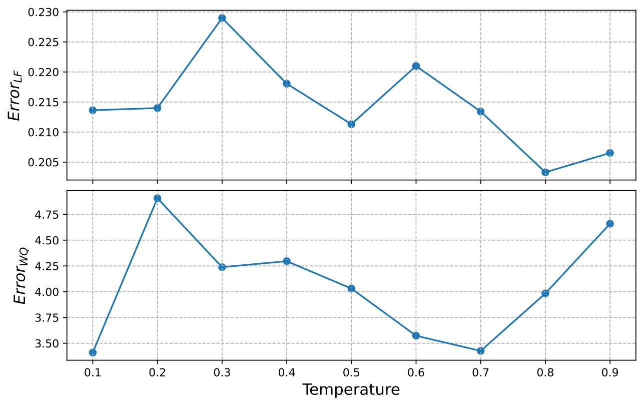


Figure 9:

It turns out that the performance of the word level model is not affected as drastically by temperature as is the character level model.

6 Conclusion

Some alternatives to the GRU-based RNN that is used in this project are: LSTM-based RNN, Markov Chains, and GAN's. These models perform well with time-series data. One issue that I encountered throughout this project is finding a balance between grammar and the generation of unique words. Markov chains are computationally cheap and relatively easy to implement, but they are defined by their 'random walk' nature. This might be an issue as the Markov chain might not tend towards the most probable solution over time and would therefore require more training or parameters to correct this [5]. LSTM-based RNN's are able to achieve similar performance to the newer GRU-based models, but since GRU cells are more simplistic, the training process is computationally more efficient than LSTM's. Given the amount of data, I thought that the GRU cells would still be an appropriate choice. GAN's may also be an effective approach to generating text, howerever this is a far more complicated task– generating adversarial text is a non-trivial task.

As for the analysis section of this project, there are other possibilites for measuring the quality of the generated sentences from both models. To more accurately categorize a generated sentence as 'good' or 'bad', I could have analyzed the number of verbs or nouns, to determine if there is a plausible sentence structure.

References

[1] L. Carroll, *Alice's adventures in wonderland*, 1997. [Online]. Available: `https://www.kaggle.com/datasets/roblexnana/alice-wonderland-dataset`.

[2] N. Y. Times, *New york times comments*, 2017. [Online]. Available: `https://www.kaggle.com/datasets/aashita/nyt-comments`.

[3] A. Geron, *Hands-On Machine Learning with Scikit-Learn, Keras, and Tensorflow*. O'Reilly, 2019.

[4] *TensorFlow: Large-scale machine learning on heterogeneous systems*, Software available from tensorflow.org. [Online]. Available: `https://www.tensorflow.org/`.

[5] I. Zane, *Jazz solo generation usnig long short-term memory recurrent neural networks*, 2022. [Online]. Available: `https://github.com/ilanazane/Jazz-Solo-Generation-Using-RNN`.