COEN 140 Final Project Report: Tolkien Story Generator

By

Justin Groves, Yen-Jung(Tim) Lu

Fall 2022

Introduction:

- Background:
    - The background of our project is that we want to create a story generator similar to NovelAI, which is a GPT-based model to generate story writing prose. It uses AI to generate new plotlines, settings, and characters based on the criteria chosen by the user. For our project, we are aiming to create a story generator, which is able to create a sentence based on a word input from the user. We are utilizing Lord of the Rings books as our dataset to create the project.
- Motivations:
    - The first motivation that triggered us to work on this project is the interest in creating story generators. Furthermore, the second motivation is our passion for the Lord of the Rings. The Lord of the Rings is one of our favorite book series of all time. We love the story, the characters, and the world that J.R.R. Tolkien created. We have seen the movies multiple times and read the books. Thus, this is why we choose to use the three books of the Lord of the Rings as our dataset.

Methodology:

- GPT2:
    - For the methodology, we are using Generative Pre-Trained Transformer 2(GPT2) as the foundation for our text generation model. GPT2 is an open-source text generation model developed by OpenAI, a research company. It is based on a

machine-learning algorithm called a recurrent neural network (RNN), which is designed to “remember” information from sentence to sentence. The GPT2 model is able to generate realistic and coherent paragraphs of text after being trained on a large corpus of text.

- Tokenization:
    - We would need to apply tokenization methods to tokenize our words into tokens. The tokenization method that we are using is the Byte-Pair Encoding(BPE), which is a data compression algorithm that is used to compress text data. The algorithm is based on the observation that most English text is composed of pairs of consecutive bytes that are either both ASCII letters or both ASCII digits. The algorithm works by replacing each pair of consecutive bytes with a single byte that represents the two bytes combined.

Alternative Models:

One alternative model we considered was using Word2Vec to generate words based on our training corpus. However, we opted to use GPT2 instead for several reasons. First, Word2Vec’s vocabulary would be entirely limited to only words used in the original corpus, meaning that it could not understand any inputted words that are outside of the vocabulary used directly in the Lord of the Rings books. Second, neither of the Word2Vec model’s main implementations, Skip-Gram or CBOW, are applicable to the problem of generating a sentence based on an inputted string. Skip-Gram gives a list of words that are expected to be found adjacent to a single inputted word, while CBOW predicts a single word based on the context of

its surrounding words. For our text generation problem, since we are adding to the end of a multi-word string, neither model would give us the result we are looking for.

Experiment:

- Dataset:
    - For the dataset, we are using the three Lord of the Rings books, and we use them as the training data. We divided the books into paragraphs, and converted these paragraphs of words into a tensor, which is a three-dimensional matrix of numerical values.
- Challenges and Modifications:
    - The first challenge we faced was the tensor operations, which requires a lot of processing power. With combining multiple iterations and more than eight thousands of paragraphs, the training step would take a large amount of time to complete. To solve this problem, we use a graphics processing unit to handle tensor operations. Also, we save the model after we generate it, which can save us time on generating the model everytime, we can just use a saved version.
- Result Evaluation:
    - We used BLEU scores and ROUGE scores to evaluate how accurate our model output was compared to the original text. In our data preprocessing, we divided our text into 90% training data and 10% testing data. Then, after the training data

was used to fine tune the GPT model, we inputted a substring of every testing data paragraph containing the first ¾ of the text, and used BLEU and ROUGE to evaluate the similarity between the generated text and the original.

- Parameters:
    - In our experiment, we tried changing three different major parameters for our model in order to find the best output. In our preprocessing phase, we tried dividing the text by paragraph (‘\n’) versus dividing the text by every 200+ word paragraph by merging shorter paragraphs together. In our training phase, we tried increasing the number of epochs, the amount of times the fine tuning algorithm uses our training data, from 5 to 10. Finally, in our generation phase we tried increasing the generated entry length limit from 30 to 50.

Results:

	Preprocessing	Training Parameters	Generation Parameters	BLEU	ROUGE-1	ROUGE-2	ROUGE-L
Test 1	divide by paragraphs	epochs = 5	entry_len = 30	7.147149838550713e-05	'r': 0.185, 'p': 0.090, 'f': 0.102	'r': 0.011 'p': 0.005 'f': 0.005	'r': 0.171 'p': 0.083 'f': 0.094
Test 2	divide by paragraphs > 200 words long	epochs = 5	entry_len = 50	3.0405782414030685e-80	'r': 0.136 'p': 0.232 'f': 0.168	'r': 0.011 'p': 0.020 'f': 0.014	'r': 0.120 'p': 0.206 'f': 0.149
Test 3	divide by paragraphs	epochs = 10	entry_len = 50	1.2941089522066173e-80	'r': 0.176 'p': 0.087 'f': 0.100	'r': 0.009 'p': 0.004 'f': 0.005	'r': 0.164 'p': 0.079 'f': 0.091
Test 4	divide by paragraphs	epochs = 10	entry_len = 30	0.000147413000793738 7	'r': 0.151 'p': 0.105 'f': 0.108	'r': 0.008 'p': 0.006 'f': 0.006	'r': 0.139 'p': 0.094 'f': 0.097

```
#compare a specific paragraph
i = 1
print(test_set['para_raw'][i])
print("~~~~~~~~~~~~~~~~~~~~~~~~~~")
print(test_set['gen_results'][i])
```

```
'Dark times,' said Strider. 'But for the present you may be left in peace, when you have got rid of us. We will leave at once.
Never mind about breakfast: a drink and a bite standing will have to do. We shall be packed in a few minutes.'
~~~~~~~~~~~~~~~~~~~~~~~~~~
'Dark times,' said Strider. 'But for the present you may be left in peace, when you have got rid of us. We will leave at once.
Never mind about breakfast: a drink and a bite standing beside us, and the Merry-Go-Round. Now, if you say so, and think it rig
ht, we will. You can have our<|endoftext|>
```

- example paragraph from our testing data: (Test 4)

```
#generate a sample sentence
text = "The party climbed the mountain"
x = generate(trained_model.to('cuda'), tokenizer, text, entry_count=1)
print(x[0])
```

```
100%|██████████████████████████████████████████████████████████████████████████████| 1/1 [00:00<00:00,  1.03it/s]
The party climbed the mountain and a long way across the mountaintop. It was far below the foothills where it came to be, but t
he ascent still took time.<|endoftext|>
```

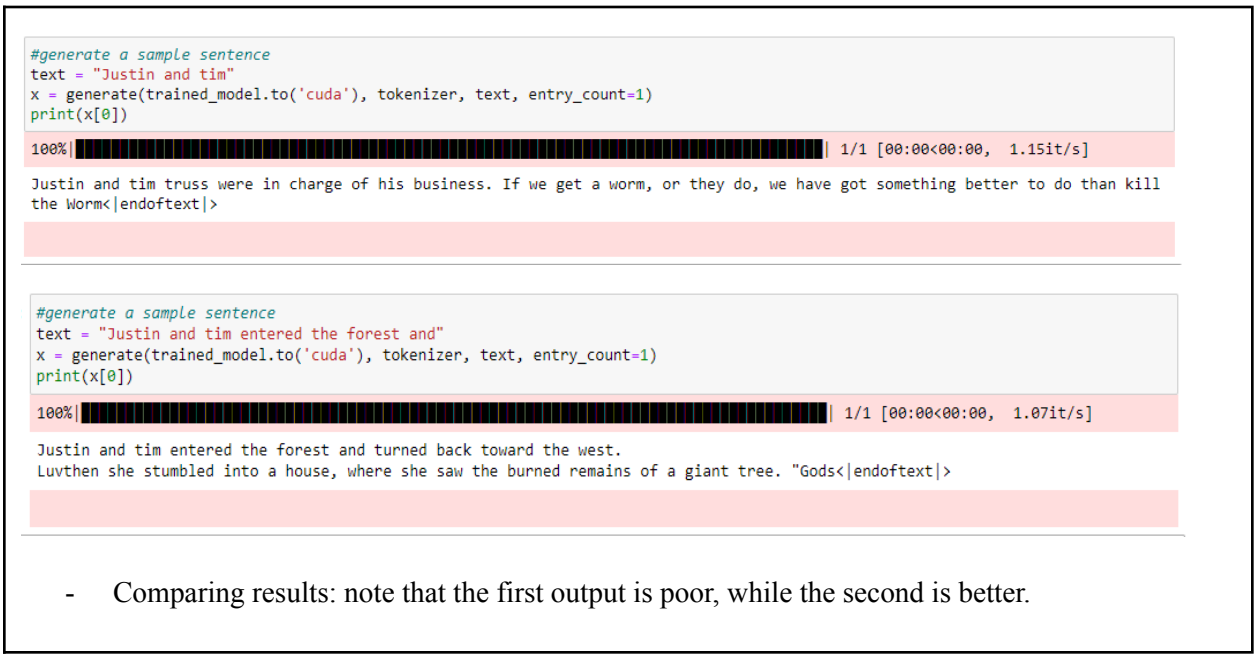
- example sentence generation with a custom text input: (Test 4)

Conclusions:

Out of the four different models that we tried, the fourth test gave us the best results for our model, with a BLEU score of 1.47*10^-4 compared to the much lower BLEU scores of tests 2 and 3, and the score 7.15*10^-5 from test 1. Tests 2 and 3 had a larger entry_len parameter, which meant that the output text could be longer and therefore have a greater variance compared to the original text, resulting in a much worse BLEU score. However, the longer text also means that the ROUGE scores were better on average, since a longer text meant that there was a greater

chance of the same words or phrases being used in the original text and the generated text. Test 2 in particular had a very high ROUGE p-score, meaning that a larger proportion of the words in the generated text were found in the original text, which makes sense because the original text blocks were larger and therefore more words would overlap.

When testing custom input, an immediate limitation that we discovered was that the trained algorithm would fail to give the proper style output if the prompt contained words that are not used in Tolkien's work. For example, entering the phrase "Justin and Tim" would give a poor output, while entering the phrase "Justin and Tim entered the forest and" would provide better results.



- Comparing results: note that the first output is poor, while the second is better.

Through this project, we learned how to create a story generator based on the GPT2 model. During the process, we also learned why the BPE tokenization method works better on our project rather than other methods. Moreover, we did an experiment on testing our model with different training parameters and divide the paragraphs with different methods. In the end, we

found out why one method is better than others, as well as how different hyperparameters and preprocessing strategies influence the result. In the future, some things we could try is modifying more of the GPT2 training and generation parameters, such as learning rate, batch size, and temperature to see if they might provide a better result.

References:

- Dayal, Divish. "How to Evaluate Text Generation Models ? Metrics for Automatic Evaluation of NLP Models." *Medium*, Towards Data Science, 7 Nov. 2020, https://towardsdatascience.com/how-to-evaluate-text-generation-models-metrics-for-automatic-evaluation-of-nlp-models-e1c251b04ec1.
- Eliaçık, Eray, et al. "What Is Novelai: Features, Pricing, and Alternatives." *Dataconomy*, 4 Oct. 2022, https://dataconomy.com/2022/10/novelai-novelaidiffusion/#NovelAI_features.
- "Implement Your Own word2vec(Skip-Gram) Model in Python." *GeeksforGeeks*, 4 Aug. 2022, https://www.geeksforgeeks.org/implement-your-own-word2vecskip-gram-model-in-python/?ref=lbp.
- St-Amant, François. "How to Fine-Tune GPT-2 for Text Generation." *Medium*, Towards Data Science, 8 May 2021, https://towardsdatascience.com/how-to-fine-tune-gpt-2-for-text-generation-ae2ea53bc272.
- *Summary of the Tokenizers*, https://huggingface.co/docs/transformers/tokenizer_summary.