End of Summer FURSCA Report

Building a Language Model using Transformer Architecture

Ilsan Kenzhebaev

Introduction:

The transformative "Transformer" architecture, introduced in 2017, has revolutionized Natural Language Processing (NLP), bestowing computers with the ability to comprehend language in a manner akin to human understanding. This research endeavor embarks on the development of a language model, leveraging the mighty Transformer to predict the next word in a sentence with precision and context-awareness, akin to OpenAI's Generatively Pretrained Transformer (GPT) model.

Results:

Step 1: Data Preparation

The foundation of this project lies in the careful preparation of the dataset. To create a robust language model, we wanted a corpus with diverse and rich linguistic content. Our choice fell upon a text dataset derived from the works of Shakespeare, a treasure trove of literary excellence. Python's versatile file handling and string manipulation capabilities proved invaluable for acquiring and preprocessing the dataset. With meticulous care, we removed any extraneous characters, special symbols, and HTML tags, ensuring that the text corpus remained pure and conducive to language modeling. The dataset was then tokenized at the character level, converting each character into a unique token. By operating at the character level, the model could grasp fine-grained details in the language, enabling a more nuanced understanding of textual patterns. Moreover, tokenizing at the character level provided the flexibility to generate not just words but also coherent sentences, extending the model's capabilities beyond traditional word-based approaches. To evaluate the model's performance effectively, we divided the dataset into a training set and a validation set using Python's data splitting libraries, such as scikit-learn's train_test_split function. The validation set acted as a reality check, enabling us to assess the model's generalization capacity to unseen data and prevent overfitting.

Step 2: The Simple Bigram

Before diving headlong into the complexities of building the GPT model, we wanted to comprehend the rudiments of language modeling. We began by implementing a simple bigram language model, laying the groundwork for subsequent advancements. The bigram language model is based on the frequency of occurrence of consecutive character pairs, or bigrams, in the training data. These bigrams act as valuable indicators of the language's inherent patterns and dynamics. By analyzing and counting the bigrams, we were able to predict the next character based on the most likely continuation of the sequence. In this step, we focused on building a robust foundation for our language model, learning to process and analyze textual data efficiently. The bigram model served as an initial reference point, enabling us to gauge the model's performance before delving into the more complex Transformer architecture.

Step 3: Pay Attention
The self-attention mechanism, the heart of the Transformer architecture, was our next milestone. Self-attention empowers the model to identify and assign different weights to various positions within a sentence, allowing it to focus on relevant information while generating the next word. To implement the self-attention mechanism, we dived into Python and NumPy, tapping into their mathematical capabilities. The process involved calculating attention scores for each character in the input sequence, indicating how important each character was in predicting the next word. This was achieved through a series of dot product operations, wherein the input sequence was multiplied with query, key, and value matrices to compute attention weights.

The self-attention mechanism revolutionized language modeling by overcoming the limitations of traditional sequential models that struggled to capture long-range dependencies. By effectively processing and weighing each character's importance within the context, the model acquired a holistic understanding of the language, facilitating more accurate and contextually relevant predictions.

Step 4: Positional Encoding
While the self-attention mechanism empowered the model to understand the relationships between characters, it lacked inherent knowledge of the characters' positions within the text. To address this limitation, we introduced positional encoding. Positional encoding provides a clever solution to incorporate positional information into the model's input embeddings. Using Python and NumPy, we generated positional encoding vectors for each character position in the input sequence. These vectors were then added to the input embeddings before passing through the self-attention mechanism. The addition of positional encoding vectors ensured that the model could differentiate characters based on their positions, thereby enhancing its ability to generate contextually relevant text. This step was pivotal in enabling the model to maintain the natural order and sequence of characters within sentences, a crucial aspect of human language understanding.

Step 5: The Transformer
Building upon the foundations of self-attention and positional encoding, we arrived at the crux of our project - the Transformer architecture. The Transformer is like an intricate symphony, harmoniously combining multiple layers of self-attention and feed-forward networks. Implementing the Transformer was an exciting challenge that required harnessing the power of

Python and PyTorch, a deep learning framework. Each layer in the Transformer comprised a self-attention module, a feed-forward neural network, and residual connections. Layer normalization was applied after each self-attention and feed-forward layer to stabilize the training process. The Transformer's multi-layered structure allowed the model to capture complex dependencies between characters, facilitating more in-depth language understanding. The inclusion of residual connections provided shortcuts for gradient flow, mitigating the vanishing gradient problem and significantly speeding up the training process. As we assembled the Transformer, our language model grew in sophistication, equipping it with the prowess to generate coherent and contextually relevant text. It was as if we were building a powerful AI architect, with each layer enhancing the model's language comprehension capabilities.

We could do only upto this step. Our future steps include:

Step 6: Model Training and Tuning
The next crucial step will be training our GPT model, which involves fine-tuning and optimizing its performance. We will use Python's versatile libraries to handle this process efficiently. Stochastic Gradient Descent (SGD) or the Adam optimizer in PyTorch will be utilized to update the model's parameters during optimization. We will implement the language modeling loss, such as cross-entropy loss, to measure the discrepancy between the predicted characters and the ground truth characters during training. Precise tuning of hyperparameters, such as learning rates, batch sizes, and the number of layers, will be crucial to achieving the best results. By leveraging PyTorch's automatic differentiation capabilities, we will efficiently compute gradients, ensuring a smooth and effective learning process. The training phase will involve multiple iterations, with each step refining the GPT model's language generation capabilities. The focus will be on unlocking the model's full potential and empowering it to generate accurate and contextually relevant text.

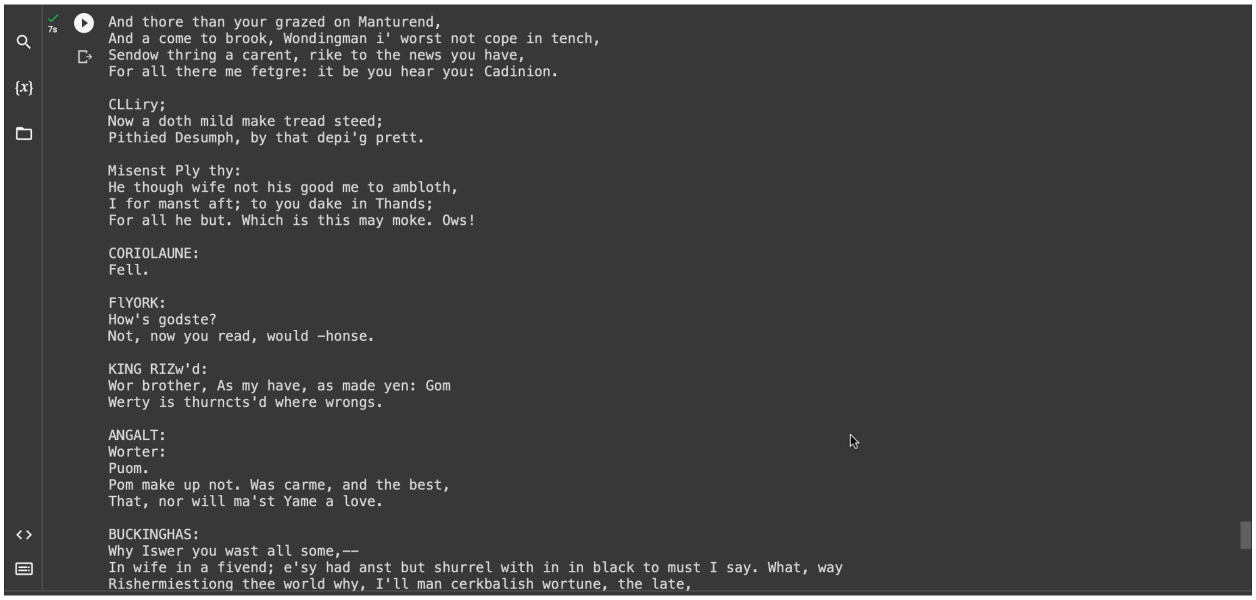
Step 7: Testing and Tweaking
With the GPT model trained and honed, it will be time to put its language generation prowess to the test. Our validation set will serve as a critical benchmark, providing a robust evaluation of the model's performance. Testing will involve generating text and meticulously evaluating its coherence, contextuality, and overall fluency. We will meticulously analyze the model's predictions and assess its ability to capture the intricacies of the English language. Based on these tests, we will iteratively refine the model, making adjustments and fine-tuning as needed.
The testing and tweaking phase will be akin to polishing a precious gem, with each refinement enhancing the model's language generation capabilities. This iterative process will aim to ensure that our GPT model surpasses its own limits and achieves optimal performance.

Conclusion:

This fall, we plan to finish up and then use what we've learned to create a chatbot. This project has been a deep dive into transformers, self-attention, and language modeling, and we've learned a ton about deep learning and natural language processing. It's set us up for even cooler AI and machine learning projects in the future.

Our sincere gratitude is extended to FURSCA, Elizabeth Palmer, Renee Kreger, and our esteemed guide, Dr. Bollman, for their unwavering support and guidance throughout this research endeavor. Their mentorship has been instrumental in transforming our ideas into reality. With the knowledge and skills gained from this project, we now embark on the path to even more ambitious AI and machine learning projects, poised to make a transformative impact on the world of NLP.

To the Jean Bengel Laughlin and Sheldon Laughlin Endowment for Student Research
Thank you! This opportunity to be a part of Albion's FURSCA program this summer was super, and I'm sure it's going to be a huge boost to my future studies and research. Thanks a ton!



```
And thore than your grazed on Manturend,
And a come to brook, Wondingman i' worst not cope in tench,
Sendow thring a carent, rike to the news you have,
For all there me fetgre: it be you hear you: Cadinion.

CLLiry;
Now a doth mild make tread steed;
Pithied Desumph, by that depi'g prett.

Misenst Ply thy:
He though wife not his good me to ambloth,
I for manst aft; to you dake in Thands;
For all he but. Which is this may moke. Ows!

CORIOLAUNE:
Fell.

FlYORK:
How's godste?
Not, now you read, would -honse.

KING RIZw'd:
Wor brother, As my have, as made yen: Gom
Werty is thurncts'd where wrongs.

ANGALT:
Worter:
Puom.
Pom make up not. Was carme, and the best,
That, nor will ma'st Yame a love.

BUCKINGHAS:
Why Iswer you wast all some,--
In wife in a fivend; e'sy had anst but shurrel with in in black to must I say. What, way
Rishermiestiong thee world why, I'll man cerkbalish wortune, the late,
```