To find the appropriate hyperparameters,

Generate from baseline (no shakespeare), generate from different 5 temperatures, generate from 5 top\_p

Baseline: language model generates text without fine-tuning. Temperature is set to 1 and Top\_p is set to 0.8

Temperatures:

Top\_p:

The train function is defined for training the GPT-2 model. It takes a dataset, model, tokenizer, batch size, number of epochs, learning rate, maximum sequence length, warmup steps, GPT-2 type, output directory, output prefix, test mode flag, and save model on each epoch flag as inputs. It trains the model using the provided dataset and hyperparameters, and saves the model weights if save\_model\_on\_epoch is True.

The generate function generates text using the trained model. It takes the model, tokenizer, prompt, number of entries to generate, maximum entry length, top-p value, and temperature as inputs. It generates text by sampling from the model's output logits and continues until the generated text reaches the maximum length or a stop token is encountered. The generated text is stored in the generated\_list and returned.

The text\_generation function generates multiple sentences given a test dataset. It takes the test data as input and iterates over each entry, generating one sentence per entry using the generate function. The generated sentences are stored in the generated\_words list and returned.

The code then reads the Shakespeare data from the "Shakespeare\_data.csv" file and preprocesses it. It drops the lines with lyrics longer than 350 tokens and creates a small test set of 200 samples.

The dataset is created using the PlayWords class with the preprocessed Shakespeare data.

The tokenizer and model are initialized using the GPT-2 pretrained weights.

The pack\_tensor function is defined to handle batching and accumulation of input tensors for GPT-2.

The model is trained using the train function, and the trained model is saved to "tensor.pt".

The text\_generation function is called to generate text for the test set, and the generated words are printed.

To demonstrate our work, we wrote code based off of GPT-2 framework for training and generating text using the GPT-2 language model. The objective of the code is to enable training a custom GPT-2 model on a user-defined dataset and subsequently generate text based on the trained model. The code encompasses essential components such as dataset creation, training function, and text generation function.

The dataset creation phase involves the definition of a custom dataset class called PlayWords. This class leverages the GPT2Tokenizer to tokenize and encode input sequences. The encoded sequences are then converted into tensors and stored in the dataset. The PlayWords class allows for optional truncation of the dataset to a specified length.

The training function, denoted as train, orchestrates the training process of the GPT-2 model. It encompasses crucial steps such as moving the model to the GPU (if available) and setting it in training mode. The function initializes an optimizer (AdamW) and a scheduler (get\_linear\_schedule\_with\_warmup) for managing the learning rate during training. It takes a dataset, model, tokenizer, batch size, number of epochs, learning rate, maximum sequence length, warmup steps, GPT-2 type, output directory, output prefix, test mode flag, and save model on each epoch flag as inputs. It trains the model using the provided dataset and hyperparameters, and saves the model weights if save\_model\_on\_epoch is True. The dataset is loaded into a DataLoader for efficient batch processing. The training loop iterates over the dataset, performing forward passes, loss computation, and backpropagation. The model's parameters are updated using the optimizer and scheduler. Additionally, the function offers an option to save the model's parameters after each epoch.

The generate function generates text using the trained model. It takes the model, tokenizer, prompt, number of entries to generate, maximum entry length, top-p value, and temperature as inputs. It is responsible for generating text using the trained GPT-2 model. It accepts inputs such as the model, tokenizer, prompt (initial text for generation), and other parameters. The function sets the model to evaluation mode and initializes variables for tracking generated entries. It employs a loop to generate the desired number of entries by iteratively predicting the next tokens based on the preceding context. The process incorporates techniques such as top-p sampling and temperature adjustment to enhance the diversity and randomness of the generated text. The generated text is then collected and returned as a list.

Moreover, the code includes a text\_generation function, which facilitates the generation of multiple sentences based on a test dataset. It iterates over the test dataset, invokes the generate function for each input text, and compiles the generated sentences into a list.

In summary, the provided code offers a comprehensive framework for training a GPT-2 language model on a custom dataset and subsequently generating text based on the trained model. Its modular structure allows researchers to experiment with different datasets, model configurations, and text generation strategies, thereby facilitating the exploration of natural language generation tasks.

Dataset Creation:

The PlayWords class is defined as a custom dataset. It takes a control\_code as input, which is a special code used to control the behavior of the language model.

The dataset is initialized with a tokenizer (GPT2Tokenizer) and an empty list called self.words to store the tokenized and encoded input sequences.

It iterates over the rows in the df['PlayerLine'] column (which should be a DataFrame containing the input text data) and tokenizes the text using the tokenizer.

The tokenized sequences are then encoded and converted to tensors, and finally appended to the self.words list.

The dataset also allows truncating the data to a specified length using the truncate parameter.

Training Function:

The train function performs the training of the GPT-2 language model.

It takes inputs such as the dataset, model, tokenizer, batch size, number of epochs, learning rate, and other parameters.

Inside the function, the model is moved to the GPU (assuming CUDA is available) and set to training mode.

It initializes an optimizer (AdamW) and a scheduler (get\_linear\_schedule\_with\_warmup) for controlling the learning rate during training.

The dataset is loaded into a DataLoader with a specified batch size.

The training loop iterates over the dataset and performs the following steps:

Packs the input tensors using the pack\_tensor function (which should be defined separately).

Moves the input tensor to the GPU and passes it through the model to obtain the outputs.

Computes the loss and performs backpropagation.

Updates the model's parameters using the optimizer and scheduler.

Resets the gradients and clears the input tensor for the next iteration.

The function also includes an option to save the model's parameters after each epoch if save\_model\_on\_epoch is set to True.

Text Generation Function:

The generate function generates text based on a trained GPT-2 model.

It takes inputs such as the model, tokenizer, prompt (initial text for generation), number of entries to generate, maximum length of each entry, top-p probability threshold, and temperature.

The function sets the model to evaluation mode and initializes variables for tracking the generated entries.

It uses a loop to generate the specified number of entries and performs the following steps for each entry:

Initializes a tensor with the encoded prompt.

Iterates until reaching the maximum entry length:

Passes the tensor through the model to obtain the logits.

Applies top-p sampling and temperature to the logits to generate the next token.

Concatenates the generated token to the tensor.

Checks if the generated token represents the end of an entry. If so, breaks the loop.

Converts the generated tensor into a list of tokens and then decodes it using the tokenizer.

Appends the generated text to the generated\_list.

The function returns a list of generated entries.

Multiple Sentence Generation Function:

The text\_generation function generates multiple sentences based on a test dataset.

It takes a test dataset (a DataFrame containing input text data) as input.

It iterates over the rows in the test dataset and generates text based on each input text using the generate function.

The generated sentences are collected and returned as a list.

Temperature and top-p sampling are techniques used during text generation to control the randomness and diversity of the generated text. Let's explore each of these techniques:

Temperature:

* In text generation, the temperature parameter is used to adjust the softmax function's output probabilities. It controls the level of randomness in the generated text.
* A higher temperature, typically above 1, increases the randomness and diversity of the generated text. It makes the model more exploratory, leading to more varied and creative outputs.
* Conversely, a lower temperature, often below 1, reduces randomness and makes the model more focused and deterministic. It generates text that is more conservative and adheres closely to the most likely predictions.
* By adjusting the temperature, researchers can tune the trade-off between randomness and coherence in the generated text, depending on their specific requirements and preferences.

Top-p Sampling (Nucleus Sampling):

* Top-p sampling is a probabilistic method used for text generation to control the diversity and relevance of the generated text.
* It involves selecting from the most probable tokens based on their cumulative probability distribution until a certain threshold (p) is reached.
* The threshold is determined dynamically, depending on the value of p. For example, if p is set to 0.8, the model selects tokens until the cumulative probability surpasses 0.8.
* This technique allows the model to focus on a subset of the most likely tokens while still maintaining diversity. It prevents the model from generating overly repetitive or nonsensical text.
* By adjusting the value of p, researchers can regulate the level of diversity in the generated text. Higher values of p result in more diverse outputs, while lower values restrict the choices to a narrower set of highly probable tokens.
* In combination, temperature and top-p sampling provide control over the trade-off between randomness and relevance in the generated text. Researchers can experiment with different values of these parameters to achieve the desired balance and produce text that aligns with their specific needs, whether it be creative, exploratory output or more focused and deterministic language generation.

1. Temperature = 1, Top\_p = 0.8:
   * This setting produces relatively coherent and meaningful text.
   * The generated text contains references to religious concepts, such as God, the Son of God, and the Holy Ghost.
   * It also mentions Antichrist and worshipping.
   * The sample seems to be influenced by religious and philosophical themes.
2. Temperature = 1.5, Top\_p = 0.8:
   * With a higher temperature, the generated text becomes more random and less coherent.
   * The text includes fragmented and nonsensical phrases, possibly resulting from the increased randomness.
   * Some words and phrases appear to be randomly combined without clear context or meaning.
   * The generated text includes references to various topics, including comedy, politics, deficits, and quotes.
3. Temperature = 2.0, Top\_p = 0.8:
   * A temperature of 2.0 further amplifies the randomness and creative elements in the generated text.
   * The text becomes even more fragmented, with words and phrases that lack clear coherence.
   * It includes a mix of different themes, ranging from personal names to locations, looting, laws, and more.
   * The generated text is highly imaginative and nonsensical, with little contextual connection.

Overall, increasing the temperature parameter leads to more random and diverse outputs, while decreasing it tends to produce more focused and coherent text. However, excessively high temperatures can result in text that lacks meaningful structure or relevance. Additionally, the top-p sampling parameter (0.8 in this case) ensures that the generated text avoids overly repetitive or nonsensical content.

1. Temperature = 1.0, Top\_p = 1.0:
   * The generated text appears to be more coherent and meaningful.
   * It includes references to personal relationships and experiences, mentioning a friend named Cathryn Robinson.
   * There are indications of personal reflection and regret, such as acknowledging the impact of base and rotten policies on one's well-being.
   * The text also briefly mentions a leadership change and a location (London road).
2. Temperature = 1.0, Top\_p = 1.2:
   * The generated text contains fragments that hint at various topics, but overall, it lacks clear coherence.
   * It mentions a dreadful model, a sovereign liege, false Laque, dangers, and present straits.
   * Some phrases and sentences seem to be cut off or incomplete, making it challenging to derive a cohesive meaning.
   * The text includes references to characters like Roger and suggests the anticipation of a threat.
3. Temperature = 1.0, Top\_p = 1.4:
   * The generated text remains fragmented and lacks clear coherence.
   * It contains references to ascologists, a sovereign liege, an ancient helmet, and a tawny robin tail.
   * The text includes phrases that may be poetic or metaphorical but lack clear context or meaning.
   * There are mentions of a house and descriptions of visual elements like a leveled slab, a massive head, and a spine.

Overall, the generated text with a temperature of 1.0 tends to be more coherent and meaningful compared to higher temperature settings. However, the samples generated with top-p values above 1.0 exhibit fragmented and incomplete phrases, making it difficult to derive clear interpretations. These samples suggest that finding the right balance of temperature and top-p value is crucial for generating text that is both coherent and diverse.