Pytorch Model

* How the Model Work:

The code trains a Bigram Language Model using the Transformer architecture.

The code can be divided into the following sections: preprocessing, model definition, model initialization and optimization, training loop, and text generation.

* 1. Reading the text data into a string.
  2. Splitting the string into individual characters.
  3. Converting the characters to integers, using a mapping from characters to integers.

Pre-processing:

* + The code begins by loading a text file (**input.txt**) that contains the training data.
  + The text is then processed to create a vocabulary of unique characters and their corresponding integer indices.
  + The input text is encoded as a sequence of integers using the created vocabulary mapping.
  + The code starts by importing the necessary libraries, such as torch for PyTorch and its modules.
  + ***Hyperparameters*** are defined, including batch\_size, block\_size, max\_iters, learning\_rate, and others. These parameters control various aspects of the model and training process

Model Definition:

* + - The code defines the components of the Bigram Language Model using the Transformer architecture.
    - The model includes the Head, **MultiHeadAttention**, **Feedforward**, and **Block classes,** which represent the building blocks of the Transformer.
    - The ***Bigram Language Model*** class combines these building blocks to define the complete language model architecture.
    - The model consists of token and position embedding tables, a sequence of Transformer blocks, layer normalization, and a linear projection for output.

Model Initialization and Optimization:

The code initializes an instance of the **Bigram Language Model** and moves it to the specified device (GPU or CPU). It also creates an optimizer (AdamW) to update the model's parameters during training.

Training Loop:

* + The code enters a training loop that iterates a specified number of times (**max\_iters**).
  + In each iteration, a batch of training data is retrieved using the **get batch** function.
  + The model is then used to compute the logits and loss for the given input sequence.
  + The optimizer's gradients are set to zero, and backpropagation is performed to compute the gradients.
  + The optimizer updates the model's parameters using the computed gradients.
  + The loss is periodically evaluated on the train and validation sets using the **estimate loss** function.

Text Generation:

* + After the training loop, the model is used to generate new text.
  + A context tensor is initialized with zeros, representing the initial input for text generation.
  + The **generate ()** method of the model is called to generate new tokens based on the given context.
  + The generated token indices are decoded back into text using the character-to-integer mapping.
  + The generated text is then printed as the final result.

*Result:*A screenshot of a computer

Description automatically generated

* During training, the code periodically evaluates the loss on the train and validation sets using the **estimate\_loss()** function and prints the results.

Training progress: The training loop ran for 5000 steps (iterations), with periodic updates on the training loss and validation loss.

Loss values: For each step, both the training loss and validation loss were reported. The training loss gradually decreased from an initial value of 4.4116 to 1.6635, while the validation loss decreased from 4.4022 to 1.8226. This suggests that the model was learning and improving over time.

* A screenshot of a computer

  Description automatically generatedAfter training, the model is used to generate new text given an initial context. The **generate()** method of the model is called, which iteratively predicts the next token based on the context. The generated indices are decoded into text using the **decode()** function and printed as the generated output.