Text Generation with Bigram Language Model

1 Introduction

In the context of the `BigramLanguageModel` class and its `generate` method, we are generating text (specifically, a sequence of characters or tokens) based on a probabilistic model that learns from previous input data. Here's a breakdown to help you understand the intuition behind what is happening during the generation process.

2 Key Concepts

2.1 Language Model

A language model is designed to predict the next token in a sequence based on the tokens that have come before it. In this case, it uses a **bigram** approach, which means it looks at the most recent token to help predict the next one.

2.2 Input and Context

The input to the `generate` method is a tensor `index`, which represents the current context. Initially, it starts as a tensor filled with zeros, indicating the initial state of the model without any prior tokens (think of it as the model starting from an empty string).

2.3 Generating New Tokens

The model generates new tokens by repeatedly predicting the next token based on the current context:

1. **Get Predictions**: It feeds the current context to the model and retrieves the logits, which contain the scores for each possible token in the vocabulary.

2. **Focus on Last Time Step**: It isolates the scores for the last token in the current context. This is because the model uses the last token to predict the next one.

3. **Apply Softmax**: It applies the softmax function to convert the logits into probabilities. This makes it easier to interpret the scores as probabilities of selecting each token.

4. **Sample from Distribution**: It samples the next token based on these probabilities. This sampling process allows for randomness in the output, making the generated text more varied and interesting.

5. **Update Context**: The newly generated token is appended to the context, allowing the model to use the updated context for the next prediction.

3 The Generation Process

Here's how the generation process works step by step:

1. **Start with a Seed**: Initially, the model starts with a blank context `context = torch.zeros((1, 1), dtype=torch.long, device=device)`. This represents that no tokens have been generated yet.

2. **Predict and Sample**: For each new token to be generated (up to `max_new_tokens`):

   (a) The model predicts what the next character should be based on the current context. Since the context is initially empty, the model might produce a token that it deems likely to start a sentence based on the training it has received.

   (b) The model then appends this token to the context.

3. **Iterate**: This process repeats, with the context continually growing. Each new prediction is made based on the most recent token, allowing the model to create a coherent string of text.

4 What Are We Generating?

- **Text Sequence**: Ultimately, we are generating a sequence of characters or tokens that could represent words, phrases, or even complete sentences, depending on how the model has been trained and the vocabulary used.

- **Diversity and Creativity**: Because the model samples from probabilities rather than selecting the most likely token every time, it can create diverse and varied outputs, making the generated text feel more human-like.

5 Example of Generation

Suppose the model has been trained on the text "Hello world." The generation might work as follows:

1. Start with an empty context.
2. Predict the first token (e.g., it might generate "H").
3. Update context to ["H"].
4. Predict the second token based on the context ["H"] (e.g., it might generate "e").
5. Update context to ["H", "e"].
6. Continue this process until it generates 500 tokens, which might end up forming a coherent sentence like "Hello world, how are you today?"

6 Conclusion

In summary, we are generating text by predicting one token at a time, using the previously generated tokens as context. This process continues until we reach the specified number of tokens to generate, creating a sequence that reflects the learned patterns of the language model.