Understanding the Bigram Language Model

An Explanation Using a 5x5 Embedding Table

1 Setup

Embedding Table: Let's assume we have a vocabulary of 5 tokens (for simplicity), represented as an embedding table:

$$\text{Embedding Table} = \begin{bmatrix} 0.1 & 0.2 & 0.3 & 0.4 & 0.5 \\ 0.2 & 0.1 & 0.4 & 0.3 & 0.6 \\ 0.3 & 0.4 & 0.2 & 0.1 & 0.7 \\ 0.4 & 0.3 & 0.1 & 0.6 & 0.2 \\ 0.5 & 0.6 & 0.7 & 0.2 & 0.1 \end{bmatrix}$$

Here, each row corresponds to a unique token, and the values in each row represent the token's embeddings in a 5-dimensional space.

2 Step 1: Input Indices

Input Indices: Let's say we have a batch of 2 sequences (i.e., $B = 2$) where each sequence contains 1 token (i.e., $T = 1$). The indices for the input tokens might be:

- Sequence 1: Token 2
- Sequence 2: Token 4

Thus, our input indices will look like this:

$$\text{Input Indices} = \begin{bmatrix} 2 \\ 4 \end{bmatrix}$$

3 Step 2: Fetching Embeddings

Getting Logits: When we pass the input indices through the embedding table, we fetch the corresponding rows from the embedding table. The resulting embeddings (or logits) will be:

$$\text{Logits} = \begin{bmatrix} 0.3 & 0.4 & 0.2 & 0.1 & 0.7 \\ 0.5 & 0.6 & 0.7 & 0.2 & 0.1 \end{bmatrix}$$

4 Step 3: Reshaping the Logits

Logits Shape: After fetching the logits, we have an output of shape (B, C), where C is the vocabulary size (5 in this case). In this step, we don't need to reshape since $T = 1$.

Thus:

- $B = 2$ (2 sequences)
- $C = 5$ (5 possible tokens)

5 Step 4: Softmax to Get Probabilities

Applying Softmax: We then apply the softmax function to the logits to convert them into probabilities. Softmax ensures that the probabilities for each token sum up to 1. The softmax function for a vector z is defined as:

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

Applying softmax to our logits yields:

$$\text{Probabilities} = \begin{bmatrix} \text{softmax}(0.3, 0.4, 0.2, 0.1, 0.7) \\ \text{softmax}(0.5, 0.6, 0.7, 0.2, 0.1) \end{bmatrix}$$

This will give us a probabilities matrix (B, C).

6 Step 5: Sampling the Next Token

Sampling: Using the 'torch.multinomial' function, we sample the next token for each sequence based on the computed probabilities. The result of this sampling will yield an index for the next token.

Suppose the resulting probabilities are:

$$\text{Probabilities} = \begin{bmatrix} 0.15 & 0.20 & 0.10 & 0.25 & 0.30 \\ 0.25 & 0.20 & 0.15 & 0.30 & 0.10 \end{bmatrix}$$

After sampling, let's say we get:

$$\text{Index Next} = \begin{bmatrix} 4 \\ 1 \end{bmatrix}$$

7 Step 6: Updating the Sequence

Appending the Sampled Token: Finally, we update our index with the newly sampled token indices. If the initial index was:

$$\text{Index} = \begin{bmatrix} 2 \\ 4 \end{bmatrix}$$

Now, after appending the new indices, we have:

$$\text{Updated Index} = \begin{bmatrix} 2 & 4 \\ 4 & 1 \end{bmatrix}$$

8 Summary of Steps

1. **Input Indices:** Identify the indices of the input tokens in the sequences.
2. **Fetch Embeddings:** Retrieve the corresponding logits from the embedding table.
3. **Softmax:** Convert logits to probabilities.
4. **Sampling:** Use the probabilities to sample the next token index for each sequence.
5. **Update Index:** Append the sampled token indices to the current context.

This process can be repeated to generate more tokens until reaching the desired length or limit.

By following these steps, you can generate new sequences of tokens based on the learned patterns from the input data!