Analysis Report

October 14, 2024

Introduction

This report presents an analysis and findings of different components of a transformer architecture. The objective is to explain everything that has been implemented in code.

1. Scaled Dot-Product Attention

This class takes the Keys, Queries, and Values which are randomly initialized vectors. The following steps are performed:

- Perform matrix multiplication between Key and Query vectors by transposing the Key.
- Divide each element by $\sqrt{d_k}$ to get the scaled attention scores for each token.
- To obtain the context vectors, we need to get the scaled attention weights by passing the scores through a softmax function and then multiply the weights with the value vector.

Why divide by $\sqrt{d_k}$?

Reason 1: Stability in Learning

The softmax function is sensitive to the magnitudes of its inputs. When the inputs are large, the differences between the exponential values of each input become much more prominent. This causes the softmax output to become peaky.

Reason 2: Stability of Variance

The dot product of Q and K increases the variance because multiplying two random numbers increases the variance. The increase in variance grows with the dimension. As dimensions increase, the variance also increases. Dividing by $\sqrt{d_k}$ keeps the variance close to 1.

2. Multihead Attention

Multihead attention has stacks of Scaled Dot-Product Causal Attentions. It splits the input embeddings into multiple heads and processes data separately in parallel.

- Causal attention is used so that the model should predict the next token by having no information about the next token. Here, the model will have access to all the previous tokens for predicting the next one, and it goes on till the sequence completion. It prevents the data leakage problem.

3. Position-Wise Feed-Forward Network

- By adding this layer, we ensure that the neural network learns things in parallel.
- Without this layer, the neural network will become a series of blocks learners, and it won't be possible for the model to share the information or the context among tokens.

4. Positional Encoding

- The PositionalEncoding class adds positional information to input embeddings and helps transformers understand token order. A matrix of size (maxlen, embeddim) stores the positional encodings. Sine is applied to even indices, and cosine to odd indices, which creates unique encodings across positions and dimensions. During the forward pass, these encodings are added to the input embeddings, allowing the model to differentiate token positions.

5. Transformer Architecture

- Input ⇒ [Embedding + Positional Encoding] ⇒ MHA ⇒ Residual Conn. ⇒ Add & Norm ⇒ Position-Wise FF ⇒ Residual Conn. ⇒ Add & Norm ⇒ N Encoder Layers ⇒ Encoder Output
- Decoder Input ⇒ [Embedding+Positional Encoding] ⇒ Masked Multi-Head Attention ⇒ Residual Conn. ⇒ Add & Norm ⇒ MHA (of Encoder Output) ⇒ Residual Conn. ⇒ Add & Norm ⇒ Position-Wise FF ⇒ Residual Conn. ⇒ Add & Norm ⇒ N Decoder Layers ⇒ Output

Challenges Faced

- The most challenging part was getting the correct shapes of the outputs while working on Multihead Attention. I tried multiple times by taking inputs manually to get the correct output before implementing it in the class.
- I faced a problem while implementing Masked Attention. I was thinking of how to get the Masked Attention matrix directly, but after some research, I handled it by creating a matrix with upper triangular values set to 0 and then to -∞. Then applied softmax to get the Masked Attention scores.

Validation Result

All test cases were passed, and output shapes were checked to ensure correctness.

Conclusion and Improvements

This assignment helped me understand each transformer component in depth. We can add different dropout methods and layer normalization strategies for improvement.