CS283: Assignment 2

Kerven Durdymyradov, KAUST ID: 187618

February 13, 2024

1 Problem 1: Transformer Questions (60 points)

Q1 (10 points):

We need it to prevent the dot product from being too large. To be more specific, if we don't scale $\sqrt{d_k}$ the dot product will be too large, and there will be too large numbers in the *softmax* function which can lead to numerical instability and degradation in the performance. d_k in here is the dimension of the key vectors. By scaling with d_k we will keep the values in a reasonable range, scaling the dot-product also helps to balance the attention scores so that they are spread across all the positions in the sequence, rather than being concentrated on a small number of positions.

Q2 (10 points):

We need it because without adding positional embedding this will not have information about the position(which is very important, especially in the sentences). I mean "machine translating" and "translating machine" will be the same. Because dot product is permutation invariant: $dotProduct(a, b) = dotProduct(b, a)$, and by adding positional embedding we want to rid of this limitation.

Q3 (10 points):

Self-attention is attending to the neighborhood. It gets the relevance of itself and to the others in the context. And gives the weight to how much relevant (importance) this word is to others (itself as well). This is used within a single layer of the Transformer. Shortly it allows the model to dynamically adjust the importance of different parts of the input sequence in computing the representation for each position. **Encoder-decoder attention** is used in the decoder, which is also attention to the encoder part as well. This enables every position within the decoder to engage with all positions within the encoder's output. The computation of attention scores in both self-attention and encoder-decoder attention involves a similar procedure, utilizing dot products among query, key, and value vectors. But the difference is for self-attention, the query, key, and value vectors are all derived from the same layer's input. But, in encoder-decoder attention, the query is from the current state of the decoder, key and value are from the output of the encoder.

Q4 (10 points):

It uses scaled-dot product attention instead of additive attention first its computational efficiency because it is just dot products and scaling. And, additive attention has more complex computations due to a feed-forward network with a hidden layer, making it less efficient. Also, scaled-dot product attention is fully differentiable, which is really good for deep learning, but in additive attention, the non-linear activation function complicates gradient calculations. Also, studies showed that scaled-dot product attention has better performance on numerous NLP tasks, most probably capturing similarities between query and key vectors by the dot product.

Q5 (10 points):

In both of them, the main goal was to learn contextual representations of words and sentences.
Bert has 2 pre-training tasks:

1 - it's masking some words in the context and trying to predict these words. (called masked language modeling)
2 - And also trying to predict whether sentence A logically follows sentence B (called next sentence prediction)
These help to understand language context and relationships between sentences.

GPT predicts the next word in a sentence given the previous words, training the model to understand and generate text based on the preceding context.

Both pre-training approaches are useful as they enable models to learn a deep understanding of language patterns, grammar, and context before being fine-tuned on specific tasks. This pre-training on large datasets in a self-supervised manner helps models to get a broad knowledge and improves the performance.

Q6 (10 points):

[CLS] token stands for - "classification", and its a special token for classification tasks, in some sense all the input information is collected to these token (I believe we can say head token) and based on this information we are doing our required tasks.
[SEP] token stands for "separator", this serves as a separator for distinct sequences within a single input, for example, in the case of a sentence-pair classification task, the two sentences would be concatenated with this token. During training, the model is trained on a large corpus of sentence-pair examples. The objective is to predict whether the two sentences in each pair are related or not. During this process, the model learns contextual relationships between the words in the sentence pairs. During evaluation, the pre-trained model can be fine-tuned on a specific task by adding a task-specific output layer and training on the task-specific data.
In short: The [CLS] token representation is used as the fixed-size representation of the input sequence for the specific task. The [SEP] token is used to separate different sequences within the same input, if necessary, for the specific task being fine-tuned for.