EECS 182/282A Deep Neural Networks Lecture 18 - 10/25/2022

Lecture 18: Transformers

Instructor: Anant Sahai *Scribe: Jianzhi Wang, Jason Yang*

1 Recap

1.1 Differences between RNN and Transformer

In both approaches, we have sequential inputs. In the RNN approach, we try to capture all the information from one state (the context token). In the transformer approach, we use the attention mechanism to learn the dependencies across the input sequence. This is because the attention mechanism allows us to query to multiple positions.

1.2 (Single-headed) Attention Block

It is helpful to think of the attention block as a "queryable softmax pooling" or soft approximation of a hash table (it contains a set of key-value pairs, where you can pass a query vector through it). You can also think of it as a differentiable black box.

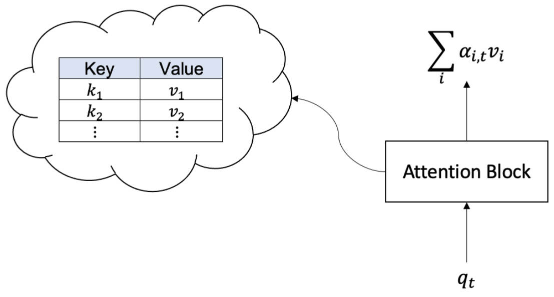

Figure 1: A single attention block

For a query, the output is approximately the value that corresponds to the nearest key, measured by inner product. Here, d is the common dimension of the key and query vector.

$$\text{sim}(q_t, k_i) = \frac{\langle q_t, k_i \rangle}{\sqrt{d}}$$

Therefore, the output is a linear combination of values: $\Sigma_i \alpha_{i,t} v_i$ where the $\alpha_{i,t} \in [0, 1]$ is obtained after a softmax operation on the similarity values. Recall that softmax allows gradient to flow to W_k, W_v and W_q. On the other hand, if argmax is used, generally there will be no gradient flows to W_k and W_q for keys that are not selected as the argmax.

1.3 Parallels with CNN

The attention block can be seen as the counterpart of a convolution layer in CNNs. In CNNs, the convolution layer plays the role of combining information from local neighbours. Here, the attention block is also aggregating information. The difference is that the attention block, unlike a convolution layer, does not have learnable parameters - they just respond to queries. We will bridge this gap in section 3.

The attention block is the counterpart to the convolution operation (aggregation of information). The table structure is similar to the concept of receptive field in CNN. However, in attention mechanism, an increase in receptive field refers to a larger number of positions in the sequence that the attention block attends to, rather than in terms of locality as in CNNs.

2 Multi-headed Attention

2.1 Introduction

Now that we know what the basic attention block looks like, we want to have the ability to attend to many things at once. We can do so with many attention heads, each equipped with its own key, value and query functions. This is the concept of multi-headed attention. Given an input token, we can query through multiple attention heads and concatenate the outputs together.

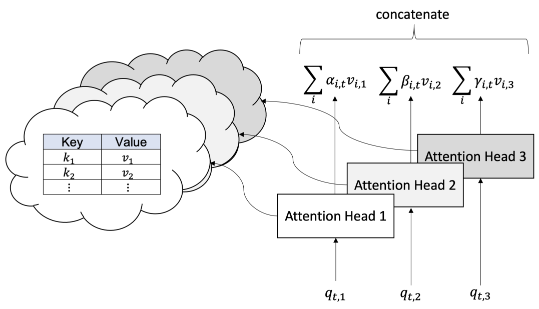

Figure 2: Multiheaded attention

Note: to ensure that the dimensions match with that of residual connection later on, we must make the dimensions of the queries smaller. This is so that when we concatenate the attention scores, we get back the same dimension as the input.

2.2 Parallels with CNN

Multi-headed attention is the counterpart of channels in CNNs. There can be many channels, but each attention head attends to its own channel. This is exactly like the depthwise convolution in ConvNext, where the convolution takes place per channel. The difference is that in CNNs, each channel is a real number, whereas here it is a vector.

Also, across different heads, the only thing that distinguishes them is the random initialization of their W_k, W_q, W_v. This is the same for different filters in CNNs, where the only thing that distinguishes them is the random initialization the weights in each filter. We hope that this broken symmetry leads to different attention heads attending to different things.

3 Motivation of the Transformer Architecture

We now present the Transformer architecture with a step-by-step walkthrough.

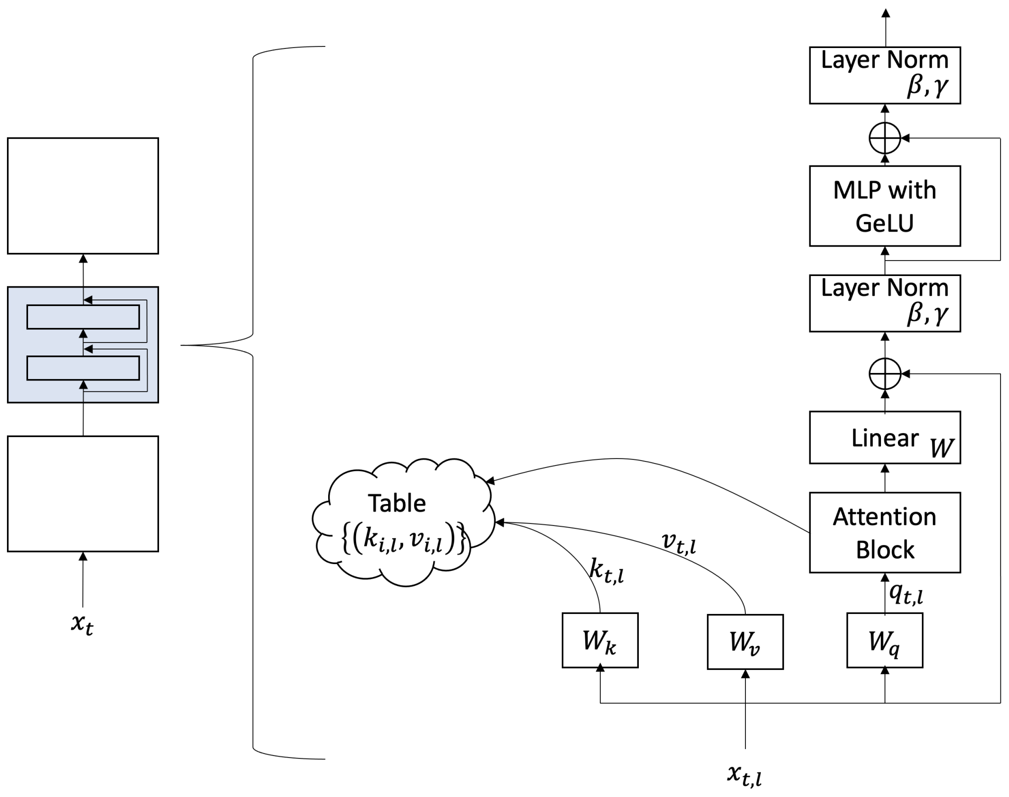

Figure 3: One layer of the Transformer architecture

1. Firstly, we receive input token $x_{t,l}$ from the previous layer. Here, t denotes time, or the position of this token in the sequence. l denotes the layer.

2. Using $x_{t,l}$, we create key vector $k_{t,l}$, query vector $q_{t,l}$ and value vector $v_{t,l}$ by applying linear layers with weights W_k, W_v and W_q respectively.

3. Store the key vector $k_{t,l}$ and value vector $v_{t,l}$ into the table. The table contains all key-value pairs generated by input sequence at layer l.

4. Pass the query vector $q_{t,l}$ as an input to the attention block (which has access to the table). Note: the attention block has a nonlinearity inside - the softmax that outputs the attention scores α.

5. The result of the attention block goes through a linear layer W. This is because the result is a linear combination of the values, and it would be more flexible to transform it.

6. This is now combined with $x_{t,l}$ via a skip connection, motivated by the ResNet architecture.
7. Now, it goes through a Layer Norm. Note that the skip connection in the previous step fixes the problem of vanishing gradients. Adding a LN now addresses the issue of exploding gradient, since of goal of Layer Norms is to control the behaviour of the output via standardization. Recall that LNs also have two learnable parameters β, γ to adjust the mean from 0 and variance from 1 so we will not lose expressive power.
8. We now add a MLP (with a nonlinearity such as GeLU) to increase the expressive power. Without it, the nonlinearity in the attention block is too weak and can be bypassed by the skip connection.
9. We use the same strategy to add a skip connection with a LN.

Previously, we have gathered a lot of ideas from the ConvNet and ResNet architectures. We now present both the ResNet and ConvNext architectures and match the ideas from these two architectures to their respective Transformer counterparts.

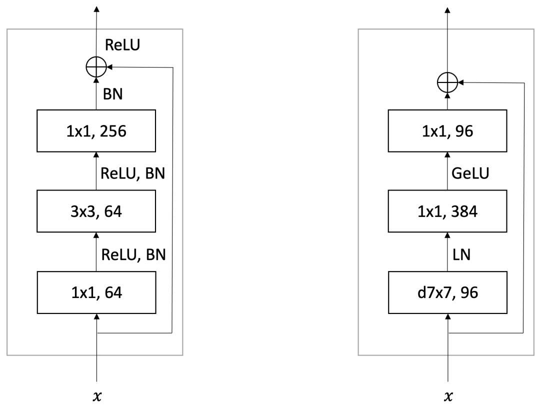

Figure 4: Classic ResNet (left) and ConvNext (right)

Similarities with ResNet:

- **Residual connections**: this was inspired by the ResNet architecture, where we want to increase complexity by adding depth and stacking multiple blocks while also ensuring a residual connection. This ensures good flow of gradients.
- **1x1 64**: Corresponds to W_k, W_q, W_v. W_k, W_v, W_q are like the filter weights and are shared in layer l across all time t, similar to how filter weights are shared in a ConvNet. All look locally at only that position t, which is similar to the 1x1 kernel.
- **3x3 64**: Corresponds to W_k, W_q, W_v and the attention block.
- **1x1 256**: Corresponds to W with the MLP + nonlinearity.

Similarities with ConvNext:

- **d7x7, 96**: Its counterpart is the entire part of the Transformer prior to the MLP, as noted in the section on Multi-headed Attention. We also see similarity in the applied LN.
- **GeLU, 1x1 96**: Exactly like the MLP + GeLU part of the Transformer.

4 Masking in Attention

Motivation: To process sequential input one token at a time, we can just populate the table one at a time in the transformer model. However, what if the entire input is available to us and we want to take advantage of the parallelism? If we naively populate the table with all key-value vectors, that would raise the issue of "peeking into the future", where we should not be able to attend to keys and values generated by future tokens.

The main idea is that we should not perform inner products with the key-value pairs from the future. This can be resolved in two ways. The first way is to surgically set the resulting value from inner products with future values to be $-\infty$ i.e. $\langle q_t, k_{t+k}\rangle = -\infty \, \forall k > 0$. This ensures that softmax assigns them a weight of 0, and their future values will never be used. The second way is to perform treatment on the αs instead. Let softmax run as per normal, then set $\alpha_{t+k} = 0 \, \forall k > 0$, then normalize the resulting αs.

5 Cross Attention

The only difference between cross attention and self attention is the tokens used to populate the table. Let's examine the encoder-decoder models.

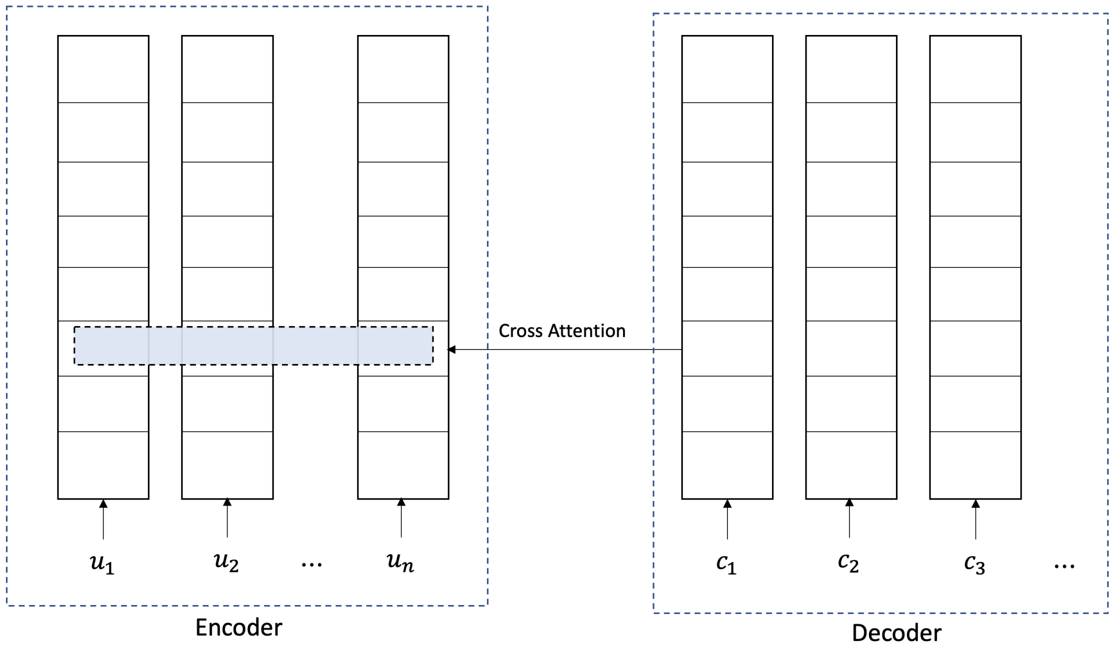

Figure 5: Decoder layers attend to key-value pairs generated by encoder layers via cross attention

The encoder part of the transformer takes in an input sequence $(u_i)_{i=1}^{n}$. Each layer will have its own table of key-value pairs. The decoder takes in a context sequence $(c_i)_{i=1}^{m}$. The attention blocks in decoder attends to the keys and values generated by the same layer in the encoder. In summary, the queries come from the decoder, and the key-value pairs are supplied by the encoder. It is also possible to use a combination of cross attention and self attention in the architecture.

6 Summary

The conception of the transformer model is not really motivated from a biological viewpoint, but rather from an engineering perspective. It is implementation friendly and good for parallelization.

However, one downside is the inherent quadratic time complexity of the attention blocks. Each attention block has to look through the entire set of key-value pairs, whose length is equal to the length of the entire input sequence. As an example, if there are N queries and M key-value pairs in table, the computation takes $O(MND)$, where D is the common dimension between the query and key/value vectors. This is because there are MN dot products, and each dot product takes $O(D)$ time. In self-attention, $M = N$, so the time complexity reduces to $O(N^2D)$, which is quadratic in N. This presents a challenge in scaling transformer models.

One way to circumvent this is via an approximate softmax attention through a kernel perspective (for example, we can approximate the softmax operation via $\text{sim}(q_i, k_j) = \phi(q_i)^T\phi(k_j))$. This approximation can effectively remove the quadratic factor and allow Transformers to scale.

Lastly, Transformers can be seen as a generalization of ConvNets. Therefore, it is a more flexible model. The weak inductive bias inherent to transformers implies that we need more training data to make learned patterns visible. This motivates the next topic on self-supervision and pre-training, both of which are absolutely critical for practical use of the Transformer model.