CS 182/282 Deep Neural Networks **Spring 2023**

Lecture 17: Transformers

Lecturer: Anant Sahai *Scribes: Samuel Pfrommer, Tanmay Gautam*

17.1 Transformers

17.1.1 Recap: Attention

In previous lectures, we were introduced to the notion of the attention mechanism, which acts as a differentiable soft hash table. Specifically, attention takes queries $\vec{q}$ as an input and returns a soft ("blended") version of the value $\vec{v}$ associated with the most similar key $\vec{k}$ to $\vec{q}$. Initially, we viewed the attention mechanism as a substitute for U-net style information flow. We also saw this concept in the context of sequence-to-sequence modeling where we discussed RNN-based encoder and decoder modules. The observed limitation with the naive RNN encoder and decoder was brought on by the bottleneck state that passes from the former to the latter. This is illustrated in Figure **??**.

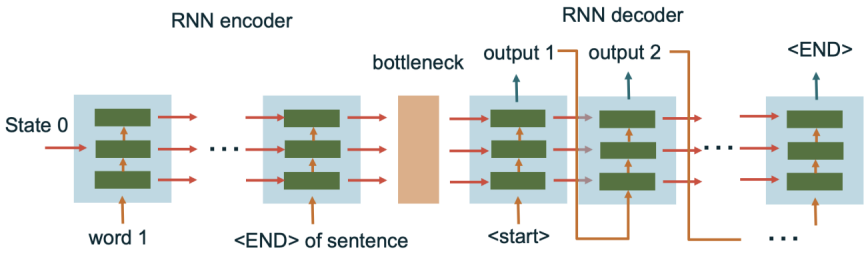

Figure 17.1: RNN Encoder-Decoder modules for sequence-to-sequence modeling **without** attention. Observe the explicit informational bottleneck between the encoder and decoder [1].

This bottleneck gives no explicit way of distinguishing contextual information from earlier in the input sequence versus later in the sequence. Augmenting the decoder with cross-attention enables the capability of looking up relevant information from different points within the encoding process. Cross-attention is implemented using learnable mappings $h_{\text{encoder}} \to \vec{k}, \vec{v}$ between the encoder states and the associated (key, value) pairs as well as learnable mappings $h_{\text{decoder}} \to \vec{q}$ between the decoder states and the queries. See Figure **??** for an illustration.

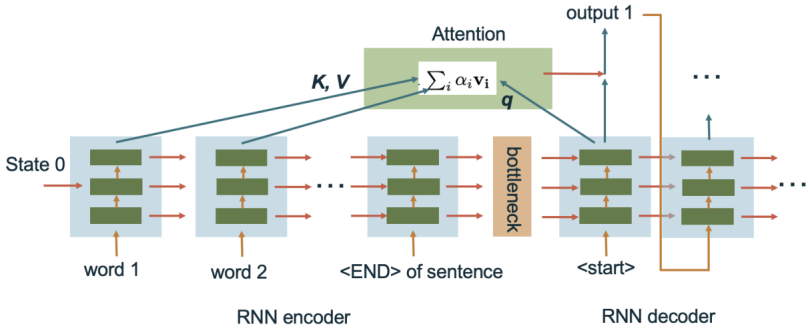

Figure 17.2: RNN Encoder-Decoder modules for sequence-to-sequence modeling **with** attention [1].

The utility of attention isn't limited to just a memory mechanism from the encoder to the decoder; rather, self-attention can also be beneficial to look up relevant information from the past within the decoder itself. This amounts to learning additional mappings between decoder states to associated (key, value) pairs and also a mapping from decoder states to query these past soft hash table entries.

Remark 17.1 *As we incorporate cross- and self-attention mechanisms into our design, we alleviate the bottleneck and burden on the information that the state passing from the encoder to the decoder needs to carry. We can also view this from the following viewpoint: initially the bottleneck was having a regularization effect due to its limiting inductive bias. By adding attention we are enhancing the underlying function expressiveness and thereby reducing the regularization.*

Remark 17.2 *Attention allows each step of the decoder to look up one item from memory. However, it is clear from natural language that this can be quite limiting. For instance, correct conjugation of verbs can rely on several bits of information: e.g. plurality, gender etc. This raises the concept of multi-headed attention. For m-headed attention, each step of the decoder performs m lookups across m attention heads. Note that these are m attention mechanisms occurring in parallel with independent queries and (key, value) pairs. The lookups are then aggregated after retrieval within the RNN state update.*

17.1.2 Global Positional Encoding

While attention provides a proxy for memory as a soft hash table, there is no explicit embedded notion of position or time. We need to find a clever way of enriching the input with positional details. This would, in turn, allow attention to encode positional information in the lookup table. This gives queries the ability to ask for either of contextual or positional information.

Consider a clock signal $\vec{c} := \begin{bmatrix} \sin(\cdot) \\ \cos(\cdot) \\ \vdots \end{bmatrix}$ that contains temporal information on the input token $\vec{i}$. There are several ways of potentially embedding $\vec{c}$ into the encoder input. We first consider global positional embedding strategies:

Choice 1: Append $\begin{bmatrix} \vec{i} \\ \vec{c} \end{bmatrix}$. This has the disadvantage of increasing the dimensionality of the encoder input which would increase the computational burden.

Choice 2: Add $\vec{i}+\vec{c}$ with $\vec{c}$ constructed to have same shape as $\vec{i}$. This has the advantage of not adding to the number of parameters needed within the RNN. Moreover, it may seem strange that we are superposing the input and temporal information. However, this works due to the incoherence between the two objects, i.e. in high dimensional spaces $\vec{i}$ and $\vec{c}$ will be approximately orthogonal. This is often what is done in practice.

We can also consider relative positional embedding. Instead of adding temporal information into the input embedding, we first think about where we want to use it. Temporal information is relevant in the lookup process of the attention mechanism. We modify the inner products during lookup:

$$\langle \vec{q}, \vec{k}_i \rangle + b_{\vec{i}-\vec{j}}$$

where $b_{\vec{i}-\vec{j}}$ is a learned bias term and $\vec{j}$ is the position of the query. To understand this, we connect it to the idea of superposition explored in global positional encoding. Suppose we can decompose $\vec{q}$ and $\vec{k}$ into a summation of informational and position components, i.e. $\vec{q} = \vec{q}_{\text{inf}} + \vec{q}_{\text{pos}}$ and $\vec{k} = \vec{k}_{\text{inf}} + \vec{k}_{\text{pos}}$. Then taking the inner product

$$\langle \vec{q}, \vec{k} \rangle = \langle \vec{q}_{\text{inf}}, \vec{k}_{\text{inf}} \rangle + \langle \vec{q}_{\text{inf}}, \vec{k}_{\text{pos}} \rangle + \langle \vec{q}_{\text{pos}}, \vec{k}_{\text{inf}} \rangle + \langle \vec{q}_{\text{pos}}, \vec{k}_{\text{pos}} \rangle,$$

where the cross-terms are approximately zero due to orthogonality in high dimensions. The positional inner product can be related to the learned bias term $b_{\vec{i}-\vec{j}}$ introduced previously.

17.2 Transformers

Now that we have an attention mechanism to lookup past information, the natural question arises of whether we need any RNN state at all [2]. The surprising answer is no: hence the name of the seminal paper "Attention is all you need" [2].

The basic framework for a transformer block is presented in Figure **??**. We walk through how this is generated below.

Figure 17.3: Schematic of a basic transformer block.

1. The input to this particular block in the transformer is $x_{t,l}$, where t is the sequence index and l is the layer. There is no "horizontal" input coming from the previous time step (outside of the shared table) as we have removed the RNN state. Gradients are thus directly routed through the query to the past time step being retrieved, avoiding any recursion and solving the dying / vanishing gradient problem.
2. Generate keys for the ith attention head, $1 \leq i \leq m$, by the linear transformation W_i^k. Generate values similarly and put them into a table specific for the lth layer in the transformer.
3. Generate m queries also using linear weights, and query the table using an attention block. For multi-headed attention, we would supply m queries and get m outputs, and we'd also put m key-value pairs into m tables in the previous step.
4. Concatenate the outputs of multi-headed attention. In order to make the residual input connection possible, we need to make the values in our table $1/m$th the size of the input.
5. Apply a layer norm, which often skips debiasing and only applies scaling to help handle gradient explosion as we add more layers.
6. We then process with a shallow MLP with one hidden layer. The idea is that we only have one nonlinearity per block (one for the multi-head attention, one for MLP).

Remark 17.3 *Transformers have a fundamentally different way of looking into the past than traditional RNN-style models that originate from signal processing ideas (infinite impulse response filters). The latter are largely motivated by considering linear operations on the past, whereas the transformer lookup is fundamentally nonlinear due to the softmax.*

Remark 17.4 *Transformers are still causal as they are only allowed to query past entries in the table, whether that's from self- or cross-attention. This leads to a practical limit on the size of the table, and we need to drop things from the far past out of the table.*

[1] X. Chen and Y. Y. Choi, "Lecture 14: Attention/self-supervision," Lecture Notes, 2022. [Online]. Available: https://inst.eecs.berkeley.edu/ cs182/sp23/assets/notes/scribe14.pdf

[2] Vaswani, Ashish, et al. "Attention is all you need." Advances in neural information processing systems 30 (2017).