EECS 182 Deep Neural Networks — Spring 2023 Anant Sahai — Discussion 10

1. Tranformers and Pretraining

Tranformer Architecture is illustrated in the schematic below.

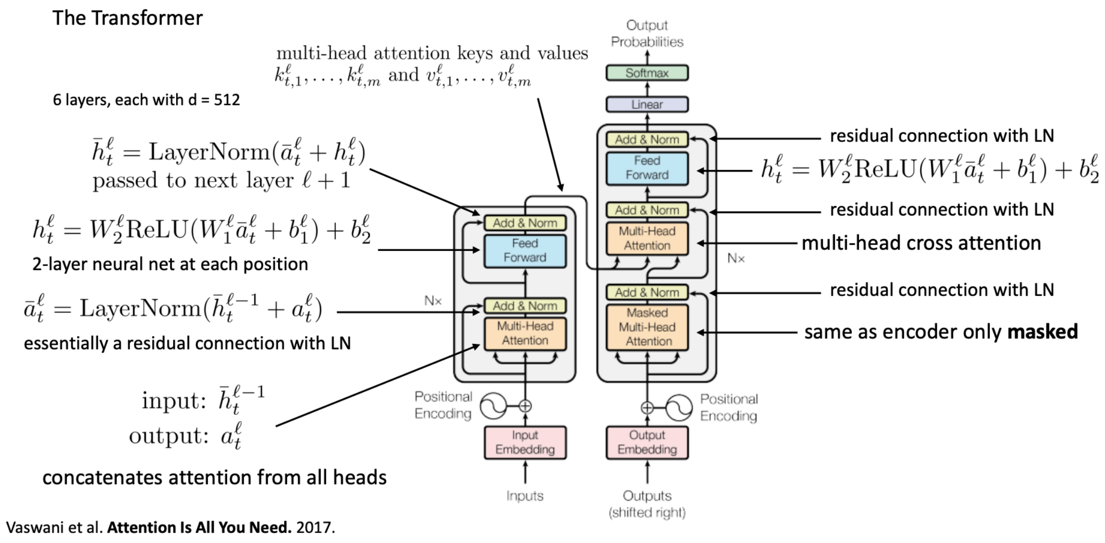
The Transformer
6 layers, each with d = 512
multi-head attention keys and values $k^\ell_{t,1}, \ldots, k^\ell_{t,m}$ and $v^\ell_{t,1}, \ldots, v^\ell_{t,m}$
$\bar{h}^\ell_t = \text{LayerNorm}(\bar{a}^\ell_t + h^\ell_t)$
passed to next layer $\ell + 1$
$h^\ell_t = W^\ell_2 \text{ReLU}(W^\ell_1 \bar{a}^\ell_t + b^\ell_1) + b^\ell_2$
2-layer neural net at each position
$\bar{a}^\ell_t = \text{LayerNorm}(\bar{h}^{\ell-1}_t + a^\ell_t)$
essentially a residual connection with LN
input: $\bar{h}^{\ell-1}_t$
output: a^ℓ_t
concatenates attention from all heads
Vaswani et al. Attention Is All You Need. 2017.
Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Multi-Head Attention
Masked Multi-Head Attention
N×
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)
residual connection with LN
$h^\ell_t = W^\ell_2 \text{ReLU}(W^\ell_1 \bar{a}^\ell_t + b^\ell_1) + b^\ell_2$
residual connection with LN
multi-head cross attention
residual connection with LN
same as encoder only masked

Figure 1: Overview of Transformer architecture

(a) **Why do we need positional encoding? Describe a situation where word order information is necessary for the task performed.**

Solution: Position encoding is used to ensure that word position is known. Because attention is applied symmetrically to all input vectors from the layer below, there is no way for the network to know which positions were filtered through to the output of the attention block.

Position encoding also allows the network to compare words (nearby position encodings have high inner product) and find nearby words. It is necessary in language translation tasks, where the order of the words affects the meaning. For example, "the man chased the dog" and "the dog chased the man" have very different meanings.

(b) When using an absolute positional encoding (e.g. sinusoids at different frequencies like hands of a clock), we can either add it to the input embedding or concatenate it. That is, if x_i is our word embedding and p_i is our position embedding, we can either use $z = x_i + p_i$ or $z = [x_i, p_i]$ Consider a simple example where the query and key for the attention layer are both simply $q = k = z$. **If we compute a dot-product of a query with another key in the attention layer, what would be the result in either case? Discuss the implications of this.**

Solution:

Let x_1, p_1 be the word embedding and position embedding of word 1, and x_2, p_2 for word 2.

If we add the embeddings to get our input, our dot-product is

$$(x_1 + p_1) \cdot (x_2 + p_2) = x_1 \cdot x_2 + x_1 \cdot p_2 + p_1 \cdot x_2 + p_1 \cdot p_2$$

If we concatenate our embeddings to get the inputs, our dot-product is

$$[x_1, p_1] \cdot [x_2, p_2] = x_1 \cdot x_2 + p_1 \cdot p_2$$

Discuss - with concatenation you avoid having the cross-terms, and thus are only comparing words to words and positions to positions. This will combine to determine what we pay attention to. With addition you also have the cross-terms, but they are typically small, at least at initialization (either the $x_1 \cdot x_2$ or $p_1 \cdot p_2$ term typically would dominate and so it doesn't matter too much) since in high-dimensional spaces, vectors tend to be orthogonal. If anything, at initialization, the position embeddings would have nontrivial dot-products since they are structured in such a way that nearby positions are not orthogonal.

(c) It turns out we can extend the self-attention mechanics to have relative positions matter without cross terms and without having to explicitly concatenate (and thereby increase the length of) two kinds of embeddings.

i. Relative position embedding explicitly adds a learnable set of biases π_{i-j} to the dot-product scores before the softmax operation. **For what $\pi_{i,j}$ would we get the same behavior from attention as concatenating the position embeddings $q_i^{\textbf{(pos)}}, k_j^{\textbf{(pos)}}$ to both the query q_i and the keys k_j?**

Solution: First note that both q_i and k_j are word embeddings (at any arbitrary layer). To have each bias vector $\pi_{i,j}$ match the results of concatenating position embedding, as you've seen in the previous part, we want $q_i k_j + \pi_{i,j} = q_i k_j + q_i^{(\text{pos})} k_j^{(\text{pos})}$.

Thus $\pi_{i,j} = q_i^{(\text{pos})} k_j^{(\text{pos})}$. A self-attention layer will have multiple of these bias terms to represent relative positions between multiple time steps.

If we complete the full query-key-value mechanics with the learnt $\pi_{i,j}$ biases, we get:

$$\alpha_{ij} = \exp\{q_i k_j + \pi_{i,j}\} = e^{q_i k_j} \cdot e^{\pi_{i,j}} \qquad \text{(pre-softmax coefficient)}$$
$$a_{ij} = \sum_{j=1}^{N} (e^{\pi_{i,j}} \cdot e^{q_i k_j}) \cdot v_i$$

where $e^{\pi_{i,j}}$ can be viewed as "relative position aware" scaling factor in self-attention!

ii. If T is the maximum context-length and we have m attention heads, **how many extra parameters have to be learned if we insist that these learned (per-head) attention biases $\pi_{i,j} = \pi_{i-j}$ must only depend on relative position?**

Solution:

This is essentially the idea of Shaw et al (2018). The modified self-attention will now contain extra parameters $\pi_{i,j}$ for every pair of query i and key j, representing their pairwise relative position info. Since we explicitly set $\pi_{i,j} = \pi_{i-j}$ meaning that $\pi_{1,4} = \pi_{2,5}$ as they both model the relative position between two tokens that are +3 timesteps away, what previously was T position embeddings included at input, is **now** $(2 \cdot (T-1)+1) \cdot m$ **embeddings** at each self-attention layer.

This design works well in applications such as machine translation, even without the absolute position embedding. In practice, the number of parameter $\pi_{i,j}$ is clipped to only model relative position information of tokens maximally k-token away from each other (i.e. there are $2k+1$ biases to learn). Check the linked paper if you want to learn more!

(d) **What is the advantage of multi-headed attention?** Give some examples of structures that can be found using multi-headed attention.

Solution: Multi-Head attention allows for a single attention module to attend to multiple parts of an input sequence. This is useful when the output is dependent on multiple inputs (such as in the case of the tense of a verb in translation). Attention heads find features like start of sentence and paragraph, subject/object relations, pronouns, etc.

(e) Let's say we're using argmax attention, which uses argmax rather than softmax, like we saw on the midterm. **What is the size of the receptive field of a node at level n...**
If we have only a single head?
If we have two heads?
If we have k heads?

Solution: With only a single head, we only have attention with one other time step (ie. the key vector), so with the residual connection in the transformer block, a branching factor of 2 at each level. Hence total size is 2^n.

With two heads, each hidden state can pay attention to itself and two other hidden states, so we have a branching factor of 3. Total size of receptive field is 3^n.

Similarly, with k heads, size of the receptive field is $(k+1)^n$

(f) For input sequences of length M and output sequences of length N, **what are the complexities of (1) Encoder Self-Attention (2) Cross Attention (3) Decoder Self-Attention.** Let k be the hidden dimension of the network.

Solution: (1) $\mathcal{O}(M^2k)$ (2) $\mathcal{O}(MNk)$ (3) $\mathcal{O}(N^2k)$

(g) **True or False: With transformer masked autoencoders, masking out a token typically involves replacing both the token value and the positional encoding at an index with a special "mask" token.**

Solution: False. Using "mask" tokens is common, but we do not mask out the positional encoding (if we did, the model would not know where the masked token belongs in the sequence. The autoencoder would produce identical representations for all masked tokens.

(h) A group of CS 182 students are creating a language model, and one student suggests that they use random text from novels for pre-training. Another student says that this is just arbitrary text isn't useful because there aren't any labels. **Who's right and why?**

Solution: Pretraining for large language models is typically self-supervised. This means that the "labels" are inferred automatically from the inputs. An example of such a scheme could be to predict the next word or fill in some masked words.

(i) **Would an encoder model or a encoder-decoder model be better suited for the following tasks?**
Summarizing text in an article
Classify written restaurant reviews by their sentiment
Identifying useful pages when retrieving web search results
Translating one language to another

Solution:
Summarizing text in an article: Seq-to-seq

Classify written restaurant reviews by their sentiment: Encoder Model
Identifying useful pages when retrieving web search results: Encoder Model
Translating one language to another: Seq-to-Seq

Contributors:

- Kevin Li.
- Anant Sahai.
- Olivia Watkins.
- Jerome Quenum.
- Saagar Sanghavi.
- CS 182/282A Staff from previous semesters.