EECS 182 Deep Neural Networks
Fall 2022 Anant Sahai

Discussion 9

1. Tranformers and Pretraining

Tranformer Architecture is illustrated in the schematic below.

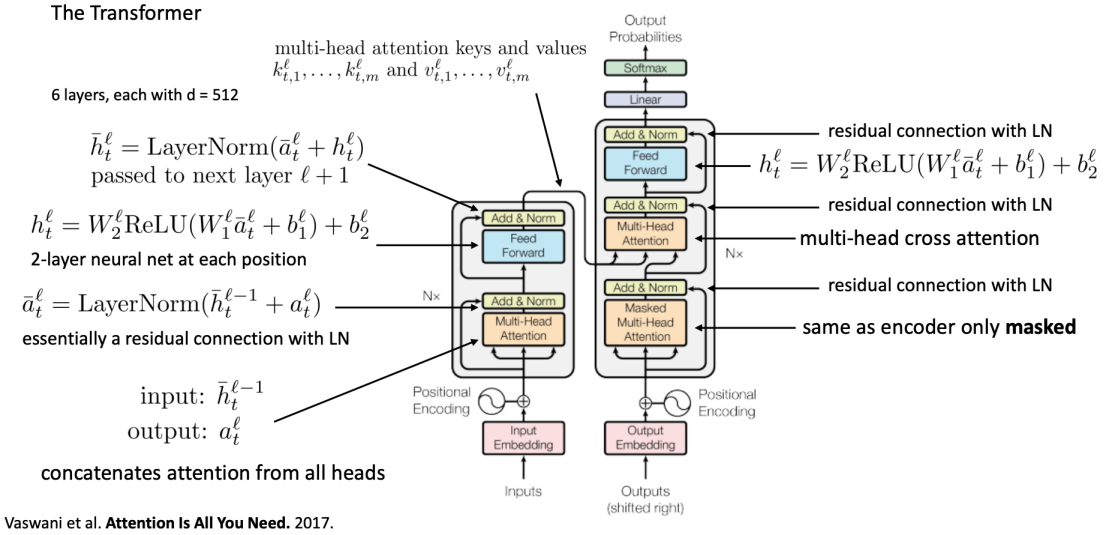

Figure 1: Overview of Transformer architecture

(a) Why do we need positional encoding? Describe a situation where positional encoding 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 the positional encoding, 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 input $z = [x_i, p_i]$ 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 input embedding, 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 input embedding, 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. One of the terms will dominate and that will affect what we pay attention to later. With addition you have the cross-terms, but they are typically small (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)

(c) 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.

(d) 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 self-attention and attention with one other hidden state, so 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 $(k+1)^n$

(e) 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) $\mathcal{O}(N^2k)$

(f) 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.

(g) 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.

(h) Would an encoder model or a seq-to-seq 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

(i) What are the pros and cons of each of the discussed pretrained language models: ELMo, BERT, and GPT? In which situations is each type of model most useful for?

Solution: ELMo and BERT are most useful for downstream tasks that require bidirectional context for understanding the content of some text. BERT is typically even better suited for these tasks because its pre-training task is inherently bidirectional, unlike ELMo. BERT also use transformers instead of LSTMs, which can be helpful for modeling long-term dependencies and parallelization of training on large datasets.

GPT models are less suitable for downstream tasks that require text understanding because its representations only incorporate unidirectional context, but they are better suited for text generation because they were explicitly trained for this purpose.

Contributors:

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

