CS 189 Introduction to Machine Learning
Spring 2022 Marvin Zhang

DIS12

1 Paper Walkthrough: Attention is All You Need

In this discussion, we will walk through the paper that introduced transformers, which can be found at this link. The goal is to improve your conceptual understanding of the attention mechanism and transformer model, while also giving practice in breaking down and digesting academic computer science papers, which is an important skill.

We will be using a three-pass approach to reading the paper, as suggested here. These three passes progress from a high-level overview of the main ideas to a low-level deep-dive of the implementation details.

(a) **Pass 1:** Read the title, abstract, introduction, and conclusion sections. Discuss to make sure you know the answers to the following questions:

i. What are the main benefits of the transformer model as it's presented in the paper?

Solution: More suitable to parallel computation compared to recurrent models (which were popular at the time for sequence to sequence modeling tasks), since we can feed in an entire sequence at at time. Easily captures relationships between far-away dependencies in the input. The architecture also significantly outperforms others on translation tasks.

ii. What task is the transformer created to solve? What is it being tested on?

Solution: Machine translation. It's tested on English-to-German and English-to-French translation tasks.

iii. What is the paper's main contribution? What makes it unique compared to previous work?

Solution: It claims to be the first at developing a model that is based entirely around attention, rather than having attention be an additional element to a recurrent model.

(b) **Pass 2:** Now, we'll dive deeper into the specifics of the ideas presented. The second pass involves taking note of key figures, equations, and results, as well as finding other papers to read in the background/relevant work section.

i. Let's take a closer look at Equation (1):

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}})\mathbf{V} \tag{1}$$

1. Given an input $\mathbf{X} \in \mathbb{R}^{T \times d_{model}}$ representing a sequence of T inputs stacked as rows, how do we compute $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ corresponding to the input sequence?
What are the shapes of $\mathbf{Q}, \mathbf{K}, \mathbf{V}$? Let d_v be the value dimension and let d_k be the key and query dimension.

Solution: Q, K, V represent the queries, keys, and values of the input sequence stacked as rows, where $Q \in \mathbb{R}^{T \times d_k}$, $K \in \mathbb{R}^{T \times d_k}$, and $V \in \mathbb{R}^{T \times d_v}$.
We can model Q, K, V as a simple linear layer that transforms each input in the sequence.

$$Q = XW^Q = \begin{bmatrix} x_1^\top \\ \vdots \\ x_T^\top \end{bmatrix} W^Q = \begin{bmatrix} x_1^\top W^Q \\ \vdots \\ x_T^\top W^Q \end{bmatrix}$$

In lecture, we define this linear function $q(x_t) = (W^Q)^\top x_t$, so we can write the query matrix Q as:

$$Q = \begin{bmatrix} q(x_1)^\top \\ \vdots \\ q(x_T)^\top \end{bmatrix}$$

K, V follow the same steps, applying independent linear transformations $k(x_t), v(x_t)$ instead.

2. Which part of this equation is calculating the attention weights?

Solution: Those are calculated by $\text{softmax}(\frac{QK^\top}{\sqrt{d_k}})$.

$$QK^\top = \begin{bmatrix} q_1^\top \\ \vdots \\ q_T^\top \end{bmatrix} \begin{bmatrix} k_1 & \dots & k_T \end{bmatrix} = \begin{bmatrix} q_1^\top k_1 & \dots & q_1^\top k_T \\ \vdots & \ddots & \vdots \\ q_T^\top k_1 & \dots & q_T^\top k_T \end{bmatrix}$$

This results in all the pairwise dot products, which are used as a similarity metric between queries and keys. Queries can be thought of as the learned representation for "outgoing" attention: who should I attend to? Keys can be thought of as a learned representation for incoming attention: who should attend to me? Computing the dot product on these two vectors will select out the ones pointing in similar directions.
Then, we divide all these dot products by $\sqrt{d_k}$.
Then, we have an optional masking step. We can do this by masking this matrix of inner product with $-\infty$ in locations where we don't want to attend between.
Finally, we compute the softmax on each *row* of the matrix to extract the linear combination weights for each query based on the values of the dot products with all keys. These are the attention weights!

$$\text{softmax}(\frac{QK^\top}{\sqrt{d_k}}) = \begin{bmatrix} \text{softmax}(q_1^\top k_1, \dots, q_1^\top k_T) \\ \vdots \\ \text{softmax}(q_T^\top k_1, \dots, q_T^\top k_T) \end{bmatrix} = \begin{bmatrix} \alpha_1^\top \\ \vdots \\ \alpha_T^\top \end{bmatrix}$$

At the end, we multiply these attention weights by the values V, which computes the linear combinations of each row with the values $v_1, \dots, v_T$:

$$\begin{bmatrix} \alpha_1^\top V \\ \vdots \\ \alpha_T^\top V \end{bmatrix}$$

Each row here is as linear combination $V^\top \alpha = \sum_{i=1}^{T} \alpha_i v_i$.

3. Why do we perform *scaled* dot-product attention: why do we divide by $\sqrt{d_k}$ within the softmax?

   Solution: The paper mentions that for large d_k, the "dot products can grow large in magnitude, pushing the softmax function into regions where it has extremely small gradients." Therefore, the scaling factor is meant to counteract this effect.

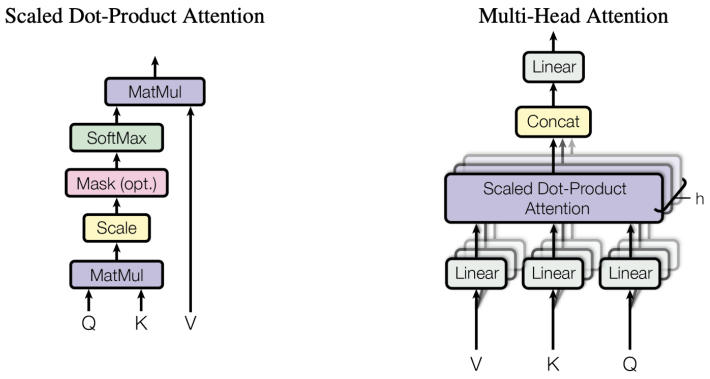

Figure 2: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel.

ii. Next, let's examine how to add multiple attention heads:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O$$
$$\text{where head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

1. Show how we can achieve this multi-head attention by passing out input into h separate self-attention mechanisms, where h is the number of heads.

   Solution: We size down the dimension of queries, keys, and values compared to when we are just performing single-head attention. $d_v = d_k = \frac{d_{model}}{h}$. After that, the process is the same, we just do it h times parallel and concatenate the results at the end. Finally, we perform a final linear transformation back to the original dimension d_{model}. Note that we do not do this step in the homework.

2. What's the benefit of adding more attention heads in parallel?

   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.

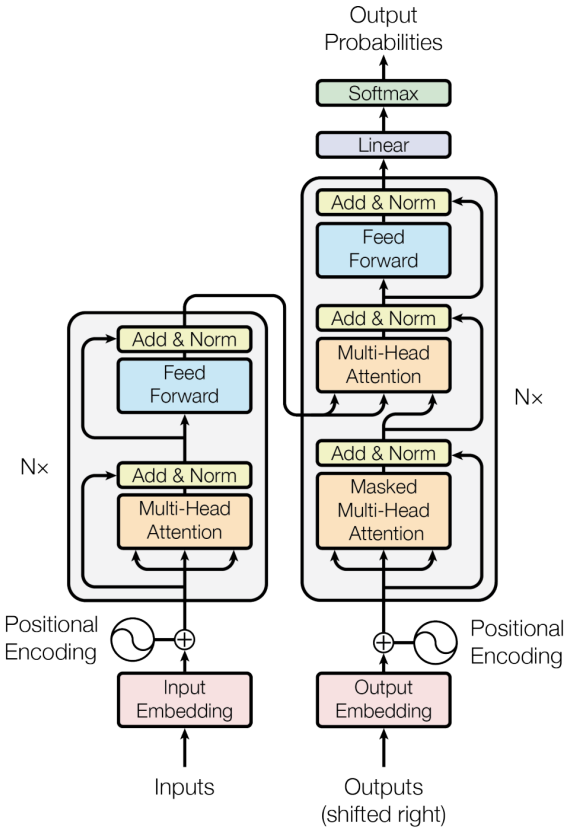

Figure 1: The Transformer - model architecture.

iii. The figure above shows the transformer encoder-decoder architecture. The encoder's job is to condense a source text input (for example, in German) into a real-valued representation which gets fed into the decoder to generate a desired output sequence (for example, an English translation).

1. How exactly does the encoder's representation get used in the decoder?

   Solution: It gets used in the "cross-attention" layer, also called the "encoding-decoding" attention layer in section 3.2.3. This is where the keys and values are supplied by the encoder output, and the queries are supplied by the output string we're in the middle of generating, or by the target string we're training the model with.

   $$Q = q(Y), K = k(Z), V = k(Z)$$

   , where Z is the output of the encoder and Y is the output of the first masked-attention layer of the decoder. See the figure for a visual interpretation.

2. Why does the decoder include a *masked* multi-head attention unit?

   Solution: This is so that we don't do any lookahead in training, since that is impossible at generation time. We don't use information about the generated text before we've generated it.

iv. A few other details addressed in sections 3.3-3.5:

1. What is the reason for positional encoding? How is this typically implemented?

   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.

2. How is this position-wise feedforward network placed between self-attention layers different from a regular feedforward layer?

   Solution: The feedforward layer gets applied to each timestep/position in the sequence independently.

3. For a machine translation task, how is the source text string fed into the encoder (or target text into the decoder)?

   Solution: Words of a specified vocabulary (a set of possible input and output words) are given one-hot vector representations. The network also includes a learned linear embedding at the input to map these one-hot vectors to a continuous representation that is fed through the rest of the transformer architecture.

v. Lastly, take a look at the results in section 6. How does the transformer model compare against the baseline models? What tasks and metrics are used?

   Comparison to baseline models across multiple seeds is important to look for in a paper's results section. Make sure that the experimental findings back up the claims presented elsewhere in a paper.

   Solution: Take a look at Table 2 in the paper.

(c) **Pass 3:** The last step is to implement the paper to try and reproduce the results. You will do this in the homework!