Notes for: Attention Is All You Need

Michael N.*

April 14, 2020

1 High Level and Motivation

Firstly As the name suggests, the paper deals with "transformation" tasks such as, among other things, language models (think gpt-2) and machine translation.

1.1 Limitations of previous methods

RNNs, LSTMs and GRUs suffer because they are not parallelizable, struggle when they need to maintain long-distance relationships and some of their use-cases in seq-to-seq could be seen as immoral (*gasp!*). Are there better solutions?

1.1.1 Parallelizable

Recurrent models tend to operate by taking the history of previously generated tokens and predicting the next one conditioned on that history:

$$p(t_n|t_{n-1}, t_{n-2}, \cdots, t_1, t_0)$$

By this definition we can see that if you want to predict 100 tokens, e.g. 100 words, you'll have to follow this sequential process:

1. predict the first token
2. predict the next conditioned on the first
3. predict the next conditioned on the previous two
4. etc x100.

If you wanted to generate a massive n tokens it will take you $O(n)$ time, and there's no way to parallelize this operation.

*paper: https://arxiv.org/abs/1706.03762

1.1.2 Long distance relationships

The vanishing gradient problem is the problem with RNNs and to a lesser extent LSTMs and GRUs that inputs from many time-steps ago become "diluted" and "lost" amongst newer inputs.

This is a concerning issue with NLP and many seq-to-seq tasks in general because, using language as an example, contextual information such as the conversation topic may only be mentioned once at an early time in a sentence. For example, consider this current document, an RNN would have a hard time storing the information that the current topic is the "Limitations of previous methods" because it happened very early on.

Recurrent Neural Netork

$$h_t = f_\theta(h_{t-1}, x_t)$$

$$h_t = f_\theta(f_\theta(h_{t-2}, x_{t-1}), x_t)$$

As x_t, x_{t-1}, $\cdots$ are combined, it's likely that unless θ is configured perfectly there will be a larger likelyhood of t_0's information being "lost".

1.1.3 Immorality

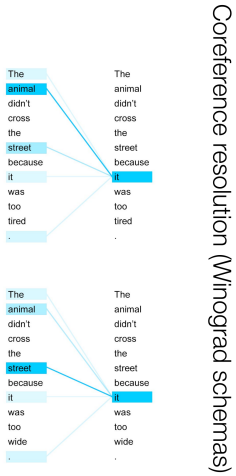
Is position that important? RNNs, LSTMs, GRUs and even non-recurrent methods such as Wavenet (which uses CNNs) all capture the spacial relationships of tokens well. However, in problems such as NLP, they don't capture the highly interconnectedness of words. Take for example Winograd schemas...

Winograd Schema Challenge Winograd Schema questions simply require the machine to identify the antecedent of an ambiguous pronoun in a statement.

- The animal didn't cross the street because it was too tired.
- The animal didn't cross the street because it was too wide.

What is it? All of the techniques discussed so far fail pitifully (or are not great) at these tasks

What's needed? Perhaps some mechanism which relates tokens between each other directly...



1.2 Introducing attention

1.2.1 Intuition of attention

As shown in this figure, each output node "attends" to Values (the lower row of nodes) and essentially is the weighted sum of that row.

1.2.2 Dot product attention

This is the vanilla way to do attention.

There are 3 inputs into an attention algorithim

Key K n vectors* of dimension d_K

Value V n vectors* of dimension d_V

Query Q Vector of dimension $d_Q \equiv d_K$

*These are matrices in reality.

What do we do?

1. Each Key is associated with a Value. (think: dictionary)
2. We need to find the similarity between our Query and each Key
3. Multiply each Value by the corresponding similarity which we just calculated between the Query and Key
4. Each weighted Value is then summed.
5. Output the result

Dot product reminder The dot-product between two vectors:

$$\vec{A} \cdot \vec{B} = |A||B| \cos \theta$$

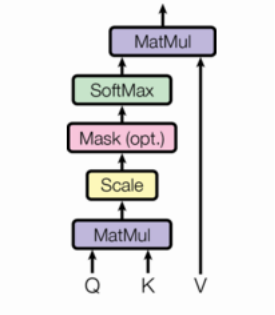
When the two vectors are orthogonal $\cos(\frac{\pi}{2}) = 0$ and therefore the total goes to 0. When they are similar (smaller angle between them) $\cos(0) = 1$ and the resulting product will be higher.

How does this relate to attention? Remember that we want to compute the similarity between the single Query and the Key matrix. Based on what was just discussed, we can compute the dot product. We must also divide by a regularizer to account for large magnitudes at higher dimensions.

$$\frac{QK^T}{\sqrt{d_K}}$$

Adding Values Now that we have the similarity metric between the query and keys we apply the softmax function over the resulting vector and multiply the result by the Value matrix.

$$A(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_K}}\right) V$$



1.3 What is a transformer?

~~All this talk about attention might be making you lose attention~~

A transformer is very simple conceptually (the name gives it away). It simply maps a sequence of inputs $(x_1, \ldots, x_n)$ to a sequence of representations $(z_1, \ldots, z_n)$. Given z, the transformer then generates an output sequence $(y_1, \ldots, y_m)$. Example: machine translation, you start with English sentences, these sentences are mapped to a representation, then the transformer generates the desired language based on the representation.

2 Core content

2.1 Architecture

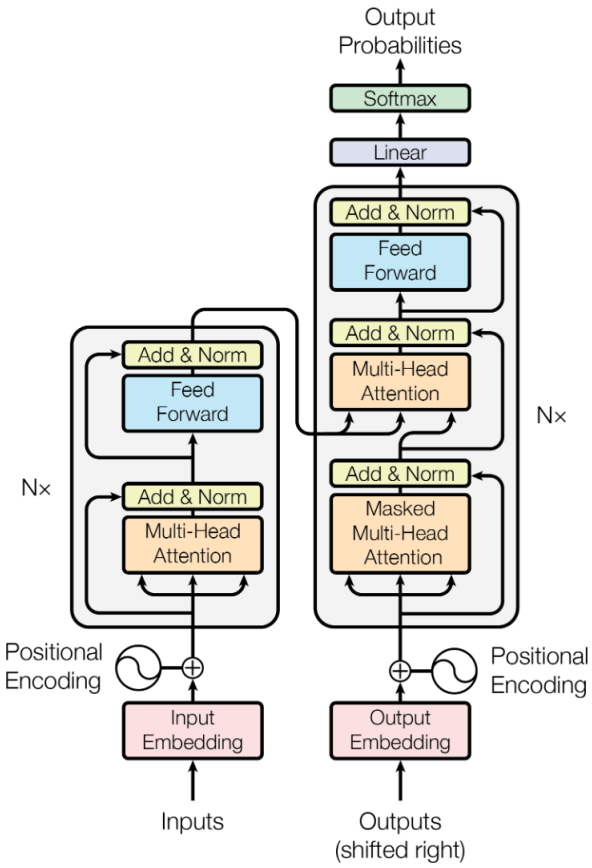


Figure 1: The Transformer - model architecture.

The architecture is made up of different components which can be explained at different levels.

1. Inputs
    - Word embeddings
    - Encoder inputs
    - Decoder inputs (labelled Outputs)
    - Positional encodings
2. Encoder*
    - Multi-head self-attention mechanism
    - Position-wise fully connected feed forward network
    - Layer normalization
3. Decoder*
    - Masked Multi-head self-attention mechanism
    - Multi-head attention over encoder outputs
    - Position-wise fully connected feed forward network
    - Layer normalization
4. Outputs
    - Linear transformation
    - Softmax operation

*Made up of $N\times$ of the sections with a rectangle around (they use 6)

2.2 Inputs

2.2.1 Intuition

The encoder (left column) takes an input of tokens. This input is converted into a representation. The decoder receives any previously generated tokens as input and then it takes this representation generated by the encoder and generates a new token.

2.2.2 "Inputs"

The literal sequence of tokens which will be processed. For MT this might be an english sentence, e.g. "Hello World"

2.2.3 Embeddings

Words are embedded using some method (not really important). E.g. the two words "Hello World" my be encoded in the following two vectors (details not necessary).

$$\text{Hello} = \begin{bmatrix} 0 & 0.2 & 0.8 & 0.3 \end{bmatrix}$$
$$\text{World} = \begin{bmatrix} 0.4 & 0.4 & 0.1 & 0.6 \end{bmatrix}$$

2.2.4 Positional encodings

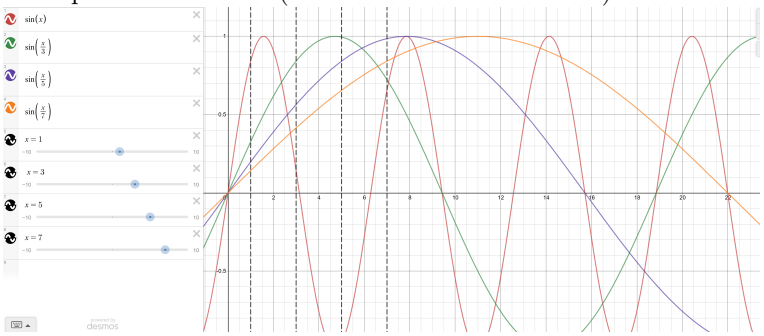
Because attention networks don't "see" position and therefore the distance between tokens, we need to explicitly add the token's position information.

They do this by encoding position using a number of sine waves and then finding the values of each sine wave evaluated at the token's position. For example: the token at position 1 might have a position embedding which looks like:

$$\begin{bmatrix} 0.84 & 0.33 & 0.2 & 0.14 \end{bmatrix}$$

From what I can tell, this value is added elementwise with the word embedding.

Example of sine curves (values are not true to form)



2.3 Encoder

2.3.1 What does the encoder do?

It converts some input sequence into a hidden representation

2.4 Self-Attention

2.4.1 What is it?

In the architecture, self-attention is used, inputs seemingly are converted into Keys, Values **and** Queries. What does this mean!?

The answer is quite simple really... The input matrix X is multiplied by 3 weight matrices:

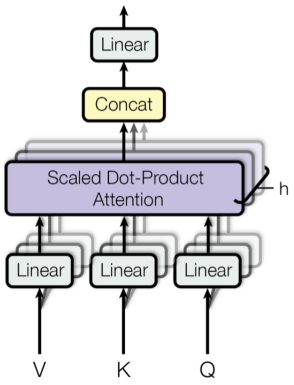
$$X \times W^Q = Q$$
$$X \times W^K = K$$
$$X \times W^V = V$$

These weight matrices are parameters to be learned by the network.

2.4.2 Intuition?

You generate an output of values which are some hidden representation of the input X which (after training) better codes for relationships between elements as the output "attends" to the whole input X.

2.5 Multi-Head Self-Attention



2.5.1 Purpose

In regular attention we softmax over the weighted average of our result vectors. This process is lossy because...

> Multi-headed attention allows the model to jointly attend to information from different representation subspaces at different positions.

2.5.2 How it's made

We linearly project our input, X, h times (for queries, keys and values) with different parameter matrices each time.

$$X_i \times W_i^K = K_i$$
$$X_i \times W_i^Q = Q_i$$
$$X_i \times W_i^V = V_i$$

From the paper Instead of performing a single attention function with d_{model}-dimensional keys, values and queries, we found it beneficial to linearly project the queries, keys and values h times with different, learned linear projections to d_k, d_k and d_v dimensions respectively.

2.6 Masked Multi-Head Self-Attention

2.6.1 What's the purpose of masking?

To preserve the autoregressive properties of the decoder. If you have a sequence of tokens and you would like to perform self attention, how do you prevent the attention system from looking ahead at tokens which it shouldn't.

Example Translating English to German: "I'm a zoomer" $\mapsto$ "Ich bin ein Zoomer."

If your decoder is predicting the final token in the sequence "zoomer", the decoder should not be able to attend to the (not yet generated) token "Zoomer".

2.6.2 How to implement masking

We will set the values of illegal connections to $-\infty$ before they are processed by the softmax function. This ensures they are given no consideration/weight.

$$A(Q, K, V) = \text{softmax}\left([a, b, c, -\infty]\right) V$$

2.7 Point of residual connections

So every position can attend over all other positions. Everything is interconnected man.

3 Their Experiments and Comments

it does well at some stuff