Attention is All You Need

1 Citation

Vaswani, Ashish, et al. ”Attention is all you need.” Advances in Neural Information Processing Systems. 2017.

https://arxiv.org/pdf/1706.03762.pdf

2 Abstract

State of the art sequence processing models have a recurrent (or convolutional) encoder and decoder that are connected with attention. We make a model that only uses attention. This makes it faster to train because it's parallelizable and we set state of the art on English to German translation (beating previous best by over 2 BLEU).

3 Introduction

Encoder-decoder architectures are limited because the encoder and decoder are sequential - they can only process one symbol at a time. This makes them slow and makes it hard to batch examples because we are bottlenecked by the longest example. The Transformer overcomes these limitations by only using attention.

4 Background

Prior work has used convolution as a faster alternative to recurrent units, but Transformer is even faster. Prior work has used self attention inside models.

5 Model Architecture

Our encoder maps $(x_1, ..., x_n)$ to $(z_1, ..., z_n)$. The decoder maps the encoder output to $(y_1, ..., y_m)$.

The encoder has $N = 6$ layers. Each layer has two sub-layers. The first sub-layer is a multi-head attention (more on this later) with residual connection and normalization. The second sub-layer is a feedfoward layer with residual connection and normalization.

The decoder also has $N = 6$ layers. Each layer has three sub-layers. The first is masked multi-head attention (the masking prevents reading future outputs to compute the current output - it sets those values to $-\infty$) with residual connection and normalization. The input are the previously generated outputs. The second is multi-head attention with residual connection and normalization (we inject the encoder output at this point). Finally, we have a feedforward layer with residual connection and normalization.

After the decoder, we feed it's final output to a linear layer and softmax for classification.

Attention basically maps a query and some key-value pairs to an output. It computes a weighted sum of the values, where the weight is a function of how similar the query is to the key.

We use scaled dot product attention (here d_k is the key dimensionality, which must be the same as the query dimensionality):

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

Multi-head attention uses the above approach, but we project the keys, queries, and values to various different dimensionalities. That is, we pick a set of (d_k, d_v) pairs, where each pair defines a head.

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_h)W^O \tag{2}$$

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \tag{3}$$

The feedfoward layer is simply two affine transforms separated by a ReLU.

$$FFN(x) = \max(0, xW_1 + b_1)W_2 + b_2 \tag{4}$$

Now, how do we represent our input/output tokens? We simply learn an embedding vector for each token (or use a pretrained embedding). How do we represent the relative position of the tokens in the input sequence? We use positional embeddings for this. Basically, for position *pos*, the dimensions $2i$ and $2i + 1$ of our positional embedding are given by:

$$PE_{(pos,2i)} = \sin \frac{pos}{10000^{2i/d_{model}}} \tag{5}$$

$$PE_{(pos,2i+1)} = \cos \frac{pos}{10000^{2i/d_{model}}} \tag{6}$$

We use this because PE_{pos+k} is a linear function of PE_{pos}.

6 Why Self-Attention

For two arbitrary tokens in the input sequence, there are a constant number of computations between them (a recurrent layer would have $O(n)$). This, along with the fact that we can parallelize a lot of computation, makes the Transformer model fast. Additionally, attention allows us to learn long range dependencies between tokens that are far away. Attention is faster than convolution as well, and it can even be interpretable (see which tokens in the input sequence have the highest attention weight).

7 Training

We train on WMT 2014 English to German dataset, which has 4.5 million sentence pairs. Our vocabulary had about 40 thousand tokens. Our training batch consisted of about 25 thousand source tokens and 25 thousand target tokens.

We trained on 8 NVIDIA GPUs for 12 hours for our base model. The large model trained for 3.5 days.

We train with ADAM ($\beta_1 = 0.9, \beta_2 = 0.98, \epsilon = 10^{-9}$) and update our learning rate as ($warmupsteps = 4000$):

$$lrate = d_{model}^{-0.5} \cdot \min\left(stepnum^{-0.5}, stepnum \cdot warmupsteps^{-1.5}\right) \tag{7}$$

We use dropout $P_{drop} = 0.1$ for the sublayers and embedding sums. We do label smoothing with $\epsilon_{ls} = 0.1$.

8 Results

We set state of the art on English to German translation, even beating ensembles. We use beam search with size 4 with a length penalty $\alpha = 0.4$.

Don't use too many heads and don't make d_k too small. Learning positional embeddings gives no advantage over using our sinusoidal positional embeddings.

We also trained a Transformer to do English Constituency Parsing. We beat almost all reported models on this task.

9 Conclusion

The Transformer replaces recurrence (and convolution) with attention. It's fast to run and gets state of the art performance on English to German translation. Use it for your sequence to sequence tasks.