Attention is all you need

Shubham Gupta

January 29, 2020

1 Introduction

- This paper review is following the blog from Jay Alammar's blog on the **Illustrated Transformer**. The blog can be found here.

2 Paper Introduction

- New architecture based solely on attention mechanisms called **Transformer**. Gets rids of recurrent and convolution networks completely.
- Generally, RNN used to seq-to-seq tasks such as translation, language modelling, etc.
- Transformer allows for significant parallelization and relies only on attention.

3 Background

- *Self attention* Attention to different positions of a sequence in order to compute a representation of the sequence.

4 Model Architecture

- Transformer uses the following:
  - Encoder decode mechanism
  - Stacked self attention
  - Point wise fully connected layer for encoder and decoder

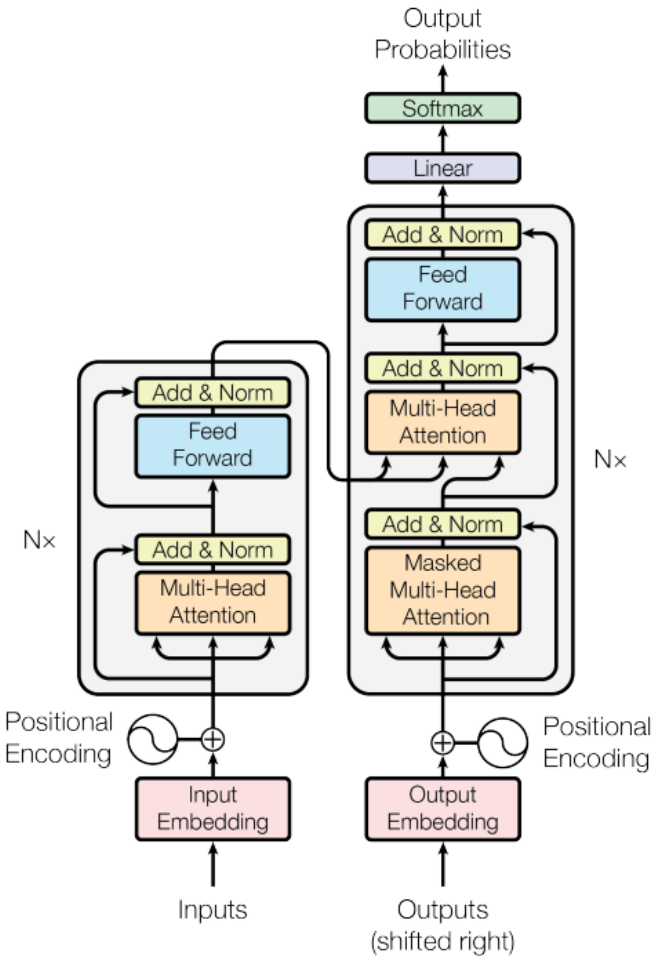

Figure 1: The Transformer - model architecture.

Figure 1: transformer

4.1 Encoder and decoder stacks

- **Encoder**: 6 identical layers. 2 sub layers per layer
- *First*: multi-head self attention mechanism
- *Second*: Fully connected feed forward network
- Apply residual connection for each of the two laters
- Apply layer normalization
- **Decoder**: 6 identical layers. 2 sub layers as above + 1 more which performs multi-head attention over output of encoder stack
- Resodual locks around all 3 sub layers

- Layer normalization
- Modify self-attention sub layer to prevent positions from attending to subsequent positions. Ensures that i output depends only on words before i.

4.2 Attention

- 3 vectors: Query(Q), Key(K) and Value(V)
- Output = Weighted sum of values. Weights assigned as a function of query with key.
- Scaled dot-product attention and multi-head attention

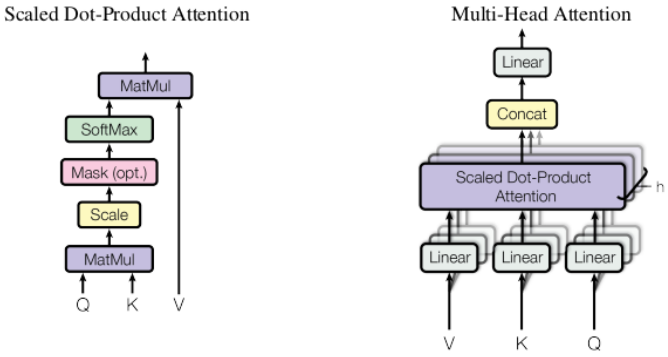

Figure 2: Types of attention

- Attention is calculated as:

$$Attention(Q, K, V) = softmax(\frac{QK6T}{\sqrt{d_k}})V \tag{1}$$

- Dot product attention is **faster and more space-efficient** than additive attention.

4.3 Multi head attention

- Using multile q, k and v vectors. Get the final output, concatenate them and get another final projection d_v.

$$MultiHead(Q, K, V) = Concat(head_1, ..., head_h)W^O werehead_i = Attention(QW_i^Q, KW_i^K, VW_i^V) \tag{2}$$

- $d_k = d_v = d_{model}/h = 64$

4.4 Applications of attention

- **Encoder-decoder attention**: Q from previours decoder, K and V from output of decoder. Attend to all positions in the input sequence.
- **Encoder**: Self attentnion laters. Q,K and V from output of previous layer in the encoder. Some talk about leftward flow, didn't really understand this bit. Will come back to this in sometime.

4.5 Position-wise Feed-Forward Networks

- Each layer contains feed-forward network.

$$FFN(x) = max(o, xW_1, +b_1)W_2 + b_2 \tag{3}$$

4.6 Embeddings and Softmax

- Convert input and output string to vectors of dim d_{model}
- Share weight matrix between two embedding layers and the pre-softmaax linear transformation

4.7 Positional Encoding

- Encode positions of the tokens for the input and output.
- Same vector size i.e d_{model}

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{model}} PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{model}} \tag{4}$$

- Might allow approximation of longer sequence lenghts than seen in the training set

4.8 Why self attentnion?

- Total computational complexity per layer
- Parallel Computation
- Path length between long-range dependencies in the network.

5 Training

5.1 Optimizer

- Use Adam. Vary learning rate according to formula: $lrate = d_{model}^{-0.5}.min(step_num^{-0.5}, step_num.warmu$
- Increase LR for warmup steps, then decrease propotionally to inverse square root of step number. Warmup steps = 4000

5.2 Regularization

- Residual Dropout