Attention is all you need

Abstract

* Current models (pre 2018) are based on recurrent CNNs
* The transformer is based solely on attention mechanisms which dispenses with recurrence and convolutions
* Non recurrence allows training to be parallelizable

Background

* Prior to transformers, CNNs were the premier stuff for translation
* Self-attention – an attention mechanism relating different positions of a single sequence in order to compute a representation of the sequence
* Relies entirely on self-attention to compute representations of its input and output without using sequence aligned RNNs or convolution

Model Architecture

* Encoder structure – encoder maps an input sequence of symbol representations (x1, …, xn) to a sequence of continuous representations z = (z1, …, zn)
* Decoder structure – given z, the decoder generates a symbol of one element at a time (y1, …, ym)
* At each step the model is auto regressive
  + This means comusiming previously generated symbols as additional input when generating the next
* Transformer accomplishes this using stacked attention and point-wise, fully connected layers for the encoder and decoder

Encoder

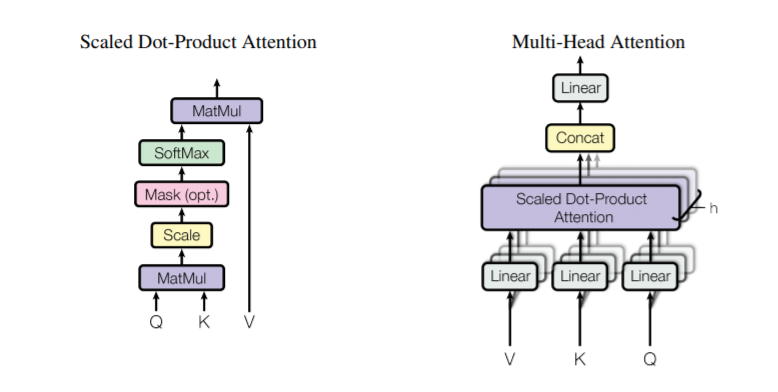
* Composed of a stack of N=6 indentical layers. Each layer has 2 sub layers
* The first sub layer is a multihead attention mechanism
* The second sublayer is a position wise fully connected feed forward network
* “residual connection” around each of the 2 sub layers followed by a layer normalization
* The output of each sublayer is LayerNorm(x + Sublayer(x)) where sublayer(x) is the function implemented by the sublayer itself
* All sublayers as well as embedding layers produce outputs of dimension dmodel = 512

Decoder

* Also a stack of N= 6 layers
* In addition to the 2 sub layers the encoder has, the decoder inserts a 3rd sub layer which performs multi-headed attention over the output of the encoder stack
* We also keep the residual connections and layer normalization we had before
* Masking – modify the self-attention sub layer in the decoder stack to prevent positions from attending subsequent positions

Attention

* Attention function – maps a query and a set of key-value pairs to an output, where query, keys, values, and output are all vectors
* The output is a weighted sum of the values where the weight assigned to each value is computed by a compatibility function of the query with the corresponding value



* Queries and keys are of dimension dk, values are of dimension dv
* We compute the dot products of the query with all keys and divide each by sqrt(dk) and apply softmax
* In practice we compute attention on a set of queries packed together into matrix Q
* Keys are packed into matrix K and values into matrix V
* Attention(Q, K, V) = softmax((QKT)/sqrt(dk))V
* Mutlihead attention learns linear projection of the q, k, and v for dk, dk, and dv dimensions respectively
* Mutlihead(Q, K, V) = concat(head1,…,headh)WO where headi = Attention(QWiQ, KWKi, VWiV)

Applications of Attention in our model

* In encoder and decoder attention layers, the queries come from the previous decoder layer
* FFN(x) = relu(xW1+b1)W2+b2
* For 2i, we do positional encoding = sin(pos/100002i/dmodel)
* For 2i+1, we do positional encoding = cos(pos/100002i/dmodel)

Why self-attention

* Computational complexity per layer
* Amount of computation that can be parallelizable
* Path length between long-range dependencies in the network
  + The shorter these paths the easier it is to learn

Take away

* Transformers seem very complicated to implement, however they also seem to achieve state of the art results
* It makes me wonder if a simpler architecture can be implemented with attention mechanism and positional encoding, kind of like a more streamlined version of an end-to-end memory network