Transformers

AI+X
INSTITUTE FOR ARTIFICIAL INTELLIGENCE

Dominate natural language

- Transformers are the dominate natural language processing approach for both understanding and translation
- They treat language as temporal (meaning depends on where words are in sentence)
  - Outperform LSTM because much of their calculation can be done in parallel and they can have longer influences from past words (history)
- They use attention mechanisms rather than convolutional layers or recurrent neural networks
  - They have had some recent success with image classification

Attention mechanisms

- Attention mechanisms have become an integral part of compelling sequence modeling and transduction
- They allow modeling of dependencies without regard to their distance in either the input or output sequences
- For images the input can be patches
- We will look at the original text based derivation as it is a little simpler

Encoders and decoders

- An Encoder maps an input sequence of symbol representations, words, $(x_1,\ldots, x_n)$ to a sequence of continuous representations $z = (z_1,\ldots, z_n)$
- A decoder, given z, generates an output sequence $(y_1, \ldots, y_m)$ of symbols one element at a time
- The model at every step uses the previously generated symbols as additional input when generating the next

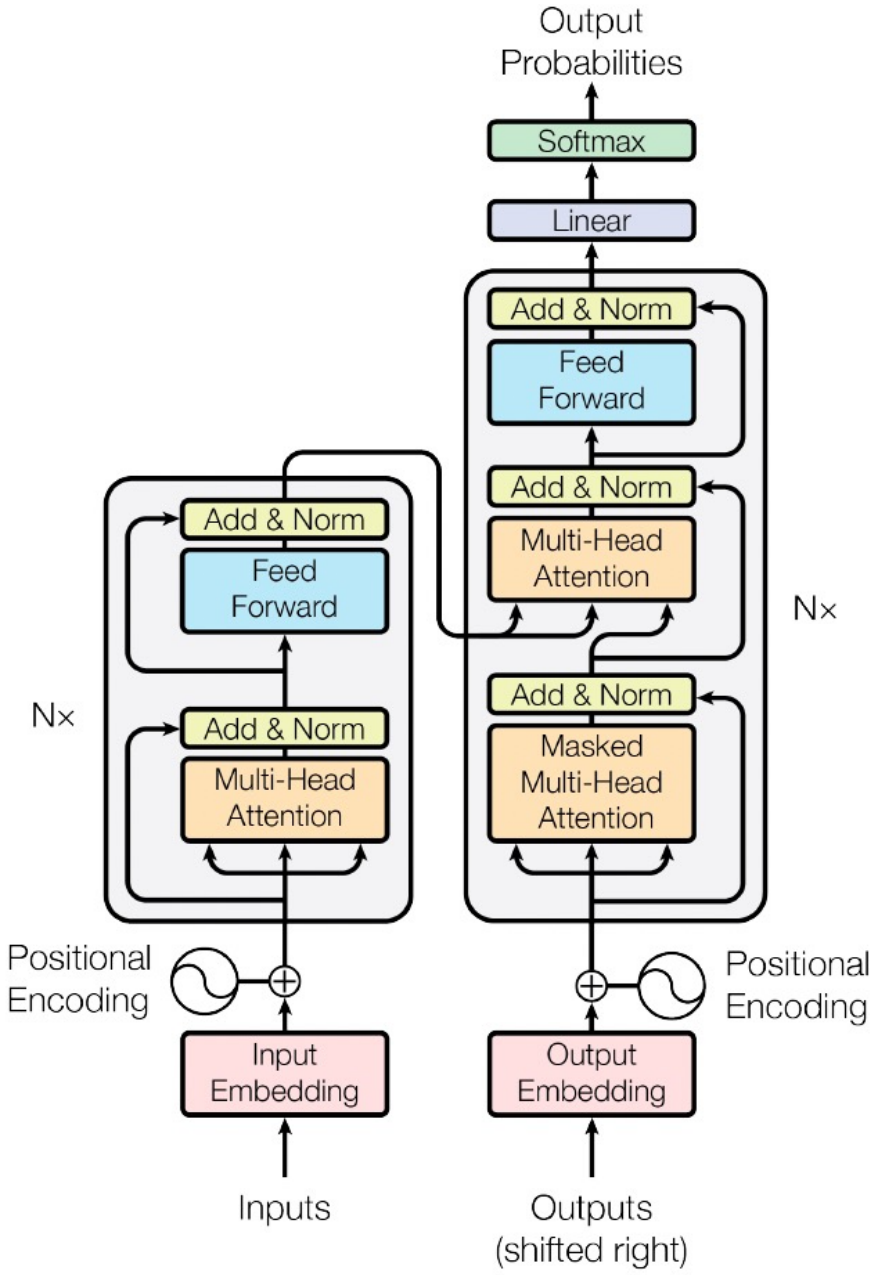
Architecture

- The **encoder** has a variable number of identical layers N, which is 6 in the original Transformer
- The layers are made up of 2 sublayers
- First a multi-head attention layer which is followed by a position-wise fully connected neural layer
- There is a residual connection around each of the two sub-layers, followed by layer normalization
- Output of each sub-layer is LayerNorm(x + Sublayer(x)), where Sublayer(x) is the function implemented by the sub-layer
- All sub-layers in the model, as well as the embedding layers, produce outputs of dimension 512

Decoder

- Also stack of N = 6 identical layers and it has a third sub-layer, which performs multi-head attention over the output of the encoder
- Has residual connections around each of the sub-layers, followed by layer normalization
- The self-attention sub-layer in the decoder stack is modified to prevent positions from attending to subsequent positions
- Combined with fact that the output embeddings are offset by one position, ensures predictions for position *i* depend only on the known outputs at positions less than *i*

Visual depiction of Transformers



From: Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. "Attention is all you need." *Advances in neural information processing systems* 30 (2017).

Attention

- An attention function can be described as mapping a query and a set of key-value pairs to an output
- The query, keys, values, and output are all vectors
- The output is computed as a weighted sum of the values
- The weight assigned to each value is computed by a compatibility function of the query with the corresponding key

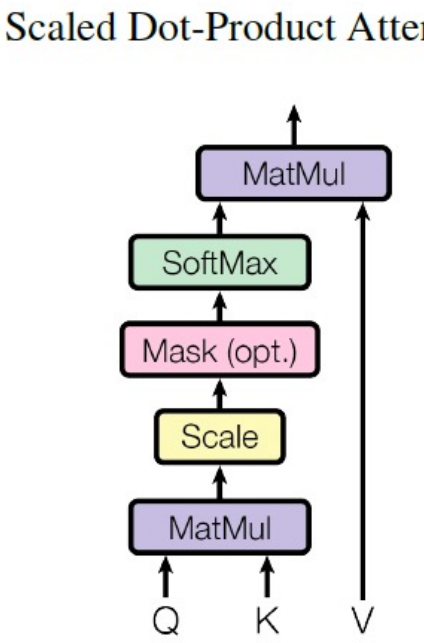
Attention

- Input consists of queries and keys of dimension d_k, and values of dimension d_v
- Compute dot product of query with all keys, divide each by sqrt(d_k), and apply a softmax function to obtain the weights on the values
- In practice, compute the attention function on a set of queries simultaneously, packed together into a matrix Q. The keys and values are also packed together into matrices K and V
- Attention(Q,K,V) = softmax(QK^T)/ sqrt(d_k)) V

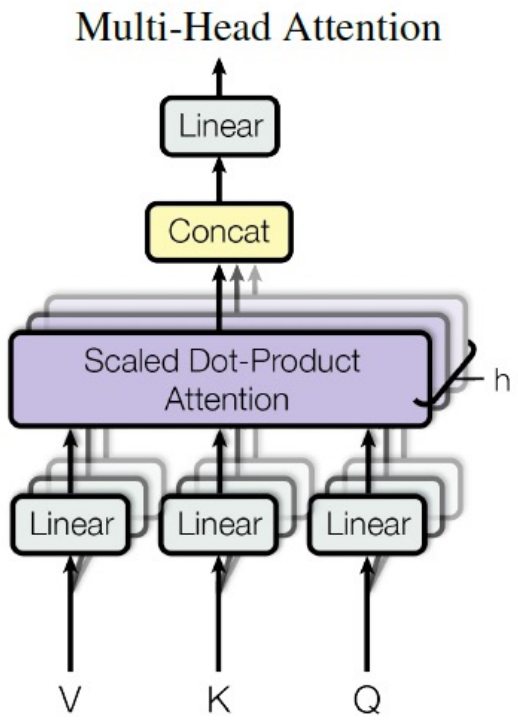
Multi-Head Attention

- Instead of performing a single attention function with d_{model}-dimensional keys, values and queries; linearly project the queries, keys and values **h** times with different, learned linear projections to d_k, d_k and d_v dimensions, respectively
- For each of these projected versions of queries, keys and values the attention function is done in parallel, yielding d_v-dimensional output values
- These are concatenated and once again projected, resulting in the final values

Attention visualized



Scaled Dot-Product Attention



Multi-Head Attention consists of **h** attention layers running in parallel

Why Multi-head attention?

- It allows the model to jointly attend to information from different representation subspaces at different positions
- With a single attention head, averaging will be done which inhibits looking at different representation subspaces

Multi-Head Attention Math

- MultiHead(Q,K,V) = Concat($head_1$,…,$head_h$)W

  where $head_i$=Attention(QW_i^Q, KW_i^K, VW_i^V) and the projections are the parameter matrices

- $W_i^Q \in R^{d_{model} \times d_k}, W_i^K \in R^{d_{model} \times d_k}, W_i^V \in R^{d_{model} \times d_v}$
- $W^O \in R^{hd_v \times d_{model}}$

In original work h = 8 parallel attention layers, or heads.
Each of these use $d_k = d_v = d_{model}/h = 64$

Use of Attention for word sequences

- For *encoder-decoder attention* layers, the queries come from the previous decoder layer, with the memory keys and values coming from the output of the encoder
  - So every position in the decoder can attend over all positions in the input sequence
- The encoder contains self-attention layers where all of the keys, values and queries come from the same place
  - the output of the previous layer in the encoder. Each position in the encoder can attend to all positions in the previous layer of the encoder
- Self-attention layers in the decoder allow each position in the decoder to attend to all positions in the decoder up to and including that position
- To prevent leftward information flow in the decoder to preserve the auto-regressive property inside of scaled dot-product attention there is a masking out (setting to $-\infty$) all values in the input of the softmax which correspond to illegal connections

Position-wise Feed-Forward Networks

- Each of the layers in the encoder and decoder contains a fully connected feed-forward network, applied to each position separately and identically
- This consists of two linear transformations with a ReLU activation in between: FFN(x) = $\max(0; xW_1 + b_1)W_2 + b_2$
- The linear transformations are the same across different positions, though with different parameters from layer to layer

Positional Encoding

- For a model to make use of the order of the sequence, some information about the relative or absolute position of the tokens in the sequence is needed
- So, "positional encodings" are added to the input embeddings at bottoms of encoder and decoder stacks
- Positional encodings have the same dimension d_{model} as the embeddings, so that the two can be summed

Positional Encoding

- Sine and cosine functions of different frequencies can be used. Learned embeddings also work well, but require more work (learning)
- $PE_{(pos,2i)}=\sin(\frac{pos}{10000^{d_{model}}})$
- $PE_{(pos,2i+1)}=\cos(\frac{pos}{10000^{d_{model}}})$
- Where pos is the position and i the dimension.

Why self-attention

- Less computational complexity per layer
- Lots that can be parallelized unlike LSTMs
- Path length can be long, i.e. long range dependencies can be modeled
- Self-attention layers are faster than recurrent layers when sequence length n is smaller than the representation dimensionality d

You have reached the end of the lecture.