The "Transformer"

"Attention is All You Need"

Attention is a way of

- encoding long-range dependencies
- between an arbitrary number of elements
- with a fixed number of parameters

... We don't need the RNN anymore.

building blocks

- attention mechanism
- scaled dot-product attention
- multi-head attention
- self-attention
- positional encoding

Self-attention

- each input token attends to each other
- captures inter-input dependencies that RNN formerly captured

Positional encoding

All notion of sequence position is lost when removing the RNN.

We want an encoding such that:

- encodings are unique
- distance between time steps is consistent for sequences of different lengths
- values are bounded - can hopefully extrapolate to longer sequences

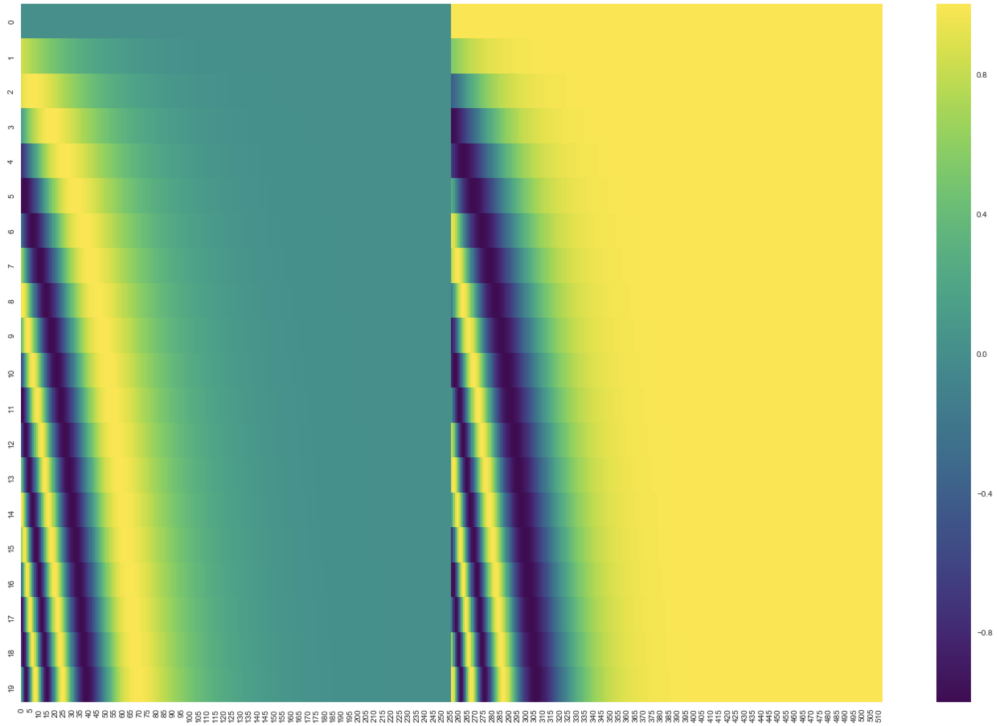
we want the distance between the first and the second token to be the same as the distance between the second and the third

the distance between these encoding vectors is indicative of the distance between those tokens in the sequence

the value of an element in the encoded vector, because the network is not good at extrapolating to values that it hasn't seen before

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

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



A real example of positional encoding for 20 words (rows) with an embedding size of 512 (columns). You can see that it appears split in half down the center. That's because the values of the left half are generated by one function (which uses sine), and the right half is generated by another function (which uses cosine). They're then concatenated to form each of the positional encoding vectors.

http://jalammar.github.io/illustrated-transformer/

Positional encodings, cont.

- positional encodings are added to word encodings
  - no convincing justification for this...
  - but it works.

instead of taking our inputs and sending them straight into the output attention, we go through the self attention layer first and get the meaning of the input sequence tokens before we decide how the input tokens will attend to them

Self-attention benefits

- all pairs of inputs/outputs are separated by a path of constant length → learning dependencies is easier
- encoding/decoding is distributed → more interpretable
- the longest sequence of computations is shorter → more parallelizable → *speed*

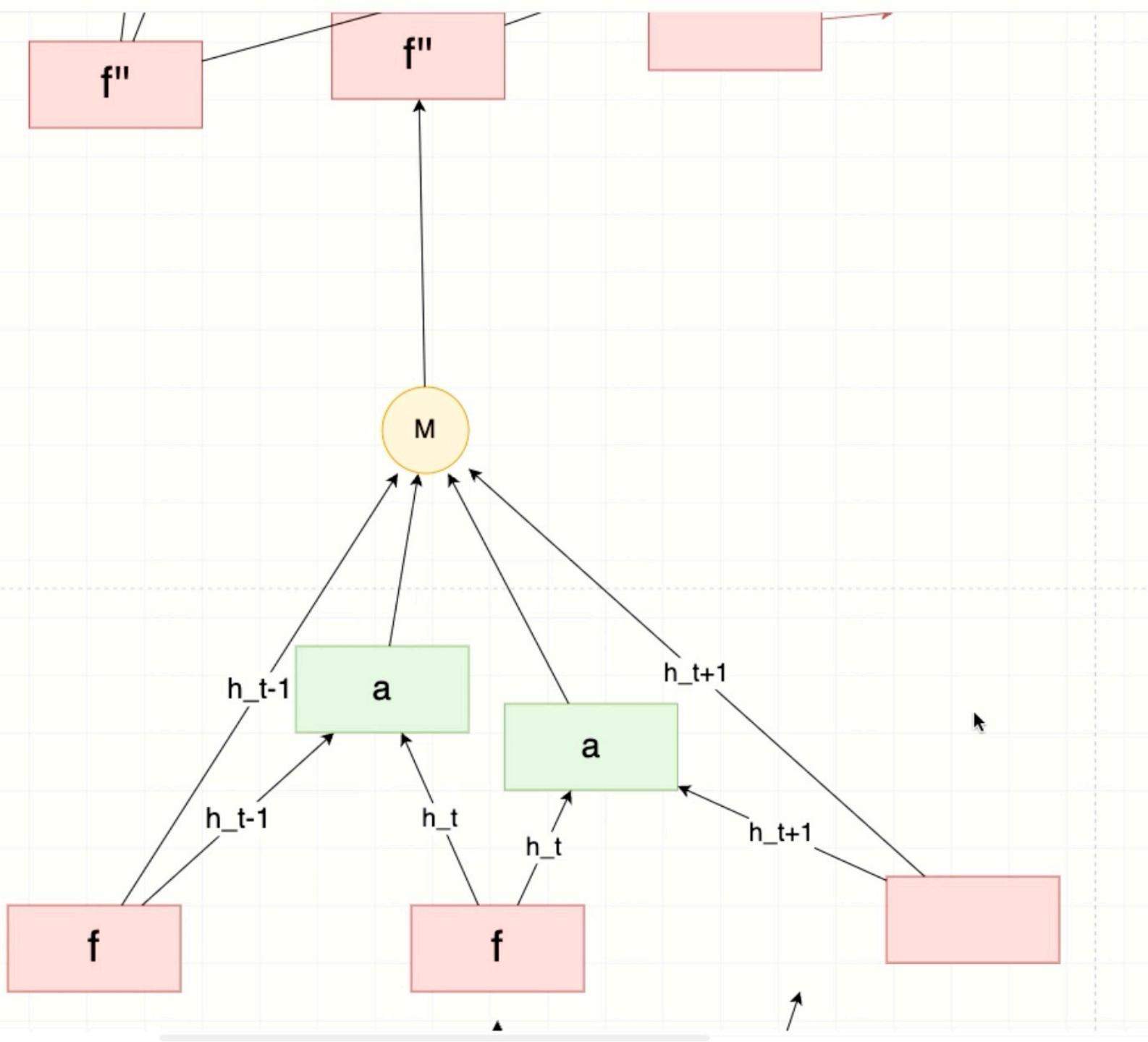

f''
f''
M
h_t-1
a
h_t+1
a
h_t-1
h_t
h_t
h_t+1
f
f

Table 1: Maximum path lengths, per-layer complexity and minimum number of sequential operations for different layer types. n is the sequence length, d is the representation dimension, k is the kernel size of convolutions and r the size of the neighborhood in restricted self-attention.

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

Quadratic complexity

n^2 can be large. How can we avoid it?

- limit self-attention to a fixed window (r)
- add structural sparsity-inducing prior of some kind
- ...
- Rethinking Attention with Performers

Transformer architecture

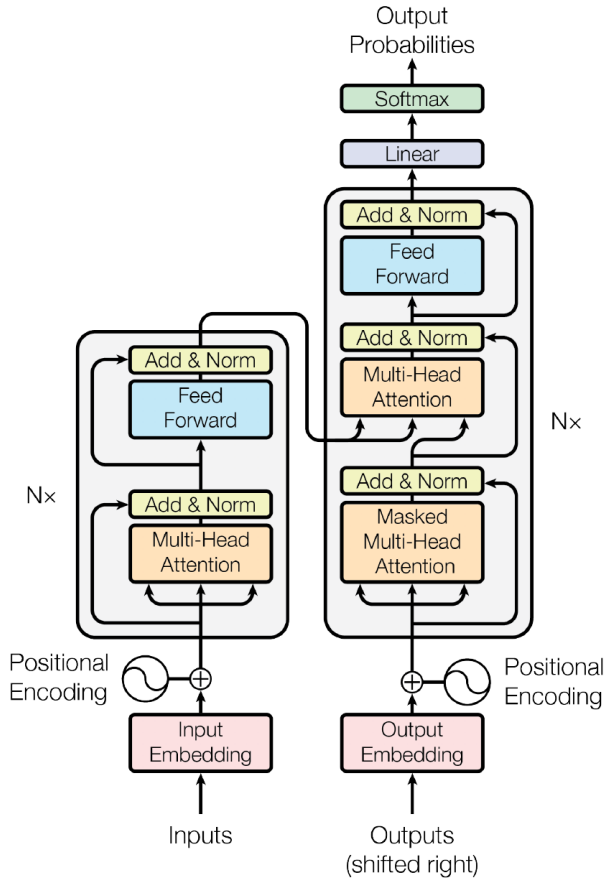


Figure 1: The Transformer - model architecture.

mask: only allowed to look back
though we have all information during train time, we need to force this to only use information from the past because at runtime won't have the future content that masking

References

- Attention Is All You Need
- https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html
- http://jalammar.github.io/illustrated-transformer/