Neural Machine Translation models

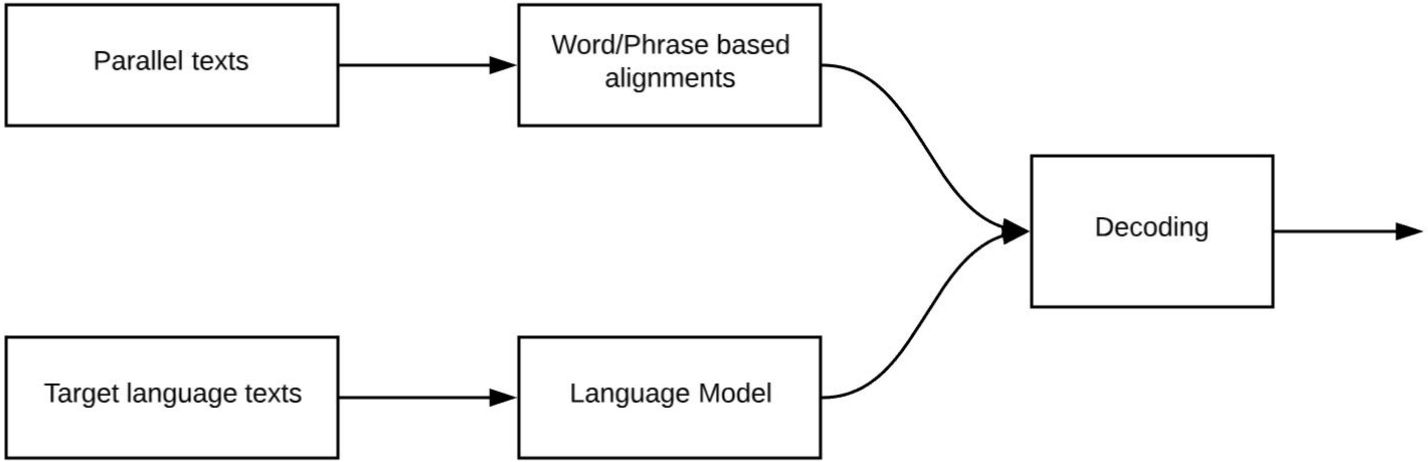
High level overview of Machine Translation Models

- We want to maximize the likelihood of a given translation given a source sentence

$$\arg\max_{e} P(e|f) \propto P(f|e)P(e)$$

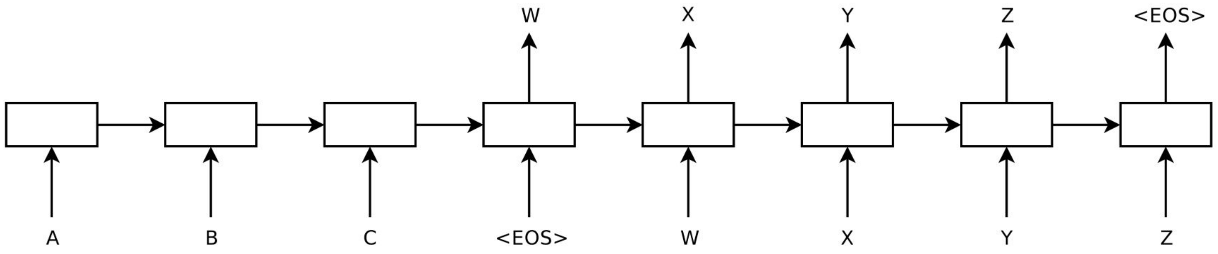
- Where e is the target language (i.e. english) and f is the source language (i.e. foreign)
- We can think of the likelihood term as a translation model and the prior as a language model for the target language

Traditional Machine Translation Pipeline



An end to end RNN encoder-decoder architecture

- For the encoder stage, the source language is processed token by token by an RNN model (LTSM or similar)
- The Decoder stage acts just as a language model by generating translated words one at a time and using the previous word as input to generate the next word.
- We use the hidden state of the final encoder time step to initialize the decoder's hidden state (known as a context vector)
- Generation stops when the <EOS> token is generated.



Sequence to Sequence Learning with Neural Networks (2014)

Ilya Sutskever, Oriol Vinyals, Quoc V. Le

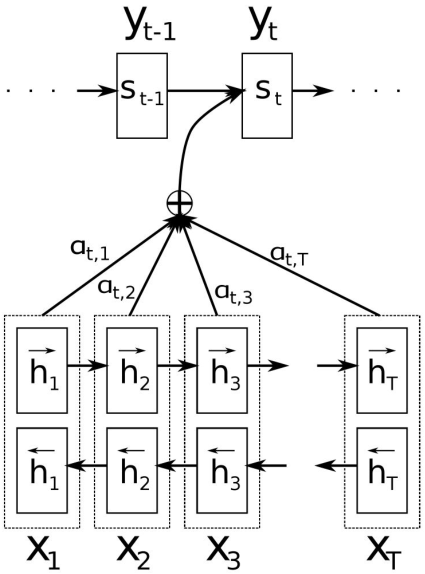
- Such a simple model was able to be competitive with the state of the art phrase based machine translation systems at the time
- One trick they employed was to reverse the source sequence in the encoder
    - The rationale behind this was that this would put parallel words closer together in the encoder-decoder setup.

Drawbacks to this approach

- The encoder needs to be able to compress all the information in the source sentence into a single context vector to initialize the state of the decoder network.
- This can cause performance to suffer on longer sequences.

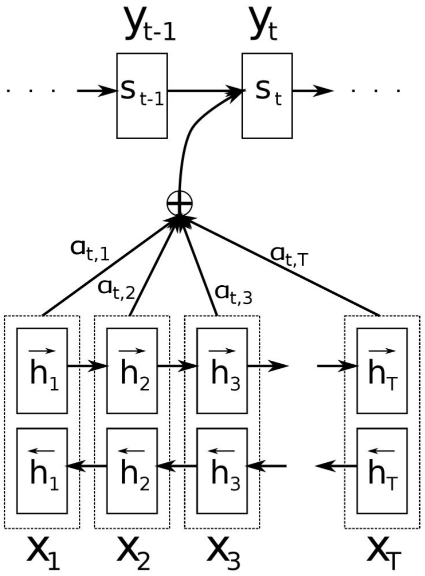
Neural Machine Translation By Jointly Learning to Align and Translate (2015)

Dzmitry Bahdanau, Kyung Hyun Cho, Yoshua Bengio



- The main contribution of this paper was instead of relying on a single context vector to represent a source input, they introduced the notion of an **attention mechanism** to provide context vectors at each timestep to inform what parts of the source sentence were relevant to the output at a particular time step.

Attention in the encoder-decoder model



- For each time step t in the output sequence
  - We calculate a context vector

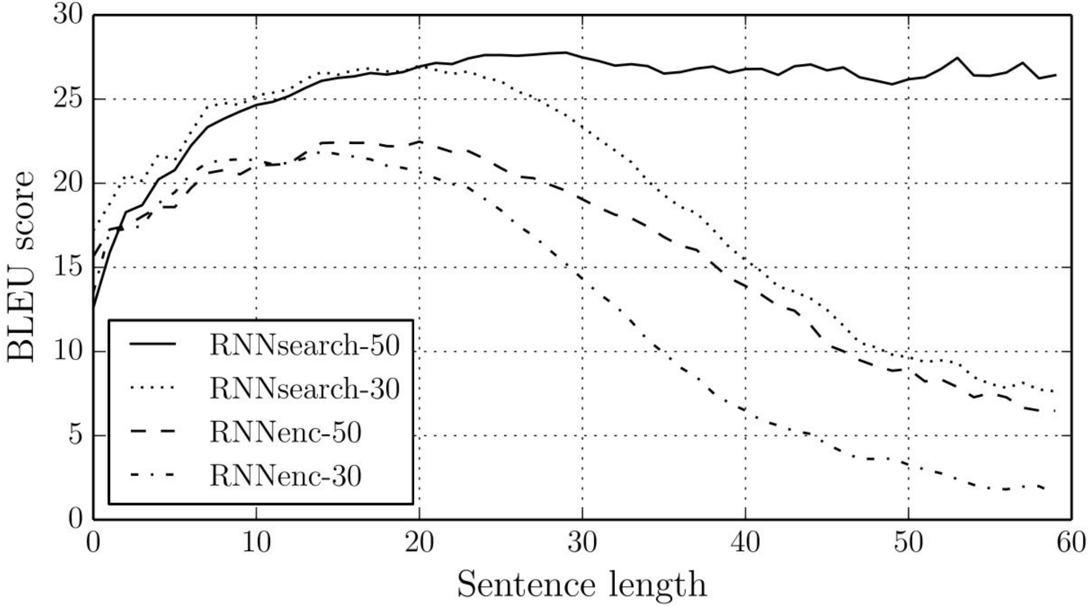
$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$$

Where

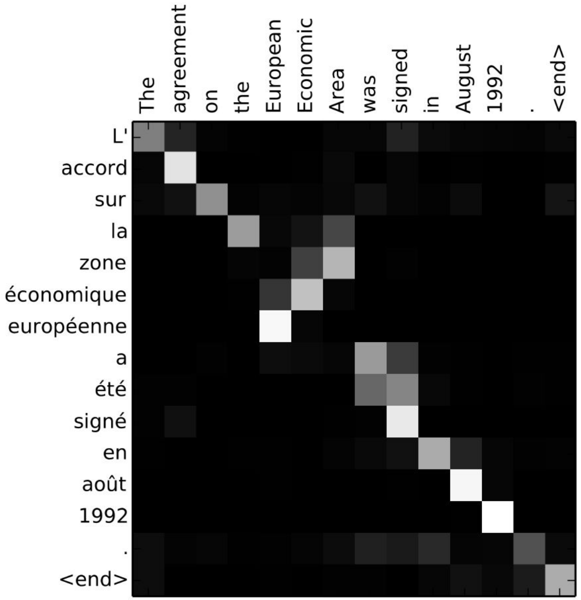
$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})}$$

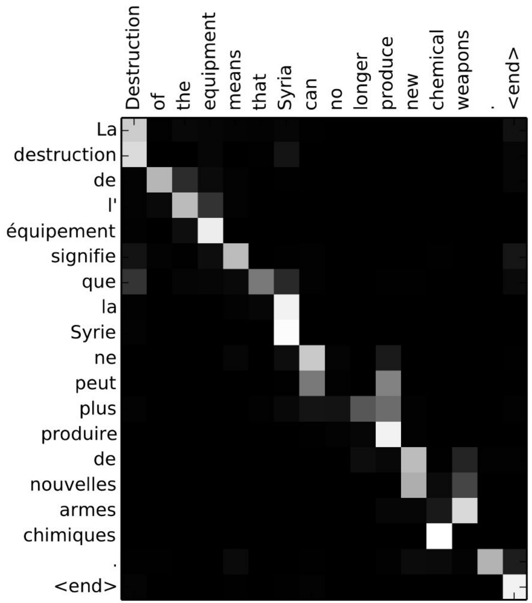
$$e_{ij} = a(s_{i-1}, h_j)$$

Result: Performs better for longer sentences



Visualizing the Attention “Soft Alignments”



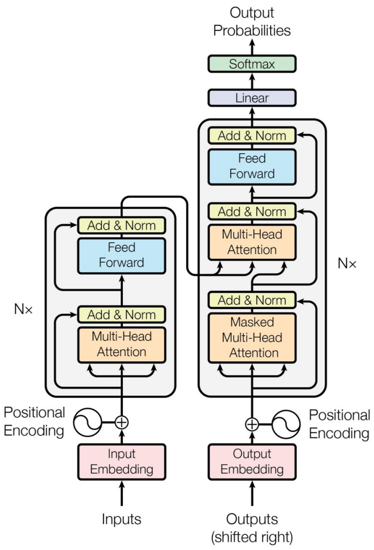


Attention Is All You Need (2017)

Vaswani et al.

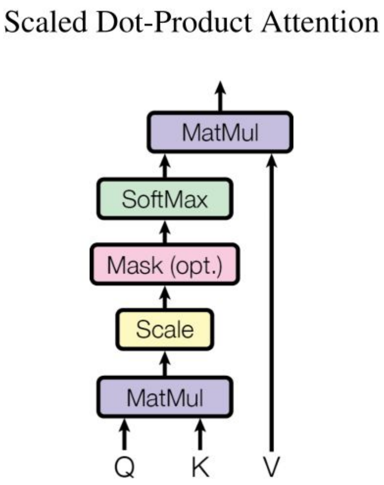
- Google introduces a new architecture that dispenses with RNN/CNNs entirely in favor of what is essentially a feedforward model.
- Replace RNNs with what they refer to as multi-head self attention with positional encoding

Model Architecture



- Consists of two main components
  - encoder/decoder like before
- The RNN stages are now replaced by multi-head self attention
- Additionally to capture the sequential nature of the input, they add a positional encoding to the input embeddings before they enter the encoder/decoder

Scaled Dot-Product Attention

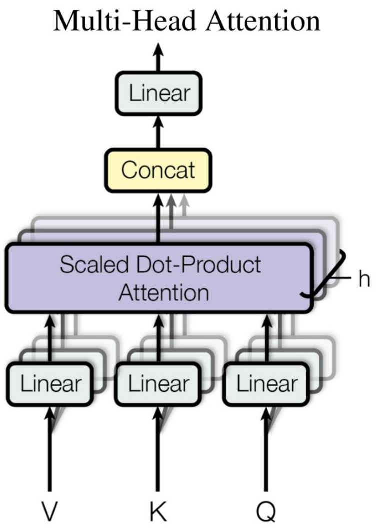


- Multiplicative attention as opposed to the summing attention we saw previously (better suited for hardware acceleration)
- The inputs to an attention layer are
  - Queries -- Q
  - Keys -- K
  - Values -- V

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

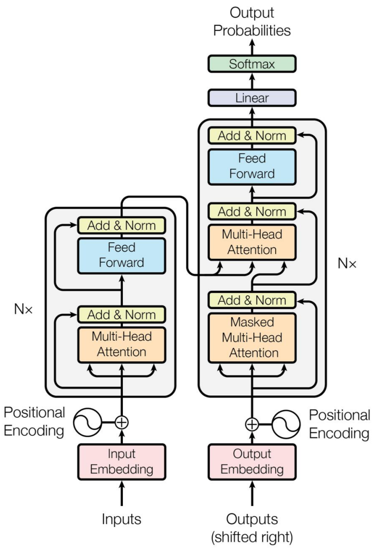
- Computes the similarity between Q and K.
  - Based on the softmax score, we will weight V accordingly
- Think of K matrix as an index over a set of values V
  - If the Query matches a particular key, it will scale the values V accordingly to “retrieve” it

Multi-Head Attention



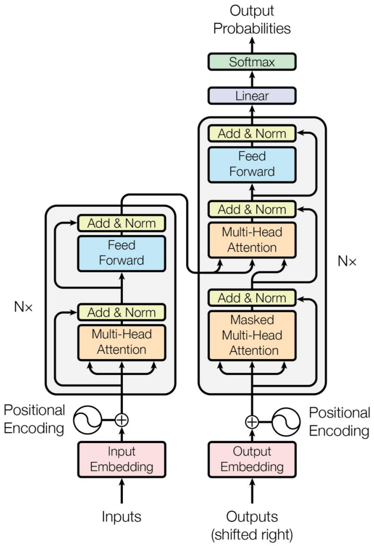
- For each set of Keys, Values, and Queries, we project them into several smaller subspaces with a linear transformation
- Apply attention to each "head", concatenate and project again
- This allows each head to attend to different parts of the input while maintaining similar computational complexity of a larger attention layer.

Understanding Each Attention Component



- We can view the attention layer between the encoder/decoder just as with the RNN version.
  - Keys/Values come from the encoder and Queries come from the decoder
- In the encoder K, V, and Q all come from the input embeddings, hence the term self attention
- Similarly the decoder also contains a self attention layer, but there is an addition of an autoregressive mask to prevent the output from attending to positions in the future.

Position-wise Feed-Forward Layers

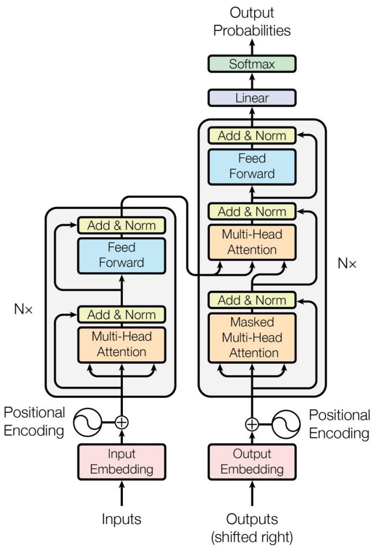


- The output of each main component has what is referred to as a position-wise feed-forward network
- For each layer (eg for an encoder layer) we apply the following **at each position with the same weights**

$$FFN(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

Does this remind you of anything?

Positional Encoding

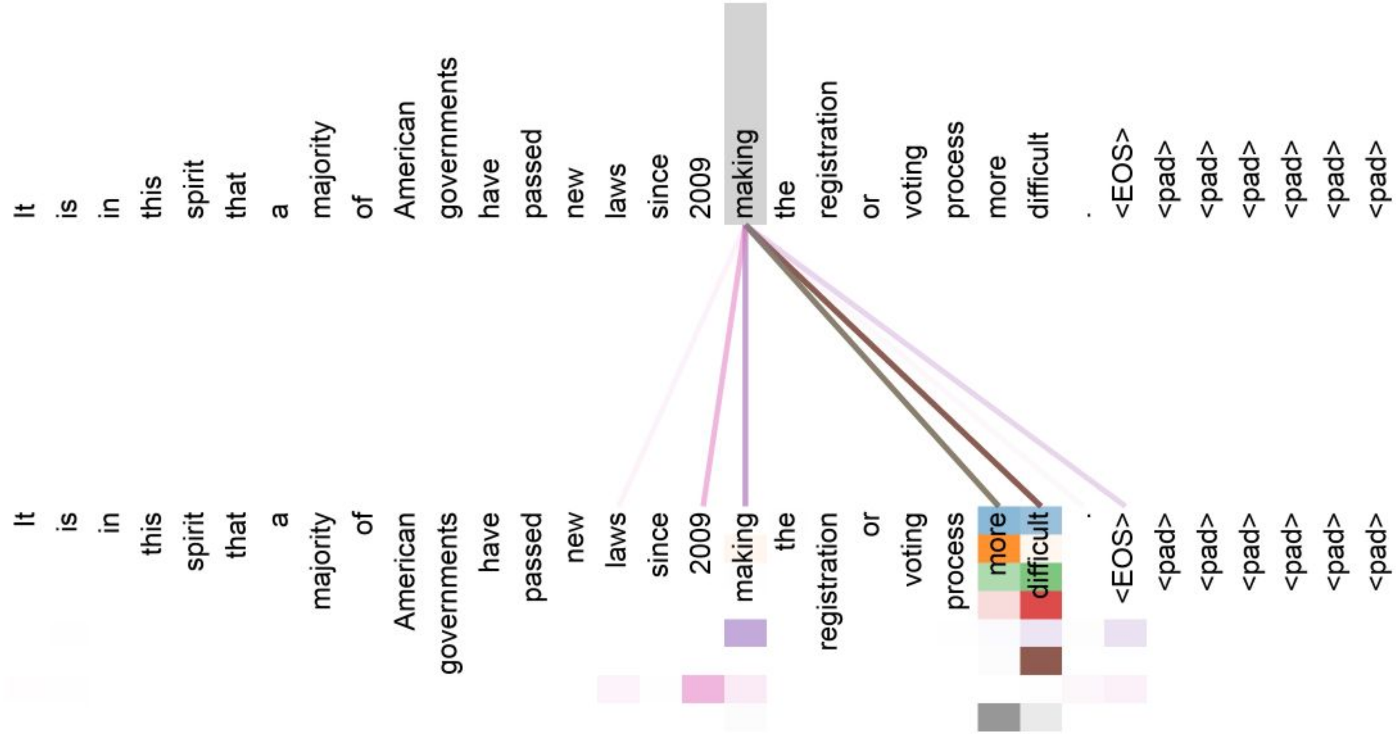


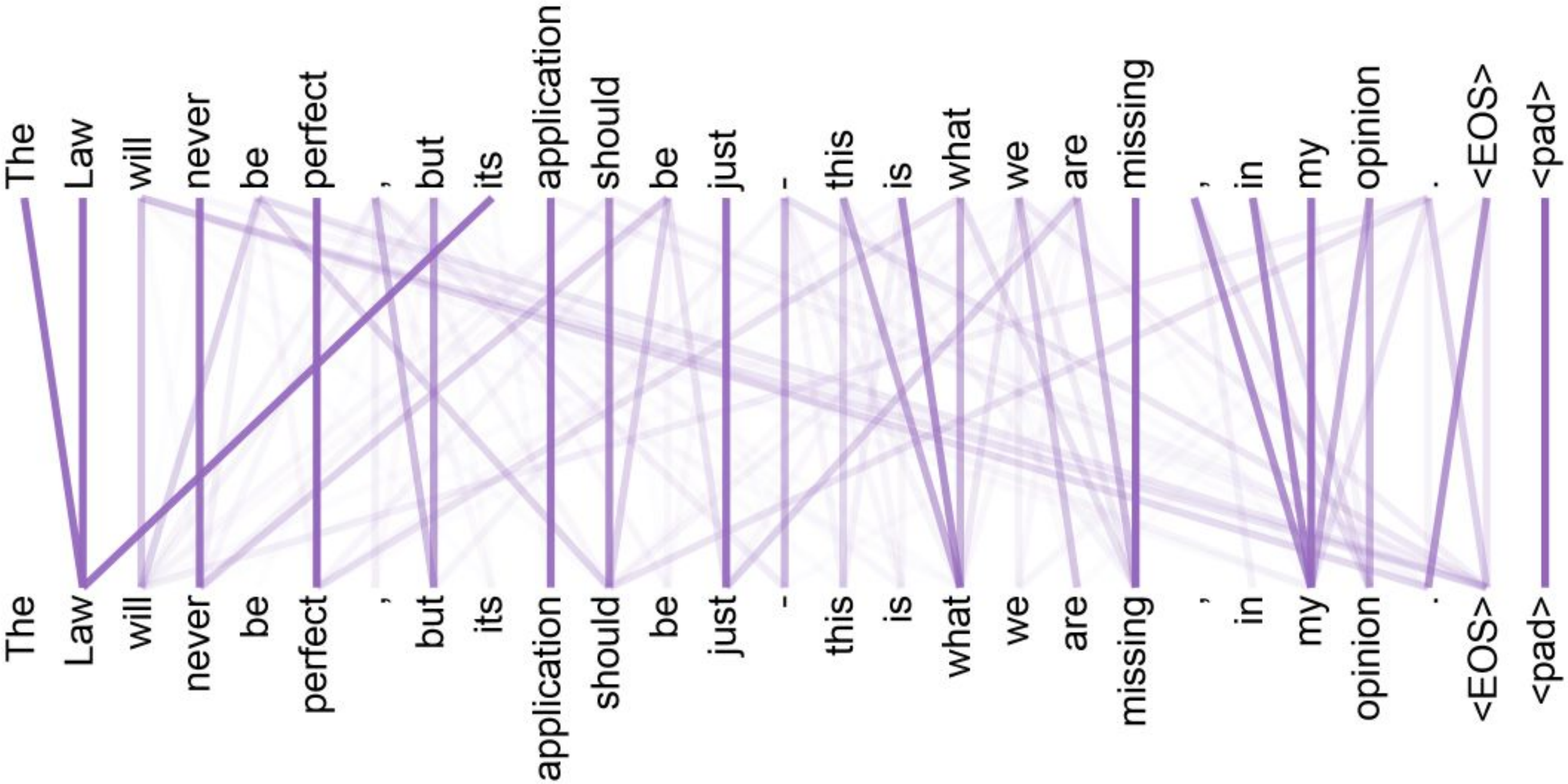
- Necessary to capture the sequential nature of the input now that we have dispensed with recurrent/convolutional layers
- The authors tested two methods
    - Learned positional encoding
    - Hand crafted sinusoidal encoding
        - I.e. each dimension of an input vector consists of a different frequency sinusoid

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

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

Attention Visualizations




The Law will never be perfect , but its application should be just - this is what we are missing , in my opinion . <EOS> <pad>
The Law will never be perfect , but its application should be just - this is what we are missing , in my opinion . <EOS> <pad>