CS60010: Deep Learning

Sudeshna Sarkar

Spring 2019

ATTENTION - 2

1 Mar 2019

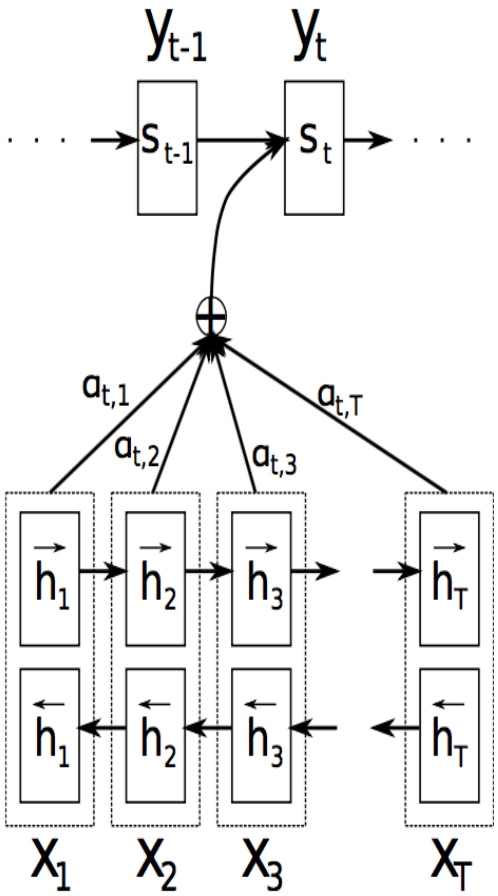
RNN seq2seq

- Encoder-decoder with a soft-attention
  - Encoder:

$$\overrightarrow{\mathbf{h}}_t = \overrightarrow{\Psi}_{\text{enc}}\left(\overrightarrow{\mathbf{h}}_{t-1}, \mathbf{E}_x\left[x_t\right]\right) \qquad \mathbf{h}_t = \left[\overrightarrow{\mathbf{h}}_t; \overleftarrow{\mathbf{h}}_t\right]$$

  - D

$$\overleftarrow{\mathbf{h}}_t = \overleftarrow{\Psi}_{\text{enc}}\left(\overleftarrow{\mathbf{h}}_{t+1}, \mathbf{E}_x\left[x_t\right]\right)$$



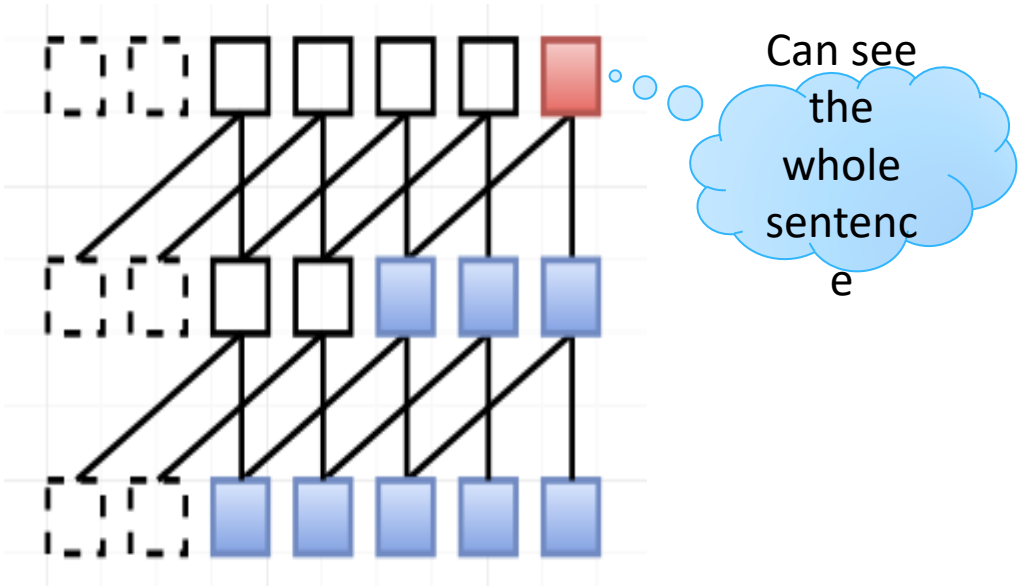
$$\mathbf{z}_t = \Psi_{\text{dec}}\left(\mathbf{z}_{t-1}, \mathbf{E}_y\left[\tilde{y}_{t-1}\right], \mathbf{c}_t\right)$$

$$e_{t,i} = f_{\text{score}}\left(\mathbf{h}_i, \mathbf{z}_{t-1}, \mathbf{E}_y\left[\tilde{y}_{t-1}\right]\right)$$

$$\alpha_{t,i} = \frac{\exp\left(e_{t,i}\right)}{\sum_{j=1}^{T_x} \exp\left(e_{t,j}\right)} \qquad \mathbf{c}_t = \sum_{i=1}^{T_x} \alpha_{t,i}\mathbf{h}_i$$

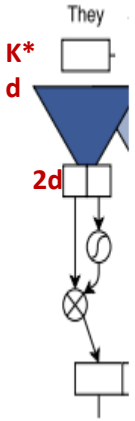
CNN seq2seq

- Advantages:
  - Do not depend on previous steps => parallelization
  - Hierarchical structure provides a shorter path to capture long-range dependencies
  - O(n) -> O(n/k)



Convolution filter

Each convolution kernel is parameterized as $W \in \mathbb{R}^{2d \times kd}$, $b_w \in \mathbb{R}^{2d}$ and takes as input $X \in \mathbb{R}^{k \times d}$ which is a concatenation of k input elements embedded in d dimensions and maps them to a single output element $Y \in \mathbb{R}^{2d}$ that has twice the dimensionality of the input elements;



Nonlinearity: Gated linear unit

$$Y = [A\ B] \in \mathbb{R}^{2d}:$$

$$v([A\ B]) = A \otimes \sigma(B)$$

Pointwise multiplication

$\sigma(B)$ control which part of A is relevant

Overall encoder-decoder structure (see later)

$\mathbf{h}^l = (h_1^l, \ldots, h_n^l)$ Output of the decoder

$\mathbf{z}^l = (z_1^l, \ldots, z_m^l)$ Output of the encoder

To enable deep convolutional networks, we add residual connections from the input of each convolution to the output of the block (He et al., 2015a).

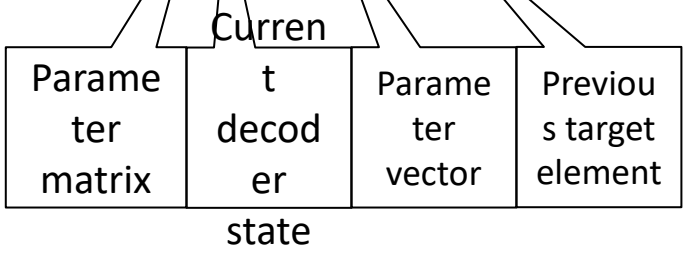
$$h_i^l = v(W^l[h_{i-k/2}^{l-1}, \ldots, h_{i+k/2}^{l-1}] + b_w^l) + h_i^{l-1}$$

Multi-step attention

- attention mechanism for each decoder layer.

Decoder state i summary:

$$d_i^l = W_d^l h_i^l + b_d^l + g_i$$



For decoder layer l the attention a_{ij}^l of state i and source element j is computed as a dot-product between the decoder state summary d_i^l and each output z_j^u of the last encoder block u:

$$a_{ij}^l = \frac{\exp\left(d_i^l \cdot z_j^u\right)}{\sum_{t=1}^{m} \exp\left(d_i^l \cdot z_t^u\right)}$$

$\mathbf{h}^l = (h_1^l, \ldots, h_n^l)$ Output of the decoder

$\mathbf{z}^l = (z_1^l, \ldots, z_m^l)$ Output of the encoder

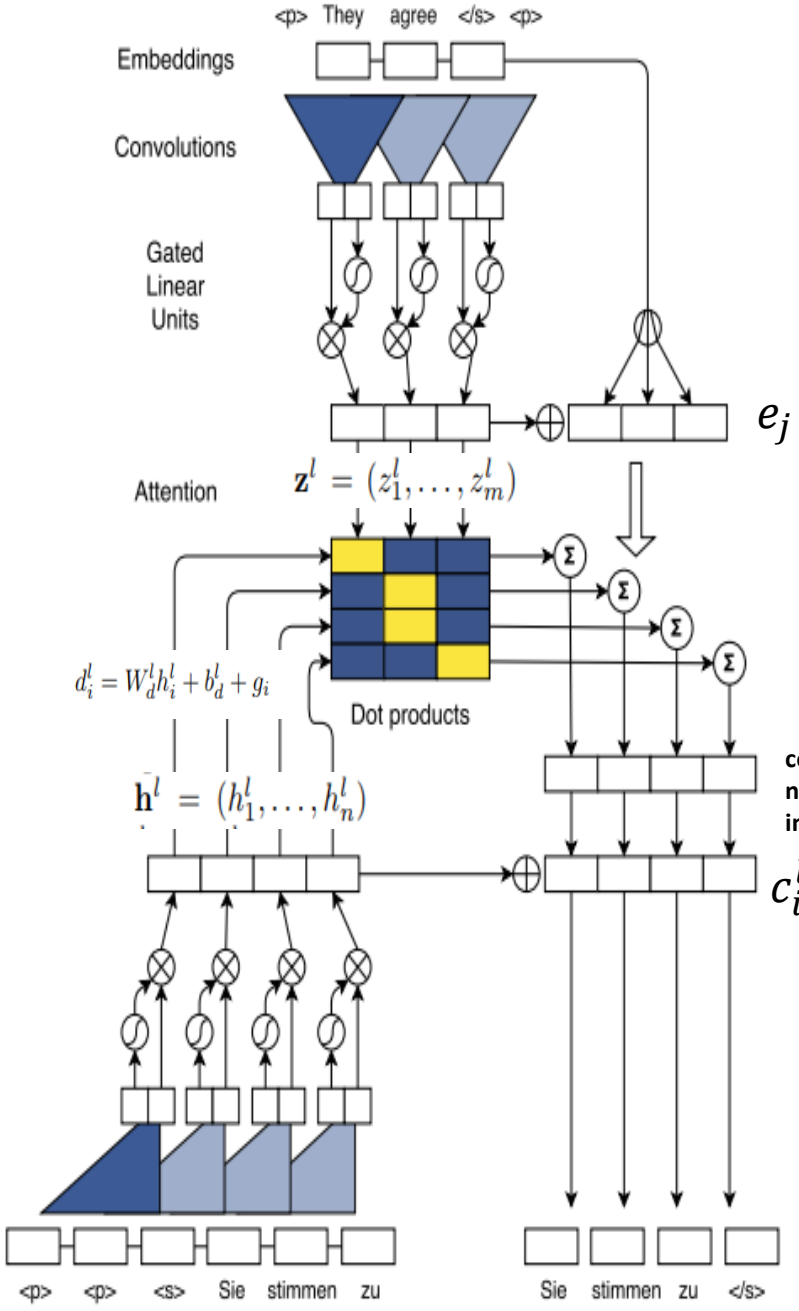


Figure 1. Illustration of batching during training. The English source sentence is encoded (top) and we compute all attention values for the four German target words (center) simultaneously. Our attentions are just dot products between decoder context representations (bottom left) and encoder representations. We add the conditional inputs computed by the attention (center right) to the decoder states which then predict the target words (bottom right). The sigmoid and multiplicative boxes illustrate Gated Linear Units.

$$a_{ij}^l = \frac{\exp\left(d_i^l \cdot z_j^u\right)}{\sum_{t=1}^{m} \exp\left(d_i^l \cdot z_t^u\right)}$$

conditional input $c_i^l = \sum_{j=1}^{m} a_{ij}^l (z_j^u + e_j)$ e_j:element input embedding

c_i^l: *added to* h_i^l

padding

Training phase

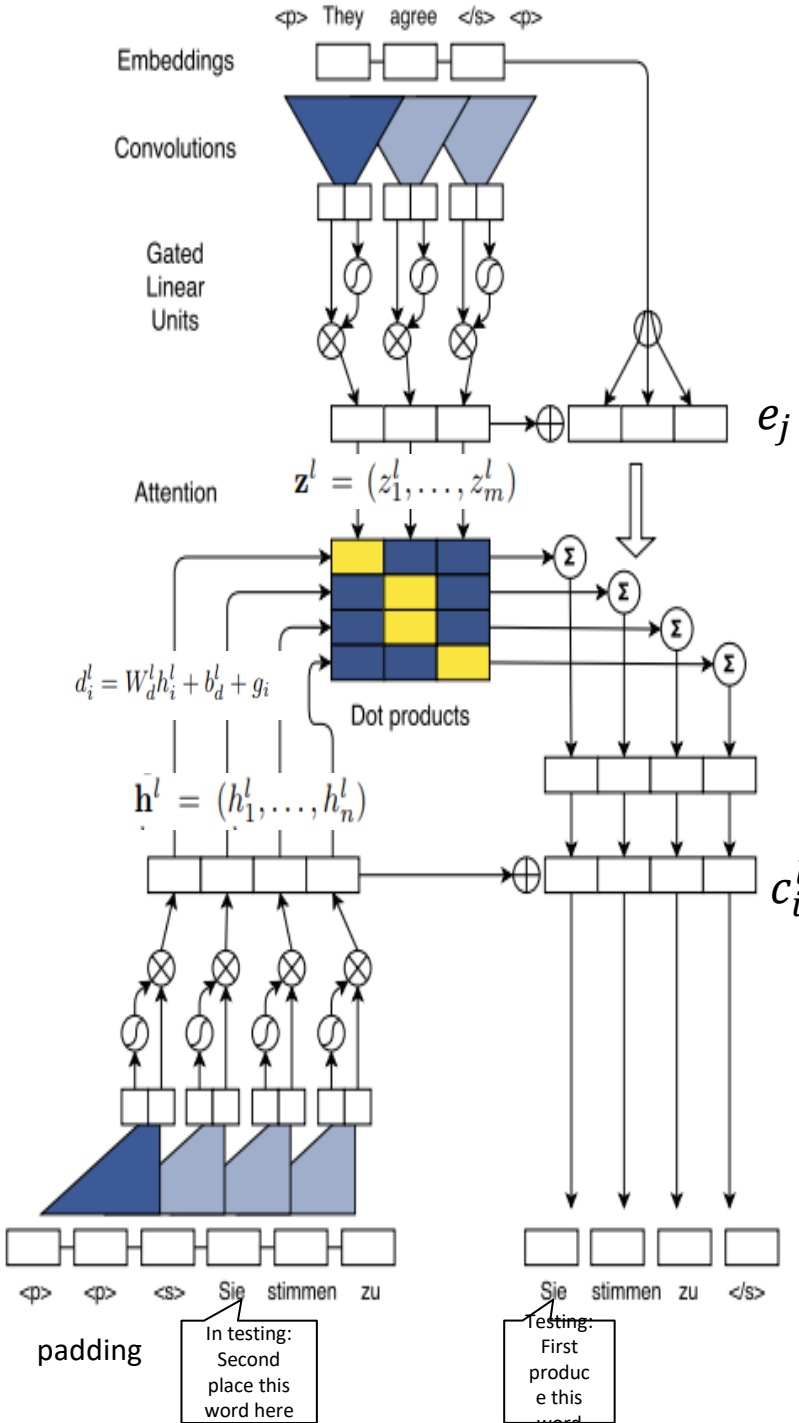


Figure 1. Illustration of batching during training. The English source sentence is encoded (top) and we compute all attention values for the four German target words (center) simultaneously. Our attentions are just dot products between decoder context representations (bottom left) and encoder representations. We add the conditional inputs computed by the attention (center right) to the decoder states which then predict the target words (bottom right). The sigmoid and multiplicative boxes illustrate Gated Linear Units.

$$a_{ij}^l = \frac{\exp\left(d_i^l \cdot z_j^u\right)}{\sum_{t=1}^m \exp\left(d_i^l \cdot z_t^u\right)}$$

$$c_i^l = \sum_{j=1}^m a_{ij}^l (z_j^u + e_j)$$

e_j:element input embedding

$\mathbf{h}^L$ final layer of the decoder just before the softmax

Output: $p(y_{i+1}|y_1, \ldots, y_i, \mathbf{x}) = \text{softmax}(W_o h_i^L + b_o) \in \mathbb{R}^T$

Multi-step attention (in testing)

In particular, the attention of the first layer determines a useful source context which is then fed to the second layer that takes this information into account when computing attention etc.
The decoder also has immediate access to the attention history of the k − 1 previous time steps because the conditional inputs

A trick: positional embedding

First, we embed input elements $\mathbf{x} = (x_1, \ldots, x_m)$ in distributional space as $\mathbf{w} = (w_1, \ldots, w_m)$, where $w_j \in \mathbb{R}^f$ is a column in an embedding matrix $\mathcal{D} \in \mathbb{R}^{V \times f}$. We also equip our model with a sense of order by embedding the absolute position of input elements $\mathbf{p} = (p_1, \ldots, p_m)$ where $p_j \in \mathbb{R}^f$. Both are combined to obtain input element representations $\mathbf{e} = (w_1 + p_1, \ldots, w_m + p_m)$.

- capturing a sense of order in a sequence
- This encoding gives the model a sense of which portion of the sequence of the input (or output) it is currently dealing with. The positional encoding can be learned, or fixed. Authors made tests (PPL, BLEU) showing that both: learned and fixed positional encodings perform similarly.

Architecture

encoder

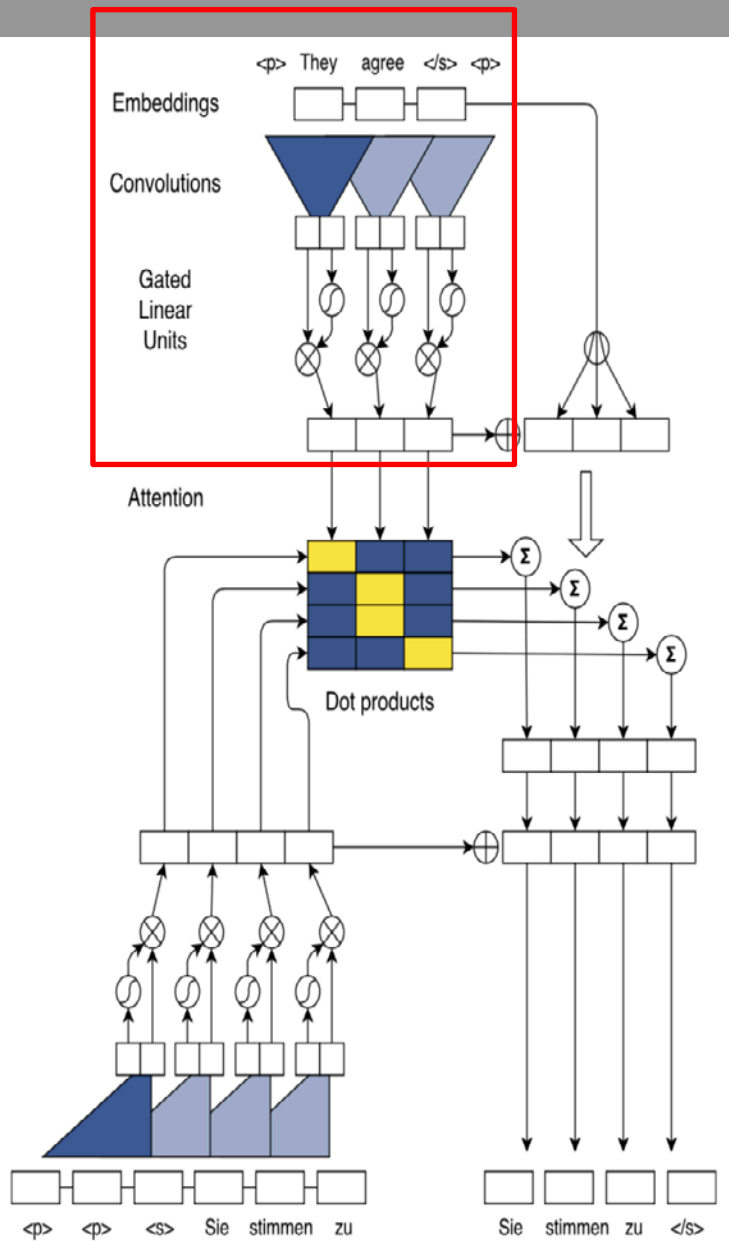
- Input: word + position $\mathbf{e} = (w_1 + p_1, \ldots, w_m + p_m)$
- Kernel parameter: W (kd × 2d), b(2d)
- GLU (gated linear units):

$$Y = [A\ B] \in \mathbb{R}^{2d}; \quad v([A\ B]) = A \otimes \sigma(B)$$

  - Non-linearity allow the networks to exploit full input field or to focus on fewer elements;
  - gated Sigmoid(b) control which inputs A are relevant

- $z_i^l = \nu\left(W^l \left[z_{i-\frac{k}{2}}^{l-1}, \ldots, z_{i+\frac{k}{2}}^{l-1}\right] + b_w^l\right) + z_i^{l-1}$

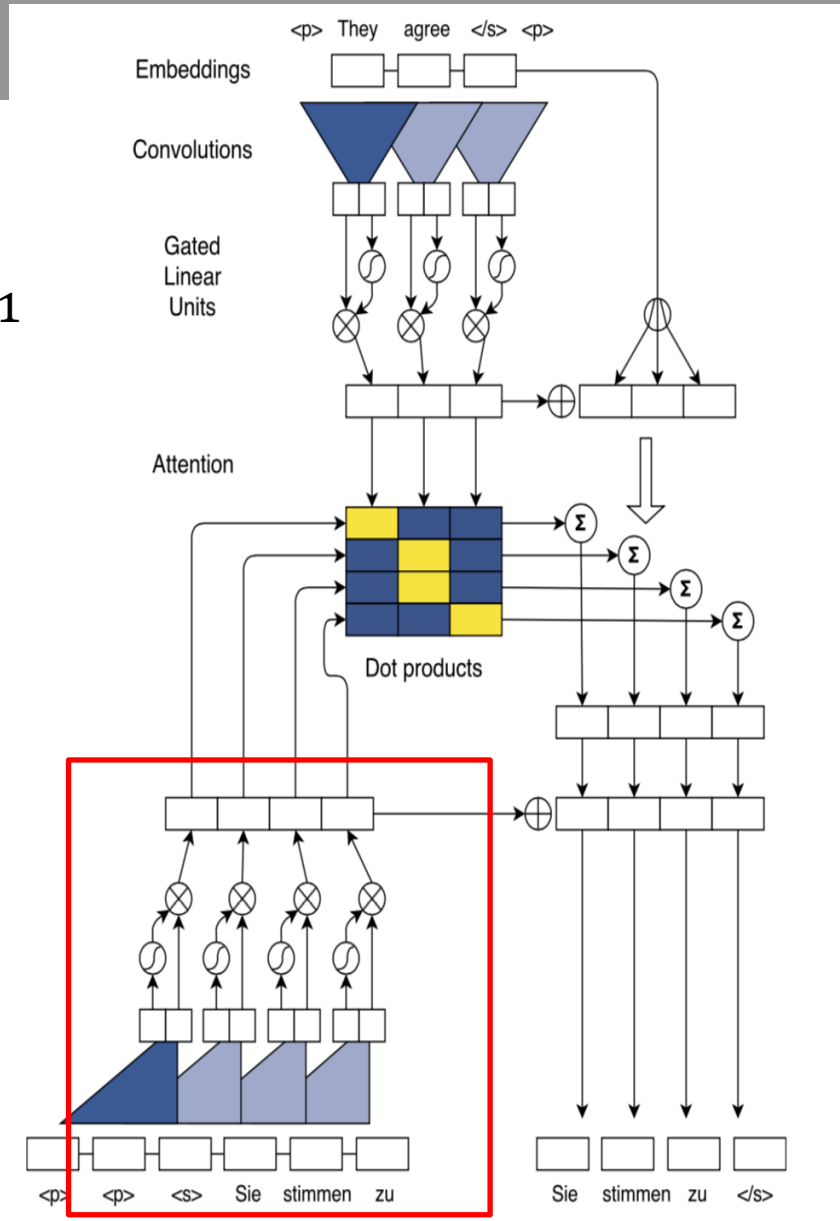
Residual connection



Architecture

- Decoder

$$h_i^l = \nu\left(W^l\left[h_{i-\frac{k}{2}}^{l-1}, \ldots, h_{i+\frac{k}{2}}^{l-1}\right] + b_w^l\right) + h_i^{l-1}$$



Architecture

- Attention
  - Separate attentions for each decoder layer

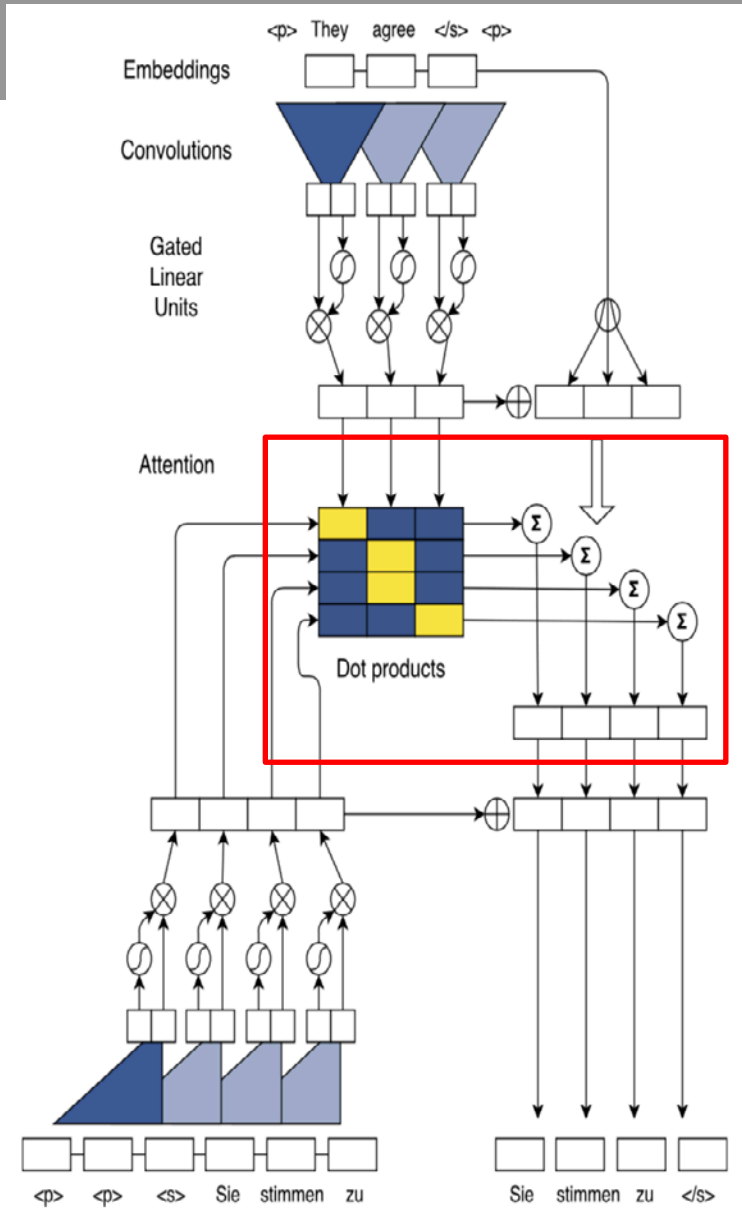
$$d_i^l = W_d^l h_i^l + b_d^l + g_i$$

$$a_{ij}^l = \frac{\exp\left(d_i^l \cdot z_j^u\right)}{\sum_{t=1}^{m} \exp\left(d_i^l \cdot z_t^u\right)}$$

$$c_i^l = \sum_{j=1}^{m} a_{ij}^l (z_j^u + e_j)$$

  - c is simply added to h

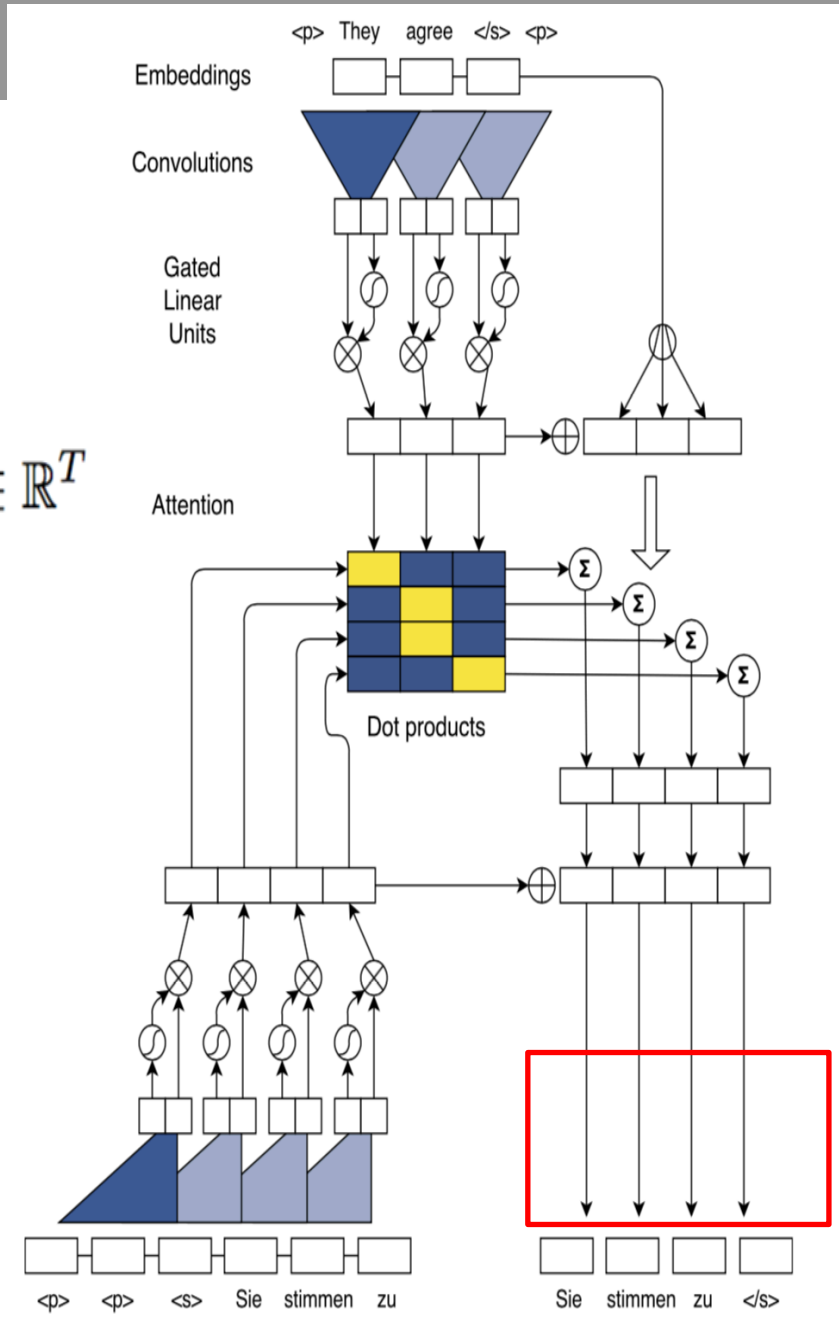
using current hidden state of the decoder and previous embedding. For decoder layer l the attention is calculated as a state I and source element j computed as a dot product between the decoder state summary d_i_l and each out of the last decoder step.



Architecture

Output

$$p(y_{i+1}|y_1, \ldots, y_i, \mathbf{x}) = \text{softmax}(W_o h_i^L + b_o) \in \mathbb{R}^T$$



- **RNN:**
- **Advantages:** are successful for variable-length representations such as sequences (e.g. languages), images, etc.
  Gating models LSTM or GRU are for long-range error propagation.
- **Problems:** The sequentiality prohibits parallelization within instances. Long-range dependencies still tricky, despite gating.
- **CNN:**
- **Advantages:** Trivial to parallelize (per layer) and fit intuition that most dependencies are local.
- **Problems:** Path length between positions can be logarithmic when using dilated convolutions, left-padding for text.
- **Solution:** Multi-head self-attention mechanism. Why attention? such attention networks can save 2–3 orders of magnitude of operations!

Attention-only Translation Models

- Problems with recurrent networks:
- Sequential training and inference: time grows in proportion to sentence length. Hard to parallelize.
- Long-range dependencies have to be remembered across many single time steps.
- Tricky to learn hierarchical structures ("car", "blue car", "into the blue car"…)

- Alternative:
- Convolution – but has other limitations.

Self-attention

- An attention mechanism relating different positions of a single sequence in order to compute a representation of the same sequence.
- Information flows from within the same subnetwork (either encoder or decoder).
- Convolution applies fixed transform weights. Self-attention applies variable weights

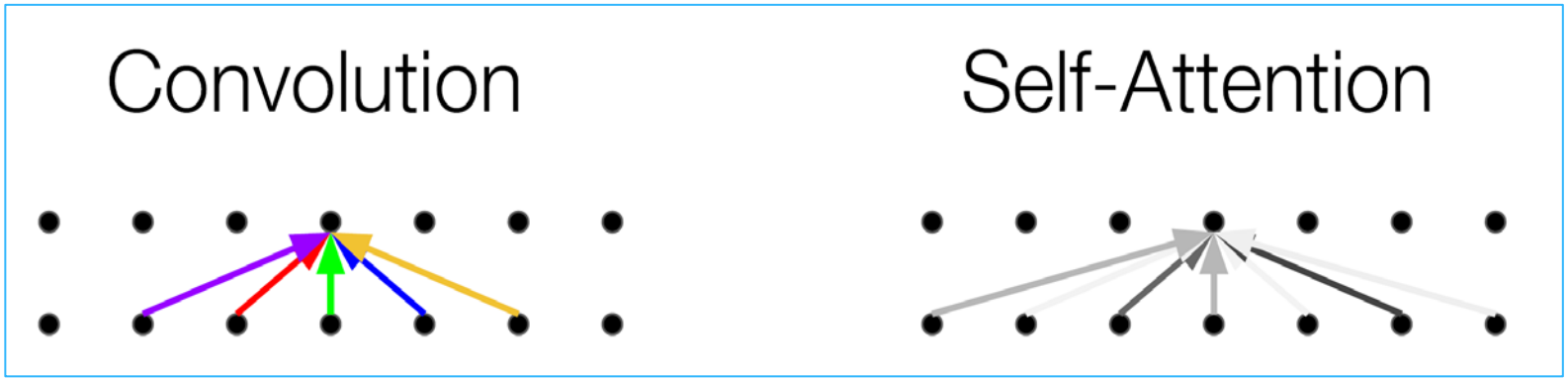


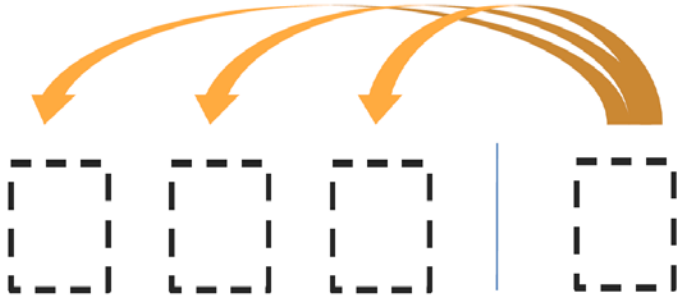
image from Lukas Kaiser, Stanford NLP seminar

Self-Attention “Transformers

- The Transformer starts by generating initial representations for each word.
- the Transformer performs a small, constant number of steps (chosen empirically).
- In each step, it applies a self-attention mechanism which directly models relationships between all words in a sentence.
- “I arrived at the bank after crossing the river”: to determine that the word “bank” refers to the shore of a river, the Transformer can learn to immediately attend to the word “river”.
- To compute the next representation for a given word - “bank”- the Transformer compares it to every other word in the sentence.
  This provides an attention score for every other word in the sentence.
  These attention scores determine how much each of the other words should contribute to the next representation of “bank”
- This self-attention step is then repeated multiple times in parallel for all words, successively generating new representations.

Vaswani et al. “Attention is all you need”, arXiv 2017

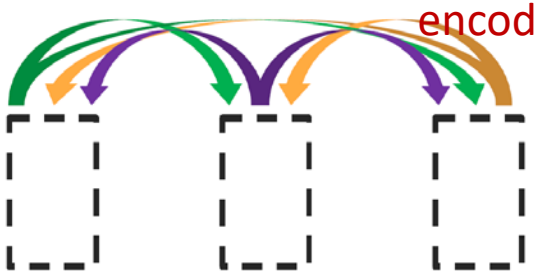
Attention in Transformer Networks



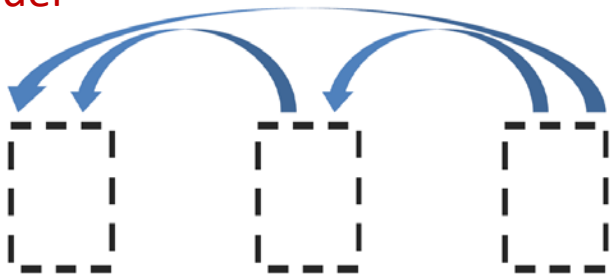
Encoder-Decoder Attention

Bahdanau
and Luong models

Replaces word recurrence in
encoder and decoder



Encoder Self-Attention



MaskedDecoder Self-Attention

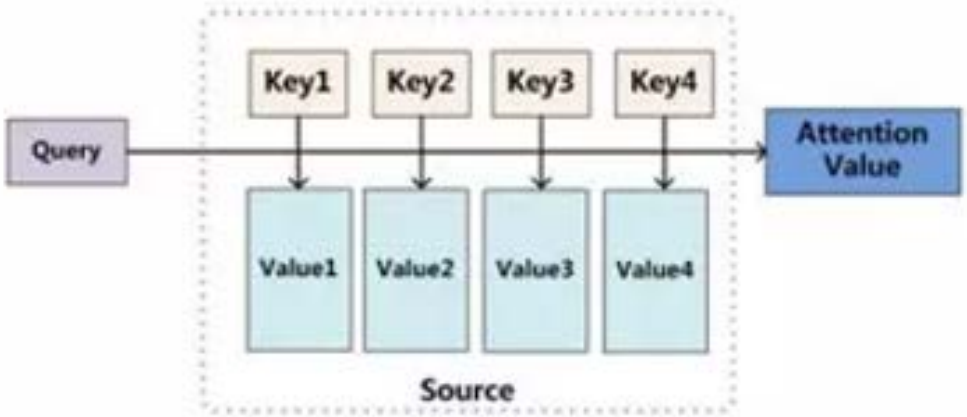
Masking limits attention to earlier units:
y_i depends only on y_j for $j < i$.

image from Lukas Kaiser, Stanford NLP seminar

Flow of Information

- we can visualize what other parts of a sentence the network attends to when processing or translating a given word, thus gaining insights into how information travels through the network.

- An attention function can be described as **mapping a query and a set of key-value pairs** to an output.



$$\text{Attention}(\text{Query}, \text{Source}) = \sum_{i=1}^{L_x} \text{Similarity}(\text{Query}, \text{Key}_i) * \text{Value}_i$$

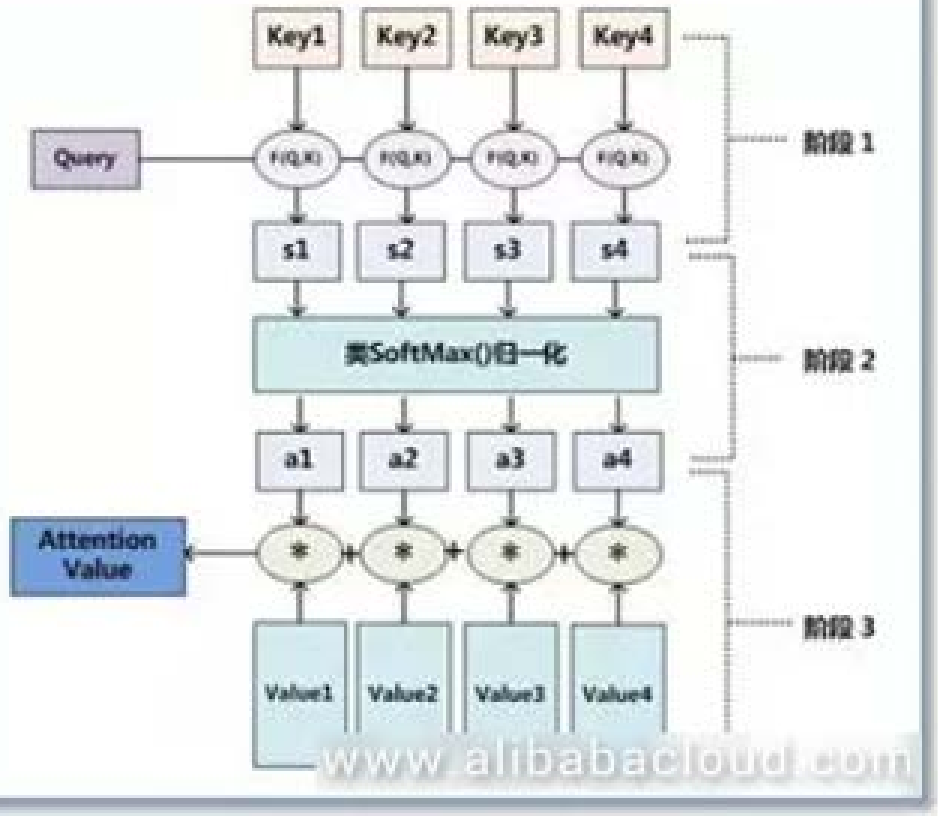
Calculate attention in three steps.

1. Take the query and each key and compute the similarity between the two to obtain a weight.
2. Use a softmax function to normalize these weights
3. weight these weights in conjunction with the corresponding values and obtain the final Attention.

$$f(Q,K_i)=\begin{cases} Q^T K_i & dot \\ Q^T W_a K_i & general \\ W_a[Q;K_i] & concat \\ v_a^T \tanh(W_a Q + U_a K_i) & perceptron \end{cases}$$

$$a_i = soft\max(f(Q,K_i)) = \frac{\exp(f(Q,K_i))}{\sum_j \exp(f(Q,K_j))}$$

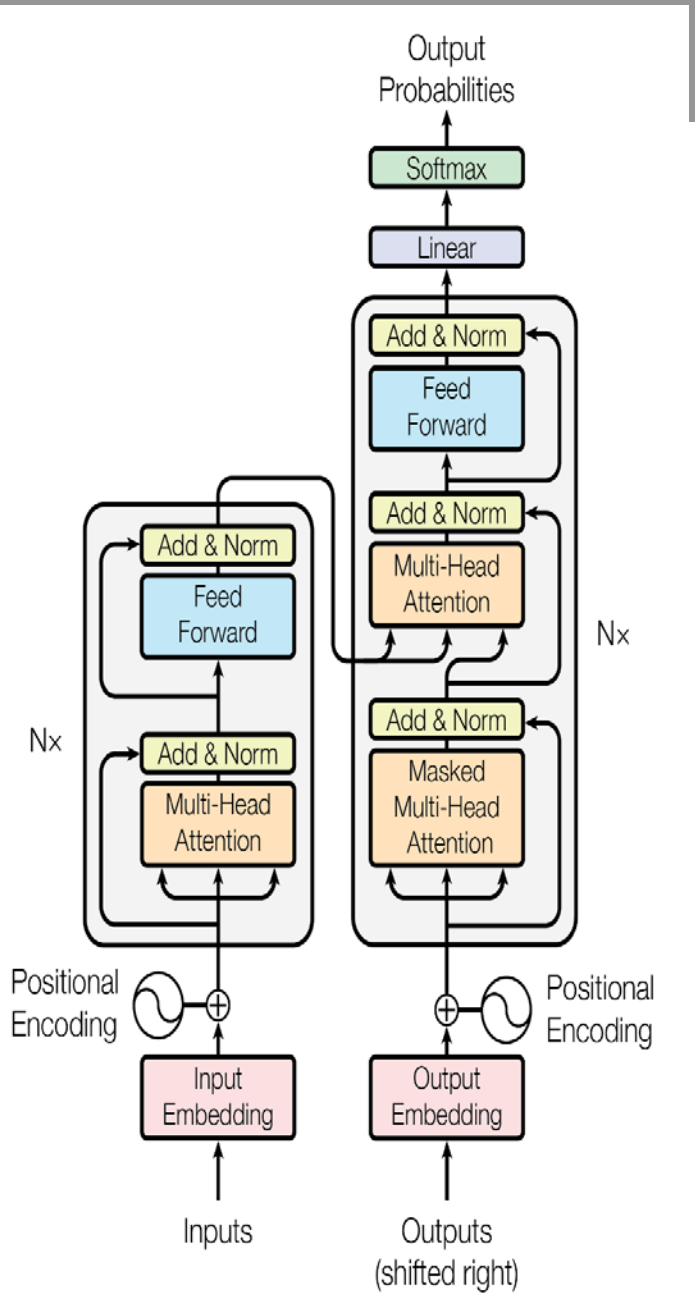
$$\text{Attention}(Q,K,V) = \sum_i a_i V_i$$



In current NLP work, the key and value are frequently the same, therefore key=value

The Transformer

- Encoder and Decoder Stacks
  - Attention
  - Position-wise FF Networks
  - Positional Encoding
  - Add & Norm



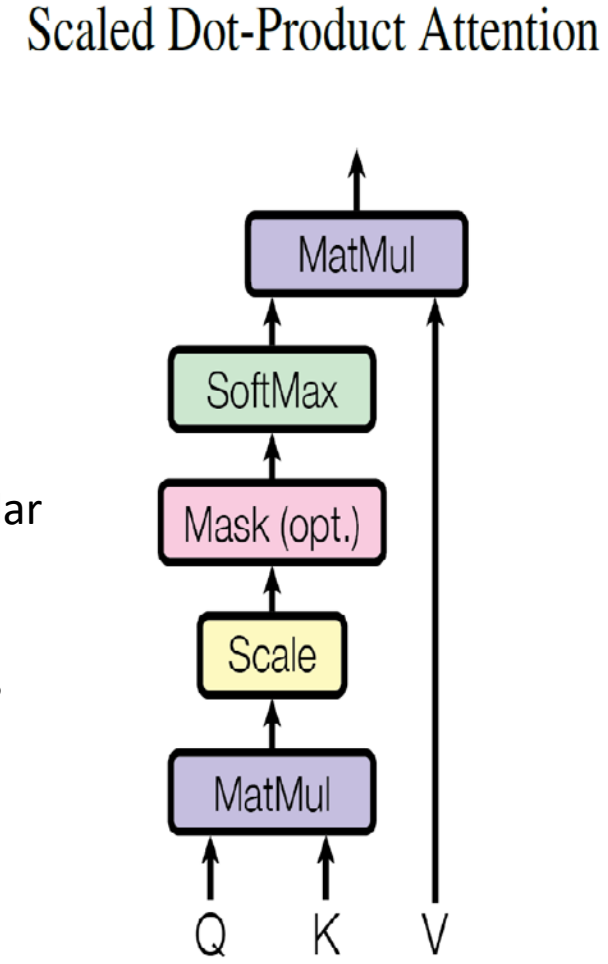
Attention Implementation

- Attention is modeled as a key-value store:
  Q = query vector
  K = key
  V = value

Encoder-decoder layer: the queries come from the previous decoder layer, and the memory keys and values come from the output of the encoder. (Similar to Bahdanau).

Self-attention layer: all of the keys, values and queries come from the output of the previous layer in the encoder.

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



Multi-Headed Attention

- Simple attention blends the results of all the attended-to inputs. It doesn't allow a per-input transformation, as convolution does.
- The solution is to use "multi-headed attention":

Convolution

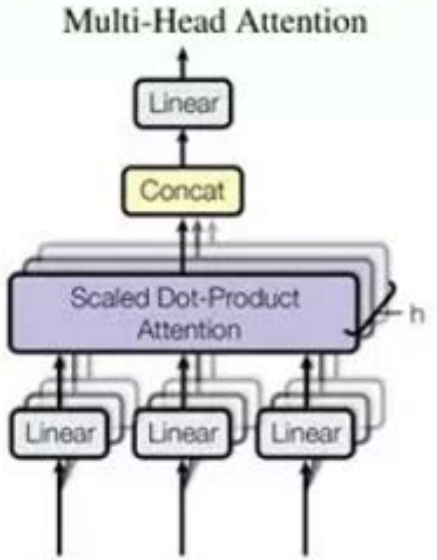
Multi-Head Attention

Multi-Head Attention

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_h)W^O$$

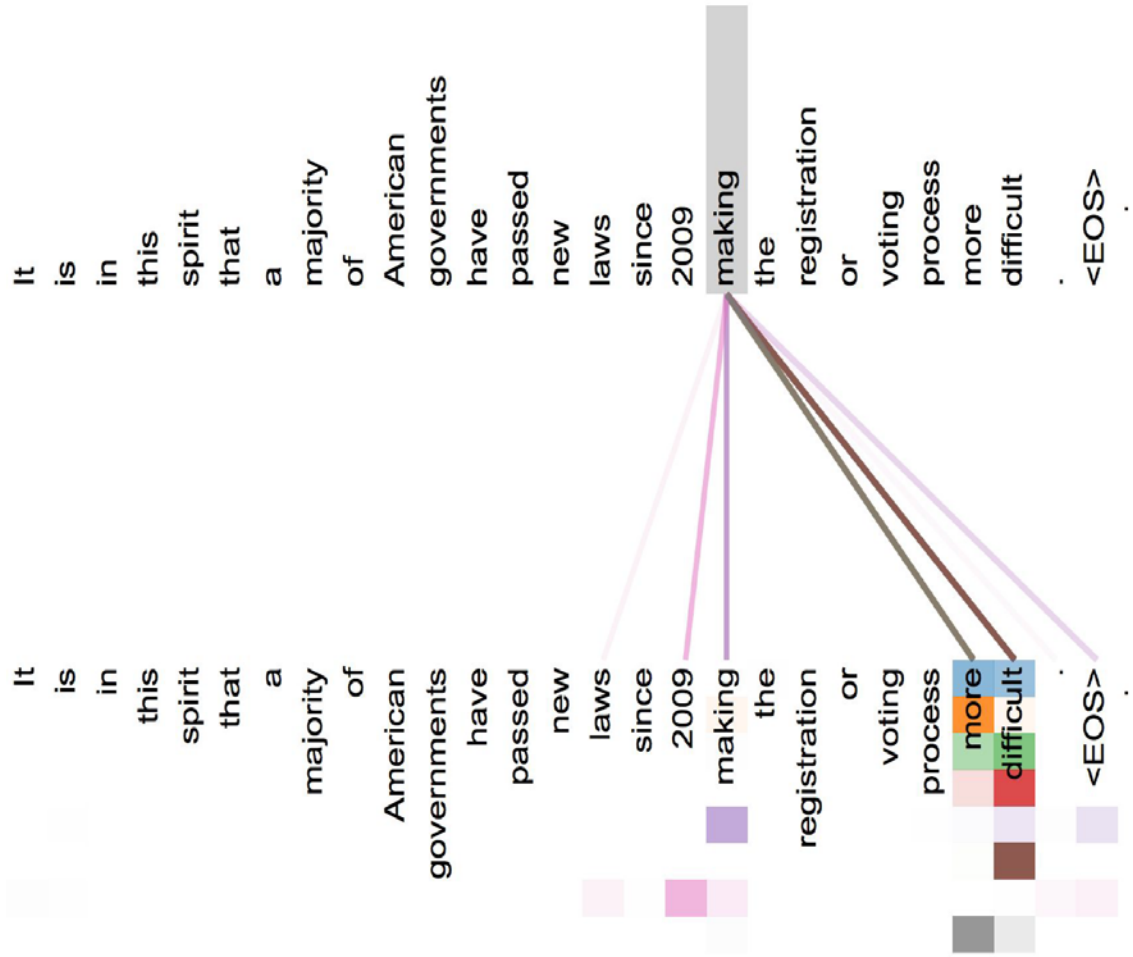
- Multi-head attention allows the model to jointly attend to **information from different representation subspaces** at different positions.



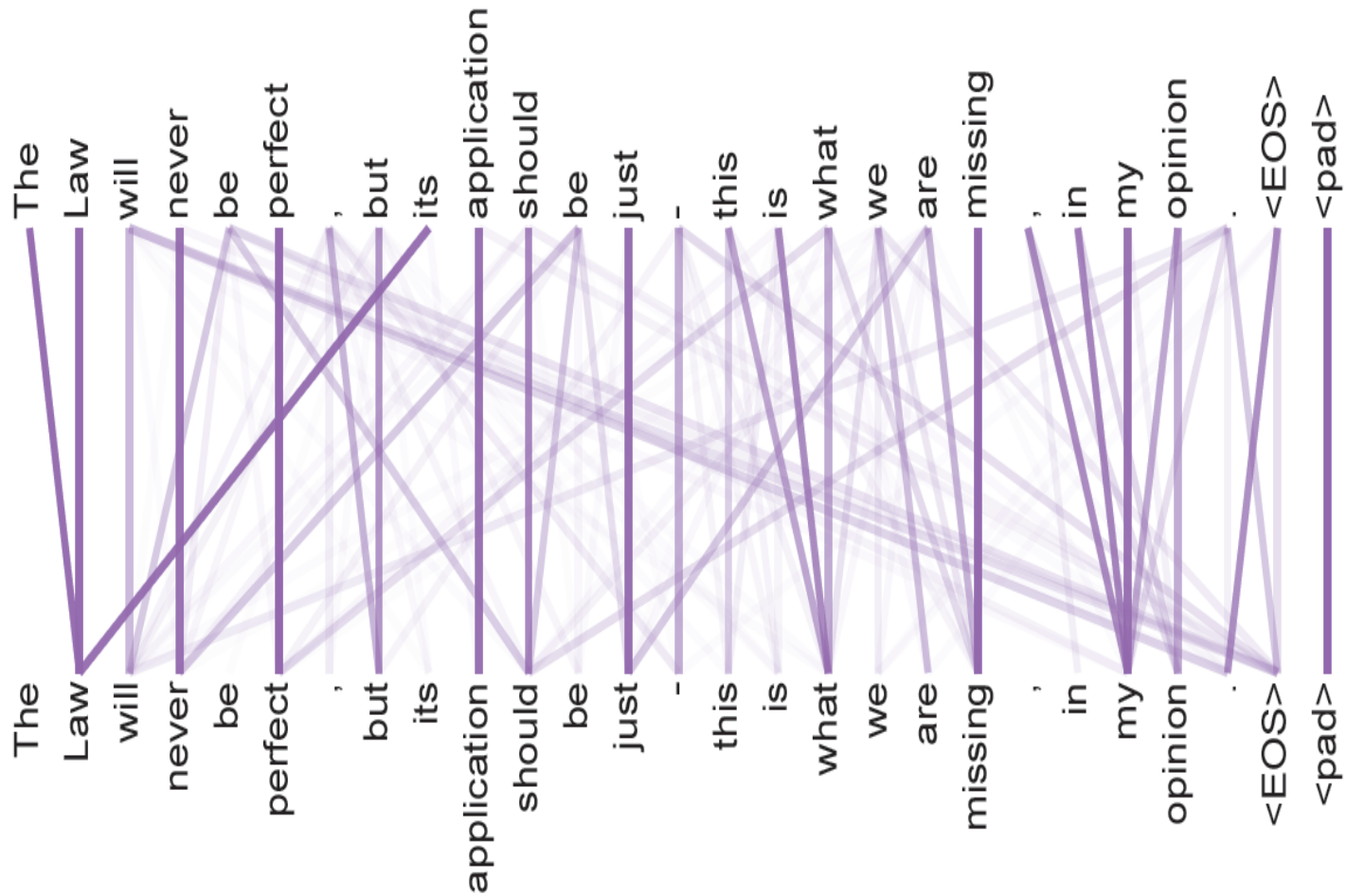
Positional Encoding

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

Multi-Headed Attention

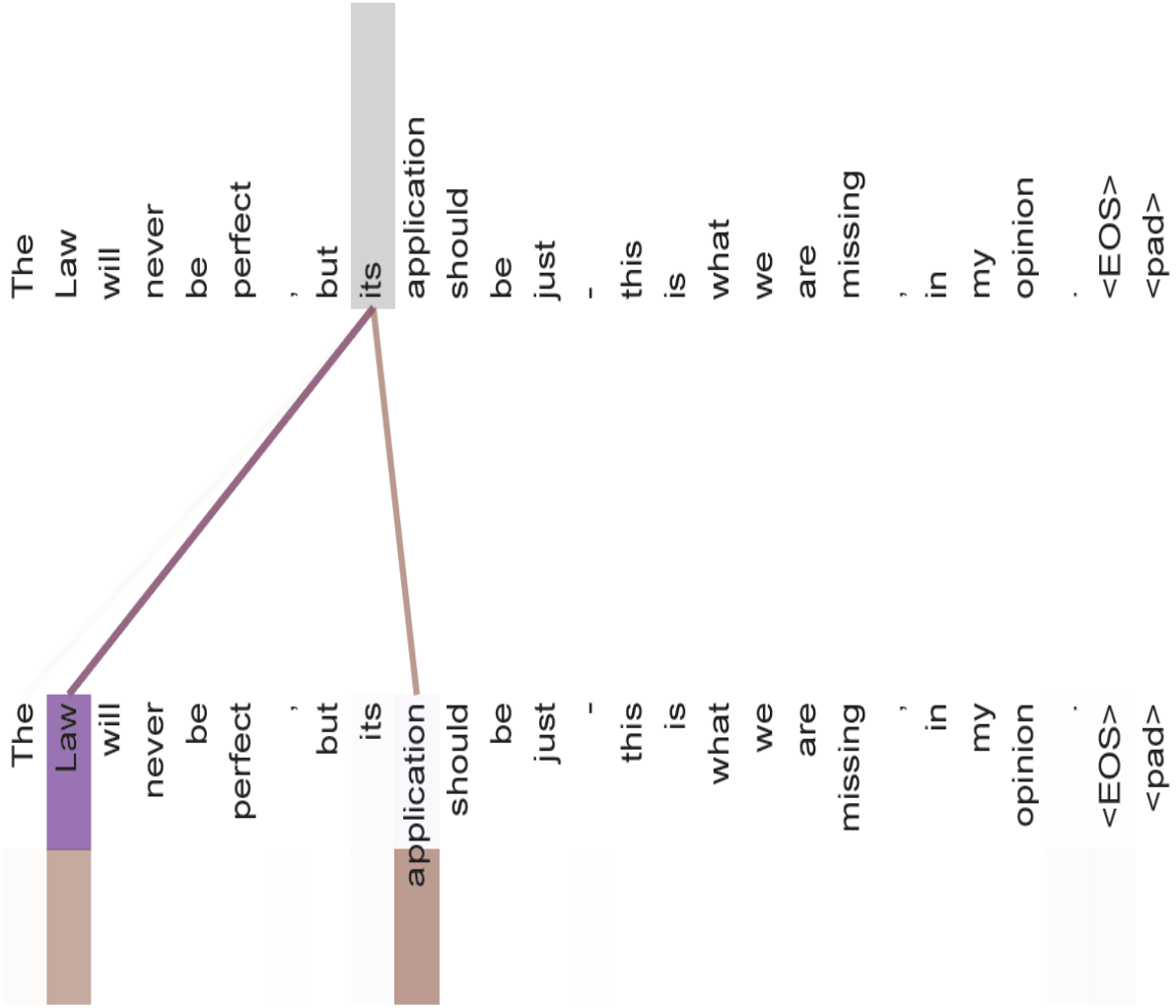


Multi-Headed Attention



Anaphora (pronoun or article) resolution

Multi-Headed Attention



Anaphora (pronoun or article) resolution

Attention and Interpretability

- Attention models learn to predict salient (important) inputs.
- Attention visualizations help users understand the causes of the network's behavior.
- Not every attended region is actually important, but post-processing can remove regions that aren't.
-