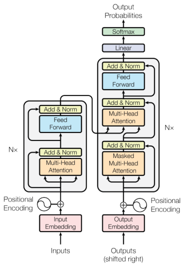
Transformer

The Encoder



Learning goals

- Understand Self-Attention and the role of position embeddings
- Understand all the subtleties of parallelized mult-head attention

ACKNOWLEDGMENTS

- This presentation is based on slides originally authored by:
  - Ben Roth
  - Christian Heumann
  - Goran Glavas
  - Ivan Habernal
  - Leonie Weissweiler
  - Matthias Assenmacher
  - Nina Poerner
- `https://slds-lmu.github.io/dl4nlp/`

THE TRANSFORMER ARCHITECTURE

(For simpler problems (e.g., classification, tagging), you would simply use the encoder.)

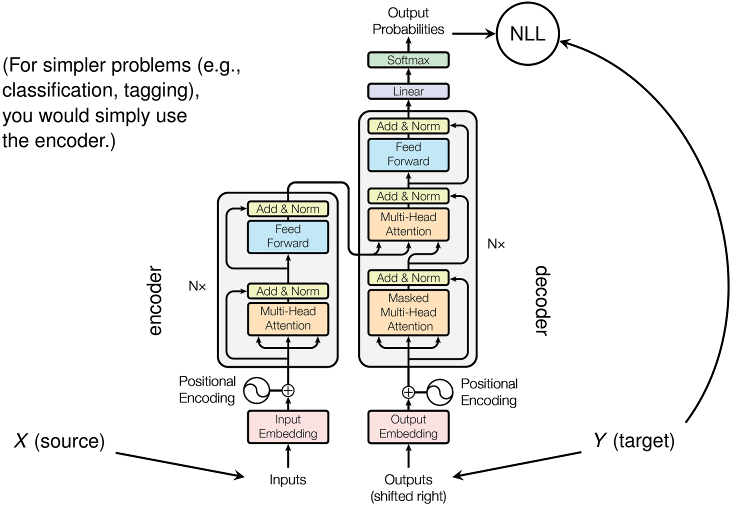


Figure from Vaswani et al. 2017: Attention is all you need

ATTENTION IN THE TRANSFORMER

- We can use attention on many different “things”, including:
  - The pixels of images
  - The nodes of knowledge graphs
  - The words of a vocabulary
- Here, we focus on scenarios where the query, key and value vectors represent tokens (e.g., words, characters, etc.) in sequences (e.g., sentences, paragraphs, etc.).

ATTENTION IN THE TRANSFORMER

Cross-attention:

- Let $X = (x_1 \dots x_{J_x})$, $Y = (y_1 \dots y_{J_y})$ be two sequences (e.g., source and target in a sequence-to-sequence problem)
- The query vectors represent tokens in Y and the key/value vectors represent tokens in X ("Y attends to X")

Self-attention:

- There is only one sequence $X = (x_1 \dots x_J)$
- The query, key and value vectors represent tokens in X ("X attends to itself")

CROSS-ATTENTION (1)

- Here, we describe cross-attention. Self-attention can easily be derived by assuming $\vec{X} = \vec{Y}$.
- Let $\vec{X} \in \mathbb{R}^{J_x \times d_x}$, $\vec{Y} \in \mathbb{R}^{J_y \times d_y}$ be representations of X, Y (e.g., stacked word embeddings, or the outputs of a previous layer)
- Let $\theta = \{\vec{W}^{(q)} \in \mathbb{R}^{d_y \times d_q}, \vec{W}^{(k)} \in \mathbb{R}^{d_x \times d_k}, \vec{W}^{(v)} \in \mathbb{R}^{d_x \times d_v}\}$ be trainable weight matrices
- We transform $\vec{Y}$ into a matrix of query vectors:

$$\vec{Q} = \vec{Y}\vec{W}^{(q)}$$

- We transform $\vec{X}$ into matrices of key and value vectors:

$$\vec{K} = \vec{X}\vec{W}^{(k)}; \qquad \vec{V} = \vec{X}\vec{W}^{(v)}$$

CROSS-ATTENTION (2)

- To calculate the e scores (step 1 of the basic recipe), Vaswani et al. use a parameter-less scaled dot product instead of Bahdanau's complicated FFN:

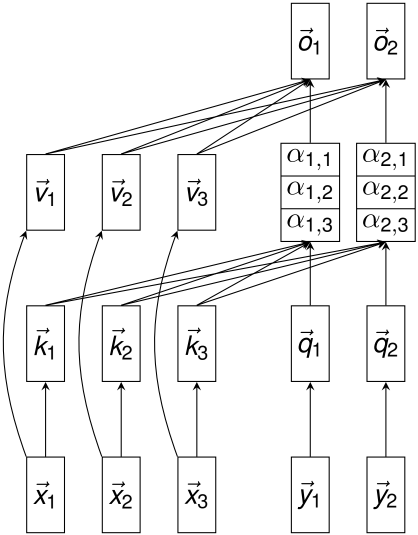
$$e_{j,j'} = a(\vec{q}_j, \vec{k}_{j'}) = \frac{\vec{q}_j^T \vec{k}_{j'}}{\sqrt{d_k}}$$

- Note: This requires that $d_q = d_k$
- Attention weights and outputs are defined like before (steps 2 and 3 of the basic recipe):

$$\alpha_{j,j'} = \frac{\exp(e_{j,j'})}{\sum_{j''=1}^{J_x} \exp(e_{j,j''})}$$

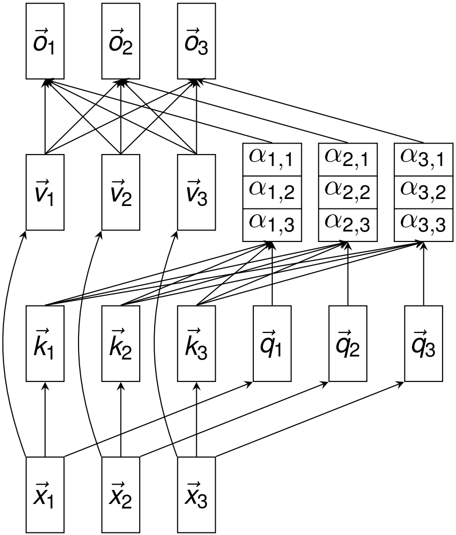
$$\vec{o}_j = \sum_{j'=1}^{J_x} \alpha_{j,j'} \vec{v}_{j'}$$

CROSS-ATTENTION (3)



Cross-attention

CROSS-ATTENTION (4)



Self-attention

SELF-ATTENTION FORMALIZED

- Let $\vec{X} \in \mathbb{R}^{J_x \times d_x}$ be a representation of X (e.g., stacked word embeddings, or the outputs of a previous layer)
- Let $\theta = \{\vec{W}^{(q)} \in \mathbb{R}^{d_x \times d_q}, \vec{W}^{(k)} \in \mathbb{R}^{d_x \times d_k}, \vec{W}^{(v)} \in \mathbb{R}^{d_x \times d_v}\}$ be trainable weight matrices
- We transform $\vec{X}$ into a matrix of query vectors:

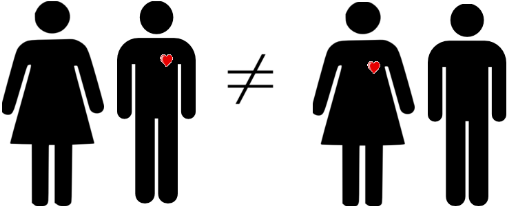
$$\vec{Q} = \vec{X}\vec{W}^{(q)}$$

- And we transform $\vec{X}$ into matrices of key and value vectors:

$$\vec{K} = \vec{X}\vec{W}^{(k)}; \qquad \vec{V} = \vec{X}\vec{W}^{(v)}$$

CAN SELF-ATTENTION MODEL WORD ORDER?

- Our model consists of a self-attention layer on top of a simple word embedding lookup layer. (For simplicity, we only consider one head, but this applies to multi-head attention as well.)
- Let $X^{(1)}$, $X^{(2)}$ be two sentences of the same length J, which contain the same words in a different order
- Example: "john loves mary" vs. "mary loves john"



CAN SELF-ATTENTION MODEL WORD ORDER?

- Definition of $\vec{o}_j$:

$$\vec{o}_j = \sum_{j'=1}^{J} \alpha_{j,j'} \vec{v}_{j'}$$

- Since addition is commutative, and the permutation is bijective, it is sufficient to show that:

$$\forall_{j \in \{1,\dots,J\}, j' \in \{1,\dots,J\}} \alpha^{(1)}_{j,j'} \vec{v}^{(1)}_{j'} = \alpha^{(2)}_{g_j, g_{j'}} \vec{v}^{(2)}_{g_{j'}}$$

- Step 1: Let's show that $\forall_j \vec{v}^{(1)}_j = \vec{v}^{(2)}_{g_j}$
- Definition of $\vec{v}_j$:

$$\vec{v}_j = \vec{W}^{(v)T} \vec{x}_j$$

- Then:

$$\vec{x}^{(1)}_j = \vec{x}^{(2)}_{g_j} \implies \vec{W}^{(v)T} \vec{x}^{(1)}_j = \vec{W}^{(v)T} \vec{x}^{(2)}_{g_j} \implies \vec{v}^{(1)}_j = \vec{v}^{(2)}_{g_j}$$

CAN SELF-ATTENTION MODEL WORD ORDER?

- Step 2: Let's show that $\forall_{j \in \{1,...,J\}, j' \in \{1,...,J\}} \alpha^{(1)}_{j,j'} = \alpha^{(2)}_{g_j, g_{j'}}$
- Definition of $\alpha_{j,j'}$:

$$\alpha_{j,j'} = \frac{\exp(e_{j,j'})}{\sum_{j''=1}^{J} \exp(e_{j,j''})}$$

- Since the sum in the denominator is commutative, and the permutation is bijective, it is sufficient to show that

$$\forall_{j \in \{1,...,J\}, j' \in \{1,...,J\}} e^{(1)}_{j,j'} = e^{(2)}_{g_j, g_{j'}}$$

CAN SELF-ATTENTION MODEL WORD ORDER?

- Definition of $e_{j,j'}$:

$$e_{j,j'} = \frac{1}{\sqrt{d_k}} \vec{q}_j^T \vec{k}_{j'} = \frac{1}{\sqrt{d_k}} (\vec{W}^{(q)T} \vec{x}_j)^T (\vec{W}^{(k)T} \vec{x}_{j'})$$

- Then:

$$\begin{aligned}
& \vec{x}_j^{(1)} = \vec{x}_{g_j}^{(2)} \wedge \vec{x}_{j'}^{(1)} = \vec{x}_{g_{j'}}^{(2)} \\
\Longrightarrow\ & \vec{W}^{(q)T} \vec{x}_j^{(1)} = \vec{W}^{(q)T} \vec{x}_{g_j}^{(2)} \wedge \vec{W}^{(k)T} \vec{x}_{j'}^{(1)} = \vec{W}^{(k)T} \vec{x}_{g_{j'}}^{(2)} \\
\Longrightarrow\ & \vec{q}_j^{(1)} = \vec{q}_{g_j}^{(2)} \wedge \vec{k}_{j'}^{(1)} = \vec{k}_{g_{j'}}^{(2)} \\
\Longrightarrow\ & \vec{q}_j^{(1)T} \vec{k}_{j'}^{(1)} = \vec{q}_{g_j}^{(2)T} \vec{k}_{g_{j'}}^{(2)} \\
\Longrightarrow\ & \frac{1}{\sqrt{d_k}} \vec{q}_j^{(1)T} \vec{k}_{j'}^{(1)} = \frac{1}{\sqrt{d_k}} \vec{q}_{g_j}^{(2)T} \vec{k}_{g_{j'}}^{(2)} \\
\Longrightarrow\ & e_{j,j'}^{(1)} = e_{g_j,g_{j'}}^{(2)}
\end{aligned}$$

CAN SELF-ATTENTION MODEL WORD ORDER?

- So, $\forall_j \vec{o}_j^{(1)} = \vec{o}_{g_j}^{(2)}$
- In other words: The representation of mary is identical to that of mary, and the representation of john is identical to that of john
- **Question:** Can the other layers in the Transformer architecture (feed-forward net, layer normalization) help with the problem?
  - No, because they apply the same function to all positions.
- **Question:** Would it help to apply more self-attention layers?
  - No. Since the representations of identical words are still identical in $\vec{O}$, the next self-attention layer will have the same problem.
- So... does that mean the Transformer is unusable?
- Luckily not. We just need to ensure that input embeddings of identical words at different positions are not identical.

POSITION EMBEDDINGS

- Add to every input word embedding a position embedding $\vec{p} \in \mathbb{R}^d$:
- Input embedding of word "mary" in position j: $\vec{x}_j = \vec{w}_{\mathcal{I}(\text{mary})} + \mathbf{p}_j$

$$\vec{w}_{\mathcal{I}(\text{mary})} + \vec{p}_j \neq \vec{w}_{\mathcal{I}(\text{mary})} + \vec{p}_{j'} \text{ if } j \neq j'$$

- Option 1 (Vaswani et al., 2017): Sinusoidal position embeddings (deterministic):

$$p_{j,i} = \begin{cases} \sin\left(\frac{j}{10000^{\frac{i}{d}}}\right) & \text{if } i \text{ is even} \\ \cos\left(\frac{j}{10000^{\frac{i-1}{d}}}\right) & \text{if } i \text{ is odd} \end{cases}$$

- Option 2 (Devlin et al., 2018):
  Trainable position embeddings: $\vec{P} \in \mathbb{R}^{J^{\max} \times d}$
  - Disadvantage:
    Cannot deal with sentences that are longer than $J^{\max}$

PARALLELIZED ATTENTION

- We want to apply our attention recipe to every query vector $\vec{q}_j$
- We could simply loop over all time steps $1 \leq j \leq J_x$ (or J_y) and calculate each $\vec{o}_j$ independently.
- Then stack all $\vec{o}_j$ into an output matrix $\vec{O} \in \mathbb{R}^{J_x \times d_v}$ (or $\mathbb{R}^{J_y \times d_v}$)
- But a loop does not use the GPU's capacity for parallelization
- So it might be unnecessarily slow

PARALLELIZED SELF-ATTENTION

- Do some inputs (e.g., $\vec{q}_j$) depend on previous outputs (e.g., $\vec{o}_{j-1}$)? If not, we can parallelize the loop into a single function:

$$\vec{O} = \mathcal{F}^{\text{attn}}(\vec{X}, \vec{X}; \theta)$$

- Attention in Transformers is usually parallelizable, unless we are doing autoregressive inference (more on that later).
- By the way: The Bahdanau model is not parallelizable in this way, because s_i (a.k.a. the query of the $i+1$'st step) depends on c_i (a.k.a. the attention output of the i'th step), see last lecture:

The hidden state s_i of the decoder given the annotations from the encoder is computed by

$$s_i = (1 - z_i) \circ s_{i-1} + z_i \circ \tilde{s}_i,$$

where

$$\tilde{s}_i = \tanh\left(W E y_{i-1} + U\left[r_i \circ s_{i-1}\right] + C c_i\right)$$
$$z_i = \sigma\left(W_z E y_{i-1} + U_z s_{i-1} + C_z c_i\right)$$
$$r_i = \sigma\left(W_r E y_{i-1} + U_r s_{i-1} + C_r c_i\right)$$

PARALLELIZED SELF-ATTENTION

- **Step 1**: The parallel application of the scaled dot product to all query-key pairs can be written as:

$$\vec{E} = \frac{\vec{Q}\vec{K}^T}{\sqrt{d_k}}; \quad \vec{E} \in \mathbb{R}^{J_x \times J_x}$$

$$\begin{matrix} \\ \downarrow \\ \textit{queries} \\ \downarrow \end{matrix} \overset{\rightarrow \textit{keys} \rightarrow}{\begin{bmatrix} e_{1,1} & \dots & e_{1,J_x} \\ \vdots & \ddots & \vdots \\ e_{J_x,1} & \dots & e_{J_x,J_x} \end{bmatrix}} = \frac{1}{\sqrt{d_k}} \begin{bmatrix} - & \vec{q}_1 & - \\ & \vdots & \\ - & \vec{q}_{J_x} & - \end{bmatrix} \begin{bmatrix} | & & | \\ \vec{k}_1 & \dots & \vec{k}_{J_x} \\ | & & | \end{bmatrix}$$

PARALLELIZED SCALED DOT PRODUCT SELF-ATTENTION

- **Step 2**: Softmax with normalization over the second axis (key axis):

$$\alpha_{j,j'} = \frac{\exp(e_{j,j'})}{\sum_{j''=1}^{J_x} \exp(e_{j,j''})}$$

- Let's call this new normalized matrix $\vec{A} \in (0,1)^{J_x \times J_x}$
- The rows of $\vec{A}$, denoted $\vec{\alpha}_j$, are probability distributions (one $\vec{\alpha}_j$ per $\vec{q}_j$)

PARALLELIZED SCALED DOT PRODUCT SELF-ATTENTION

- **Step 3**: Weighted sum

$$\vec{O} = \vec{A}\vec{V}; \vec{O} \in \mathbb{R}^{J_x \times d_v}$$

$$\begin{array}{c} \downarrow \\ \textit{queries} \\ \downarrow \end{array} \overset{\rightarrow d_v \text{(value dims)} \rightarrow}{\begin{bmatrix} o_{1,1} & \dots & o_{1,d_v} \\ \vdots & \ddots & \vdots \\ o_{J_x,1} & \dots & o_{J_x,d_v} \end{bmatrix}} = \begin{bmatrix} - & \boldsymbol{\alpha}_1 & - \\ & \vdots & \\ - & \boldsymbol{\alpha}_{J_x} & - \end{bmatrix} \begin{bmatrix} | & & | \\ \vec{v}_{:,1} & \dots & \vec{v}_{:,d_v} \\ | & & | \end{bmatrix}$$

... AS A ONE-LINER

$$\vec{O} = \mathcal{F}^{\mathrm{attn}}(\vec{X}, \vec{X}; \theta) = \mathrm{softmax}\Big(\frac{(\vec{X}\vec{W}^{(q)})(\vec{X}\vec{W}^{(k)})^T}{\sqrt{d_k}}\Big)(\vec{X}\vec{W}^{(v)})$$

- GPUs like matrix multiplications
  $\rightarrow$ usually a lot faster than RNN!
- But: The memory requirements of $\vec{E}$ and $\vec{A}$ are $\mathcal{O}(J_x^2)$
- A length up to about 500 is usually ok on a medium-sized GPU (and most sentences are shorter than that anyway).
- But when we consider inputs that span several sentences (e.g., paragraphs or whole documents), we need tricks to reduce memory. These are beyond the scope of this lecture.

ADD-ON: CHANGES FOR CROSS ATTENTION

- **Step 1**: Scaled dot-product

$$\vec{E} = \frac{\vec{Q}\vec{K}^T}{\sqrt{d_k}}; \quad \vec{E} \in \mathbb{R}^{J_y \times J_x}$$

- **Step 2**: Softmax

$$\alpha_{j,j'} = \frac{\exp(e_{j,j'})}{\sum_{j''=1}^{J_x} \exp(e_{j,j''})}$$

- **Step 3**: Output

$$\vec{O} = \vec{A}\vec{V}; \vec{O} \in \mathbb{R}^{J_y \times d_v}$$

- As one-liner:

$$\vec{O} = \mathcal{F}^{\text{attn}}(\vec{X}, \vec{Y}; \theta) = \text{softmax}\Big(\frac{(\vec{Y}\vec{W}^{(q)})(\vec{X}\vec{W}^{(k)})^T}{\sqrt{d_k}}\Big)(\vec{X}\vec{W}^{(v)})$$

MULTI-LAYER ATTENTION

- Sequential application of several attention layers, with separate parameters $\{\theta^{(1)} \dots \theta^{(N)}\}$
- In Transformer: sequential application of Transformer blocks
- There are some additional position-wise layers inside the Transformer block, i.e., $\vec{O}^{(n)}$ undergoes some additional transformations before becoming the input to the next Transformer block $n + 1$

MULTI-HEAD ATTENTION

- Application of several attention layers ("heads") in parallel
- M sets of parameters $\{\theta^{(1)}, \ldots, \theta^{(M)}\}$, with $\theta^{(m)} = \{\vec{W}^{(m,q)}, \vec{W}^{(m,k)}, \vec{W}^{(m,v)}\}$
- For every head, compute in parallel:

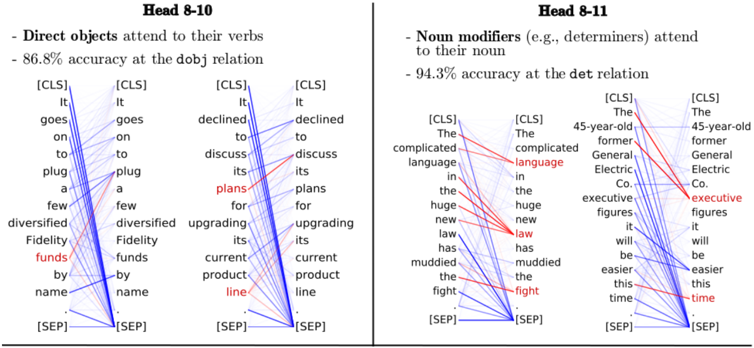
$$\vec{O}^{(m)} = \mathcal{F}^{\mathrm{attn}}(\vec{X}, \vec{Y}; \theta^{(m)})$$

- Concatenate all $\vec{O}^{(m)}$ along their last axis; then down-project the concatenation with an additional parameter matrix $\vec{W}^{(o)} \in \mathbb{R}^{Md_v \times d_v}$:

$$\vec{O} = [\vec{O}^{(1)}; \ldots; \vec{O}^{(M)}]\vec{W}^{(o)}$$

MULTI-HEAD ATTENTION

- Conceptually, multi-head attention is to single-head attention like a filter bank is to a single filter (Lecture on CNNs)
- Division of labor: different heads model different kinds of inter-word relationships



Clark et al. (2018): What Does BERT Look At? An Analysis of BERT's Attention