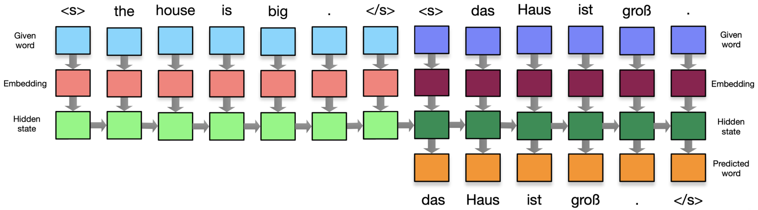
Attention is All You Need

Bill Watson

S&P Global

November 18, 2019

Recap: Encoder-Decoder Models



- **Encoders:** Give the source sentence meaning
- **Decoders:** Emit a variable-length sequence
- **Seq2Seq** models rely on propagating the hidden state for generation

How can we improve this model?

- Add an **alignment model** to force the model to 'attend' to different parts of the input
- Gives the decoder direct access to the encodings to make generations
- Known as a **context vector**

What are Context Vectors?

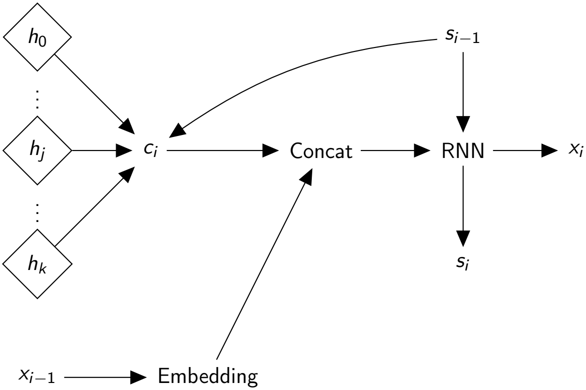
$$c_i = \sum_j \alpha_{ij} \cdot h_j$$

- A **context vector** c_i is the weighted sum of the encodings for the i-th iteration of the decoder
- s_{i-1} is the previous hidden state of the decoder
- h_j is the encoding for the j-th input word
- α_{ij} is the weight associated for the j-th input word for the i-th deocding iteration

How to use Context Vectors?

- No general rule
- Usually, **concat** the token embedding with context vector c_i
- Pass this as the input to the RNN, along with the previous decoder state s_{i-1} as the hidden state

Visual



(Simple) Attention Mechanisms

Mean Attention

$$\alpha_{ij} = \frac{1}{S}$$

$$c_i = \frac{1}{S}\sum_{j}^{S} h_j = \sum_{j} \alpha_{ij} \cdot h_j$$

- Very simple and naive approach is to apply uniform weights to each encoding h_j
- Fancy way of saying average
- S is the length of the source sentence

Location Based: Laplace

$$\alpha_{ij} = f(j \mid i, b) = \frac{1}{2b} \exp\left(-\frac{|j - i|}{b}\right)$$

$$c_i = \sum_j \alpha_{ij} \cdot h_j$$

- Weighted by location in sequence
  - Center a Laplace at $\mu = i$, scale b
  - Weight is the probability of position j relative to i
- Penalizes elements away from the diagonal
- Heavier tail than a Gaussian

Location Based: Gaussian

$$\alpha_{ij} = f\left(j \mid i, \sigma^2\right) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(j-i)^2}{2\sigma^2}\right)$$

$$c_i = \sum_j \alpha_{ij} \cdot h_j$$

- Weighted by location in sequence
  - Center a Gaussian at $\mu = i$, scale σ
  - Weight is the probability of position j relative to i
- Smoother distribution closer to the diagonal
- Smaller tail than Laplace, so outliers are penalized more

Review: Laplace and Gaussian

Visualizing Location-Based Distributions

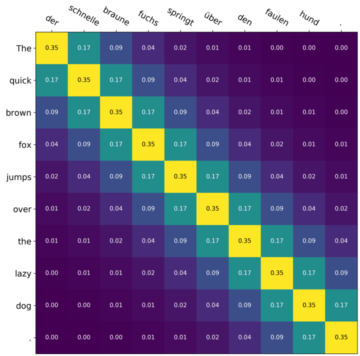


Figure: Laplace

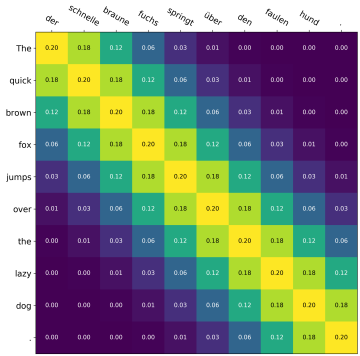


Figure: Gaussian

(Parameter-based) Attention Mechanisms

Parameter-based Attention

- Why not use learnable weights to effectively mix the encodings?
- Let deep learning figure out the alignment
- We now define a scoring function conditioned on the previous decoder state s_{i-1} and input encodings h_j

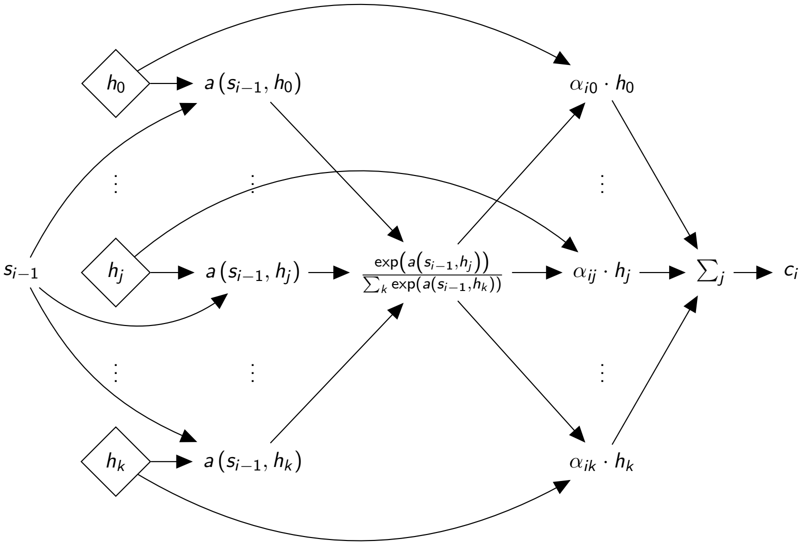
$$a\left(s_{i-1}, h_j\right) = ?$$

- In order to constrain the weights to a valid distribution, we softmax

$$\alpha_{ij} = \frac{\exp\left(a\left(s_{i-1}, h_j\right)\right)}{\sum_k \exp\left(a\left(s_{i-1}, h_k\right)\right)}$$

$$c_i = \sum_j \alpha_{ij} \cdot h_j$$

Attention Computation Graph

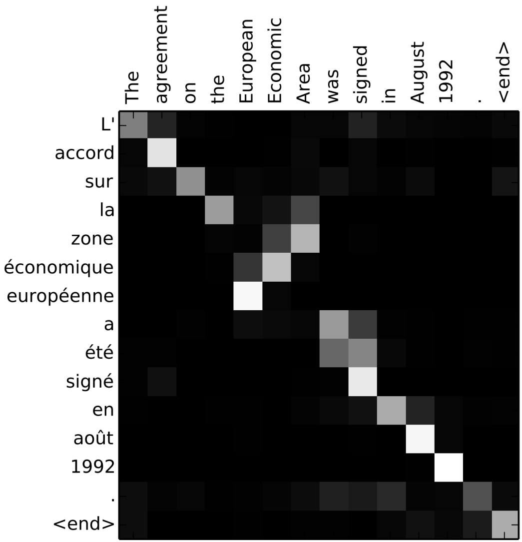


Bahdanau Attention

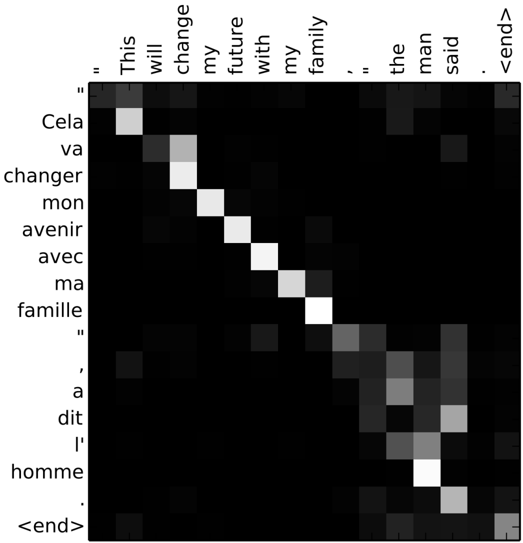
$$a\left(s_{i-1}, h_j\right) = v_a^T \tanh\left(W_a s_{i-1} + U_a h_j + b\right)$$

- W_a, U_a, are learnable matrices
- v_a, b are learnable vectors
- Output is a scalar value
- AKA: Additive Attention, Concat Attention

Bahdanau Attention Visual



Bahdanau Attention Visual



Luong Attention: Dot Product

$$a\left(s_{i-1}, h_j\right) = s_{i-1}^T h_j$$

- Notice: No Learnable Parameters
- Simple dot product between decoder state and encoding
- Output is a scalar value

Luong Attention: Scaled Dot Product

$$a\left(s_{i-1}, h_j\right) = \frac{1}{\sqrt{|h_j|}} s_{i-1}^T h_j$$

- Rescale the dot product accoridng to the dimensionality of the vectors
- **Why?** Because for large dimensionality, the dot-product is large in magnitude, resulting in small gradients from the softmax
- Output is a scalar value

Luong Attention: General

$$a\left(s_{i-1}, h_j\right) = s_{i-1}^T W_a h_j$$

- Allow a learnable matrix W_a inbetween the states
- AKA a Bilinear Layer: $x_0^T A x_1$
- Output is a scalar value

Luong Attention: Location Attention

$$a\left(s_{i-1}, h_j\right) = W_a h_j$$

- An approach to reweight exclusively on the decoder state
- Single learnable matrix W_a
- Output is scalar

Luong Attention Visual

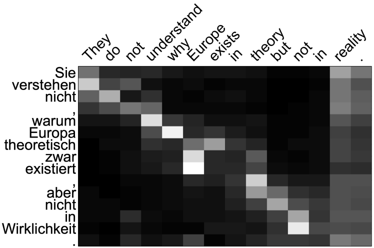


Figure: Luong Attention

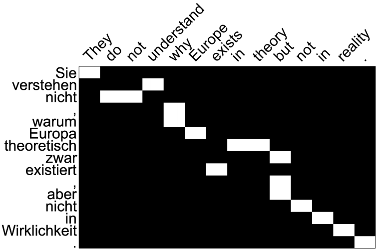


Figure: Gold Alignments

(Advanced) Attention Mechanisms

Luong Attention: Local Attention

- What if we focused on a window instead of all the vectors?
- This approach is known as *local* attention
- Before, we were using *global* attention

Luong Attention: Local Attention

- Define a window $\mathcal{W}$ of size $2D$ at alignment point p_t:

$$\mathcal{W} = [p_t - D, p_t + D]$$

- Our context vector is now an average over this window

$$c_i = \sum_{j \in \mathcal{W}} \alpha_{ij} \cdot h_j$$

Luong Attention: Local Attention

- How do we select p_t, the alignment position?
- **Monotonic** alignment - set $p_t = t$
  - Assumes `src` and `trg` are roughly monotonically aligned
- **Predictive** alignment - predict the aligned position

Luong Attention: Local Predictive Attention

$$p_t = S \cdot \sigma\left(v_p^T \tanh\left(W_p \cdot s_{i-1}\right)\right)$$

- **Predictive** alignment allows for learnable positions
- S is the length of the source sentence
- W_p and v_p are learnable
- Sigmoid enforces the constraint $p_t \in [0, S]$

Luong Attention: Local Predictive Attention

$$a_p\left(s_{i-1}, h_j\right) = a\left(s_{i-1}, h_j\right) \cdot \exp\left(-\frac{\left(j - p_t\right)^2}{2\sigma^2}\right)$$

- Use a gaussian penalty to favor words near p_t
- Note: $p_t \in \mathbb{R}$, $j \in \mathbb{Z}$
- j is the *integer* position of the encoding
- $a\left(s_{i-1}, h_j\right)$ is any previous attention mechanism

Luong Attention Visuals

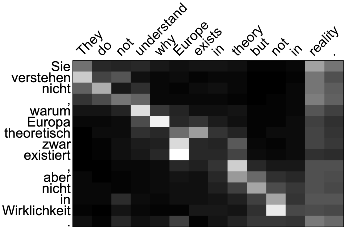


Figure: Global Attention

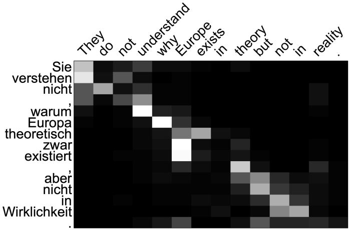


Figure: Local-M

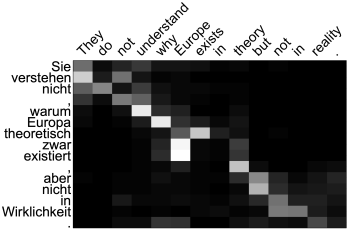


Figure: Local-P

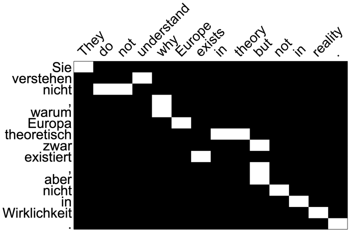


Figure: Gold Alignments

Fine-Grained Attention

$$
\begin{aligned}
a^k\left(s_{i-1}, h_j\right) &= \mathrm{FF}^k(s_{i-1}, h_j) \\
\alpha_{ij}^d &= \frac{\exp\left(a^d\left(s_{i-1}, h_j\right)\right)}{\sum_k \exp\left(a^d\left(s_{i-1}, h_k\right)\right)} \\
c_i &= \sum_j \alpha_{ij} \odot h_j
\end{aligned}
$$

- What's different?
- Instead of a scalar scoring function, have it output a weight *per* dimension
- Now a weighted average on the feature dimension *instead* of the entire encoding

Fine-Grained Attention

$$c_i = 0.64 \cdot \begin{bmatrix} 0.17 \\ 0.53 \\ -0.22 \end{bmatrix} + 0.12 \cdot \begin{bmatrix} 1.34 \\ 0.39 \\ 0.45 \end{bmatrix} + 0.24 \cdot \begin{bmatrix} -0.08 \\ -1.39 \\ 1.21 \end{bmatrix}$$

$$c_i = \begin{bmatrix} 0.59 \\ 0.18 \\ 0.16 \end{bmatrix} \cdot \begin{bmatrix} 0.17 \\ 0.53 \\ -0.22 \end{bmatrix} + \begin{bmatrix} 0.09 \\ 0.36 \\ 0.74 \end{bmatrix} \cdot \begin{bmatrix} 1.34 \\ 0.39 \\ 0.45 \end{bmatrix} + \begin{bmatrix} 0.32 \\ 0.46 \\ 0.10 \end{bmatrix} \cdot \begin{bmatrix} -0.08 \\ -1.39 \\ 1.21 \end{bmatrix}$$

Multi-Headed Attention

$$\alpha_{ij}^{d} = \frac{\exp\left(a^{d}\left(s_{i-1}, h_j\right)\right)}{\sum_k \exp\left(a^{d}\left(s_{i-1}, h_k\right)\right)}$$

- Why not have K attention weights?

$$\alpha_{ij} = \frac{1}{k}\sum_k \alpha_{ij}^{k}$$

- *Multi-Headed Attention* is just aggregating several attended representations of the encodings
- Fancy way of ensembling attention mechanisms

Multi-Headed Attention

$$\alpha_{ij}^{d} = \frac{\exp\left(a^{d}\left(s_{i-1}, h_j\right)\right)}{\sum_k \exp\left(a^{d}\left(s_{i-1}, h_k\right)\right)}$$

$$\alpha_{ij} = \begin{bmatrix} \alpha_{ij}^{0} \\ \vdots \\ \alpha_{ij}^{k} \\ \vdots \\ \alpha_{ij}^{K} \end{bmatrix} W_a$$

- **Alternative:** Concat each weight and use a linear layer to reduce the dimensionality
- Weighted Average of K Weighted Averages!

MHA Visual from the Paper

Self-Attention: Generalized

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

- Query Q, Key K, and Value V
- d_k is the dimensionality of the query and keys
- **Note** that this implies that we can re-mix the encodings via linear layers before attending
- **Note** the similiarity to Luong's Scaled Dot Attention

Self-Attention: What are Queries, Keys, and Values?

- **Encoders**
  - Source Embeddings, optionally re-mixed with linear layers
- **Decoders**
  - More complicated...
  - Technically 2 Self-Attentions
  - One that is the decoded embeddings
  - Then use that as the value for a second attention

Self Attention: Familiar Notation

$$a_{ij} = \frac{1}{|h|} h_j h_k^T$$

$$\alpha_{jk} = \frac{\exp(a_{jk})}{\sum_k \exp(a_{jk})}$$

$$\texttt{self-attention}(h_j) = \sum_k \alpha_{jk} h_k$$

Practical Considerations

Masking

Which one to use?

- Trend seems to be moving towards **Transformers**
- AKA a series of Self-Attentions inside Multi-headed Attentions
- Ideally, experiment and go crazy

Mixing and Matching

- Remember! You can always mix and match!
- Different attention mechanisms under MHA
- Etc, etc

Tools, References, and Further Reading

Papers

- Bahdanau et al., Neural Machine Translation by Jointly Learning to Align and Translate
- Luong et al.m Effective Approaches to Attention-based Neural Machine Translation
- Vaswani et al., Attention is All You Need
- Dyer et al., A Simple, Fast, and Effective Reparameterization of IBM Model 2