Deep Learning Techniques

DL4. Neural Attention Models

1

Outline

- Attention in Neural Machine Translation
- Self-attention and Transformer

2

Origin of Attention in Machine Translation

- **Example of a training pair**
  - Source sentence (x): le programme a été mis en œuvre
  - Target sentence (y): The program has been implemented.
- **Training**: Given parallel corpus $\mathcal{D} = \{(x, y)\}$, use a neural network to optimize the model parameter θ as

$$\theta^* = \underset{\theta}{\text{argmax}}\ \mathbb{E}_{(x,y)\epsilon\mathcal{D}}\ logP_\theta\ (y|x)$$

- **Testing**: Use trained model to predict y given x

$$y_{\theta*}(x) = \underset{y}{\text{argmax}}\ logP_{\theta*}\ (y|x)$$

Sequence-to-Sequence Model [Sutskever et al., NIPS 2014]

Let's denote $x = (x_1, x_2, \cdots, x_S)$ and $y = (y_1, y_2, \cdots, y_T)$.

- **Auto-regressive factorization**

$$P_\theta(\text{y}|\text{x}) = \prod_{j=1}^{T} P_\theta\left(y_j\middle|\text{x}, y_{<j}\right) = \prod_{j=1}^{T} P_\theta\left(y_j\middle|h_{i-1}\right) = \frac{exp\left(y_j^T h_{j-1}\right)}{\sum_{j'=1}^{M} exp\left(y_{j'}^T h_{j-1}\right)}$$

(h_{j-1} encodes the information of x, $y_{<j}$)

- **Encoder**: producing source hidden states $\boldsymbol{g} = (g_1, g_2, \cdots, g_S)$

  $g_0 = \mathbf{0}$ *and* $g_i = RNN_\theta^{enc}(x_\text{i}, g_{\text{i}-1})$

- **Decoder:** producing target hidden states $\boldsymbol{h} = (h_1, h_2, \cdots, h_T)$

  $h_0 = \text{g}_S$ *and* $h_j = RNN_\theta^{dec}(y_j, h_{j-1})$

Seq2seq Model Illustration

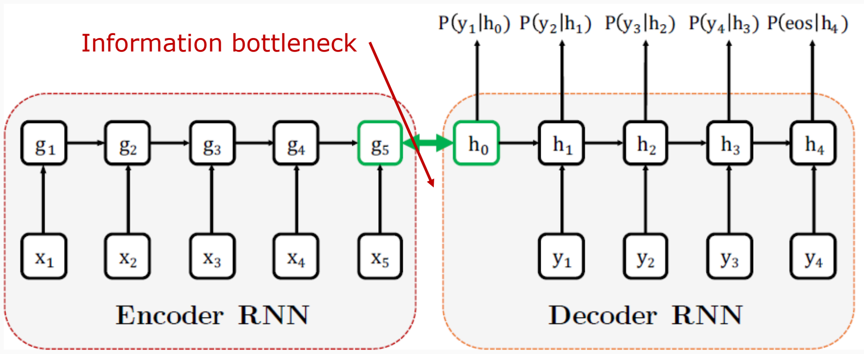


Figure 1: One-layer Seq2Seq Model.

Summarizing the whole sentence in single vector g_s

Alignment-based (Non-neural) MT

French: `Le programme a ete mise en application`
English: `The programme has been implemented`

Alignment: locally translate sub-units (words/phrase) of a sentence

$$
\begin{array}{rrcll}
(x_1) & \texttt{Le} & \Longrightarrow & \texttt{The} & (y_1) \\
(x_2) & \texttt{Programme} & \Longrightarrow & \texttt{program} & (y_2) \\
(x_3) & \texttt{a} & \Longrightarrow & \texttt{has} & (y_3) \\
(x_4) & \texttt{ete} & \Longrightarrow & \texttt{been} & (y_4) \\
(x_5, x_6, x_7) & \texttt{mise en application} & \Longrightarrow & \texttt{implemented} & (y_5)
\end{array}
$$

Observation: Predicting a target word mostly relies on a particular part of the source sentence

Alignment-based Factorization

- Original Factorization

  $P_\theta(\mathrm{y}|\mathrm{x}) = \prod_{j=1}^{T} P_\theta(y_j|x, y_{<j}) = \prod_{j=1}^{T} P_\theta(y_j|h_{j-1})$

  Compressing all the information in x by vector $g_S = h_0$

- Alignment-based Approximation

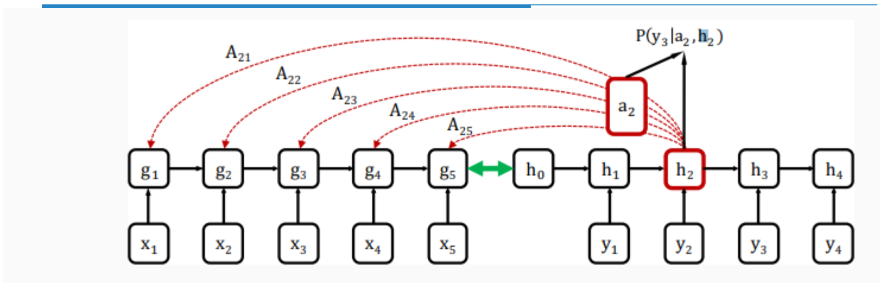
  $P_\theta(\mathrm{y}|\mathrm{x}) \approx \prod_{j=1}^{T} P_\theta(y_j|Align(y_j, x), h_{j-1})$

  Localizing the alignment in x for y_j

  e.g., Align(implemented, x) → (mise en application)

- Dynamic programming is used for cross-language token alignments in traditional MT (Brown et al., CL 1990)

Seq2Seq with Attention



$$Align(y_j, x) \approx Attention(h_{j-1}, \boldsymbol{g}) = \sum_{i=1}^{S} \overbrace{\frac{exp\left(f(h_{j-1}, \boldsymbol{g}_i)\right)}{\sum_{i'=1}^{S} exp\left(f(h_{j-1}, \boldsymbol{g}_{i'})\right)}}^{A_{ji}} g_i \equiv a_{j-1}$$

Why do we use h_{j-1} instead of y_j? Because we do not know y_j yet in the (testing-phase) decoding when computing the alignment.

Target Distribution

$$P_\theta\left(y_j \middle| a_{j-1}, h_{j-1}\right) = \frac{exp\left(\phi(y_j, a_{j-1}, h_{j-1})\right)}{\sum_{m=1}^{M} exp\left(\phi(y_m, a_{j-1}, h_{j-1})\right)}$$

Design choices:

1) $h'_{j-1} \coloneqq concat(a_{j-1}, h_{j-1}) \in \mathbb{R}^{2d}$, $y_j \in \mathbb{R}^{2d}$

2) $h'_{j-1} \coloneqq MLP\left(concat(a_{j-1}, h_{j-1})\right) \in \mathbb{R}^{d}$, $y_j \in \mathbb{R}^{d}$

Both cases have $\phi\left(y_j, w\right) = \frac{\exp\left(f\left(y_j, h'_{j-1}\right)\right)}{\sum_{m=1}^{M} \exp\left(f\left(y_m, h'_{j-1}\right)\right)}$

Soft Alignment via Attention
[D. Bahdanau et al., ICLR 2015]

- **Target Word Embedding via Attention**

embedding of context $y_{<j}$

$$h_j \coloneqq \sum_{i=1}^{S} \frac{exp\left(f\left(\boxed{h_{j-1}}, g_i\right)\right)}{\sum_{i'=1}^{S} exp\left(f\left(h_{j-1}, g_{i'}\right)\right)} g_i$$

attention from g_i to y_j

- f is a similarity function (more details later);
- target-token embedding is the weighted sum of the source-token embeddings;
- Attentions (weights) can be viewed as a soft alignment.

Choices of $f(h, g)$

- Dot-Product (most popular choice)

$$f(h, g) = h^T g$$

- Bilinear

$$f(h, g) = h^T W g$$

- MLP (Multi-Layer Perceptron)

$$f(h, g) = v^T \tanh\left(W_h h + W_g g + b\right)$$

Learned Attention Pattern

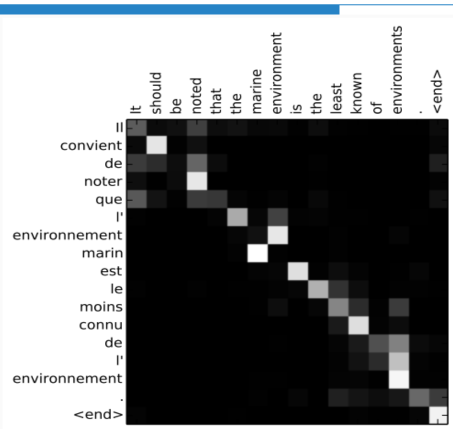


Figure 3: Visualize attention pattern.

Empirical Results

Performance gain on English-French translation:

Models	BLEU
Seq2Seq w/o Attn	21.50
Seq2Seq with Attn	28.45

Table 1: Performance gain with attention. The higher the better.

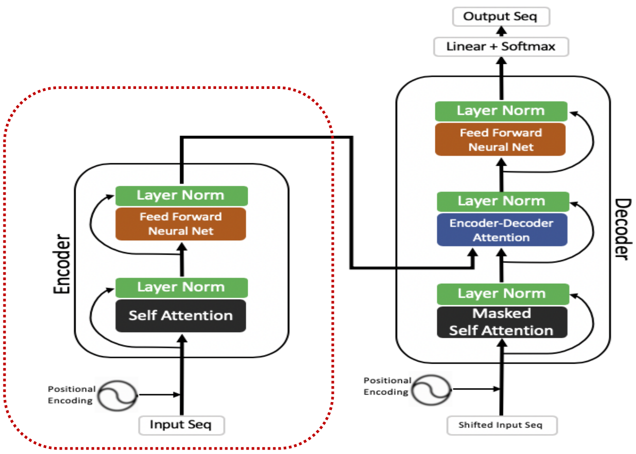
What makes attention so good?

- Soft-alignment b/w source-target tokens
- Avoid information bottleneck in RNNs

Outline

- ☐ Attention in Neural Machine Translation
- ☐ **Self-attention and Transformer**

Transformer for seq2seq Modeling
[A Vaswani et al. NeurIPS 2017]



Attention is all you need
[Vaswani et al., NeurIPS 2017]

- **Query** vectors (target words): $\boldsymbol{q} = (q_1, q_2, \cdots, q_T)) \in \mathbb{R}^{T\times D}$
- **Key** vectors (source words): $\boldsymbol{k} = (k_1, k_2, \cdots, k_S)\ ) \in \mathbb{R}^{S\times D}$
- **Value** vectors (source words): $\boldsymbol{v} = (v_1, v_2, \cdots, v_S) \in \mathbb{R}^{S\times D}$
- **Attention for target words**

$$q_j = attention(q_{j-1}, \boldsymbol{k}, \boldsymbol{v}) = \sum_{i=1}^{S} \frac{exp\left(f(q_{j-1}, \boldsymbol{k}_i)\right)}{\sum_{i'=1}^{S} exp\left(f(q_{j-1}, \boldsymbol{k}_{i'})\right)} v_i$$

or $$\underbrace{\boldsymbol{q}}_{\mathbb{R}^{T\times D}} = attention(\boldsymbol{q}, \boldsymbol{k}, \boldsymbol{v}) = \underbrace{Softmax\left(\boldsymbol{q}\boldsymbol{k}^T\right)}_{\mathbb{R}^{T\times S}} \underbrace{\boldsymbol{v}}_{\mathbb{R}^{S\times D}}$$

row-wise normalized attentions (probabilities)

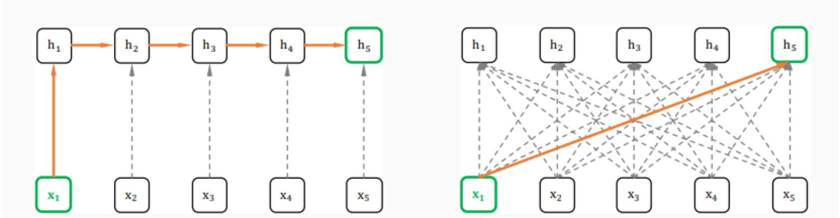
Self-Attention

- Consider a sequence ($\boldsymbol{q}$) of tokens in the same language.
- We want to use attention to obtain a new sequence in the same language

$$\boldsymbol{q}' = Attention(\boldsymbol{q}, \boldsymbol{k}, \boldsymbol{v}) = \mathrm{F}(\mathbf{q}) \quad \text{where } \boldsymbol{q} = \boldsymbol{k} = \boldsymbol{v}.$$

- Mapping $F: \mathbb{R}^{T\times D} \longmapsto \mathbb{R}^{T\times D}$ is called self-attention (a sequence attends to itself).
- Intuitively, each position gather/retrieve information from all positions with the pairwise similarities for a weighted sum.

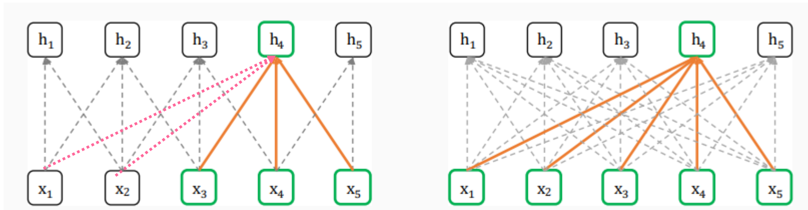
Self-Attention vs. RNN



(a) RNN (b) Self-Attention

- Self-attention directly connects each pair of nodes without information bottleneck and vanishing gradient issues
- As a result, optimizing self-attention models is much easier than optimizing RNNs

Self-Attention vs. CNN



(a) CNN (b) Self-Attention

- Self-attention takes the weighted sum of all tokens while CNN takes the weighted sum of near-by tokens;
- Dynamic weights in self-attention vs. fixed weights in CNN during testing, i.e., the former is more flexible than the latter.

Transformer Encoder

Four Key Ideas

1) **Multi-head self-attention** (MH-Self-Attn) for exchanging sequential information
2) **Feed-Forward Network** (FFN) for position-wise linear and non-linear transformations (ReLU) of hidden vectors
3) **Position Embedding** (PosEmbed)
4) Ease optimization via **residual connection** [5] and **layer normalization** (LayerNorm) [1]

Transformer: Multi-layer Info. Flow

- Initialization of hidden states

$$\boldsymbol{h}^0 = \text{WordEbed }(x) + \text{PosEmbed }(x)$$

- For layer $m = 1, \cdots, L$

$$\boldsymbol{g}^m = \text{LayerNorm}(h^{m-1} + \text{MH-Self-Attn }(h^{m-1}))$$

$$\boldsymbol{h}_i^m = \text{LayerNorm}(g_i^m + \text{FFN}(g_i^m)), \ \ \forall i = 1, \cdots, T$$

Transformer: MH-Self-Attn

("Using multiple heads to improve flexibility/capacity.")

Core Idea: Use multiple (N) smaller attentions ("heads") with different parameters and running in parallel

For head $\mathrm{n} \in \{1, \cdots, N\}$

$$query_n = \boldsymbol{h}W_{\boldsymbol{n}}^{\boldsymbol{q}}, \qquad W_{\boldsymbol{n}}^{\boldsymbol{q}} \in \mathbb{R}^{\boldsymbol{D}\times\frac{\boldsymbol{D}}{\boldsymbol{N}}}$$

$$key_n = \boldsymbol{h}W_{\boldsymbol{n}}^{\boldsymbol{k}}, \qquad W_{\boldsymbol{n}}^{\boldsymbol{k}} \in \mathbb{R}^{\boldsymbol{D}\times\frac{\boldsymbol{D}}{\boldsymbol{N}}}$$

$$value_n = \boldsymbol{h}W_{\boldsymbol{n}}^{\boldsymbol{v}}, \qquad W_{\boldsymbol{n}}^{\boldsymbol{v}} \in \mathbb{R}^{\boldsymbol{D}\times\frac{\boldsymbol{D}}{\boldsymbol{N}}}$$

$$head_n = \text{Attention}(query_n,\ key_n,\ value_n) \in \mathbb{R}^{\boldsymbol{T}\times\frac{\boldsymbol{D}}{\boldsymbol{N}}}$$

$$\text{Cancatnate}(head_1,\ head_2, \cdots, head_n) \in \mathbb{R}^{\boldsymbol{T}\times\boldsymbol{D}}$$

$$\text{MultiHead(Q,K,V)} = \text{Cancatnate}(head_1,\ head_2, \cdots, head_n)W^o$$

Transformer: Feed-Forward Network (FFN)

FFT is a position-wise 2-layer MLP being shared in all positions:

$$\text{FFN}(\boldsymbol{h}_i) = W_2 \text{ReLU}(W_1 \boldsymbol{h}_i + \boldsymbol{b}_1) + \boldsymbol{b}_2 \;, \;\; \forall i = 1, \cdots, T$$

- $W_1 \in \mathbb{R}^{\boldsymbol{MD} \times \boldsymbol{D}}$ projects the hidden state to MD-dimensional and $\boldsymbol{M} > \mathbf{1}$ is often referred to as the expansion rate;
- $W_2 \in \mathbb{R}^{\boldsymbol{D} \times \boldsymbol{MD}}$ projects the hidden state back to D-dimensional;
- The "expansion-squeezing" design is also referred to as "Inverted Bottleneck" [8].

Transformer: Positional Encoding (PosEmbed)

Recal: Attention scores MAY NOT depend on word order

$$\text{Self-Atten}(j, \boldsymbol{h}) = \sum_{i=1}^{S} \frac{exp(f(\boldsymbol{h}_j, \boldsymbol{h}_i))}{\sum_{i'=1}^{S} exp(f(\boldsymbol{h}_j, \boldsymbol{h}_{i'}))} \boldsymbol{h}_i$$

If f is a symmetric function (such as dot-prod or cosine), swapping the order of its arguments does not change the similarity value.

Remedy: Add position embedding in the hidden states

$$\boldsymbol{h}^0 = \text{WordEbed } (x) + \text{PosEmbed } (x)$$

- Position embedding is learned just like word embedding [4];
- Position embedding is based on cosine or sine waves [10].

Empirical Results

Word (wikitext-103) and character (enwik8) **language modeling**

Models	WikiText-103 (PPL)	Enwik8 (BPC)
Best LSTM Variant	29.9	1.23
Transformer	**20.5**	**1.06**

Table 2: PPL = perplexity and BPC = bits per character. The lower the better.

Machine translation

Models	En-Fr	En-De
LSTM + Attention	40.56	26.03
Transformer	**41.8**	**28.4**

Table 3: BLEU score on two datasets. The higher the better.

Concluding Remarks

- **Attention**
  - Dynamic information gathering based on pairwise similarity
  - Has been plugged into various models (Bert, GPT, etc.)
- **Transformer**
  - SOTA performance in many large-scale LM applications
  - Performance keeps improving when massive training data are available and if computation/memory is affordable (many works have focused on Transformer with scalable attention)

References

[1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton.
Layer normalization.
arXiv preprint arXiv:1607.06450, 2016.

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio.
Neural machine translation by jointly learning to align and translate.
arXiv preprint arXiv:1409.0473, 2014.

[3] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov.
Transformer-xl: Attentive language models beyond a fixed-length context.
arXiv preprint arXiv:1901.02860, 2019.

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova.
Bert: Pre-training of deep bidirectional transformers for language understanding.
arXiv preprint arXiv:1810.04805, 2018.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun.
Deep residual learning for image recognition.
In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[6] Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya.
Reformer: The efficient transformer.
arXiv preprint arXiv:2001.04451, 2020.

[7] Minh-Thang Luong, Hieu Pham, and Christopher D Manning.
Effective approaches to attention-based neural machine translation.
arXiv preprint arXiv:1508.04025, 2015.

References

[8] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen.
Mobilenetv2: Inverted residuals and linear bottlenecks.
In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

[9] Ilya Sutskever, Oriol Vinyals, and Quoc V Le.
Sequence to sequence learning with neural networks.
In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[10] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin.
Attention is all you need.
In *Advances in neural information processing systems*, pages 5998–6008, 2017.