Lecture 4. Attention is All You Need

Young, Shen

August 11, 2024

1 When Input is a Sequence

When input is a sentence or a graph, they are a sequence of vectors. In NLP we call the vector **Token** of the sequence. Every token is related to another, so we can't simply processing them only looking on one token, but considering the whole sequence, just like image identification, in which we used convolution to consider the pixels around a certain pixel to find if it is a certain pattern. Here, we introduce the attention mechanism.

2 Transformer

2.1 Embedding

To deal with the different shape of token, we **embed** them into a designed feature vector. Sometimes this operation is learnt and sometimes set in advance.

2.2 Query, Key and Value

After embedding the input, we get N vectors of the same size (N represents the number of tokens). Supposing each feature vector has dimension M.

$$\left[\mathbf{x}^1, \mathbf{x}^2, \mathbf{x}^3, \cdots, \mathbf{x}^N\right] = \mathbf{I}_{M\times N}$$

Where $\mathbf{x^i}$ denotes the ith element of the input sequence, which is a column vector as we embedded before.

To each input vector, we transform it into **Query, Key and Value** by multiplying corresponding matrix to it.

$$\begin{aligned}
\mathbf{Q}_{d\times N} = \mathbf{W}_{q,d\times M}\mathbf{I} = \mathbf{W_q}\left[\mathbf{x}^1, \mathbf{x}^2, \mathbf{x}^3, \cdots, \mathbf{x}^N\right] = \left[\mathbf{q}^1, \mathbf{q}^2, \mathbf{q}^3, \cdots, \mathbf{q}^N\right] \\
\mathbf{K}_{d\times N} = \mathbf{W}_{k,d\times M}\mathbf{I} = \mathbf{W_k}\left[\mathbf{x}^1, \mathbf{x}^2, \mathbf{x}^3, \cdots, \mathbf{x}^N\right] = \left[\mathbf{k}^1, \mathbf{k}^2, \mathbf{k}^3, \cdots, \mathbf{k}^N\right] \\
\mathbf{V}_{d'\times N} = \mathbf{W}_{v,d'\times M}\mathbf{I} = \mathbf{W_v}\left[\mathbf{x}^1, \mathbf{x}^2, \mathbf{x}^3, \cdots, \mathbf{x}^N\right] = \left[\mathbf{v}^1, \mathbf{v}^2, \mathbf{v}^3, \cdots, \mathbf{v}^N\right]
\end{aligned}$$

where $\mathbf{W}_q$, $\mathbf{W}_k$, $\mathbf{W}_v$ denotes the transform matrix to query, key and value and they are **LEARNED** during training. Each column of $\mathbf{Q}$. $\mathbf{K}$, $\mathbf{V}$ denotes the corresponding query, key and value of the input. Note that $\mathbf{Q}$ and $\mathbf{K}$ always have same dimension, but $\mathbf{V}$ sometimes can have different row dimension. Define:

$$\alpha_{ij} = \mathbf{k^i} \cdot \mathbf{q^j}$$

where α_{ij} denotes the attention from j to i input, we have:

$$\mathbf{A} = \begin{bmatrix} \alpha_{11} & \alpha_{12} & \cdots & \alpha_{1N} \\ \alpha_{21} & \alpha_{22} & \cdots & \alpha_{2N} \\ & & \cdots & \\ \alpha_{N1} & \alpha_{N2} & \cdots & \alpha_{NN} \end{bmatrix} = \mathbf{K}^\top\mathbf{Q}$$

Obviously, the jth column contains the attention from jth input to the others. Usually we have to normalize it, softmax often used. Note that softmax is not the only option, ReLU can also be used.

$$\alpha_{kj}^{'} = \frac{exp(\alpha_{kj})}{\sum_{i=1}^{N} exp(\alpha_{ij})}$$

with each α is replaced by the softmax over its column, which forms another matrix $\mathbf{A}^{'}$.

In the last, the output of the jth input is the sum of all the Value, weighted by the normalized attention from jth input:

$$\mathbf{b^j} = \sum_{k=1}^{N} \alpha_{kj}^{'} \mathbf{v^k}$$

which can be represented in the matrix form:

$$\mathbf{O} = \left[\mathbf{b}^1, \mathbf{b}^2, \mathbf{b}^3, \cdots, \mathbf{b}^N\right] = \mathbf{V}\mathbf{A}^{'}$$

3 Linear Transformer: higher computational efficiency

In practise, transformer has a few problems:

- **Time consuming**: to compute the exponential of a inner product is expensive.
- **Accuracy**: exponential computation usually has low accuracy, causing numerical instability.
- **Memory**: using transformer solely is memory consuming.

So we introduce Linear-transformer. Considering:

$$\begin{aligned}\mathbf{b^j} &= \sum_{k=1}^{N} \alpha_{kj}^{'} \mathbf{v^k} \\ &= \sum_{k=1}^{N} \frac{exp(\alpha_{kj})}{\sum_{i=1}^{N} exp(\alpha_{ij})} \mathbf{v^k} \\ &= \frac{\sum_{k=1}^{N} exp(\mathbf{k}^k \cdot \mathbf{q}^j)\mathbf{v}^k}{\sum_{i=1}^{N} exp(\mathbf{k}^i \cdot \mathbf{q}^j)}\end{aligned}$$

Note that k is indexing the weighted sum over the columns of $\mathbf{V}$, and ii is indexing in the softmax part, the sum of exponential attention from input j. Here, to reduce computational cost, we introduce a trick:

$$exp(\mathbf{k} \cdot \mathbf{q}) \approx \Phi(\mathbf{k}) \cdot \Phi(\mathbf{q})$$

where Φ is a function mapping the vector to higher dimension. Given appropriately chosen Φ, the approximation can be accurate. Let's consider the denominator first:

$$\sum_{i=1}^{N} exp(\mathbf{k}^i \cdot \mathbf{q}^j) = \sum_{i=1}^{N} \Phi(\mathbf{k}^i) \cdot \Phi(\mathbf{q}^j) = \Phi(\mathbf{q}^j) \cdot \sum_{i=1}^{N} \Phi(\mathbf{k}^i)$$

We then consider the numerator (note that there IS NO association law for inner product!!!):

$$\begin{aligned}&\sum_{k=1}^{N} exp(\mathbf{k}^k \cdot \mathbf{q}^j)\mathbf{v}^k \\ =&\sum_{k=1}^{N} (\Phi(\mathbf{k}^k) \cdot \Phi(\mathbf{q}^j))\mathbf{v}^k\end{aligned}$$

let (to avoid confusion, remember only the kk as superscript is index):

$$\Phi(\mathbf{k}^k) = \begin{bmatrix} k_1^k \\ k_2^k \\ k_3^k \\ \cdots \\ k_l^k \end{bmatrix}, \quad \Phi(\mathbf{q}^j) = \begin{bmatrix} q_1^j \\ q_2^j \\ q_2^j \\ \cdots \\ q_l^j \end{bmatrix}$$

where l denotes the dimension of the higher dimension Φ maps to. Take this into the numerator:

$$numerator = \sum_{k=1}^{N} \sum_{i=1}^{l} k_i^k q_i^j \mathbf{v}^k = \sum_{i=1}^{l} \sum_{k=1}^{N} k_i^k q_i^j \mathbf{v}^k = \sum_{i=1}^{l} q_i^j \sum_{k=1}^{N} k_i^k \mathbf{v}^k$$

define:

$$\Phi(\mathbf{K}) = \left[\Phi(\mathbf{k}^1), \Phi(\mathbf{k}^2), \Phi(\mathbf{k}^3), \cdots, \Phi(\mathbf{k}^N)\right]$$

Then:

$$\sum_{k=1}^{N} k_i^k \mathbf{v}^k = [v1, v2, v3, \cdots, vN] \left[\mathbf{v}^1, \mathbf{v}^2, \mathbf{v}^3, \cdots, \mathbf{v}^N\right] \Phi(\mathbf{K})^\top(:, i)$$

where (:,i)(:,i) means the ith column (which is a column vector) of the matrix,thus:

$$numerator = [v1, v2, v3, \cdots, vN] \left[\mathbf{v}^1, \mathbf{v}^2, \mathbf{v}^3, \cdots, \mathbf{v}^N\right] \Phi(\mathbf{K})^\top \Phi(\mathbf{q}^j) = \mathbf{V}\Phi(\mathbf{K})^\top \Phi(\mathbf{q}^j)$$

Finally, we get the expression of $\mathbf{b}^j$:

$$\mathbf{b}^j = \frac{\mathbf{V}\Phi(\mathbf{K})^\top \Phi(\mathbf{q}^j)}{(\sum_{i=1}^{N} \Phi(\mathbf{k}^i)) \cdot \Phi(\mathbf{q}^j)}$$

Notice that aside from $\Phi(\mathbf{q}^j)$, the other components are not related to j, which means that we only need to compute them once in one upgrade. This saves our computational cost.

To understand this expression, we can see the denominator as normalization factor, column vectors of $\mathbf{V}\Phi(\mathbf{K})^\top$ are processed keys of attention, which are weighted by $\Phi(\mathbf{q}^j)$.

3.1 How to choose mapping function

Here are some references:[4][5][8]

4 More Methods

Based on the idea of transformer, various type of methods are developed:

4.1 Human Knowledge

Local Attention

Local attention mechanisms restrict the range of tokens that each token can attend to, typically to nearby tokens. This reduces computational complexity and memory usage, making it feasible to apply transformers to longer sequences. [10]

Big Bird

Big Bird extends transformers by combining sparse attention with global tokens and random tokens, allowing it to handle sequences of length up to 4096 tokens. This approach maintains the benefits of the transformer model while significantly reducing its computational complexity. [10]

4.2 Clustering

Reformer

Reformer uses locality-sensitive hashing (LSH) to approximate the attention mechanism, reducing the quadratic complexity of traditional transformers to linear. It also introduces reversible layers to save memory. This makes Reformer particularly effective for tasks requiring processing very long sequences. [3]

4.3 Learnable Pattern

Sinkhorn

The Sinkhorn Transformer uses Sinkhorn sorting networks to reorder tokens into a more structured sequence that can be attended to more efficiently. It blends ideas from optimal transport with self-attention to improve model efficiency. [6]

4.4 Representative Key

Linformer

Linformer approximates the self-attention mechanism by projecting the attention matrix into a lower-dimensional space, achieving linear complexity with respect to sequence length. This reduces the memory and computational requirements, making transformers more scalable for long sequences. [9]

4.5 k, q first $\rightarrow$ v, k first

Linear Transformer

Linear Transformers use kernel methods to approximate the softmax attention mechanism, transforming the quadratic complexity of traditional transformers to linear complexity. They process sequences in a way that is similar to fast weight programmers. [2]

Performer

Performers use a mechanism called FAVOR+ (Fast Attention Via positive Orthogonal Random features) to approximate the attention mechanism. This allows them to achieve linear complexity while maintaining the benefits of traditional self-attention. [1]

4.6 New Framework

Synthesizer

Synthesizers replace the dot-product attention mechanism with simpler parameterized attention patterns. Instead of computing attention scores from token similarities, they generate attention scores directly from learned parameters. This approach simplifies the model and reduces computational overhead while maintaining competitive performance on various NLP tasks. [7]

References

[1] Krzysztof Choromanski et al. *Rethinking Attention with Performers*. 2020. arXiv: 2009.14794 [cs.LG]. URL: https://arxiv.org/abs/2009.14794.

[2] Angelos Katharopoulos et al. *Transformers are RNNs: Fast Autoregressive Transformers with Linear Attention*. 2020. arXiv: 2006.16236 [cs.CL]. URL: https://arxiv.org/abs/2006.16236.

[3] Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. *Reformer: The Efficient Transformer*. 2020. arXiv: 2001.04451 [cs.LG]. URL: https://arxiv.org/abs/2001.04451.

[4] Kevin Lee-Thorp et al. *FNet: Mixing Tokens with Fourier Transforms*. 2021. arXiv: 2105.03824 [cs.LG]. URL: https://arxiv.org/abs/2105.03824.

[5] Hanxiao Liu et al. *Pay Attention to MLPs*. 2021. arXiv: 2105.08050 [cs.LG]. URL: https://arxiv.org/abs/2105.08050.

[6] Yi Tay et al. *Sinkhorn Transformer: Adding Implicit Rewiring to Reformers for Linear Time Complexity*. 2020. arXiv: 2002.11296 [cs.LG]. URL: https://arxiv.org/abs/2002.11296.

[7] Yi Tay et al. *Synthesizer: Rethinking Self-Attention for Transformer Models*. 2020. arXiv: 2005.00743 [cs.CL]. URL: https://arxiv.org/abs/2005.00743.

[8] Ilya Tolstikhin et al. *MLP-Mixer: An all-MLP Architecture for Vision*. 2021. arXiv: 2105.01601 [cs.CV]. URL: https://arxiv.org/abs/2105.01601.

[9] Sinong Wang et al. *Linformer: Self-Attention with Linear Complexity*. 2020. arXiv: 2006.04768 [cs.LG]. URL: https://arxiv.org/abs/2006.04768.

[10] Manzil Zaheer et al. *Big Bird: Transformers for Longer Sequences*. 2020. arXiv: 2007.14062 [cs.LG]. URL: https://arxiv.org/abs/2007.14062.