Transformer:

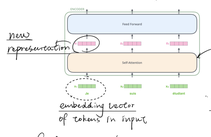
RNN: Sequential $y_1 \to y_2 \to \cdots$. Precludes parallelization

Transformer: "Parallelization"

Base solely on attention, dispensing w/ recurrence (不用循环)

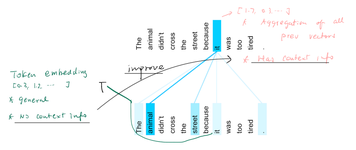
⇒ less training time, better result

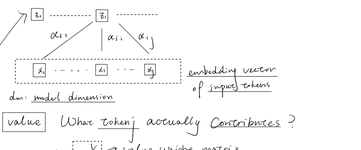
1. Self-attention layer



work in parallel

Self-attention layer improve the representation of each token by combining information from other tokens





value: What $token_j$ actually contributes?

$v_j = x_j \cdot W^V$ → value weight matrix

Query: What $token_j$ actually needs?

$q_j = x_j \cdot W^Q$ → query weight matrix

key: How to describe what $Token_j$ has?

$k_j = x_j \cdot W^K$ → key weight matrix

x_j: $1 \times dm$ ⇒ W^V: $dm \times dV$, W^Q: $dm \times dk$, W^K: $dm \times dk$ ⇒ v_j: $1 \times dV$ "value vector"; q_j: $1 \times dk$ "query vector"; k_j: $1 \times dk$ (actually $dV = dk$) "key vector"

$\alpha_{ij} \leftarrow \text{softmax}\left(\frac{q_i k_j^T}{\sqrt{dk}}\right)$ "Attention of i on j"

$z_i = \sum \alpha_{ij} v_j$ "value" "Weighted Sum of values" "$[1 \times dV]$"

Self-attention layer
- x_i, z_i: vectors.
- learnable weight: W^Q, W^K, W^V
- α_{ij}: dynamic (depends on input)

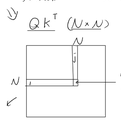
FNN layer
- x_i, z_i: scalars.
- learnable weight: W_{ij}
- Not dynamic

Parallelization

$X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix}$ $N \times dm$, $Q = \begin{bmatrix} q_1 \\ q_2 \\ \vdots \\ q_N \end{bmatrix}$ $N \times dk$, $K = \begin{bmatrix} k_1 \\ k_2 \\ \vdots \\ k_N \end{bmatrix}$ $N \times dk$, $V = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_N \end{bmatrix}$ $N \times dv$.

$Q = X W^Q$, $dm \times dq$; $K = X W^K$, $dm \times dk$; $V = X W^V$, $dm \times dv$

QK^T $(N \times N)$



$\text{softmax}\left(\frac{QK^T}{\sqrt{dk}}\right) = \begin{bmatrix} \frac{q_1 k_1^T}{\sqrt{dk}} & \cdots & \frac{q_1 k_N^T}{\sqrt{dk}} \\ \vdots & \frac{q_i k_j^T}{\sqrt{dk}} & \vdots \\ \frac{q_N k_1^T}{\sqrt{dk}} & \cdots & \frac{q_N k_N^T}{\sqrt{dk}} \end{bmatrix} = \begin{bmatrix} \alpha_{11} & \alpha_{12} & \cdots & \alpha_{1N} \\ \alpha_{21} & & & \vdots \\ \vdots & & \ddots & \vdots \\ \alpha_{N1} & \cdots & & \alpha_{NN} \end{bmatrix}$

$Z = \text{softmax}\left(\frac{QK^T}{\sqrt{dk}}\right) \cdot V$ $([N \times dv])$

Time complexity $O(N^2 dk)$ → Context size

Multi-Head Attention

Consider multiple ways to combine informations

$[W_i^Q, W_i^K, W_i^V]_{i=1}^{h}$

single-head: $X \xrightarrow{W^Q W^K W^V} z$

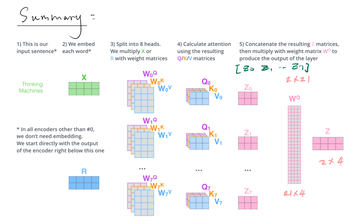
Multiple-head: $X \xrightarrow{W_1^Q W_1^K W_1^V} z_1$, $\xrightarrow{W_2^Q W_2^K W_2^V} z_2$, z_3, ⋮, z_h

$N \left[z_1, z_2, \cdots z_h \right]$ $\xrightarrow{hdv \to dv} W_h^Q W_h^K W_h^V$

$z = \text{cat}(z_1, \cdots z_h) \cdot W^O$ (output weight matrix)

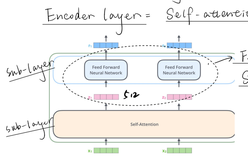
$N \times dm$ $(N \times hdv)$ $(hdv \times dm)$

Summary:



Encoder
- FNN
- Residual connection
- Layer Normalization
- Positional embedding

Encoder layer = Self-attention layer + FNN layer



FNN at different positions Share parameters

FNN at Pos 1 is the same with FNN at Pos 2

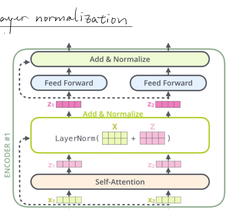
Residual connection & layer normalization

Residual connection

$H(x) = x + F(x)$

Layer Normalization

$x_{norm} = \gamma_b \frac{x - \hat{m}_b}{\sqrt{\sigma_b + \varepsilon}} + \beta_b$



Why Layer normalization instead of batch normalization

① Sequence length variability

Batch: inputs: ○ ○ ○