Linformer - Paper

→ The self attention mechanism runs in $O(n^2)$ because Attention = $\mathrm{Softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)$

→ Other solutions : → 1. Use FP16 instead FP32, doesn't reduce $O(N^2)$
→ 2. Knowledge Distillation : → Student loses accuracy
→ Does NOT speed up training of the large MODEL.

→ Sparse Attention : → Only computes a subset of attention matrix.
→ REDUCE performance
→ Speed up is small.

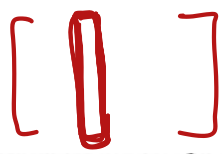
Rank : Number of independent directions in the matrix.

* When we calculate the Attention Matrix. We compute the singular values using $X = U \Sigma V^T$. Then they take

$$\boxed{\frac{\sum_{i=1}^{k} \sigma_i}{\sum_{i=1}^{N} \sigma_i} = \text{energy}}$$

→ this higher means that the singular values up to 'k' takes most of the information into account.

→ Found that the singular values decays fast and becomes low later. Thus we can project our attention matrix into lower dimension.



Theorem 1. (self-attention is low rank) *For any $Q, K, V \in \mathbb{R}^{n \times d}$ and $W_i^Q, W_i^K, W_i^V \in \mathbb{R}^{d \times d}$, for any column vector $w \in \mathbb{R}^n$ of matrix VW_i^V, there exists a low-rank matrix $\tilde{P} \in \mathbb{R}^{n \times n}$ such that*

$$\Pr(\|\tilde{P}w^T - Pw^T\| < \epsilon\|Pw^T\|) > 1 - o(1) \text{ and } \operatorname{rank}(\tilde{P}) = \Theta(\log(n)), \quad (3)$$

→ goes to 0 as $n \to \infty$

where the context mapping matrix P is defined in (2).

→ we know that Pw^T represents feature 'i' of token 'n'. So when we multiply Pw^T → How much each feature 'i' of token 1...N contributes weighted by how much token 'j' pays attention to 1 — N for feature 'j' of token 'j)

EASY WAY TO THINK ABOUT IT

Given the low-rank property of the context mapping matrix P, one straightforward idea is to use singular value decomposition (SVD) to approximate P with a low-rank matrix P_{low}, as follows

$$P \approx P_{\text{low}} = \sum_{i=1}^{k} \sigma_i u_i v_i^T = \underbrace{\begin{bmatrix} u_1, \cdots, u_k \end{bmatrix}}_{k} \operatorname{diag}\{\sigma_1, \cdots, \sigma_k\} \left.\begin{bmatrix} v_1 \\ \vdots \\ v_k \end{bmatrix}\right\} k \tag{6}$$

where σ_i, u_i and v_i are the i largest singular values and their corresponding singular vectors. Based on the results in Theorem 1 and the Eckart–Young–Mirsky Theorem (Eckart & Young, 1936), one can use P_{low} to approximate self-attention (2) with ϵ error and $O(nk)$ time and space complexity. However, this approach requires performing an SVD decomposition in *each* self-attention matrix, which adds additional complexity. Therefore, we propose another approach for low-rank approximation that avoids this added complexity.

MODEL

→ Project our key and our Query matrix into smaller values. Thus

$(n \times d) \rightarrow (n \times k)$

$(n \times d) \rightarrow (n \times k)$

Thus, we can say the following → (QW_i^Q)

$= (N \times d)(d \times d) = (N \times d)$

→ Project the key value to a lower dimension.

$E_i K W_i^K \rightarrow (n \times d)(d \times d)$

$(k \times n)$

↳ this's a $(N \times d)$

↳ compresses the token space have 'k' representative tokens.

Thus

$\boxed{E_i K W_i^K = (k \times d)}$

* The $\boxed{\text{softmax}\left(\dfrac{QW_i^Q(E_iKW_i^K)^T}{\sqrt{d_q}}\right)\cdot F_iVW_i}$ = head_i

$$\frac{((n\times d).(d\times d)).((k\times n)(n\times d)(d\times d))^T)}{\sqrt{d_q}}$$

$(n\times d).(k\times d)^T = (n\times d)(d\times k) = \boxed{(n\times k)}$

$F_iVW_i \rightarrow (d\times d) = \boxed{(k\times d)}$

$\hookrightarrow (n\times d)$

$\hookrightarrow (k\times n)$

So we project the key into 'k' tokens and then we basically come out with an attention matrix where we see how our tokens pay attention to those 'k' tokens.

Then we reproject our attention matrix in our original dimension space.

→ thus if we choose a very small projected dimension 'k'.

$(k \ll n)$

→ You can choose different kinds of low dimensional projection methods in a simple

SUMMARY OF RESULTS

↳ Linformer matches transformer accuracy with far lower runtime and memory. Performance depends on 'k'

↳ larger k improves accuracy

Inference-Time Efficiency

↳ speed-up

↳ memory saved