Linformer

Monday, June 28, 2021 3:30 AM

- Paper link https://arxiv.org/pdf/2006.04768.pdf

- Abstract
  - ⇒ Self attention mechanism in transformers take $O(n^2)$ time and space.
  - ⇒ It can, however, be approximated by a low-rank matrix.
  - ⇒ This reduces the complexity from $O(n^2)$ to $O(n)$.
  - ⇒ The resulting model that uses the mechanism above, called Linformer, performs on par with standard transformer models.

- Intro
  - ⇒ Question: How to avoid quadratic time complexity of self-attention mechanism?
  - ⇒ Approaches
    - Child et al. 2019 ⟶ $O(n\sqrt{n})$ [Link]
      - ↳ large performance drop (2%) with limited speedup (20%)
    - Kitaev et al. 2020 ⟶ $O(n \log n)$ [Link]
      - ↳ uses locality sensitive hashing
      - ↳ gains only appear with length > 2048
      - ↳ multi-round hashing → increased sequential operations
  - ⇒ Insight: self-attention is low rank
    - The stochastic matrix formed by self-attention can be approximated by a low-rank matrix through the

the stochastic matrix formed by self-attention can be approximated by a low-rank matrix through the Johnson-Lindenstrauss lemma.

- Turn the scaled dot product attention to a number of smaller attentions through linear projections.

⇒ Linformer models performed well compared to standard transformers

- pretraining performance
- fine-tuned performance

⇒ $O(n)$ complexity leads to significant training and inference speedup.

- Background

⇒ Transformers are made of multi-head attention (MHA) blocks.

$$\text{MHA}(Q, V, K) = \text{Concat}(\text{head}_1, \dots, \text{head}_h) W^O$$

where $Q, V, K \in \mathbb{R}^{n \times d_m}$

d_m = embeding dimension

n = sequence length.

$W^O \in \mathbb{R}^{hd_v \times d_m}$

d_v = dimension of the projection subspace of each head.

⇒ For each head,

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^W)$$

$$= \text{softmax}\left(\frac{QW_i^Q (KW_i^K)^T}{\sqrt{d_k}}\right) VW_i^W$$

where $W_i^Q, W_i^K \in \mathbb{R}^{d_m \times d_k}$

$W_i^V \in \mathbb{R}^{d_m \times d_v}$

where $W_i^Q, W_i^K \in \mathbb{R}$

$W_i^V \in \mathbb{R}^{d_m \times d_v}$

d_k = dimension of the embeddings of the keys and queries.

$\Rightarrow$ Define $P = \text{softmax}\left(\dfrac{QW_i^Q (KW_i^K)^T}{\sqrt{d_k}} \right)$ "context mapping matrix"

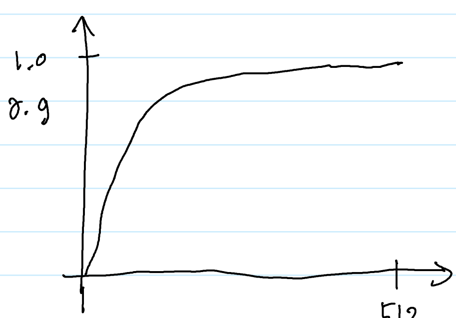
$\Rightarrow$ Computing $P \in \mathbb{R}^{n \times n}$ involves multiplying $QW_i^Q \in \mathbb{R}^{n \times d_k}$ and $KW_i^K \in \mathbb{R}^{n \times d_k}$

which takes $O(n^2 d_k)$

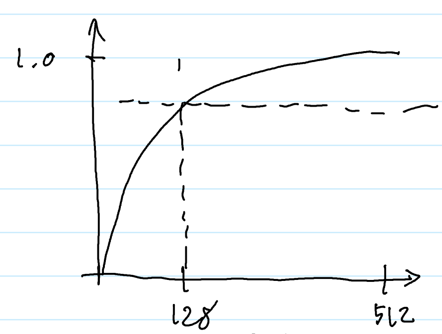
- Self-attention is low-rank.

$\Rightarrow$ The context mapping matrix P is low-rank.

- The paper took two pretrained transformer models
  - $\Rightarrow$ RoBERTa-base (12 layers)
  - $\Rightarrow$ RoBERTa-large (24 layers)
- They computed P over different layers, different heads, over 10k sentences.
- They find the singular values of the matrices and plotted the cumulative normalized singular value graphs. The graphs are then averaged.
- The graphs look kind of like this.



RoBERTa-base



RoBERTa-large

RoBERTa - base^{512} $\qquad$ RoBERTa - large $^{128}\ {}^{512}$

- So, it seems that 90% of the mass of the singular values are located in the first 128.

- In higher layers, the mass concentrates on the large singular values even more.

⇒ Theoretically, we have the following result.

<u>Thm</u> 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 random matrix $\tilde{P} \in \mathbb{R}^n \times \mathbb{R}^n$ of rank $\Theta(\log n)$ s.t.

$$\Pr\left(\| \tilde{P} w^T - P w^T \| < \varepsilon \| P w^T \| \right) > 1 - o(1)$$

⇒ The proof uses the distributional Jonson-Lindenstrauss lemma.

⇒ In fact, $\tilde{P} = PR^TR$ where R is a $k \times n$ matrix whose entries are sampled from the distribution $\frac{1}{\sqrt{k}} \mathcal{N}(0, 1)$.

- <u>Model</u>

⇒ First, let us say that $d_k = d_v = d$ to simplify stuffs.

⇒ We introduce linear mappings $E_i, F_i \in \mathbb{R}^{k \times n}$ when computing the keys and values. So,

$$\boxed{\begin{array}{ccc} KW_i^K & \rightarrow & E_i K W_i^K \\ (n \times d) & & (k \times d) \end{array}} \qquad \boxed{\begin{array}{ccc} VW_i^V & \longrightarrow & F_i V W_i^V \\ (n \times d) & & (k \times d) \end{array}}$$

⇒ So, the new head function becomes

$$\overline{\text{head}_i} = \text{Attention}(QW_i^Q, E_i K W_i^K, F_i V W_i^V)$$

$$= \underbrace{\text{softmax}\left(\frac{QW_i^Q (E_i K W_i^K)^T}{\sqrt{d}} \right)} F_i V W_i^V$$

$n \times d$ $\qquad$ $k \times d$ $\qquad$ $k \times d$

$\tilde{P} \in \mathbb{R}^{n \times k}$

Note that the calculation above can be carried out in $O(ndk)$ time, which is linear in n if k is not a function of n.

⇒ Thm Let $k = c \min\left\{ \frac{9 d \log d}{\varepsilon^2}, \frac{5 \log n}{\varepsilon^2} \right\}$. There exists a way to sample $E_i, F_i \in \mathbb{R}^{k \times n}$ such that, for any row vector w of $QW_i^Q (KW_i^K)^T / \sqrt{d}$, we have that

$$\Pr\left(\| \mathrm{softmax}(wE_i^T) F_i V W_i^V - \mathrm{softmax}(w) V W_i^V \| \leq \varepsilon \| \mathrm{softmax}(w) V W_i^V \| \right)$$

$$< 1 - O(1).$$

⇒ The forms of E_i and F_i are

$$E_i = \delta R, \qquad F_i \quad e^{-\delta} R$$

where $R \in \mathbb{R}^{k \times n}$ is a matrix whose entries are sampled from the distribusion $\frac{1}{\sqrt{k}} \mathcal{N}(0, 1)$.

⇒ Note that the projection matrices can be shared between heads and layers. Moreover, we can use the same matrix for the keys and values.

- Experiments

⇒ Pretraining performance

- Measured with perplexity curves.
- Linformer with $k = 128$, $n = 512$ / $k = 256$, $n = 1024$ } are already on par with standard transformer
- The paper also evaluated projection matrix sharing schemes, and it found that performances were similar.
- If we fix k and increase n, we find that performance

degrades when the model is not trained to convergence. However, performances become about the same after convergence.

⇒ Finetuning performance

- Linformer with $n = 512$, $k = 128$ performed on par with RoBERTa
- $k = 256$ actually performed better.
- Best results were obtained by sharing one projection matrix across the whole model.