Efficient Pretraining and Finetuning Techniques

He He

NEW YORK UNIVERSITY

March 21, 2023

Logistics

- HW3 released: finetuning BERT!
- Proposal due today and we'll provide brief feedback on Gradescope (or come to OH)
- Midterm grades will be released soon.

Introduction

Plan for today:

- How to train larger models on larger data with less compute
- How to finetune larger models with less compute

Introduction

Plan for today:

- How to train larger models on larger data with less compute
- How to finetune larger models with less compute

Why care about efficiency?

- Practical reasons: training and running these models are expensive!
- Methods that help scaling may eventually leads to *better* models (e.g., transformers)

> *"The bitter lesson is based on the historical observations that 1) AI researchers have often tried to build knowledge into their agents, 2) this always helps in the short term, and is personally satisfying to the researcher, but 3) in the long run it plateaus and even inhibits further progress, and 4) breakthrough progress eventually arrives by an opposing approach based on scaling computation by search and learning."* — Richard Sutton "The bitter lesson"

Table of Contents

Transformer recap

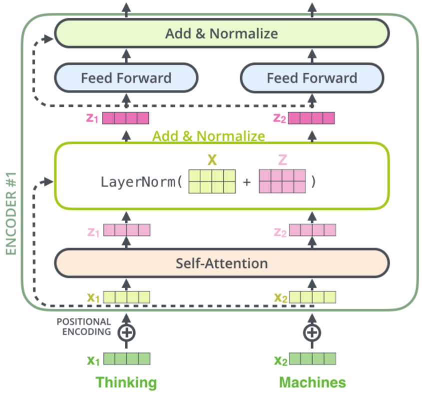


Figure: From The Illustrated Transformer

Which components require matrix multiplication?

Transformer recap

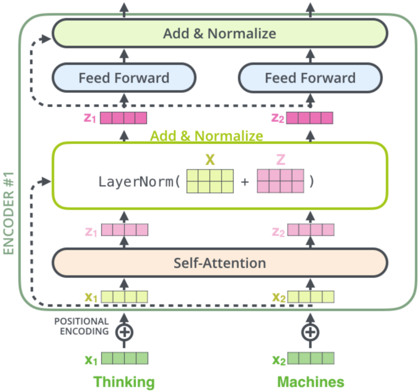


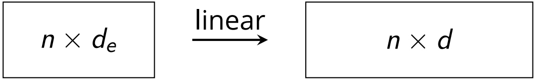
Figure: From The Illustrated Transformer

Which components require matrix multiplication?

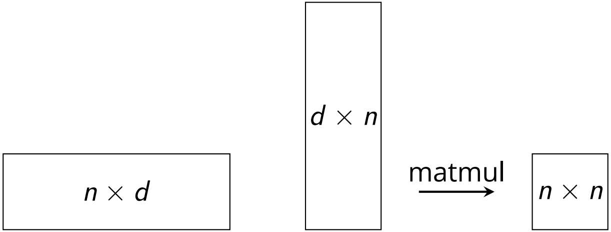
- Self-attention
  - Q,K,V projection
  - Scaled dot-product attention
- Feed-forward layer

Compute cost of transformers

Q, K, V projection:



Scaled dot-product attention:



Compute cost of transformers

Q, K, V projection:

$n \times d_e \xrightarrow{\text{linear}} n \times d \qquad O(n \times d_e \times d)$

Scaled dot-product attention:

$n \times d \quad d \times n \xrightarrow{\text{matmul}} n \times n \qquad O(d \times n^2)$

Compute cost of transformers

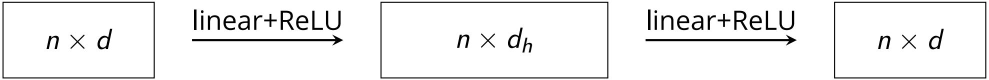
Feed-forward layer (GPT-2):

$$n \times d \xrightarrow{\text{linear+ReLU}} n \times d_h$$

$O(n \times d \times d_h)$

Compute cost of transformers

Feed-forward layer (GPT-2):



$O(n \times d \times d_h)$

- Two-layer FFN
- $d_h = 4d$ ($d > 1K$) by default in GPT-2
- Approximately half of the compute time

Improve efficiency of transformers

How to scale transformer models to larger number of parameters and larger data?

- Quantization (training and inference)
- Weight sharing (training and inference)
- Sparsely-activated models (training and inference)
- Pruning (inference)
- Distillation (inference)

Improve efficiency of transformers

This lecture: Improve efficiency of self-attention (for long sequences)

Improve efficiency of transformers

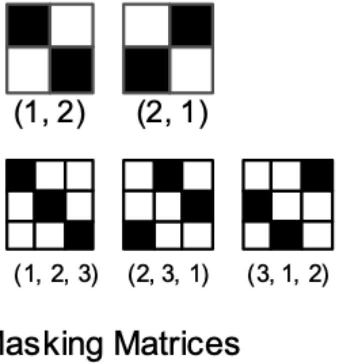
This lecture: Improve efficiency of self-attention (for long sequences)

Key idea: reduce the $O(n^2)$ time and memory cost

- Sparsify the attention matrix
  - Deterministic mask
  - Data-dependent mask
- Compress the key-value memory
  - Low-rank projection
  - Attention-based projection

Limiting receptive field of self-attention

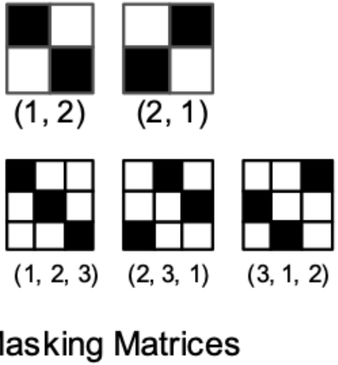
Blockwise self-attention [Qiu et al., 2020]: attention within a local window



- Divide a $n \times n$ matrix into $m \times m$ blocks
- Compute one block per row and mask the rest (i.e. set to 0)
- Allocate groups of attention heads to each mask configuration
  - Which configuration should use more attention heads?

Limiting receptive field of self-attention

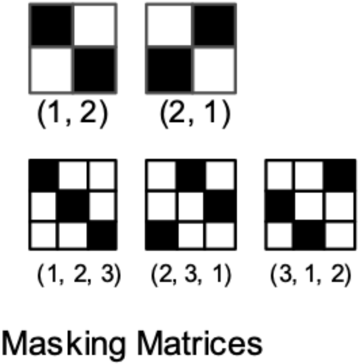
Blockwise self-attention [Qiu et al., 2020]: attention within a local window



- Divide a $n \times n$ matrix into $m \times m$ blocks
- Compute one block per row and mask the rest (i.e. set to 0)
- Allocate groups of attention heads to each mask configuration
  - Which configuration should use more attention heads?
- What's the time complexity?

Limiting receptive field of self-attention

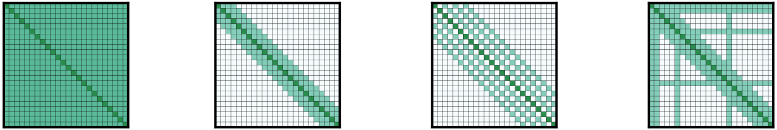
Blockwise self-attention [Qiu et al., 2020]: attention within a local window



- Divide a $n \times n$ matrix into $m \times m$ blocks
- Compute one block per row and mask the rest (i.e. set to 0)
- Allocate groups of attention heads to each mask configuration
  - Which configuration should use more attention heads?
- What's the time complexity?
  - $O(n^2) \longrightarrow O(n^2/m)$

Limiting receptive field of self-attention

Longformer [Beltagy et al., 2020]: attention within a local window

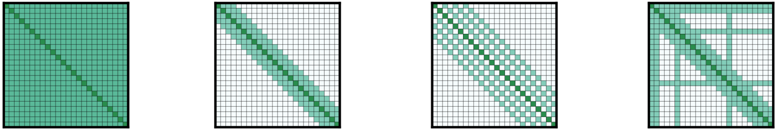


(a) Full n^2 attention (b) Sliding window attention (c) Dilated sliding window (d) Global+sliding window

- Sliding window: attending to a *local* window of size w around each token $O(n \times w)$
- Dilated sliding window: reaching *longer range* with a larger window size with gaps
- Global window: *full attention* on specific tokens, e.g., `[CLS]` in BERT

Limiting receptive field of self-attention

Longformer [Beltagy et al., 2020]: attention within a local window



(a) Full n^2 attention (b) Sliding window attention (c) Dilated sliding window (d) Global+sliding window

- Sliding window: attending to a *local* window of size w around each token
  $O(n \times w)$
- Dilated sliding window: reaching *longer range* with a larger window size with gaps
- Global window: *full attention* on specific tokens, e.g., `[CLS]` in BERT
- Details: balancing efficiency and performance
  - Adding dilation on some heads
  - Using small window size on lower layers and larger ones on higher layers

Limiting receptive field of self-attention

Reformer [Kitaev et al., 2020]: attention within an adaptive local window

Key idea:

- We want to compute the attention scores for a query q_i:

$$a_i = \mathrm{softmax}\left(\left[\frac{q_i \cdot k_1}{\sqrt{d}}, \ldots, \frac{q_i \cdot k_n}{\sqrt{d}}\right]\right)$$

Limiting receptive field of self-attention

Reformer [Kitaev et al., 2020]: attention within an adaptive local window

Key idea:

- We want to compute the attention scores for a query q_i:

$$a_i = \mathrm{softmax}\left(\left[\frac{q_i \cdot k_1}{\sqrt{d}}, \ldots, \frac{q_i \cdot k_n}{\sqrt{d}}\right]\right)$$

- **Goal**: can we approximate $a_i \in \mathbb{R}^n$ by computing $< n$ dot products?

Limiting receptive field of self-attention

Reformer [Kitaev et al., 2020]: attention within an adaptive local window

Key idea:

- We want to compute the attention scores for a query q_i:

$$a_i = \mathrm{softmax}\left(\left[\frac{q_i \cdot k_1}{\sqrt{d}}, \ldots, \frac{q_i \cdot k_n}{\sqrt{d}}\right]\right)$$

- **Goal**: can we approximate $a_i \in \mathbb{R}^n$ by computing $< n$ dot products?
- Which dot products (k_j's) have large influence on the value a_i?

Limiting receptive field of self-attention

Reformer [Kitaev et al., 2020]: attention within an adaptive local window

Key idea:

- We want to compute the attention scores for a query q_i:

$$a_i = \operatorname{softmax}\left(\left[\frac{q_i \cdot k_1}{\sqrt{d}}, \ldots, \frac{q_i \cdot k_n}{\sqrt{d}}\right]\right)$$

- **Goal**: can we approximate $a_i \in \mathbb{R}^n$ by computing $< n$ dot products?
- Which dot products (k_j's) have large influence on the value a_i?
  - k_j's that has large dot products with ("close to") q_i

Limiting receptive field of self-attention

Reformer [Kitaev et al., 2020]: attention within an adaptive local window

Key idea:

- We want to compute the attention scores for a query q_i:

$$a_i = \mathrm{softmax}\left(\left[\frac{q_i \cdot k_1}{\sqrt{d}}, \ldots, \frac{q_i \cdot k_n}{\sqrt{d}}\right]\right)$$

- **Goal**: can we approximate $a_i \in \mathbb{R}^n$ by computing $< n$ dot products?
- Which dot products (k_j's) have large influence on the value a_i?
  - k_j's that has large dot products with ("close to") q_i
- How do we find such k_j's (nearest neighbors of q_i) fast?

Limiting receptive field of self-attention

Reformer [Kitaev et al., 2020]: attention within an adaptive local window

Key idea:

- We want to compute the attention scores for a query q_i:

$$a_i = \mathrm{softmax}\left(\left[\frac{q_i \cdot k_1}{\sqrt{d}}, \dots, \frac{q_i \cdot k_n}{\sqrt{d}}\right]\right)$$

- **Goal**: can we approximate $a_i \in \mathbb{R}^n$ by computing $< n$ dot products?
- Which dot products (k_j's) have large influence on the value a_i?
  - k_j's that has large dot products with ("close to") q_i
- How do we find such k_j's (nearest neighbors of q_i) fast?
  - **Locality sensitive hashing** (LSH): close vectors are put in the same bucket: $h(k_s) = h(k_t)$ if k_s is close to k_t

Limiting receptive field of self-attention

Reformer [Kitaev et al., 2020]: attention within an adaptive local window

Key idea:

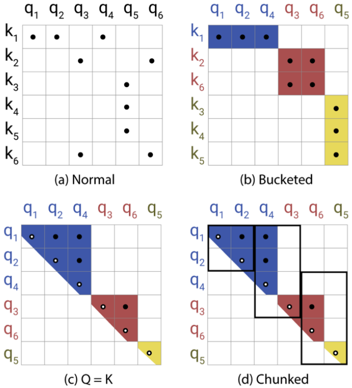
- We want to compute the attention scores for a query q_i:

$$a_i = \mathrm{softmax}\left(\left[\frac{q_i \cdot k_1}{\sqrt{d}}, \ldots, \frac{q_i \cdot k_n}{\sqrt{d}}\right]\right)$$

- **Goal**: can we approximate $a_i \in \mathbb{R}^n$ by computing $< n$ dot products?
- Which dot products (k_j's) have large influence on the value a_i?
  - k_j's that has large dot products with ("close to") q_i
- How do we find such k_j's (nearest neighbors of q_i) fast?
  - **Locality sensitive hashing** (LSH): close vectors are put in the same bucket: $h(k_s) = h(k_t)$ if k_s is close to k_t
- Compute attention between q_i and k_i only if they fall in the same hash bucket

Limiting receptive field of self-attention

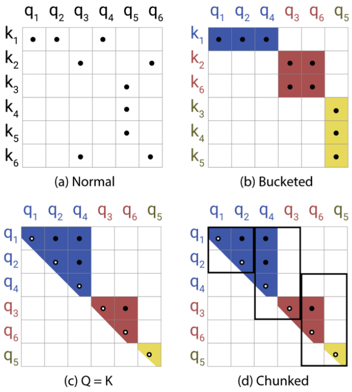
Reformer [Kitaev et al., 2020] implementation



(a) Leverage the sparsity of the attention matrix

Limiting receptive field of self-attention

Reformer [Kitaev et al., 2020] implementation

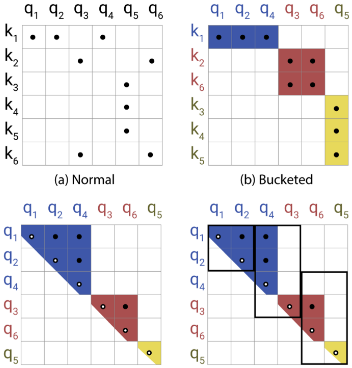


(a) Leverage the sparsity of the attention matrix

Challenge 1: find the nearest neighbors

Limiting receptive field of self-attention

Reformer [Kitaev et al., 2020] implementation



(a) Normal (b) Bucketed (c) Q = K (d) Chunked

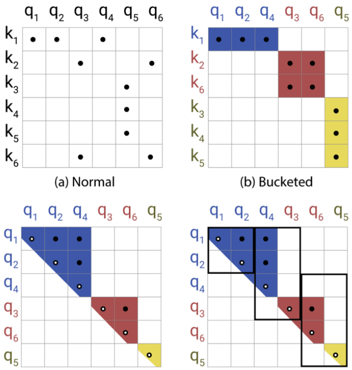
(a) Leverage the sparsity of the attention matrix

Challenge 1: find the nearest neighbors

(b) Sort q_i's and k_i's by their hash codes such that vectors in the same bucket are grouped

Limiting receptive field of self-attention

Reformer [Kitaev et al., 2020] implementation



(a) Leverage the sparsity of the attention matrix

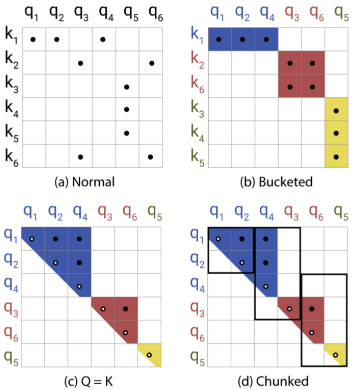
Challenge 1: find the nearest neighbors

(b) Sort q_i's and k_i's by their hash codes such that vectors in the same bucket are grouped

Challenge 2: batch the computation

Limiting receptive field of self-attention

Reformer [Kitaev et al., 2020] implementation



(a) Leverage the sparsity of the attention matrix

Challenge 1: find the nearest neighbors

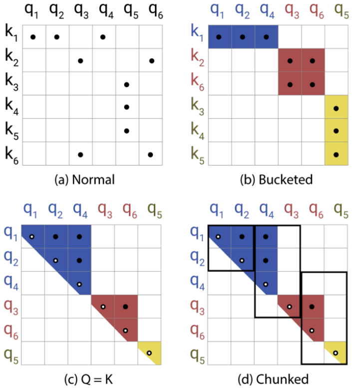
(b) Sort q_i's and k_i's by their hash codes such that vectors in the same bucket are grouped

Challenge 2: batch the computation

(c) Set $k_i = q_i$ such that similar vectors are grouped along the diagonal

Limiting receptive field of self-attention

Reformer [Kitaev et al., 2020] implementation



(a) Leverage the sparsity of the attention matrix

Challenge 1: find the nearest neighbors

(b) Sort q_i's and k_i's by their hash codes such that vectors in the same bucket are grouped

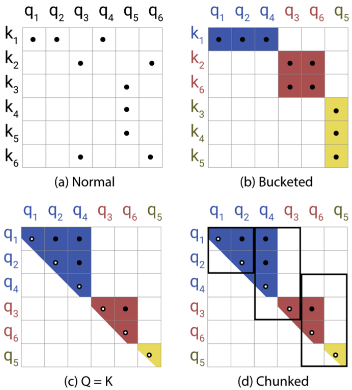
Challenge 2: batch the computation

(c) Set $k_i = q_i$ such that similar vectors are grouped along the diagonal

(d) Chunk it by equal size (cf. blockwise attention)
a group may be split in two chunks

Limiting receptive field of self-attention

Reformer [Kitaev et al., 2020] implementation



(a) Leverage the sparsity of the attention matrix

Challenge 1: find the nearest neighbors

(b) Sort q_i's and k_i's by their hash codes such that vectors in the same bucket are grouped

Challenge 2: batch the computation

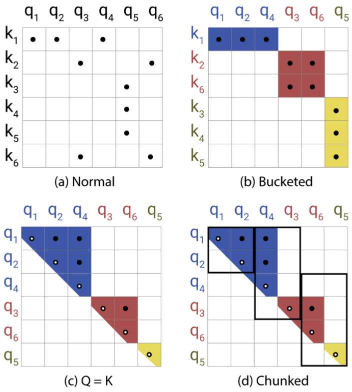
(c) Set $k_i = q_i$ such that similar vectors are grouped along the diagonal

(d) Chunk it by equal size (cf. blockwise attention) *a group may be split in two chunks*

- Each chunk attends to itself and the previous chunk

Limiting receptive field of self-attention

Reformer [Kitaev et al., 2020] implementation



(a) Leverage the sparsity of the attention matrix

Challenge 1: find the nearest neighbors

(b) Sort q_i's and k_i's by their hash codes such that vectors in the same bucket are grouped

Challenge 2: batch the computation

(c) Set $k_i = q_i$ such that similar vectors are grouped along the diagonal

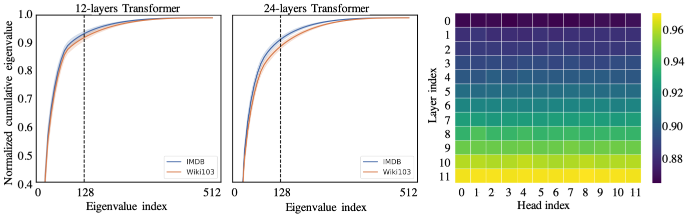
(d) Chunk it by equal size (cf. blockwise attention)
a group may be split in two chunks

- Each chunk attends to itself and the previous chunk

Better accuracy with more hashes

Summarize the KV memory

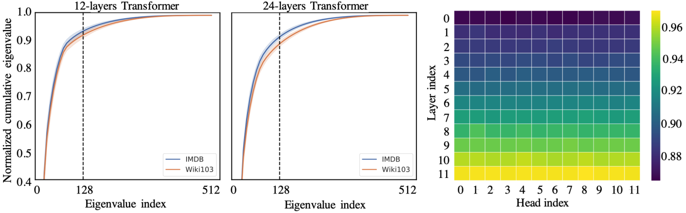
Self-attention is low rank [Wang et al., 2020]



- Left: cumulative eigenvalues of pretrained transformer with $n = 512$

Summarize the KV memory

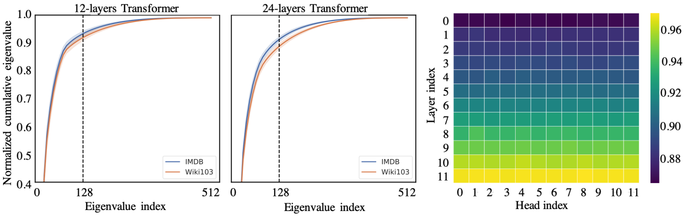
Self-attention is low rank [Wang et al., 2020]



- Left: cumulative eigenvalues of pretrained transformer with $n = 512$
  - Most information in the attention matrix can be recovered by the top 128 eigenvectors

Summarize the KV memory

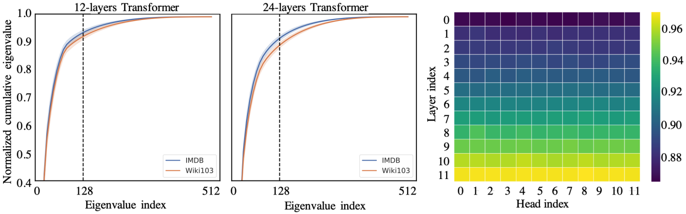
Self-attention is low rank [Wang et al., 2020]



- Left: cumulative eigenvalues of pretrained transformer with $n = 512$
  - Most information in the attention matrix can be recovered by the top 128 eigenvectors
- Right: cumulative eigenvalues of the top 128 eigenvalues across layers

Summarize the KV memory

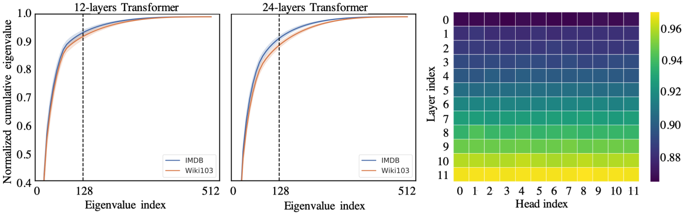
Self-attention is low rank [Wang et al., 2020]



- Left: cumulative eigenvalues of pretrained transformer with $n = 512$
  - Most information in the attention matrix can be recovered by the top 128 eigenvectors
- Right: cumulative eigenvalues of the top 128 eigenvalues across layers
  - Higher layers are more low-rank

Summarize the KV memory

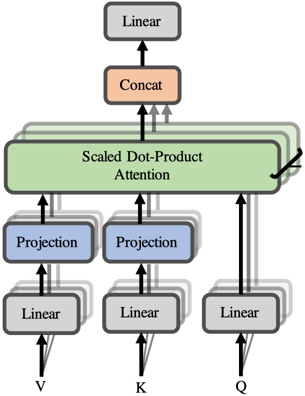
Self-attention is low rank [Wang et al., 2020]



- Left: cumulative eigenvalues of pretrained transformer with $n = 512$
  - Most information in the attention matrix can be recovered by the top 128 eigenvectors
- Right: cumulative eigenvalues of the top 128 eigenvalues across layers
  - Higher layers are more low-rank
- **Idea**: instead of attending to n tokens, attend to k principal components

Summarize the KV memory

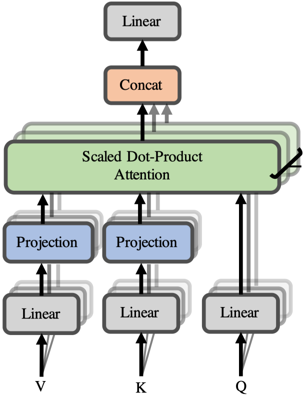
Linformer [Wang et al., 2020]: compute self-attention in a lower dimension



- Reduce dimensionality of the "memory": Map K, V from $n \times d$ to $k \times d$

Summarize the KV memory

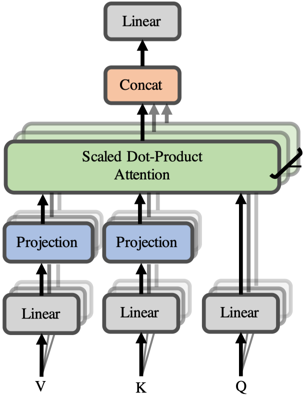
Linformer [Wang et al., 2020]: compute self-attention in a lower dimension



- Reduce dimensionality of the "memory": Map K, V from $n \times d$ to $k \times d$
- Attend to the lower-dimensional memory:
  $\mathrm{softmax}\left(Q_{n\times d}K^{T}_{k\times d}/\sqrt{d}\right)$

Summarize the KV memory

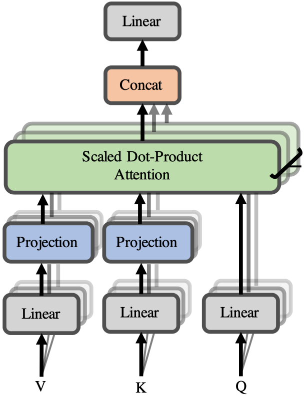
Linformer [Wang et al., 2020]: compute self-attention in a lower dimension



- Reduce dimensionality of the "memory": Map K, V from $n \times d$ to $k \times d$
- Attend to the lower-dimensional memory: $\mathrm{softmax}\left(Q_{n\times d} K^T_{k\times d} / \sqrt{d}\right)$
  - What's the dimension of the attention matrix?

Summarize the KV memory

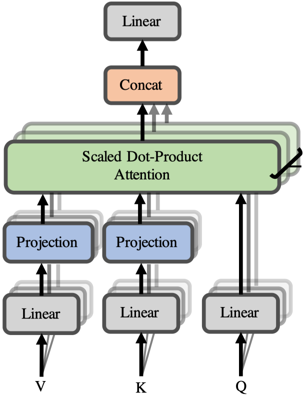
Linformer [Wang et al., 2020]: compute self-attention in a lower dimension



- Reduce dimensionality of the "memory": Map K, V from $n \times d$ to $k \times d$
- Attend to the lower-dimensional memory: $\mathrm{softmax}\left(Q_{n\times d}K^T_{k\times d}/\sqrt{d}\right)$
  - What's the dimension of the attention matrix?
  - What's the dimension of the self-attention output?

Summarize the KV memory

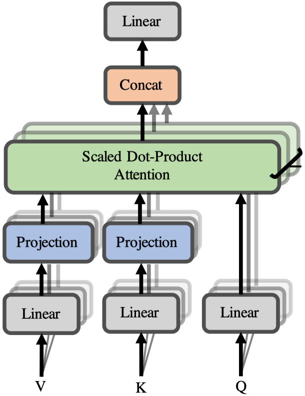
Linformer [Wang et al., 2020]: compute self-attention in a lower dimension



- Reduce dimensionality of the "memory": Map K, V from $n \times d$ to $k \times d$
- Attend to the lower-dimensional memory: $\mathrm{softmax}\left(Q_{n\times d} K_{k\times d}^{T} / \sqrt{d}\right)$
  - What's the dimension of the attention matrix?
  - What's the dimension of the self-attention output?
- Computation cost: $O(nk)$

Summarize the KV memory

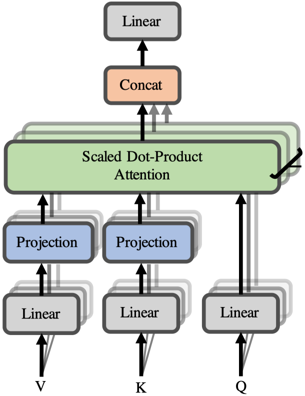
Linformer [Wang et al., 2020]: compute self-attention in a lower dimension



- Reduce dimensionality of the "memory": Map K, V from $n \times d$ to $k \times d$
- Attend to the lower-dimensional memory: $\mathrm{softmax}\left(Q_{n\times d} K_{k\times d}^T / \sqrt{d}\right)$
  - What's the dimension of the attention matrix?
  - What's the dimension of the self-attention output?
- Computation cost: $O(nk)$
- Downside of uisng Linformer as a decoder?

Summarize the KV memory

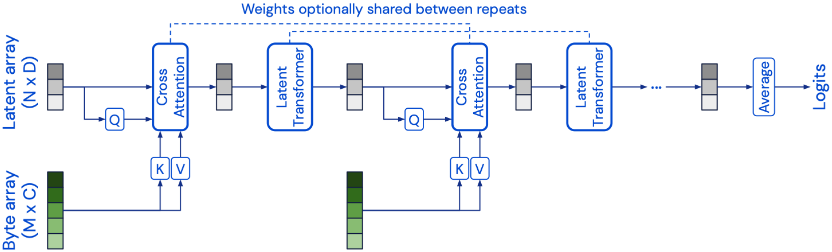
Linformer [Wang et al., 2020]: compute self-attention in a lower dimension



- Reduce dimensionality of the "memory": Map K, V from $n \times d$ to $k \times d$
- Attend to the lower-dimensional memory: $\mathrm{softmax}\left(Q_{n\times d} K^T_{k\times d}/\sqrt{d}\right)$
  - What's the dimension of the attention matrix?
  - What's the dimension of the self-attention output?
- Computation cost: $O(nk)$
- Downside of uisng Linformer as a decoder?
  - Unclear how to mask: past and future are mixed

Summarize the KV memory

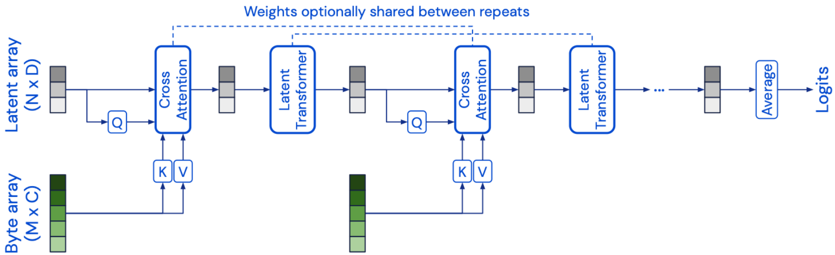
Perceiver [Jaegle et al., 2021]: use latent states to compress the KV memory



- Use latent states ($k \times d_s$) as queries to attend to K,V ($n \times d$) $\longrightarrow$ lower dimensional states ($k \times d_s$)

Summarize the KV memory

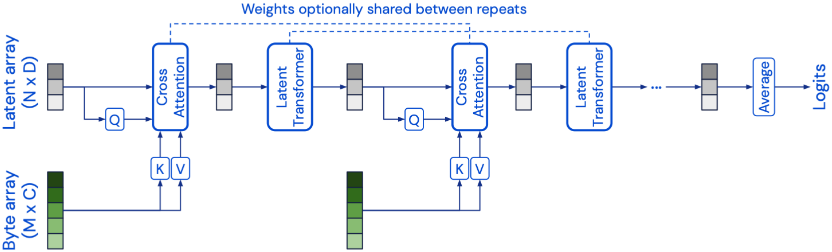
Perceiver [Jaegle et al., 2021]: use latent states to compress the KV memory



- Use latent states ($k \times d_s$) as queries to attend to K,V ($n \times d$) $\longrightarrow$ lower dimensional states ($k \times d_s$)
- Stack self-attention layers on *latent states*: decoupling depth and input size

Summarize the KV memory

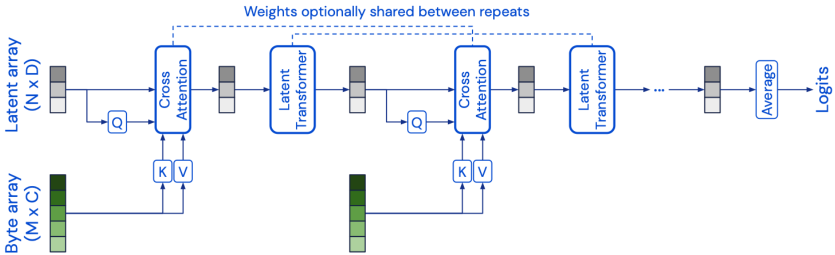
Perceiver [Jaegle et al., 2021]: use latent states to compress the KV memory



- Use latent states ($k \times d_s$) as queries to attend to K,V ($n \times d$) $\longrightarrow$ lower dimensional states ($k \times d_s$)
- Stack self-attention layers on *latent states*: decoupling depth and input size
- Map to latent states using cross attention: $O(nm)$

Summarize the KV memory

Perceiver [Jaegle et al., 2021]: use latent states to compress the KV memory



- Use latent states ($k \times d_s$) as queries to attend to K,V ($n \times d$) $\longrightarrow$ lower dimensional states ($k \times d_s$)
- Stack self-attention layers on *latent states*: decoupling depth and input size
- Map to latent states using cross attention: $O(nm)$
- Self-attention layers: $O(Lm^2)$

Summary

Improve the quadratic time and space complexity of self-attention

- Sparsify the attention matrix
- Compress the KV memory

Summary

Improve the quadratic time and space complexity of self-attention

- Sparsify the attention matrix
- Compress the KV memory

Bad news: Most techniques are not widely used in large pretrained models now. Why?

- Improvement in time/space complexity doesn't always translate to real time/space savings
- These techniques often breaks structure and sacrifice the batching ability on GPUs
- Only see improvement on very long sequences

Summary

Improve the quadratic time and space complexity of self-attention

- Sparsify the attention matrix
- Compress the KV memory

Bad news: Most techniques are not widely used in large pretrained models now. Why?

- Improvement in time/space complexity doesn't always translate to real time/space savings
- These techniques often breaks structure and sacrifice the batching ability on GPUs
- Only see improvement on very long sequences

Takeaways:

- Attention structure is important
- Low-rank techniques

Table of Contents

Improve finetuning efficiency

Problem:

- In NLP, typically all parameters of the pretrained models (e.g., BERT) are finetuned, which is expensive for large models.
- Saving and loading finetuned models for different tasks is costly.

Improve finetuning efficiency

Problem:

- In NLP, typically all parameters of the pretrained models (e.g., BERT) are finetuned, which is expensive for large models.
- Saving and loading finetuned models for different tasks is costly.

Can we finetune a smaller number of parameters to achieve performance similar to full finetuning?

Improve finetuning efficiency

Problem:

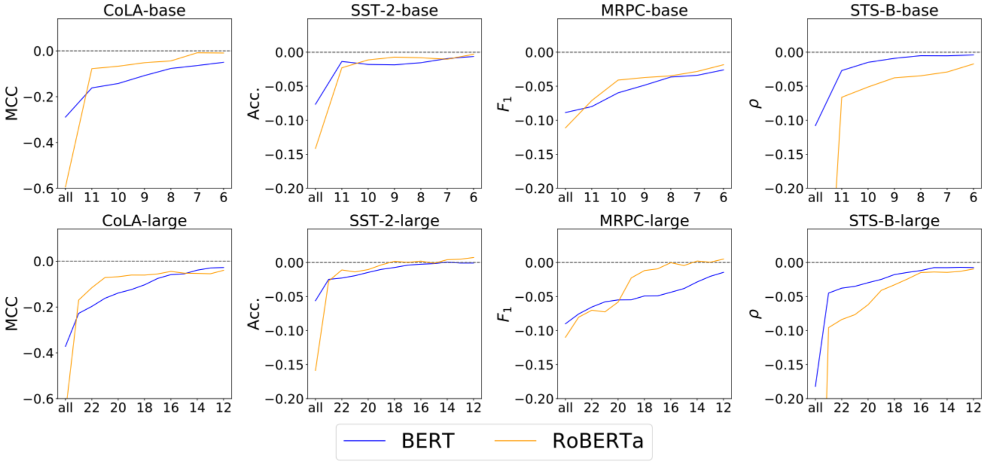
- In NLP, typically all parameters of the pretrained models (e.g., BERT) are finetuned, which is expensive for large models.
- Saving and loading finetuned models for different tasks is costly.

Can we finetune a smaller number of parameters to achieve performance similar to full finetuning?

- Select a subset of parameters from the pretrained weights to update
- Add a small number of parameters to adapte the (frozen) pretrained model

Finetune a subset of parameters

Freezing the first X layers [Lee et al., 2019]



A fourth of the layers need to be fine-tuned to obtain 90% of the performance.

Finetune a subset of parameters

BitFit [Ben-Zaken et al., 2022]: only finetune the bias term (0.1% of the parameters)

Bias terms in QKV projection

$$\mathbf{Q}^{m,\ell}(\mathbf{x}) = \mathbf{W}_q^{m,\ell}\mathbf{x} + \mathbf{b}_q^{m,\ell}$$
$$\mathbf{K}^{m,\ell}(\mathbf{x}) = \mathbf{W}_k^{m,\ell}\mathbf{x} + \mathbf{b}_k^{m,\ell}$$
$$\mathbf{V}^{m,\ell}(\mathbf{x}) = \mathbf{W}_v^{m,\ell}\mathbf{x} + \mathbf{b}_v^{m,\ell}$$

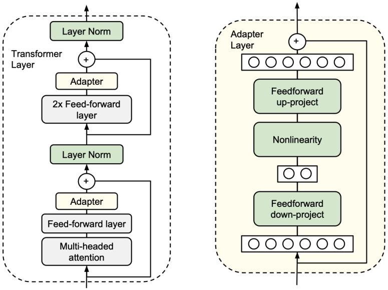
Bias terms in MLP layers

$$\mathbf{h}_2^\ell = \text{Dropout}\left(\mathbf{W}_{m_1}^\ell \cdot \mathbf{h}_1^\ell + \mathbf{b}_{m_1}^\ell\right)$$
$$\mathbf{h}_3^\ell = \mathbf{g}_{LN_1}^\ell \odot \frac{(\mathbf{h}_2^\ell + \mathbf{x}) - \mu}{\sigma} + \mathbf{b}_{LN_1}^\ell$$
$$\mathbf{h}_4^\ell = \text{GELU}\left(\mathbf{W}_{m_2}^\ell \cdot \mathbf{h}_3^\ell + \mathbf{b}_{m_2}^\ell\right)$$
$$\mathbf{h}_5^\ell = \text{Dropout}\left(\mathbf{W}_{m_3}^\ell \cdot \mathbf{h}_4^\ell + \mathbf{b}_{m_3}^\ell\right)$$
$$\text{out}^\ell = \mathbf{g}_{LN_2}^\ell \odot \frac{(\mathbf{h}_5^\ell + \mathbf{h}_3^\ell) - \mu}{\sigma} + \mathbf{b}_{LN_2}^\ell$$

Result: 80.9 (BitFit, 0.08% params) vs 81.8 (full finetuning) on GLUE

Adapt the frozen pretrained model

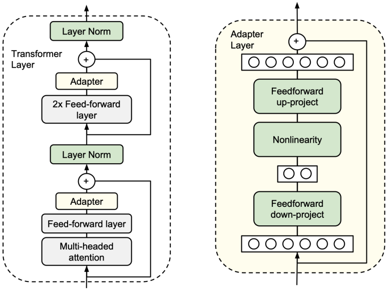
Adapter [Houlsby et al., 2019]: insert small networks to the pretrained model



- Insert learnable ”adapters” in-between layers
- Adapters uses a bottleneck structure to reduce parameters
- Adapters uses a skip-connection

Adapt the frozen pretrained model

Adapter [Houlsby et al., 2019]: insert small networks to the pretrained model

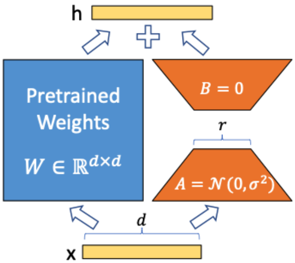


- Insert learnable ”adapters” in-between layers
- Adapters uses a bottleneck structure to reduce parameters
- Adapters uses a skip-connection such that it can be “reduced” to the original frozen model

Result: less than 0.4% performance drop with 3% more parameters on GLUE

Adapt the frozen pretrained model

LoRA [Hu et al., 2021]: add low-rank matrices as additional parameters



Hypothesis: weight matrices are low rank

Adapters: For any matrix multiplication $h = W_0 x$, we modify it to

$$h = W_0 x + \Delta W x = W_0 x + BAx$$

- $W_0 \in \mathbb{R}^{d \times k}, B \in \mathbb{R}^{d \times r}, A \in \mathbb{R}^{r \times k} (r \ll k)$
- Initialization: $BA = 0$
- Can be applied to any weight matrices, e.g., QKV projection matrices

Adapt the frozen pretrained model

Compare LoRA and the original adapters:

- LoRA recovers full finetuning by increasing r

  Adapter recovers an MLP model with increasing params

Adapt the frozen pretrained model

Compare LoRA and the original adapters:

- LoRA recovers full finetuning by increasing r
  Adapter recovers an MLP model with increasing params
- LoRA has no additional inference cost

Adapt the frozen pretrained model

Compare LoRA and the original adapters:

- LoRA recovers full finetuning by increasing r

  Adapter recovers an MLP model with increasing params
- LoRA has no additional inference cost by setting $W_0 \leftarrow W_0 + BA$ (doesn't work for multiple tasks)

  Adapter incurs additional inference cost due to the added params

The most widely used efficient finetuning method on very large models ($>$100B).

Summary

Reduce finetuning cost by reducing the number of parameters to update

- Finetune a subset of parameters
- Finetune an additional adapters inserted to the model

Not widely used for SOTA large models, but used sometimes in resource-constrained settings.

Other ways to adapt the model without parameter update: prompting, in-context learning (later)

Lots of open research questions!