Pretraining and Finetuning (continued)

He He

NEW YORK UNIVERSITY

October 16, 2024

Plan for today

- Subword tokenization
- Efficient pre-training
- Parameter efficient finetuning

Table of Contents

Tokenization

Goal: represent input string as a sequence of (meaningful) symbols

Approach so far: splitting string into “words”

Tokenization

Goal: represent input string as a sequence of (meaningful) symbols

Approach so far: splitting string into “words”

Challenges:

- What about words unseen in training?
  - Use `UNK` (limitations?)

Tokenization

Goal: represent input string as a sequence of (meaningful) symbols

Approach so far: splitting string into "words"

Challenges:

- What about words unseen in training?
  - Use `UNK` (limitations?)
  - We can guess word meaning based on its form, e.g., degrade $\rightarrow$ degradation

Tokenization

Goal: represent input string as a sequence of (meaningful) symbols

Approach so far: splitting string into "words"

Challenges:

- What about words unseen in training?
  - Use `UNK` (limitations?)
  - We can guess word meaning based on its form, e.g., degrade $\rightarrow$ degradation
- What about non-English languages? (Thai, Turkish, Chinese, Java, Lean etc.)
  - Take each character as a token (limitations?)

Tokenization

Goal: represent input string as a sequence of (meaningful) symbols

Approach so far: splitting string into "words"

Challenges:

- What about words unseen in training?
  - Use `UNK` (limitations?)
  - We can guess word meaning based on its form, e.g., degrade $\rightarrow$ degradation
- What about non-English languages? (Thai, Turkish, Chinese, Java, Lean etc.)
  - Take each character as a token (limitations?)
  - Very long sequences; model needs to learn to compose characters

Subword tokenization: byte pair encoding

What is a “token”?

- A sequence of characters that carries some meaning and *re-occurs* in a corpora
- Can we find these character units based on their *frequency*?

Subword tokenization: byte pair encoding

What is a “token”?

- A sequence of characters that carries some meaning and *re-occurs* in a corpora
- Can we find these character units based on their *frequency*?

BPE tokenization: A middle ground between ”words” and ”characters”

- Origin: a compression algorithm that iteratively replace the most common character sequences by a single symbol
  - Output: compressed text + a look-up table
- Start with individual characters (or bytes) as tokens
- Count the frequency of every consecutive pair of tokens
- Merge the most frequent pair of tokens and treat them as a single token
- Update the text with the new token and repeat the process

BPE Example (Step-by-Step)

Initial Sequence:

- Words: `banana`, `band`, `ban`
- Initial tokenization:
  - `b a n a n a`
  - `b a n d`
  - `b a n`

BPE Example (Step-by-Step)

Initial Sequence:

- Words: `banana`, `band`, `ban`
- Initial tokenization:
  - `b a n a n a`
  - `b a n d`
  - `b a n`

Step 1: Count Pairs

- Most frequent pair: `a n`

BPE Example (Step-by-Step)

Initial Sequence:

- Words: `banana`, `band`, `ban`
- Initial tokenization:
  - `b a n a n a`
  - `b a n d`
  - `b a n`

Step 1: Count Pairs

- Most frequent pair: `a n`

Step 2: Merge

- New token: `an`
- Updated tokenization:
  - `b an an a`
  - `b an d`
  - `b an`

BPE Example (Step-by-Step)

Step 3: Count Pairs Again

- Updated tokenization:
  - `b an an a`
  - `b an d`
  - `b an`
- Most frequent pair: `b an`

Step 4: Merge

- New token: `ban`
- Updated tokenization:
  - `ban an a`
  - `ban d`
  - `ban`

BPE: practicalities

- Repeat the process until the desired number of merges or vocabulary size is reached (a hyperparameter to decide).
- Typically vocabulary sizes are 32-64K
- Break ties deterministically, e.g., lexicographical order, occurrence in the corpus etc.
- Use bytes as the initial tokens (adopted by GPT-2)

Table of Contents

Overview

Approaches to speed up pretraining

Overview

Approaches to speed up pretraining

- Reduce model size
- Design more sample-efficient learning objectives
- Improve efficiency of self-attention
- Improve system-level efficiency

Approach 1: Reduce model size

Idea 1: reduce the number of parameters

ALBERT (a lite BERT) [Lan et al., 2020]

- **Factorization**:
  - Recall that in Transformer, we first need to map the one-hot encoding (of size V) of a token to Q, K, V embeddings (of size H)
  - The number of parameters is $V \times H$
  - We can instead first map it to a lower-dim space (of size E) so that the number of params is $V \times E + E \times H$

Approach 1: Reduce model size

Idea 1: reduce the number of parameters

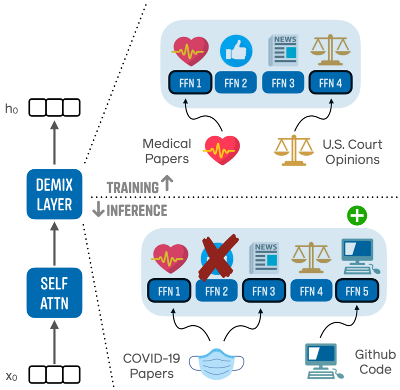
ALBERT (a lite BERT) [Lan et al., 2020]

- **Parameter sharing**:
  - Share feedforward network weights across layers
  - Share self-attention weights across layers
  - ALBERT: share all params across layers

Approach 1: Reduce model size

Idea 2: reduce interaction among parameters (sparse/modular architectures)

DEMix [Gururangan et al., 2022]



- Replace the FFN layer with an ensemble of n experts
- Route examples to experts corresponding to its domain determinstically

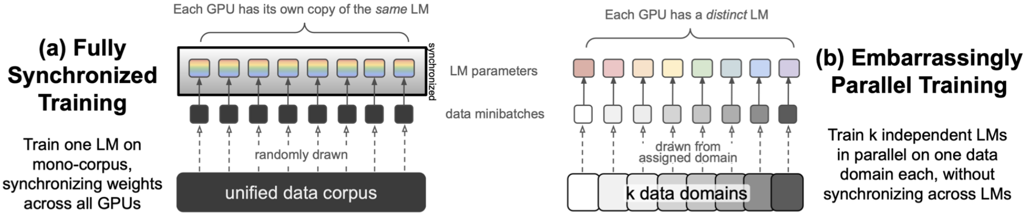
$$\mathrm{FFN}(h) = \sum_{i=1}^{n} \mathbb{I}[x \in \text{domain } i]\mathrm{FFN}_i(x)$$

- Only a subset of params are active for each example/batch

Approach 1: Reduce model size

Idea 2: reduce interaction among parameters (sparse/modular architectures)

Branch-Train-Merge [Li et al., 2022]



- Train domain experts in parallel and ensemble them (or take weighted average of their parameters)
- Reduce synchronization among GPUs at the cost of increased model size
- Easy to expand/remove domain experts

Approach 2: design sample-efficient learning objectives

ALBERT: Inter-sentence coherence loss

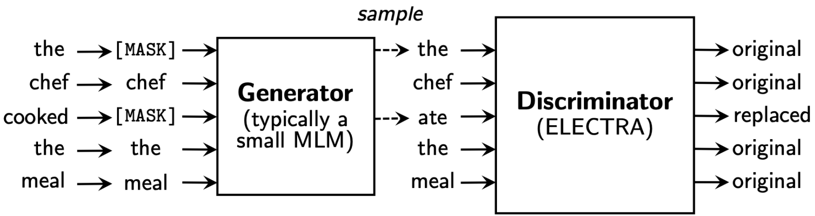
- Motivation: the next sentence prediction task is too easy
- Design hard negative examples
- Input: take two consecutive sentences, swap their order randomly
- Output: predict if they are in natural order

  I went home. SEP *I slept.* +1

  I slept. SEP *I went home.* -1
- Model needs to learn temporal order of events (commonsense, causality etc.)

Approach 2: design sample-efficient learning objectives

ELECTRA [Clark et al., 2020]: discriminate from true vs guessed tokens



- First train the generator for n steps using the MLM objective.
- Freeze generator weights. Train the discriminator using the sequence classification objective. Keep discriminator for finetuning.
- Comparison with MLM: predict at every position; hard negative examples.

Approach 2: design sample-efficient learning objectives

ELECTRA result:

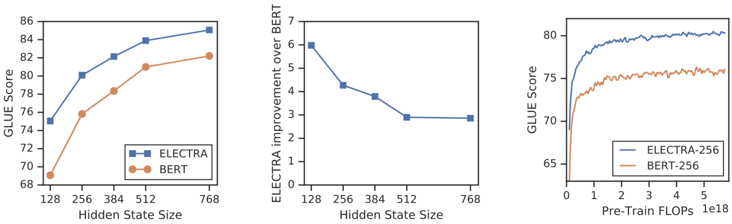


Figure: Finetuning result on the GLUE benchmark

- Larger improvement at smaller model sizes
- Faster training
- An effective approach if you don't have large compute for pretraining

Approach 3: alternatives to self-attention

Transformer recap

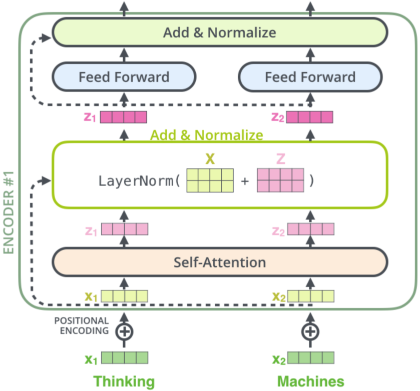


Figure: From The Illustrated Transformer

Which components require matrix multiplication?

Approach 3: alternatives to self-attention

Transformer recap

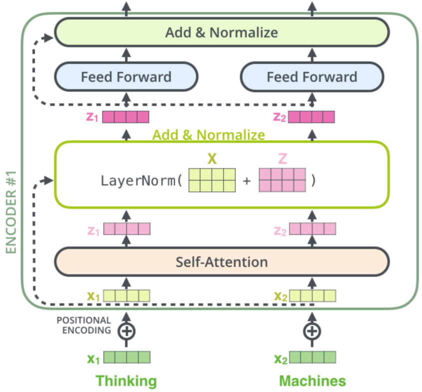


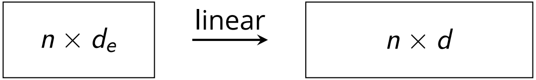
Figure: From The Illustrated Transformer

Which components require matrix multiplication?

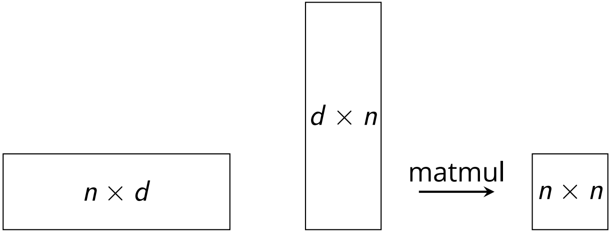
- Self-attention
  - Q,K,V projection
  - Scaled dot-product attention
- Feed-forward layer

Compute cost of transformers

Q, K, V projection:



Scaled dot-product attention:



Compute cost of transformers

Q, K, V projection:

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

Scaled dot-product attention:

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

Compute cost of transformers

Feed-forward layer (GPT-2):

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

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

- Two-layer FFN
- $d_h = 4d$ $(d > 1K)$ by default in GPT-2
- Approximately half of the compute time
- Concept check: how to reduce compute in FFN?

Compute cost of transformers

Feed-forward layer (GPT-2):

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

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

- Two-layer FFN
- $d_h = 4d$ ($d > 1K$) by default in GPT-2
- Approximately half of the compute time
- Concept check: how to reduce compute in FFN?

Improve efficiency of self-attention (for long sequences)

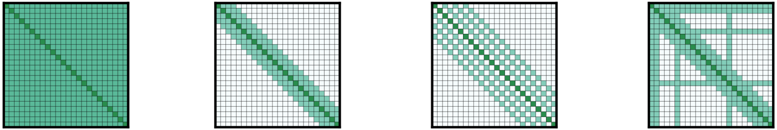
Improve efficiency of self-attention (for long sequences)

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

- Sparsify the attention matrix
    - Deterministic mask
    - Data-dependent mask (Reformer [Kitaev et al., 2020])
- Compress the key-value memory
    - Low-rank projection

Sparse attention

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

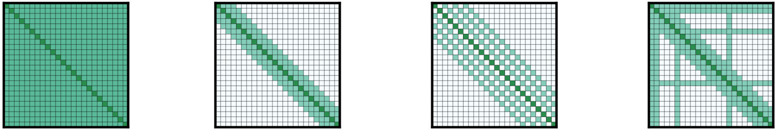


(a) Full n^2 attention (b) Sliding window attention (c) Dilated sliding window (d) Global+sliding window

- 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

Sparse attention

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

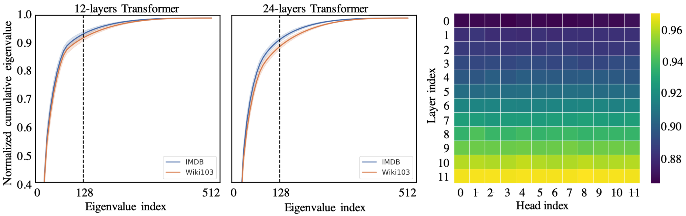


(a) Full n^2 attention (b) Sliding window attention (c) Dilated sliding window (d) Global+sliding window

- 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

Compress the KV memory

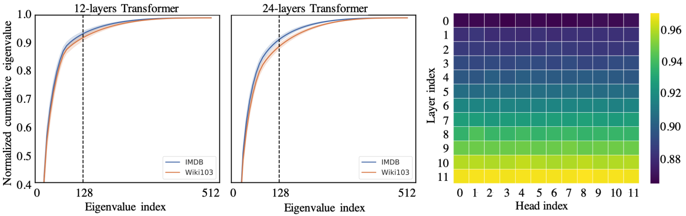
Self-attention is low rank [Wang et al., 2020]



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

Compress the KV memory

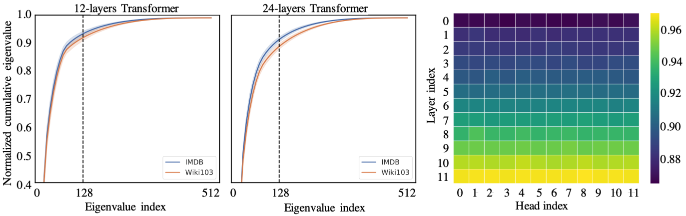
Self-attention is low rank [Wang et al., 2020]



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

Compress the KV memory

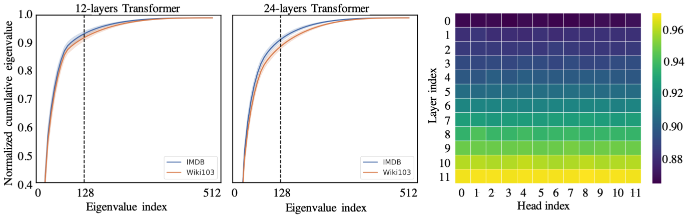
Self-attention is low rank [Wang et al., 2020]



- 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

Compress the KV memory

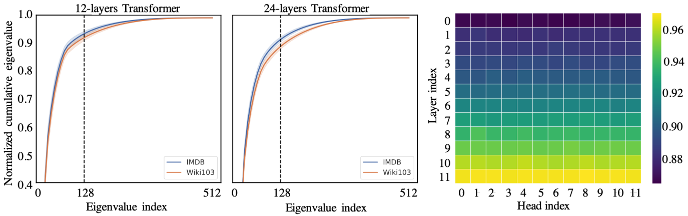
Self-attention is low rank [Wang et al., 2020]



- 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

Compress the KV memory

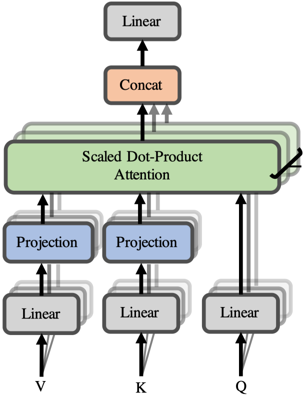
Self-attention is low rank [Wang et al., 2020]



- 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

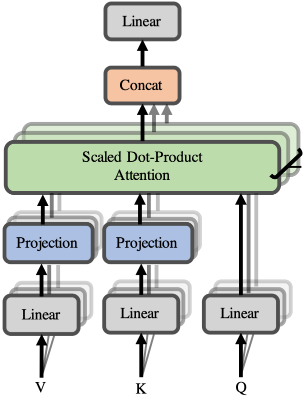
Linformer [Wang et al., 2020]: compute self-attention in a lower dimension



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

Summarize the KV memory

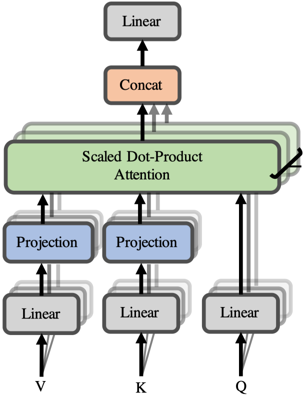
Linformer [Wang et al., 2020]: compute self-attention in a lower dimension



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

Summarize the KV memory

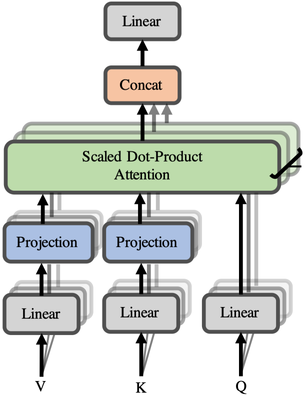
Linformer [Wang et al., 2020]: compute self-attention in a lower dimension



- 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

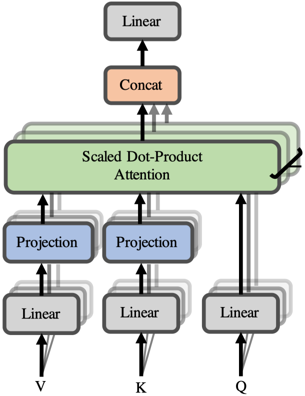
Linformer [Wang et al., 2020]: compute self-attention in a lower dimension



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

Summarize the KV memory

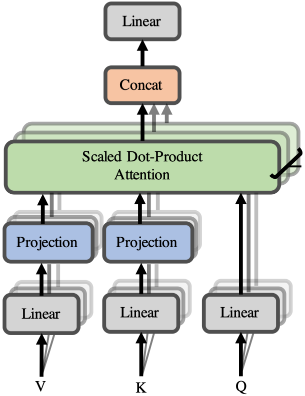
Linformer [Wang et al., 2020]: compute self-attention in a lower dimension



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

Summarize the KV memory

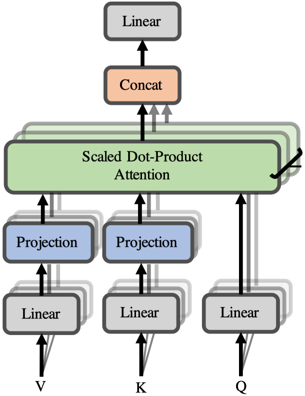
Linformer [Wang et al., 2020]: compute self-attention in a lower dimension



- 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: $O(dnk)$ (linear in n)
- 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: $O(dnk)$ (linear in n)
- Downside of uisng Linformer as a decoder?
  - Unclear how to mask: past and future are mixed

Summary on efficient self-attention

Improve the quadratic time and space complexity of self-attention

- Sparsify the attention matrix
- Compress the KV memory

Summary on efficient self-attention

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

Approach 4: system-level approaches

- Operates at a lower abstraction level
- Often brings more direct impact on efficiency
- Example:
  - Gradient accumulation
  - Model and data parallelism (e.g., deepspeed)
  - Flash attention: exploit GPU memory asymmetry

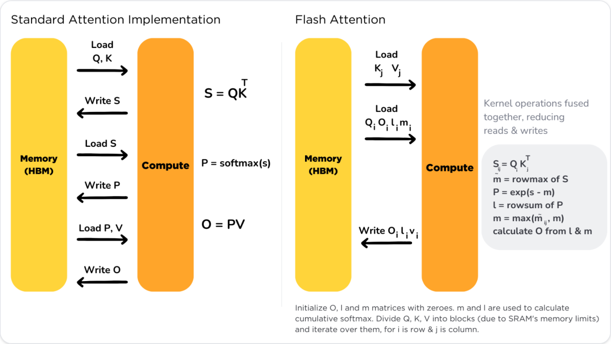


Table of Contents

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.
- Potentially destroy pretrained representations

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.
- Potentially destroy pretrained representations

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

Improve finetuning efficiency

Problem:

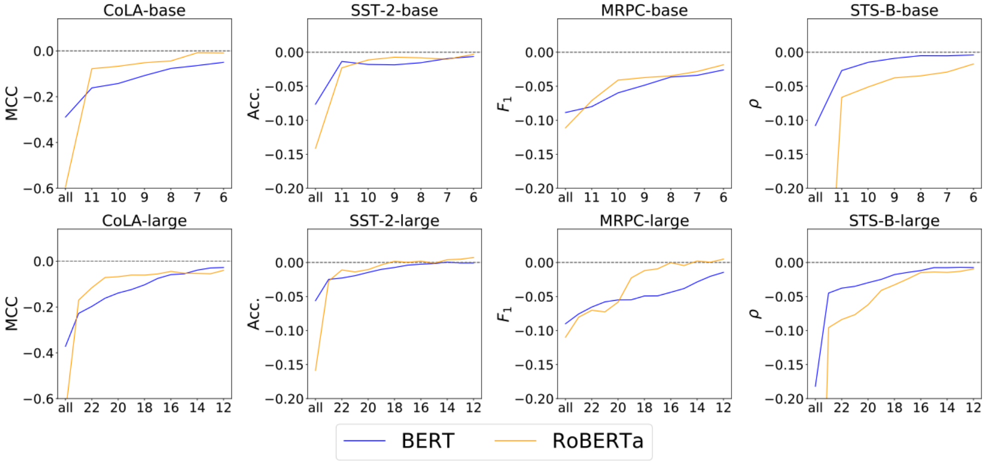
- In NLP, typically all parameters of the pretrained models (e.g., BERT) are finetuned, which is expensive for large models.
- Saving and loading finetuned models for different tasks is costly.
- Potentially destroy pretrained representations

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)

Amplifying / suppressing certain features?

Bias terms in QKV projection

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

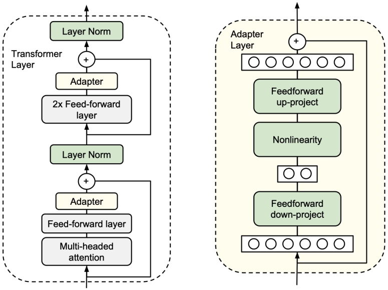
Bias terms in MLP layers

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

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

Adapt the frozen pretrained model

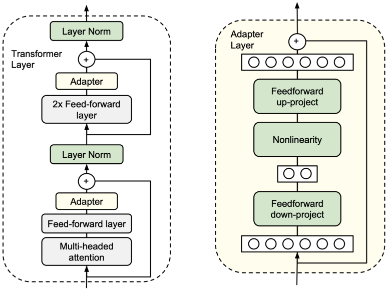
Adapter [Houlsby et al., 2019]: insert small networks to the pretrained model



- 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

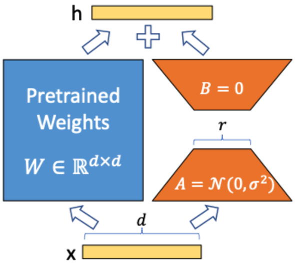


- Insert learnable ”adapters” in-between layers
- Adapters uses a bottleneck structure to reduce parameters
- Adapters uses a skip-connection such that it can be “reduced” to the original frozen model

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

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

- $W_0 \in \mathbb{R}^{d \times k}, B \in \mathbb{R}^{d \times r}, A \in \mathbb{R}^{r \times k} (r \ll k)$
- 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
- System approach: mixed-precision training (e.g., converting some or all params to fp16)

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