

PARAMETER-EFFICIENT FINE-TUNING

2026F INF385T: Post-Training of LLMs | Week 6

CHAN YOUNG PARK

Assistant Professor, iSchool, School of Computing, The University of Texas at Austin

Announcements

- HW 1 is getting released tonight
 - SFT data curation
 - LoRA vs Fine-tuning
- Next week, quick HW1 walkthrough in class
- HW 1 due: Oct 14th (you have 4 late days total!)

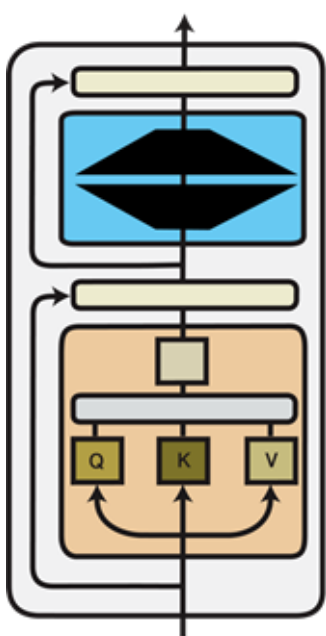
Recap: week 5 (SFT)

- Q1. What objective are we optimizing?
- Q2. What token sequence does the model see?
- Q3. Which tokens contribute to the loss?
- Q4. what engineering choices control the run?
- Q5. how do we know the fine-tuning worked?

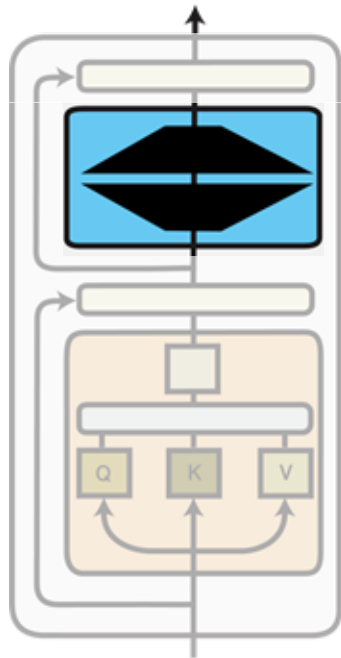
Week 6 Overview

- PEFT
 - Addition
 - Specification
 - Reparameterization
- Lab 6: Shared Task on Emotion Classification
- Many slides are based on materials by
 - Stanford CS224N: https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1254/slides_w25/cs224n-2025-lecture11-adapatation.pdf
 - CMU 11667: <https://cmu.box.com/s/8x5xnusopr6gwatqq9kh3vbbkmnpnd9c>
 - EMNLP 2022 Tutorial on [Modular and Parameter-Efficient Fine-Tuning for NLP Models](#)
 - Blog post: <https://www.leewayhertz.com/parameter-efficient-fine-tuning/>

What is parameter efficient fine-tuning?



Full Fine-tuning
 Update **all** model
 parameters

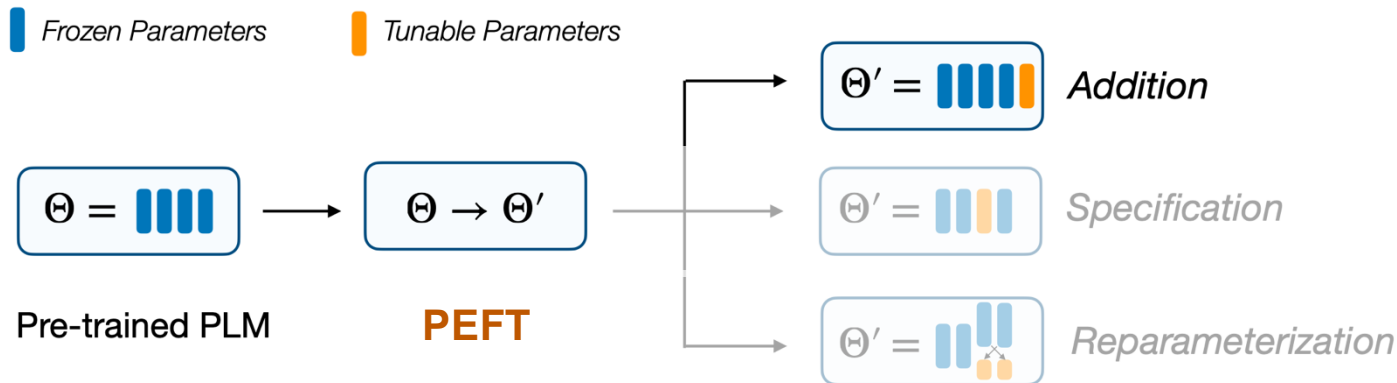


**Parameter-efficient
 Fine-tuning**
 Update a **small subset** of
 model parameters

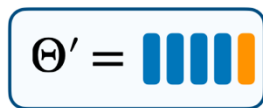
- PEFT: fine-tune only small amounts of weights
- Why only *some* parameters?
 - Full fine-tuning is impractical with large models
 - SOTA models could be over-parameterized (for our tasks)

Three categories of PEFT

- **Addition**: what if we introduce additional trainable parameters and just train those?
- **Specification**: What if we pick a small subset of the parameters and just tune those?
- **Reparameterization**: What if we re-parameterize the model into something that is more efficient to train?

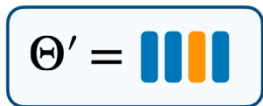


Three categories of PEFT



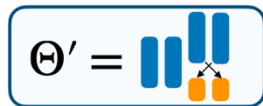
Addition

Prompt tuning, prefix tuning, adapters, ...



Specification

Layer freezing, Pruning, ...



Reparameterization

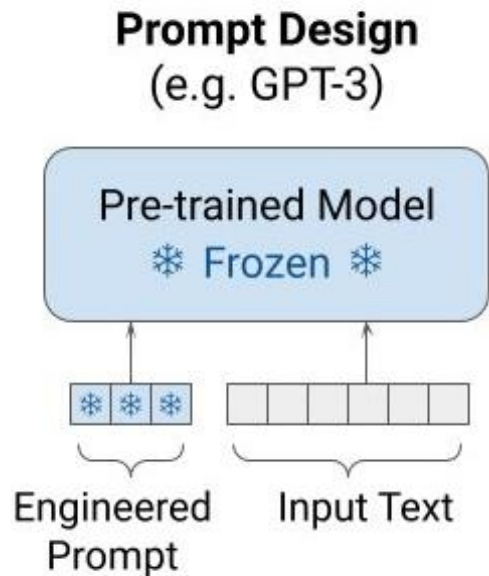
LoRA, QLoRA, ...

what if we introduce additional trainable parameters and just train those?

ADDITION

PROMPT TUNING, PREFIX TUNING, ADAPTERS

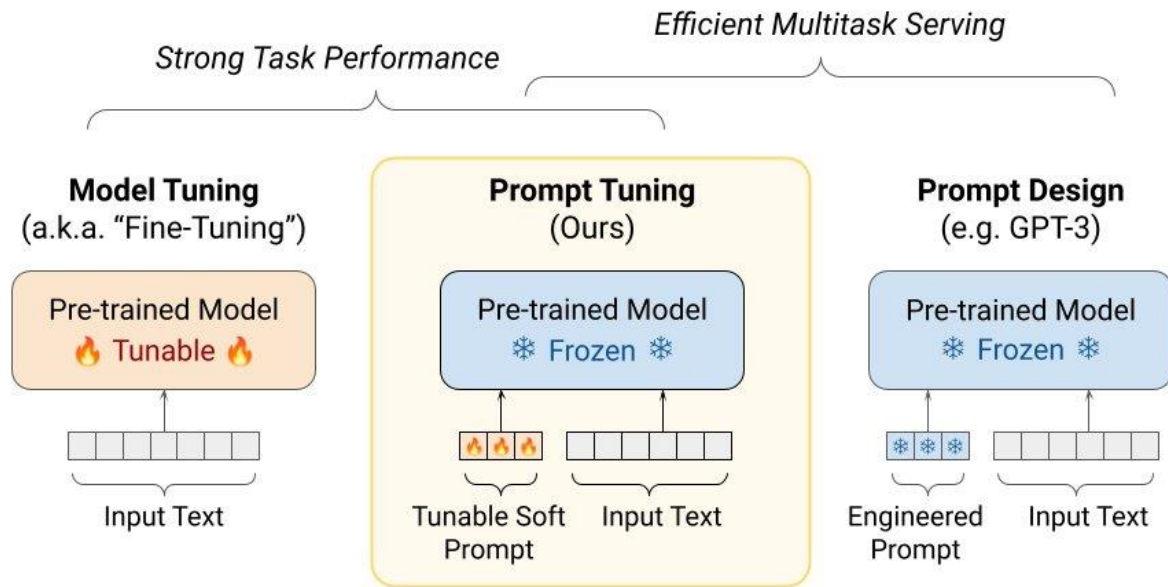
Prompt Tuning (Lester et al., 2021)



- Prompt engineering requires a lot of human decision-making and can be very finicky to get working
- If we have a bunch of examples of the task, why can't we just train a neural network to produce a good prompt for the task?

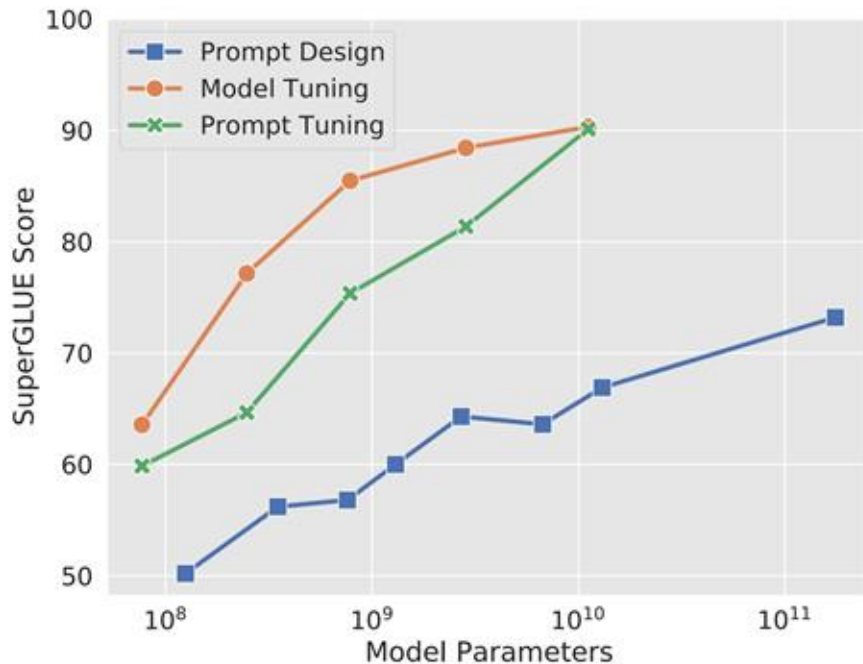
Prompt Tuning (Lester et al., 2021)

- Learning “soft prompts” to condition frozen LMs to perform downstream tasks
- Prepend virtual tokens to input and learn embeddings of these special tokens only
- Freeze → new embedding matrix → train on your data



Prompt tuning only works well at scale

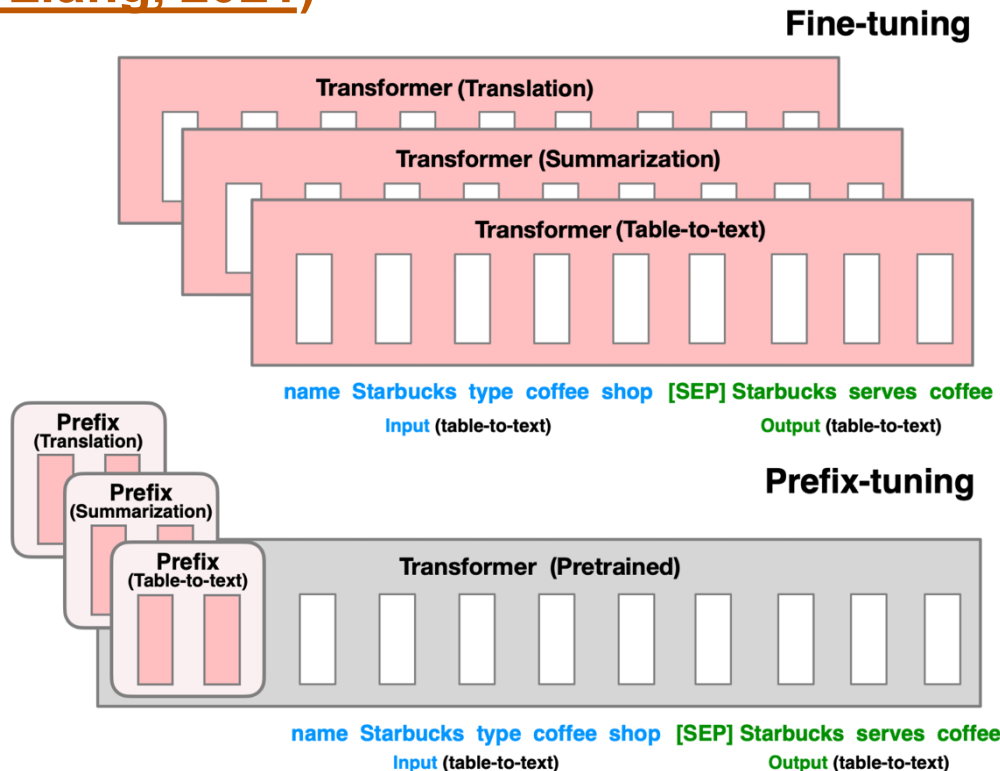
- Standard model tuning achieves strong performances but requires scoring separate copies of model for each end task
- Prompt tuning only matches fine-tuning at the largest model size
- Prompt tuning performs poorly at smaller model sizes and on harder tasks



Prompt tuning vs standard fine-tuning vs prompt engineering across T5 models of different sizes [Lester et al., 2021]

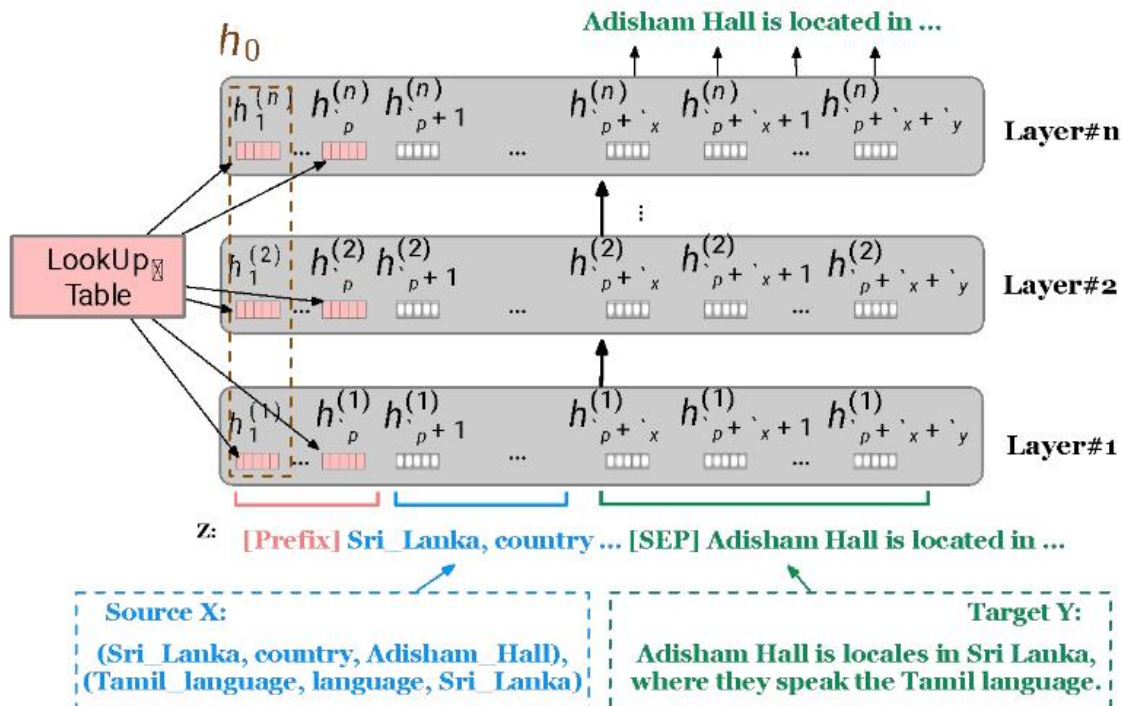
Prefix Tuning (Li and Liang, 2021)

- the prefix is a sequence of continuous task-specific vector
- processed by the model just like real words would be (“virtual tokens”)



Prefix Tuning (Li and Liang, 2021)

- Same idea as prompt tuning
- except that the learned prefix is appended not just to the input embeddings, but rather at each layer of the Transformer
- More parameters, higher expressiveness



Advantages of Prefix/Prompt Tuning

- The learned embeddings tend to be relatively small, just a few megabytes or less.
 - It is cheap to keep around one set of embeddings per task
- The pre-trained LLM can be loaded into memory, and at inference time, the appropriate task-specific embeddings can be swapped in

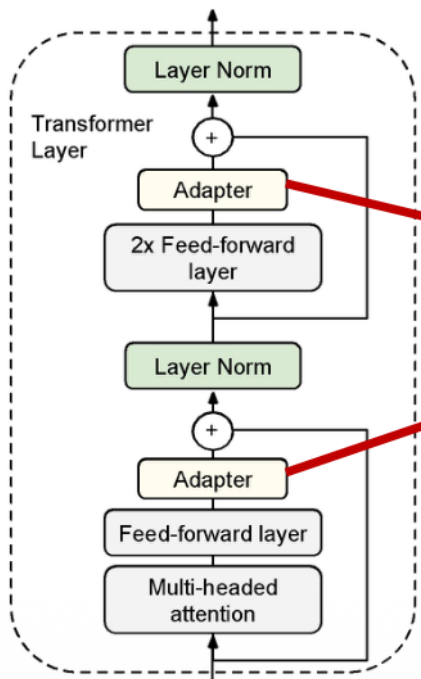
Limitations & Decline

- Limitations
 - Uses up context length and adds attention cost
 - Unstable training (sensitive to prefix length and initialization)
 - Weak performance on smaller models
 - Learned vectors are uninterpretable and non-transferable
- Why it no long is mainstream
 - LoRA performs better and merges into weights with zero inference overhead
 - Multi-LoRA serving (e.g., vLLM) removed its batching advantage
 - Instruction-tuned models follow plain-text prompts well enough

Adapter (Houlsby et al. 2019)

Instead of steering the model through its input, what if we add small trainable modules inside the model itself?

- New NN module that inserts a new function between layers of a pre-trained model to adapt to a downstream task

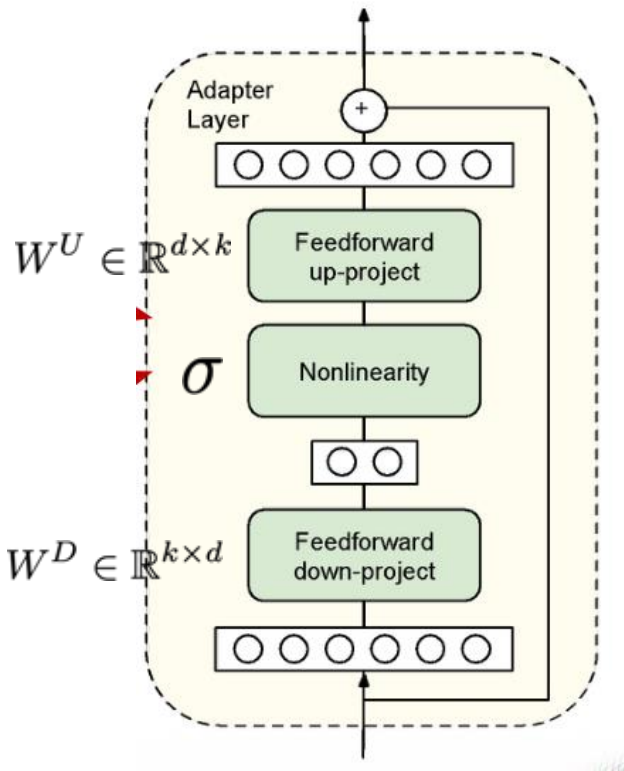


- Design of adapters is model-specific
- The adapter is usually placed after the multi-head attention and after the feed-forward layer

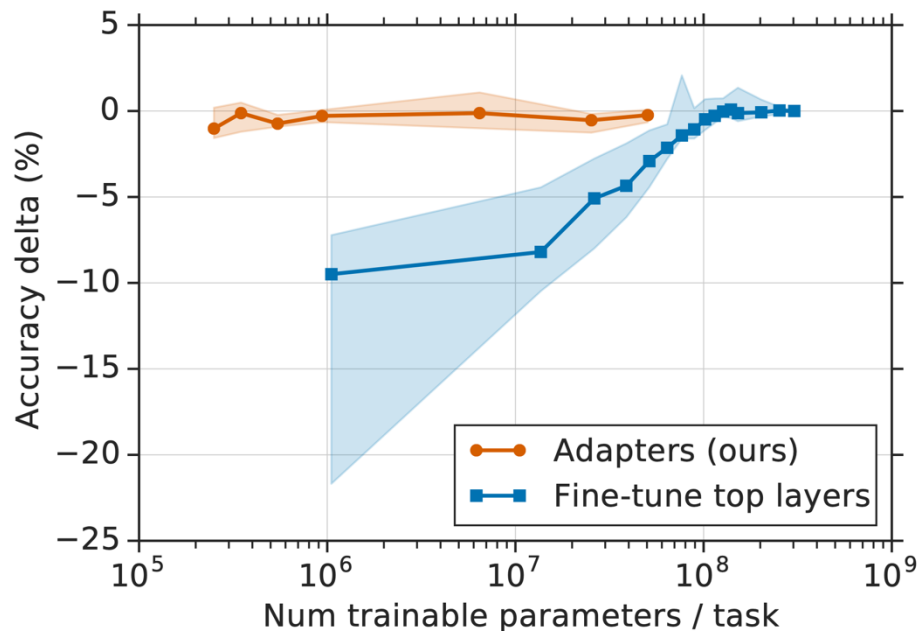
Adapter (Houlsby et al. 2019)

- An adapter in a Transformer layer typically consists of :
 - a FF down-projection
 - a FF up-projection
 - Nonlinearity (typically ReLU)

$$f_{\phi}(\mathbf{x}) = W^U(\sigma(W^D \mathbf{x}))$$



Adapters are stable and parameter-efficient



- 20th, 50th, and 80th performance percentiles across nine tasks from the GLUE benchmark
- attains a similar performance to full finetuning with two orders of magnitude fewer trained parameters

However, adding new layers = making inference slower, model bigger

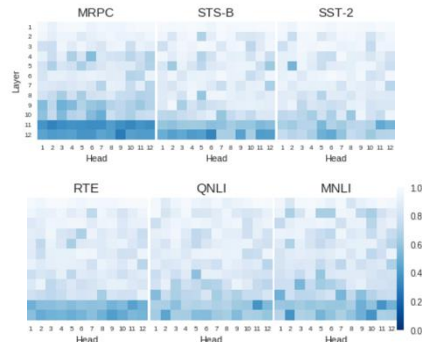
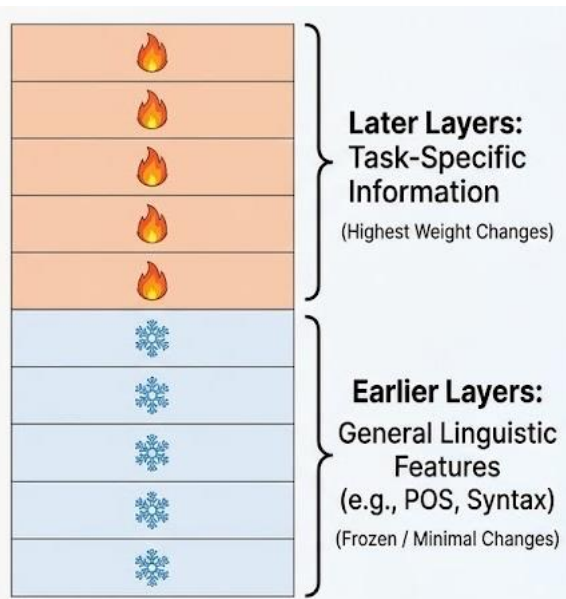
What if we pick a small subset of the parameters and just tune those?

SPECIFICATION

LAYER FREEZING, PRUNING

Layer freezing

- Earlier layers of Transformers capture general linguistic features (e.g., POS, syntax), while later layers encode more task-specific information [1, 2]
- During fine-tuning, most weights barely change; the largest changes occur in the last few layers [3]
- Fine-tuning only the last 25% of layers retains ~90% of full fine-tuning quality, and tuning all layers does not always help [4]



Cosine similarity between pre-trained vs fine-tuned model's self-attention for the selected GLUE tasks
 darker = greater differences

[1] Tenney et al. (2019). [BERT Rediscovered the Classical NLP Pipeline](#). *ACL*.

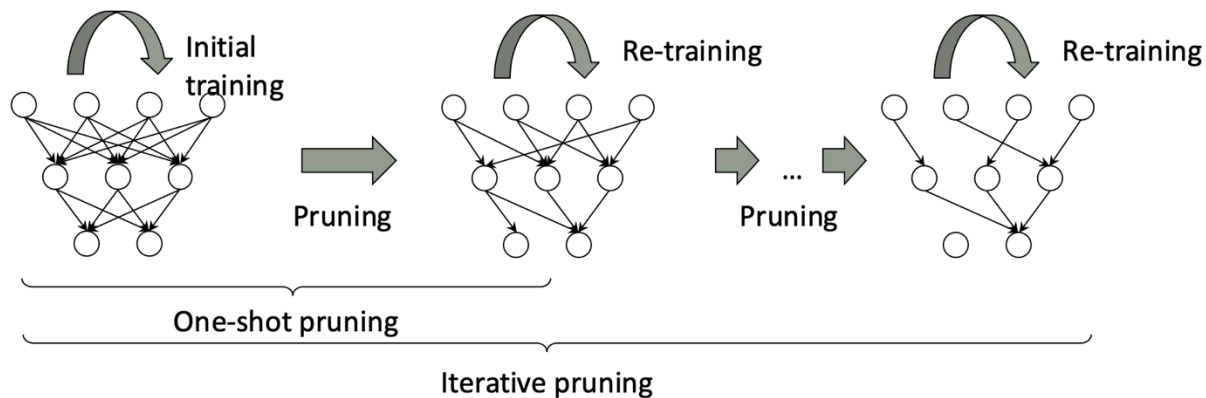
[2] Rogers et al. (2020). [A Primer in BERTology: What We Know About How BERT Works](#). *TACL*.

[3] Kovaleva et al. (2019). [Revealing the Dark Secrets of BERT](#). *EMNLP*.

[4] Lee, Tang & Lin (2019). [What Would Elsa Do? Freezing Layers During Transformer Fine-Tuning](#). *arXiv*.

Pruning

- Remove unimportant connections from a trained network to produce a smaller subnetwork
- Most common criterion: weight magnitude, i.e., remove the lowest-magnitude weights ([Han et al., 2015](#))
- Standard process: train $\rightarrow$ prune $\rightarrow$ re-train the remaining weights to recover performance.
- Iterative pruning (repeating prune + re-train in small steps) works better than one-shot pruning ([Frankle & Carbin, 2019](#))



Pruning as a Binary Mask & Task Vector

- Pruning = applying a binary mask $\mathbf{b} \in \{0, 1\}^{|\theta|}$ that keeps or removes each connection: $f(\theta \circ \mathbf{b})$
- Fine-tuning can be viewed as adding a task-specific vector ϕ
$$f(\theta') = f(\theta + \phi)$$
- Choosing which parameters to update=choosing a pruning mask
- For a sparse final model: $f(\theta') = f(\theta \circ \mathbf{b} + \phi)$

Diff Pruning

- Prior methods chose which parameters to freeze or tune manually. Why not learn it? ([Guo et al, 2021](#))
- Freeze θ and learn a per-parameter delta ϕ : $\theta_{\text{task}} = \theta + \phi$
- Encourage sparsity with an L0-norm penalty: $\mathcal{L}_{\text{task}}(\theta + \phi) + \lambda \|\phi\|_0$
- Prune based on the magnitude of ϕ , not $\theta + \phi$
- Matches full fine-tuning on GLUE while modifying only ~0.5% of parameters per task
- Limitation: training memory is higher than full fine-tuning; savings are in storage only

What if we re-parameterize the model into something that is more efficient to train?

REPARAMETERIZATION

LoRA

LoRA: Low Rank Adaptation

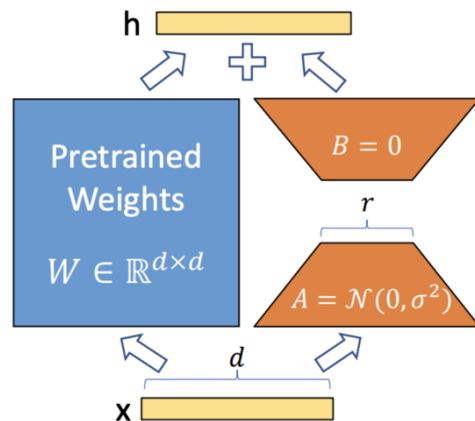
- Pretrained models have a low intrinsic dimension
 - fine-tuning within a small subspace nearly matches full fine-tuning ([Aghajanyan et al., 2021](#))
- Hypothesis: the weight update ΔW also has a low intrinsic rank ([Hu et al., 2021](#))
→ let's encode the task-specific ΔW by a smaller sized set of parameters
- Fine-tuning can be viewed as learning an update: $W = W_0 + \Delta W$
 - Diff pruning: make ΔW **sparse**
 - LoRA: make ΔW **low-rank**

LoRA: Method

- Freeze W_0 and decompose the update into two small (low-rank) matrices:

$$W_0 + \Delta W = W_0 + \alpha B A$$

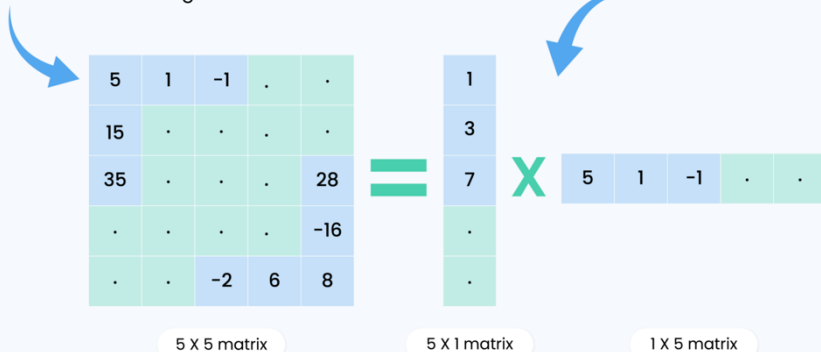
, where $B \in \mathbb{R}^{d \times r}$, $A \in \mathbb{R}^{r \times k}$, $r \ll \min(d, k)$



Low-rank Matrix Decomposition

Full fine-tuned weight matrix

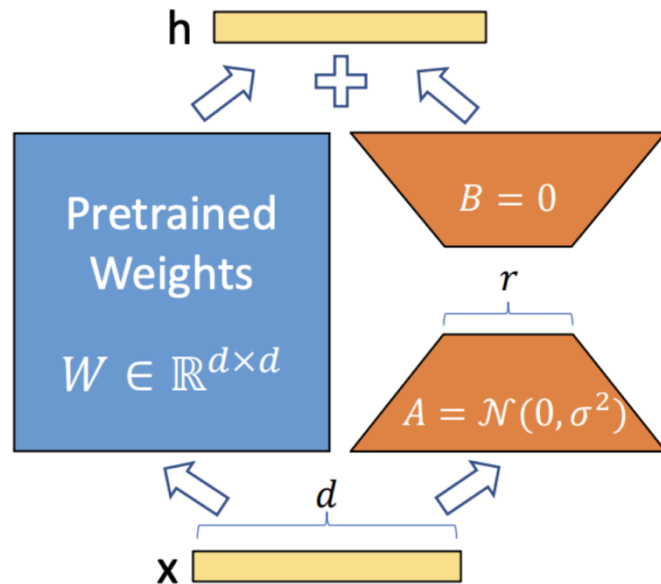
Low-rank matrices



LoRA: Method

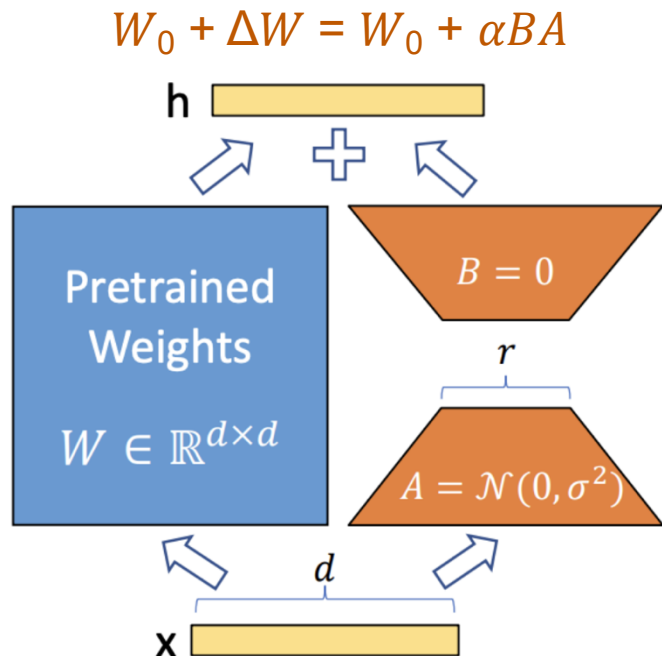
$$W_0 + \Delta W = W_0 + \alpha B A$$

- α is the tradeoff between pre-trained “knowledge” and task-specific “knowledge”
- Initialize A randomly and $B=0$ so $\Delta W = 0$ at the start of training
- Typically applied to attention weights (e.g., W_q, W_v)
- Only A and B contain trainable parameters



LoRA: low rank adaptation (Hu et al., 2021)

- Reduces trainable parameters by $\sim 10,000\times$ and GPU memory by $\sim 3\times$ on GPT-3 175B, with performance on par with full fine-tuning
- vs. Adapters: no added latency
vs. Prefix tuning: no context usage
vs. Diff pruning: lower training memory
- Easy task switching: swap BA while sharing one base model
- Extensions: QLoRA enables fine-tuning 4-bit quantized models on a single GPU
- Limitation: may learn less than full fine-tuning on harder domains, but also forgets less ([Biderman et al., 2024](#))



LoRA with Hugging Face PEFT

```
from transformers import AutoModelForCausalLM
from peft import LoraConfig, get_peft_model

model = AutoModelForCausalLM.from_pretrained("Qwen/Qwen2.5-0.5B")

config = LoraConfig(
    r=8,                                # rank r
    lora_alpha=16,                       # scaling factor ( $\Delta W$  scaled by  $\alpha/r$ )
    target_modules=["q_proj", "v_proj"], # which W to adapt ( $W_q$ ,  $W_v$ )
    lora_dropout=0.05,
    task_type="CAUSAL_LM",
)

model = get_peft_model(model, config)    # freezes  $W_0$ , adds A and B
model.print_trainable_parameters()      # ~0.1% of params are trainable
```

Train, Save, Merge

```
# Train as usual (e.g., transformers Trainer or a custom loop)
trainer.train()
```

```
# Save only the LoRA weights (a few MB, not the full model)
model.save_pretrained("my-lora-adapter")
```

```
# Load later on top of the same base model
from peft import PeftModel
model = PeftModel.from_pretrained(base_model, "my-lora-adapter")
```

```
# Merge BA into W_0 for zero-latency inference
model = model.merge_and_unload()
```

LAB 6

Lab 6: Shared task on emotion classification

- Fine-tune Qwen2.5-0.5B-Instruct with LoRA to label Reddit comments with one or more of 28 emotions, then push performance as high as you can with your partner
- **Plan**
 - **On your own:** run Steps 0 and 1 (zero-shot baseline → basic SFT baseline).
 - **With your partner (~10 min):** decide what to change and who runs which experiments.
 - **On your own (~20 min):** run your experiments in Step 2 and find your best setting.
 - **Together (~15 min):** combine what worked, train the final model, and submit test predictions (Step 3).
 - **Form submission (~5min):** fill out and submit the microsoft form
 - **Wrap-up & Presentations (~10 min):** The top-performing team in each eval metric (3 teams total) will briefly present their approach and **earn 1 late day credit!**