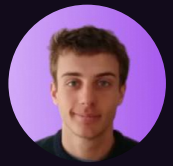
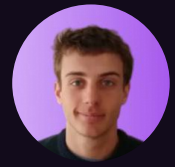
Pruna AI

Fine-tuning for Language Models



Bertrand Charpentier
Founder, President & Chief Scientist

Overview – Fine-tuning for Language Models



- What is Fine-tuning?
- Fine-tuning All parameters
- Fine-tuning New parameters
  - Which layer type/depth to attach LoRA?
- Fine-tuning Selected parameters
  - Which existing layer type/depth to finetune?
- Fine-tuning Quantized models
- State-of-the-art Fine-tuning Methods
  - SVFT, DoRA, Lora, Qlora, LQLoRA, LoftQ

Reference:

- https://docs.nvidia.com/deeplearning/performance/dl-performance-gpu-background/index.html#understand-perf

What is Fine-tuning?

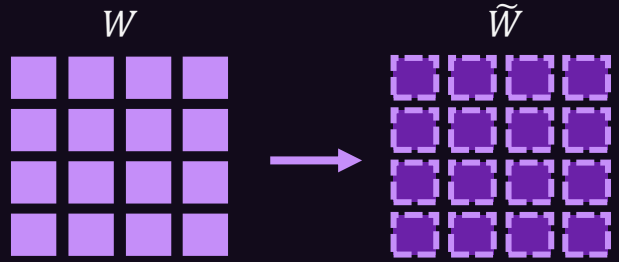
Fine-tuning the process of continuing training a model with pretrained parameters W on a selected dataset D for a selected task.

Input - A model with a set of pretrained parameters W.

Output - A fine-tuned model with a new set of pretrained parameters $\tilde{W}$ with improved performance for a selected task on the dataset $\boldsymbol{D}$.

Algorithm

- Many options!



Benefits

- It can **specialize a model** on for a specific task.
- It can **recover model performance** after aggressive compression.

Remarks

- The fine-tuned model can have a (slightly) different structure (e.g. additional components) compared to the base model.
- Given a (potentially large) pretrained model, it is more efficient to fine-tune or distill a (potentially small) model.

Reference:
- https://arxiv.org/abs/2502.08606

Fine-tuning All Parameters

Input - A model with a set of pretrained parameters W.

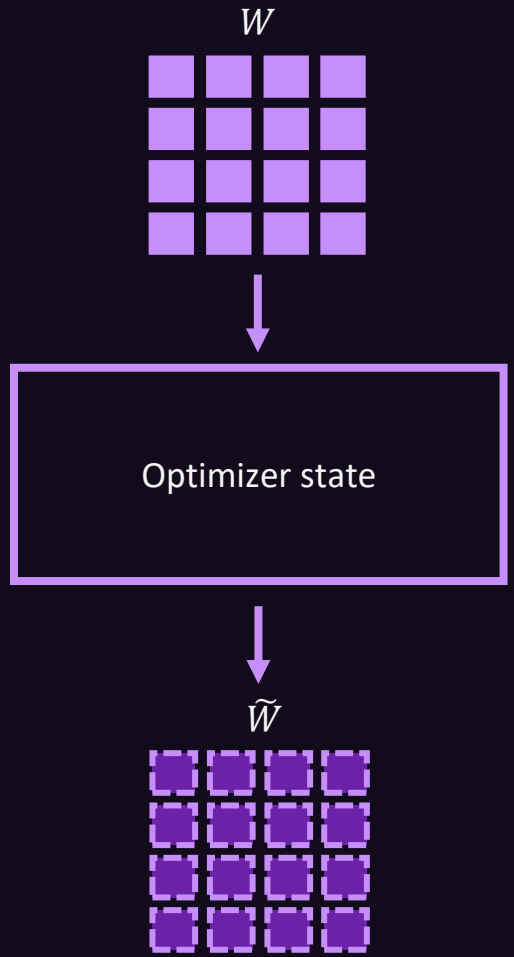
Output - A fine-tuned model with an updated set of parameters $\widetilde{W}$ with improved performance for a selected task on the dataset $\boldsymbol{D}$.

Algorithm

- Make all pretrained parameters W trainable.
- Iteratively optimize all trainable parameters W into $\widetilde{W}$ (e.g. with gradient descent) on a selected task on the dataset $\boldsymbol{D}$.

Remarks

- Full fine-tuning can lead to the best performance for the fine-tuned model.
- Full fine-tuning has a **slow training speed** and **large memory requirement**. It requires to compute and store gradients to update all the weights.
- Each fine-tuned model has a completely new set of fine-tuned parameters, thus requiring a lot of storage and data movement when switching between fine-tuned models for multiple tasks.



Reference:
- https://arxiv.org/pdf/2403.14608

Fine-tuning New Parameters

Input - A model with a set of pretrained parameters W.

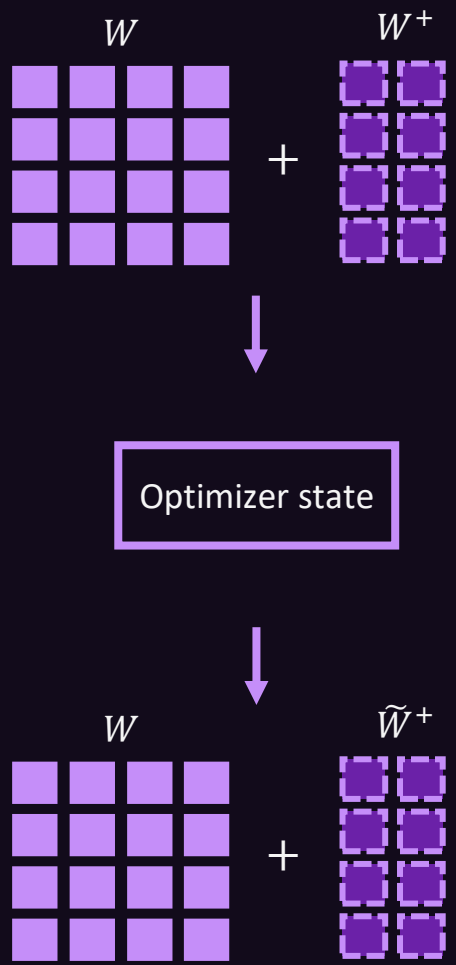
Output - A fine-tuned model with a new set of parameters $\tilde{W}$ with improved performance for a selected task on the dataset $\boldsymbol{D}$.

Algorithm

- Add adapters with new trainable parameters W^+ to the frozen pretrained parameters W.
- Iteratively optimize new trainable parameters W^+ into $\tilde{W}^+$ (e.g. with gradient descent) selected task on the dataset $\boldsymbol{D}$.

Remarks

- Fine-tuning adapters has a **fast training speed** and **small memory requirement**. It requires to compute and store gradients to update only adapters weights.
- Fine-tuned model have only adapters $\tilde{W}^+$ as different parameters, thus enabling easy switch between fine-tuned models for different tasks at low storage costs.



Reference:

- https://arxiv.org/pdf/2403.14608

Fine-tuning New Parameters

Option 1: Apply linear adapters in the form of: $\boldsymbol{WX} \rightarrow \boldsymbol{WX} + \boldsymbol{\alpha}\, \boldsymbol{W}^{+}_{\boldsymbol{down}} \boldsymbol{W}^{+}_{\boldsymbol{up}} \boldsymbol{X}$

- Key examples are **LoRA, DoRA, QLoRA.**
- Adapters can be merged/fused in $\boldsymbol{W} + \boldsymbol{W}^{+}_{\boldsymbol{down}} \boldsymbol{W}^{+}_{\boldsymbol{up}}$ for no inference compute overhead.

Option 2: Apply non-linear adapters in the form of: $f(\boldsymbol{X}) \rightarrow f(\boldsymbol{X}) + \boldsymbol{W}^{+}_{\boldsymbol{down}} \sigma(\boldsymbol{W}^{+}_{\boldsymbol{up}} \boldsymbol{X})$

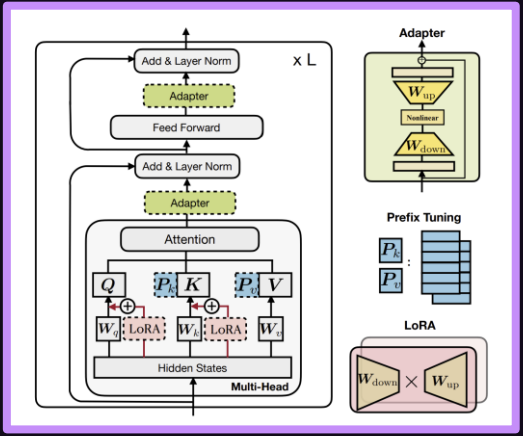
- Key examples are **serial adapters** and **parallel adapters**.
- Adapters create a compute overhead. They cannot be merged/fused.

Option 3: Apply token adapters in the form of: $\boldsymbol{X} \rightarrow \boldsymbol{concat}(\boldsymbol{P}, \boldsymbol{X})$

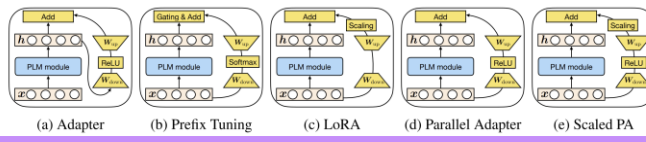
- Key examples are **prefix-tuning, prompt-tuning**, **p-tuning**.

Remarks

- All these methods can be viewed as adding an update Δ in a sequential/parallel fashion to a specific module. The update can generally be seen as adding down/up projections W_{down}, W_{up} with or without non-linearity.
- While applying adapters to all modules achieve better performance, some adapters can be partially or completely pruned.



Method	Δh functional form	insertion form	modified representation	composition function
	Existing Methods			
Prefix Tuning	$\text{softmax}(xW_q P_k^\top) P_v$	parallel	head attn	$h \leftarrow (1-\lambda)h + \lambda \Delta h$
Adapter	$\text{ReLU}(hW_{\text{down}})W_{\text{up}}$	sequential	ffn/attn	$h \leftarrow h + \Delta h$
LoRA	$xW_{\text{down}}W_{\text{up}}$	parallel	attn key/val	$h \leftarrow h + s \cdot \Delta h$



	# of Trainable Parameters = 18M						
Weight Type	W_q	W_k	W_v	W_o	W_q, W_k	W_q, W_v	W_q, W_k, W_v, W_o
Rank r	8	8	8	8	4	4	2
WikiSQL (±0.5%)	70.4	70.0	73.0	73.2	71.4	**73.7**	**73.7**
MultiNLI (±0.1%)	91.0	90.8	91.0	91.3	91.3	91.3	**91.7**

Table 5: Validation accuracy on WikiSQL and MultiNLI after applying LoRA to different types of attention weights in GPT-3, given the same number of trainable parameters. Adapting both W_q and W_v gives the best performance overall. We find the standard deviation across random seeds to be consistent for a given dataset, which we report in the first column.

Reference:

- https://arxiv.org/pdf/2110.04366
- https://arxiv.org/pdf/2403.14608
- https://arxiv.org/pdf/2106.09685

Fine-tuning Selected Parameters

Input - A model with a set of pretrained parameters W.

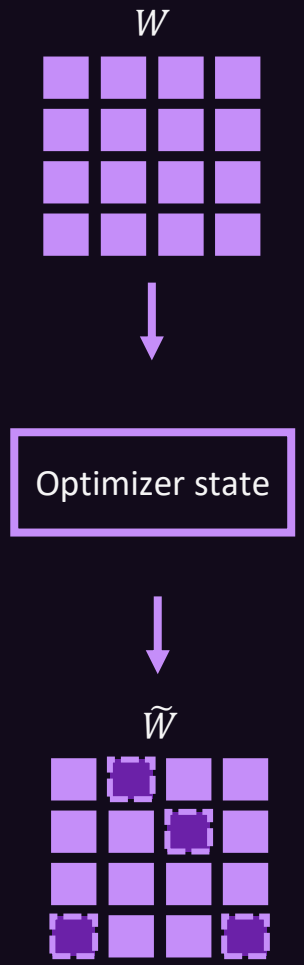
Output - A fine-tuned model with a new set of pretrained parameters $\widetilde{W}$ with improved performance for a selected task on the dataset $\boldsymbol{D}$.

Algorithm

- Select parameters $w_1, .., w_k$ in W and make them trainable. It can be individual weights, channels, or layers.
- Iteratively optimize selected trainable parameters (e.g. with gradient descent) selected task on the dataset $\boldsymbol{D}$.

Remarks

- Fine-tuning selected parameters has a **fast training speed** and **small memory requirement**. It requires to compute and store gradients to update a small set of weights.
- Selected fine-tuned parameters do not require any additional space after training.



Reference:

- https://hanlab.mit.edu/courses/2024-fall-65940

Fine-tuning Selected Parameters

Option 1: Fine-tuning selected layers

- A key example are **Perp, BitFit**.
- Common layers to fine-tuned are bias, batch and layer norms, head/embedding layers.

Option 2: Fine-tuning selected weights/channels

- Key examples are **Diff pruning**, **PaFi**, **FishMask/Dip**.
- The fine-tuned weights/channels selection can be based on smallest magnitude, largest fisher information, second order information, or simply learned.

Remarks

- Selecting structured parameters (e.g. layers, channels) benefits better hardware efficiency compared to selecting unstructured parameters.

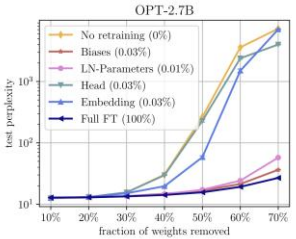
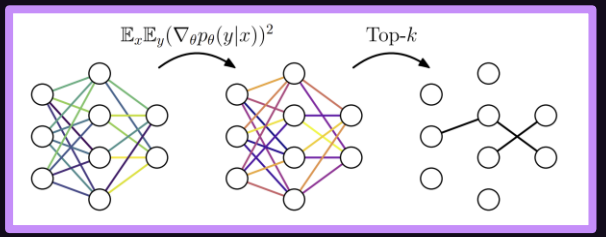
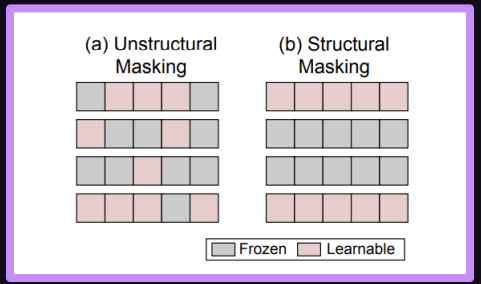


Figure 1. OPT-2.7B evaluated on WikiText: Final perplexity vs. sparsity after pruning, followed by retraining only the specified parameter subset. We indicate the percentage of trainable parameters in parentheses. Full FT refers to full retraining of all parameters.





Reference:

- https://arxiv.org/abs/2312.15230
- https://arxiv.org/pdf/2012.07463

Fine-tuning Quantized Models

Option 1: Full fine-tuning

- Case A: Low precision training
- Case B: Dequantizing to train high precision and then requantize.

Option 2: New parameter fine-tuning

- Initialization need to start from the pretrained model state → While $\boldsymbol{WX} + \boldsymbol{\alpha}\, \boldsymbol{W}^{+}_{\boldsymbol{down}} \boldsymbol{W}^{+}_{\boldsymbol{up}} \boldsymbol{X} = \boldsymbol{WX}$ if $\boldsymbol{W}^{+}_{\boldsymbol{down}} = \boldsymbol{0}$, $\mathrm{quant}(\boldsymbol{W})\boldsymbol{X} + \boldsymbol{\alpha}\, \boldsymbol{W}^{+}_{\boldsymbol{down}} \boldsymbol{W}^{+}_{\boldsymbol{up}} \boldsymbol{X} \neq \boldsymbol{WX}$ if $\boldsymbol{W}^{+}_{\boldsymbol{down}} = \boldsymbol{0}$.
- If adapters are trained in full precision, they are not ready to be quantized and merged with the quantized weights. If adapter are trained in low precision, it adds fine-tuning stability complexity.
- Except if fused kernels exist for $\mathrm{quant}(\boldsymbol{W})\boldsymbol{X} + \boldsymbol{\alpha}\, \boldsymbol{W}^{+}_{\boldsymbol{down}} \boldsymbol{W}^{+}_{\boldsymbol{up}} \boldsymbol{X}$, the fine-tuned quantized model will be slower than the quantized model.
- Key examples are QLoRA, LQLoRA, QA LoRA, LoftQ.

Option 3: Selected parameter fine-tuning

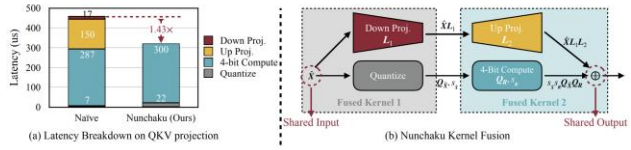
- Case A: Low precision training
- Case B: Dequantizing to train high precision and then requantize.

Remarks

- While quantized models create challenges to fine-tune, this is not always the case for other compression methods (e.g. pruning, dilation, …).

$$\arg\min_{\mathbf{Q},\mathbf{L}_1,\mathbf{L}_2} \|\mathbf{W} - (\mathbf{Q} + \mathbf{L}_1\mathbf{L}_2)\|_F, \quad \text{where } \mathbf{Q} \in \mathbb{Q}_b^{d\times k}, \mathbf{L}_1 \in \mathbb{R}^{d\times r}, \mathbf{L}_2 \in \mathbb{R}^{r\times k}.$$

$$\begin{aligned} \mathbf{L}_1^{(t)}, \mathbf{L}_2^{(t)} &\leftarrow \mathrm{SVD}(\mathbf{W} - \mathbf{Q}^{(t-1)}, r), & &= \arg\min_{\mathrm{rank}(\mathbf{L})\le r} \|\mathbf{W} - (\mathbf{Q}^{(t-1)} + \mathbf{L})\|_F, \\ \mathbf{Q}^{(t)} &\leftarrow \mathrm{Quantize}(\mathbf{W} - \mathbf{L}_1^{(t)}\mathbf{L}_2^{(t)}), & &\approx \arg\min_{\mathbf{Q}\in\mathbb{Q}_b^{d\times k}} \|\mathbf{W} - (\mathbf{Q} + \mathbf{L}_1^{(t)}\mathbf{L}_2^{(t)})\|_F, \end{aligned}$$



(a) Latency Breakdown on QKV projection (b) Nunchaku Kernel Fusion

Figure 6: (a) Naïvely running low-rank branch with rank 32 will introduce 57% latency overhead due to extra read of 16-bit inputs in *Down Projection* and extra write of 16-bit outputs in *Up Projection*. Our Nunchaku engine optimizes this overhead with kernel fusion. (b) *Down Projection* and *Quantize* kernels use the same input, while *Up Projection* and *4-Bit Compute* kernels share the same output. To reduce data movement overhead, we fuse the first two and the latter two kernels together.

Reference:

- https://arxiv.org/pdf/2311.12023
- https://arxiv.org/pdf/2411.05007

State-of-the-art Fine-tuning

	Fine-tuning All Parameters	Fine-tuning New Parameters			Fine-tuning Selected Parameters		Quantized Model Compatibilty
		Non-linear Adapter	Low rank	Prefix tuning	Structured	Unstructured	
Serial Adapter	❌	✅	❌	❌	❌	❌	❌
MAM Adapter	❌	✅	✅	✅	❌	❌	❌
LoRA	❌	❌	✅	❌	❌	❌	❌
QLoRA	❌	❌	✅	❌	❌	❌	✅
LQLoRA	❌	❌	✅	❌	❌	❌	✅
DoRA	❌	❌	✅	❌	❌	❌	❌
PERP	❌	❌	✅	❌	✅	❌	⚠️
BitFit	❌	❌	❌	❌	✅	❌	⚠️
PaFi	❌	❌	❌	❌	❌	✅	⚠️
DiffPruning	❌	❌	❌	❌	✅	✅	⚠️
Prompt-tuning	❌	❌	❌	✅	❌	❌	⚠️
P-tuning	❌	❌	❌	✅	❌	❌	⚠️
Prefix-tuning	❌	❌	❌	✅	❌	❌	⚠️