LoRA
(Low-Rank Adaptation)

Arpita Rattan & Adrien Dubois
Oct. 11th, 2024

LoRA: Low-Rank Adaptation of Large Language Models

Edward Hu* **Yelong Shen*** **Phillip Wallis** **Zeyuan Allen-Zhu**
Yuanzhi Li **Shean Wang** **Lu Wang** **Weizhu Chen**
Microsoft Corporation
{edwardhu, yeshe, phwallis, zeyuana,
yuanzhil, swang, luw, wzchen}@microsoft.com
yuanzhil@andrew.cmu.edu
(Version 2)

Abstract

An important paradigm of natural language processing consists of large-scale pre-training on general domain data and adaptation to particular tasks or domains. As we pre-train larger models, full fine-tuning, which retrains all model parameters, becomes less feasible. Using GPT-3 175B as an example – deploying independent instances of fine-tuned models, each with 175B parameters, is prohibitively expensive. We propose **Lo**w-**R**ank **A**daptation, or LoRA, which freezes the pre-trained model weights and injects trainable rank decomposition matrices into each layer of the Transformer architecture, greatly reducing the number of trainable parameters for downstream tasks. Compared to GPT-3 175B fine-tuned with Adam, LoRA can reduce the number of trainable parameters by 10,000 times and the GPU memory requirement by 3 times. LoRA performs on-par or better than fine-tuning in model quality on RoBERTa, DeBERTa, GPT-2, and GPT-3, despite having fewer trainable parameters, a higher training throughput, and, unlike adapters, ***no additional inference latency***. We also provide an empirical investigation into rank-deficiency in language model adaptation, which sheds light on the efficacy of LoRA. We release a package that facilitates the integration of LoRA with PyTorch models and provide our implementations and model checkpoints for RoBERTa, DeBERTa, and GPT-2 at https://github.com/microsoft/LoRA.

1 Introduction

Many applications in natural language processing rely on adapting *one* large-scale, pre-trained language model to *multiple* downstream applications. Such adaptation is usually done via *fine-tuning*, which updates all the parameters of the pre-trained model. The major downside of fine-tuning is that the new model contains as many parameters as in the original model. As larger models are trained every few months, this changes from a mere "inconvenience" for GPT-2 (Radford et al., b) or RoBERTa large (Liu et al., 2019) to a critical deployment challenge for GPT-3 (Brown et al., 2020) with 175 billion trainable parameters.[1]

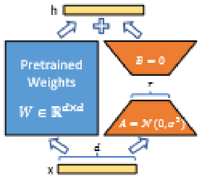


Figure 1: Our reparametrization. We only train A and B.

Many sought to mitigate this by adapting only some parameters or learning external modules for new tasks. This way, we only need to store and load a small number of task-specific parameters in addition to the pre-trained model for each task, greatly boosting the operational efficiency when deployed. However, existing techniques

*Equal contribution.

[0]Compared to V1, this draft includes better baselines, experiments on GLUE, and more on adapter latency.

[1]While GPT-3 175B achieves non-trivial performance with few-shot learning, fine-tuning boosts its performance significantly as shown in Appendix A.

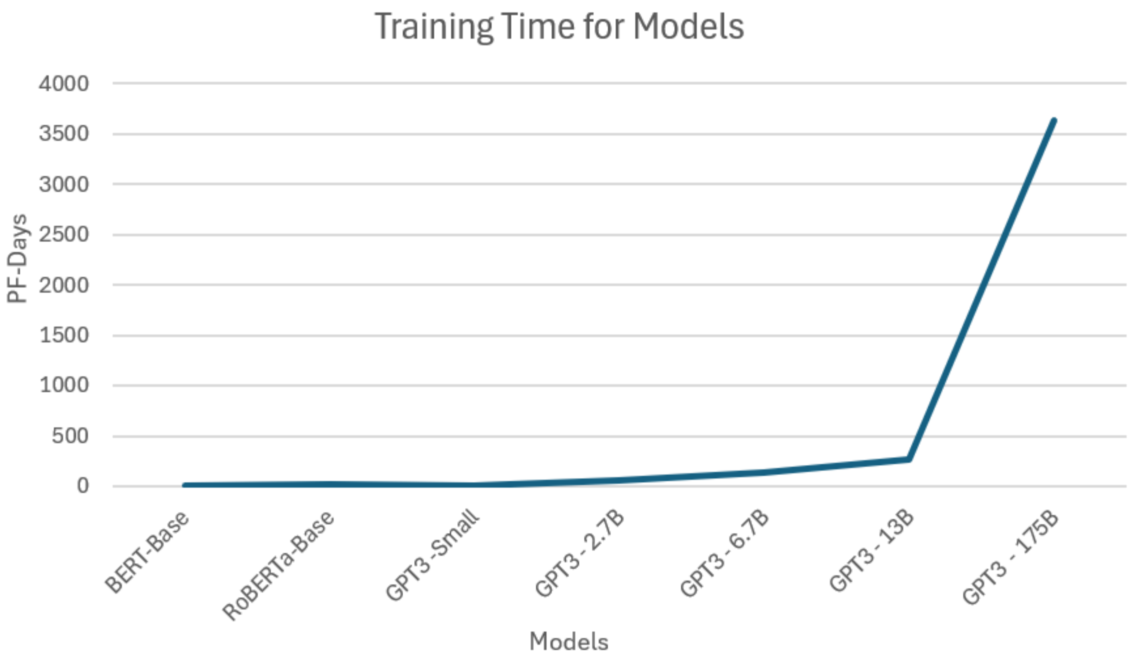
LoRA : Low-Rank Adaptation of Large Language Models

Hu , Shen, et. al.

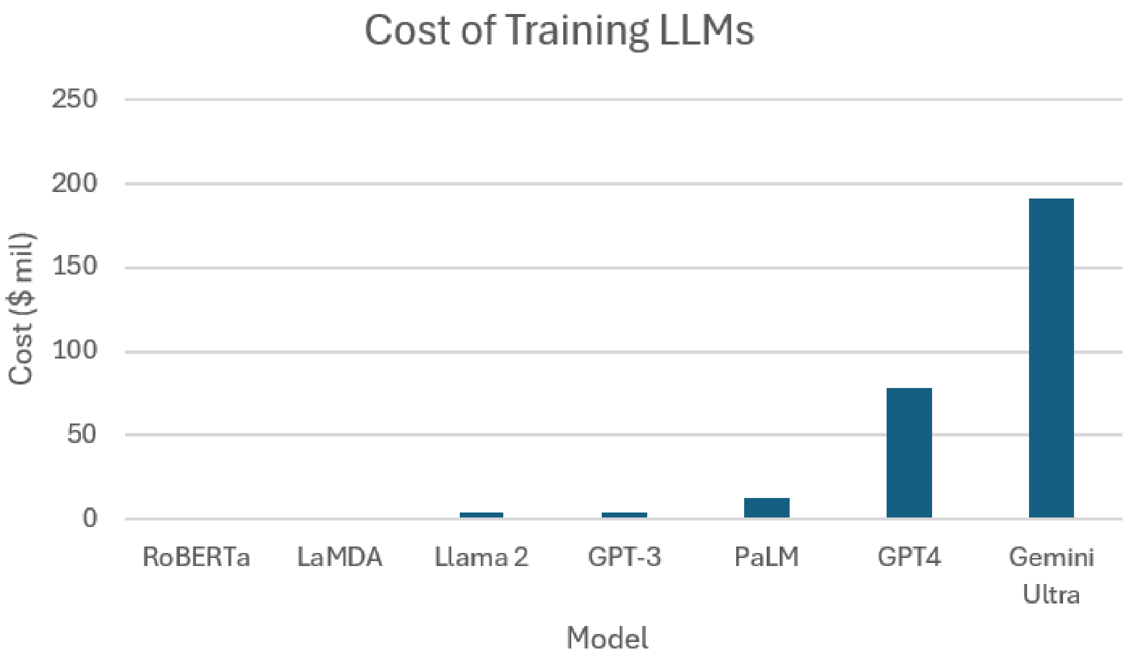
Microsoft Corporation

ICLR, 2022

Motivation



Training Time for Models
4000
3500
3000
2500
2000
1500
1000
500
0
PF-Days
BERT-Base
RoBERTa-Base
GPT3 -Small
GPT3 - 2.7B
GPT3 - 6.7B
GPT3 - 13B
GPT3 - 175B
Models



Cost of Training LLMs
250
200
150
100
50
0
Cost ($ mil)
RoBERTa
LaMDA
Llama 2
GPT-3
PaLM
GPT4
Gemini Ultra
Model

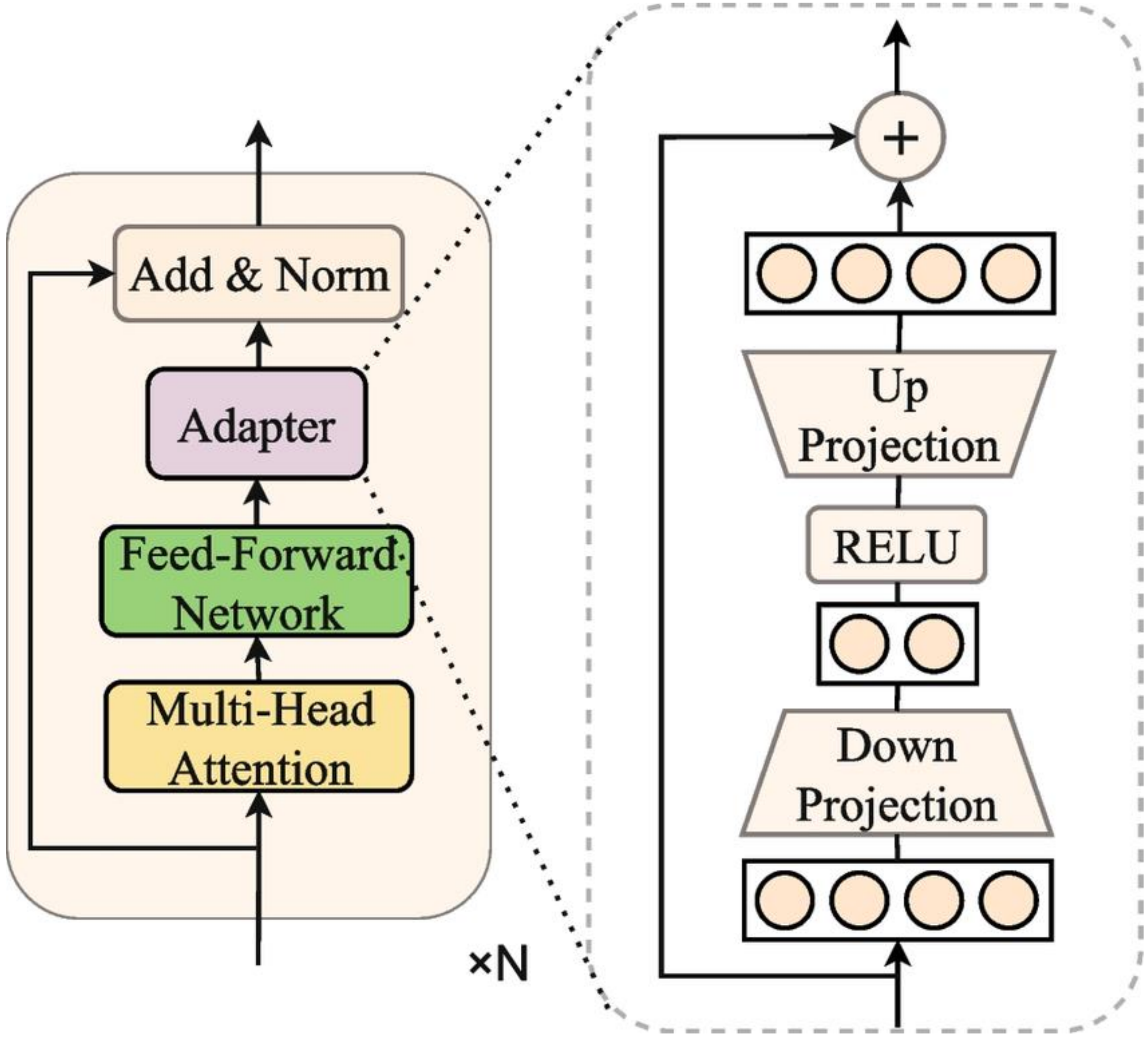
Motivation

- Most companies use **one** model for **many** different downstream tasks
- Retraining models like GPT2/BERT was still regarded as feasible, with the rise of GPT4 this has become nearly impossible (~1.76 trillion parameters).

Finetuning methods

Adapter Layers

- Inserted between network layers to learn task-specific adjustments, while keeping original weights mostly unchanged.
- Clean and modular to implement
- Adding more depth to original model creates greater latency, no way to get around extra computation required
- LLMs need to be parallelized on hardware to keep latency low, adapter layers need to be processed sequentially.

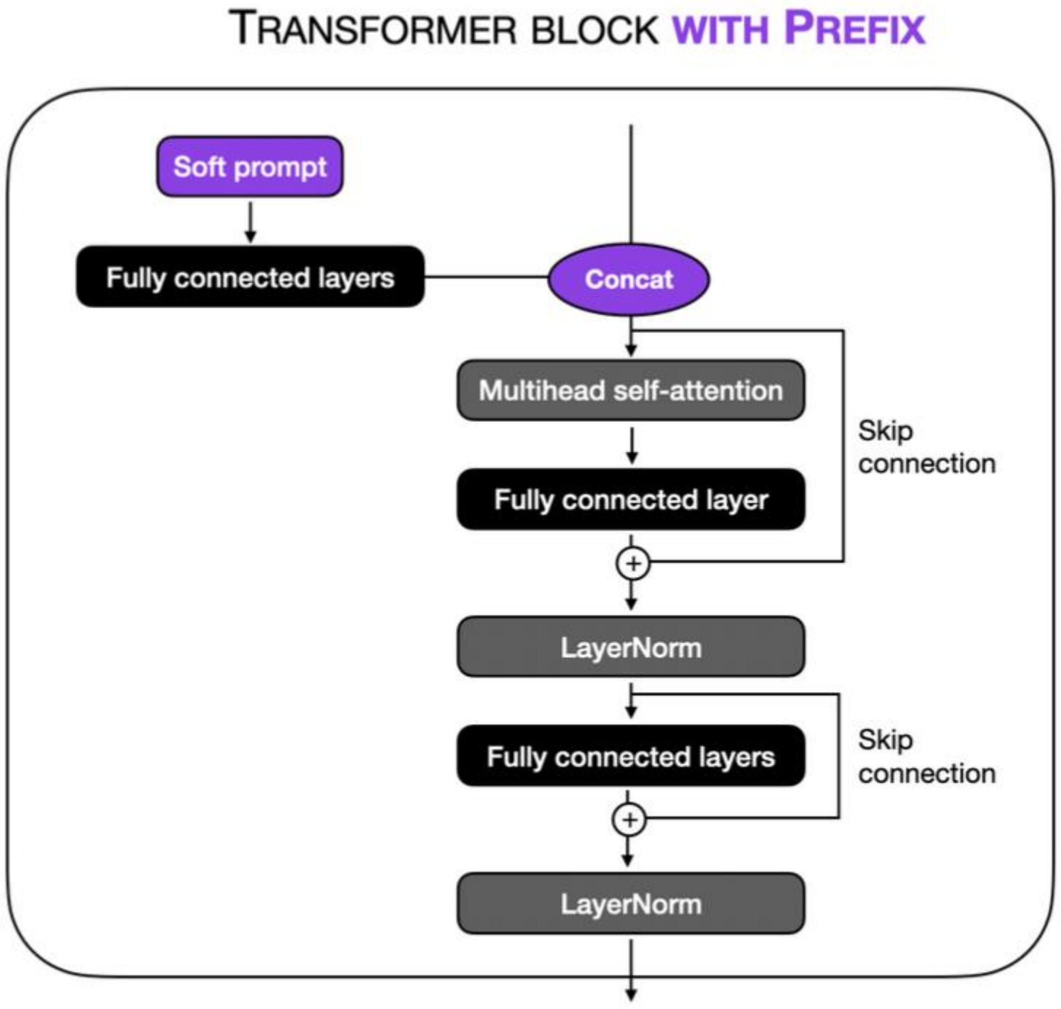


Source: http://dx.doi.org/10.1007/s10489-023-04723-0

Finetuning methods

Prefix tuning

- Applied to prompt before attention.
- Vary prompt by concatenating embeddings of input tensor with trainable tensor for target task.
- Shows inferior performance modeling long sequences.
- Memory footprint decreased, high hardware barrier with larger models.



Source: https://magazine.sebastianraschka.com/p/understanding-parameter-efficient

Inspiration for LoRA

3255v1 [cs.LG] 22 Dec 2020

INTRINSIC DIMENSIONALITY EXPLAINS THE EFFECTIVENESS OF LANGUAGE MODEL FINE-TUNING

Armen Aghajanyan, Luke Zettlemoyer, Sonal Gupta
Facebook
{armenag,lsz,sonalgupta}@fb.com

ABSTRACT

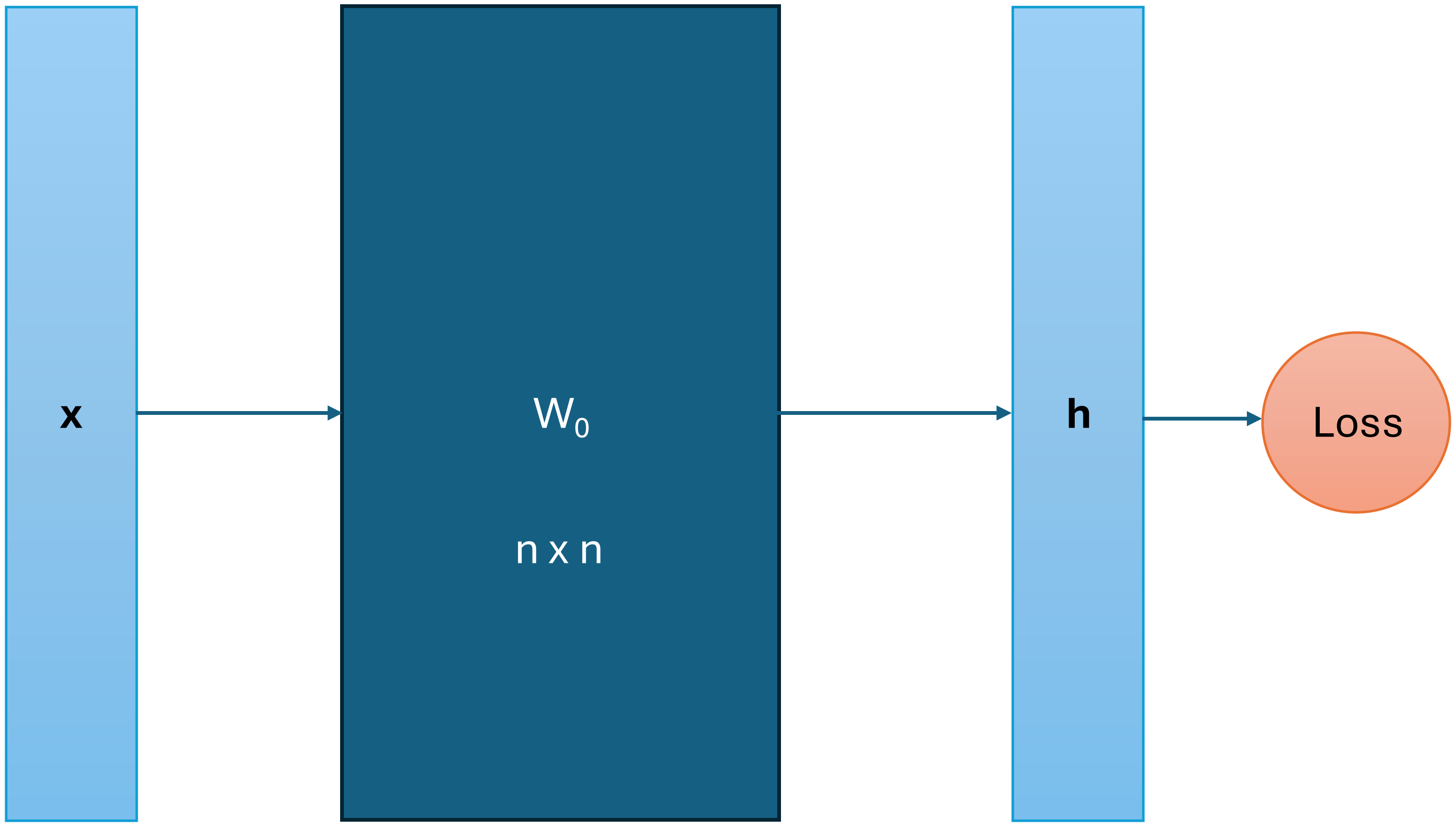
Although pretrained language models can be fine-tuned to produce state-of-the-art results for a very wide range of language understanding tasks, the dynamics of this process are not well understood, especially in the low data regime. Why can we use relatively vanilla gradient descent algorithms (e.g., without strong regularization) to tune a model with hundreds of millions of parameters on datasets with only hundreds or thousands of labeled examples? In this paper, we argue that analyzing fine-tuning through the lens of intrinsic dimension provides us with empirical and theoretical intuitions to explain this remarkable phenomenon. We empirically show that common pre-trained models have a very low intrinsic dimension; in other words, there exists a low dimension reparameterization that is as effective for fine-tuning as the full parameter space. For example, by optimizing only 200 trainable parameters randomly projected back into the full space, we can tune a RoBERTa model to achieve 90% of the full parameter performance levels on MRPC. Furthermore, we empirically show that pre-training implicitly minimizes intrinsic dimension and, perhaps surprisingly, larger models tend to have lower intrinsic dimension after a fixed number of pre-training updates, at least in part explaining their extreme effectiveness. Lastly, we connect intrinsic dimensionality with low dimensional task representations and compression based generalization bounds to provide intrinsic-dimension-based generalization bounds that are independent of the full parameter count.

- The paper shows that common pre-trained models have a very low intrinsic dimension:
    - There exists a low-dimensional reparameterization that is almost as effective for fine-tuning as the full parameter space.
- Pre-training minimizes this intrinsic dimension, such that larger models tend to have a smaller intrinsic dimension.

- They used arbitrary projection into a subspace with user-defined dimensionality.
- LoRA builds on this method through a much smarter low-dimensionality representation.

Finetuning

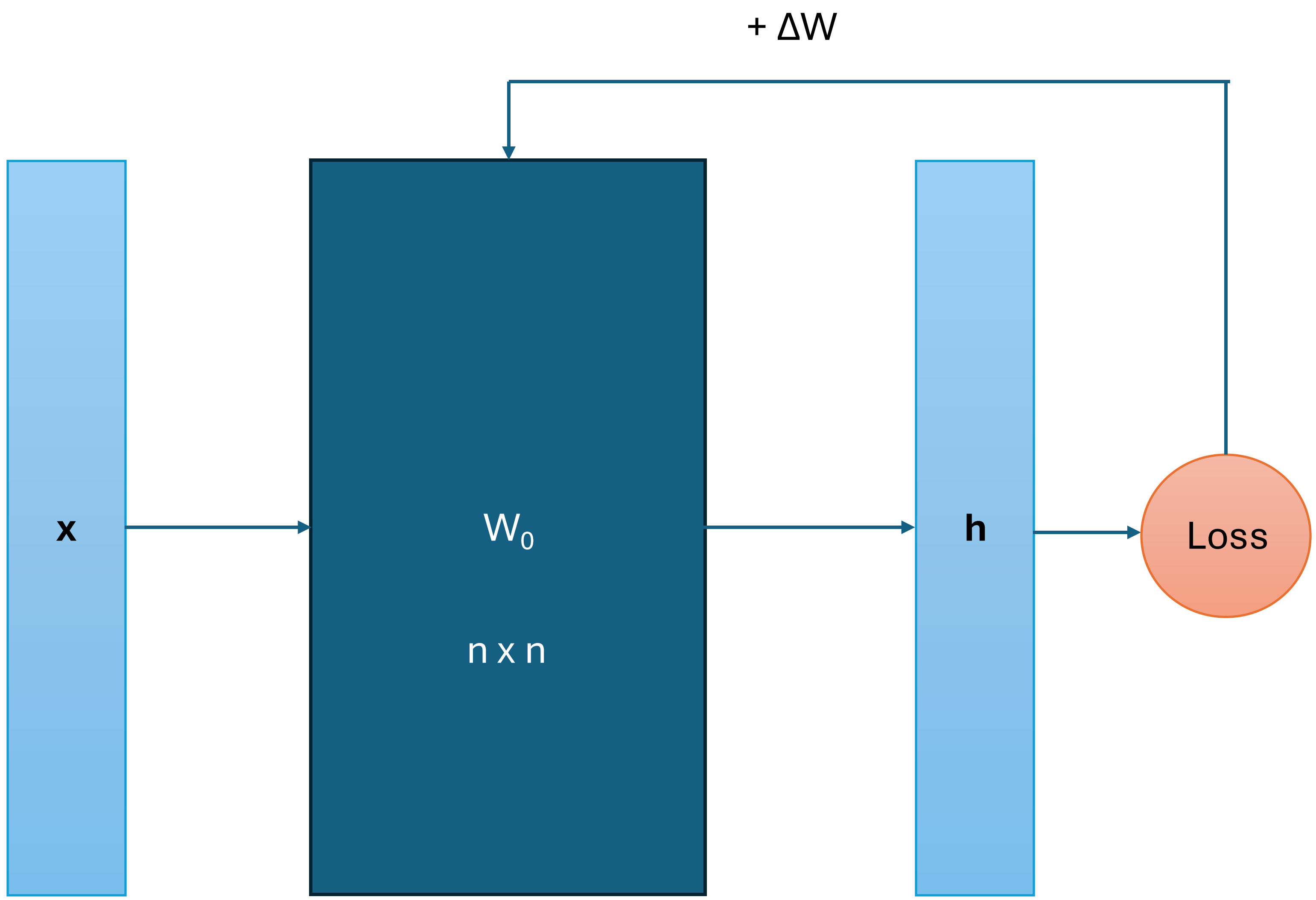
Forward Pass:
$h = W_0 x$



Finetuning

Forward Pass:
$h = W_0 x$

Backpropagation:
$W_0 = W_0 + \Delta W$

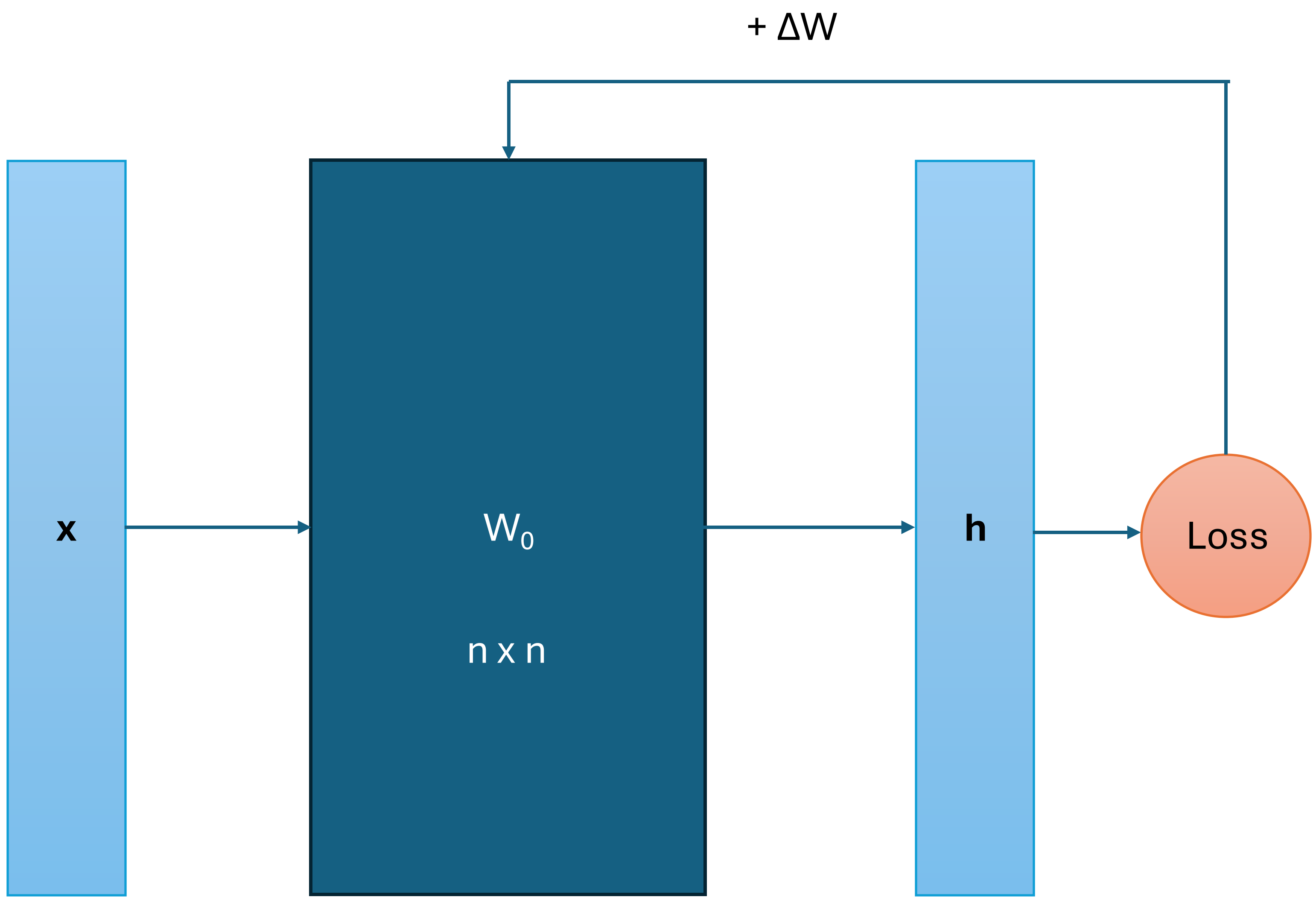


Finetuning

Forward Pass:
$h = W_0 x$

Backpropagation:
$W_0 = W_0 + \Delta W$

$$|\Delta W| = |W_0|$$



AB Decomposition

Basic Gradient Updates:

$$W' = W + \Delta W$$

$$\begin{pmatrix} 0.276 & -0.59 & 0.325 & 1.386 \\ -1.061 & 0.187 & 0.304 & -0.488 \\ 0.314 & 0.763 & 1.398 & -0.049 \\ 0.303 & -0.115 & 0.446 & 0 \end{pmatrix} \quad \begin{pmatrix} 0.123 & -0.445 & 0.543 & 0.774 \\ -0.841 & -0.017 & -0.004 & 0.376 \\ 0.009 & 0.923 & 0.764 & -0.196 \\ 0.373 & 0.089 & 0.2 & -0.523 \end{pmatrix} \quad \begin{pmatrix} 0.153 & -0.145 & -0.218 & 0.612 \\ -0.22 & 0.204 & 0.308 & -0.864 \\ 0.305 & -0.16 & 0.634 & 0.147 \\ -0.07 & -0.204 & 0.246 & 0.523 \end{pmatrix}$$

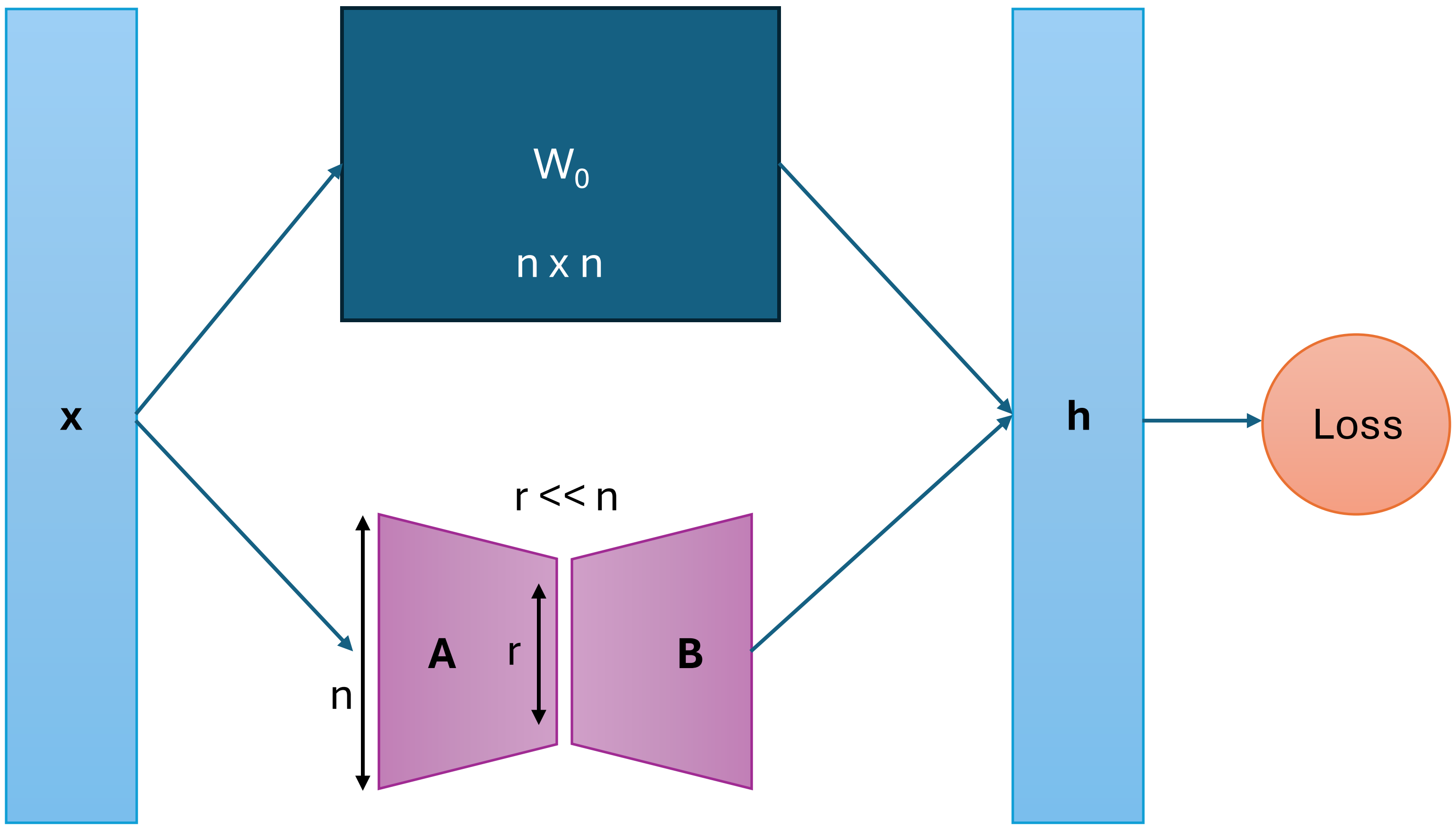
LoRA Gradient Updates:

$$W' = W + alpha/r * B @ A$$

$$\begin{pmatrix} 0.123 & -0.445 & 0.543 & 0.774 \\ -0.841 & -0.017 & -0.004 & 0.376 \\ 0.009 & 0.923 & 0.764 & -0.196 \\ 0.373 & 0.089 & 0.2 & -0.523 \end{pmatrix} \quad \begin{pmatrix} 0.123 & -0.445 & 0.543 & 0.774 \\ -0.841 & -0.017 & -0.004 & 0.376 \\ 0.009 & 0.923 & 0.764 & -0.196 \\ 0.373 & 0.089 & 0.2 & -0.523 \end{pmatrix} \quad \begin{pmatrix} 0.3 & -0.14 \\ -0.42 & 0.201 \\ 0.46 & 0.38 \\ 0.5 & 0.14 \end{pmatrix} \begin{pmatrix} 0.1 & -0.44 & 0.04 & 1.42 \\ -0.92 & 0.1 & 1.62 & -1.33 \end{pmatrix}$$

Finetuning

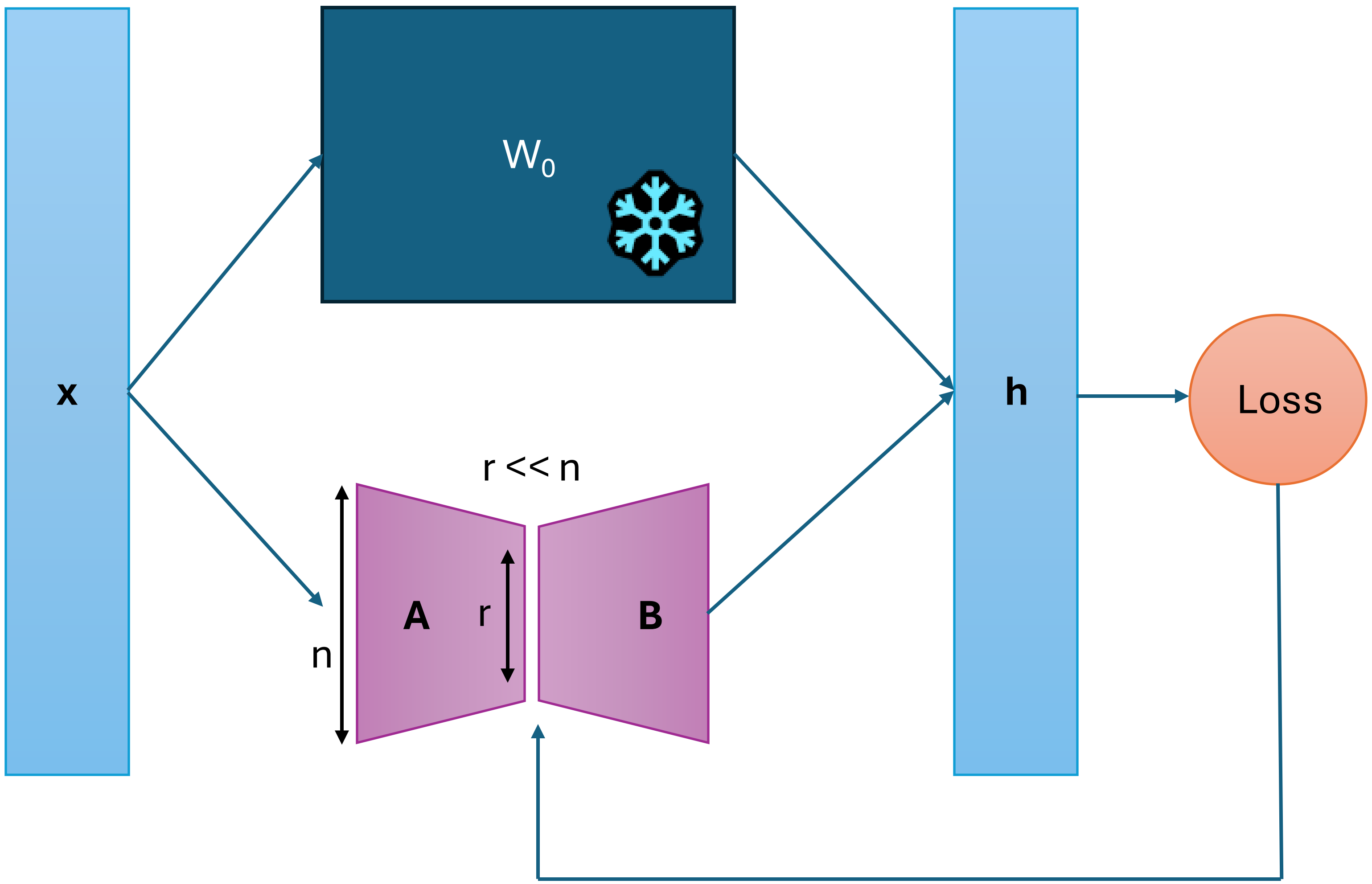
Forward Pass:
$h = W_0x + BA\,x$



Finetuning

Forward Pass:
$h = W_0x + BA\ x$

Backpropagation:
Only update parameters in B & A matrices



LoRA Intuition

- During regular finetuning, the model is initialized to pretrained weights.
- Model: $P_\varphi(y\,|x)$

Downstream Task Data: $Z = \{(x_i, y_i)\}_{i=1..,N}$
(both x_i & y_i are sequence of tokens)

Model initialized to φ_0 and updated using: $\varphi_0 + \Delta\varphi$

By repeatedly following the conditional language modeling objective:

$$\max_{\Phi} \sum_{(x,y)\in\mathcal{Z}} \sum_{t=1}^{|y|} \log\left(P_{\Phi}(y_t|x, y_{<t})\right)$$

LoRA Intuition

- During regular finetuning, the model is initialized to pretrained weights.
- Model: $P_{\varphi}(y \mid x)$

Downstream Task Data: $Z = \{(x_i, y_i)\}_{i=1..,N}$
(both x_i & y_i are sequence of tokens)

Model initialized to φ_0 and updated using: $\varphi_0 + \Delta\varphi$

By repeatedly following the conditional language modeling objective:

$$\max_{\Phi} \sum_{(x,y)\in\mathcal{Z}} \sum_{t=1}^{|y|} \log\left(P_{\Phi}(y_t|x, y_{<t})\right)$$

$$|\Delta\varphi| = |\varphi_0|$$

$r = n$

LoRA Intuition

Encode $\Delta\varphi = \Delta\varphi(\theta)$ where $|\theta| \ll |\varphi_0|$

Therefore, finding $\Delta\varphi$ we need to optimize over smaller parameter space θ

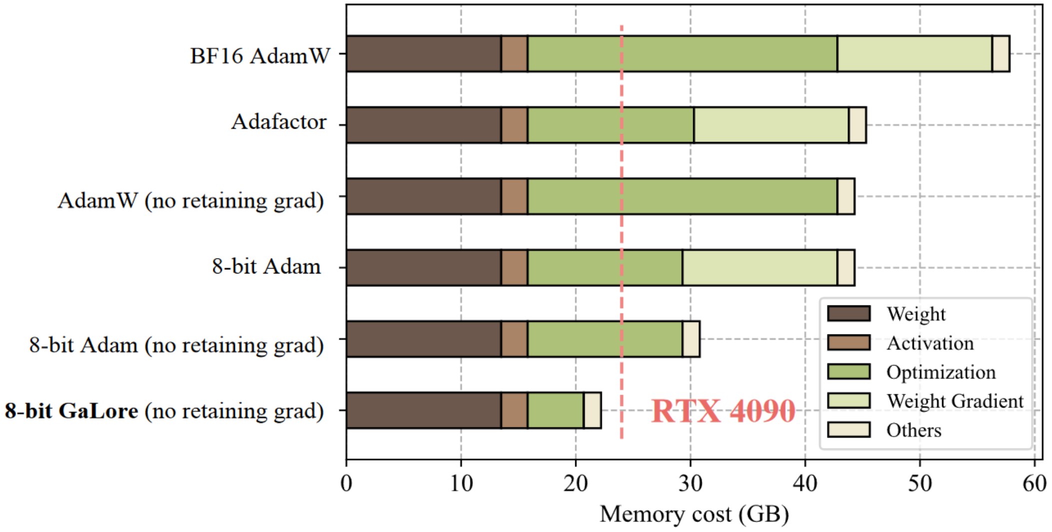
$$\max_{\Theta} \sum_{(x,y)\in\mathcal{Z}} \sum_{t=1}^{|y|} \log\left(p_{\Phi_0+\Delta\Phi(\Theta)}(y_t|x, y_{<t})\right)$$

Benefits

- Switching overhead reduced, as only need to swap 2 matrices (A & B) to change context, when pretrained model is already stored.

"Another benefit is that we can switch between tasks while deployed at a much lower cost by only swapping the LoRA weights as opposed to all the parameters" – Hu, et. al.

- Can be combined with other finetuning methods.
- Better HW optimization as don't need to store the optimization states of frozen weights, only finetuning.
- Freezing pre-trained weights ensures both less overfitting and catastrophic forgetting.



Source: https://doi.org/10.48550/arXiv.2403.03507

Code Implementation

1. Applying LoRA on Avi's transformers implementation
2. Applying LoRA to finetune models such as LLAMA3