THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
RELAXED SYSTEM LAB

Parameter Efficient Fine-Tuning

COMP4901Y

Binhang Yuan

Fine-Tuning

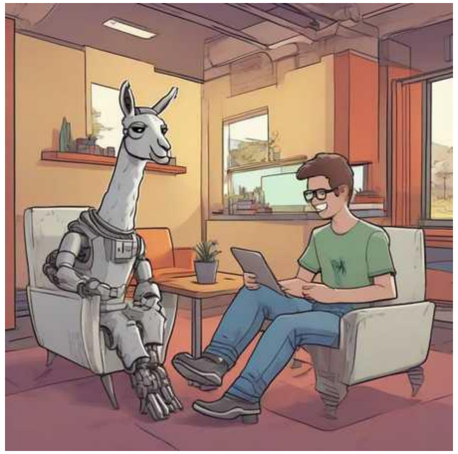
How LLM Are Usually Deployed?

- **Pre-training** is the initial phase of training an LLM, during which it learns from a large, diverse dataset, often consisting of trillions of tokens.
  - The goal is to develop a broad understanding of language, context, and various types of knowledge for the model.
  - Pre-training is usually computationally intensive (thousands of GPUs for weeks) and requires huge amounts of data (trillions of tokens).
- **Fine-tuning** is where you take an already pre-trained model and further train it on a more specific dataset.
  - This dataset is typically smaller and focused on a particular domain or task.
  - The purpose of fine-tuning is to adapt the model to perform better in specific scenarios or on tasks that were not well covered during pre-training.
  - The new knowledge added during fine-tuning enhances the model's performance in specific contexts rather than broadly expanding its general knowledge.

How LLM Are Usually Deployed?

<u>Stage 1: Pretraining</u>

1. Prepare 10TB of text as the training corpus.
2. Use a cluster of thousands of GPUs to train a neural network with the corpus from scratch.
3. Obtain the ***base model***.



<u>Stage 2: Fine-tuning</u>

1. Prepare 100K high-quality text as the task-specific training dataset.
2. Use a small-scale GPU cluster to fine-tune the base model with the training dataset.
3. Obtain the ***domain-specific model***.
4. Run the domain-specific model for evaluation.
5. Repeat until we are happy with the results.

Instruction Tuning

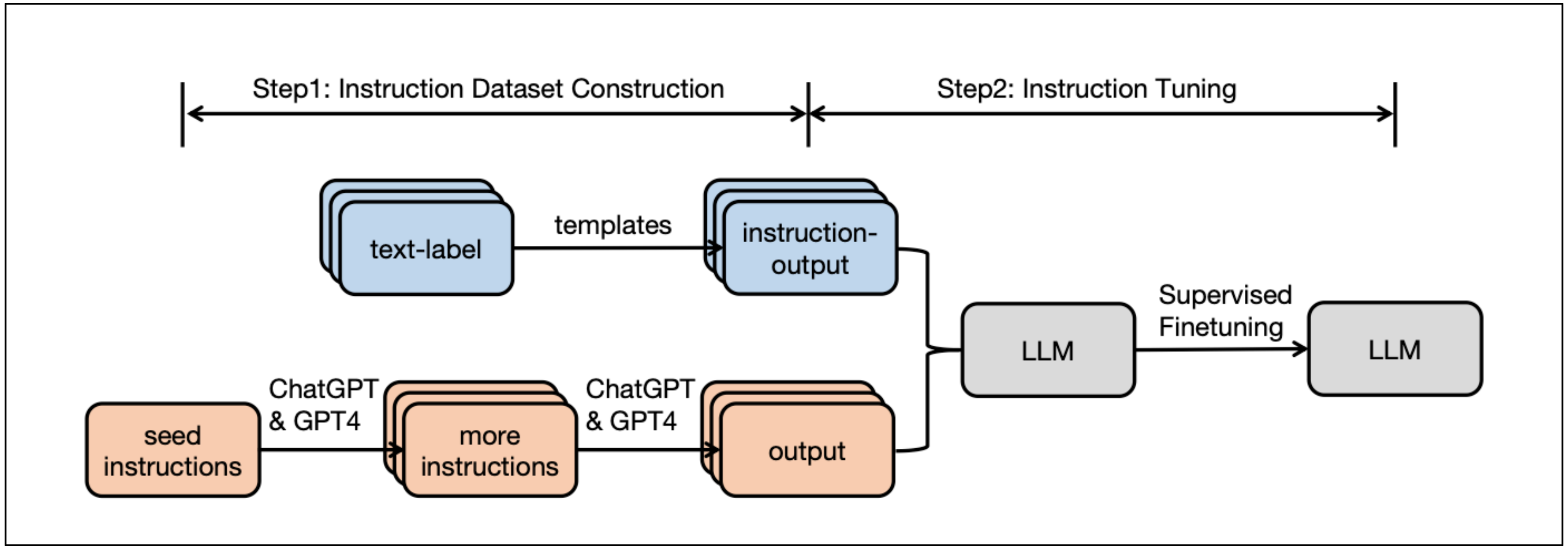
RELAXED SYSTEM LAB

- Fine-tune the LLM using ***(Instruction, output)*** pairs:
  - *Instruction*: denotes the human instruction for the model;
  - *Output*: denotes the desired output that follows the instruction.
- Finetuning an LLM on the instruction dataset bridges the gap between the next-word prediction objective of LLMs and the users' objective of instruction following;
- Instruction tuning allows for a more controllable and predictable model behavior compared to standard LLMs.
  - The instructions serve to constrain the model's outputs to align with the desired response characteristics or domain knowledge, providing a channel for humans to intervene with the model's behaviors.
- Instruction tuning can help LLMs rapidly adapt to a specific domain without extensive retraining or architectural changes.

Instruction Tuning

RELAXED SYSTEM LAB

- Two methods to construct the instruction datasets:
  - Data integration from annotated natural language datasets: (instruction, output) pairs are collected from existing annotated natural language datasets by using templates to transform text-label pairs to (instruction, output) pairs.
  - Generate outputs using more advanced LLMs: employ LLMs such as GPT-3.5-Turbo or GPT4 to gather the desired outputs given the instructions instead of manually collecting the outputs.



Fine-Tuning v.s. Prompt Engeering

- Suppose we have:
  - A dataset $D = \{(x_i, y_i)\}_{i=1}^{N}$ and N is rather small.
  - A pre-trained LLM.
- How to fit it to your task?

- Option A: Fine-tuning:
  - Fine-tune the LLM on the training data using:
    - A standard training objective;
    - SGD to update (part of) the LLM's parameters.
  - Advantages:
    - Fits into the standard ML recipe;
    - Still works if N becomes relatively large.
  - Disadvantages:
    - Backward pass is computationally expensive in terms of FLOPs and memory footprint;
    - You have to have full access of the pre-trained LLM.

- Option B: Prompty engineering (in-context learning):
  - Feed training examples to the LLM as a prompt:
    - Allow the LLM to infer patterns in the training examples during inference;
    - Take the output of the LLM following the prompt as its prediction.
  - Advantages:
    - No backpropagation required and only one pass through the training data;
    - Does not require model weights, only API access.
  - Disadvantages:
    - The prompt may be very long.

Fine-Tuning v.s. Prompt Engeering

RELAXED SYSTEM LAB

- Why would we ever bother with fine-tuning if it's so inefficient?
  - Because, even for very large LMs, fine-tuning often beats in-context learning.
  - In a fair comparison of fine-tuning (FT) and in-context learning (ICL), we find that FT outperforms ICL for most model sizes.

	FT						
ICL	**125M**	**350M**	**1.3B**	**2.7B**	**6.7B**	**13B**	**30B**
125M	−0.00	0.01	0.02	0.03	0.12	0.14	0.09
350M	−0.00	0.01	0.02	0.03	0.12	0.14	0.09
1.3B	−0.00	0.01	0.02	0.03	0.12	0.14	0.09
2.7B	−0.00	0.01	0.02	0.03	0.12	0.14	0.09
6.7B	−0.00	0.01	0.02	0.03	0.12	0.14	0.09
13B	−0.04	−0.02	−0.01	−0.00	0.09	0.11	0.05
30B	−0.11	−0.09	−0.08	−0.08	0.02	0.03	−0.02

(a) RTE

	FT						
ICL	**125M**	**350M**	**1.3B**	**2.7B**	**6.7B**	**13B**	**30B**
125M	−0.00	0.00	0.02	0.01	0.10	0.11	0.07
350M	−0.00	0.00	0.02	0.01	0.10	0.11	0.07
1.3B	−0.01	−0.00	0.01	0.01	0.10	0.11	0.07
2.7B	−0.01	−0.00	0.01	0.01	0.09	0.10	0.07
6.7B	−0.01	−0.01	0.01	0.00	0.09	0.10	0.06
13B	−0.03	−0.03	−0.02	−0.02	0.07	0.08	0.04
30B	−0.07	−0.07	−0.05	−0.06	0.03	0.04	0.00

(b) MNLI

Table 1: Difference between average **out-of-domain performance** of ICL and FT on RTE (a) and MNLI (b) across model sizes. We use 16 examples and 10 random seeds for both approaches. For ICL, we use the `gpt-3` pattern. For FT, we use pattern-based fine-tuning (PBFT) and select checkpoints according to in-domain performance. We perform a Welch's t-test and color cells according to whether: ICL performs significantly better than FT, FT performs significantly better than ICL. For cells without color, there is no significant difference.

https://aclanthology.org/2023.findings-acl.779.pdf

RELAXED
SYSTEM LAB

Parameter Efficient Fine-Tuning

Parameter Efficient Fine-Tuning

- *Parameter efficient fine-tuning (PEFT):* Rather than finetuning the entire model, we finetune only small amounts of weights.
- Goal: achieve performance on a downstream task that is comparable to fine-tuning all parameters.
- Some approaches:
    - Frozen layer/Subset fine-tuning: pick a subset of the parameters, fine-tune only those layers, and freeze the rest of the layers.
    - Adapters: add additional layers that have few parameters and tune only the parameters of those layers, keeping all others fixed.
    - Low-rank adaption (LoRA): learn a low rank approximation of the weight matrices.

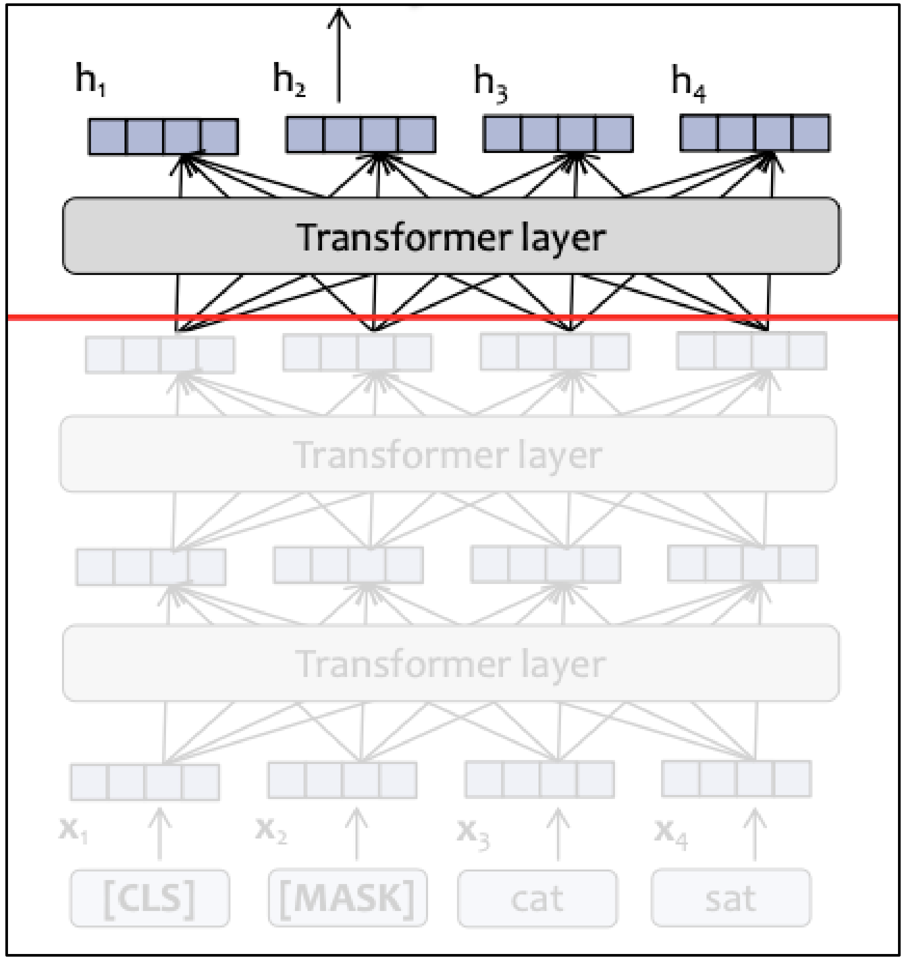
Subset Fine-Tuning

- Some interpretations from NLP research:
  - Earlier layers of the transformer tend to capture linguistic phenomena and basic language understanding;
  - Later layers are where the task-specific learning happens.
- We should be able to learn new tasks by freezing the earlier layers and tuning the later ones.
- This can be a simple baseline for PEFT.

Subset Fine-Tuning

RELAXED SYSTEM LAB

- Keep all parameters fixed except for the top K layers.
- Gradients only need to flow through top K layers instead of all of layers.
- Reduce computation load.
- Reduce memory usage.

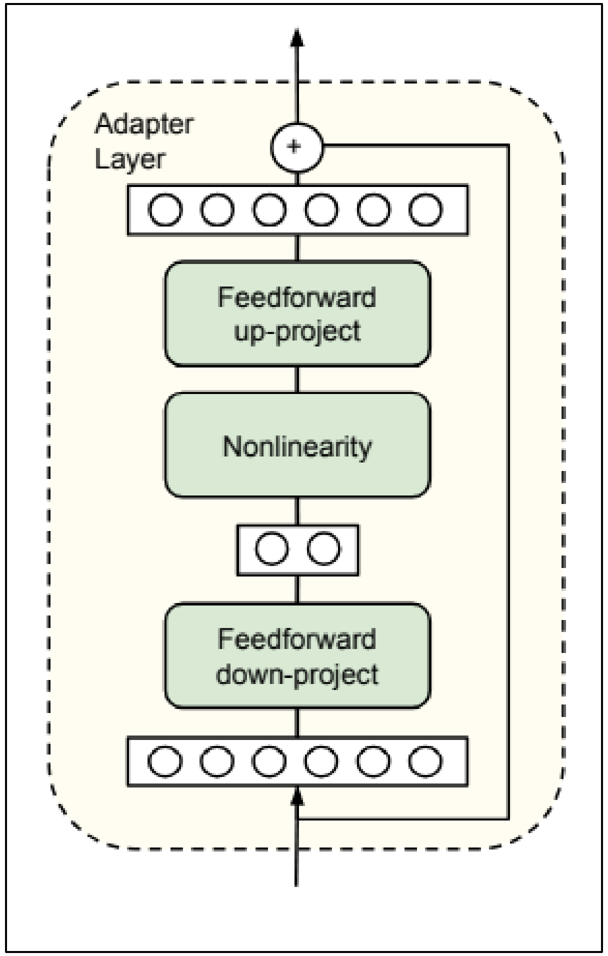


The gradient does not backpropagate to lower layers.

Adapter

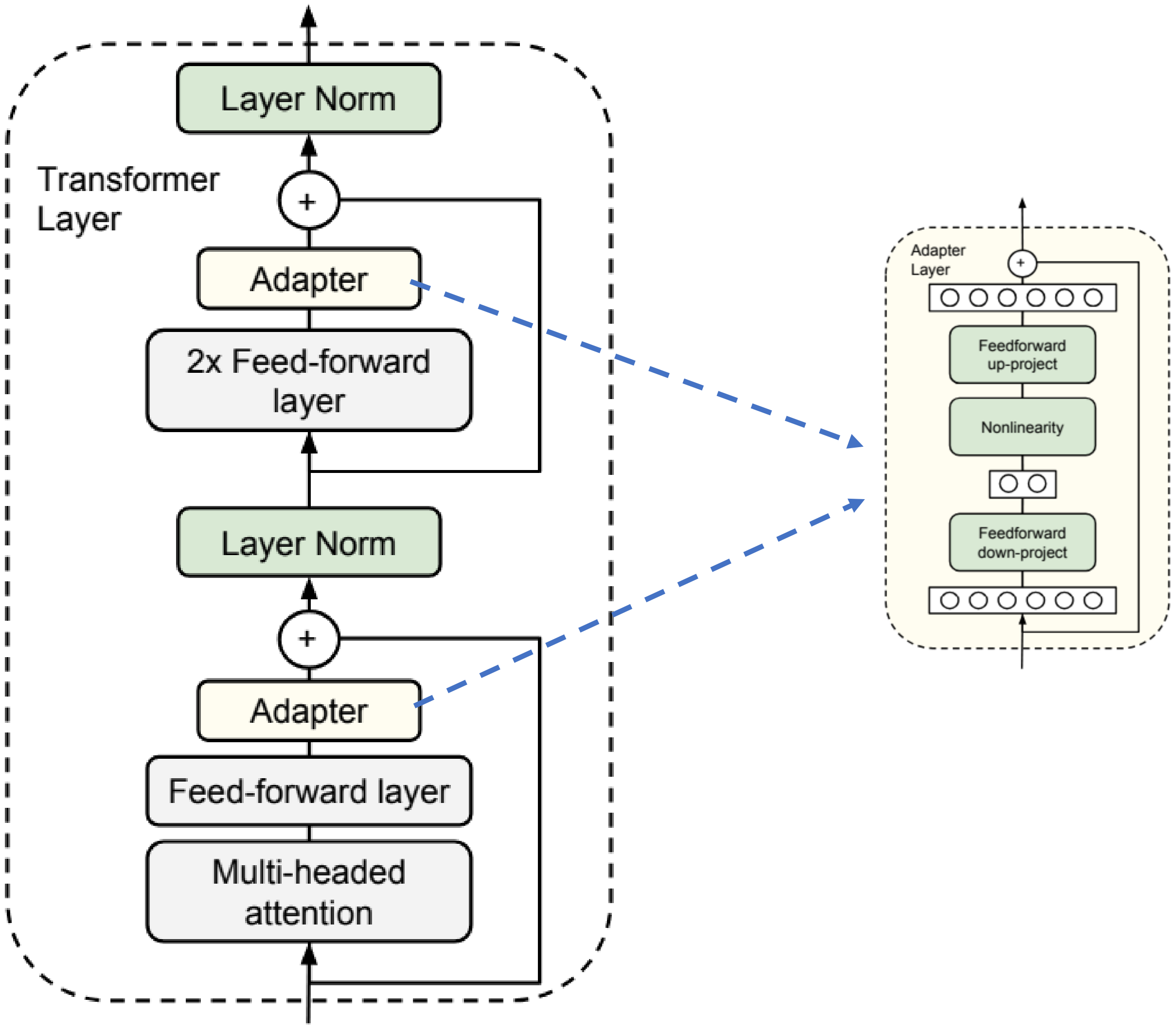
RELAXED SYSTEM LAB

- Adapters are new modules are added between layers of a pre-trained network.
  - The original model weights are fixed;
  - Just the adapter modules are tuned.
- An adapter layer is simply a feed- forward neural network with one hidden layer, and a residual connection.
- Suppose the original LLM has a model dimension of D:
  - For the down-project weight matrix: $W_A \in \mathbb{R}^{D \times R}$;
  - For the up-project weight matrix: $W_B \in \mathbb{R}^{R \times D}$;
  - We have $R \ll D$.



Adapter

- Given $R \ll D$, in practice the adapter layers contain only 0.5% – 8% of the total parameters.
- When added to a deep neural network (e.g. transformer) all the other parameters of the pretrained model are kept fixed, and only the adapter layer parameters are fine-tuned.



Low-Rank Adaptation (LoRA)

Low-Rank Adaption

RELAXED SYSTEM LAB

- Central idea:
  - *"How can we re-parameterize the model into something more efficient to train?"*
- Finetuning has a low *intrinsic dimension*, that is, the minimum number of parameters needed to be modified to reach satisfactory performance is not very large.
- This means we can re-parameterize a subset of the original model parameters with low-dimensional proxy parameters, and just optimize the proxy.

Intrinsic Dimension

- An objective function's intrinsic dimension measures the minimum number of parameters needed to reach a satisfactory solution to the objective.
- Can also be thought of as the lowest dimensional subspace in which one can optimize the original objective function to within a certain level of approximation error.
- Details in this paper: https://arxiv.org/abs/2012.13255
  - Suppose we have model parameters $\theta^{(D)} \in \mathbb{R}^D$, D is the number of parameters;
  - Instead of optimizing $\theta^{(D)}$, we could instead optimize a smaller set of parameters $\theta^{(d)} \in \mathbb{R}^d$, where $d \ll D$.
  - This done through clever factorization:
    - $\theta^{(D)} = \theta_0^{(D)} + P\left(\theta^{(d)}\right)$; where $P\colon \mathbb{R}^d \to \mathbb{R}^D$
    - P is typically a linear projection: $\theta^{(D)} = \theta_0^{(D)} + \theta^{(d)} M$.
  - Intuitively, we do an arbitrary random projection onto a much smaller space; usually, a linear projection, we then solve the optimization problem in that smaller subspace. If we reach a satisfactory solution, we say the dimensionality of that subspace is the intrinsic dimension.

Low-Rank Adaption

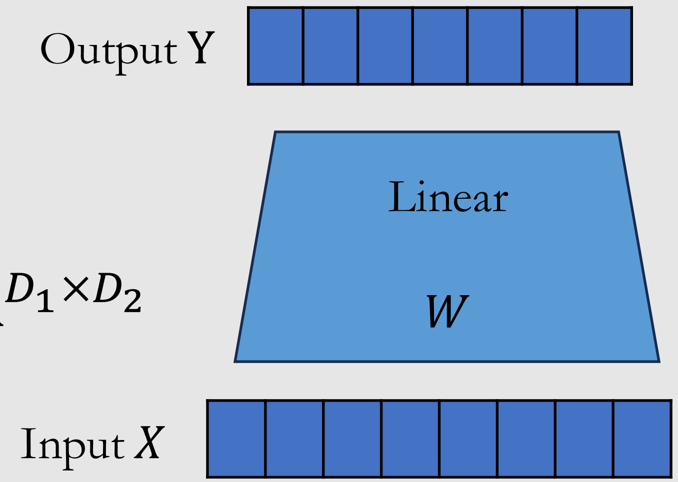
- Full paper: https://arxiv.org/pdf/2106.09685.
- Intuition: It's not just the model weights that are low rank, updates to the model weights are also low-rank.
- LoRA freezes the pre-trained model weights and injects trainable rank decomposition matrices into some or all layers.

LoRA Key Idea

- Keep the original pre-trained parameters W fixed during fine-tuning;
- Learn an additive modification to those parameters $\Delta \mathrm{W}$;
- Define $\Delta \mathrm{W}$ as a low-rank decomposition: $\Delta \mathrm{W} = AB$.

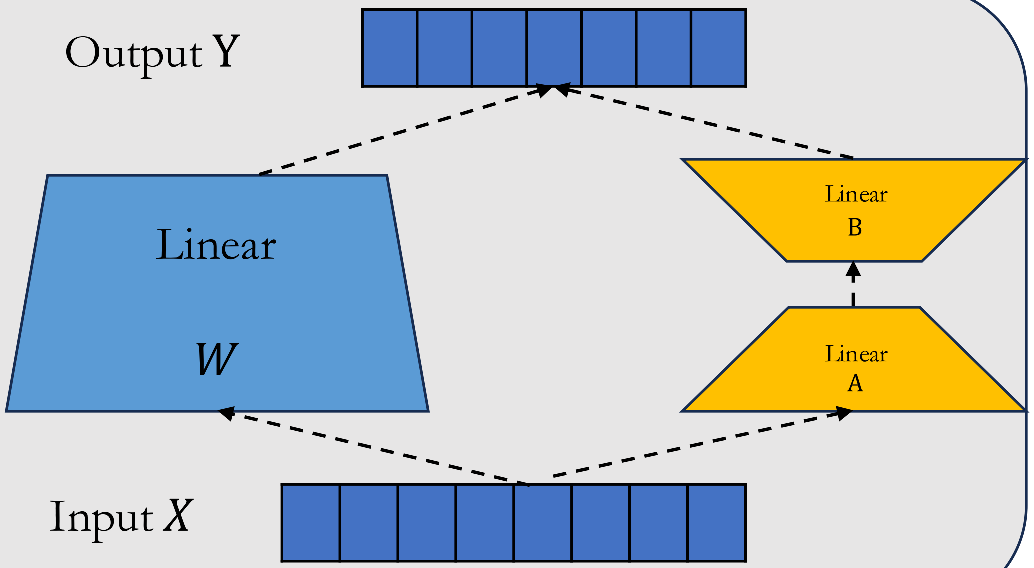
Standard Linear Layer

- $Y = XW$
- $X \in \mathbb{R}^{D_1}, Y \in \mathbb{R}^{D_2}, \mathrm{W} \in \mathbb{R}^{D_1 \times D_2}$



LoRA Linear Layer

- $Y = XW + XAB = X(W + AB)$
- $X \in \mathbb{R}^{D_1}, Y \in \mathbb{R}^{D_2}, \mathrm{W} \in \mathbb{R}^{D_1 \times D_2}$
- $A \in \mathbb{R}^{D_1 \times R}, B \in \mathbb{R}^{R \times D_2}$

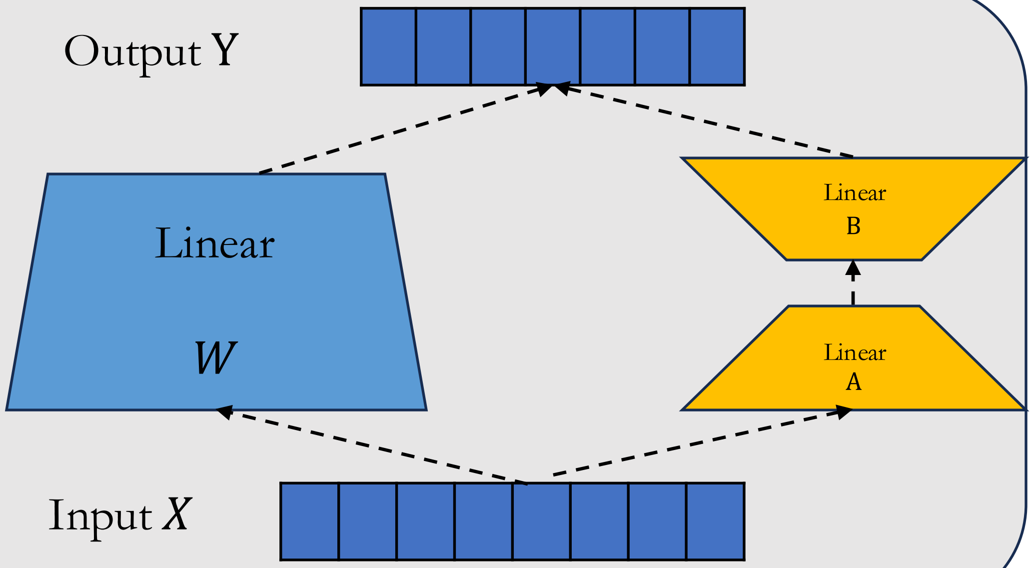


LoRA Initialization

- We initialize the trainable parameters:
  - $A_{ij} \sim \mathcal{N}(0, \sigma^2) \quad B_{ij} = 0$
- This ensures that, at the start of fine-tuning, the parameters have their pre-trained values:
  - $\Delta W = AB = 0$

LoRA Linear Layer

- $Y = XW + XAB = X(W + AB)$
- $X \in \mathbb{R}^{D_1}, Y \in \mathbb{R}^{D_2}, \mathrm{W} \in \mathbb{R}^{D_1 \times D_2}$
- $A \in \mathbb{R}^{D_1 \times R}, B \in \mathbb{R}^{R \times D_2}$



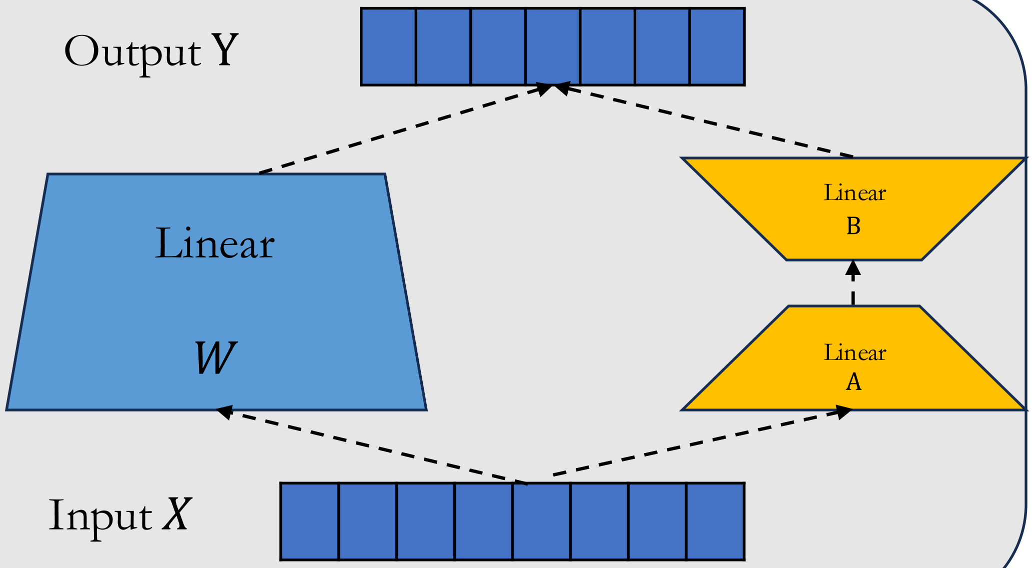
LoRA Hot Swapping Parameters

- W and AB the same dimension, so we can swap the LoRA parameters in and out of a Standard Linear Layer.

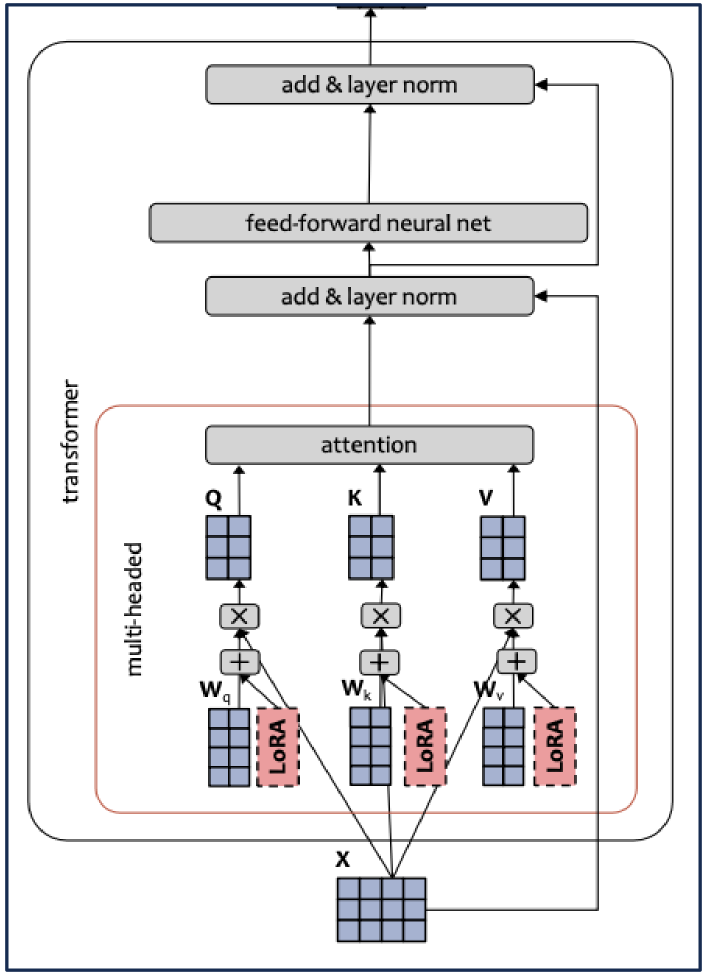
- To include LoRA:
  - $W' \leftarrow W + AB$
- To remove LoRA:
  - $W \leftarrow W' - AB$

LoRA Linear Layer

- $Y = XW + XAB = X(W + AB)$
- $X \in \mathbb{R}^{D_1}, Y \in \mathbb{R}^{D_2}, \mathrm{W} \in \mathbb{R}^{D_1 \times D_2}$
- $A \in \mathbb{R}^{D_1 \times R}, B \in \mathbb{R}^{R \times D_2}$



LoRA for Transformers



- LoRA linear layers could replace every linear layer in the Transformer layer;
- But the original paper only applies LoRA to the attention weights:
  - $Q = \text{LoRALinear}(X, W_q, A_q, B_q)$
  - $K = \text{LoRALinear}(X, W_k, A_k, B_k)$
  - $V = \text{LoRALinear}(X, W_v, A_v, B_v)$
  - Some further research found that most efficient to include LoRA only on the query and key linear layers.

LoRA Results

- Some emprical takeaways:
  - Applied to GPT-3, LoRA achieves performance almost as good as full parameter fine-tuning, but with far fewer parameters.
  - On some tasks, it even outperforms full fine-tuning.
  - For some datasets, a rank of $R = 1$ is sufficient.
  - LoRA performs well when the dataset is large or small.

References

- https://www.youtube.com/watch?v=zjkBMFhNj_g
- https://arxiv.org/abs/2308.10792
- https://www.andrew.cmu.edu/course/11-667/lectures/W4L2_PETM.pptx.pdf
- https://www.cs.cmu.edu/~mgormley/courses/10423//slides/lecture11-peft-ink.pdf
- https://arxiv.org/abs/2012.13255
- https://arxiv.org/pdf/2106.09685

RELAXED SYSTEM LAB