New Jersey Institute of Technology

LoRa: Low Rank Adaptation of Large Language Models

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen

Naren Khatwani, Sabbir Ahmed Saqlain

Introduction

Large Language Models:

- Advanced artificial intelligence systems are designed to understand, generate, and interact using human language by processing vast amounts of textual data.

Challenges with LLMs:

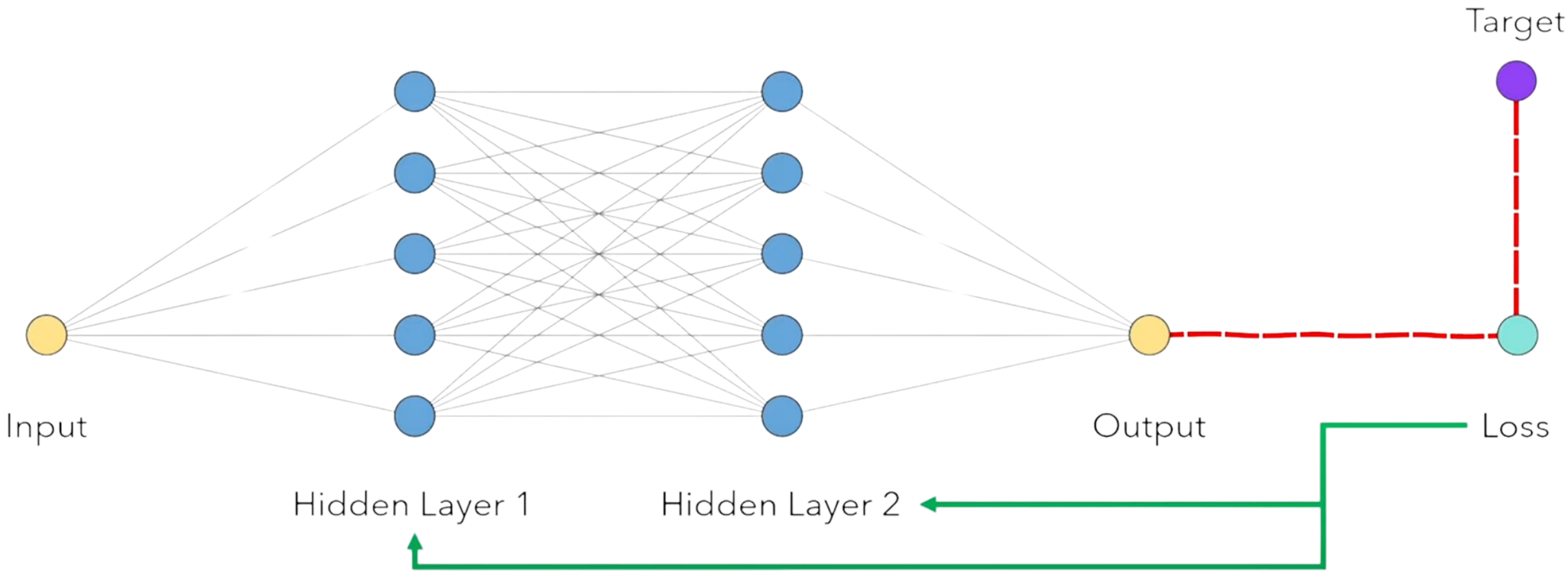
- Computational Cost
- Model Size
- Inflexibility

Why Adaptation Matters:

- Need for Task-Specific Models
- Potential Solutions

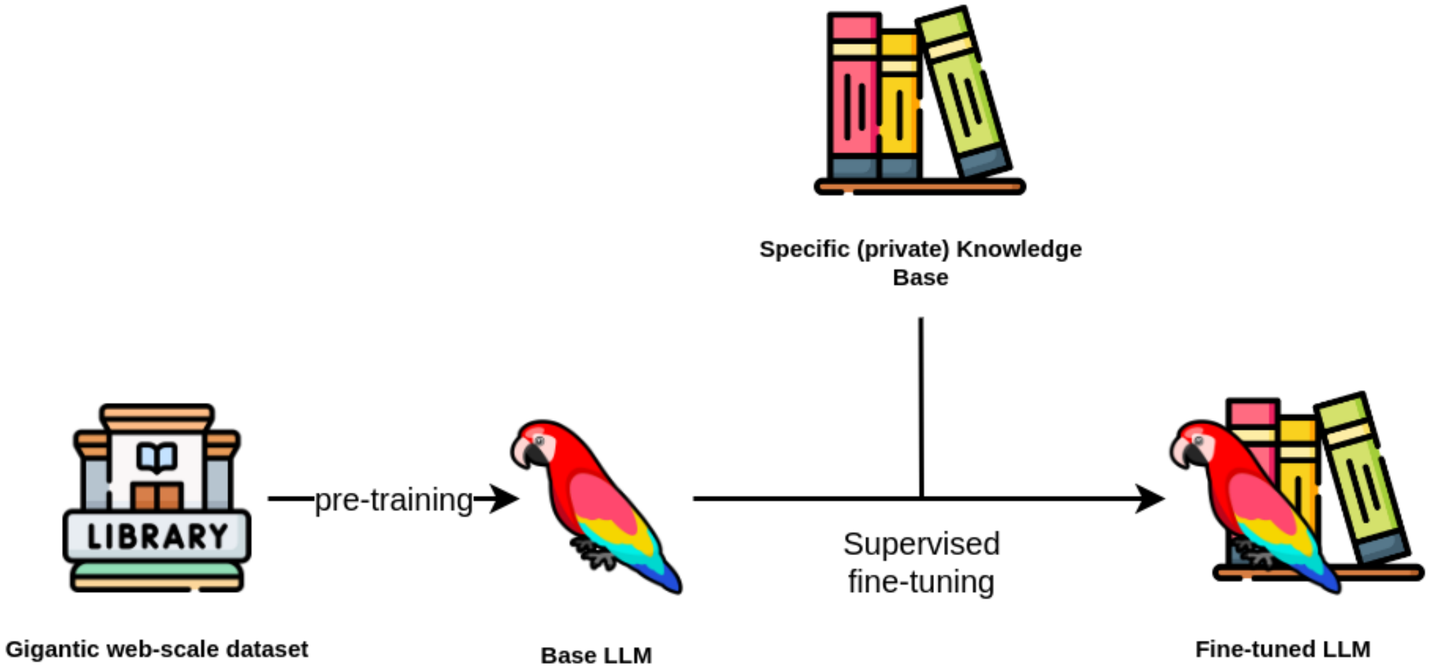
Background

How do neural networks work ?



What is Fine Tuning ?

- Fine tuning means **training a pre-trained network** on new data to improve its performance on a specific task



- **Example:** Base LLM - trained on many programming languages
  - **Fine Tune** it for a **new/specific dialect of SQL**

Motivation

Why Study This Problem? Why did the authors choose this problem?

Fine Tuning Models is DIFFICULT !!!

Motivation

Why Fine Tuning Models is difficult?

- RoBERTa base **(125M)**
- RoBERTa large **(355M)**
- DeBERTa XXL **(1.5B)**
- GPT- 2 Medium **(355M)**
- GPT - 2 Large **(774M)**
- GPT - 3 **(175B)**

Large Number of Parameters

Fine Tuning consumes a lot of RAM and time

- Need to store the full model for each task
- GPT - 3 fine tuning requires 1.2TB VRAM

Problem Statement

- **Scaling up Large Language Models (LLMs)** for specific tasks **involves significant computational and resource challenges.**
- **Traditional fine-tuning methods** can be **inefficient and resource-intensive**, significantly as model sizes increase.

Goals of LoRa

- **Efficient Adaptation**
- **Maintaining Scalability**
- **Preserving Model Integrity**

Key Contributions of the Paper

- **Innovative Approach**
- **Experimental Validation**
- **Versatility and Performance**

LoRa revolutionizes the adaptation of large language models by introducing a low-rank matrix approach that enhances efficiency, preserves model integrity, and ensures scalability across various tasks.

Existing Solutions ?

- **Fine-tune the last k layers (say k =2)**

Example:
- **Transformer Model trained to translate b/w English and French**

New Goal:
- **Translate b/w English and Spanish**

Retrain the entire model ❌

Adjust parameters of last 2 layers of the model ✅

- **Bias only - Bit Fit**
  - Trains the Bias Vectors only

Adjust the threshold at which neurons activate

Effectively shifting the decision boundaries

Sentiment Analysis

Happy

Sad

Fine Tune Sarcasm

Existing Solutions ?

- **Prefix Embedding Tuning**
  - **Prefixing:** Prepends special tokens to the prompt
  - **Infixing:** Appends to the prompt
  - Special tokens have trainable word embeddings

Example - Prefixing

- [WEATHER_INFO] [LOCATION] What is the weather like today in Paris?
- **During Training**
  - The model learns that [WEATHER_INFO] should cue it to provide weather details
- **After Training**
  - Primed to focus on providing the weather for the mentioned location.

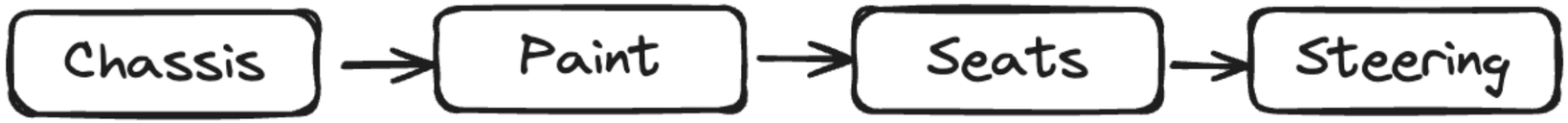
Example - Infixing

- I have a problem with [PRODUCT_NAME] my router, it [ISSUE_DESC] won't connect to the internet
- **After Training**
  - The model understands the structure of the inquiry

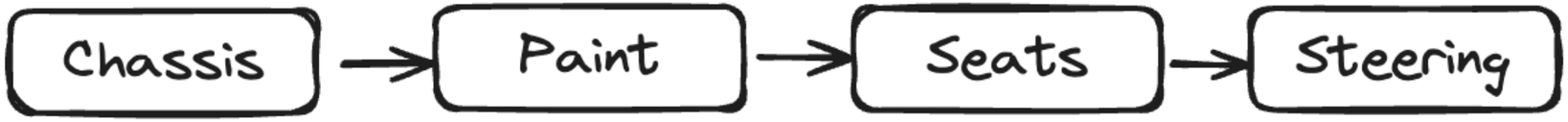
More Existing Solutions ?

- **Prefix-layer tuning (PreLayer)**
  - A technique where you **introduce additional trainable parameters** at different layers of the transformer model.
  - Instead of simply learning embeddings for special tokens, PreLayer involves learning a set of vectors (activations) that are added to the model's activations at each transformer layer during the forward pass.

Example - Car Factory



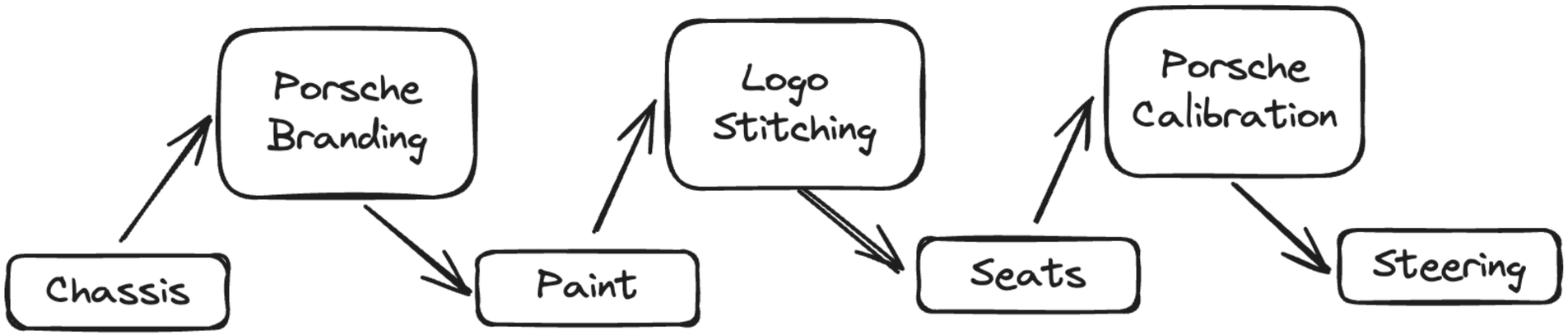
More Existing Solutions ?



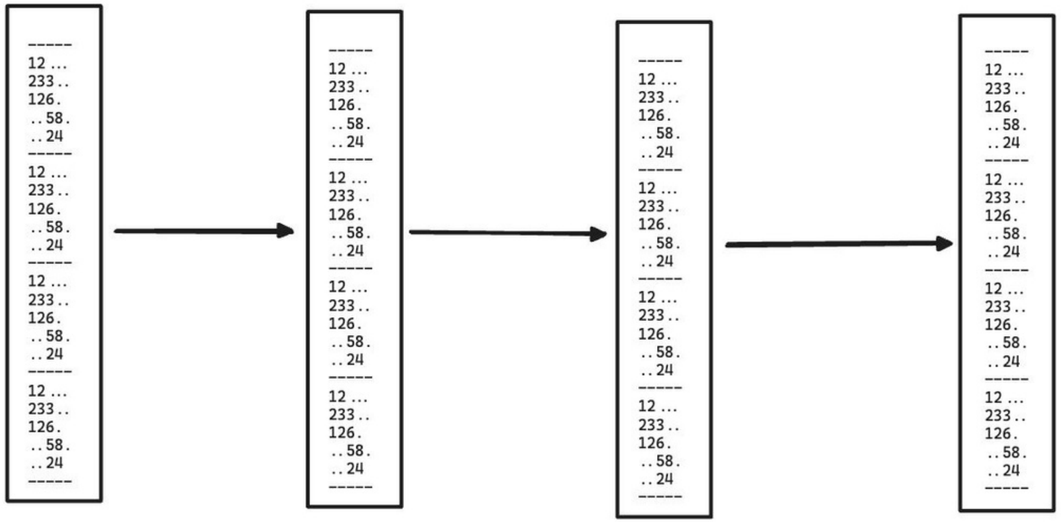
Prefix Layer Tuning

More Existing Solutions ?

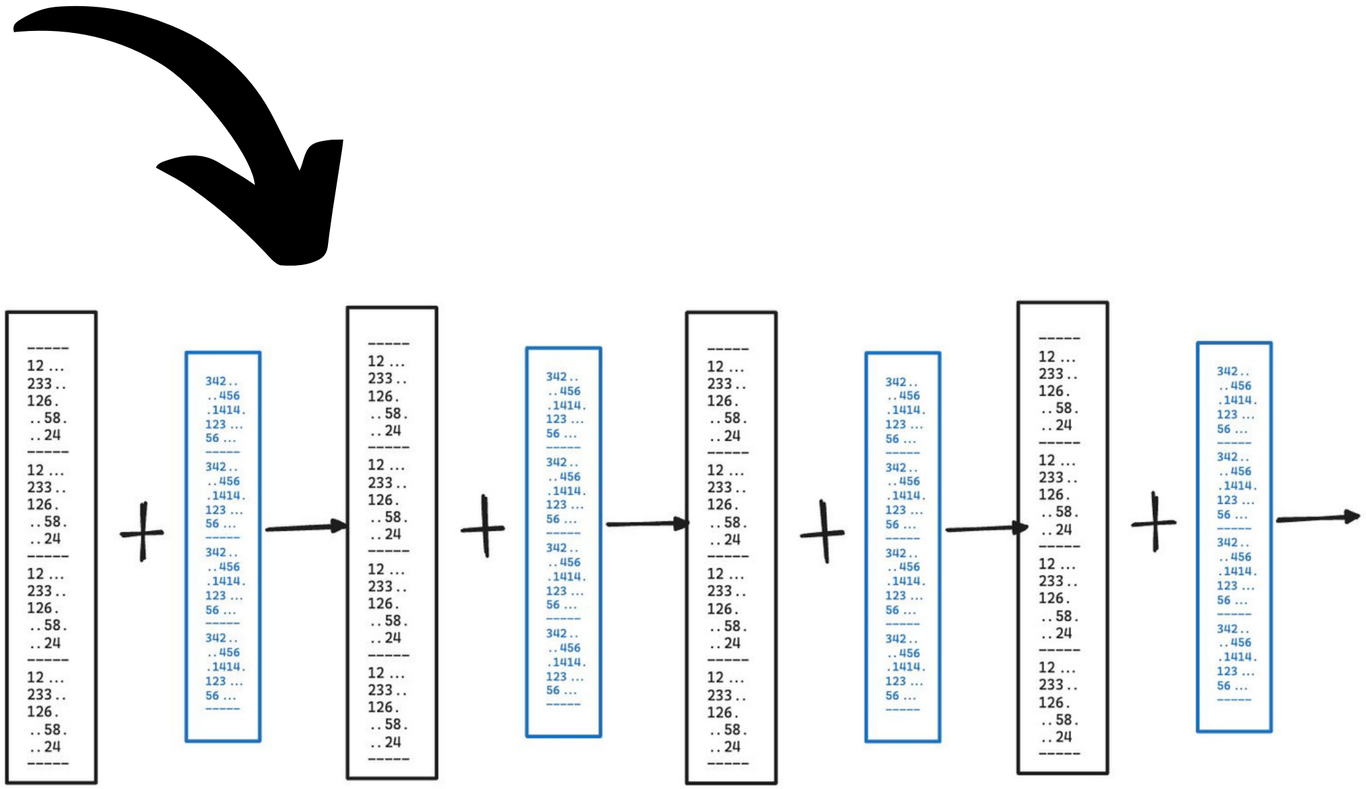
Prefix Layer Tuning



More Existing Solutions ?



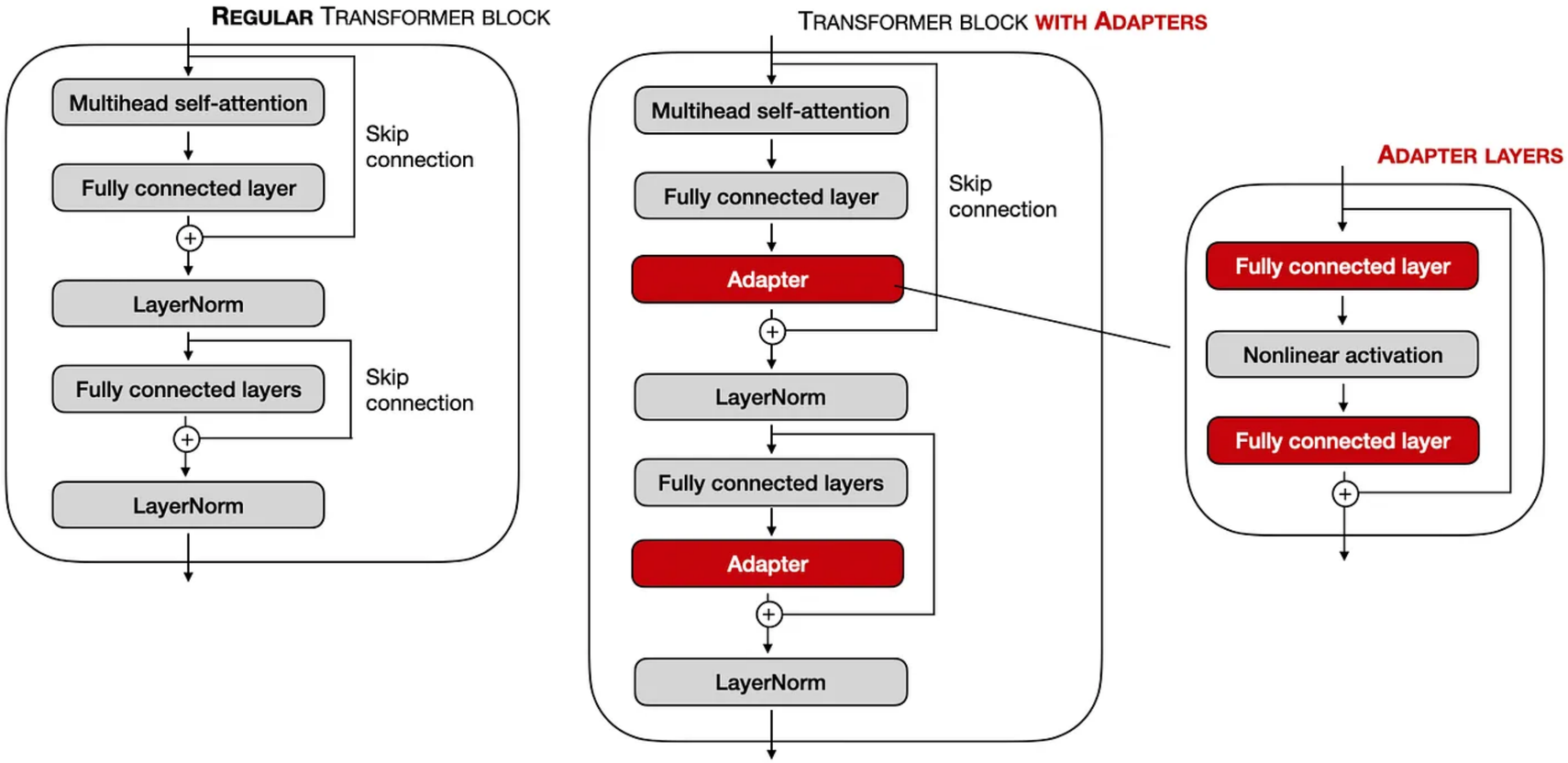
General Hidden Layer



Hidden Layers with Prefix layers for fine tuning

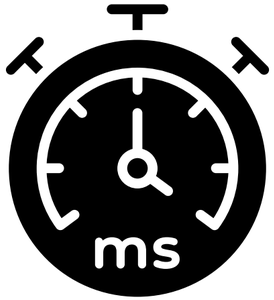
More Existing Solutions ?

- **Adapter tuning**
  - Compared to Prefix Layer tuning -> **prepends tunable tensors to the embeddings**
  - Adapter Tuning -> **adds adapter layers in two places**



Existing Solutions are not good enough

- **Adapter Layers Introduce Inference Latency**



- **Prefix-embedding tuning and Prefix-layer tuning**
  - Directly Optimizing the Prompt is Hard
  - Prefix Embedding adds “instructions” to the prompt so the model can understand it better

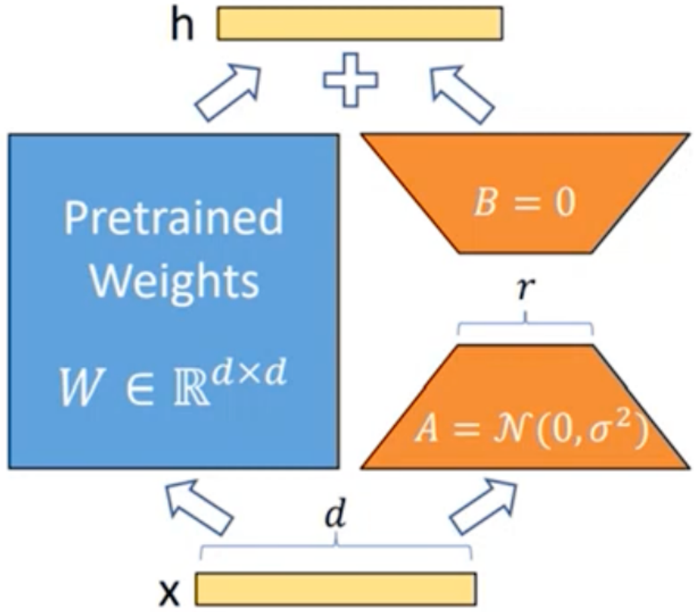
  - Effectiveness of the prefixes can depend significantly on the internal architecture and design of each layer in the transformer model - limiting flexibility

LoRA in working

Contribution – What is LoRA ?

Low-Rank Adaptation

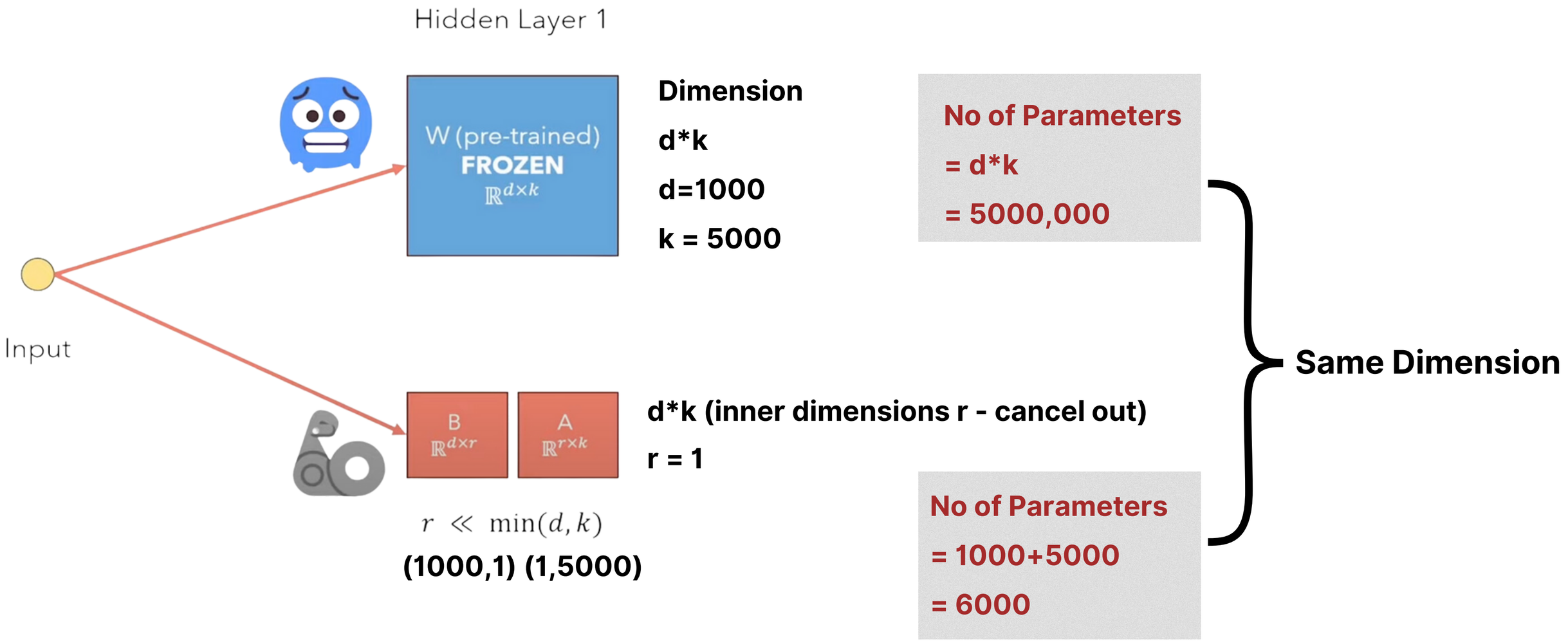
- **Freeze the pre-trained model weights** and inject trainable rank decomposition matrices into each layer.
  - Reparametrization: **Train A and B only.**
- GPT-3 175B finetuning: LORA reduces trainable parameters by 10,000 times and the GPU memory requirement by 3 times.
- Performs on par or better than finetuning in model quality on ROBERTA, DeBERTA, GPT-2, and GPT-3.



That was too technical and quite the high level idea

Lets break it down

Contribution – What is LoRA ?

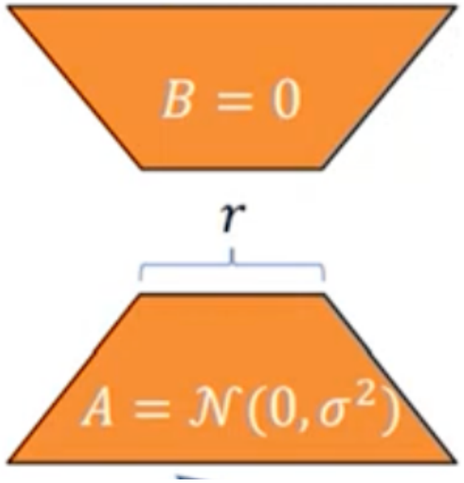


Contribution – What is LoRA ?

Finetuned Weights LoRA

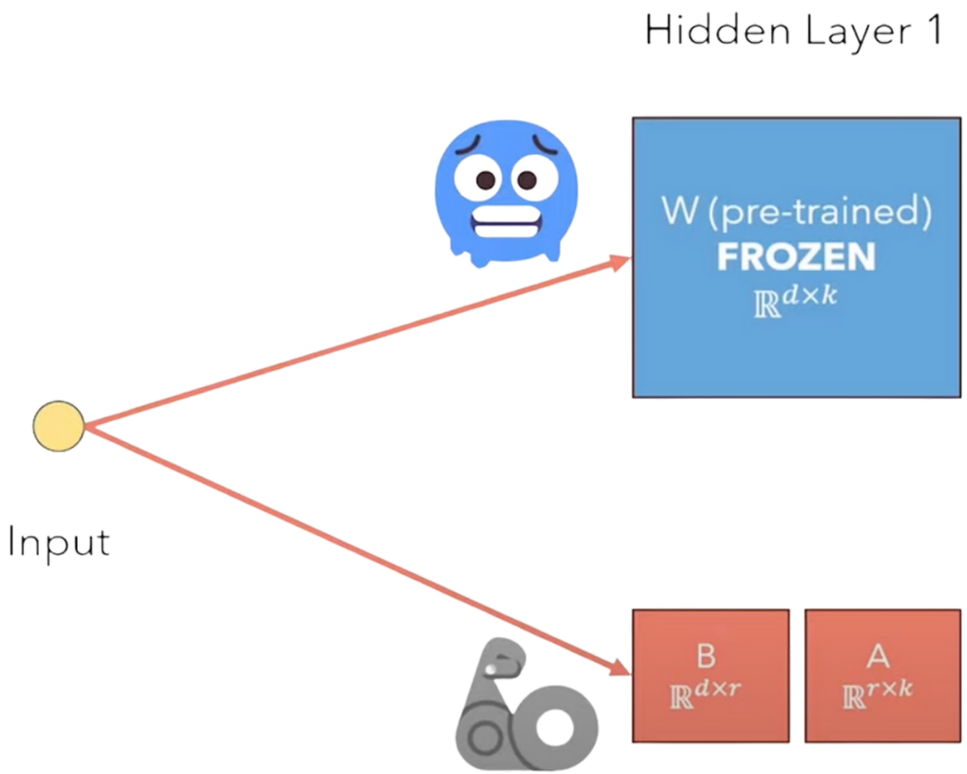
$$\begin{bmatrix} \Delta w_{11} & \Delta w_{12} & \Delta w_{13} \\ \Delta w_{21} & \Delta w_{22} & \Delta w_{23} \\ \Delta w_{31} & \Delta w_{32} & \Delta w_{33} \end{bmatrix} = \underset{(d,k)\ (3,3)}{\Delta W}$$

$$\underset{(d\times r)}{B}\underset{(r\times k)}{A} = \underset{(3\times 1)}{\begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}} \cdot \underset{(1\times 3)}{\begin{pmatrix} a_1 & a_2 & a_3 \end{pmatrix}}$$



Contribution – What is LoRA ?

Dimensionality Reduction



Dimension

d*k

d=1000

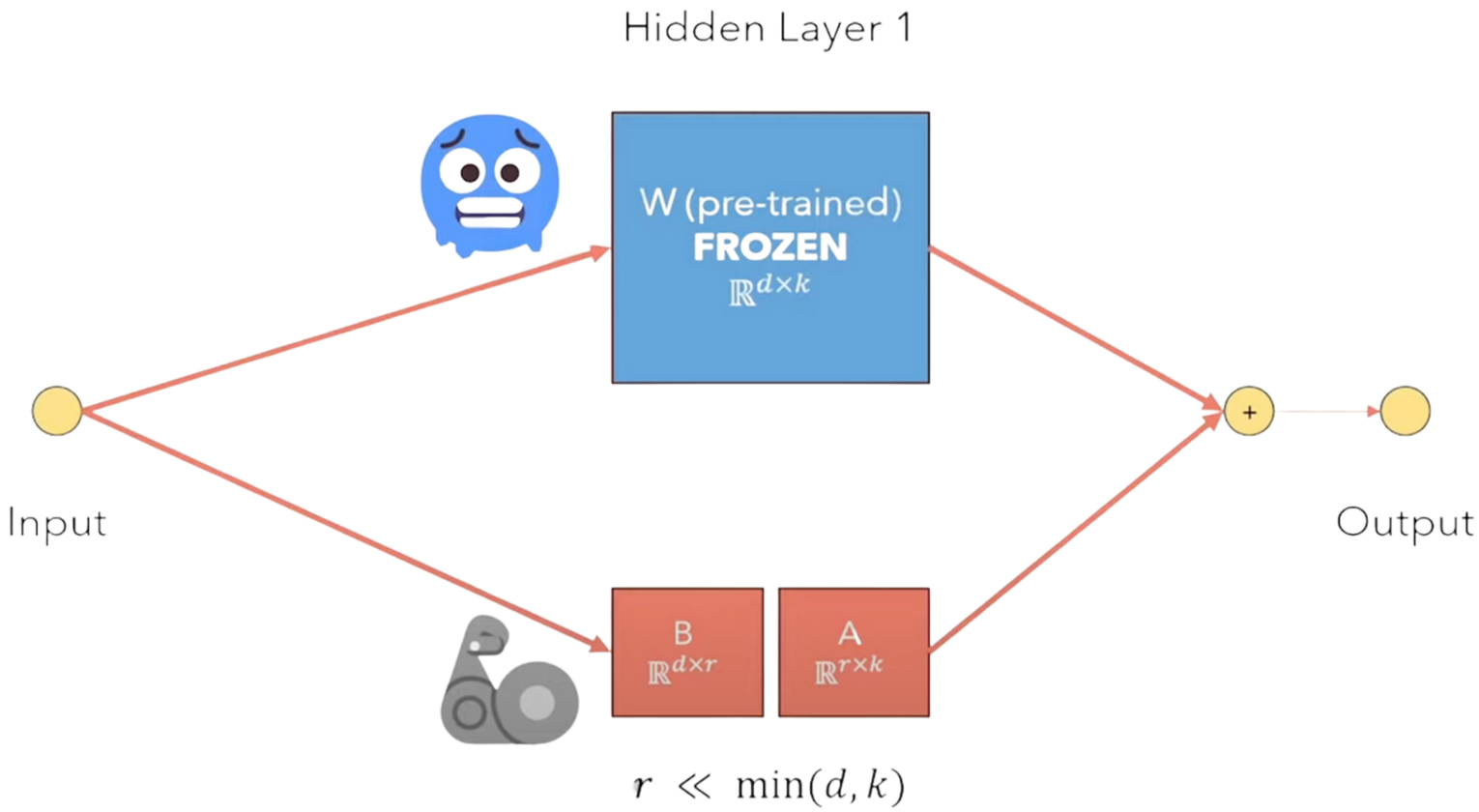
k = 5000

d*k (inner dimensions r - cancel out)

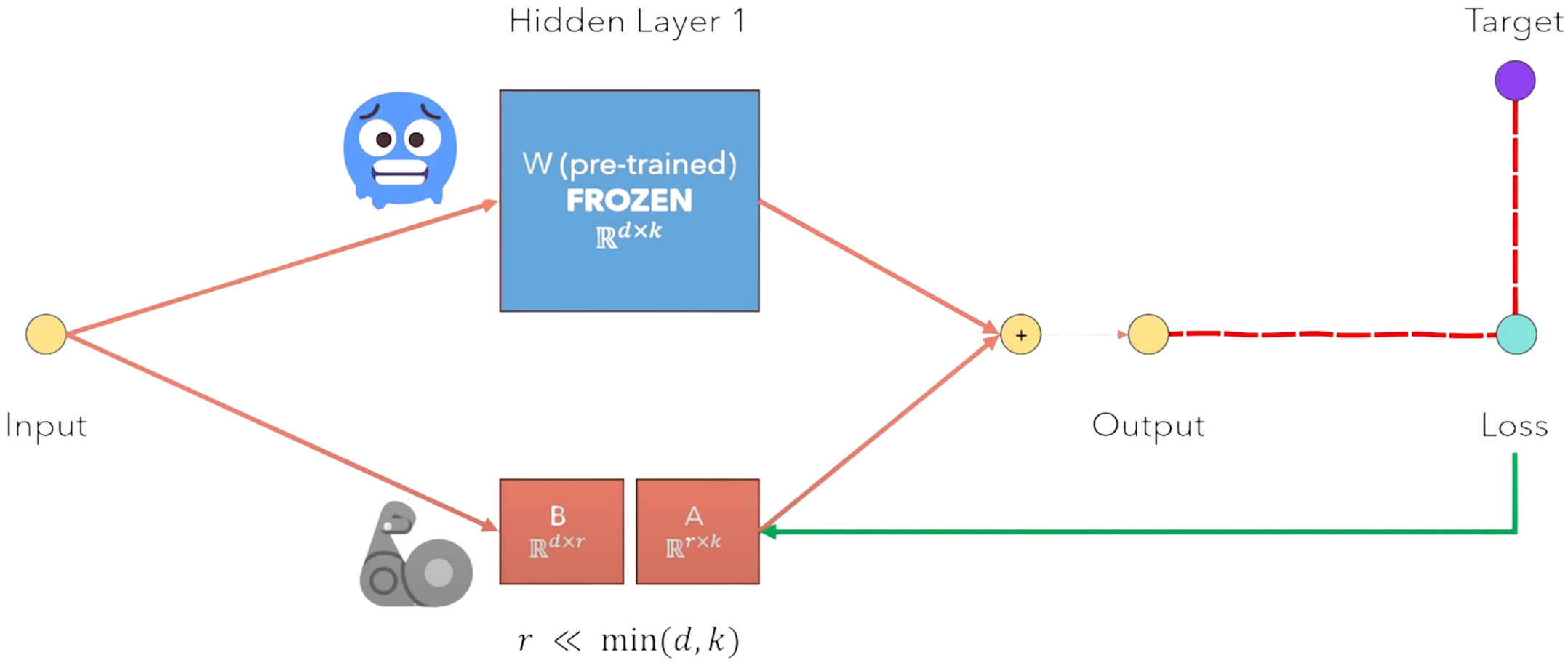
r = 1

(1000,1) (1,5000)

Contribution – What is LoRA ?



Contribution – What is LoRA ?



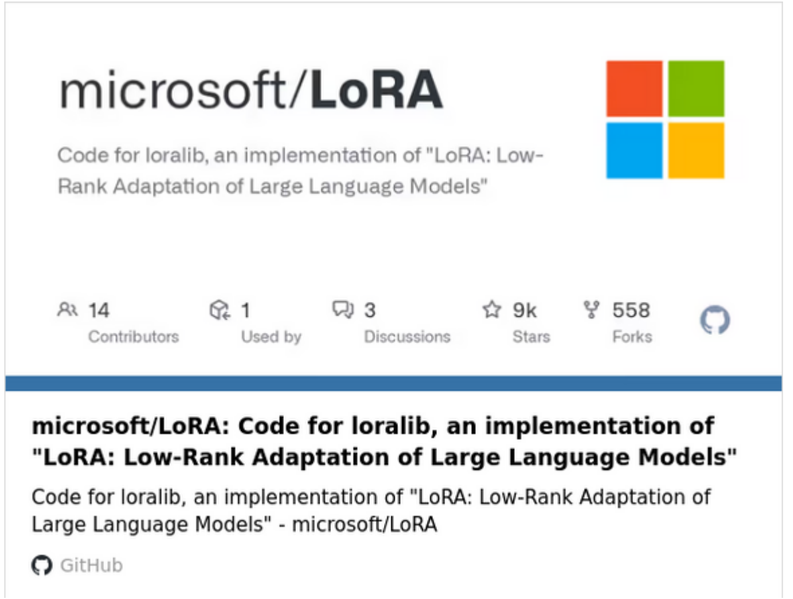
Contribution – What is LoRA ?

- GPT-3 175B: a very low rank (i.e., r=1 or r=2) suffices even when the full rank (i.e., d) is as high as 12,288.

- If r=d, LORA → finetuning

- **LoRA** is very popular for finetuning LLMs like instruction-based models, including **Alpaca and Vicuna**

- **Used to tune stable diffusion** to adapt the style of generated images.



Advantages of LoRA

- **Rank-decomposition matrices have significantly fewer parameters** than the original model, meaning trained LORA weights are easily portable.
- A pre-trained model can be shared to build many **small LoRA modules for different tasks.**
- Previous **pre-trained weights are kept frozen** so the model is less prone to catastrophic forgetting.
- LORA makes training more efficient by up to **3 times** since we do not need to calculate the gradients or maintain the optimizer states for most parameters.
- LORA is orthogonal to many prior methods and can be combined with many of them, such as prefix-tuning

How can we apply LORA to Transformers?

- Let $\mathbf{W}_0$ be a pretrained weight matrix and $\Delta\mathbf{W}$ be its accumulated gradient update during adaptation.
- For $h = \mathbf{W}_0\mathbf{x}$, modified forward pass yields:
  $h = \mathbf{W}_0\mathbf{x} + \alpha\Delta\mathbf{W}\mathbf{x} = \mathbf{W}_0\mathbf{x} + \alpha\mathbf{B}\mathbf{A}\mathbf{x}$.

  α is merging ratio.
- Random Gaussian initialization for $\mathbf{A}$ and zero for $\mathbf{B}$, so $\Delta\mathbf{W} = \mathbf{B}\mathbf{A}$ is zero at the beginning of training.
- Transformer: four weight matrices in the self-attention module (W_q, W_k, W_v, W_o) and two in the MLP module.

How can we apply LORA to Transformers?

- Adapt only attention weights; freeze the MLP modules.
- GPT-3 175B: Reduce VRAM consumption during training from 1.2TB to 350GB. With $r = 4$ and only W_q and W_v being adapted, the checkpoint size is reduced by $\sim 10,000\times$ (from 350GB to 35MB).
- Storing 100 adapted models only requires $350\text{GB} + 35\text{MB} \times 100 \approx 354\text{GB}$ as opposed to $100 \times 350\text{GB} \approx 35\text{TB}$.
- 25% speedup during training on GPT-3 175B compared to full fine-tuning as we do not need to calculate the gradient for the vast majority of the parameters.

Experimental Findings

How does LoRA perform for RoBERTa, DeBERTa and GPT – 3

Model & Method	# Trainable Parameters	MNLI	SST-2	MRPC	CoLA	QNLI	QQP	RTE	STS-B	Avg.
RoB_{base} (FT)*	125.0M	**87.6**	94.8	90.2	**63.6**	92.8	**91.9**	78.7	91.2	86.4
RoB_{base} (BitFit)*	0.1M	84.7	93.7	**92.7**	62.0	91.8	84.0	81.5	90.8	85.2
RoB_{base} (Adpt^{D})*	0.3M	$87.1_{\pm.0}$	$94.2_{\pm.1}$	$88.5_{\pm1.1}$	$60.8_{\pm.4}$	$93.1_{\pm.1}$	$90.2_{\pm.0}$	$71.5_{\pm2.7}$	$89.7_{\pm.3}$	84.4
RoB_{base} (Adpt^{D})*	0.9M	$87.3_{\pm.1}$	$94.7_{\pm.3}$	$88.4_{\pm.1}$	$62.6_{\pm.9}$	$93.0_{\pm.2}$	$90.6_{\pm.0}$	$75.9_{\pm2.2}$	$90.3_{\pm.1}$	85.4
RoB_{base} (LoRA)	0.3M	$87.5_{\pm.3}$	$\mathbf{95.1}_{\pm.2}$	$89.7_{\pm.7}$	$63.4_{\pm1.2}$	$\mathbf{93.3}_{\pm.3}$	$90.8_{\pm.1}$	$\mathbf{86.6}_{\pm.7}$	$\mathbf{91.5}_{\pm.2}$	**87.2**
$\text{RoB}_{\text{large}}$ (FT)*	355.0M	90.2	**96.4**	**90.9**	68.0	94.7	**92.2**	86.6	92.4	88.9
$\text{RoB}_{\text{large}}$ (LoRA)	0.8M	$\mathbf{90.6}_{\pm.2}$	$96.2_{\pm.5}$	$\mathbf{90.9}_{\pm1.2}$	$\mathbf{68.2}_{\pm1.9}$	$\mathbf{94.9}_{\pm.3}$	$91.6_{\pm.1}$	$\mathbf{87.4}_{\pm2.5}$	$\mathbf{92.6}_{\pm.2}$	**89.0**
$\text{RoB}_{\text{large}}$ (Adpt^{P})†	3.0M	$90.2_{\pm.3}$	$96.1_{\pm.3}$	$90.2_{\pm.7}$	$\mathbf{68.3}_{\pm1.0}$	$\mathbf{94.8}_{\pm.2}$	$\mathbf{91.9}_{\pm.1}$	$83.8_{\pm2.9}$	$92.1_{\pm.7}$	88.4
$\text{RoB}_{\text{large}}$ (Adpt^{P})†	0.8M	$\mathbf{90.5}_{\pm.3}$	$\mathbf{96.6}_{\pm.2}$	$89.7_{\pm1.2}$	$67.8_{\pm2.5}$	$\mathbf{94.8}_{\pm.3}$	$91.7_{\pm.2}$	$80.1_{\pm2.9}$	$91.9_{\pm.4}$	87.9
$\text{RoB}_{\text{large}}$ (Adpt^{H})†	6.0M	$89.9_{\pm.5}$	$96.2_{\pm.3}$	$88.7_{\pm2.9}$	$66.5_{\pm4.4}$	$94.7_{\pm.2}$	$92.1_{\pm.1}$	$83.4_{\pm1.1}$	$91.0_{\pm1.7}$	87.8
$\text{RoB}_{\text{large}}$ (Adpt^{H})†	0.8M	$90.3_{\pm.3}$	$96.3_{\pm.5}$	$87.7_{\pm1.7}$	$66.3_{\pm2.0}$	$94.7_{\pm.2}$	$91.5_{\pm.1}$	$72.9_{\pm2.9}$	$91.5_{\pm.5}$	86.4
$\text{RoB}_{\text{large}}$ (LoRA)†	0.8M	$\mathbf{90.6}_{\pm.2}$	$96.2_{\pm.5}$	$\mathbf{90.2}_{\pm1.0}$	$68.2_{\pm1.9}$	$\mathbf{94.8}_{\pm.3}$	$91.6_{\pm.2}$	$\mathbf{85.2}_{\pm1.1}$	$\mathbf{92.3}_{\pm.5}$	**88.6**
DeB_{XXL} (FT)*	1500.0M	91.8	**97.2**	92.0	72.0	**96.0**	92.7	93.9	92.9	91.1
DeB_{XXL} (LoRA)	4.7M	$\mathbf{91.9}_{\pm.2}$	$96.9_{\pm.2}$	$\mathbf{92.6}_{\pm.6}$	$\mathbf{72.4}_{\pm1.1}$	$\mathbf{96.0}_{\pm.1}$	$\mathbf{92.9}_{\pm.1}$	$\mathbf{94.9}_{\pm.4}$	$\mathbf{93.0}_{\pm.2}$	**91.3**

How does LoRA perform for RoBERTa, DeBERTa and GPT - 3

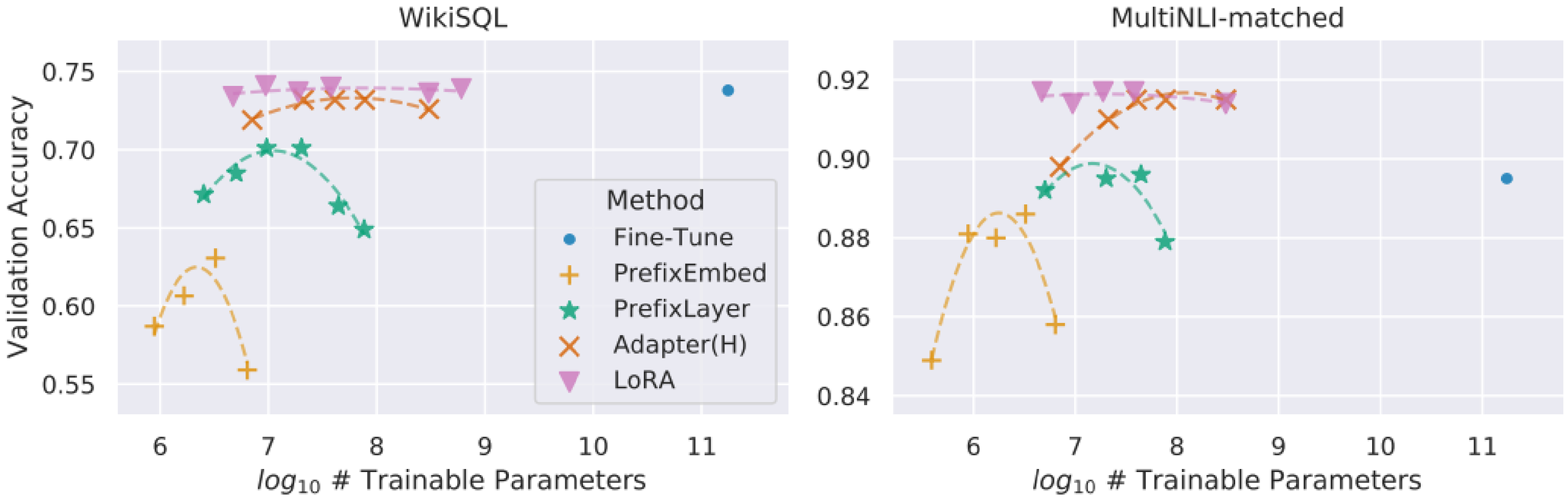
Model & Method	# Trainable Parameters	E2E NLG Challenge				
		BLEU	NIST	MET	ROUGE-L	CIDEr
GPT-2 M (FT)*	354.92M	68.2	8.62	46.2	71.0	2.47
GPT-2 M (Adapter^{L})*	0.37M	66.3	8.41	45.0	69.8	2.40
GPT-2 M (Adapter^{L})*	11.09M	68.9	8.71	46.1	71.3	2.47
GPT-2 M (Adapter^{H})	11.09M	$67.3_{\pm.6}$	$8.50_{\pm.07}$	$46.0_{\pm.2}$	$70.7_{\pm.2}$	$2.44_{\pm.01}$
GPT-2 M (FT^{Top2})*	25.19M	68.1	8.59	46.0	70.8	2.41
GPT-2 M (PreLayer)*	0.35M	69.7	8.81	46.1	71.4	2.49
→ GPT-2 M (LoRA)	0.35M	$\mathbf{70.4}_{\pm.1}$	$\mathbf{8.85}_{\pm.02}$	$\mathbf{46.8}_{\pm.2}$	$\mathbf{71.8}_{\pm.1}$	$\mathbf{2.53}_{\pm.02}$
GPT-2 L (FT)*	774.03M	68.5	8.78	46.0	69.9	2.45
GPT-2 L (Adapter^{L})	0.88M	$69.1_{\pm.1}$	$8.68_{\pm.03}$	$46.3_{\pm.0}$	$71.4_{\pm.2}$	$\mathbf{2.49}_{\pm.0}$
GPT-2 L (Adapter^{L})	23.00M	$68.9_{\pm.3}$	$8.70_{\pm.04}$	$46.1_{\pm.1}$	$71.3_{\pm.2}$	$2.45_{\pm.02}$
GPT-2 L (PreLayer)*	0.77M	70.3	8.85	46.2	71.7	2.47
→ GPT-2 L (LoRA)	0.77M	$\mathbf{70.4}_{\pm.1}$	$\mathbf{8.89}_{\pm.02}$	$\mathbf{46.8}_{\pm.2}$	$\mathbf{72.0}_{\pm.2}$	$2.47_{\pm.02}$

How does LoRA perform for RoBERTa, DeBERTa and GPT - 3

Model&Method	# Trainable Parameters	WikiSQL Acc. (%)	MNLI-m Acc. (%)	SAMSum R1/R2/RL
GPT-3 (FT)	175,255.8M	**73.8**	89.5	52.0/28.0/44.5
GPT-3 (BitFit)	14.2M	71.3	91.0	51.3/27.4/43.5
GPT-3 (PreEmbed)	3.2M	63.1	88.6	48.3/24.2/40.5
GPT-3 (PreLayer)	20.2M	70.1	89.5	50.8/27.3/43.5
GPT-3 ($Adapter^H$)	7.1M	71.9	89.8	53.0/28.9/44.8
GPT-3 ($Adapter^H$)	40.1M	73.2	**91.5**	53.2/29.0/45.1
GPT-3 (LoRA)	4.7M	73.4	**91.7**	**53.8/29.8/45.9**
GPT-3 (LoRA)	37.7M	**74.0**	**91.6**	53.4/29.2/45.1

Understanding Low-rank updates done by LORA

- **LoRA exhibits better scalability and task performance; not all methods benefit monotonically from having more trainable parameters.**



Understanding Low-rank updates done by LORA

- **Which weight matrices should we apply LORA to?**
  - Apply LoRA to Wq and Wv (Query and Value weights) for better performance, as indicated by the higher validation accuracies in both WikiSQL and MultiNLI tasks.
  - Avoid applying LoRA to all weight matrices Wq, Wk, Wv, and Wo simultaneously, as it does not significantly outperform selective application and might increase complexity.

	# 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 ($\pm 0.5\%$)	70.4	70.0	73.0	73.2	71.4	**73.7**	**73.7**
MultiNLI ($\pm 0.1\%$)	91.0	90.8	91.0	91.3	91.3	91.3	**91.7**

What is the optimal rank r?

- LoRA performs competitively with a very small r=1. This suggests the update matrix ΔW could have a very small "intrinsic rank".
- Increasing r does not cover a more meaningful subspace, which suggests that a low-rank adaptation matrix is sufficient.

	Weight Type	$r = 1$	$r = 2$	$r = 4$	$r = 8$	$r = 64$
WikiSQL(±0.5%)	W_q	68.8	69.6	70.5	70.4	70.0
	W_q, W_v	73.4	73.3	73.7	73.8	73.5
	W_q, W_k, W_v, W_o	74.1	73.7	74.0	74.0	73.9
MultiNLI (±0.1%)	W_q	90.7	90.9	91.1	90.7	90.7
	W_q, W_v	91.3	91.4	91.3	91.6	91.4
	W_q, W_k, W_v, W_o	91.2	91.7	91.7	91.5	91.4

Limitations of LoRA

Batching Complexity

- LoRA is not straightforward for batching inputs to different tasks within a single forward pass if one chooses to absorb the adaptation matrices into the weight matrix to eliminate additional inference latency

Batch Inference Latency

- Adapter layers introduced by LoRA can add extra compute time, which is not ideal for online inference settings where batch size is typically small.

Future Scope

- **Combining LoRA with Other Adaptation Methods**
  - Exploring how LoRA can be integrated with other efficient adaptation strategies to provide orthogonal improvements potentially

- **Expanding LoRA's Application Beyond Language Models**
  - Applying the principles of LoRA to other neural networks with dense layers to see if similar efficiency gains can be achieved

Conclusion

- LoRA stands for Low-Rank Adaptation, which is a strategy designed to efficiently adapt large pre-trained models like language models without compromising their performance

- It is an efficient adaptation strategy that maintains high model quality while ensuring there is no increase in inference latency or reduction in the input sequence length

- It emphasizes the importance of parameter-efficient strategies, which allow for significant improvements without the need for extensive additional parameters.

Related Work on Neural Network Stability and Generalization

Adapter Layers: The paper discusses the addition of adapter layers as a strategy for efficient adaptations, citing works by Houlsby et al. (2019), Rebuffi et al. (2017), Pfeiffer et al. (2021), and Rücklé et al. (2020).

Input Layer Activations: It also mentions optimizing forms of the input layer activations, with references to Li & Liang (2021), Lester et al. (2021), Hambardzumyan et al. (2020), and Liu et al. (2021).

Low-Rank Structures: The significance of low-rank structures is highlighted, referencing prior works that impose a low-rank constraint when training neural networks, such as those by Sainath et al. (2013), Povey et al. (2018), Zhang et al. (2014), Jaderberg et al. (2014), Zhao et al. (2016), Khodak et al. (2021), and Denil et al. (2014).

Neural Networks and Low-Rank Structure: The paper also touches on theoretical literature that discusses the performance of neural networks with low-rank structures, citing Allen-Zhu et al. (2019), Li & Liang (2018), Ghorbani et al. (2020), Allen-Zhu & Li (2019), and Allen-Zhu & Li (2020a).

Thank you and Q/A

- **What is the problem?**
  - The problem addressed is the inefficiency of fully fine-tuning large pre-trained language models due to their large parameter size and the associated high computational and storage costs.

- **Why does this paper/project study this problem?**
  - Deploying independent instances of fully fine-tuned models, each with an extensive number of parameters, becomes prohibitively expensive as models grow larger

- **What are the major contributions of the paper?**
  - Introduction of Low-Rank Adaptation (LoRA), which significantly reduces the number of trainable parameters and GPU memory requirements and performs comparably or better than full fine-tuning

- **What are the limitations of the paper?**
  - **Batching Complexity:** LoRA is not straightforward for batching inputs to different tasks within a single forward pass if one chooses to absorb the adaptation matrices into the weight matrix to eliminate additional inference latency