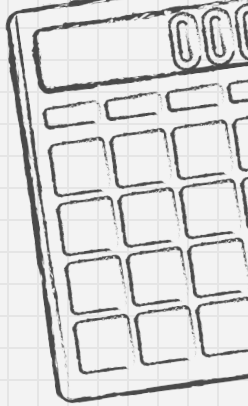
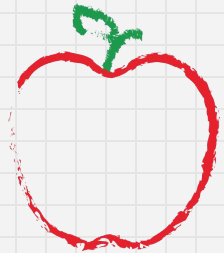
LORA: LOW-RANK ADAPTATION OF LARGE LANGUAGE MODELS

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

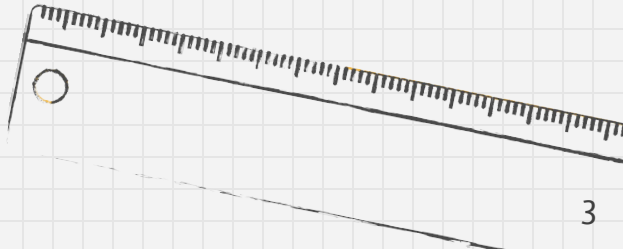
Table of contents



01



Paper discussion



Motivation

- Since pretrained opensource large language models (LLMs) are readily available (and training them from scratch is not possible and pointless when we think about the regular applications). Fine tuning them towards our task should suffice. **BUT** fine tuning them itself becomes impossible due to their large size. Lets see why?
    - During training or fine tuning the amount of VRAM (GPU RAM) taken by the model is considerably higher than the amount of memory it takes during inference.
- If we want fine tune for two different task, then we will get 2 different fine tuned models. In the case of LLMs, storing these models in the disk becomes very cumbersome.
- Once fine tuning happens, a phenomenon known as catastrophic forgetting occurs. This where the model forgets the what it learnt during pre-training and only remembers the task it learnt during fine tuning.

Authors Claims why existing solutions do not scale to LLM paradigm

The authors take language modeling as an example scenario and identify two prominent strategies in efficient adaptation and discuss. The two strategies are,

- Adding adapter layers: Here we insert adapter layers at different positions of the model and train those adapter layers, the authors say by doing so we introduce inference latency.
- Directly Optimizing the Prompt: The authors observed that prefix tuning is difficult to optimize and that its performance changes non monotonically in trainable parameters.

Advantage of LoRA

- A pre-trained model can be shared and used to build many small LoRA modules for different tasks. By doing this task switching overhead is reduced significantly. (we will see the reason later on)
- LoRA makes training more efficient and lowers the hardware barrier to entry by up to 3 times when using adaptive optimizers since we do not need to calculate the gradients or maintain the optimizer states for most parameters. Instead, we only optimize the injected, much smaller low-rank matrices.
- Due to the simple linear design, the trainable matrices can be easily merged with frozen weights during deployment, thereby nullifying inference latency.
- LoRA is orthogonal to many prior methods and can be combined with many of them, such as prefix-tuning

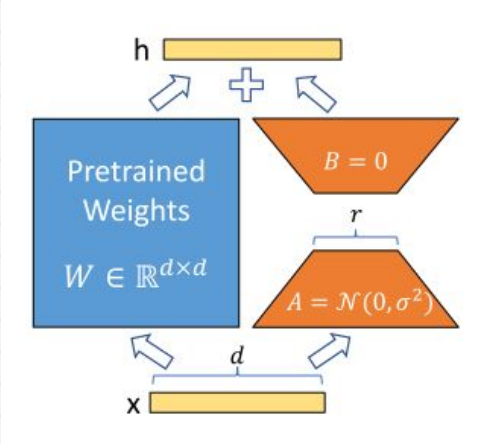
Core Hypothesis of this work

- Authors take inspiration from the work of [1] where the paper claims that pre-trained language models have a low intrinsic dimension when adapting to a specific task.
- Taking this as inspiration the authors of LoRA hypothesized that the **updates to the weights are of low intrinsic ranks**.

Proposed Solution

$$h = W_0 x + \Delta W x = W_0 x + BAx$$

h is the output vector of a layer.
Wo is pretrained weight of the layer (frozen, no learning allowed). Dimension d x k
delta_W is the change in weight during fine tuning. Dimension d x k
x is the input vector to that layer. Dimension d x 1
B, A will be our low rank matrices which approximate **delta_W.** Dim of B d x r and Dim of A r x k. Where r is rank.



Applying LoRA to Transformers

Theoretically LoRA can can be applied to any weight matrix in a model, but the authors study on applying on Transformers only to the self attention module.

number of trainable parameters. In the Transformer architecture, there are four weight matrices in the self-attention module (W_q, W_k, W_v, W_o) and two in the MLP module. We treat W_q (or W_k, W_v) as a single matrix of dimension $d_{model} \times d_{model}$, even though the output dimension is usually sliced into attention heads. We limit our study to **only adapting the attention weights** for downstream tasks and freeze the MLP modules (so they are not trained in downstream tasks) both for simplicity and parameter-efficiency.We further study the effect on adapting different types of attention weight

Results

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**

Table 2: $\text{RoBERTa}_{\text{base}}$, $\text{RoBERTa}_{\text{large}}$, and $\text{DeBERTa}_{\text{XXL}}$ with different adaptation methods on the GLUE benchmark. We report the overall (matched and mismatched) accuracy for MNLI, Matthew's correlation for CoLA, Pearson correlation for STS-B, and accuracy for other tasks. Higher is better for all metrics. * indicates numbers published in prior works. † indicates runs configured in a setup similar to Houlsby et al. (2019) for a fair comparison.

Crucial Results for us

	# 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**

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.

Results show that applying LoRA to the Query layer (Wq) and Value Layer (Wv) should be sufficient.

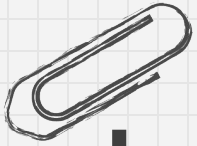
Crucial Results for us (Contd..)

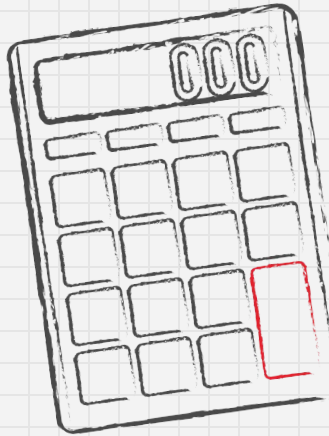
	Weight Type	$r = 1$	$r = 2$	$r = 4$	$r = 8$	$r = 64$
WikiSQL($\pm 0.5\%$)	W_q	68.8	69.6	70.5	70.4	70.0
	W_q, W_v	73.4	73.3	73.7	73.8	73.5
	W_q, W_k, W_v, W_o	74.1	73.7	74.0	74.0	73.9
MultiNLI ($\pm 0.1\%$)	W_q	90.7	90.9	91.1	90.7	90.7
	W_q, W_v	91.3	91.4	91.3	91.6	91.4
	W_q, W_k, W_v, W_o	91.2	91.7	91.7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

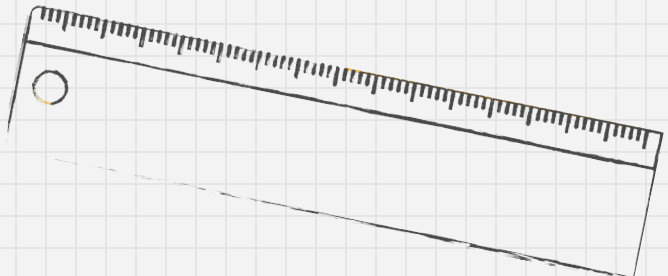
Table 6: Validation accuracy on WikiSQL and MultiNLI with different rank r. To our surprise, a rank as small as one suffices for adapting both W_q and W_v on these datasets while training W_q alone needs a larger r. We conduct a similar experiment on GPT-2 in Section H.2.

02

A little bit of Linear Algebra

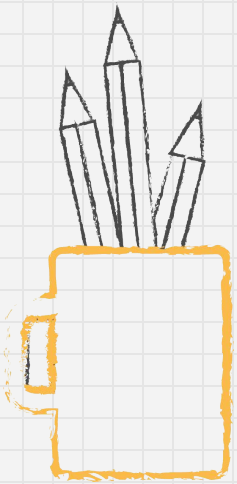


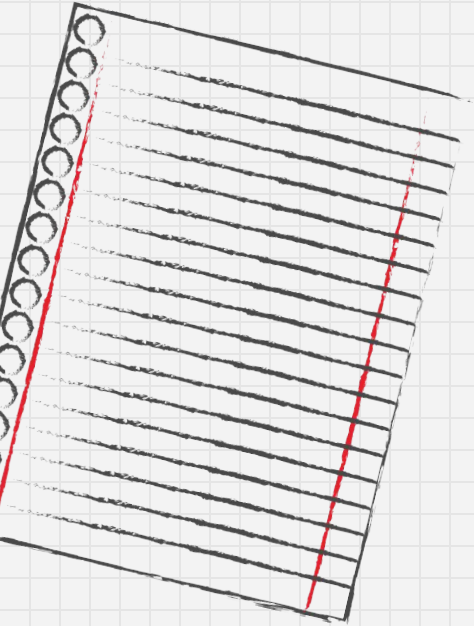




03

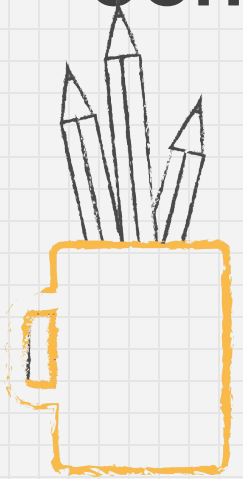
Demo Time

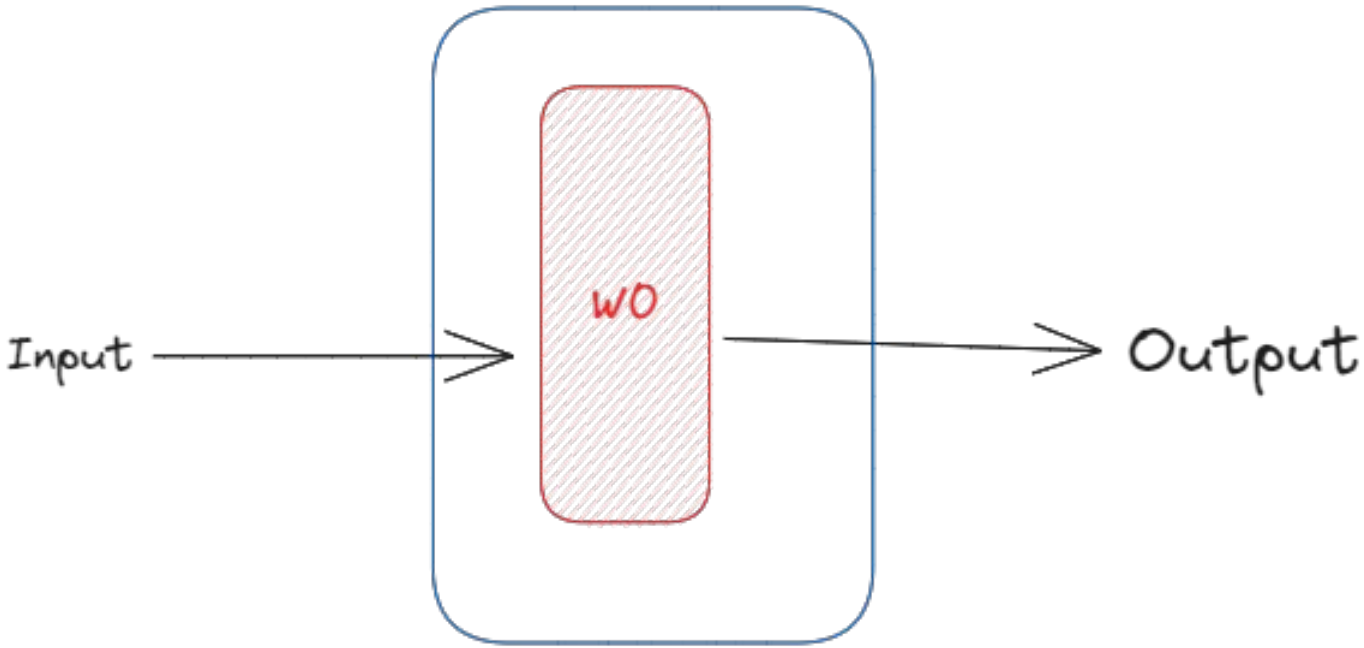


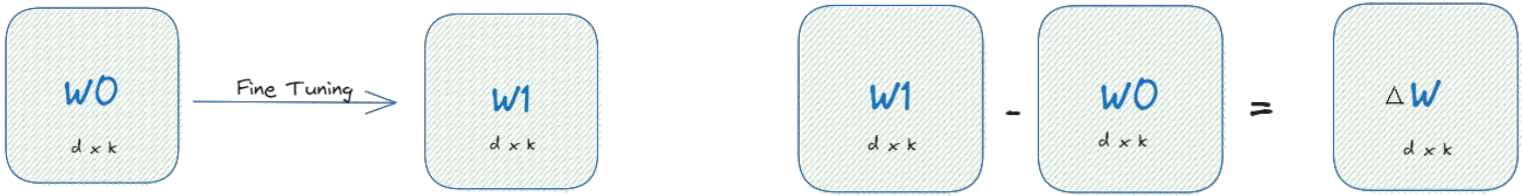


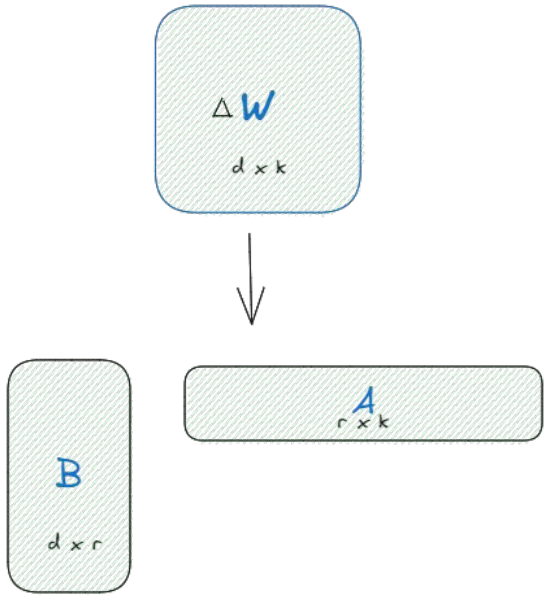
04

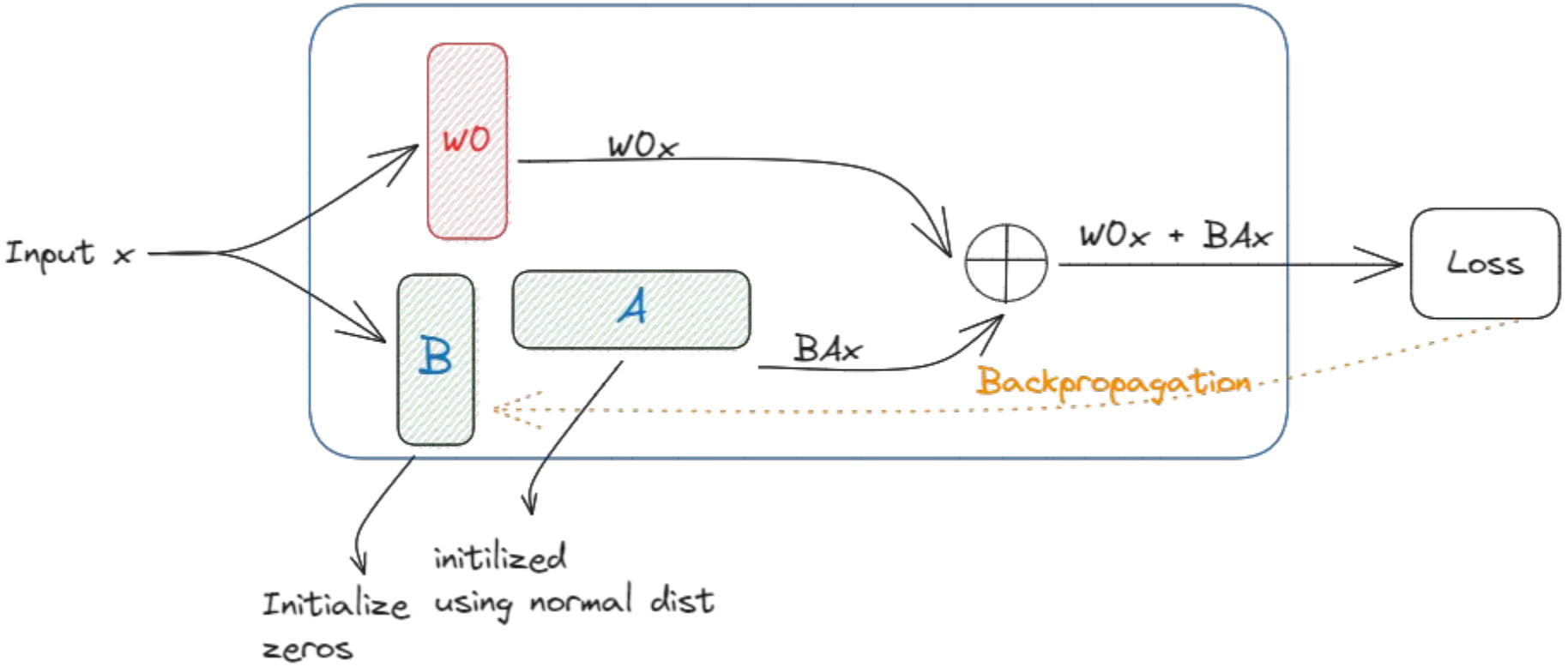
Connect LoRA implementation pictorially




Input
WO
Output


W0
d x k
Fine Tuning
W1
d x k
W1
d x k
-
W0
d x k
=
ΔW
d x k


ΔW
d x k
A
r x k
B
d x r

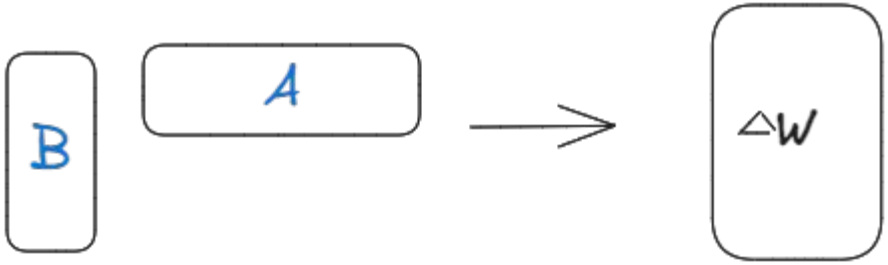


Save B and A for a particular task

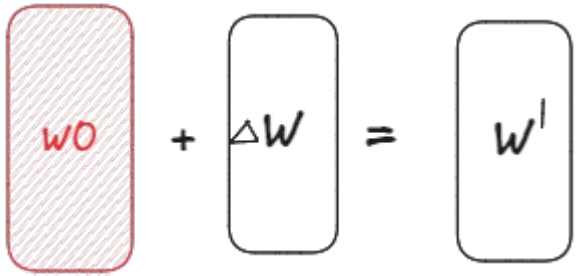
During Inference

Take the stored A, B

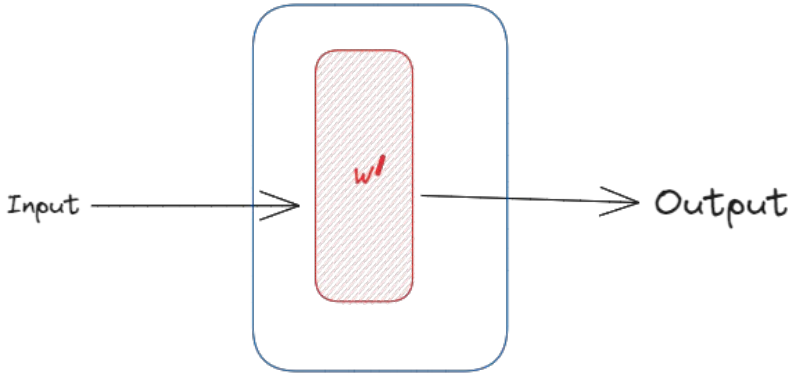
Multiply and create a delta matrix



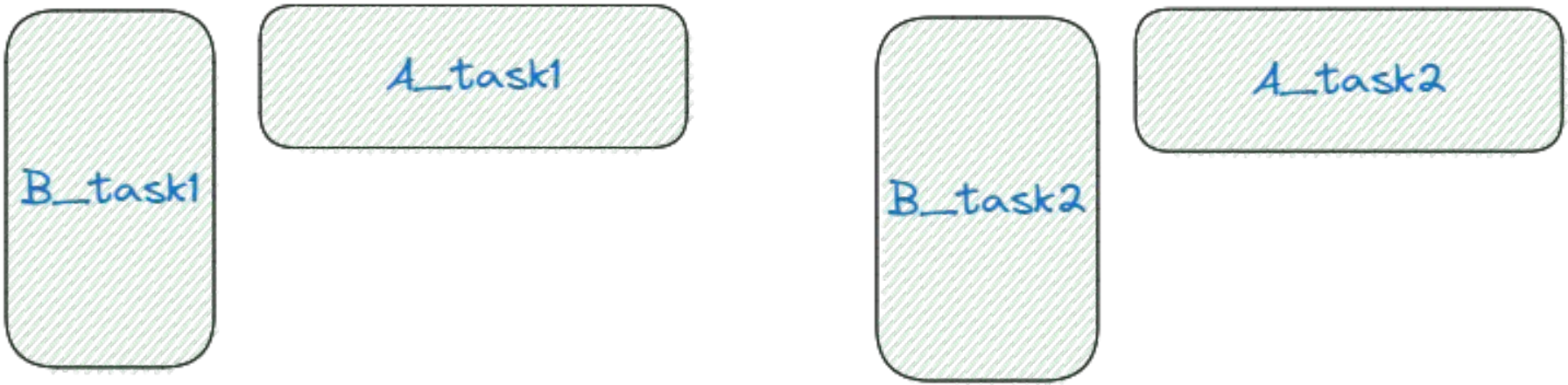
Add the matrix with the frozen weight of the layer and create updated weights for the layer



Finally model is ready for inference



Can store different A, B for different tasks



References

[1] Armen Aghajanyan, Luke Zettlemoyer, and Sonal Gupta. Intrinsic Dimensionality Explains the Effectiveness of Language Model Fine-Tuning. arXiv:2012.13255 [cs], December 2020. URL http://arxiv.org/abs/2012.13255. Zeyuan.