Parameter Efficient Fine Tuning (PEFT)

Pawan Goyal

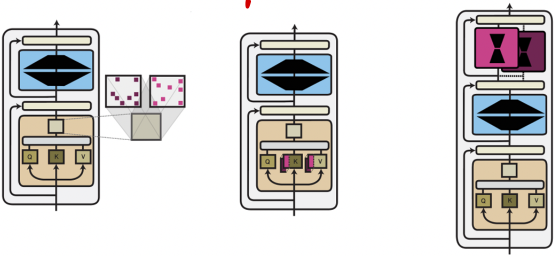
CSE, IIT Kharagpur

CS60010

Different Perspectives to think about PEFT

prefix-tuning
prompt-tuning

LoRA
QLoRA
Pruning



→ adapters

Parameter

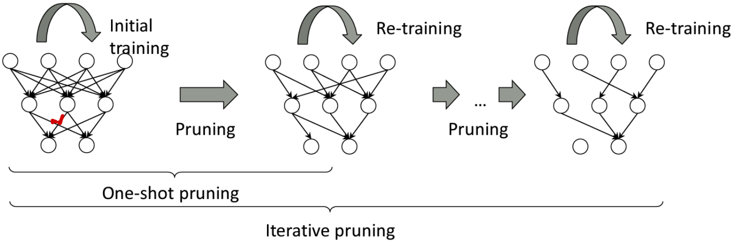
Input

Function

Task-specific

Pruning

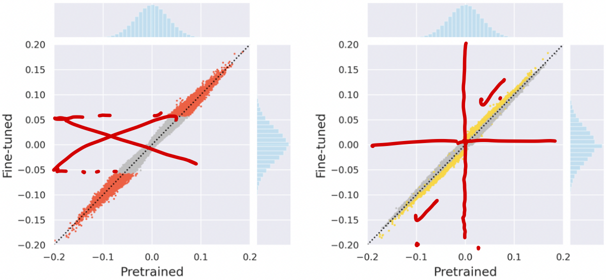
- During pruning, a fraction of the lowest-magnitude weights are removed
- The non-pruned weights are re-trained
- Pruning for multiple iterations is more common (Frankle & Carbin, 2019)



How to apply Pruning for Pre-trained models?

- Pruning does not consider how weights change during fine-tuning
- **Magnitude pruning**: keep weights farthest from 0
- **Movement pruning** [Sanh et al., 2020]: keep weights that *move the most away* from 0

Fine-tuned weights stay close to their their pre-trained values. Magnitude pruning (left) selects weights that are far from 0.



Movement pruning (right) selects weights that move away from 0.

0.004 → 0.005

Revisiting full fine-tuning

- Assume we have a pre-trained autoregressive language model $P_\phi(y|x)$
  - E.g., GPT based on Transformer
- Adapt this pretrained model to downstream tasks (e.g., summarization, NL2SQL, reading comprehension)
  - Training dataset of context-target pairs $\{(x_i, y_i)\}_{i=1,\ldots,N}$
- During full fine-tuning, we update ϕ_o to $\phi_o + \Delta\phi$ by following the gradient to maximize the conditional language modeling objective

$$\max_{\phi} \sum_{(x,y)} \sum_{t=1}^{|y|} \log(P_\phi(y_t|x, y_{<t}))$$

LoRA: Low-rank Adaptation

- For each downstream task, we learn a different set of parameters $\Delta\phi$
  - $|\Delta\phi| = |\phi_o|$
  - GPT-3 has a $|\ \phi_o|$ of 175 billion
  - Expensive and challenging for storing and deploying many independent instances

- Key idea: encode the task-specific parameter increment $\Delta\phi = \Delta\phi(\Theta)$ by a smaller-sized set of parameters Θ, $|\Theta| \ll |\phi_o|$

- The task of finding $\Delta\phi$ becomes optimizing over Θ

$$\max_{\Theta} \sum_{(x,y)} \sum_{t=1}^{|y|} \log(P_{\phi_o + \Delta\phi(\Theta)}(y_t | x, y_{<t}))$$

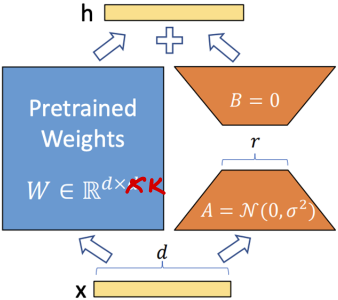
$|\Delta\phi(\Theta)| << |\Delta\phi|$

Low-rank-parameterized update matrices

- Updates to the weights have a low "intrinsic rank" during adaptation (Aghajanyan et al. 2020)
- $W_0 \in \mathbb{R}^{d\times k}$: a pretrained weight matrix
- Constrain its update with a low-rank decomposition:

$$W_0 + \Delta W = W_0 + BA$$

  where $B \in \mathbb{R}^{d\times r}, A \in \mathbb{R}^{r\times k}, r \ll \min(d,k)$
- Only A and B contain **trainable** parameters



$BA: d \times K$

$r \ll \min(d,k)$

$r \times (d+K) \ll d.K$

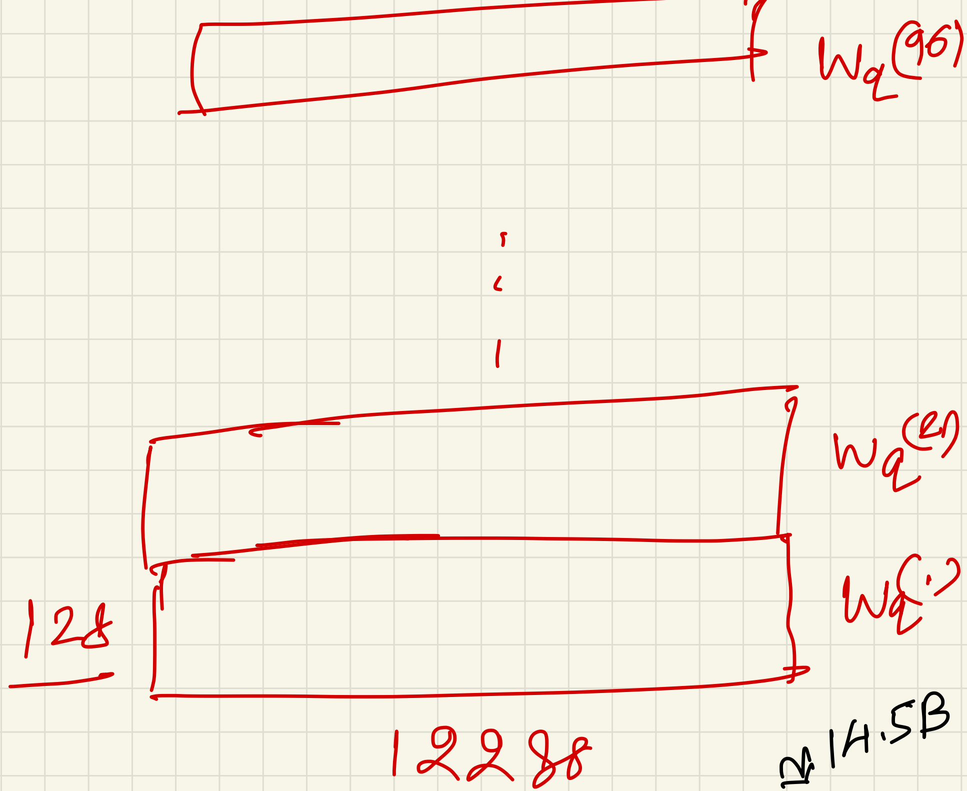
LoRA: What happens during training and inference?

- $W_0 + \Delta W = W_0 + BA$
- During training, W_0 is frozen and does not receive gradient updates.
- For $h = W_0 x$, forward pass would be

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

At inference time?

Use the matrix $W = W_0 + BA$, so no additional latency



$\approx 14.5B$

$$(12288 \times 2) \times 2 \times 96 \approx 4.7m$$

$$W_0 : \underline{(12288 \times 12288)} \times 96$$

$$W_0 + \underline{B} \cdot A$$

$\underline{B}$: $12288 \times \underline{r}$

$\underline{A}$: $r \times 12288$

$\underline{r = 2}$

Which weight matrices to apply to?

Which weight matrices in Transformers should we apply LoRA to?

	# of Trainable Parameters = 18M						
Weight Type Rank r	W_q 8	W_k 8	W_v 8	W_o 8	W_q, W_k 4	W_q, W_v 4	W_q, W_k, W_v, W_o 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**

Adapting both Wq and Wv gives the best performance overall.

What is the optimal rank r for LoRA?

	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

LoRA already performs competitively with a very small r

Understanding LoRA Parameters over GPT-3

- Trainable parameters = $2 \times \hat{L}_{LoRA} \times d_{model} \times r$
- For GPT-3, $d_{model} = 12288$, 96 decoders and 96 attention heads

LoRA	$r_v = 2$	4.7 M
	$r_q = r_v = 1$	4.7 M
	$r_q = r_v = 2$	9.4 M
	$r_q = r_k = r_v = r_o = 1$	9.4 M
	$r_q = r_v = 4$	18.8 M
	$r_q = r_k = r_v = r_o = 2$	18.8 M
	$r_q = r_v = 8$	37.7 M
	$r_q = r_k = r_v = r_o = 4$	37.7 M
	$r_q = r_v = 64$	301.9 M
	$r_q = r_k = r_v = r_o = 64$	603.8 M

Applying LoRA to Transformer

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	$\mathbf{70.4}_{\pm.1}$	$\mathbf{8.89}_{\pm.02}$	$\mathbf{46.8}_{\pm.2}$	$\mathbf{72.0}_{\pm.2}$	$2.47_{\pm.02}$

GPT-2 medium (M) and large (L) with different adaptation methods on the E2E NLG Challenge. For all metrics, higher is better. LoRA outperforms several baselines with comparable or fewer trainable parameters