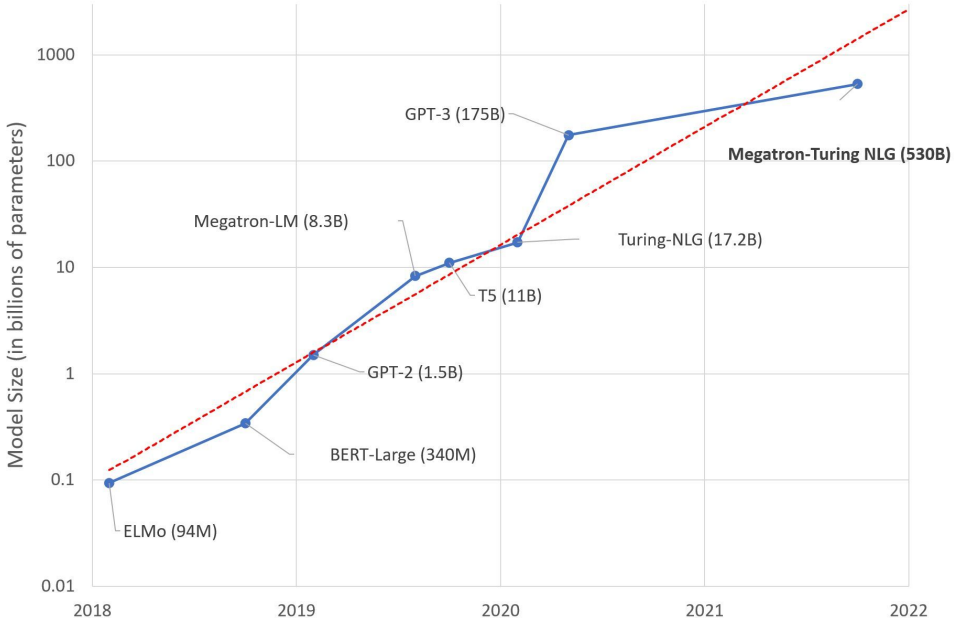
LoRA, QLoRA and beyond!

Debarshi Deka
Satya Sai Srinath

CS 839 Presentation

Motivation

Moore’s law for Deep Learning!



Source: https://huggingface.co/blog/large-language-models

Motivation

LLMs becoming good as zero/few shot learners

Language Models are Unsupervised Multitask Learners

Alec Radford [*1] Jeffrey Wu [*1] Rewon Child [1] David Luan [1] Dario Amodei [**1] Ilya Sutskever [**1]

Language Models are Few-Shot Learners

Tom B. Brown* Benjamin Mann* Nick Ryder* Melanie Subbiah*
Jared Kaplan† Prafulla Dhariwal Arvind Neelakantan Pranav Shyam Girish Sastry
Amanda Askell Sandhini Agarwal Ariel Herbert-Voss Gretchen Krueger Tom Henighan
Rewon Child Aditya Ramesh Daniel M. Ziegler Jeffrey Wu Clemens Winter
Christopher Hesse Mark Chen Eric Sigler Mateusz Litwin Scott Gray
Benjamin Chess Jack Clark Christopher Berner
Sam McCandlish Alec Radford Ilya Sutskever Dario Amodei

OpenAI

Motivation

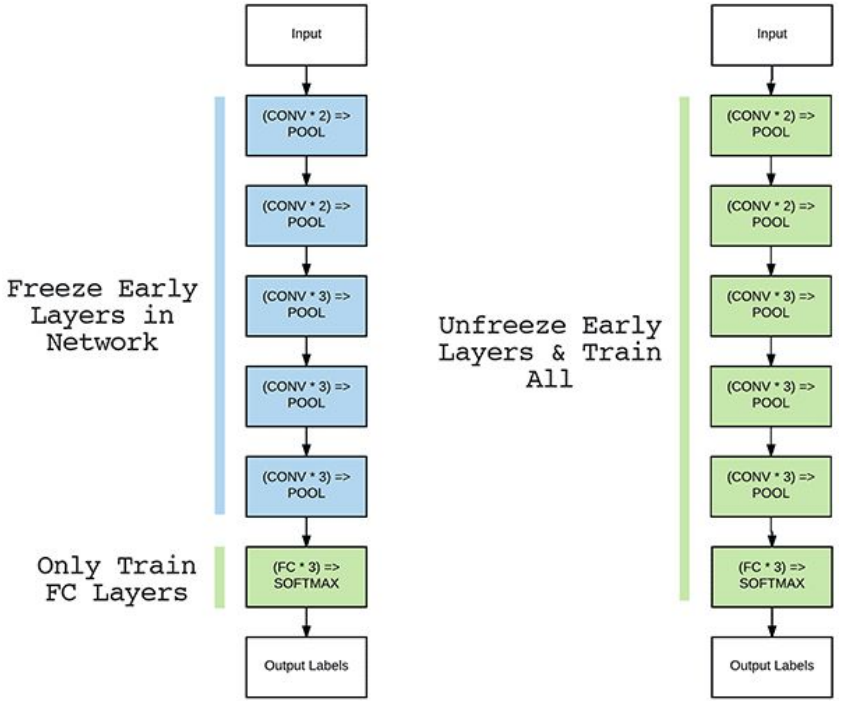
But are these models good at every task!? 🤔

Source: https://www.nytimes.com/wirecutter/reviews/best-multitool/

Motivation

One possible solution - Fine tune

- Finetune all layers
- Finetune only few layers
  - Which layers to finetune?



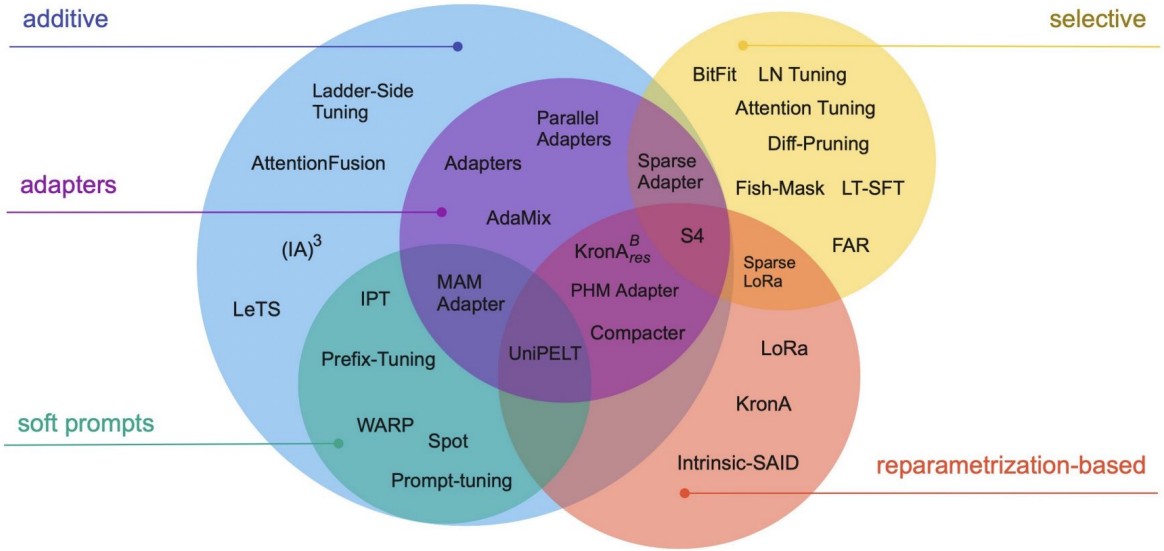
Source: https://pyimagesearch.com/2019/06/03/fine-tuning-with-keras-and-deep-learning/

PEFT (Parameter Efficient Fine tuning)

1. LoRA: LORA: LOW-RANK ADAPTATION OF LARGE LANGUAGE MODELS
2. Prefix Tuning: Prefix-Tuning: Optimizing Continuous Prompts for Generation, P-Tuning v2: Prompt Tuning Can Be Comparable to Fine-tuning Universally Across Scales and Tasks
3. P-Tuning: GPT Understands, Too
4. Prompt Tuning: The Power of Scale for Parameter-Efficient Prompt Tuning
5. AdaLoRA: Adaptive Budget Allocation for Parameter-Efficient Fine-Tuning
6. LLaMA-Adapter: Efficient Fine-tuning of Language Models with Zero-init Attention
7. IA3: Infused Adapter by Inhibiting and Amplifying Inner Activations

Source: https://huggingface.co/docs/peft/index

PEFT (Parameter Efficient Fine tuning)



Source: https://arxiv.org/pdf/2303.15647.pdf

More on PEFT

1. Huggingface
2. Advanced NLP grad level courses!

CS769 Advanced NLP

Parameter-Efficient Fine-Tuning (PEFT)

Rheeya Uppaal, Junjie Hu

WISCONSIN
UNIVERSITY OF WISCONSIN-MADISON

https://junjiehu.github.io/cs769-fall23/

Source: https://junjiehu.github.io/cs769-fall23/assets/pdf/anlp-15-peft.pdf

LoRA

- Became quite popular - 1400 citations in 2yrs

Hypothesis from previous works

- The weights of a pre trained transformer have a low “intrinsic dimension” and can still learn efficiently despite a random projection to a smaller subspace

  i.e the weights might be over-parametrized.

Reference: https://arxiv.org/pdf/2012.13255.pdf

LoRA

LoRA hypothesis - The *updates* to the weights can also be in “low rank”

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

If $W \in R^{dxk}$, then $B \in R^{dxr}$;$A \in R^{rxk}$ where $r << \min(d,k)$

- W has d*k while A,B in total has (d+k)*r

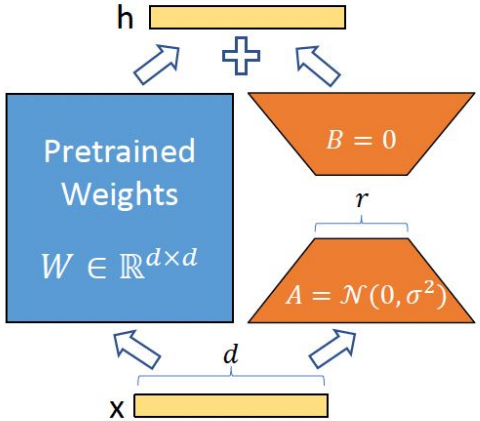


Figure 1: Our reparametrization. We only train A and B.

LoRA

Motivating use case - Language modeling

Original Fine Tuning

- Number of trainable parameters: Same as original model

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

LoRA

- Number of trainable parameters: *f*(original model) ~ can be as small as 0.01%

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

LoRA

Advantages

- Reduction in memory and storage usage

  On GPT-3 175B, we reduce the VRAM consumption during training from 1.2TB to 350GB. With r = 4 and only the query and value projection matrices being adapted, the checkpoint size is reduced by roughly 10,000 (from 350GB to 35MB)

  *We still need the 350GB model during deployment; however, storing 100 adapted models only requires 350GB + 35MB * 100 = 354GB as opposed to 100 * 350GB = 35TB.*

LoRA

Advantages

- No additional latency
- Easily switchable to new tasks (just replace BA)

Limitations

- Can’t batch inputs of different tasks because the matrices A, B will be generalizable and not task specific

LoRA

LoRA is fast in terms of inference latency

The difference pronounces at low batch-sizes and sequence-lengths!

Why?

- Hardware/Model parallelism
- Adapters needs to be computed sequentially

Batch Size Sequence Length $\|\Theta\|$	32 512 0.5M	16 256 11M	1 128 11M
Fine-Tune/LoRA	1449.4±0.8	338.0±0.6	19.8±2.7
$\text{Adapter}^{\text{L}}$	1482.0±1.0 (+2.2%)	354.8±0.5 (+5.0%)	23.9±2.1 (+20.7%)
$\text{Adapter}^{\text{H}}$	1492.2±1.0 (+3.0%)	366.3±0.5 (+8.4%)	25.8±2.2 (+30.3%)

Table 1: Infernece latency of a single forward pass in GPT-2 medium measured in milliseconds, averaged over 100 trials. We use an NVIDIA Quadro RTX8000. "$|\Theta|$" denotes the number of trainable parameters in adapter layers. $\text{Adapter}^{\text{L}}$ and $\text{Adapter}^{\text{H}}$ are two variants of adapter tuning, which we describe in Section 5.1. The inference latency introduced by adapter layers can be significant in an online, short-sequence-length scenario. See the full study in Appendix B.

LoRA

Experimental setup

LoRA applied only on attention weights.

MLP, LayerNorms and biases are frozen

- **BitFt**: Train only bias
- Prefix-Embedding Tuning (**PreEmbed**): Add learnable tokens to prompts
- Prefix-Layer Tuning (**PreLayer**): Activations after each layer are learnable!
- **Adapter^{H}**: Add adapters after attention module, after MLP
- **Adapter^{L}**: Add adapters after MLP and LayerNorm
- **Adapter^{D}**: Drop some adapter layers for efficiency
- **LoRA**: Add trainable “low-rank” matrices to W_q and W_v

LoRA

Encoder-only models, NLU (Natural Language Understanding)

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.

LoRA

Decoder-only models, NLG tasks

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

Table 3: 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. Confidence intervals are shown for experiments we ran. * indicates numbers published in prior works.

LoRA

NLU on larger models!

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 ($\text{Adapter}^{\text{H}}$)	7.1M	71.9	89.8	53.0/28.9/44.8
GPT-3 ($\text{Adapter}^{\text{H}}$)	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

Table 4: Performance of different adaptation methods on GPT-3 175B. We report the logical form validation accuracy on WikiSQL, validation accuracy on MultiNLI-matched, and Rouge-1/2/L on SAMSum. LoRA performs better than prior approaches, including full fine-tuning. The results on WikiSQL have a fluctuation around $\pm 0.5\%$, MNLI-m around $\pm 0.1\%$, and SAMSum around $\pm 0.2/\pm 0.2/\pm 0.1$ for the three metrics.

LoRA

Question - Is having more trainable parameters better for any PEFT/full-finetune method?

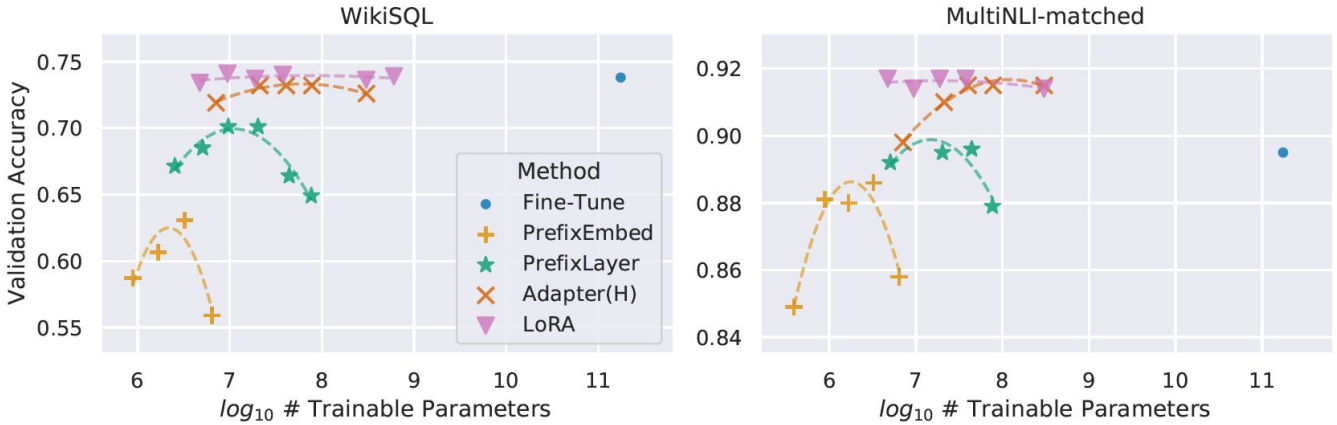


Figure 2: GPT-3 175B validation accuracy vs. number of trainable parameters of several adaptation methods on WikiSQL and MNLI-matched. LoRA exhibits better scalability and task performance. See Section F.2 for more details on the plotted data points.

More special tokens might cause the input distribution to shift further from pre training data distribution

LoRA

Question - What matrices to chose for LoRA?

	# 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.

Query and Value matrices are a good bet!

LoRA

Question - High rank but few matrices or low rank but many matrices?

	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.

Low rank but many matrices is a better choice! But why?

LoRA

Intuition: Increasing r doesn't cover a more meaningful subspace i.e whatever is covered by smaller matrix is covered by bigger matrix!

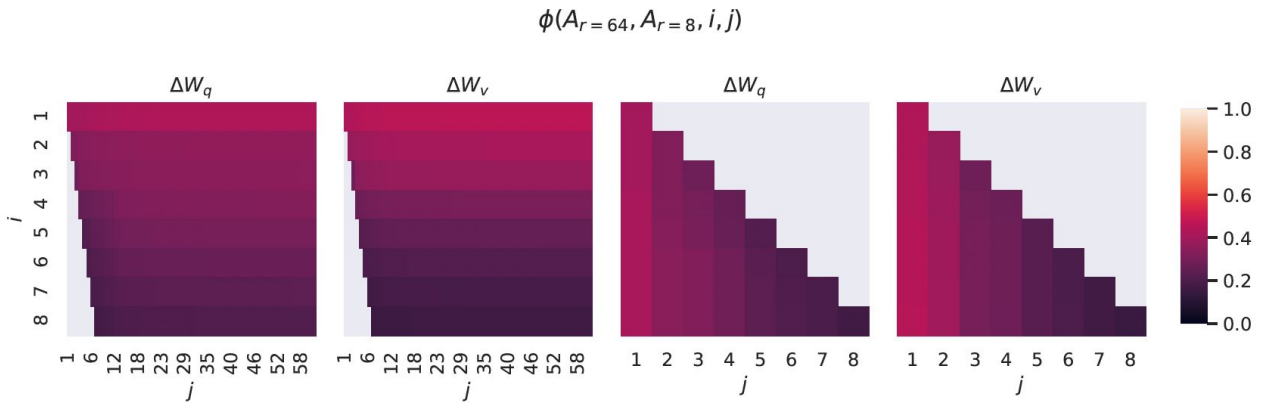


Figure 3: Subspace similarity between column vectors of $A_{r=8}$ and $A_{r=64}$ for both ΔW_q and ΔW_v. The third and the fourth figures zoom in on the lower-left triangle in the first two figures. The top directions in $r = 8$ are included in $r = 64$, and vice versa.

LoRA

Subspace similarity between different random seeds

XYZ

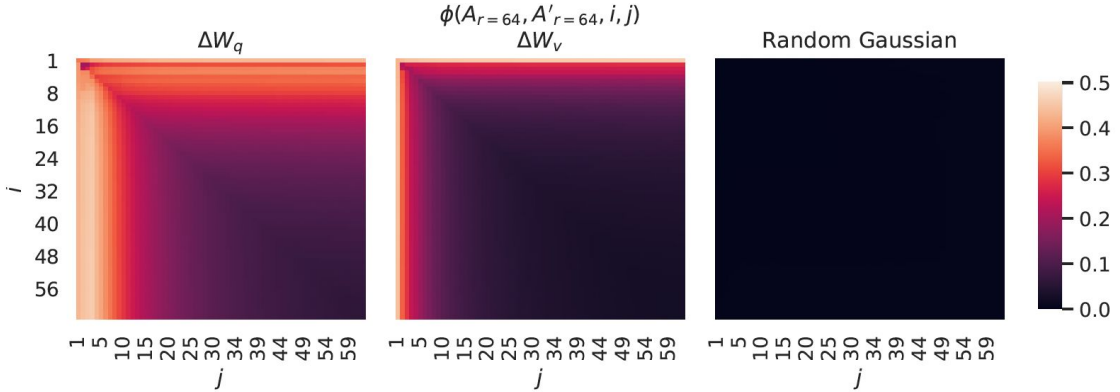


Figure 4: **Left and Middle:** Normalized subspace similarity between the column vectors of $A_{r=64}$ from two random seeds, for both ΔW_q and ΔW_v in the 48-th layer. **Right:** the same heat-map between the column vectors of two random Gaussian matrices. See Section H.1 for other layers.

LoRA

Question - Is there any correlation between W and delta W?

	$r = 4$			$r = 64$		
	ΔW_q	W_q	Random	ΔW_q	W_q	Random
$\|\|U^\top W_q V^\top\|\|_F =$	0.32	21.67	0.02	1.90	37.71	0.33
$\|\|W_q\|\|_F = 61.95$	$\|\|\Delta W_q\|\|_F = 6.91$			$\|\|\Delta W_q\|\|_F = 3.57$		

Table 7: The Frobenius norm of $U^\top W_q V^\top$ where U and V are the left/right top r singular vector directions of either (1) ΔW_q, (2) W_q, or (3) a random matrix. The weight matrices are taken from the 48th layer of GPT-3.

Both are correlated. Delta W only amplifies directions that are not emphasized in W!

LoRA

Summarizing

- **Goal**: Reduce computational and memory requirement for fine-tuning
- **Idea**: Using low-rank matrix updates

CAN WE GO FURTHER?

Precision

Floating Point Representation: [Sign] [Exponent] [Mantissa]

- Use 64 or 32 bits of memory to represent each floating point
- Can we go lower? (Say 16-bit)

How does it affect accuracy?

Precision

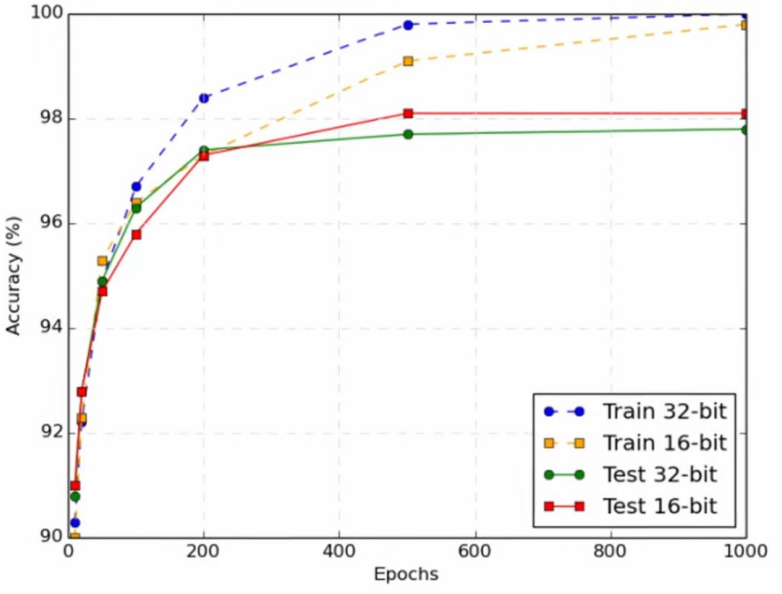
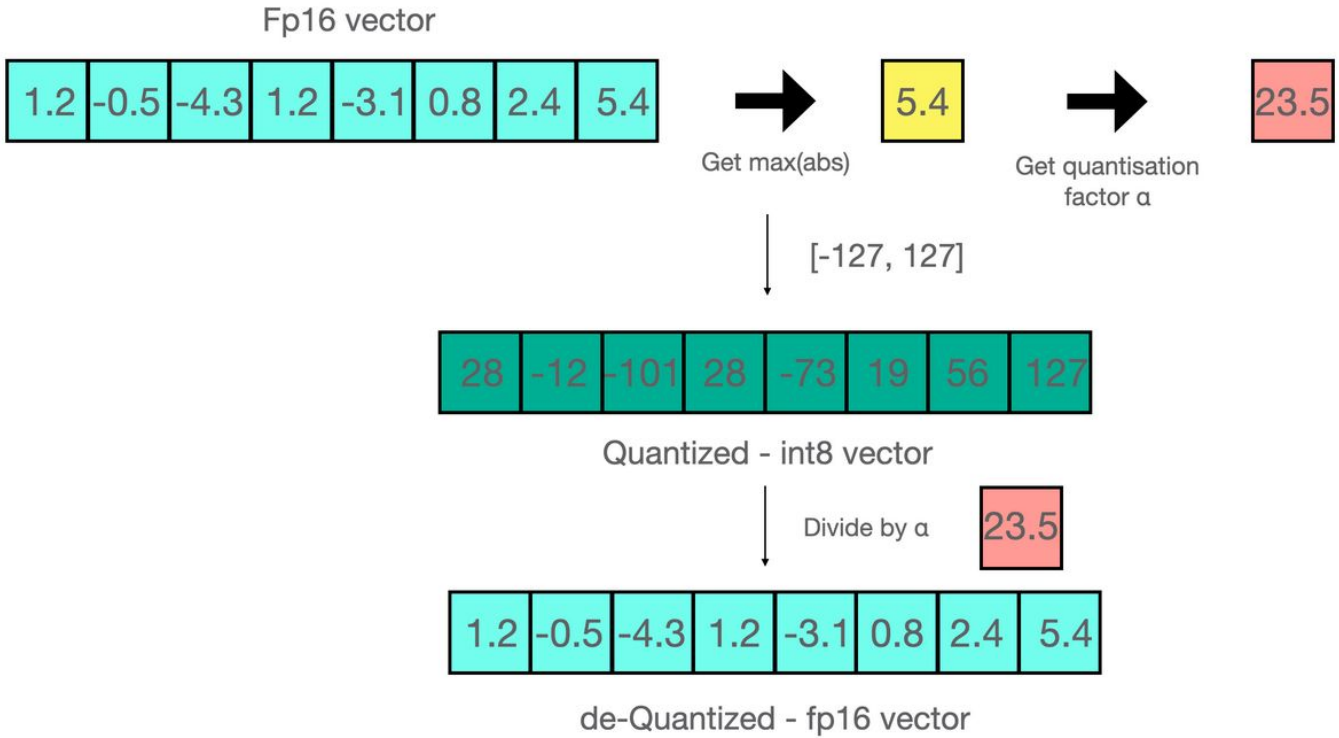


Figure 5: Accuracy comparison with various floating point representation.
https://arxiv.org/abs/2301.12809

Quantization

- Lowering precision further causes poor inference quality
- Main Idea: “rounding” from one data type to another.
  - For example: 32-bit floating point to 8-bit integers
- Problem:
  - Information loss
  - Need **some technique** to prevent loss of information
- Techniques:
  - Zero-point quantization
  - Absolute maximum (absmax) quantization

Quantization



‘Quantized’ LoRA

Advantages

- Lower computational burden
- Lower memory usage

Disadvantages

- Accuracy?

qLoRA

- BFloat16 is classical fine-tuning
- Float4 and NFloat4 + DQ are two QLoRA fine-tunings
- Almost equal performance

	Mean 5-shot MMLU Accuracy								
LLaMA Size	7B		13B		33B		65B		Mean
Dataset	Alpaca	FLAN v2	Alpaca	FLAN v2	Alpaca	FLAN v2	Alpaca	FLAN v2	
BFloat16	38.4	45.6	47.2	50.6	57.7	60.5	61.8	62.5	53.0
Float4	37.2	44.0	47.3	50.0	55.9	58.5	61.3	63.3	52.2
NFloat4 + DQ	39.0	44.5	47.5	50.7	57.3	59.2	61.8	63.9	53.1

Why?

qLoRA

- Only frozen parameters are quantized.
- During inference, the frozen 8-bit parameters are de-quantized to 32-bit.
- The trainable parameters can fix or undo the errors that were added when quantizing and de-quantizing the non-trainable weights.
- Can go even as low as 4-bit integers

Variants!

Monarch matrices - ICML 2022 Outstanding paper runner up!

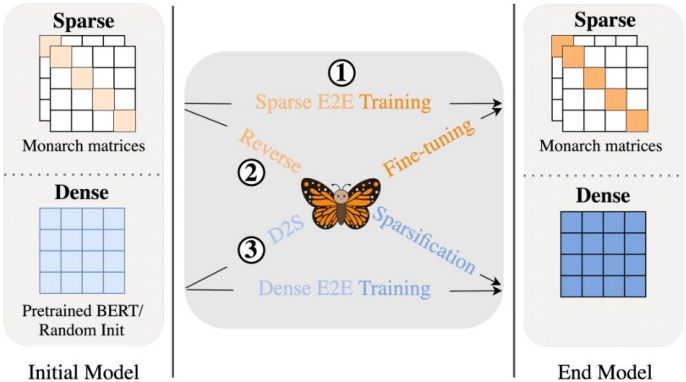


Figure 1: Monarch matrices unlock several ways to train sparse and dense models: end-to-end training a sparse (Monarch) model can be 2x faster than dense training thanks to its hardware efficiency; sparse-to-dense "reverse sparsification" can speed up training of large models such as GPT-2; and our dense-to-sparse Monarch projection algorithm can transfer knowledge from pretrained dense model to Monarch model and speed up BERT fine-tuning.

Key Takeaways and thoughts..!

- Foundational models are huge and we need smarter techniques for finetuning
- Some of the not-so efficient techniques are:
  - Traditional Transfer Learning
  - Adapter Layers
  - Prefix Tuning
- Efficient techniques:
  - LoRA: Using low-rank matrix updates
  - qLoRA: Quantize the frozen parameters in LoRA and dequantize them during inference
- Other techniques:
  - Meta-learning
  - Cross-Domain adaptation
  - BitFit etc.

Thank you