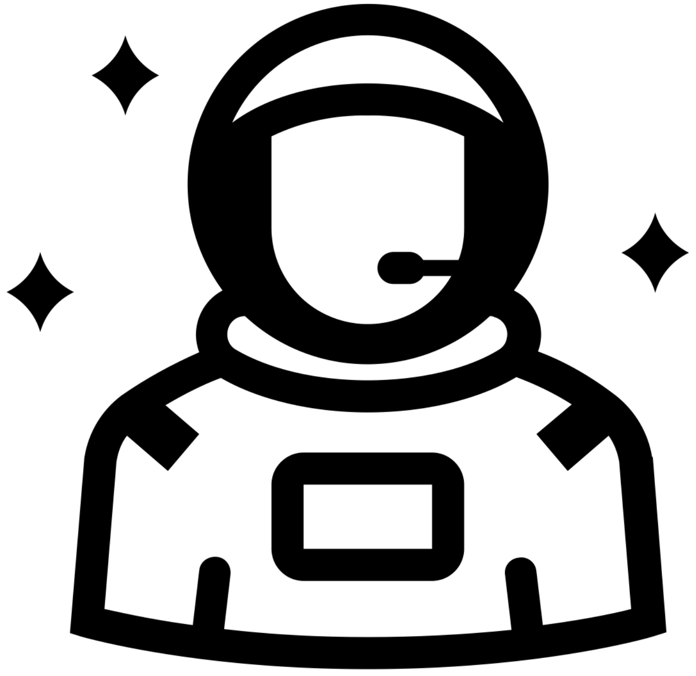
PEFT

Jack Roberts

Transformers Reading Group

18th September 2023



I want to fine-tune* a LLM

*our starting point is an existing pre-trained (foundation) model for everything we're talking about today

A GPU waiting to be used:

Photo by ilya gorborukov on Unsplash

Load the largest model that fits:

Pre-trained Model

Photo by Marc Babin on Unsplash

Attempt to fine-tune:

Photo by [Zac Harris](#) on [Unsplash](#)

Full Fine-tuning

PROS

- Max performance (maybe)

CONS

- Max resources (hardware + time)

Alternatives to Full Fine-tuning

Prompt engineering

Quantisation

Parameter-efficient fine-tuning (PEFT)

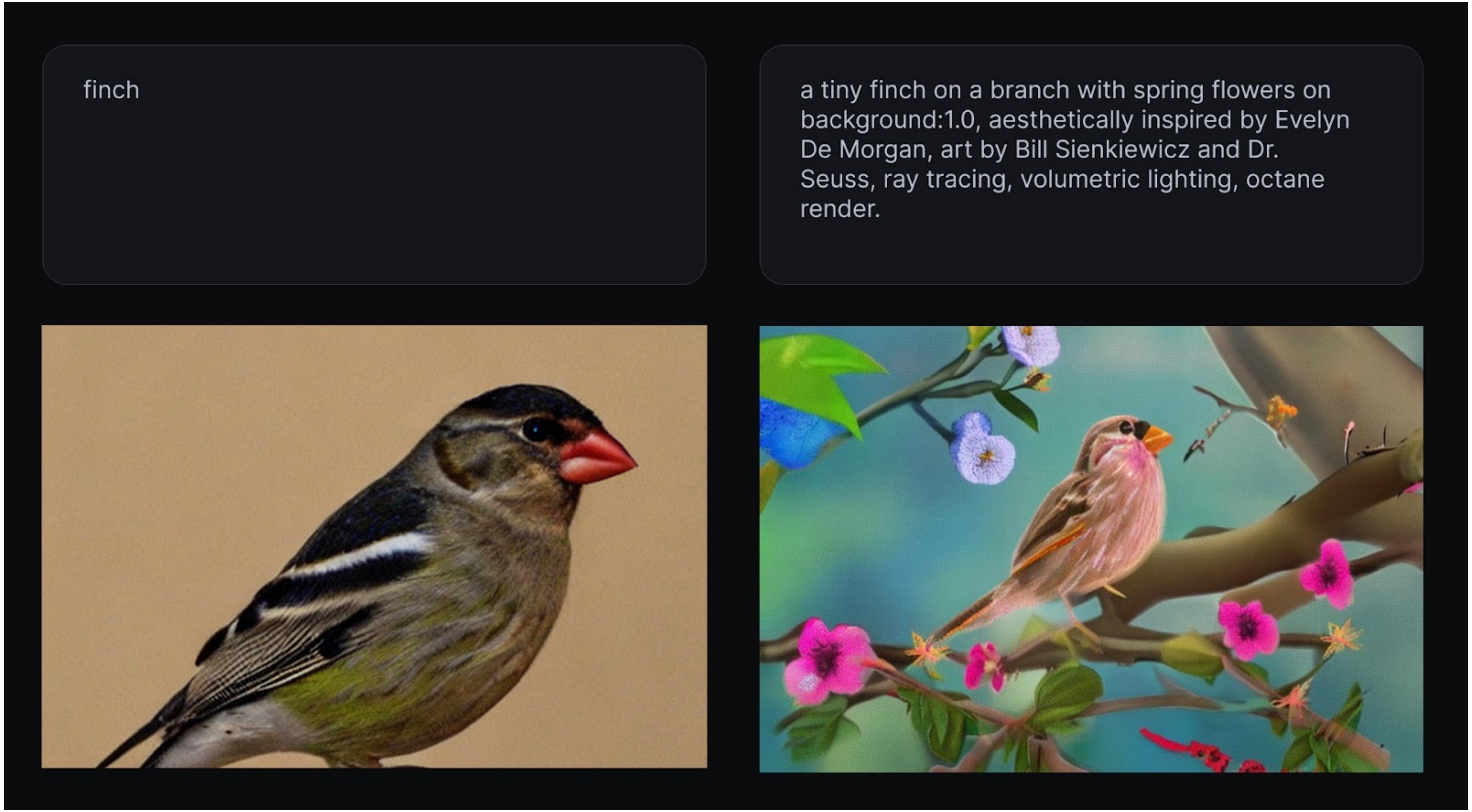
(or combinations of these)

Prompt Engineering

Don’t change the model at all, find a prompt that maximises performance for your task.

(also deciding what data to include in the prompt, e.g. LlamaIndex https://github.com/jerryjliu/llama_index)

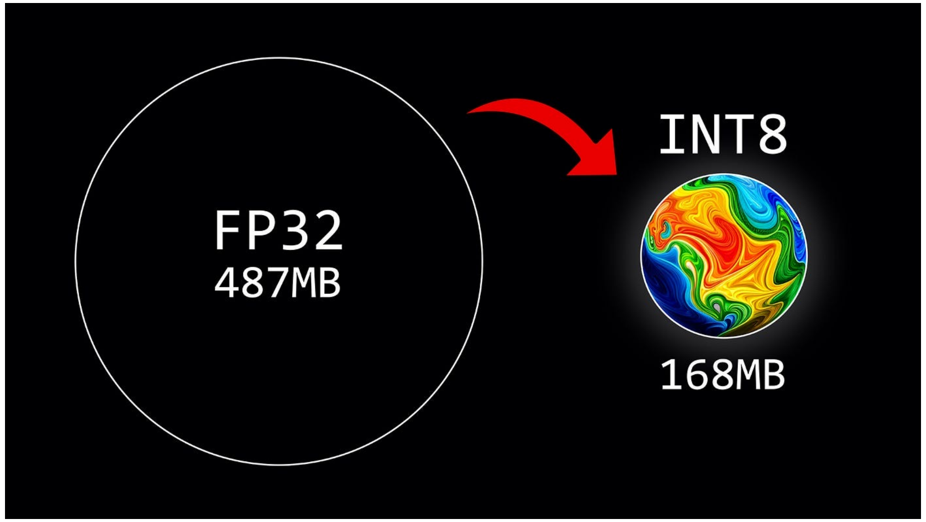
Prompt Engineering



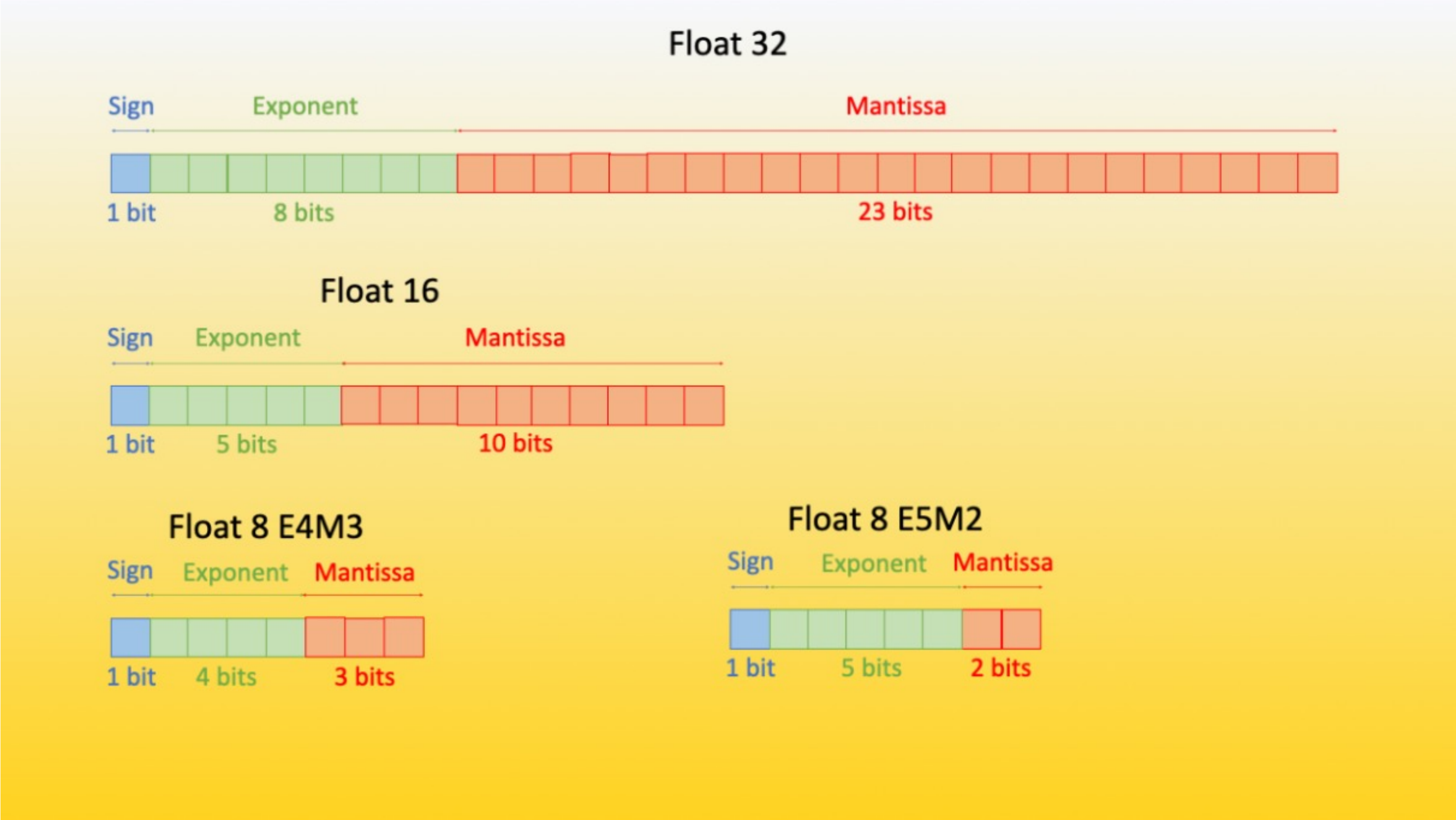
https://letsenhance.io/blog/article/ai-text-prompt-guide/

Quantisation

Load the model in lower precision (use fewer bits per parameter)



https://towardsdatascience.com/introduction-to-weight-quantization-2494701b9c0c



https://huggingface.co/blog/hf-bitsandbytes-integration
https://huggingface.co/blog/4bit-transformers-bitsandbytes

PEFT: Parameter-Efficient Fine-Tuning

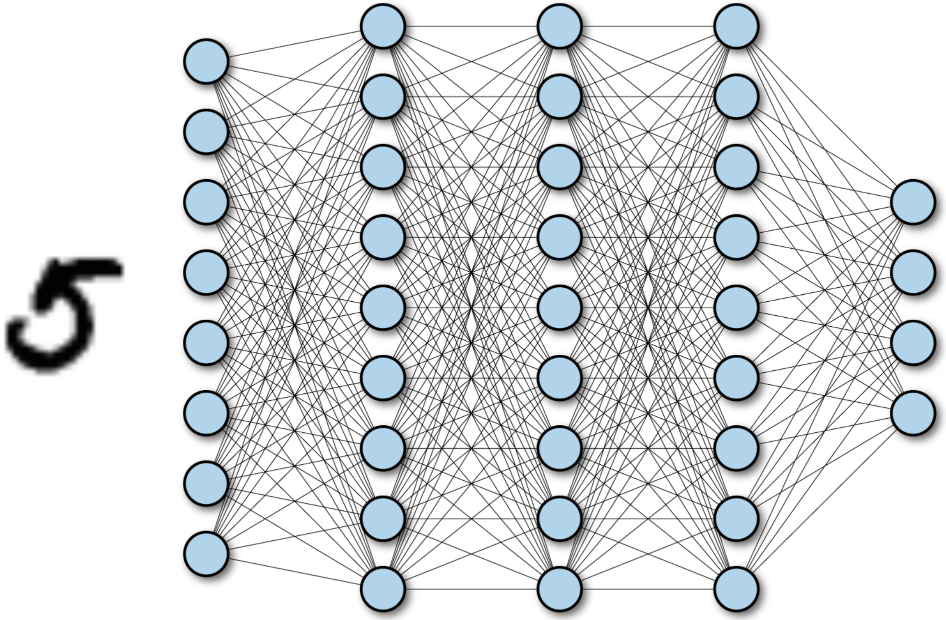
- Fine-tune the model by only modifying/adding a small number of parameters (e.g. 0.1 – 1.0% of the parameters in the original model)
  - Majority of model frozen – don’t need to track gradients for frozen parameters – smaller memory footprint.
- Counter-intuitively, PEFT may **add** parameters to the model, so the overall model could be *slightly* larger (but training is much cheaper because of the reasons above)
- Can achieve performance close to full fine-tuning (maybe better than full fine-tuning in some cases, e.g. small datasets)

PEFT

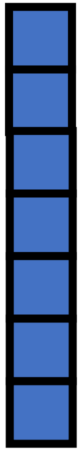
Measuring the Intrinsic Dimension of Objective Landscapes (1)

Chunyuan Li, Heerad Farkhoor, Rosanne Liu, Jason Yosinski (2018)
https://arxiv.org/pdf/1804.08838.pdf

Fully-connected network, train on MNIST:



Represent parameters as flat vector:



$\theta^{(D)}$

Measuring the Intrinsic Dimension of Objective Landscapes (2)

Chunyuan Li, Heerad Farkhoor, Rosanne Liu, Jason Yosinski (2018)
https://arxiv.org/pdf/1804.08838.pdf

- Training in lower dimensions:

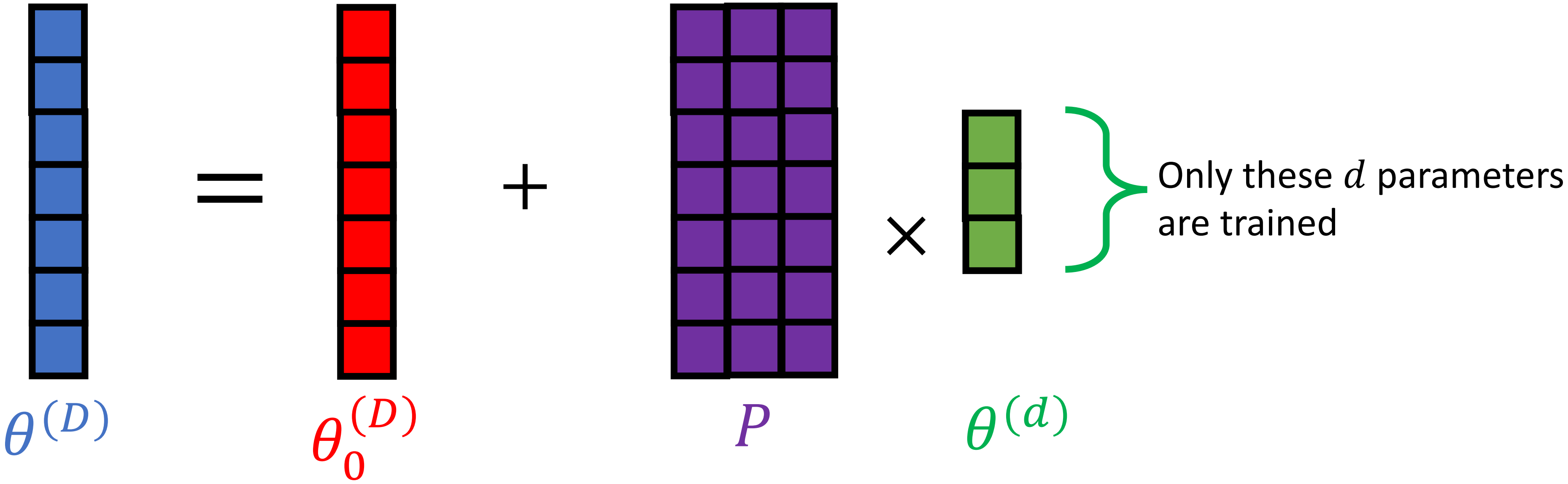
$$\theta^{(D)} = \theta_0^{(D)} + P\theta^{(d)}$$

$\theta^{(D)}$: Vector of *tuned* model parameters, dimensions $D \times 1$

$\theta_0^{(D)}$: Vector of *initial* model parameters, dimensions $D \times 1$ (frozen)

P: Random projection matrix, dimensions $D \times d$ (frozen)

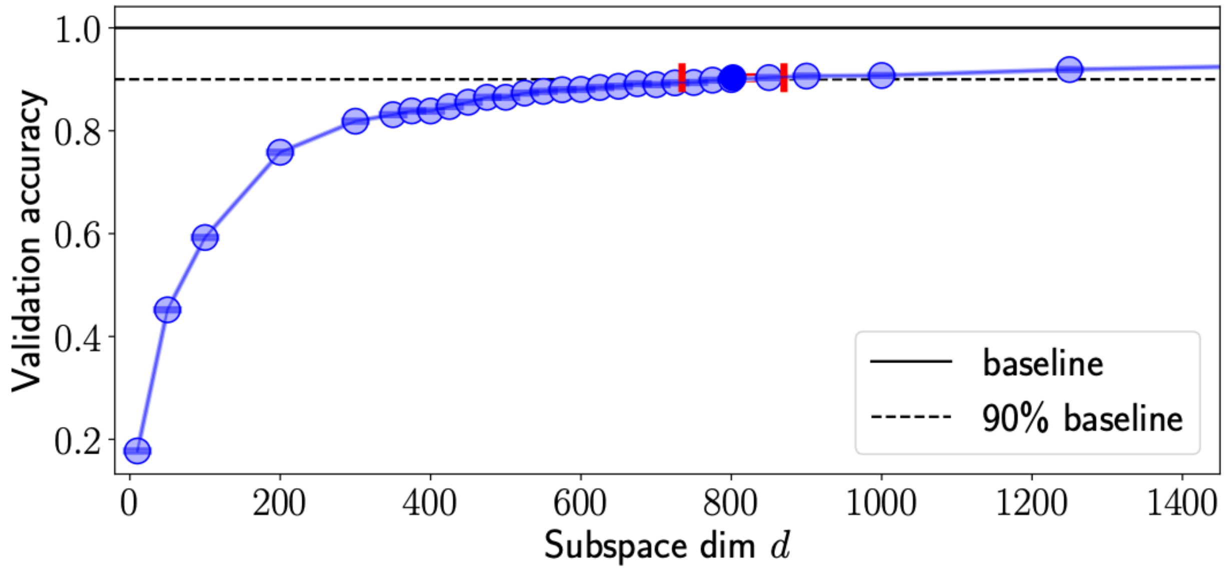
$\theta^{(d)}$: Parameter vector in smaller space ($d < D$), dimensions d×1 (trained)



Measuring the Intrinsic Dimension of Objective Landscapes (3)

Chunyuan Li, Heerad Farkhoor, Rosanne Liu, Jason Yosinski (2018)
https://arxiv.org/pdf/1804.08838.pdf

- Fully-connected network with 199 210 parameters
- Train with varying subspace dimensions
- Baseline = well-trained model
- "Intrinsic dimension": Subspace dimension where performance 90% of baseline
- Measured intrinsic dimension depends on model architecture and dataset



Intrinsic Dimensionality Explains the Effectiveness of Language Model Fine-Tuning

Armen Aghajanyan, Luke Zettlemoyer, Sonal Gupta (2020)
https://arxiv.org/abs/2012.13255

- Same approach applied to language models
- Intrinsic dimension of BERT/RoBERTa models on two datasets (MRPC, QQP)
  - MRPC: Binary classification (~are two sentences rewordings of each other)
  - QQP: Binary classification (~are two questions the same)

Model	MRPC	QQP
BERT-Base	1608	8030
BERT-Large	1037	1200
RoBERTa-Base	896	896
RoBERTa-Large	**207**	**774**

- Motivates potential to fine-tune LLMs using a subset of parameters / with small datasets.

LoRA: Low-Rank Adaptation of Large Language Models

EJ. Hu et al. (Microsoft), 2021
https://arxiv.org/abs/2106.09685
https://github.com/microsoft/LoRA

- One of the most established/most popular PEFT methods
- Uses the same intrinsic dimension concept, but applied to individual weight matrices in different layers of the model (rather than to the whole model at once)

Full Fine-Tuning

$$W_0 + \Delta W$$

$W_0 \in \mathbb{R}^{d \times k}$ Initial weights (frozen)

$\Delta W \in \mathbb{R}^{d \times k}$ Fine-tuned weight updates

LoRA Fine-Tuning

$$W_0 + BA$$

$W_0 \in \mathbb{R}^{d \times k}$ Initial weights (frozen)

$$\left.\begin{array}{l} B \in \mathbb{R}^{d \times r} \\ A \in \mathbb{R}^{r \times k} \end{array}\right\}$$ Fine-tuned LoRA weights

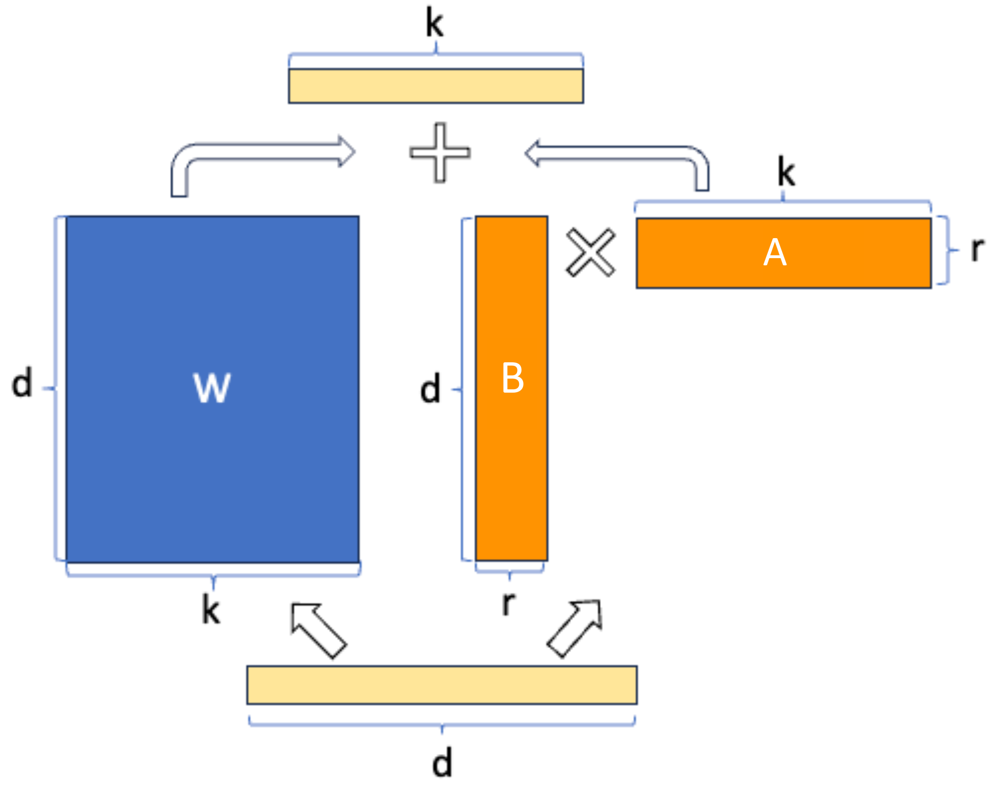
Image from: https://dataman-ai.medium.com/fine-tune-a-gpt-lora-e9b72ad4ad3

LoRA Fine-tuning

$$W_0 + BA$$

$W_0 \in \mathbb{R}^{d \times k}$ Initial weights (frozen)

$\left.\begin{array}{l} B \in \mathbb{R}^{d \times r} \\ A \in \mathbb{R}^{r \times k} \end{array}\right\}$ Fine-tuned LoRA weights



- LoRA rank: $r \ll \min(d, k)$ ($r = 8$ is the default in the HuggingFace implementation)
  - Total number of parameters in B + A << number of parameters in W_0
- In the original paper, LoRA applied to attention weights only (not weights in fully connected layers)
  - But can be applied to any weights in theory

LoRA vs. Normal Fine-tuning (FT) and Other PEFT Methods

RoBERTa Base:
0.3M LoRA parameters vs. 125M in model (0.24%)

RoBERTa Large:
0.8M LoRA parameters vs. 355M in model (0.23%)

DeBERTa XXL:
4.7M LoRA parameters vs. 1500M in model (0.31%)

Model & Method	# Trainable Parameters	Datasets								
		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$_{\pm.3}$	**95.1**$_{\pm.2}$	89.7$_{\pm.7}$	63.4$_{\pm1.2}$	**93.3**$_{\pm.3}$	90.8$_{\pm.1}$	**86.6**$_{\pm.7}$	**91.5**$_{\pm.2}$	**87.2**
RoB$_{large}$ (FT)*	355.0M	90.2	**96.4**	**90.9**	68.0	94.7	**92.2**	86.6	92.4	88.9
RoB$_{large}$ (LoRA)	0.8M	**90.6**$_{\pm.2}$	96.2$_{\pm.5}$	**90.9**$_{\pm1.2}$	**68.2**$_{\pm1.9}$	**94.9**$_{\pm.3}$	91.6$_{\pm.1}$	**87.4**$_{\pm2.5}$	**92.6**$_{\pm.2}$	**89.0**
RoB$_{large}$ (Adpt^{P})†	3.0M	90.2$_{\pm.3}$	96.1$_{\pm.3}$	90.2$_{\pm.7}$	**68.3**$_{\pm1.0}$	**94.8**$_{\pm.2}$	**91.9**$_{\pm.1}$	83.8$_{\pm2.9}$	92.1$_{\pm.7}$	88.4
RoB$_{large}$ (Adpt^{P})†	0.8M	**90.5**$_{\pm.3}$	**96.6**$_{\pm.2}$	89.7$_{\pm1.2}$	67.8$_{\pm2.5}$	**94.8**$_{\pm.3}$	91.7$_{\pm.2}$	80.1$_{\pm2.9}$	91.9$_{\pm.4}$	87.9
RoB$_{large}$ (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
RoB$_{large}$ (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
RoB$_{large}$ (LoRA)†	0.8M	**90.6**$_{\pm.2}$	96.2$_{\pm.5}$	**90.2**$_{\pm1.0}$	68.2$_{\pm1.9}$	**94.8**$_{\pm.3}$	91.6$_{\pm.2}$	**85.2**$_{\pm1.1}$	**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	**91.9**$_{\pm.2}$	96.9$_{\pm.2}$	**92.6**$_{\pm.6}$	**72.4**$_{\pm1.1}$	**96.0**$_{\pm.1}$	**92.9**$_{\pm.1}$	**94.9**$_{\pm.4}$	**93.0**$_{\pm.2}$	**91.3**

LoRA matches / outperforms full fine-tuning in many of the examples

LoRA Benefits

- Training uses much less GPU memory
  (GPT-3 175B example: 1.2TB reduced to 350GB)

- Training is faster – don’t need gradients for most parameters
  (GPT-3 175B example in paper: 25% speedup)

- Only need to save the LoRA weights to disk (not the whole model again)
  (GPT-3 175B example: full model 350 GB, LoRA weights 35 MB – 10,000x smaller)

- Can have one deployed base model and quickly swap in/out LoRA adapters
  (much faster than redploying a new full LLM)

- Adds no inference latency (some PEFT methods do)
  (when loading the model, compute $W = W_0 + BA$ – i.e. add the LoRA weights to the original model weights)

Other Applications of LoRA (1): QLoRA

- QLoRA: Efficient Finetuning of Quantized LLMs
  Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, Luke Zettlemoyer (2023)
  https://arxiv.org/abs/2305.14314

- Combines LoRA and 4-bit (!) quantisation to fine-tune Llama (65B parameters) on a single GPU, and match 16-bit fine-tuning performance.

- HuggingFace blog: https://huggingface.co/blog/4bit-transformers-bitsandbytes

Other Applications of LoRA (2): Diffusion Models (DreamBooth)

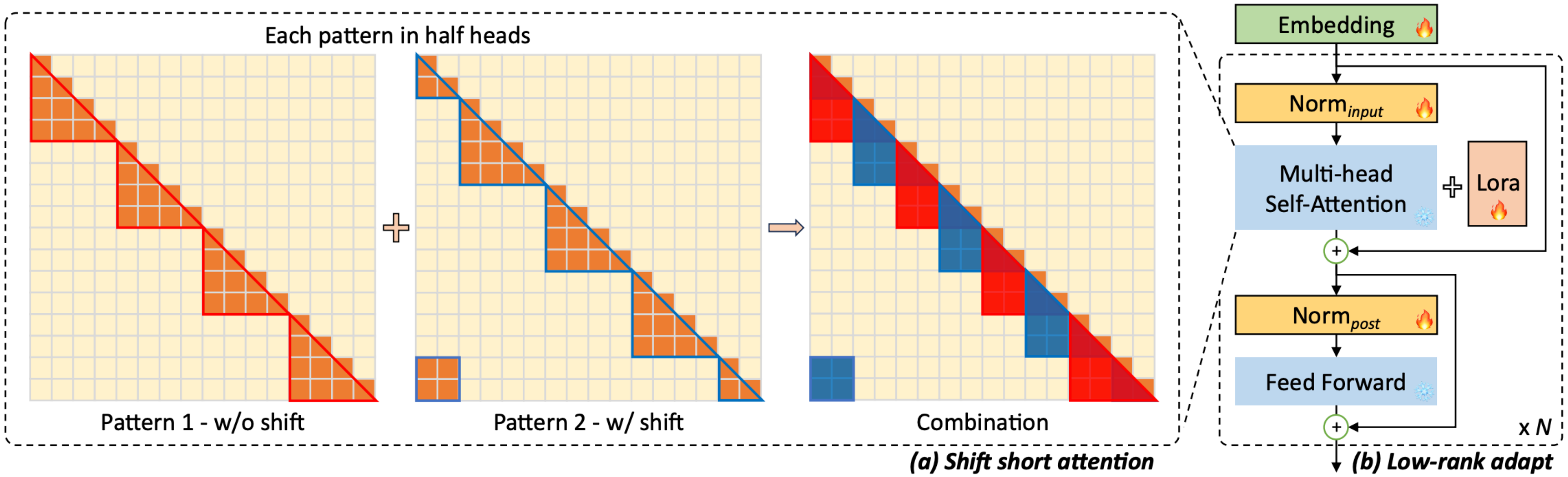
"dog" adapter

"toy" adapter

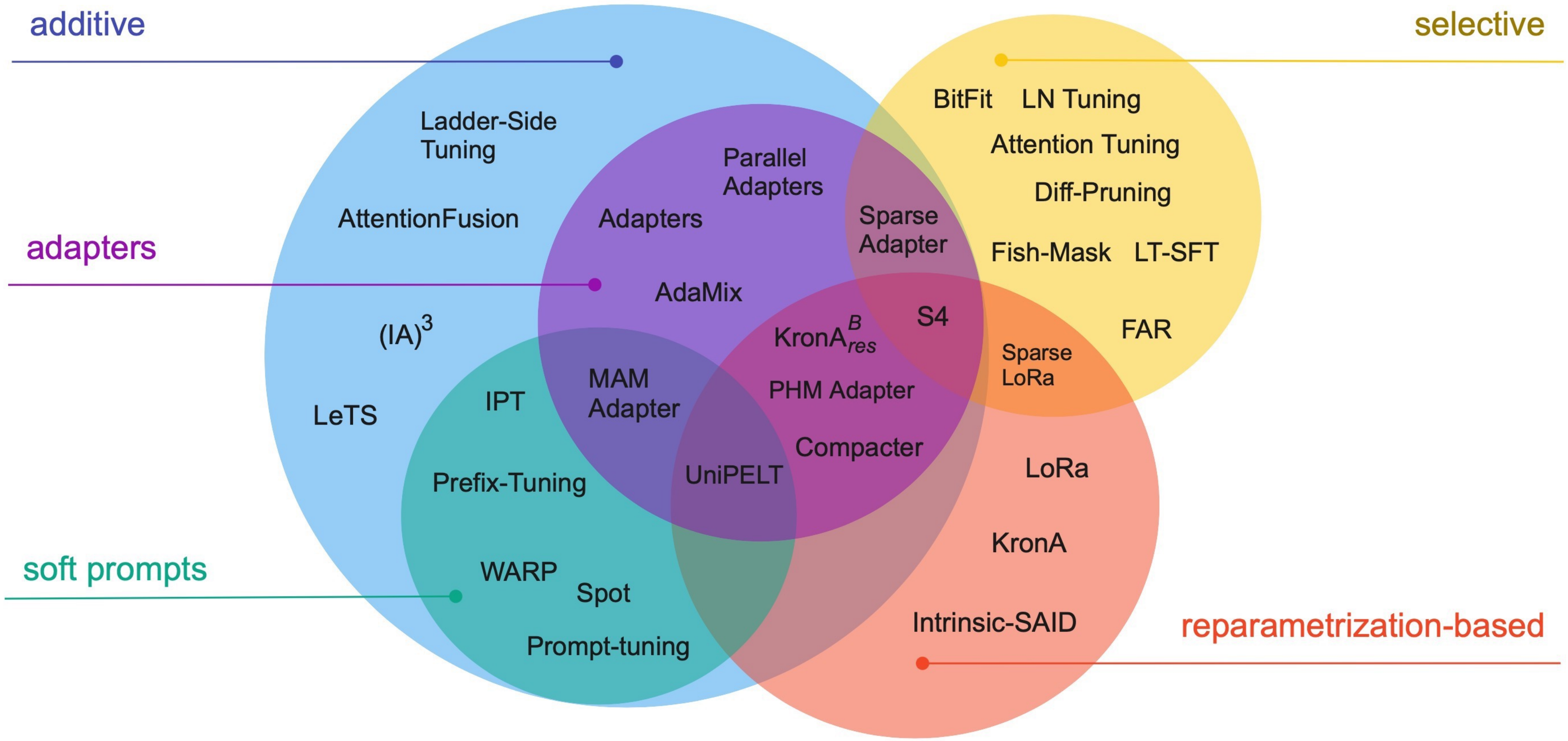
"toy" + "dog" adapter

Other Applications of LoRA (3): Long LoRA

- LongLoRA: Efficient Fine-tuning of Long-Context Large Language Models
  Yukang Chen et al. (2023)
  https://arxiv.org/abs/2309.12307
- Shift-short attention: Attention in groups of tokens (mostly between neighbouring tokens rather than between all tokens)
- LoRA in embedding and normalization layers
- Adapts base model for much longer inputs (“context sizes”) – e.g. Llama2 7B from 4k tokens to 100k tokens

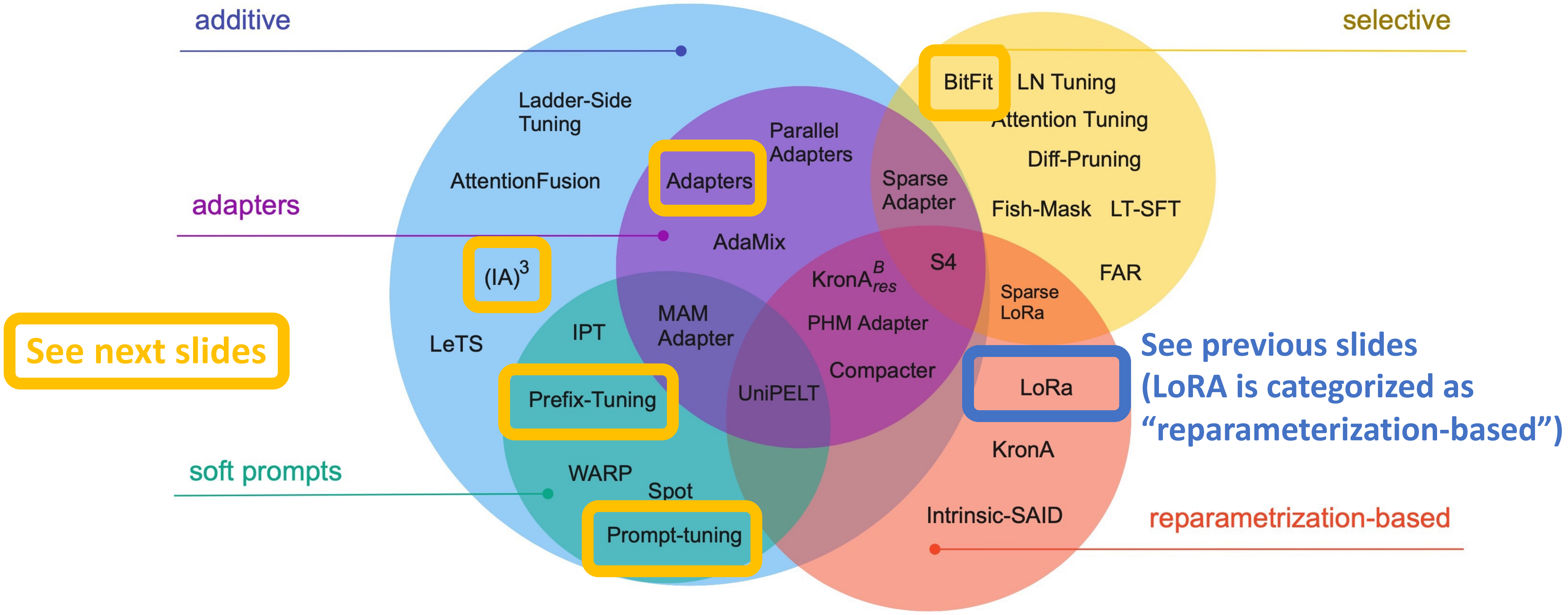


Categories of PEFT



Scaling Down to Scale Up: A Guide to Parameter-Efficient Fine-Tuning
Vladislav Lialin, Vijeta Deshpande, Anna Rumshisky (2023)
https://arxiv.org/abs/2303.15647

Categories of PEFT



Scaling Down to Scale Up: A Guide to Parameter-Efficient Fine-Tuning
Vladislav Lialin, Vijeta Deshpande, Anna Rumshisky (2023)
https://arxiv.org/abs/2303.15647

BitFit

BitFit: Simple Parameter-efficient Fine-tuning for Transformer-based Masked Language-models
Elad Ben Zaken, Shauli Ravfogel, Yoav Goldberg (2021)
https://arxiv.org/abs/2106.10199

- Train only the bias terms in the model

Attention layers:

$$\mathbf{Q}^{m,\ell}(\mathbf{x}) = \mathbf{W}_q^{m,\ell}\mathbf{x} + \mathbf{b}_q^{m,\ell}$$
$$\mathbf{K}^{m,\ell}(\mathbf{x}) = \mathbf{W}_k^{m,\ell}\mathbf{x} + \mathbf{b}_k^{m,\ell}$$
$$\mathbf{V}^{m,\ell}(\mathbf{x}) = \mathbf{W}_v^{m,\ell}\mathbf{x} + \mathbf{b}_v^{m,\ell}$$

Fully-connected layers:

$$\mathbf{h}_2^\ell = \mathrm{Dropout}(\mathbf{W}_{m_1}^\ell \cdot \mathbf{h}_1^\ell + \mathbf{b}_{m_1}^\ell)$$
$$\mathbf{h}_3^\ell = \mathbf{g}_{LN_1}^\ell \odot \frac{(\mathbf{h}_2^\ell + \mathbf{x}) - \mu}{\sigma} + \mathbf{b}_{LN_1}^\ell$$
$$\mathbf{h}_4^\ell = \mathrm{GELU}(\mathbf{W}_{m_2}^\ell \cdot \mathbf{h}_3^\ell + \mathbf{b}_{m_2}^\ell)$$
$$\mathbf{h}_5^\ell = \mathrm{Dropout}(\mathbf{W}_{m_3}^\ell \cdot \mathbf{h}_4^\ell + \mathbf{b}_{m_3}^\ell)$$
$$\mathrm{out}^\ell = \mathbf{g}_{LN_2}^\ell \odot \frac{(\mathbf{h}_5^\ell + \mathbf{h}_3^\ell) - \mu}{\sigma} + \mathbf{b}_{LN_2}^\ell$$

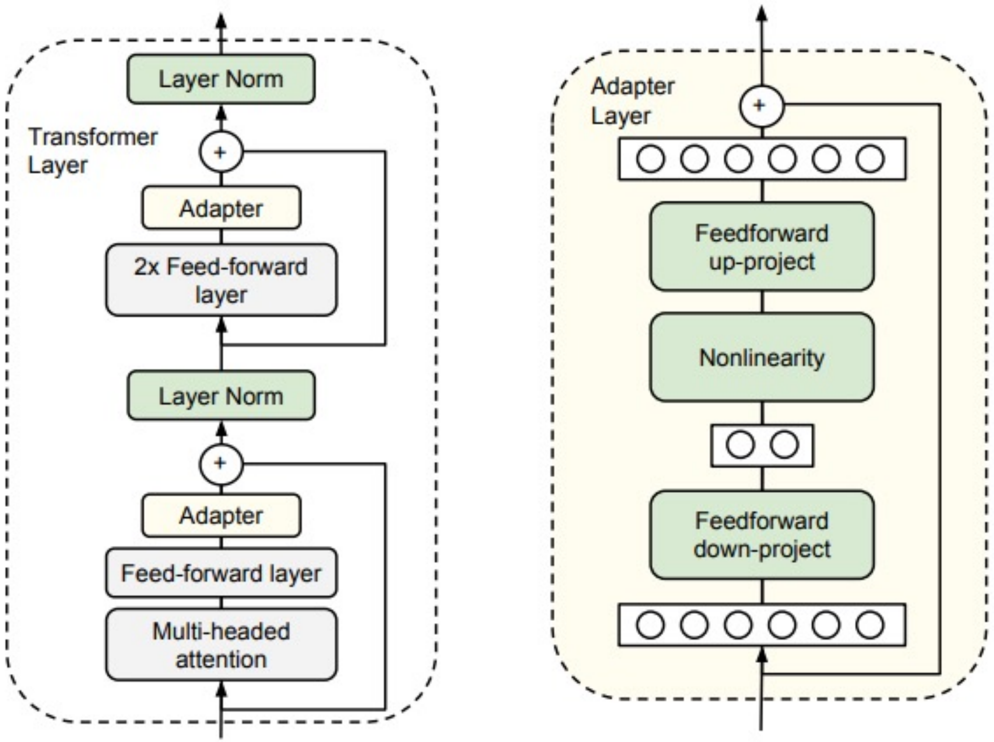
Bias terms (in red) – typically ~0.1% of the parameters in the model

Categorised as a “selective” PEFT method

Adapters

Parameter-Efficient Transfer Learning for NLP
Neil Houlsby et al. (2019)
https://arxiv.org/abs/1902.00751

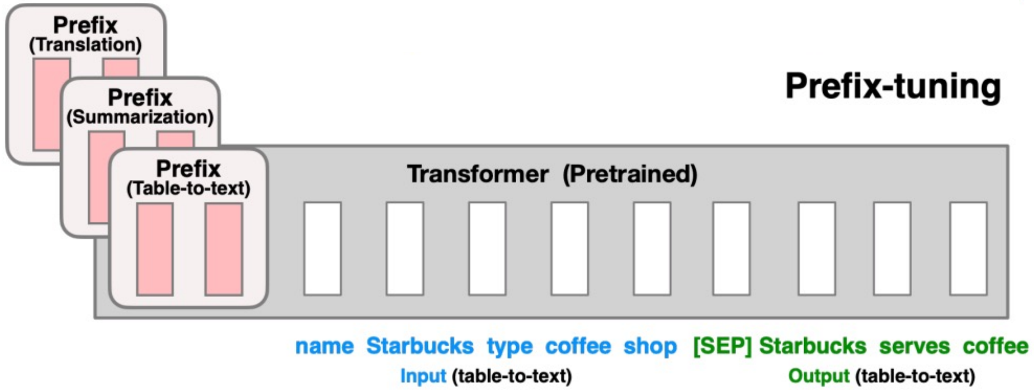
- Insert adapter layers in each transformer block
- Note this adds latency at inference time (unlike LoRA)



Prefix Tuning

Prefix-Tuning: Optimizing Continuous Prompts for Generation
Xiang Lisa Li, Percy Liang (2021)
https://arxiv.org/abs/2101.00190

- Learn a vector to prepend to the input prompt (with the whole model fixed in fine-tuning)
- Prefix is not limited to being tokens from the model - they are continuous vectors e.g. the prefix is not “Summarise this text”, it’s (0.2, 0.5, 0.8, 0.3, …)
- Use different prefixes for different tasks – can even get the model to perform multiple tasks per batch
- Can also add a prefix in every transformer layer (not only input)



Categorised as a “soft prompts” method

Prefix Tuning

Prefix-Tuning: Optimizing Continuous Prompts for Generation
Xiang Lisa Li, Percy Liang (2021)
https://arxiv.org/abs/2101.00190

Prompt Tuning

The Power of Scale for Parameter-Efficient Prompt Tuning
Brian Lester, Rami Al-Rfou, Noah Constant (2021)
https://arxiv.org/abs/2104.08691

P-Tuning

GPT Understands, Too
Xiao Liu, Yanan Zheng, Zhengxiao Du, Ming Ding, Yujie Qian, Zhilin Yang, Jie Tang (2021)
https://arxiv.org/abs/2103.10385

- All very similar, subtleties in where the soft prompt is inserted (e.g. only a prefix or throughout the prompt), and in which layers (e.g. only initial input prompt or throughout model layers)

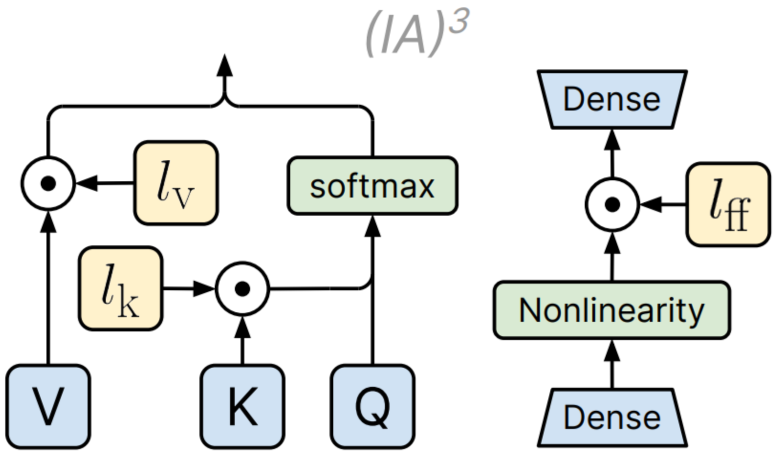
Categorised as a “soft prompts” methods

$(IA)^3$

Few-Shot Parameter-Efficient Fine-Tuning is Better and Cheaper than In-Context Learning
Haokun Liu et al. (2022)
https://arxiv.org/abs/2205.05638

- Learn vectors (l_v, l_k) to rescale activations (applied to keys and values in attention layers)
- Claims to match/out-perform LoRA with fewer parameters

How rescaling inserted in attention blocks:

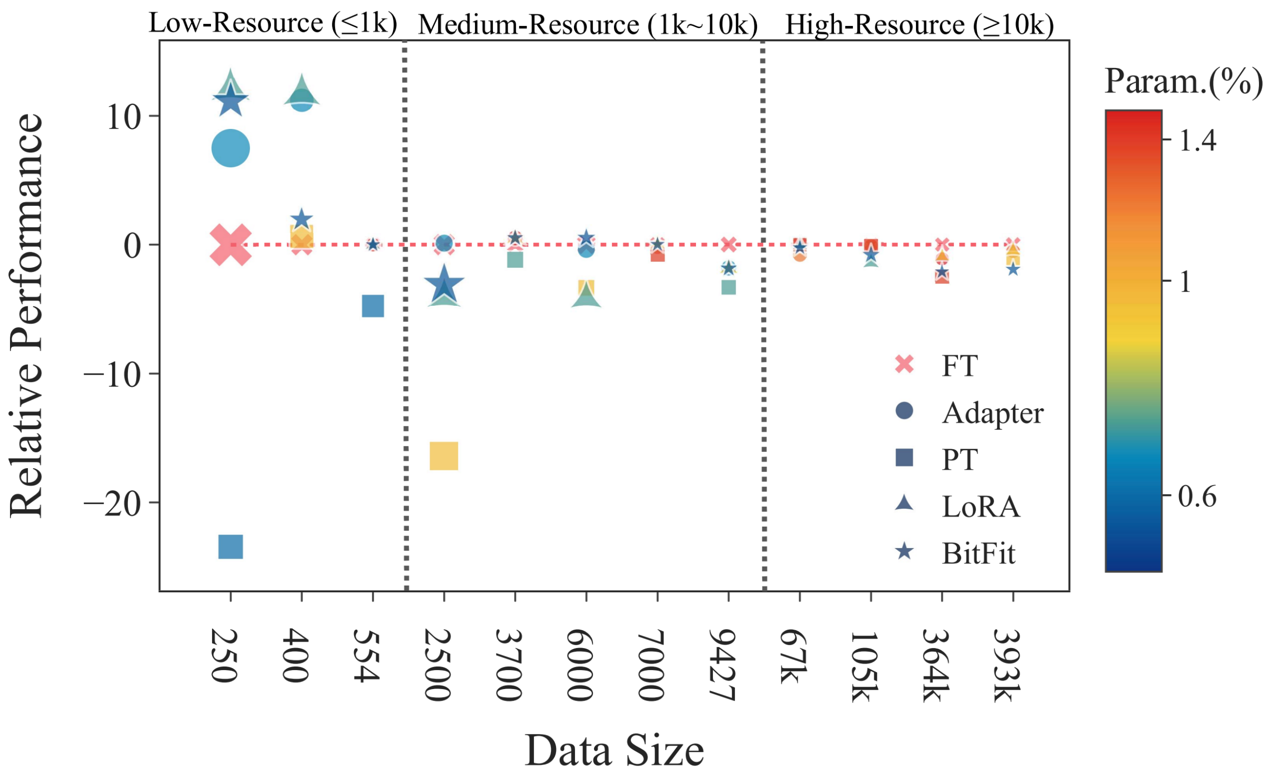


Each rescaling vector learnt using a small fully-connected module

Categorised as an “additive” PEFT method

PEFT Performance

Revisiting Parameter-Efficient Tuning: Are We Really There Yet?
Guanzheng Chen, Fangyu Liu, Zaiqiao Meng, Shangsong Liang (2022)
https://arxiv.org/abs/2202.07962



- FT (red dashed line) = full fine-tuning performance, different shapes = different PEFT methods
- Low data (resource) regime: PEFT may out-perform full fine-tuning

HuggingFace PEFT

- https://huggingface.co/docs/peft/index
- Has implementations for 5 PEFT methods (LoRA, prefix tuning, p-tuning, prompt tuning, $(IA)^3$)
- See notebook in repo: https://github.com/alan-turing-institute/transformers-reading-group/tree/main/sessions/11-peft/peft.ipynb