Parameter Efficient Fine Tuning:

Blazing fast fine-tuning!

Abbie Petulante
DSI Postdoctoral Fellow

VANDERBILT UNIVERSITY®
Data Science Institute
Discovery through data.

Training a Model

Training from scratch

Start with randomized weights

Train them fully to fit to a dataset

Fine-Tuning

Start with existing weights

Keep tuning them with additional or new data

Training a Model

Training from scratch	**Fine-Tuning**	**PEFT**
Start with randomized weights	Start with existing weights	Start with existing weights and **freeze them**
Train them fully to fit to a dataset	Keep tuning them with additional or new data	Train a much smaller set of *new* parameters

Training a Model

Training from scratch	Fine-Tuning	PEFT
Start with randomized weights	Start with existing weights	Start with existing weights and **freeze them**
Train them fully to fit to a dataset	Keep tuning them with additional or new data	Train a much smaller set of *new* parameters

Less compute needed

Fewer model adaptations

PEFT (Parameter Efficient Fine-Tuning)

- Strategies to fine tune **much** less of the model
  - Typically, <few%
  - With largely the same results!

- **<u>Core idea</u>**: large models are typically overparameterized - critical changes for new tasks reside in smaller, lower-dimensional subspaces

- Generally, most PEFT methods are **modular** (can be added/removed)

Common PEFT Methods

- **Prompt/Prefix Tuning**
- Adapters
- Low-Rank Adaptations (LoRA / DoRA)

Prompt/Prefix Tuning

<u>**Idea:**</u> Train *virtual tokens* (a small set of learned vectors) that steer the model to a desired behavior

- “Virtual tokens” - trainable vectors that *act like* tokens but aren’t in the vocabulary

- *learned representation vectors*, not weight updates

Prompt/Prefix Tuning

Method	What it learns	Where it's Added
Soft-Prompt Tuning	*k* trainable embedding vectors (no text form)	Prepended to the input embedding sequence once
Prefix Tuning	*k* learned key/value vectors per layer	Injected into each layer's attention KV-store

- Like custom "hidden context" that *doesn't need to correspond to real words*
- Changes where embedding exists in the hidden space

Common PEFT Methods

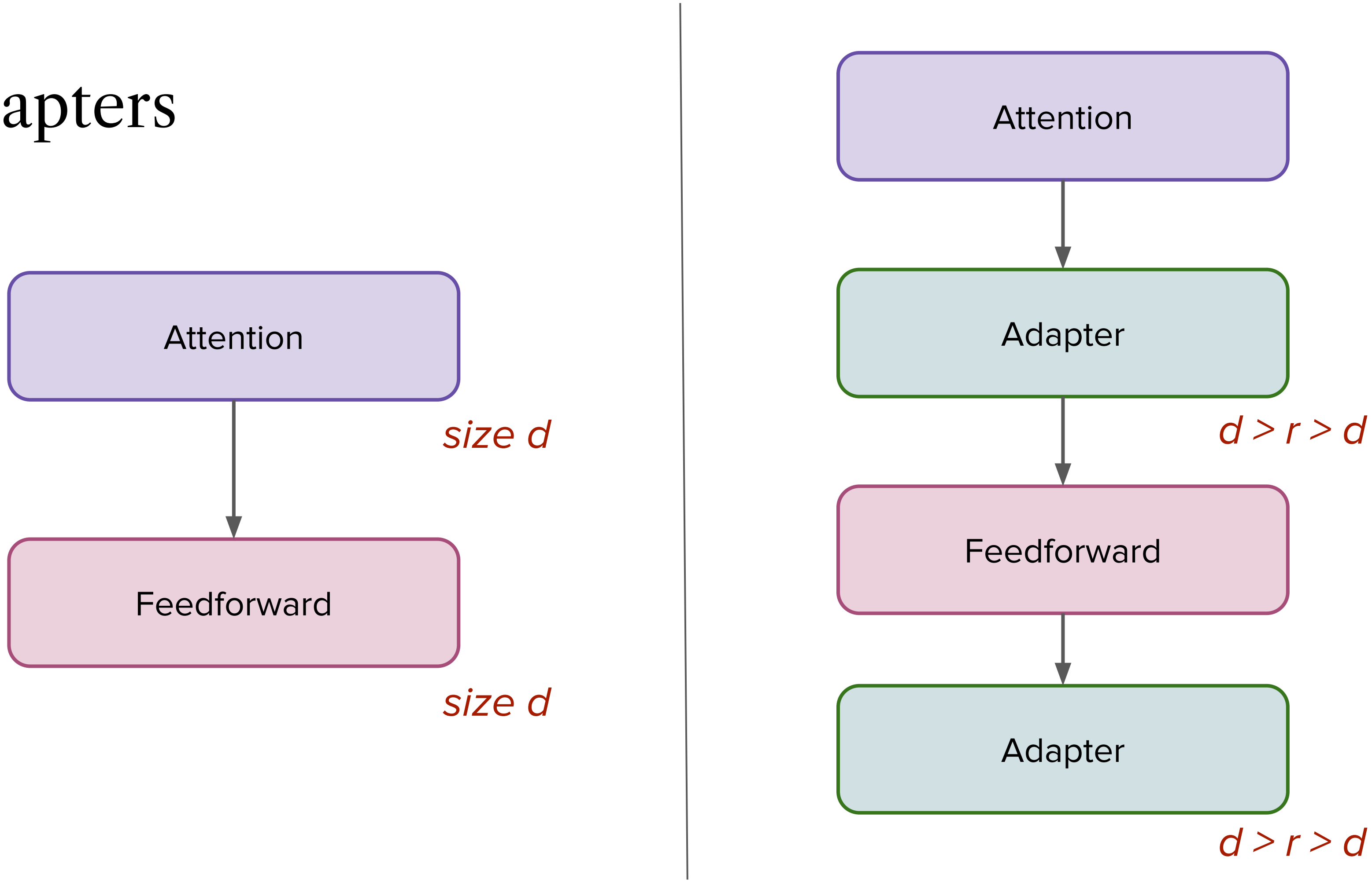
- Prompt/Prefix Tuning
- **Adapters**
- Low-Rank Adaptations (LoRA / DoRA)

Adapters

Idea: Strategically insert additional “mini” layers (modules) between existing layers of the model (like plug-ins)

- Mini layers have few weights
- Train just the adapter modules, not the original model weights
- Lightweight and modular: Adapters can be swapped out for different tasks easily

Adapters


Attention
size d
Feedforward
size d
Attention
Adapter
d > r > d
Feedforward
Adapter
d > r > d

Adapters

Variant	Placement in Transformer block	Description / use-case
Houlsby (Sequential)	After both Attention and FFN (2 per block)	Classic bottleneck MLP; best quality, still <3 % params
Post-Attention (Bapna)	After Attention only	Fewer params/latency; popular in MT
IA^3 / Pfeiffer “vector”	Multiplicative scale vectors inside each block	Ultra-light (~0.05 %); merges away at inference
Parallel adapter	Runs alongside sub-layer, outputs summed	Keeps critical path short; good for real-time
Input/Output-only	One before first layer, one after last	Simplest plug-in; lowest quality but tiny

(more common)

Adapters

```
from transformers import AutoModelForSequenceClassification, AutoTokenizer
from adapters import AutoAdapterModel

model = AutoAdapterModel.from_pretrained("bert-base-uncased")
tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")

model.add_adapter("sst2", config="houlsby", reduction_factor=16)
model.train_adapter("sst2")

#set up standard training loop, train config, etc
trainer.train()

model.save_adapter("./sst2_adapter", "sst2")
```

Common PEFT Methods

- Prompt/Prefix Tuning
- Adapters
- **Low-Rank Adaptations (LoRA + flavors)**

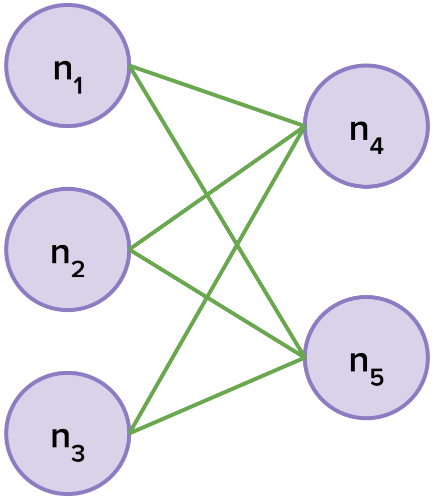
Low Rank Adaptations

Idea: Instead of updating a whole weight matrix, add a weight update that is created from much smaller, trainable matrices

- Low rank matrices = very few parameters
  - Very quick to train!
  - Original weight matrices remain frozen

Full Fine-Tuning

- All weights of the model are fair game to edit

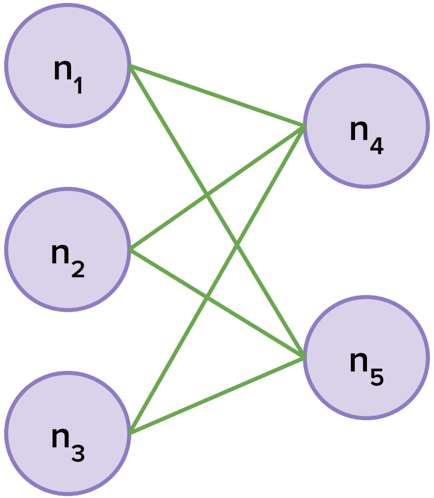


$$\begin{pmatrix} n_1 \\ n_2 \\ n_3 \end{pmatrix} \times \begin{pmatrix} W_{1,4} & W_{2,4} & W_{3,4} \\ W_{1,5} & W_{2,5} & W_{3,5} \end{pmatrix} = \begin{pmatrix} n_4 \\ n_5 \end{pmatrix}$$

All weights update, at each step, by α * update

Low Rank Adaptations (LoRA)

- Only low rank matrices A/B are tuned



$$\begin{matrix} n_1 \\ n_2 \\ n_3 \end{matrix} \times \begin{pmatrix} W_{1,4} & W_{2,4} & W_{3,4} \\ W_{1,5} & W_{2,5} & W_{3,5} \end{pmatrix} + \underset{d \times r}{\begin{bmatrix} B \end{bmatrix}} \underset{r \times d}{\begin{bmatrix} A \end{bmatrix}} = \begin{bmatrix} n_4 \\ n_5 \end{bmatrix}$$

Low-rank (r) matrices

Low-Rank Adaptations

LoRA Flavor	How Implemented	Extra Parameters
Standard LoRA (orig. Hu + al. 2021)	Add rank-r (8–32) ΔW to chosen weight matrices	$2 \cdot d \cdot r$ per target weight
QLoRA (Dettmers 23)	Freeze 4-bit quantized base weights; train LoRA in 16-bit	Same ΔW params as standard, but less VRAM
Selective-Layer LoRA	Apply LoRA only to top-N layers *or* only to Q,V matrices	Proportional to layers chosen—often < 0.05 % total
LoRA++ / Dropout-LoRA	Adds dropout & rescaling in the LoRA branch, sometimes adapter-bias	Same as standard
DoRA (Weight-Decomposed LoRA)	Decompose each weight into magnitude (γ) + direction; train γ (scalar per row/col) and a LoRA update for only direction	Same as standard + γ adds O(d) scalars per matrix