Module 2

Efficient Fine-Tuning

Doing more with less

©2023 Databricks Inc. — All rights reserved

Learning Objectives

By the end of this module you will:

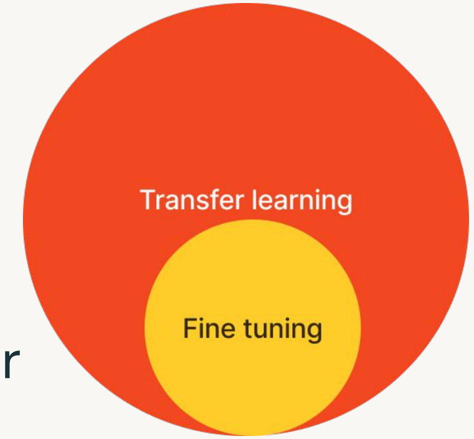
- Understand what fine-tuning is and why we do it
- Learn what parameter-efficient fine-tuning is and what the popular strategies are
- Understand the limitations of parameter-efficient fine-tuning
- Gain knowledge about data preparation best practices

©2023 Databricks Inc. — All rights reserved

Fine tuning vs. transfer learning

They are often referenced interchangeably

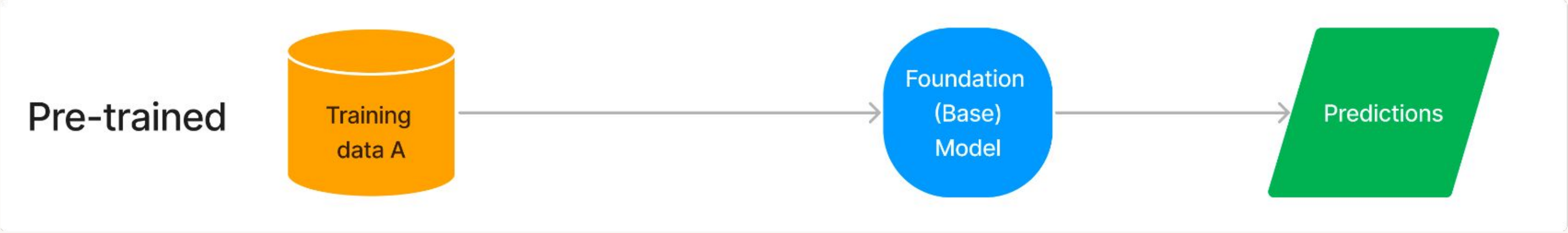
- Transfer learning
  - Apply a general pre-trained model to a new, but related task
- Fine tuning
  - Use a general pre-trained model and then train that model further
- Transfer learning ~= fine tuning
  - Train it more
  - Train on different data



©2023 Databricks Inc. — All rights reserved

Source: tensorflow.org, interview with Andrej Karpathy and Andrew Ng

How to leverage a pre-trained foundation model?

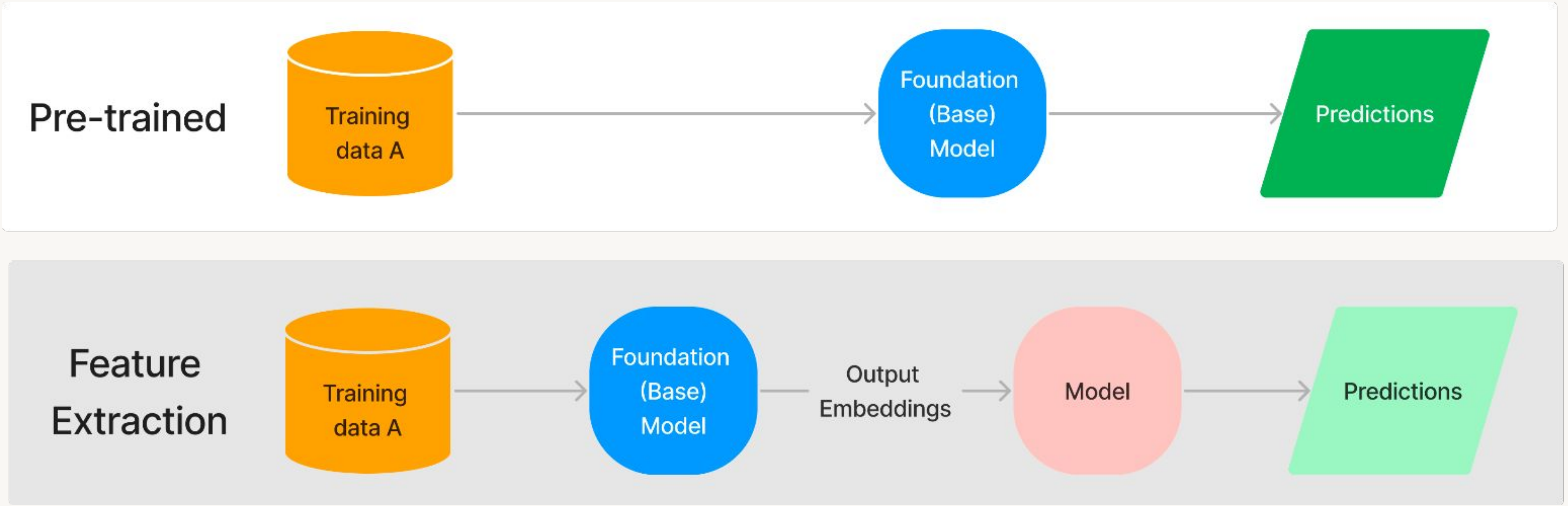


Examples:

- T5
- BloombergGPT
- GPT-4

©2023 Databricks Inc. — All rights reserved

How to leverage a pre-trained foundation model?

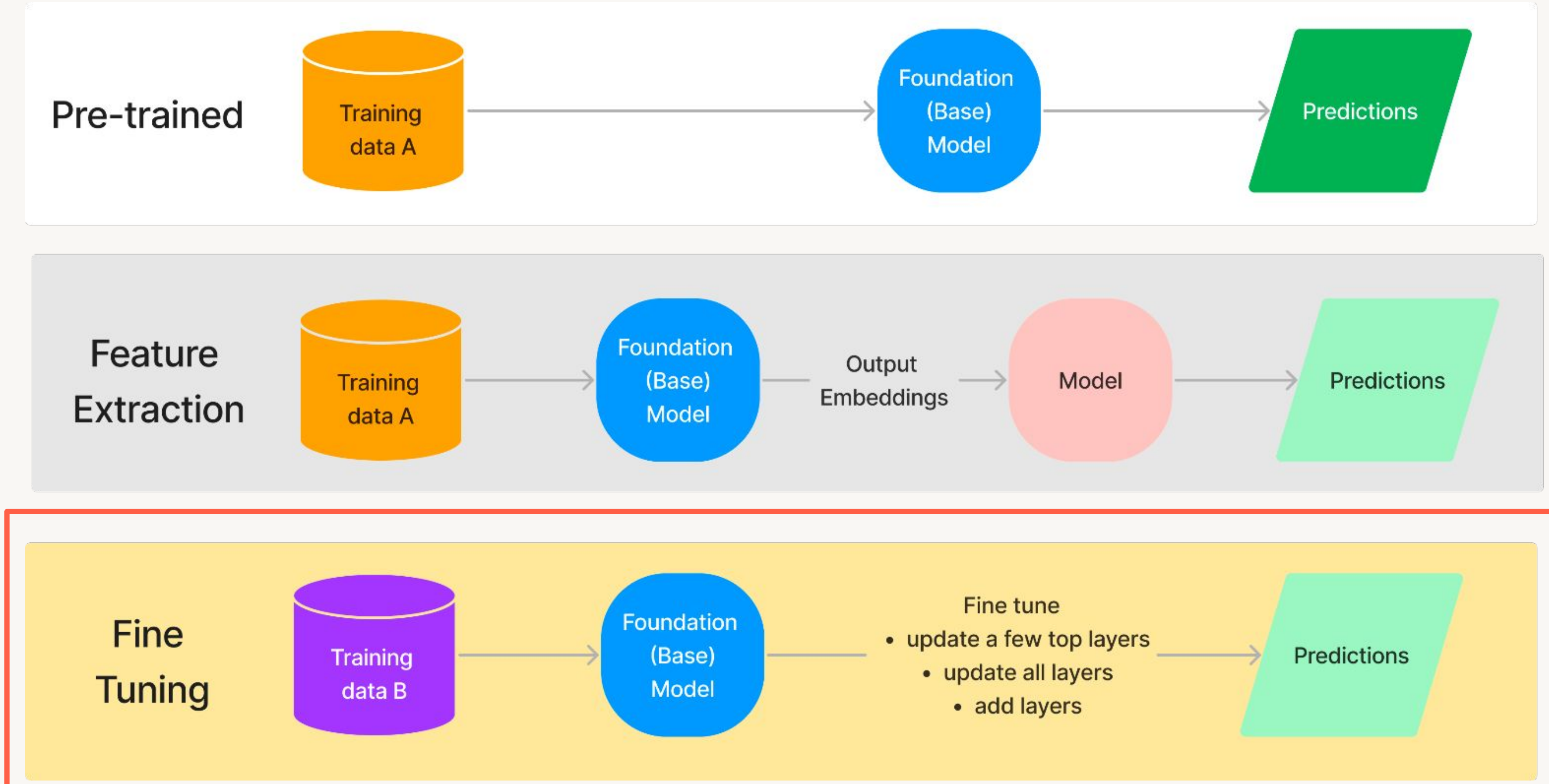


Examples:
- T5
- BloombergGPT
- GPT-4

Example:
Use BERT embeddings as inputs to a random forest classifier → classify movie reviews

©2023 Databricks Inc. — All rights reserved

How to leverage a pre-trained foundation model?



Examples:
- T5
- BloombergGPT
- GPT-4

Example:
Use BERT embeddings as inputs to a random forest classifier → classify movie reviews

Example:
- Goat: Fine tune LLaMA to perform arithmetic tasks

©2023 Databricks Inc. — All rights reserved

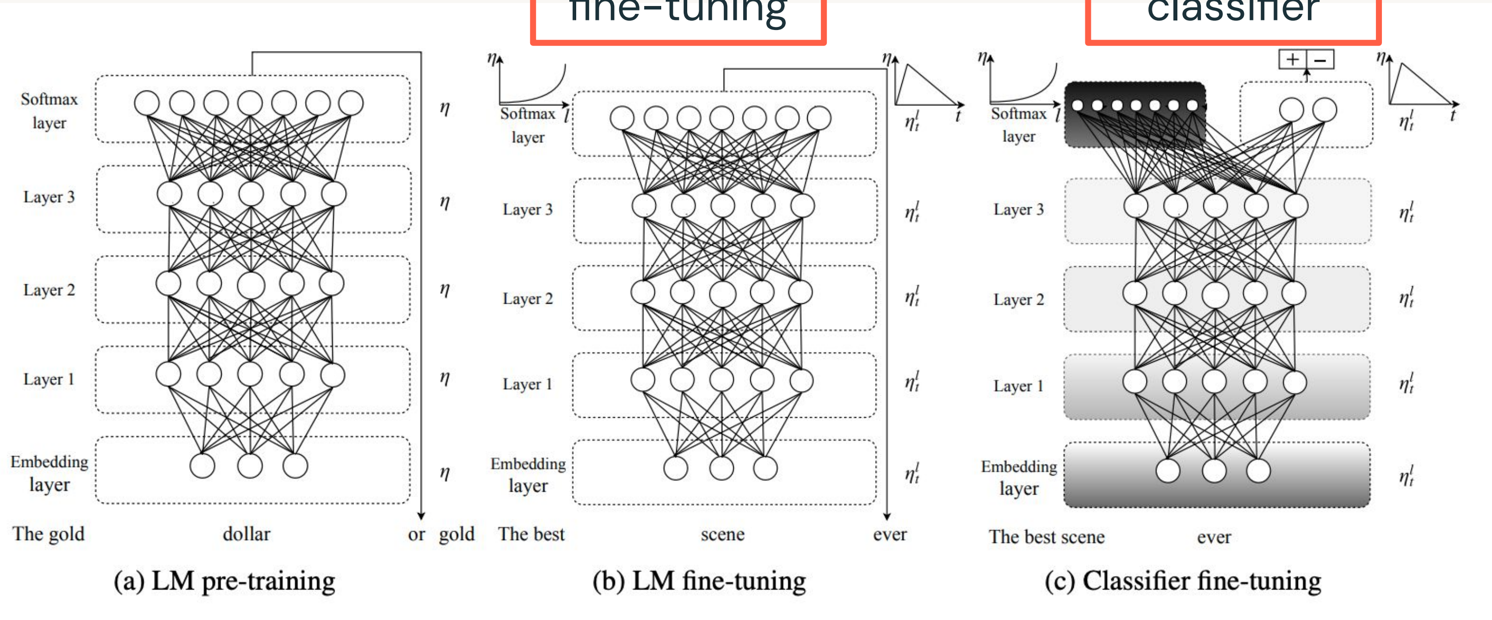
Why fine tuning?

Leverage an effective pre-trained model on our own data – it's *not* new

- Improve performance downstream
  - Different pre-trained vs fine-tuned tasks
  - Different domains
- Ensure regulatory compliance
- Not new:
  - ULMfit paper in 2018

Full fine-tuning

Fine-tune on classifier

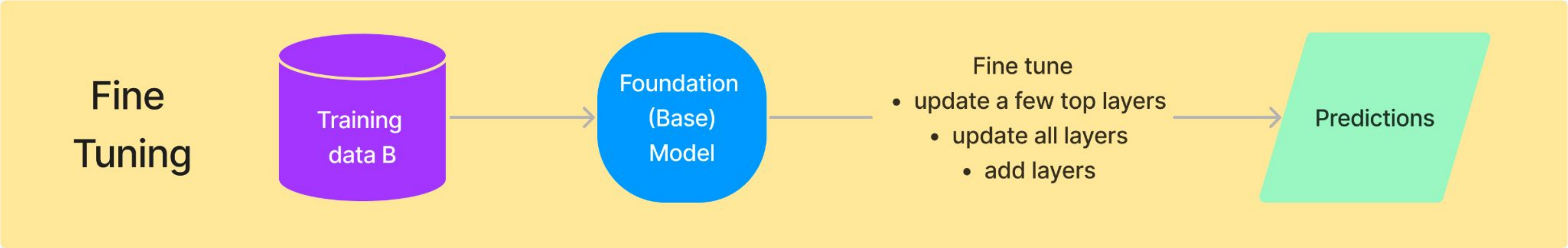


Source: Howard and Ruder 2018

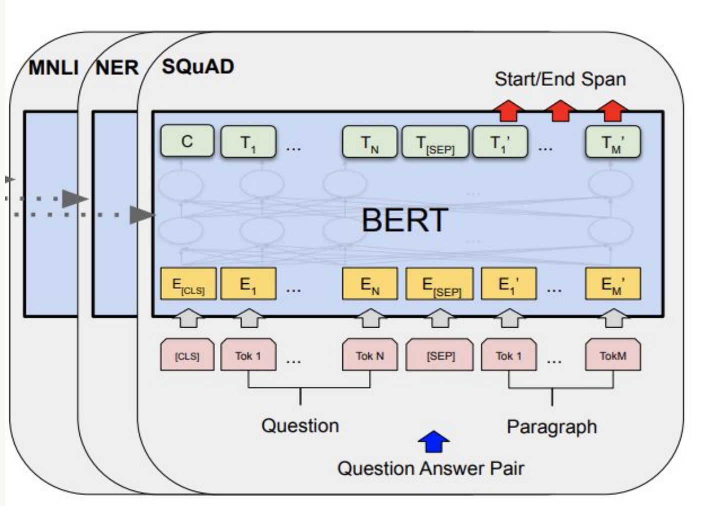
©2023 Databricks Inc. — All rights reserved

Fine tune = update foundation model weights

AKA parameter fine tuning



- Update more layers = better model performance
- Full fine-tuning typically produces one model per task
  - Serve one model per task
  - May forget other pre-trained tasks: catastrophic forgetting
- Full fine-tuning LLMs is expensive. How to avoid it?
  - X-shot learning
  - Parameter-efficient fine tuning



Fine-Tuning

Source: Devlin et al 2019

©2023 Databricks Inc. — All rights reserved

X-shot learning

Provide several examples of new tasks

Prompt engineering

= developing prompts

= prompt design

= *hard/discrete prompt tuning*

Not updating model weights

```
pipeline(
"""For each tweet, describe its sentiment:

[Tweet]: "I hate it when my phone battery dies."
[Sentiment]: Negative
###
[Tweet]: "My day has been 👍"
[Sentiment]: Positive
###
[Tweet]: "This is the link to the article"
[Sentiment]: Neutral
###
[Tweet]: "This new music video was incredible"
[Sentiment]:""")
```

Instruction

Few-shot examples

Prompt

©2023 Databricks Inc. — All rights reserved

Pros and cons of X-shot learning

Also known as in-context learning

Pros

- No need for huge labeled training data
- No need to create a copy of model for each task
    - Simplify model serving
- Text prompts feel interpretable

Cons

- Manual prompt engineering
- Prompts are specific to models
- Context length limitation
    - Add more examples? Less space for instructions
    - Longer context = higher latency
    - LLMs forget middle portion
        - Liu et al 2023 (released in July)
        - Longer context window is not the solution!
- **Performance might still be lackluster**

©2023 Databricks Inc. — All rights reserved

Fine-tuning outperforms X-shot learning

Example: GOod at Arithmetic Tasks (Goat-7B)

- Foundation model = Llama
- Trained on 1M synthetic data samples
- Accuracy outperforms
  - Few-shot PaLM-**540B** *(much bigger model !!)*
  - GPT-4
    - Typically doesn't do well in arithmetics; accuracy ~0
- SOTA on arithmetic benchmark (BIG-bench)
- Supervised **instruction fine-tuning**
  - Trained using LoRA on a 24GB VRAM GPU
    - LoRA to be covered soon!

Goat-7B

3978640188 + 42886272 =

$3978640188 + 42886272 = 4021526460$

4523646 minus 67453156

$4523646 - 67453156 = -62929510$

Calculate 397 x 4429

$$
\begin{aligned}
& 397 \times 4429 \\
& = 4429 \times (300 + 90 + 7) \\
& = 4429 \times 300 + 4429 \times 90 + 4429 \times 7 \\
& = 1328700 + 398610 + 31003 \\
& = 1727310 + 31003 \\
& = 1758313
\end{aligned}
$$

What is 8914/64?

$8914 - 64 \times 100 = 8914 - 6400 = 2514$

$2514 - 64 \times 30 = 2514 - 1920 = 594$

$594 - 64 \times 9 = 594 - 576 = 18$

Therefore, $8914 \div 64 = 139$ R 18

Image Source

Source: Liu and Low 2023
(released in May)

©2023 Databricks Inc. — All rights reserved

Important observations about Goat

- Instruction fine-tuned
- Multi-task serving

Task 1
Addition

Goat-7B

3978640188 + 42886272 =
$3978640188 + 42886272 = 4021526460$

Task 2
Subtraction, using natural language (NL)

4523646 minus 67453156
$4523646 - 67453156 = -62929510$

Task 3
Multiplication, mix of NL + mathematical symbols

Calculate 397 x 4429

$$
\begin{aligned}
& 397 \times 4429 \\
& = 4429 \times (300 + 90 + 7) \\
& = 4429 \times 300 + 4429 \times 90 + 4429 \times 7 \\
& = 1328700 + 398610 + 31003 \\
& = 1727310 + 31003 \\
& = 1758313
\end{aligned}
$$

Task 4
Division, mix of NL + mathematical symbols

What is 8914/64?

$8914 - 64 \times 100 = 8914 - 6400 = 2514$
$2514 - 64 \times 30 = 2514 - 1920 = 594$
$594 - 64 \times 9 = 594 - 576 = 18$
Therefore, $8914 \div 64 = 139$ R 18

©2023 Databricks Inc. — All rights reserved

Instruction-tuned, multi-task LLM

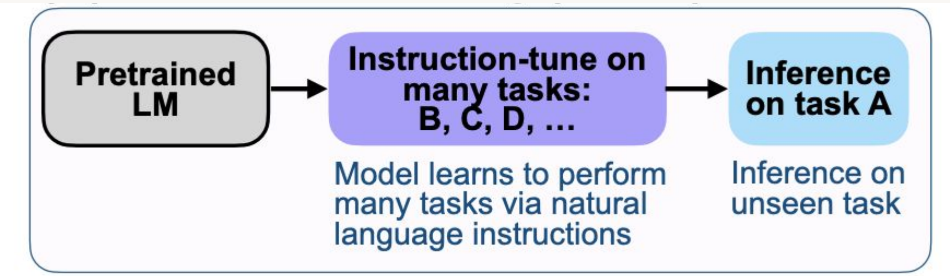
Instruction-tuned = tune general purpose LLMs to follow instructions

FLAN (Fine-tuned LAnguage Net)

- Foundation model = 137B model
- Instruction-tuned on over 60 NLP datasets with different task types
  - Task types: Q/A, translation, reasoning, comprehension, etc.
- Examples
  - T5 -> FLAN-T5
  - PaLM -> FLAN-PaLM

Dolly

- Foundation model = Pythia-12B
- Instruction-tuned on 15k prompt/response pairs
  - Task types: Q/A, classification, information extraction, etc.



Source: Wei et al 2022

©2023 Databricks Inc. — All rights reserved

Quick recap

We want efficient training, serving, and storage

- Full fine-tuning can be computationally prohibitive
  - Memory usage: activation, optimizer states, gradients, parameters
  - This gives the best performance

- Compromise: Do *some*, but not full, fine-tuning
  - Saves cost to use low-memory GPUs

- We want multi-task serving, rather than one model per task
  - E.g. one model for Q/A, summarization, classification

*Enter **parameter-efficient** fine-tuning*

©2023 Databricks Inc. — All rights reserved

Parameter-efficient fine-tuning (PEFT)

©2023 Databricks Inc. — All rights reserved

3 categories of PEFT methods

Additive

- Soft prompt
  - Prompt tuning
  - Prefix tuning

Selective

- Akin to updating a few foundation model layers
  - BitFit
    - Only updates bias parameters
  - Diff Pruning
    - Creates task-specific “diff” vectors and only updates them

Re-parameterization

- Decompose weight matrix updates into smaller-rank matrices
  - LoRA

©2023 Databricks Inc. — All rights reserved

We will cover additive and reparameterization

Additive

- Soft prompt
  - Prompt tuning
  - Prefix tuning

Selective

- **Model quality performance is not as good**
- Akin to updating a few foundation model layers
  - BitFit
    - Only updates bias parameters
  - DiffPruning
    - Creates task-specific “diff” vectors and only update them

Re-parameterization

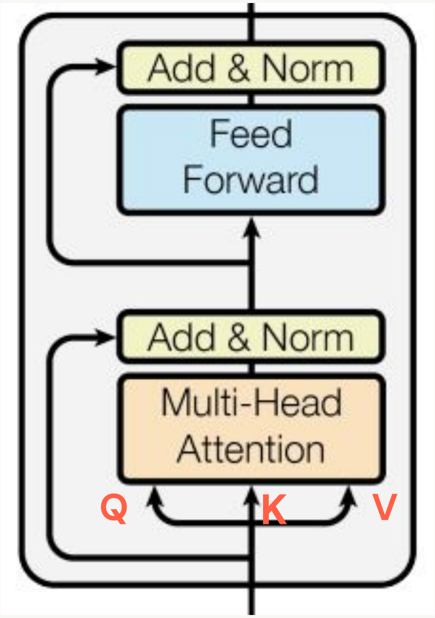
- Decompose weight matrix into smaller-rank matrices
  - LoRA

©2023 Databricks Inc. — All rights reserved

High-level overview of PEFT

Active research area: >100 papers in last few years!

- Additive: Add new tunable layers to model
  - Keep the foundation model weights frozen and update only the new layer weights
- Reparameterization: Decompose a weight matrix into lower-rank matrices
- Implementation:
  - Acts on the core Transformer block
    - Basic multi-head attention and/or feed forward network
  - Some act specifically on the weight matrices: `Query`, `Key`, `Value`
    - These matrices pass information from one token to another



Source: [Vasmani et al 2021]

©2023 Databricks Inc. — All rights reserved

Additive: Prompt Tuning

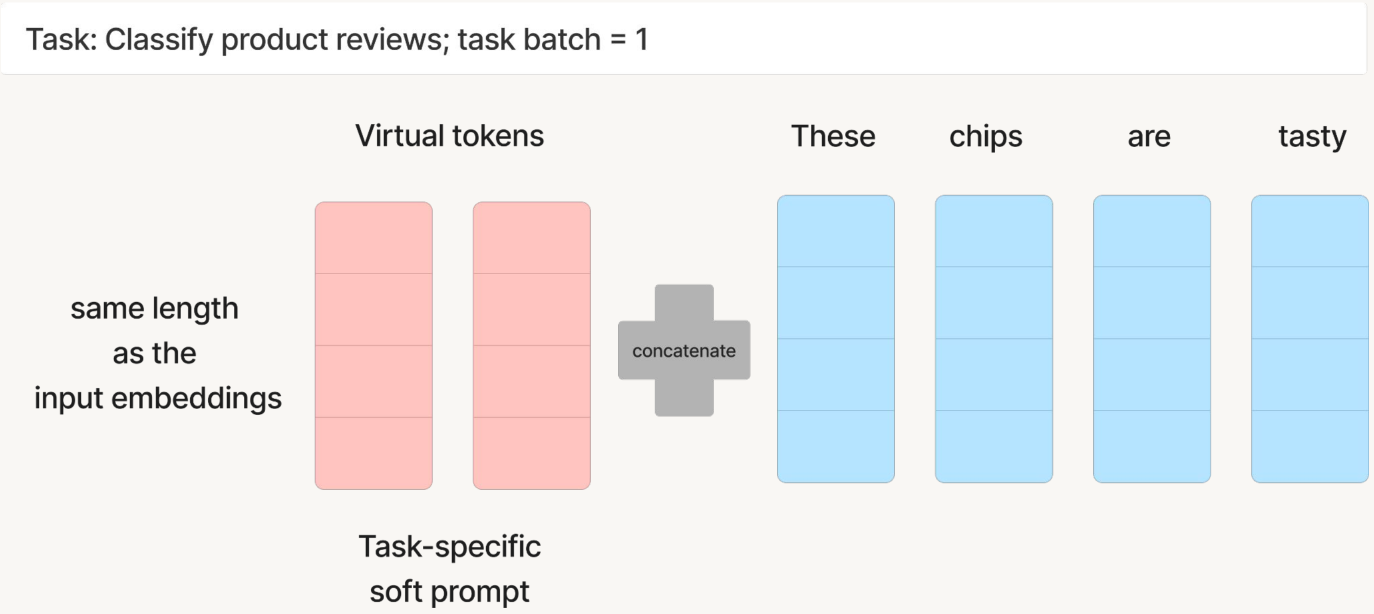
(and prefix tuning)

©2023 Databricks Inc. — All rights reserved

Soft prompt tuning

Concatenates trainable parameters with the input embeddings

- Learn a new sequence of task-specific embeddings
- We call this prompt tuning, not model tuning, because we only update prompt weights



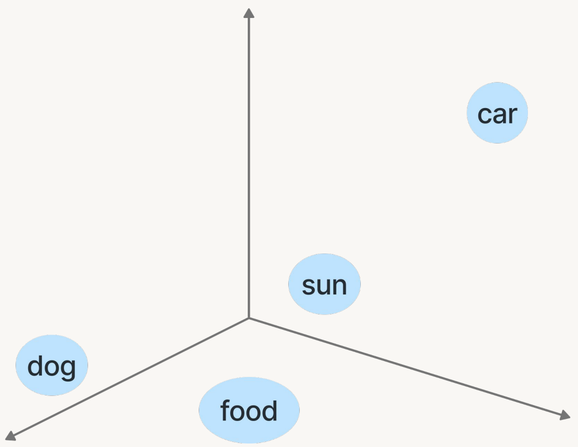
©2023 Databricks Inc. — All rights reserved

What are these *virtual* tokens?

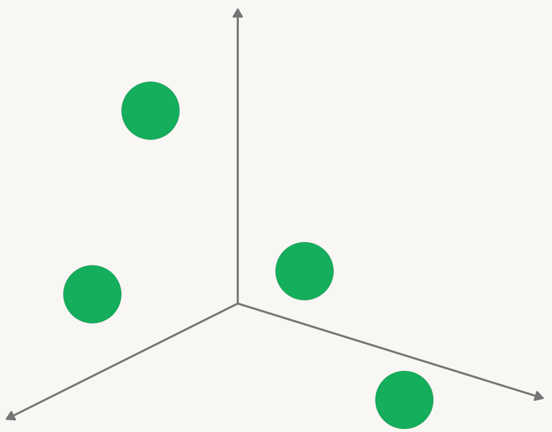
Goal: remove manual element of engineering prompts!

- Randomly initialized embedding vectors
- Not part of vocabulary
- Analogy:
  - Bitcoin: We can't touch it like cash. We don't know how it "looks", but it exists and works.

Real input tokens in embedding space



Virtual tokens in embedding space



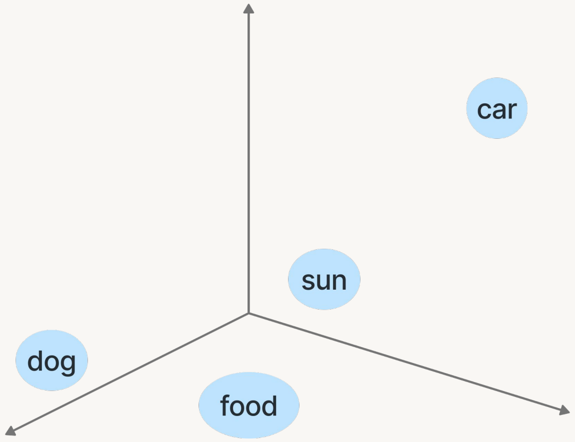
©2023 Databricks Inc. — All rights reserved

What are these *virtual* tokens?

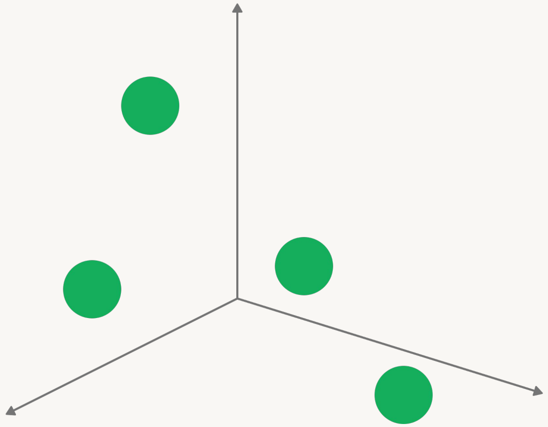
Goal: remove manual element of engineering prompts!

- Randomly initialized embedding vectors
  - We can also initialize to discrete prompts
  - But random initialization is nearly as good as informed initialization (Qin and Eisner 2021)
- Not part of vocabulary
- Analogy:
  - Bitcoin: We can't touch it like cash. We don't know how it "looks", but it exists and works.

Real input tokens in embedding space



Virtual tokens in embedding space



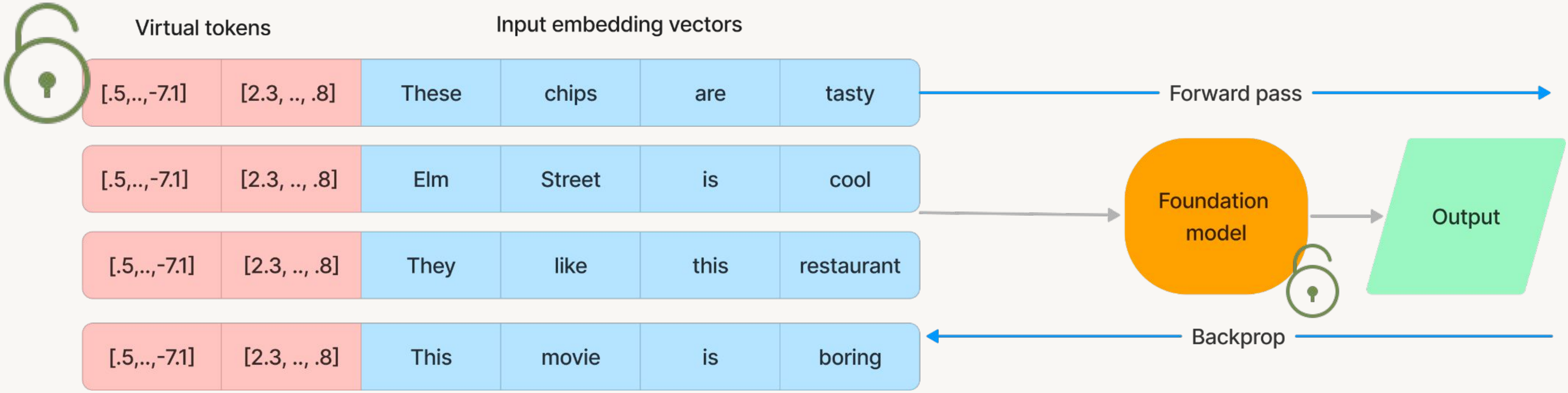
©2023 Databricks Inc. — All rights reserved

Compare full fine-tuning vs prompt tuning

Scenario: full fine-tuning

Backprop: update **all** weights based on loss

Task: Classify sentiment; task batch = 4



©2023 Databricks Inc. — All rights reserved

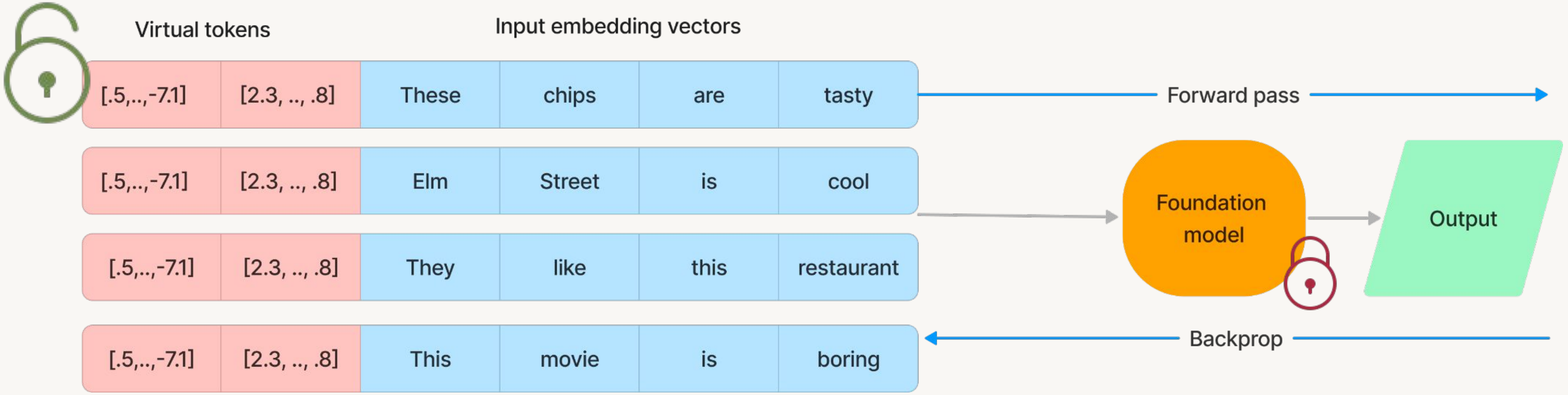
Compare full fine-tuning vs prompt tuning

Scenario: prompt tuning

Backprop: update **only prompt** weights based on loss

- The model learns the optimal representation of the prompt automatically

Task: Classify sentiment; task batch = 4

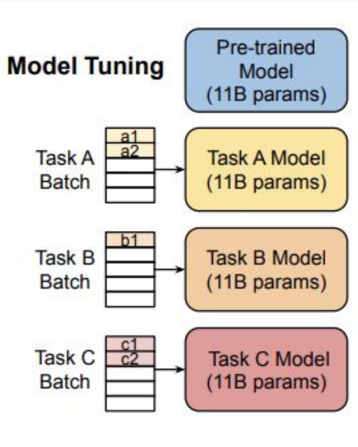


©2023 Databricks Inc. — All rights reserved

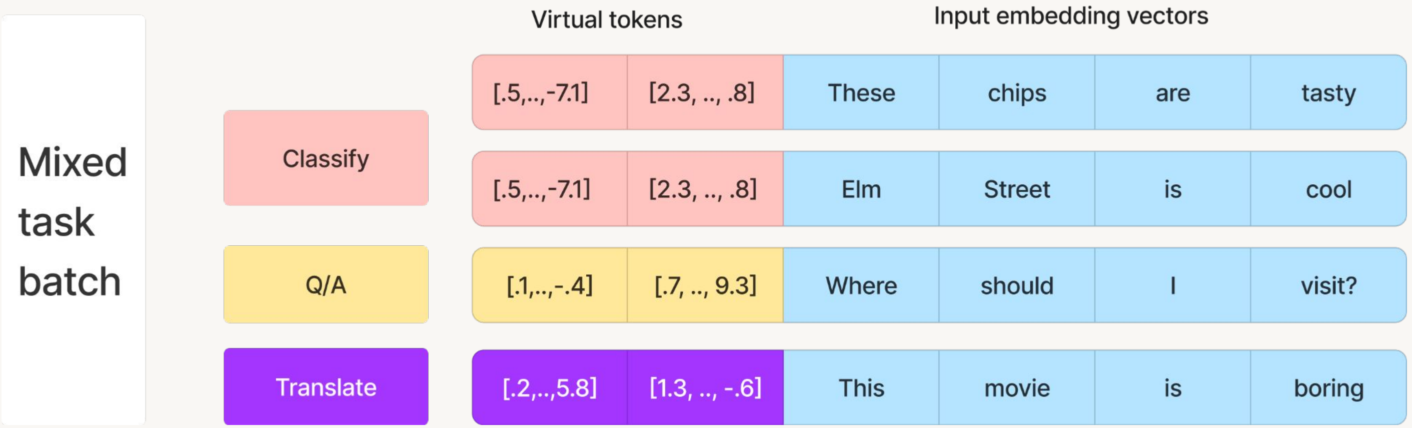
Allows swapping of task prompts

Efficient for multi-task serving

- Each task is a prompt, not a model
  - Only need to serve a single copy of the frozen model for multi-task serving
- Prompts for various tasks can be applied to different inputs
  - A serving request can be a single, larger mixed task batch



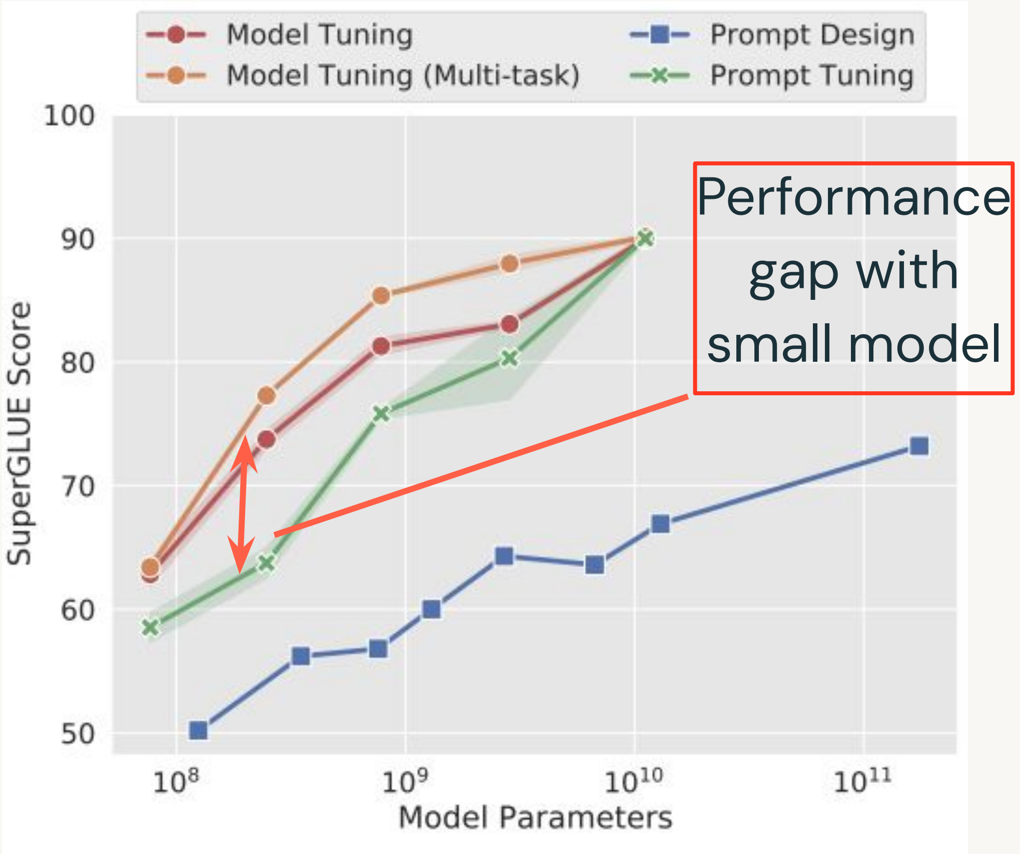
Source: [Lester et al 2021]()



©2023 Databricks Inc. — All rights reserved

Matches fine tuning performance for >11B model

- Comparable with full fine-tuning at the 10B model scale
- More applicable to larger models
- SuperGLUE (2019)
  - Styled after GLUE, but more difficult and diverse
  - Boolean questions, comprehension, etc.

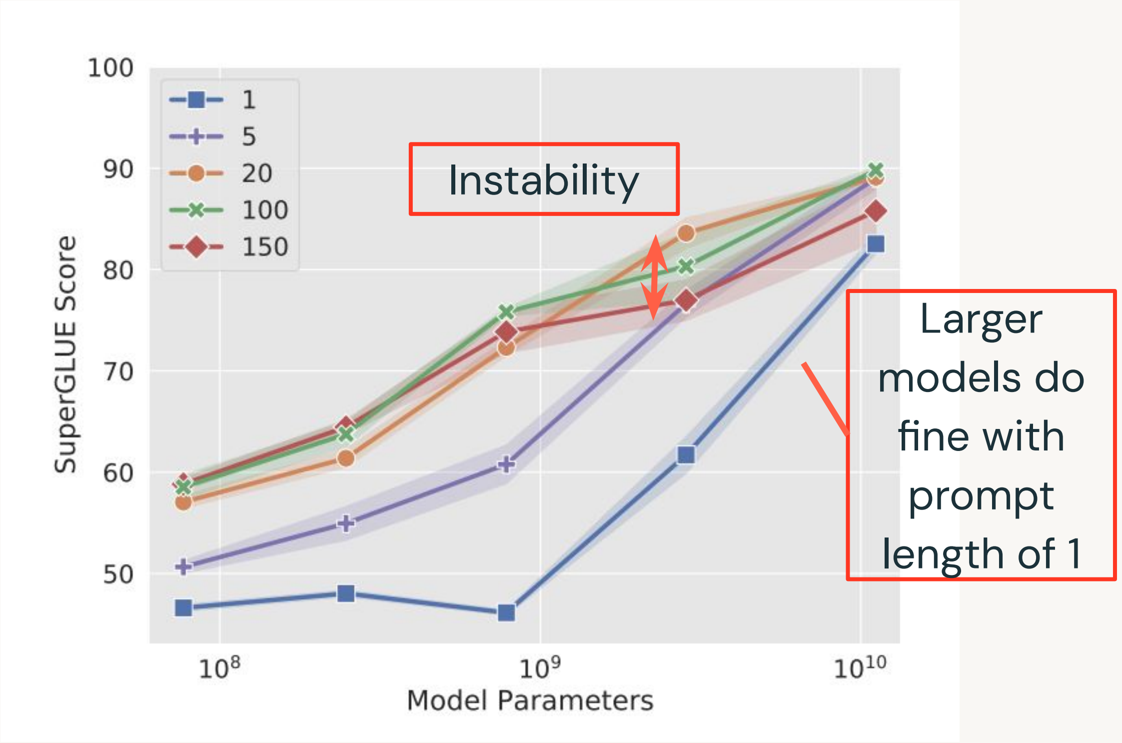


Source: Lester et al 2021

©2023 Databricks Inc. — All rights reserved

Prompt length affects larger models less

Prompt length of 20-100 is typical



Source: Lester et al 2021

In this example:

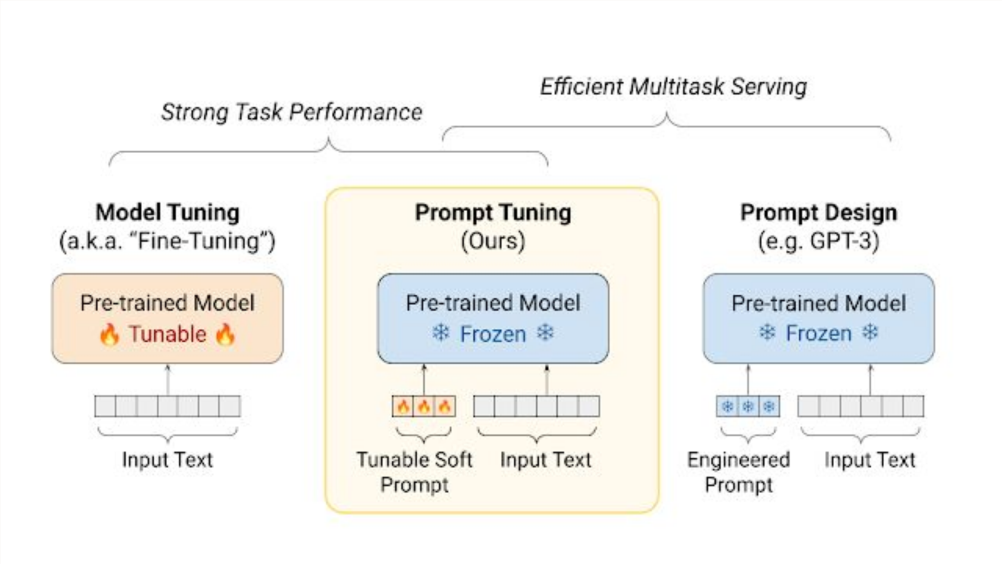
- (Virtual) prompt length = 2

Virtual tokens		Input embedding vectors			
[.5,..,-7.1]	[2.3, .., .8]	These	chips	are	tasty
[.5,..,-7.1]	[2.3, .., .8]	Elm	Street	is	cool
[.1,..,-.4]	[.7, .., 9.3]	Where	should	I	visit?
[.2,..,5.8]	[1.3, .., -.6]	This	movie	is	boring

©2023 Databricks Inc. — All rights reserved

Advantages of prompt tuning

- Use whole training set
  - Not limited by # of examples that can fit in the context
- Automatically learn a new prompt for a new model
  - Backprop helps us find the best representation
- One foundation model copy only
- Resilient to domain shift



Source: Google AI Blog

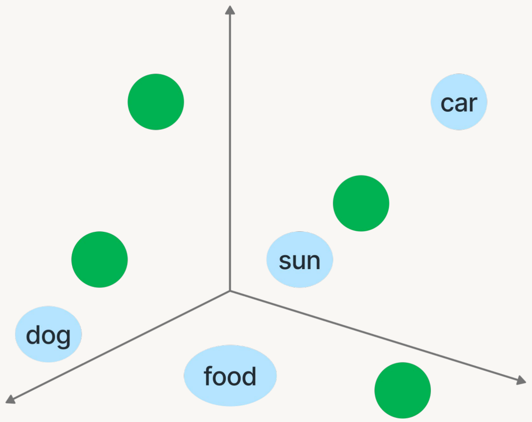
©2023 Databricks Inc. — All rights reserved

Disadvantages of prompt tuning

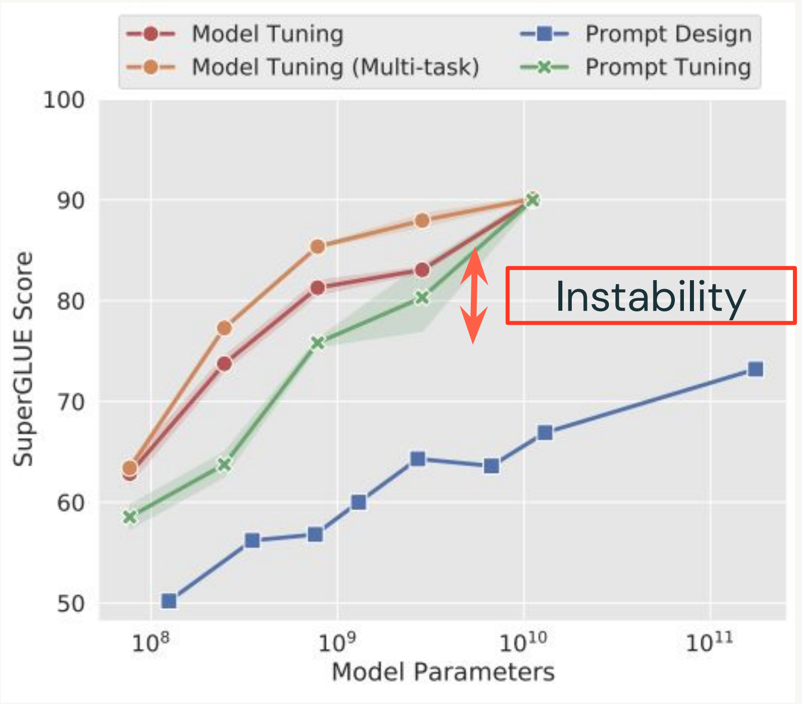
Less interpretable

- Need to convert the embeddings back to tokens
- Use cosine distance to find the top-K nearest neighbors

Find which tokens are nearest to the virtual tokens



Unstable performance

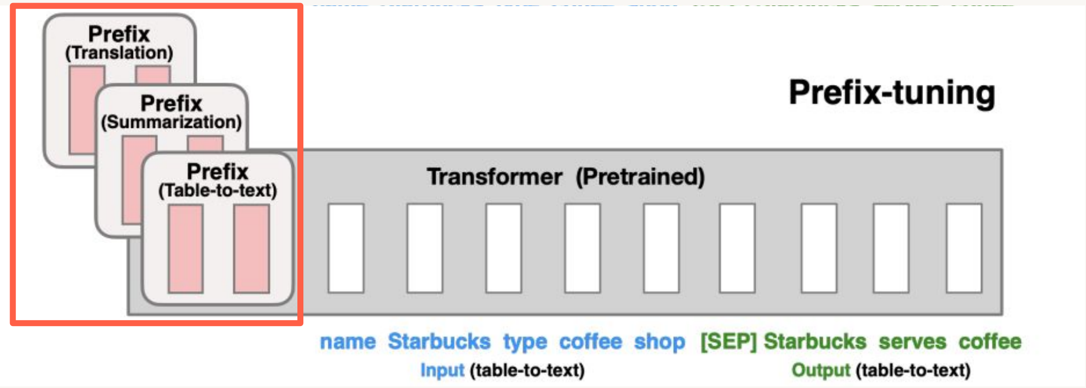


Source: [Lester et al 2021]

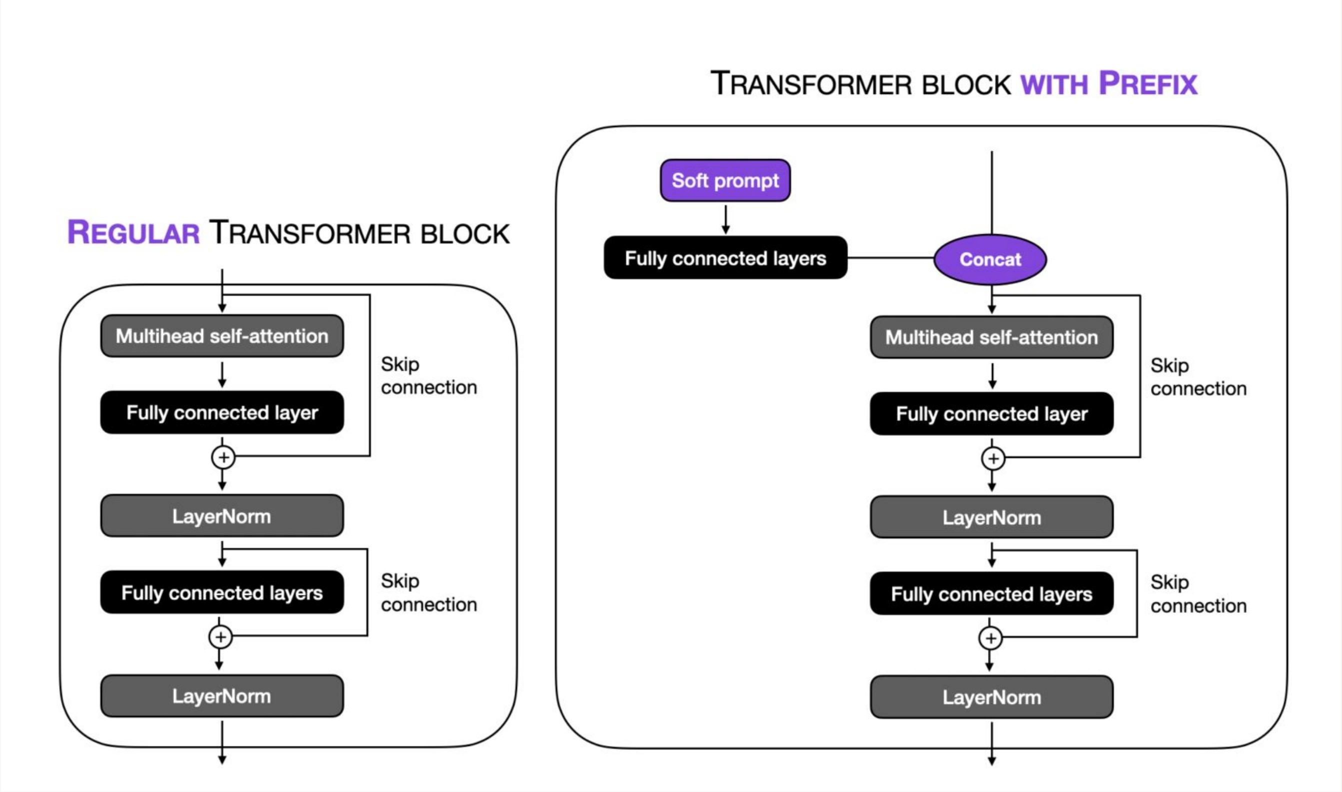
©2023 Databricks Inc. — All rights reserved

Prefix tuning is very similar to prompt tuning

Adding tunable layer to each transformer block, rather than just the input layer



Source: Li and Liang 2021



Source: Lightning AI

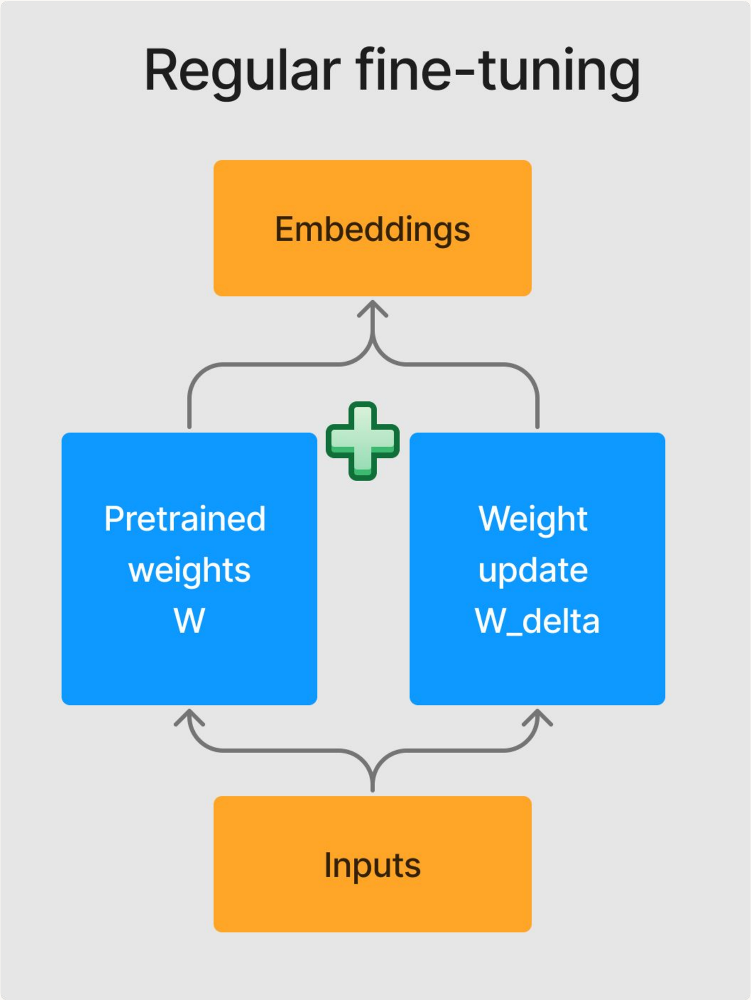
©2023 Databricks Inc. — All rights reserved

Re-parameterization: LoRA

©2023 Databricks Inc. — All rights reserved

Low-Rank Adaptation (LoRA)

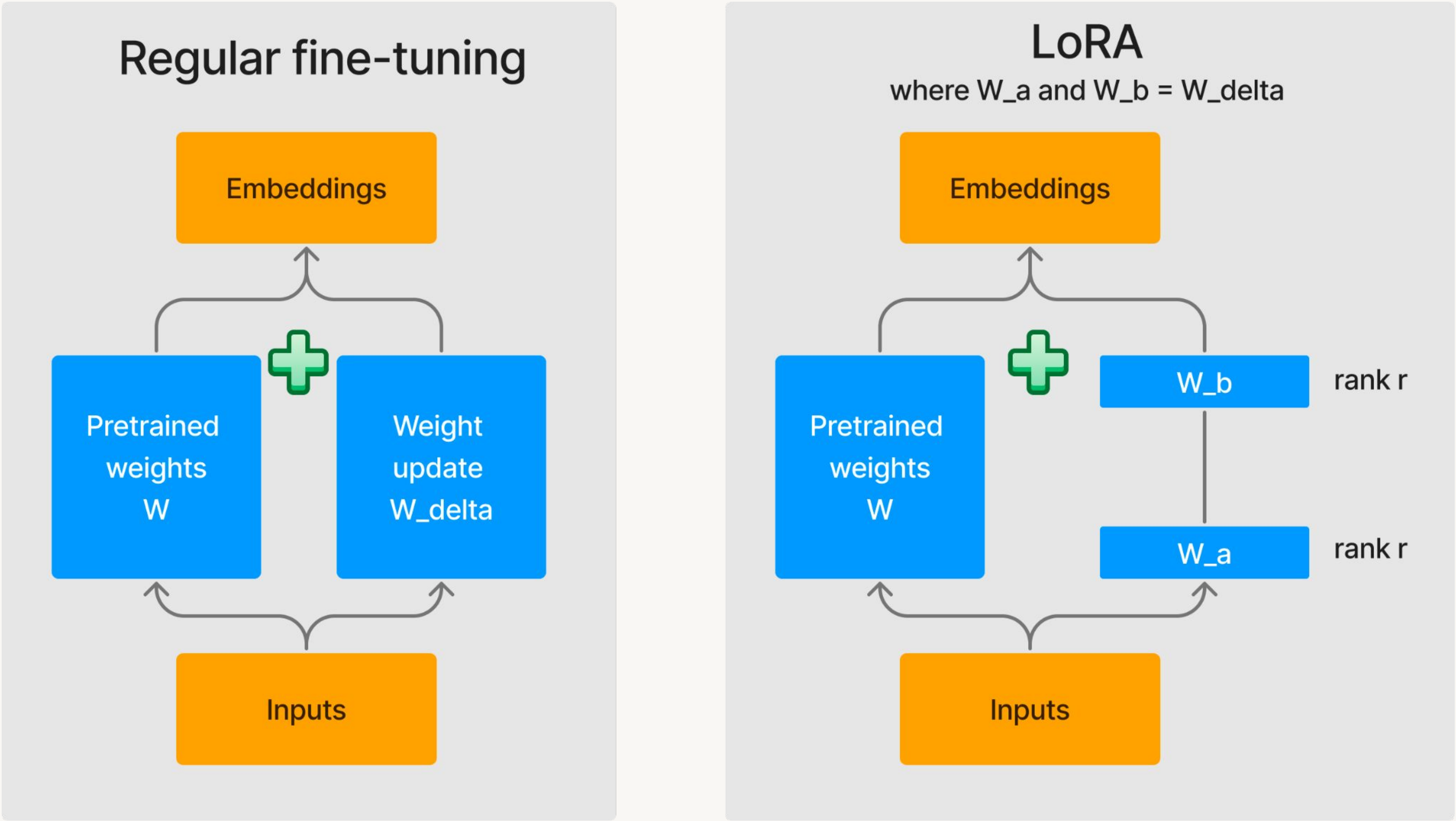
Decomposes the weight change matrix into lower-rank matrices



©2023 Databricks Inc. — All rights reserved

Low-Rank Adaptation (LoRA)

Decomposes the weight change matrix into lower-rank matrices



©2023 Databricks Inc. — All rights reserved

Rank? Brief visit to linear algebra

Maximum # of linearly independent columns or rows

- How many unique rows or columns?
- Full rank = no redundant row or column in the matrix
- Linear = can multiply by a constant
- Independence = no dependence on each other

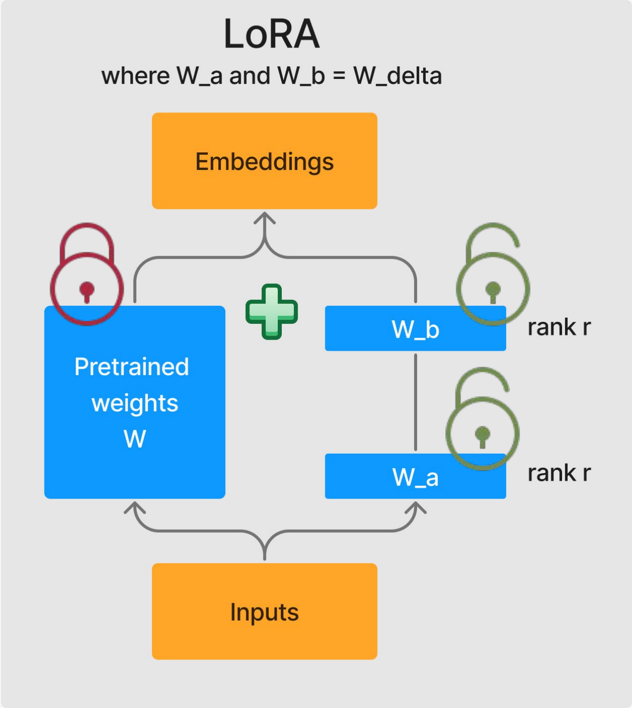
$$\begin{bmatrix} 1 & 2 & 3 \\ 3 & 6 & 9 \end{bmatrix}$$

- Row rank: 1
  - 2nd row = 3x 1st row
- Column rank: 1
  - 2nd column = 2x 1st column
  - 3rd column = 2x 2nd column

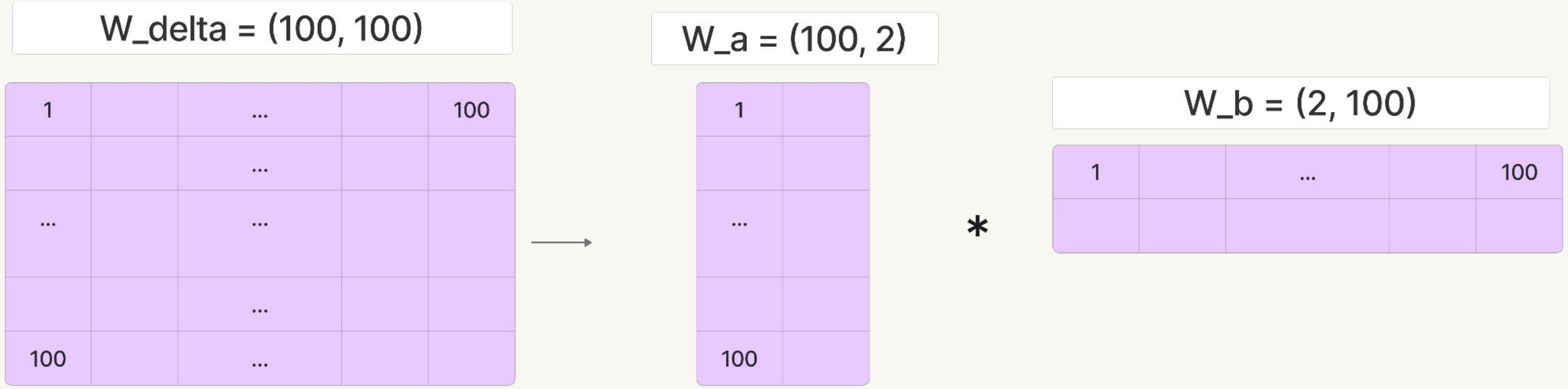
©2023 Databricks Inc. — All rights reserved

How does weight matrix decomposition work?

Observation: Actual rank of the attention weight matrices is low



$$W_{delta} = W_a * W_b$$



W_a * W_b still gives the same shape = (100, 100)

- Total parameters = (100 x 2) + (2 x 100) = 400
- Original parameters = (100 x 100) = 10,000 parameters
- Reduction = 10,000 - 400 = 96%!

©2023 Databricks Inc. — All rights reserved

LoRA matches/~outperforms full fine-tuning

- 37.7 / 175255.8
  = 0.0002
  = 0.02% of parameters!

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 (Adapter^H)	7.1M	71.9	89.8	53.0/28.9/44.8
GPT-3 (Adapter^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

Rouge

Source: Hu et al 2021

©2023 Databricks Inc. — All rights reserved

LoRA performs well with very small ranks

GPT-3's validation accuracies are similar across rank sizes

W_q = query

W_k = key

W_v = value

W_o = output

	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

Source: Hu et al 2021

But, small r likely won't work for all tasks/datasets.

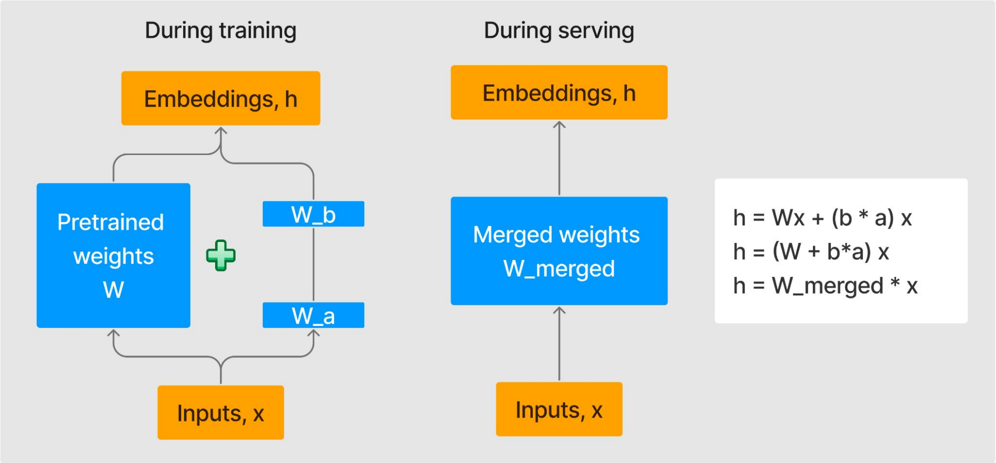
- E.g. downstream task is in a different language

©2023 Databricks Inc. — All rights reserved

Advantages of LoRA

Similar to prompt tuning, majority of the model weights are frozen

- Able to share and re-use the foundation model
  - Swap different LoRA weights for serving different tasks
- Improves training efficiency
  - Lower hardware barrier (no need to calculate most gradients or optimizer states)
- Adds no additional serving latency
  - `W_a * W_b` can be merged



- Can be combined with other PEFT methods

©2023 Databricks Inc. — All rights reserved

Limitations of LoRA

- Not straightforward to do multi-task serving
  - How to swap different combos of A and B in a **single** forward pass?
  - If dynamically choose A and B based on tasks, there is additional serving latency

- Future research
  - From LoRA authors: If W_{delta} is rank-deficient, is W too?
  - Newer PEFT technique: IA3 (2022)
    - Reduces even more trainable parameters than LoRA!

©2023 Databricks Inc. — All rights reserved

PEFT Limitations

©2023 Databricks Inc. — All rights reserved

Model performance limitations

- Difficult to match the performance of full fine-tuning
  - Sensitive to hyperparameters
  - Unstable performance
- Current research area: where is best to apply PEFT?
  - E.g. why apply PEFT to only attention weight matrices? Soft prompts?
  - Vu et al 2022: Soft prompt transfer
- We may still need full-parameter fine-tuning
  - Lv et al 2023 (released in June): use new optimizer, LOMO, to reduce memory usage to ~11%

©2023 Databricks Inc. — All rights reserved

Compute limitations

Doesn’t always make **inference** more efficient

Doesn’t reduce the cost of **storing** massive foundation models

Doesn’t reduce time complexity of **training**

Requires full forward and backward passes

©2023 Databricks Inc. — All rights reserved

Data Preparation Best Practices

©2023 Databricks Inc. — All rights reserved

Better models from better training data

Many newer good models use C4 (e.g. MPT-7B)

Llama

- Trained on 20 most-spoken languages, focusing on those with Latin and Cyrillic alphabets

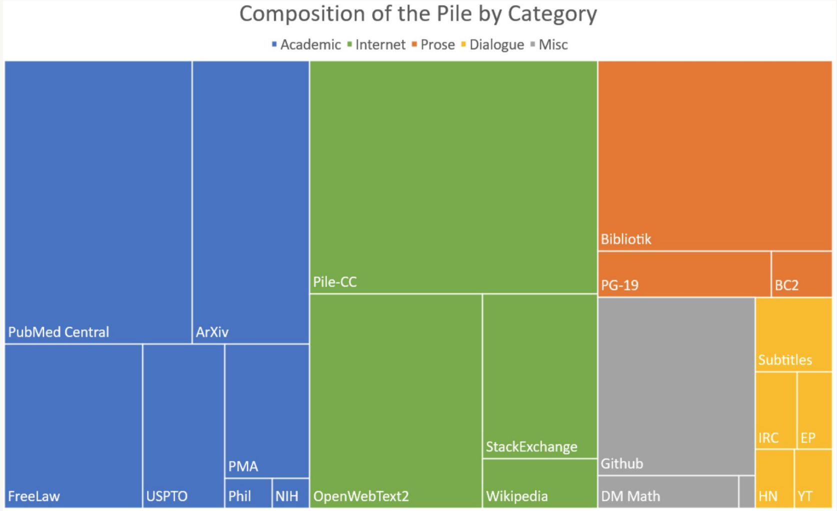
Colossal Cleaned Crawled Corpus

Dataset	Sampling prop.	Epochs	Disk size
CommonCrawl	67.0%	1.10	3.3 TB
C4	15.0%	1.06	783 GB
Github	4.5%	0.64	328 GB
Wikipedia	4.5%	2.45	83 GB
Books	4.5%	2.23	85 GB
ArXiv	2.5%	1.06	92 GB
StackExchange	2.0%	1.03	78 GB

Source: Touvron et al 2023

GPT-Neo and GPT-J

- Trained on the Pile: 22 diverse datasets
- Outperformed GPT-3 in some instances (Read more here)



Source: Gao et al 2020

©2023 Databricks Inc. — All rights reserved

Training data makes the biggest difference

Not necessarily the model architecture

- Bloomberg created 363B-token dataset of English financial documents spanning 40 years
  - Augmented with 345B-token public dataset
- Outperforms existing open models on financial tasks

Finance-Specific	**BloombergGPT**	**GPT-NeoX**	**OPT-66B**	**BLOOM-176B**
Financial Tasks	**62.51**	51.90	53.01	54.35
Bloomberg Tasks (Sentiment Analysis)	**62.47**	29.23	35.76	33.39

General-Purpose	**BloombergGPT**	**GPT-NeoX**	**OPT-66B**	**BLOOM-176B**	**GPT-3**
MMLU	**39.18**	35.95	35.99	39.13	**43.9**
Reading Comprehension	**61.22**	42.81	50.21	49.37	**67.0**
Linguistic Scenarios	**60.63**	57.18	58.59	58.26	**63.4**

Source: Wu et al 2023

©2023 Databricks Inc. — All rights reserved

How much fine-tuning data do I need?

- Zhou et al 2023 (May): fine-tune 1,000 high-**quality** labeled examples from LLaMa 65B
  - When scaling up data **quantity**, need to scale up prompt **diversity**
- OpenAI: At least a couple hundred
  - Doubling dataset size leads to linear increase in model performance
- How to get more data? Synthetic data
  - Synonym replacement / rewrite
  - Word deletion: “brilliantly expressed” => “expressed”
  - Word position swapping: “It is lovely” -> “Lovely, it is”
  - Noise injection: introduce typos

©2023 Databricks Inc. — All rights reserved

Data preparation best practices

Quantity, diversity, and quality

- Don’t provide detailed instructions. Only prompt and completion.
  - Fixed separator `\n\n###\n\n` to inform when the prompt ends and completion begins
  - The separator shouldn’t appear anywhere else

```
{"prompt": "<prompt text>", "completion": "<ideal generated text>"}
{"prompt": "<prompt text>", "completion": "<ideal generated text>"}
{"prompt": "<prompt text>", "completion": "<ideal generated text>"}
...
```

Source: OpenAI

[Manually Authored – Writing]
I am planning to start a book club with some friends. Can you write an email invitation for the same?

Here is an example email invitation for a book club:

Subject: Join me for a book club!

Hi [Friend’s Name], how are you?

I’m starting a book club with some friends, and I’d love to have you join us!

The book club is a casual and fun way to read and discuss books that we love or want to learn more about, and also make new friends. We plan to meet more-or-less once a month.

We’re thinking of having our first meeting on [Date & Time]. Could that work for you?

Would love to see you there!

Thanks,
[Your Name]

Source: Zhou et al 2023

©2023 Databricks Inc. — All rights reserved

Data preparation best practices

- Remove undesired data
  - Offensive, toxic content
  - Private or confidential information
- Using LLM output as data is not always the answer
  - Imitation models learn style, rather than content (Gudibande et al 2023)
  - Consistent with Zhou et al 2023: knowledge is largely learned during pre-training
- Manually verify data quality

©2023 Databricks Inc. — All rights reserved

Module Summary

Efficient Fine-Tuning - What have we learned?

- Fine-tuning gives the best results, but can be computationally expensive
- Parameter-efficient fine-tuning reduces # of trainable parameters
- Prompt tuning allows virtual prompts to be learned automatically
- LoRA decomposes the weight change matrix into lower-rank matrices
- Fine-tuning data quality and diversity matters a lot

©2023 Databricks Inc. — All rights reserved

Time for some code!

©2023 Databricks Inc. — All rights reserved

Course Outline

©2023 Databricks Inc. — All rights reserved