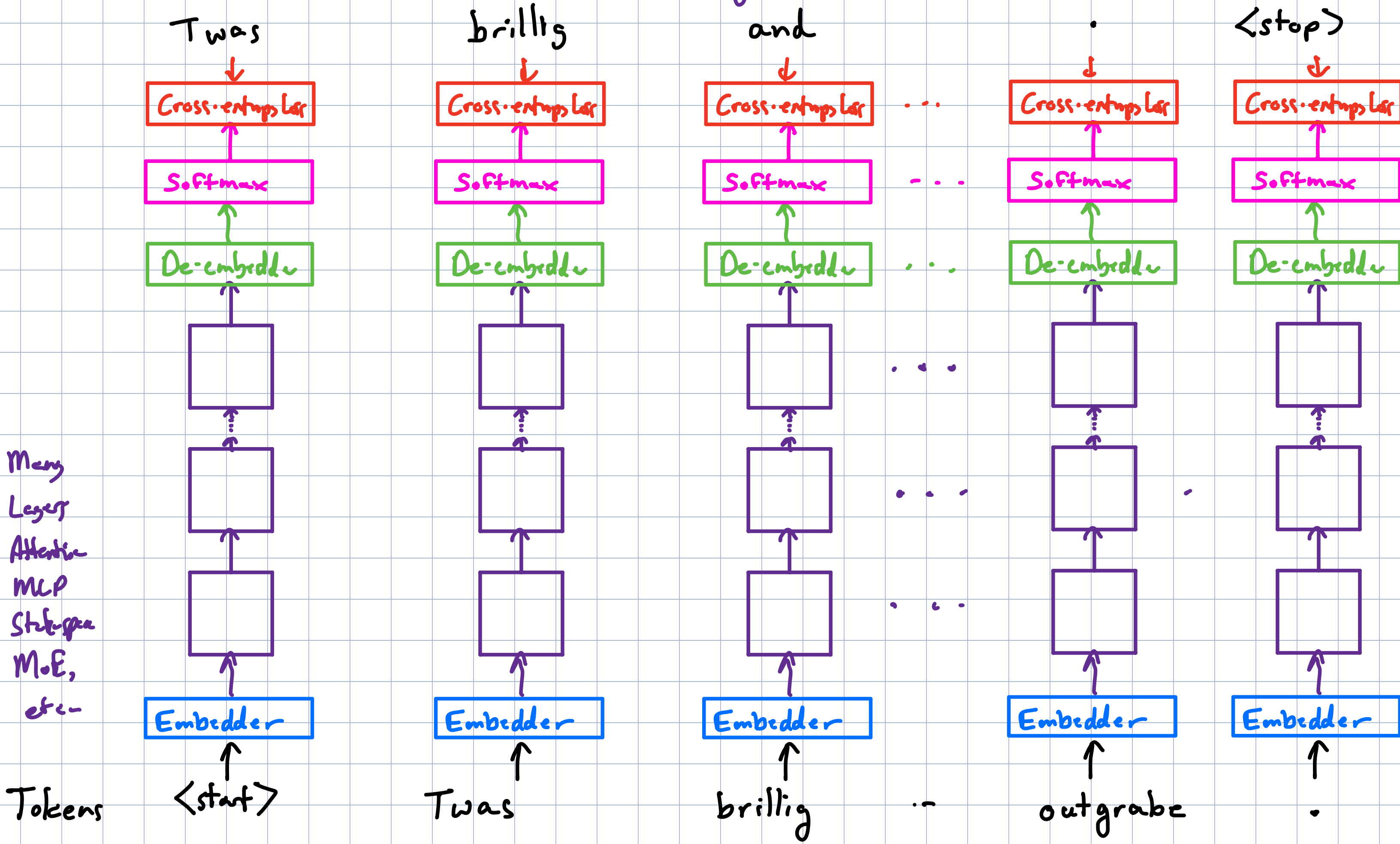
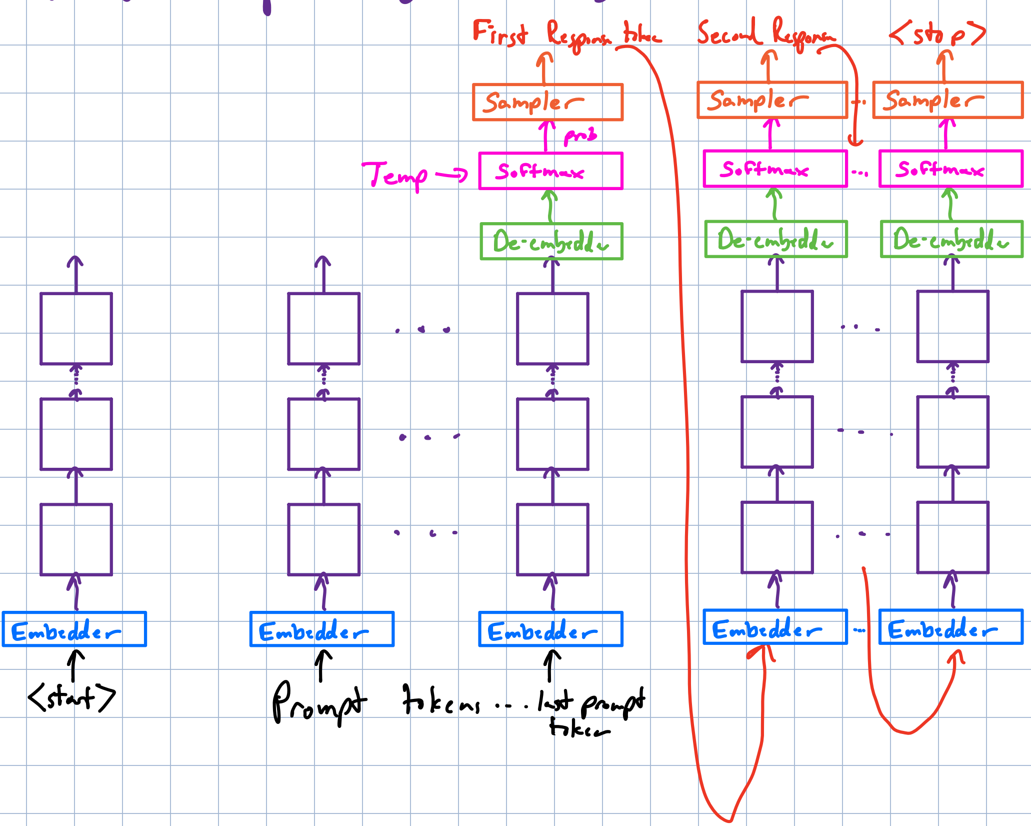
Today: Parameter-efficient fine-tuning
Transfer learning considerations
Meta-learning

Announce: Fill out survey

GPT-style models... Recall Pretraining: (Next token Prediction)



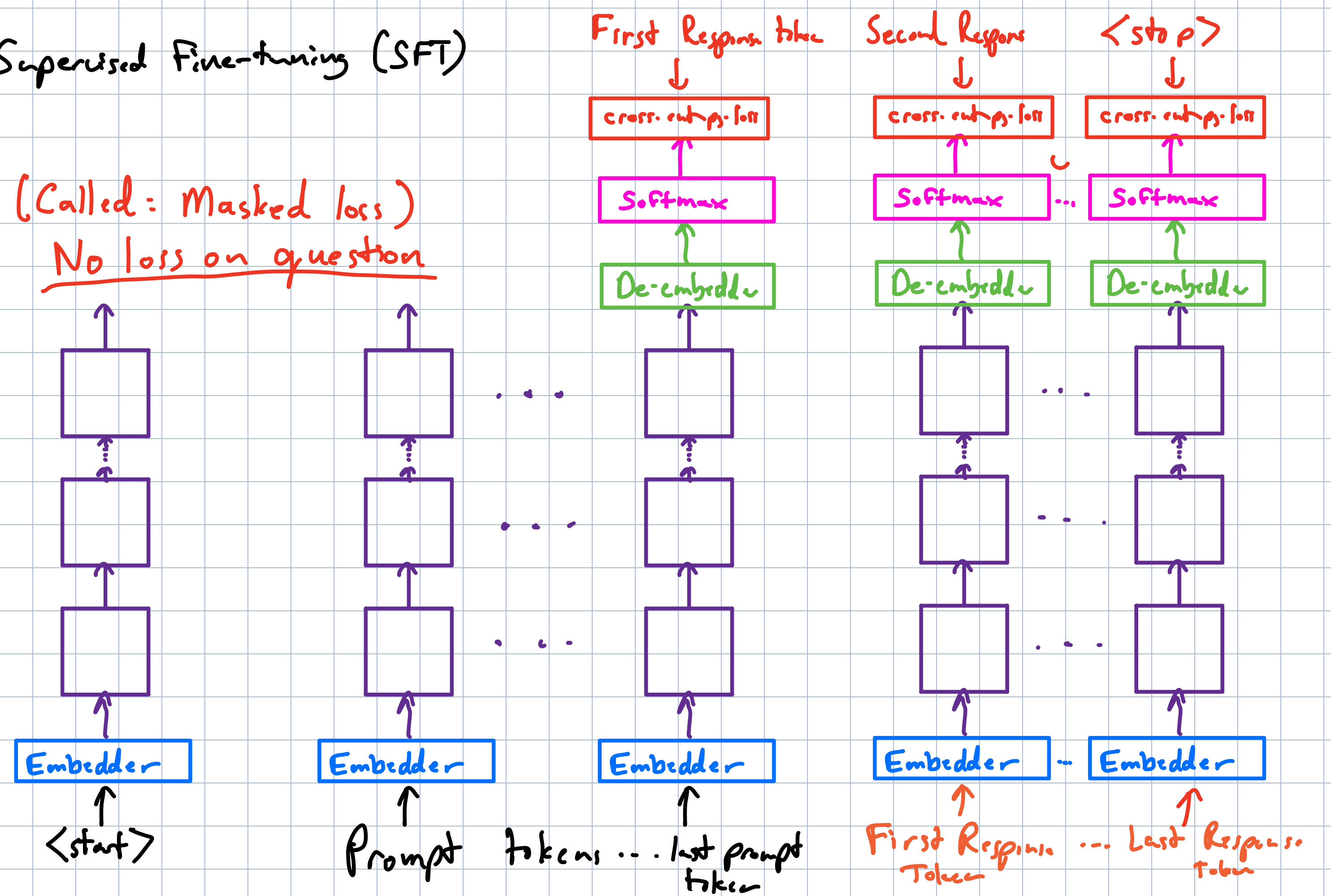
vs Inference... (Autocomplete Style Autoregressive generation)



Fine-tuning for Instruction Following or Question Answering...

Supervised Fine-tuning (SFT)

(Called: Masked loss)
No loss on question



Step Back: Fine-tuning or using a pretrained model for a new task:

0) Pure prompting — No gradient-descent steps

0.25) "DSPy-style" Prompt optimization while keeping hard prompts

0.5) Soft-prompting — Use white-box model access to train a soft-prompt
— Custom "pre-prompt" embedding that optimizes response to prompt for a given task

0.75) Soft-prefix — Like soft-prompting, but tune all $\vec{k}, \vec{v}$ in the pre-prompt segment

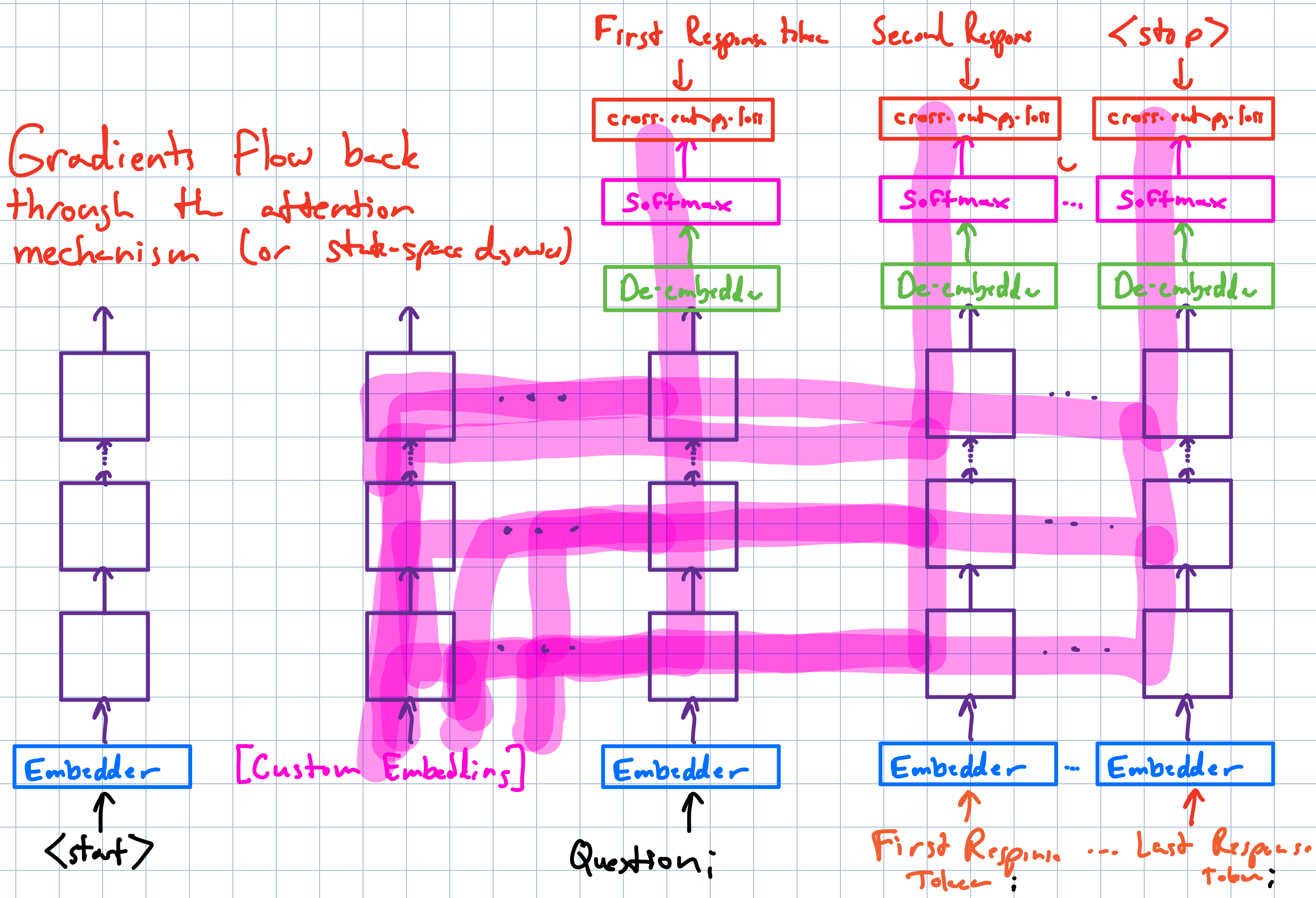
1) Treat as embedding — a feature extractor. Do classical ML to train a separate model
Also called "linear probing"

1.5) LoRA-style fine-tuning

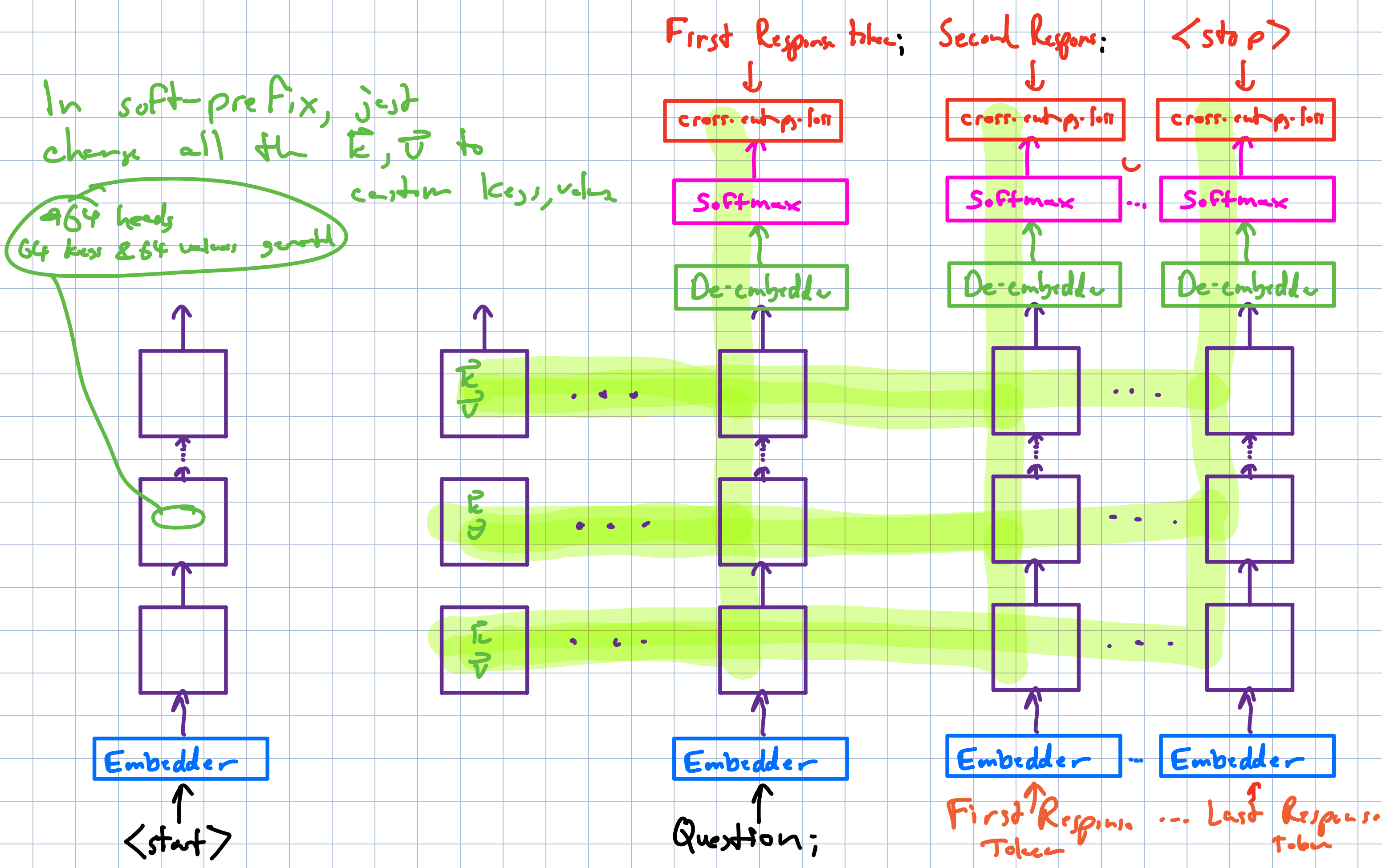
2) Full fine-tune — adjust the weights of the pretrained model

Soft-prompting and Soft-prefix

Gradients flow back through the attention mechanism (or state-space dynamics)



In soft-prefix, just change all the $\vec{K}, \vec{V}$ to custom keys, values



Soft Prompts & Prefixes work pretty well.

And far far fewer parameter than the entire model.

example for a soft-prompts: Embedding 4096 (Llama 3, Olmo 2, etc.)

Prompt-length 100

Total Params: 40K vs 7 or 8 Billion

Soft Prefix (Just the K-V cache) Assume 32 layers

$$100 \times (\underbrace{4096}_{\text{keys}} + \underbrace{4096}_{\text{values}}) \times \underbrace{32}_{\text{layers}} = 262K \times 100 = 26M$$

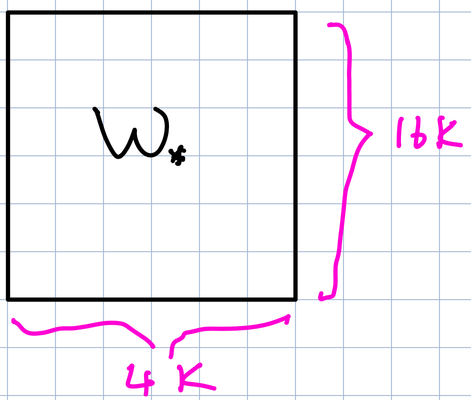
Idea: Why not do parameter-efficient fine-tuning beyond soft-prompts??

LoRA: Low Rank Adaptation

In pretrained model, weight-matrices W_* — e.g. In MLPs, ~~Attention~~ (K, V, Q, O)

Note: Embedder, De-embedder/Output layer are different.

e.g



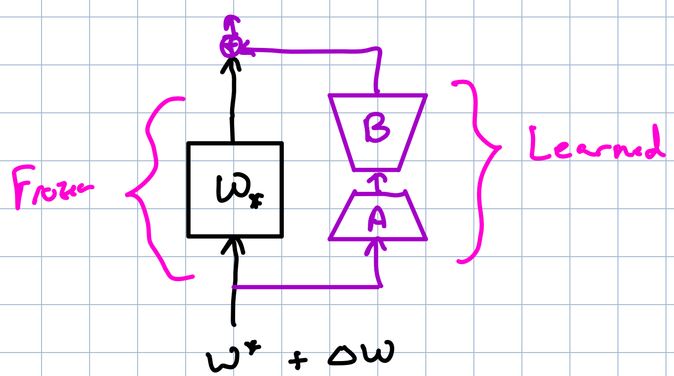
has 64M parameters

Want updates $W = W_* + \Delta W$ where ΔW is rank $r \ll 4K$

Parameterize $\Delta W = BA$ where $16k\{\overbrace{B}^{r}$ $\quad r\{\overbrace{A}^{4k}$

Note: $r \cdot 16k$ params $\quad r \cdot 4k$ params

So total is $r \cdot 4k \cdot (4+1)$ params



Best Practice Using AdamW

Initialize B to zero

Initialize A to nonzero

Use a larger learning rate for B by a factor λ (depends on problem)

Once LoRA fine-tuning done, can merge $W_* + \Delta W$ into a new W.

Initialization:

0) Init $A=0$, $B=0$ ← Why Not? Gradients are zero.

1) Init A Xavier B Xavier ← Why Not?

Hint: Think about how we think about gradient based learning locally.

ΔW would be initialized to something big.

2) One of A or B should be 0. ☺

This way, we start at pretrained model.

How to Initialize A?

1) Xavier
2) Compute a batch of gradients for W_* (Large)
   Take the SVD
   Pick top r.
3) Take SVD of W_* itself
   Pick top r.

Don't forget to tune your learning rate.
   Can be different from best pre-training learning rate.

(Typical choices for AdamW — $1\times10^{-4} \to 5\times10^{-4}$ for LR)
Higher than typical weight-decay — 0.01 to 0.001-ish

Meta-learning: Making a model better at being fine-tuned for tasks

What's a good baseline approach?

0) Do Nothing: Random Initialization
1) General Foundation Model
2) MAML

What do we need?

A) A collection of tasks from the family.
   i.e. Training Data for these different tasks
   + Loss Function.

B) Approach to fine-tuning.
   e.g. Use a LoRA and the SGD optimizer on training data.

C) Approach to evaluation
   e.g. Eval performance on held-out set

Key Insight: In ML, default is train like you'll be tested.

Second Insight: Learning Process of SGD is like an RNN.