Carnegie Mellon University

Recitation: HW3
10-423/10-623

23rd February 2024

Agenda

- In context learning, COT
- LoRA
- Instruction Fine Tuning
- Code Walkthrough and Implementation Details

Learning from Small Data

How can we learn from a small amount of data?

Learning from Small Data

How can we learn from a small amount of data?

Few-Shot learning: Few examples to guide for a new task

Zero-Shot learning: No guiding examples

What is Few-Shot Learning?

What is Few-Shot Learning?

Cat Cat Cat Dog

?

Apple Grape Banana

?

Carnegie Mellon University

What is ZERO-Shot Learning?

Cat Cat Cat Dog

?

Apple Grape Banana

?

Carnegie Mellon University

What is ZERO-Shot Learning?

?

?

How to Approach Few-Shot Learning?

How to Approach Few-Shot Learning?

One Answer: Meta Learning

What is Meta Learning?

What is Meta Learning?

Learning to Learn?

What is Meta Learning?

Learning to Learn?

Optimize Few-Shot Learning Performance

What is Meta Learning?

Learning to Learn?

Optimize Few-Shot Learning Performance

Train input example

Cat Cat Cat Dog ?

Carnegie Mellon University

What is Meta Learning?

Learning to Learn?

Optimize Few-Shot Learning Performance

Train input example

Cat Cat Cat Dog ?

Train target example

Cat

How Can We Solve This Problem?

Train input example

Cat Cat Cat Dog ?

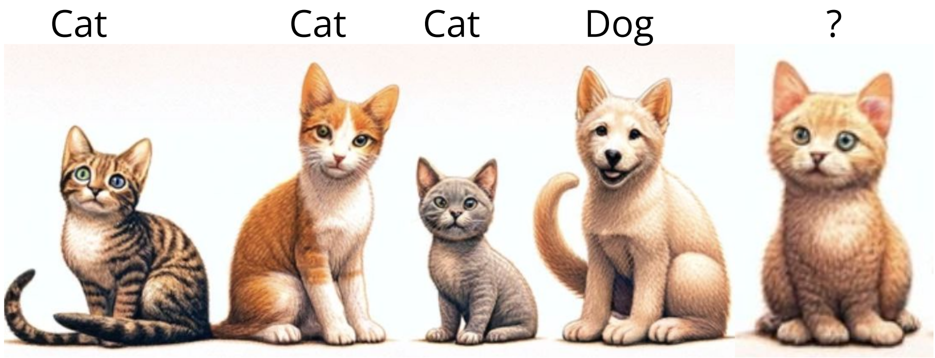
Train target example

Cat

How Can We Solve This Problem?

One Answer: Treat Like Regular Supervised Learning

Train input example



Train target example

Cat

How Can We Solve This Problem?

One Answer: Treat Like Regular Supervised Learning

Train input example

Cat Cat Cat Dog ?

Transformer /RNN

Train target example

Cat

What is In-Context Learning?

What is In-Context Learning?

Cat Cat Cat Dog ?

LLM

Cat

What is In-Context Learning?

LLMs generate predictions conditioned on the examples during inference

LLMS implicitly learns what parts of contexts to focus on to give the right answer, even for new unseen tasks - LLMs “know how to learn” even though we didn’t “learn to learn”!

Cat Cat Cat Dog ?

LLM

Cat

Carnegie Mellon University

Can We Improve In-Context Learning Using Prompt Engineering?

Prompt Engineering: Act of refining input to guide LLMs for desired outputs

Cat Cat Cat Dog ?

LLM

Cat

Carnegie Mellon University

Can We Improve In-Context Learning Using Prompt Engineering?

Cat because stripes

Cat because whiskers

Cat because nose

Dog because eyes

?

LLM

Cat because whiskers

Can We Improve In-Context Learning Using Prompt Engineering?

"Chain-of-thought prompting"

Cat because stripes

Cat because whiskers

Cat because nose

Dog because eyes

?

LLM

Cat because whiskers

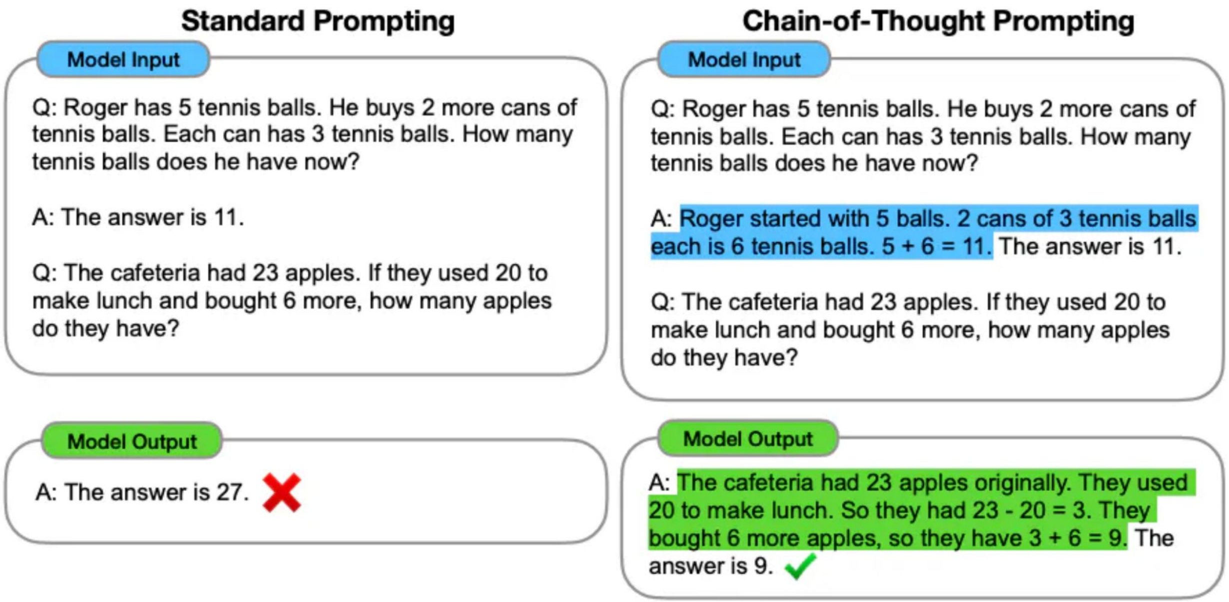
Carnegie Mellon University

Can We Improve In-Context Learning Using Prompt Engineering?

“Chain-of-thought prompting”
(a better example)

Can We Improve In-Context Learning Using Prompt Engineering?

"Chain-of-thought prompting"
(a better example)



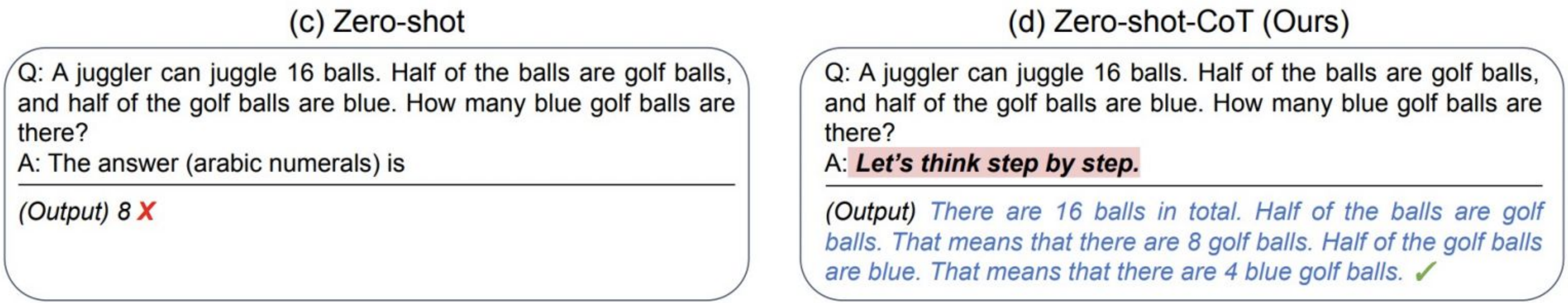
Problem with Few-Shot Learning: Context is Expensive

Problem with Few-Shot Learning: Context is Expensive

We can improve zero-shot learning with prompt engineering

Problem with Few-Shot Learning: Context is Expensive

We can improve zero-shot learning with prompt engineering



Adapting LLMs for Specific Tasks using Fine Tuning

- Although pre-trained language models like GPT possess vast language knowledge, they lack specialization in specific areas.

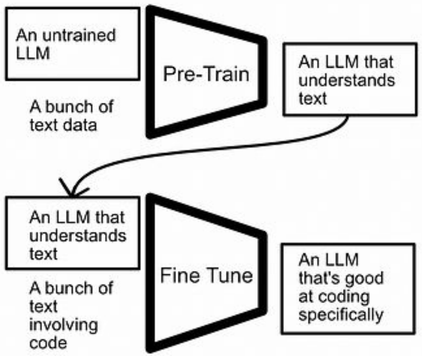
- Fine-tuning addresses this limitation by allowing the model to learn from domain-specific data to make it more accurate and effective for targeted applications.

What is Full Fine Tuning?

- Full fine-tuning is the process of training the entire model on the task-specific data.

- This means all the model layers are adjusted during the training process.

- BUT, is this always computationally feasible?



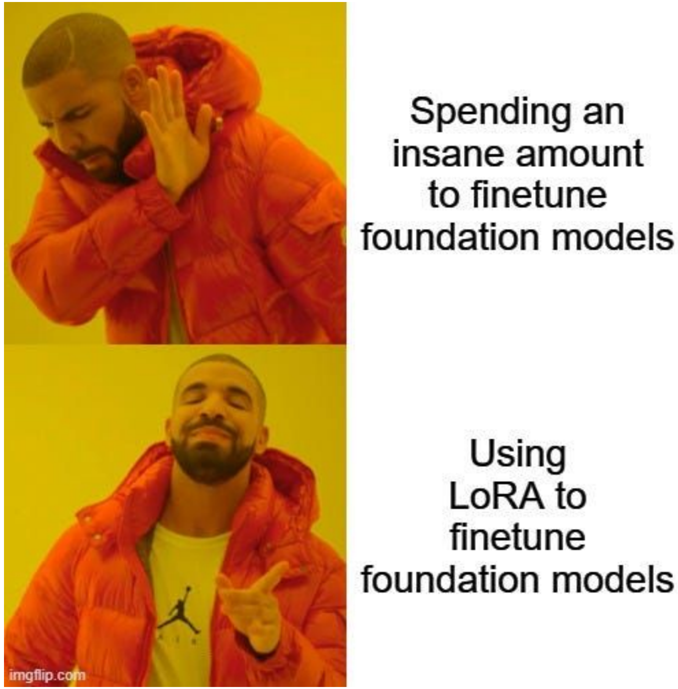
Limitations of Full Fine Tuning

- Total Training Memory for a model includes the following: Model + Optimiser + Activations + Gradients

- When full fine tuning, gradient needs to be calculated for every parameter. And in full precision training(fp32), the gradient for each parameter takes up 4 bytes of memory.

- Now imagine training a 13B parameter model. 13B * 4bytes = 52 Gigabytes of memory is required for the gradients alone!

- What about the time required to backpropagate through ALL these parameters?

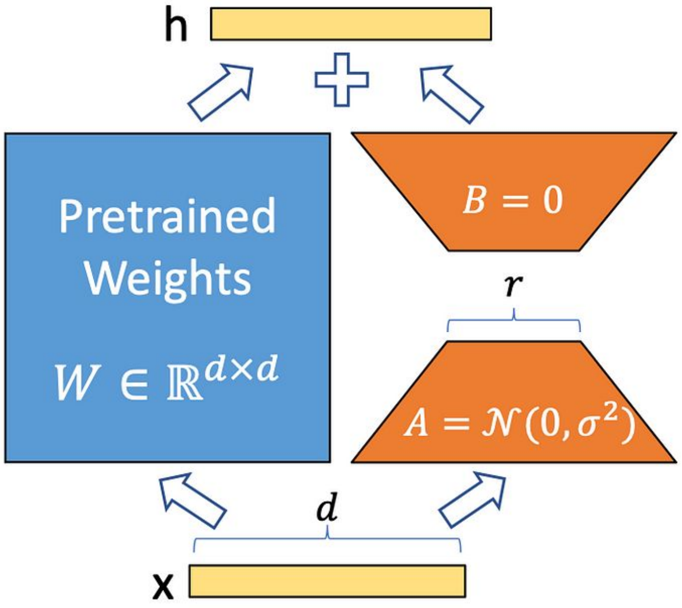


LoRA: Low Rank Adaptation

- LoRA addresses some of the drawbacks of full fine-tuning.

- **How?**

  By freezing the pre-trained model weights and injecting trainable rank decomposition matrices into each layer of the Transformer architecture.



LoRA Explained

- LoRA reimagines fine tuning not as learning better parameters, but as adjustments required to the existing parameters to make them better.

$$\overbrace{W_{\mathtt{ft}}}^{\text{Finetuned Weights}} = \underbrace{W_{\mathtt{pt}}}_{\text{Pretrained Weights}} + \overbrace{\Delta W}^{\text{Weight Update}}$$

LoRA Explained

- LoRA hinges on the following concepts:

1. Pre-trained language models have a low “intrinsic dimension”. They can still learn efficiently despite a random projection to a smaller subspace.

2. If you have a large matrix, with a significant degree of linear dependence (and thus a low intrinsic dimension), you can express that matrix as a factor of two comparatively small matrices.

$$W_0x + \Delta W x = W_0x + BAx$$

LoRA Explained

- How are we saving memory with LoRA?

  The full 5x5 matrix above has 25 values in it, whereas if we count the values in the decomposed matrices, there are just 10 (5 + 5).

- As the matrix we are trying to approximate gets larger and larger(delta W), we work with a smaller and smaller proportion of values in our decomposed matrices(A and B), compared to the full-size matrix.

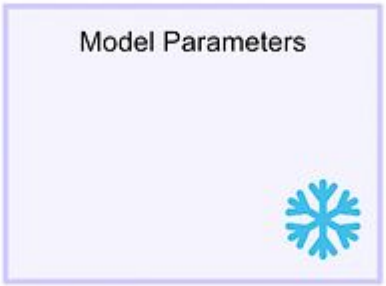
$$\Delta W$$

$$\begin{bmatrix} 1 \\ 3 \\ 7 \\ -4 \\ 2 \end{bmatrix} \times \begin{bmatrix} 5 & 1 & -1 & 3 & 4 \end{bmatrix} = \begin{bmatrix} 5 & 1 & -1 & 3 & 4 \\ 15 & 3 & -3 & 9 & 12 \\ 35 & 7 & -7 & 21 & 28 \\ -20 & -4 & 4 & -12 & -16 \\ 10 & 2 & -2 & 6 & 8 \end{bmatrix}$$

Rank	7B	13B	70B	180B
1	167,332	228,035	529,150	848,528
2	334,664	456,070	1,058,301	1,697,056
4	669,328	912,140	2,116,601	3,394,113
8	1,338,656	1,824,281	4,233,202	6,788,225
16	2,677,312	3,648,561	8,466,404	13,576,450
512	85,673,987	116,753,964	270,924,934	434,446,406

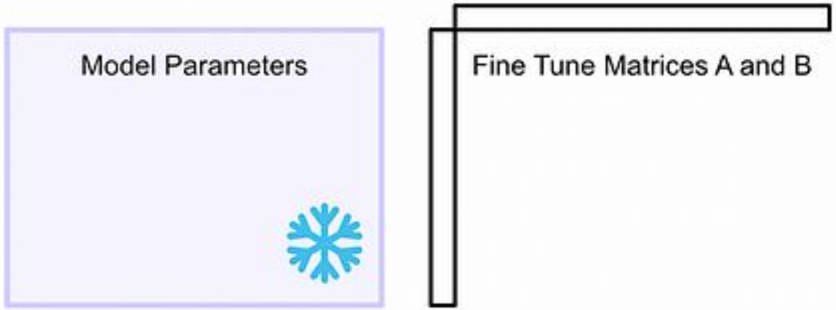
How Does LoRA Work?

- So, first, we freeze the model parameters. We'll be using these parameters to make inferences, but we won't update them.



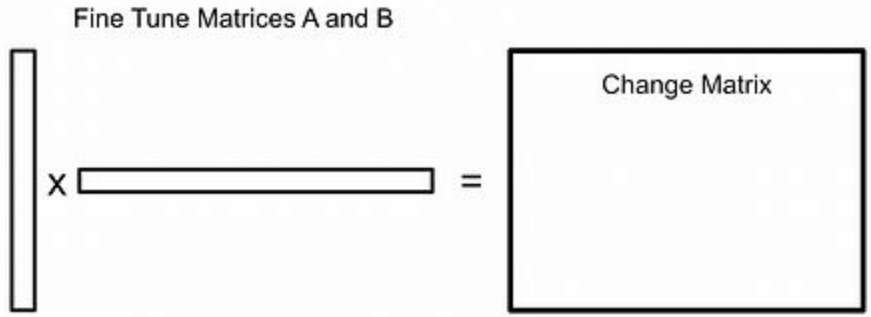
How Does LoRA Work?

- Then we create two matrices. These are sized in such a way that, when they're multiplied together, they'll be the same size as the weight matrices of the model we're fine tuning.



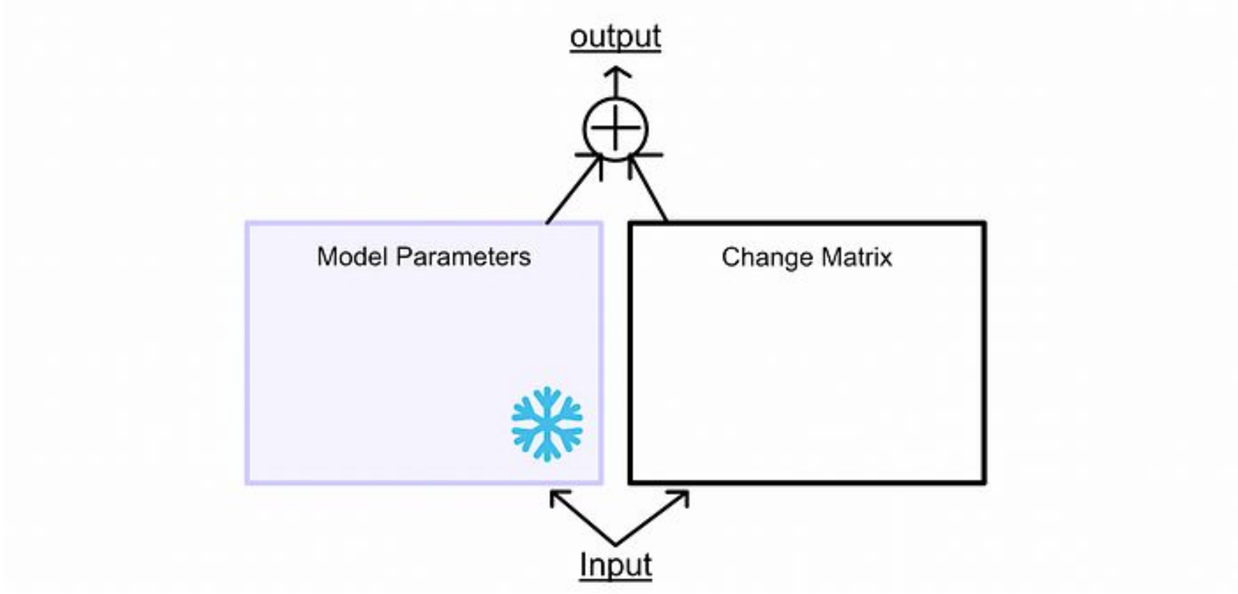
How Does LoRA Work?

- Then we calculate the the change matrix(delta W)



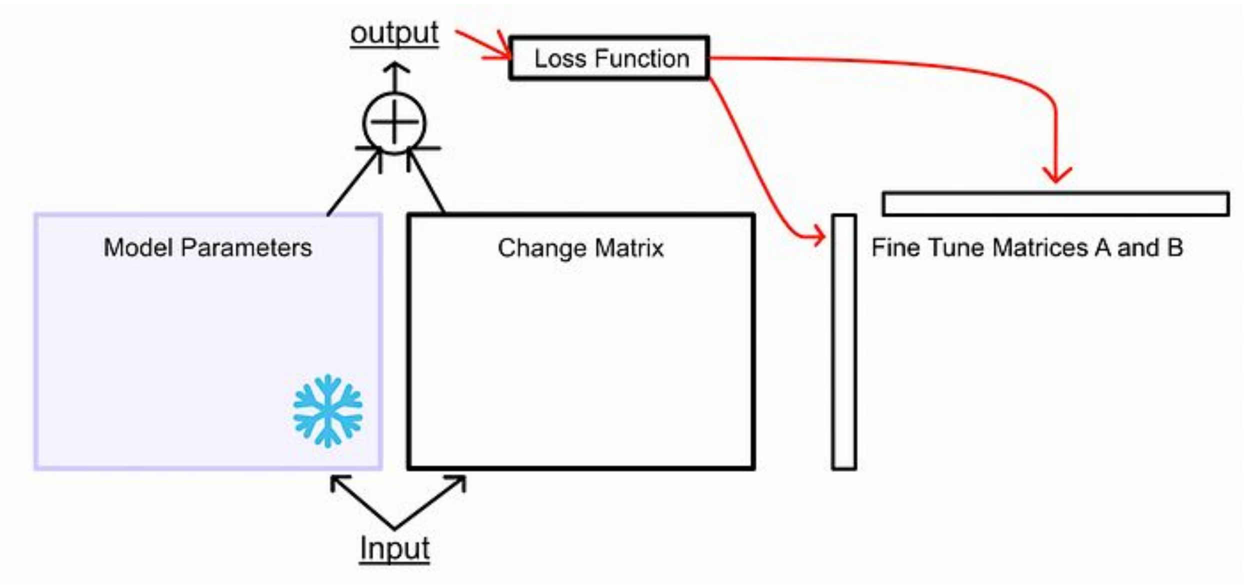
How Does LoRA Work?

- We pass our input through the frozen weights and the change matrix.



How Does LoRA Work?

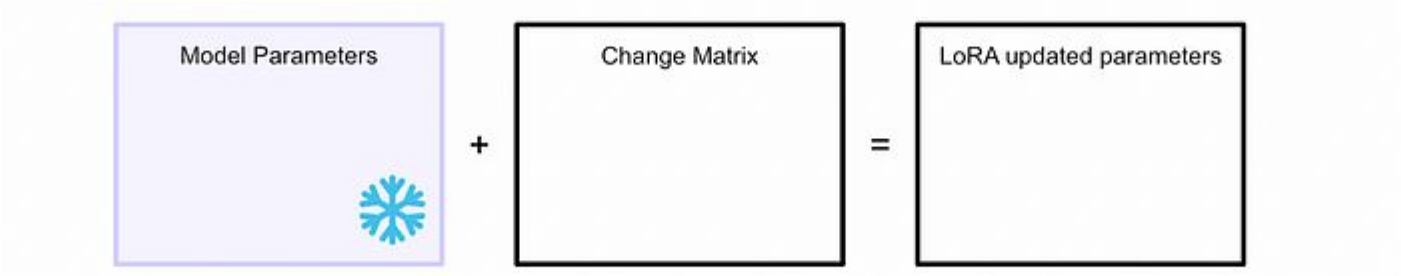
- We calculate the loss and update matrices A and B.



Carnegie Mellon University

How Does LoRA Work?

- At inference time we add the change matrix to the frozen weights and pass the input.

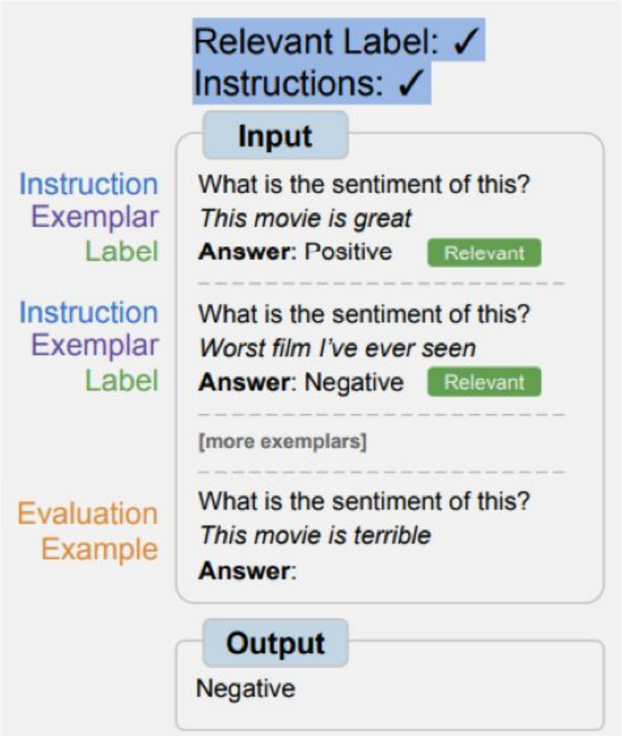


How Does LoRA Work?

- P.S: Don't forget the scaling factor!

$$h = W_0 x + \frac{\alpha}{r} \Delta W x = W_0 x + \frac{\alpha}{r} B A x$$

Instruction Fine Tuning



- Instruction fine-tuning is a technique used to train the model using examples that demonstrate how it should respond to a specific instruction.

Code Walkthrough

Dataset and Task

Dataset

- Rotten Tomatoes Dataset from HuggingFace.
- Balanced movie review dataset containing positive and negative labels denoting sentiment.

Task

- Movie review - sentiment classification
- Instruction Fine Tuning with PEFT (LoRA)

Why is the task non-trivial?

Instruction: “Predict the sentiment of the following text: You are terrible. Label: ”

GPT2 OOB Response: “ Yes. Murders of this kind...”

Code structure

hw3/

- **lora.py**
- **model.py**
- **dataloader.py**
- **train.py**
- **generate.py**
- requirements.txt
- run_in_colab.ipynb
- wandb_api.json

modify and upload to gradescope

Code structure

hw3/

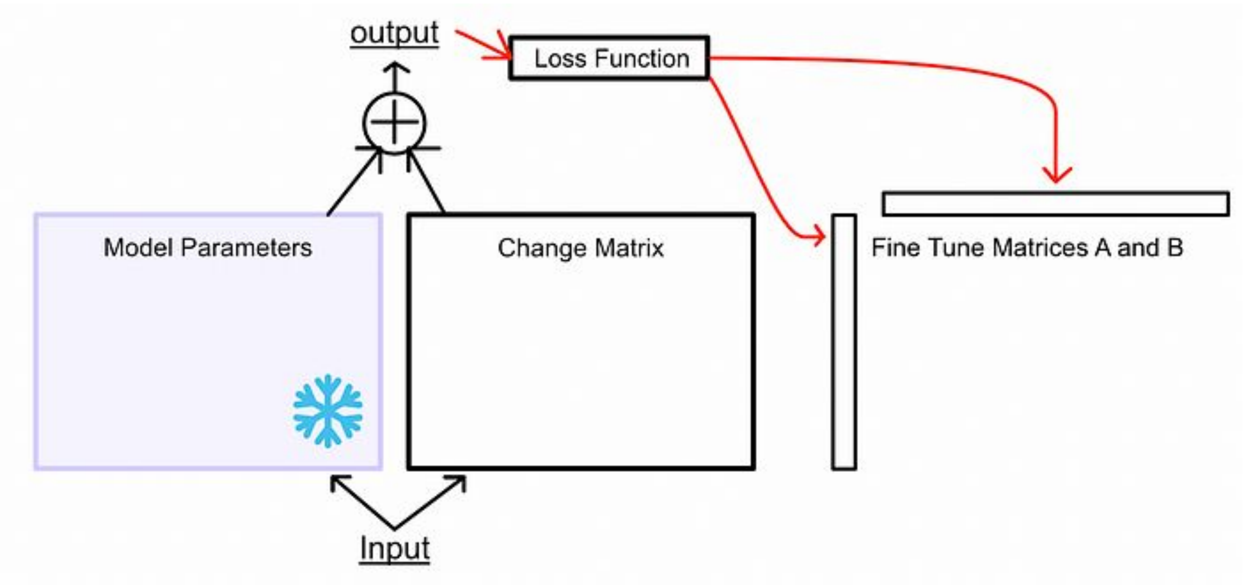
- **lora.py**
  - **Implement LoRA in this. Some starter code is provided to guide you. Only implement LoRA in a linear layer.**
  - **def mark_only_lora_as_trainable(model) - Hint: iterate through named_parameters() (Link)**
- **model.py**
  - **The vanilla working transformer implementation from HW1 (i.e. without GQA and ROPE). Use your implemented LoRA in the attention layers.**
- **dataloader.py**
  - **custom dataloader implemented for the rotten tomatoes dataset. After running other experiments, customize the prompt for one of the ques.**
- **train.py**
  - **The script for training GPT. This file is long but your only requirement is to make your model lora-friendly. Note: This is only done if we are using a pretrained model to begin with**
- **generate.py**
  - **The script for generating text with your trained (or raw) GPT model. Since we are using a classification dataset, convert text outputs from the LLM to integer labels.**
- requirements.txt
- run_in_colab.ipynb
- wandb_api.json

Carnegie Mellon University

How Does LoRA Work?

- We calculate the loss and update matrices A and B.

$$W_0x + \Delta W x = W_0x + BAx$$




Talk is cheap.
~~Show me~~ WRITE the code.
Linus Torvalds
quotefancy

minGPT

Dataset

Rotten Tomatoes - Movie Review Dataset (Classification)

gpt2 untrained:

"Predict the sentiment of the following text: You are terrible. Label: "

Response: " Yes. Murders of this kind..."

Code Structure

-handout

- lora.py
- model.py
- dataloader.py
- train.py
- generate.py
- configs
    - finetune_config_params.py
- configurator.py
- requirements.tx

Code Structure

-handout

- **lora.py** (30-35 lines)

Code Structure

-handout

- **lora.py** (30-35 lines)
- model.py (2 lines)
- dataloader.py (5 lines)
- train.py (1 lines)

Training

Code Structure

-handout

- **lora.py** (30-35 lines)
- model.py (2 lines)
- dataloader.py (5 lines)
- train.py (1 lines)

Training

- generate.py (10 lines)

Evaluation

Lora.py

- only add LoRA to the linear layer
- so we tweak Linear Layer to support LoRA
    - inherit from the Linear Layer
- We should also be able to use this tweaked layer as our normal Linear layer if rank<=0.

Helpful PyTorch functions:

- NN Linear Layer (source code to skim through the existing functions): Link

Lora.py

class LoRALinear(nn.Linear):

- __init__() -> create the parameters (only if lora rank is >0)
    - Helpful PyTorch functions:
        - torch.nn.parameter.Parameter(torch.empty(in_dim, out_dim)) Link
- reset_parameter() -> set the initial values for the parameters
    - Helpful PyTorch functions:
        - torch.nn.init (Link)
- forward() -> called in *each* forward pass of the model
- train() -> called only when model.train() is called
- eval() -> called only when model.eval() is called

Carnegie Mellon University

Wait but why do we need to (re)implement train and eval?

- How do you know if your weights have been merged in or not?
  - Use self.has_weights_merged
- When do you want your weights to be merged? (train or eval)?
- When do you want your weights to be de-merged? (train or eval)?

- Ensure that your train/eval/forward have weights in the required format (merged/de-merged) - if not, merge/de-merge them

Carnegie Mellon University

~~I want~~ *The writeup tells me* to do full fine tuning with LoRA layer implemented. How do I do that?

- set r=0
- What this does is it never initializes your lora_a, lora_b matrix
  - so your layer is now the equivalent of Linear.
- Account for this in your train, forward and eval functions! (hint: use self.is_lora())

In LoRA you are only updating lora weight (and no other weights). How do you ensure that in practice?

Implement def mark_only_lora_as_trainable(model)

Hint: iterate through named_parameters() (Link)

Additional Files:

- model.py: add lora to attention layers
- dataloader.py: Write your instruction for fine tuning. Also decide if you want to make your labels more descriptive!

- train.py : make your model actually use lora

!python train.py --init_from="gpt-medium" --out_dir="gpt_lora_r:16_alpha:32"

Where do I change values of my hyperparams?

- Hyperparameters in LoRA: r, alpha, lr, max_iters..

- finetune_config_params.py

```
init_from = 'gpt2'

eval_interval = 5
eval_iters = 40
wandb_project = 'lora_finetune'
out_dir="lora-gpt-default"

# only save checkpoints if the validation loss improves
always_save_checkpoint = False

batch_size = 1
gradient_accumulation_steps = 32
max_iters = 50

# finetune at constant LR
learning_rate = 5e-4
decay_lr = False

device = "cuda"
compile = False
compute_grad_memory = True

lora_rank = 128
lora_alpha = 512
lora_dropout = 0.05
```

- command line (Eg python train.py --init_dir=”lora-pls-work3”)

Carnegie Mellon University

Generate.py

- Encouraged to just look at the generations gpt2-untrained vs finetuned gpt2 produces (use get_generation(prompt) method in the generate.py)
- Implement your own accuracy function:
  - Check if LoRA actually produced the labels you told it to
    - *GPT2 (and other small LMs (Even 7B ones)) may have trouble generating EOS and so one hack is to ask it to generate a limited number of tokens and look for labels in the first few characters.*
    - *Often labels generated will be garbage, make sure to consider those as negative predictions in your accuracy function*

```
!python generate.py --init_from=”resume” --out_dir=”gpt_lora_r:16_alpha:32”
```

Thank you!


I just spent our entire AI budget on fine-tuning a model
But at least we have a working model now, right?
We have a working model, right?