ChatGPT 2022

- We will now roughly describe how ChatGPT 2022 works.
- We will simplify some steps so as not to get lost in the details
- More details can be found in original paper by OpenAI researchers:

Ouyang et al., “Training Language Models to Follow Instructions with Human Feedback,” NeurIPS 2022, 13,000 citations

ChatGPT 2022

Step 1

Collect demonstration data, and train a supervised policy.

A prompt is sampled from our prompt dataset.

A labeler demonstrates the desired output behavior.

This data is used to fine-tune GPT-3 with supervised learning.

Step 2

Collect comparison data, and train a reward model.

A prompt and several model outputs are sampled.

A labeler ranks the outputs from best to worst.

This data is used to train our reward model.

Step 3

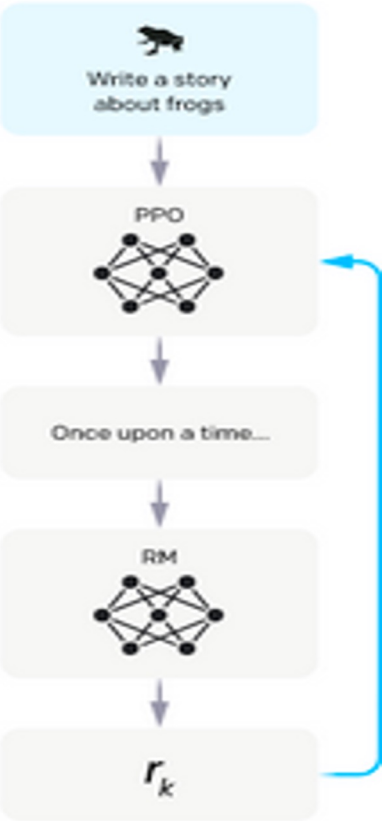
Optimize a policy against the reward model using reinforcement learning.

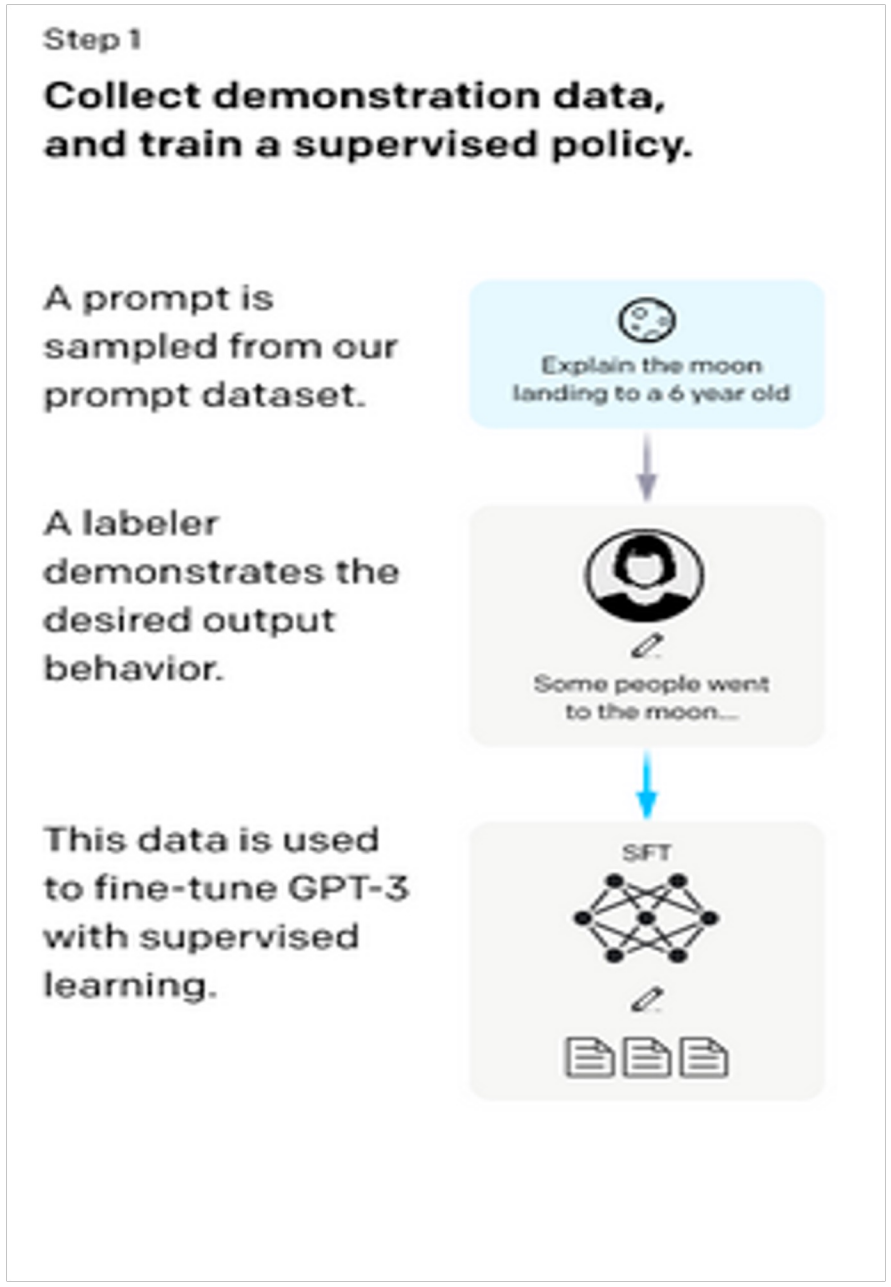
A new prompt is sampled from the dataset.

The policy generates an output.

The reward model calculates a reward for the output.

The reward is used to update the policy using PPO.





- Begin with pre-trained LLM
- Humans (labelers) create prompt/answer pairs.
- Concatenate each prompt with its answer.
- Supervised Fine Tuning (SFT): Fine tune LLM with this new data, i.e., further train the LLM with the concatenations doing next token prediction as during pre-training.

Example Prompt/Response for SFT

Training:

<|prompter|> How do you make tea? <|assistant|> To make tea, start by boiling water. Add a tea bag and let it steep for 3-5 minutes. Then remove the tea bag and enjoy. <|endoftext|>

Inference:

<|prompter|>What country won the most gold medals in the 2008 Olympics? <|assistant|>

Reduce Toxicity and Hallucinations

- Suppose we have a reward function r(x,y) that takes as input a prompt x and response y, and judges the quality of the response y for prompt x.
- Very high r(x,y) signifies very good response; very low r(x,y) signifies very bad response.
- Then beginning with a dataset of prompts, we can sample a prompt, have the LLM generate multiple responses (say 8), and use PPO update the model parameters. (More later)
- But first: how do we get r(x,y)?

Step 2a: Collect human feedback data

- First collect data: for a given prompt x, use the pretrained LLM to generate two responses y_1 and y_2. Do this for many prompts.
- Have humans judge which response is better: set y_w to winning response and y_l to losing response. **This is the human feedback**
- Data D = { $(x^{(i)}, y_w^{(i)}, y_l^{(i)})$ }
- Note different labelers may label the same (x,y_1,y_2) and have different preferences.
- Use data D to train a reward model $r_\phi(x,y)$

Step 2b: Train a reward model

- Make a copy SFT model. Add a linear layer to last transformer layer, mapping to a single real number.
- Denote $r_\phi(x,y)$ as output of model when inputting prompt x along with a response y, where ϕ is parameters of model.
- Train network using loss function:

$$\mathcal{L}_R(r_\phi, \mathcal{D}) = -\mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}}\left[\log \sigma(r_\phi(x,y_w) - r_\phi(x,y_l))\right]$$

- where is the logistic function σ( )
- Note that this loss function encourages $r_\phi(x,y_w)$ to be high and $r_\phi(x,y_l)$ to be low. What if half of the labelers say $y_1>y_2$ and half say $y_2>y_1$?
- Where does this loss function come from? Will come back to this.

Step 3: Formulate and solve an RL problem

- episodic RL problem:
  - initial state of episode = prompt
  - action space = vocabulary
  - action = token in vocabulary
  - state = prompt + all response tokens generated so far
  - policy = LLM which takes as input state and outputs probability distribution over vocabulary. Policy is parameterized by LLM parameters θ, which RL updates.
  - episode ends when <|endoftext|> token is sampled from policy distribution
  - rewards are zero except final reward in episode, which is $r_\phi(x,y)$ where x is the prompt and y is the generated response
  - goal: $\text{Max}_\theta \, E_{x\sim D', y\sim \pi\theta} [\, r_\phi(x,y) \,]$ where D' is some fixed set of prompts
  - That is, fine-tune LLM (θ) so that responses y with high values of $r_\phi(x,y)$ are more likely

How might we apply Policy Gradient?

- Sample x from prompt dataset D'.
- Have LLM with current parameters θ generate, say, 8 responses (episodes): $y_1, y_2, \ldots, y_8$
- Evaluate $r_\phi(x, y_i)$, i=1,…,8. Normalize these 8 rewards.
- Push LLM parameters θ to increase the probability of those of y_i with high values of $r_\phi(x, y_i)$, and decrease the probability of those of y_i with low values of $r_\phi(x, y_i)$

$$\theta \leftarrow \theta + \frac{\alpha}{N} \sum_{i=1}^{N} r_\phi(x, y_i) \sum_{t=1}^{T} \nabla_\theta log\, \pi_\theta(a_t^{(i)} | s_t^{(i)})$$

- Then repeat: sample new prompt from D', and for that prompt and updated θ generate 8 new response,…

PPO is instead often used

- PPO is a refinement of policy gradient
  - sample a prompt, use current LLM to generate 8 episodes (responses), etc
- Need a critic network that takes in current state s and produces estimated return using current policy V(s).
- Need to do the micro updates
- Need to use the PPO loss function with ε clipping.

Issue: Do not deviate too much from SFT policy

- Let θ_{SFT} be original pre-trained parameters (after SFT)
- Thus $\pi_{\theta_{SFT}}(a|s)$ is the original policy
- $\pi_{\theta}(a|s)$ is the policy being updated with RL.
- Do not want $\pi_{\theta}(a|s)$ to deviate too much from $\pi_{\theta_{SFT}}(a|s)$

- Overload notation: Also write $\pi_{\theta}(y|x)$ and $\pi_{\theta_{SFT}}(y|x)$

KL-Divergence and fine-tuning objective

- Measures how close one distribution is to another.
- Consider two distributions $\pi(a)$ and $\pi'(a)$
- $D_{KL}(\pi' || \pi) := E_{a\sim\pi}[\log \pi(a) - \log \pi'(a)] = \sum_a \pi(a)[log\pi(a) - log\pi'(a)]$
- For a prompt x: $D_{KL}(\pi' || \pi)(x) := E_{y\sim\pi(.|x)}[\log \pi(y|x) - \log \pi'(y|x)]$
- $\log \pi(y|x) - \log \pi'(y|x)$ with $y\sim \pi(.|x)$ is an unbiased estimator of : $D_{KL}(\pi' || \pi)(x)$

Finetuning objective (step 3):

- $\max_\theta E_{x\sim D, y\sim\pi\theta}[r_\phi(x,y) - \beta D_{KL}(\pi_\theta(y|x)||\pi_{SFT}(y|x))]$

- That is, finetune LLM parameters θ so that (i) LLM gives answers with high SFT policy.
- Question: What would be the 8 returns used in PG/PPO for this objective?

Reward model theory

- Where does the loss function come from:

$$\mathcal{L}_R(r_\phi, \mathcal{D}) = -\mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}}\left[\log \sigma(r_\phi(x, y_w) - r_\phi(x, y_l))\right]$$

- Recall that in classification, the model puts logits into softmax to assign a predicted probability to each class.
- Let's do something similar here. Given a prompt x and two responses y_1 and y_2, let's have the model assign a probability to y_1 being preferred to y_2. Instead of softmax, we use
- $P_\phi(y_1 \text{ preferred to } y_2 | x, y_1, y_2) = \frac{\exp(r_\phi(x,y_1))}{\exp\left(r_\phi(x,y_1)\right)+\exp(r_\phi(x,y_2))} = \sigma(\, r_\phi(x,y_1) - r_\phi(x,y_2)\,)$
- In the dataset D, we have the ground truth preferences. We use MLE to match the model predictions to the ground truth:

$$\text{Min}_\phi \; -\sum_{(x,y_w,y_l)} log P_\phi(y_w \text{ preferred to } y_l | x, y_w, y_l)$$

- Plugging $P_\phi(y_1 \text{ preferred to } y_2 | x)$ into MLE gives the above loss function.

Direct Preference Optimization (DPO)

- "Direct Preference Optimization: Your Language Model is Secretly a Reward Model, " Rafailov et al., Dec 2023. (3000+ citations)
- Alternative to RLHF:
  - Still uses human feedback
  - But uses neither a reward function nor trail-and-error RL
- DPO loss function:

$$\mathcal{L}_{\text{DPO}}(\pi_\theta; \pi_{\text{ref}}) = -\mathbb{E}_{(x, y_w, y_l)\sim\mathcal{D}} \left[\log \sigma \left(\beta \log \frac{\pi_\theta(y_w \mid x)}{\pi_{\text{ref}}(y_w \mid x)} - \beta \log \frac{\pi_\theta(y_l \mid x)}{\pi_{\text{ref}}(y_l \mid x)} \right) \right]$$

- Here $\pi_{\theta_{ref}}(y|x)$ is $\pi_{\theta_{SFT}}(y|x)$
- DPO: optimize this loss function directly without generating additional responses.
- Push θ to increase $\pi_\theta(y_w|x)$ and decrease $\pi_\theta(y_l|x)$
- See paper for derivation of loss function L_{DPO}

Fine-tuning Pre-trained LLMs (aka SFT)

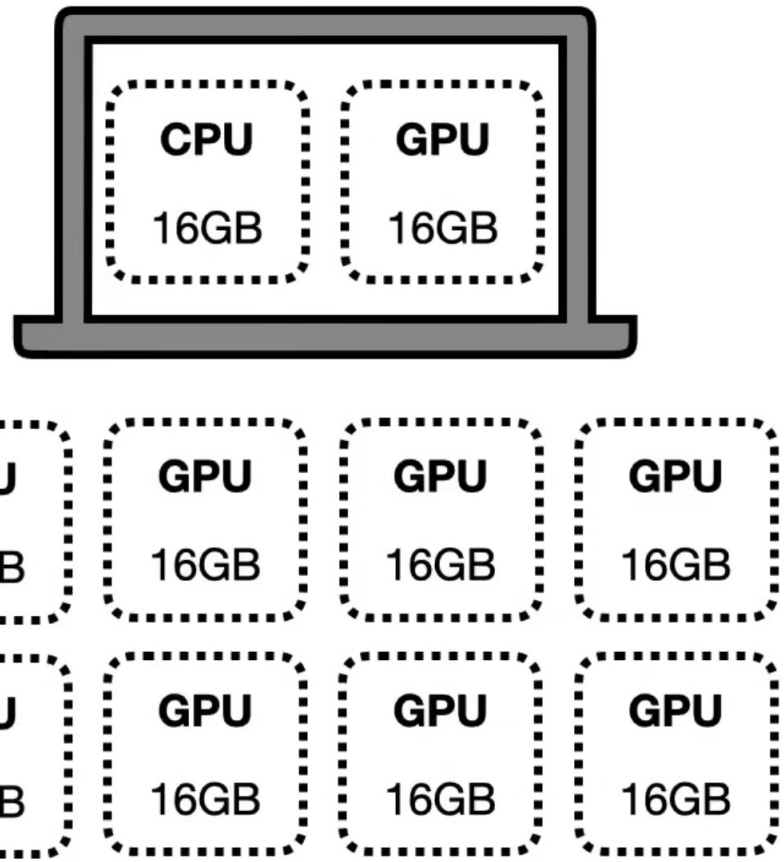
- Adapts a general-purpose model to a specific task, domain, or style
- Domain adaptation:
  - Legal: fine-tune on legal contracts to improve understanding of legal terms like indemnity, force majeure, etc.
  - Medicine: fine-tune on medical documents to handle abbreviations lilke “pt w/ hx of COPD” and ”Rx albuterol BID”.
- Specific task adaptation:
  - Question answering: fine-tune on a custom Q&A dataset for customer support.
  - Summarization: fine tune on text+summary of text
- Tone or style adaptation:
  - Fine tune so that LLM generates text in your or Shakespeare’s writing style
  - Mimic a company’s “brand voice”
- Can often fine-tune small model for your task and get better performance than if you use a large model without finetuning.

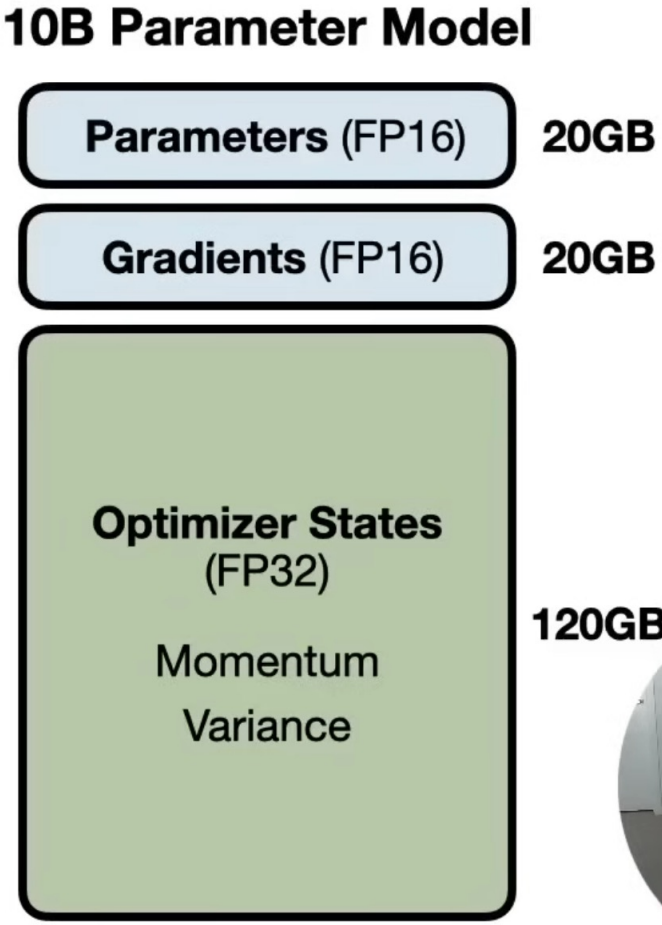
How is fine-tuning done?

- Begin with a pre-trained LLM (perhaps required millions of dollars with months of training).
- Have fine-tuning data (e.g., e-mails, text messages you wrote in your writing style).
- Using this data, continue to train the pre-trained LLM in the same way the LLM was pre-trained (next word prediction with cross-entropy loss function).
- Example: SFT for instructions, as discussed for ChatGPT
- Ideally we do not want fine-tuning to be very costly or require lots of GPUs.
- Typically only fine tune the attention matrix W_Q, W_K, W_V, W_O

The Problem

LLMs are (computationally) expensive

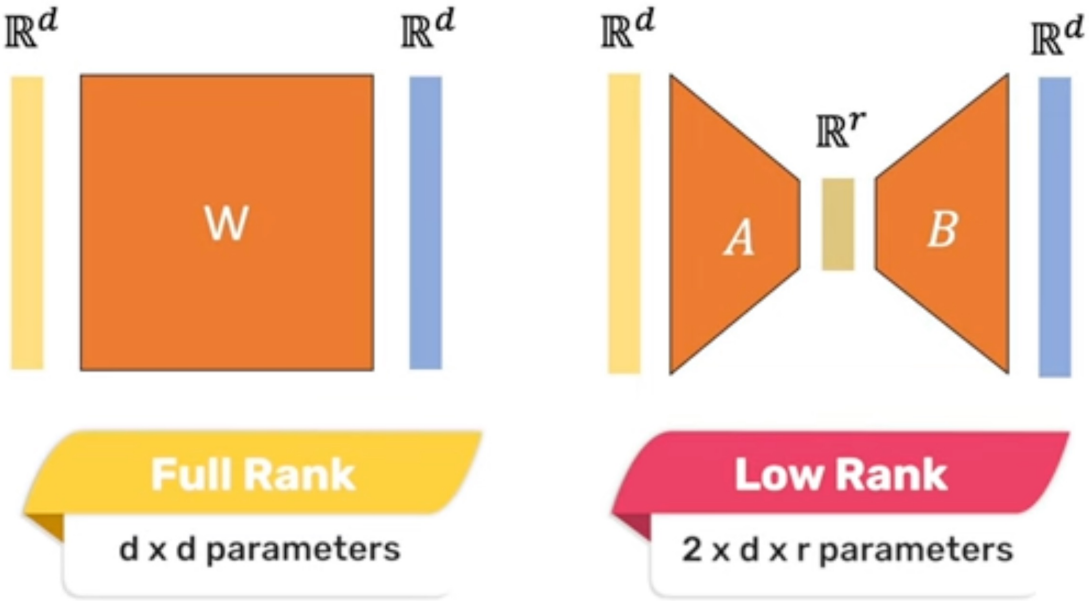


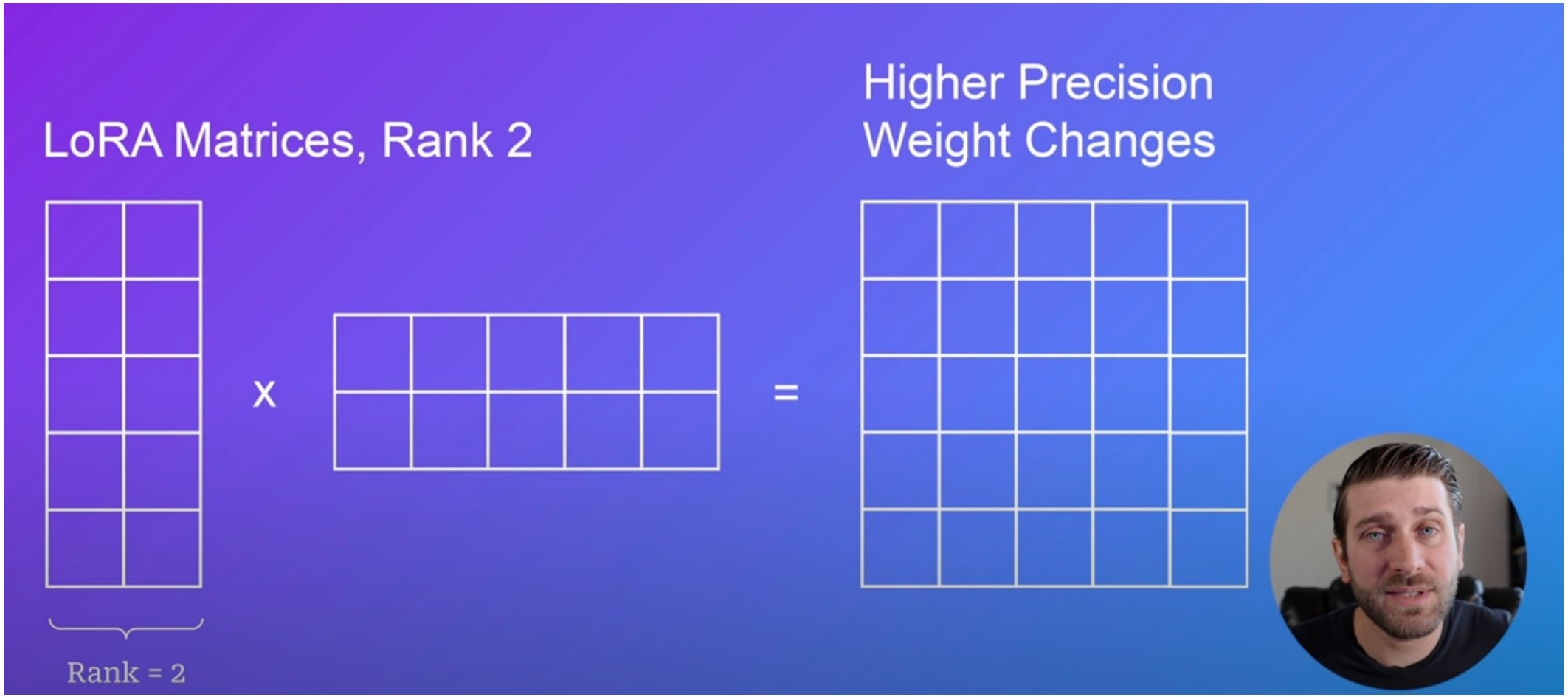


 [2]

Review: What is the rank of a matrix?

- dxd with all rows independent?
- dxd with n-1 rows independent and one row dependent on others?
- dxm with d < m and all rows independent
- Suppose C = AB where A is dxr and B is rxd with r < d. What is the dimension (shape) of C? What is the maximal rank of C?

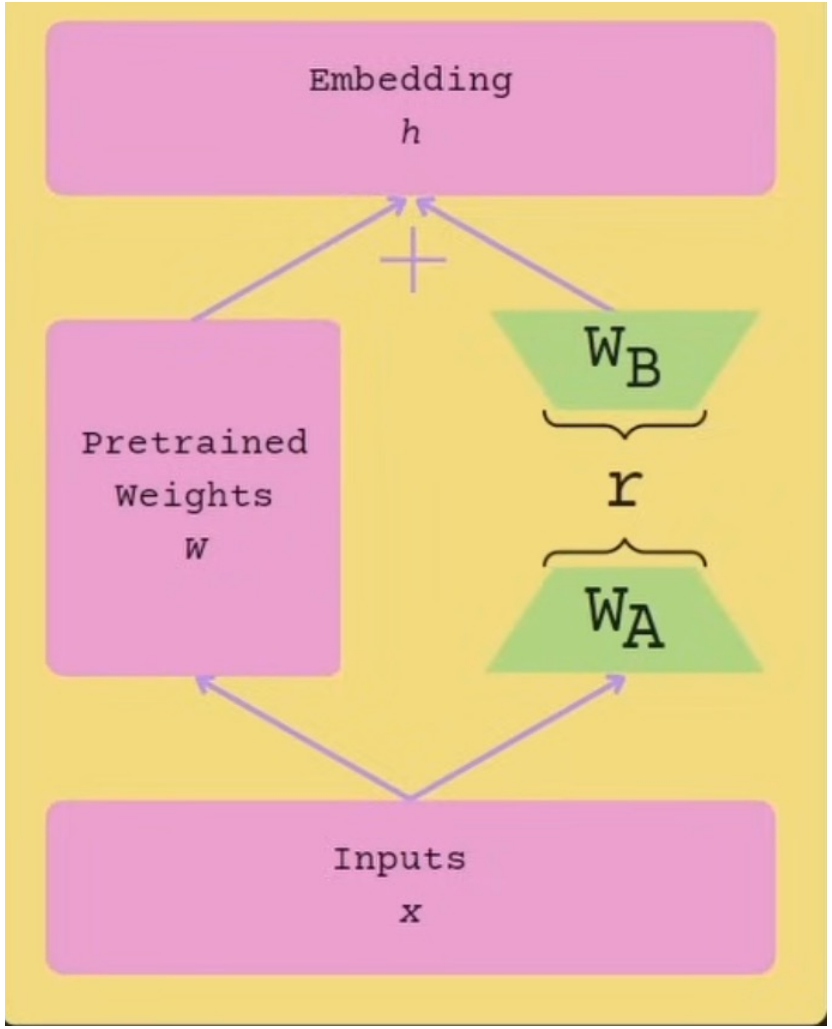



LoRA Matrices, Rank 2
x
=
Higher Precision
Weight Changes
Rank = 2

Low-Rank Adaptation (LoRA)

- "LoRA: Low-Rank Adaptation of LLMs", Xu et al, 2021, 12,000+ citations
- Efficient scheme for fine-tuning a pre-trained LLM
- Fine tune the attention matrix W_Q, W_K, W_V, W_O in all attention layers
- Adapted parameters: $W = W_0 + \Delta W$ (W represents W_Q, W_K, W_V or W_O)
- LoRA: fix W_0 and efficiently obtain ΔW
- How: During fine-tuning, replace each matrix to be updated W with a matrix of the form $W_0 + AB$, where W_0 are the frozen LLM pre-trained parameters, and A and B are matrices of weights to be updated.
- Only update A and B. Keep W_0 frozen.
- Choose A and B such $|A| + |B| << |W_0|$ where here $|A|$ denotes the number of parameters in matrix A.

Forward pass



LoRA: Parameter reduction

- Consider attention layer with 1 head for simplicity. Assume tuning W_Q, W_K, W_V all of which are 1024x1024.
- Full fine-tuning involves 3,145,728 parameters
- Let r=8
- LoRA: 3x2xdxr = 49,152 parameters, 98% savings

- During training need to store 6x parameters (gradient updates, Adam) in each layer!

QLoRA (Quantized LoRA)

- 2023, 2000+ citations
- To get further savings, can 4-bit quantize W_0. This allows large models to fit in a single GPU.
- LoRA weights remain 16-bits, ensuring stable, high-quality updates during training.
- For the quantization, use non-linear quantization so that there are more buckets for numbers near origin.
- QLoRA also applies additional quantization tricks, which are beyond the scope of this course.

In-context learning

- Fine-tuning: adds knowledge by training on the data
- In-context learning: Adds temporary knowledge in the system prompt without changing model parameters.
- With in-context learning, next word generation is conditioned on the context, typically providing better results.
- Example: Want to chat with LLM about a research paper.
  - In prompt, include the research paper itself. Then chat about research paper.

Retrieval Augmented Generation (RAG)

- Suppose you have lots of company documents.
- Suppose LLM has not been pre-trained with those documents.

RAG:

- Use embedding model to get vector representation for each document. Create database vector-document pairs
- Put user query also into embedding model. Perform dot-product between query and vectors in database to find vectors most similar to query. Retrieve corresponding documents.
- Along with the original query, include those retrieved documents as part of in-context prompt.