# Post-training of language models

He He

NEW YORK UNIVERSITY

March 12, 2025

# Table of Contents

# Recap

- **Pretraining** allows the model to acquire many capabilities from vast data
- Increasing **compute** (data + parameters) *predictably* improves **performance** (held-out perplexity)
- How to use a pretrained model for downstream tasks?

# Motivation

How do we tell the LM what we want to do?

Language LLaMA-2 (70B) 2X A100 80GB

What is the capital of Kenya?
What is the capital of Kenya? Kenya is a country in East Africa with coastline on the Indian Ocean. It encompasses savannah, lakelands, the dramatic Great Rift Valley and mountain highlands.

# Motivation

How do we tell the LM what we want to do?

Language LLaMA-2 (70B) 2X A100 80GB

What is the capital of Kenya?
What is the capital of Kenya? Kenya is a country in East Africa with coastline on the Indian Ocean. It encompasses savannah, lakelands, the dramatic Great Rift Valley and mountain highlands.

Language LLaMA-2 (70B) 2X A100 80GB

A survey on prompting in large language models
Sun, Chengcheng, Zhu, Yuan, Wang, Zhen, Zhu, Xiang
arXiv.org Machine Learning May-28-2022
We conduct a comprehensive survey on prompting in large language models (LLMs) from a technical perspective. We first identify four major types of prompting in LLMs: explicit, implicit, hybrid, and multi-task. We then summarize the different prompting methods under each type. We also analyze the different types of prompting from three aspects: the language model, the prompting method, and the downstream task. We find that the prompting methods can be categorized into three groups: input-based, output-based, and model-based. We also summarize the commonalities and differences between prompting and the traditional downstream task. We then discuss the potential advantages and limitations of prompting in LLMs. Finally, we provide a discussion on the future of prompting in LLMs.

# Early prompting approaches

Main goal: adapt the task to a native language modeling task

Example:

- Sentiment classification: [movie review] This movie is [great/awful].
- Summarization: [document] TL;DR: [summary]

🤔 What are potential limitations?

# In-context learning

How GPT-2 is evaluated on machine translation using GPT-2:

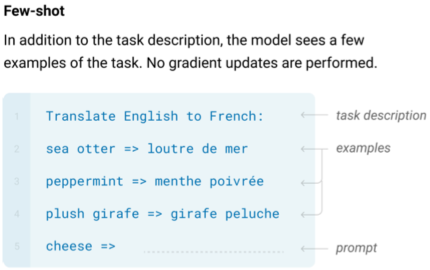
- Induce the task through a demonstration example:

  translation $\sim p(\cdot \mid$ [french sentence] = [english sentence]; [french sentence] =$)$

- WMT-14 French-English test set: 11.5 BLEU (worse than unsupervised MT)
- But, there's only 10MB french data in the 40GB training data!

# In-context learning

In-context demonstrations elicit target capabilities consistently on GPT-3:




This is surprising as this is not a native language model task!

Limitations: results can vary with the choice and order of examples [Zhao et al., 2021]

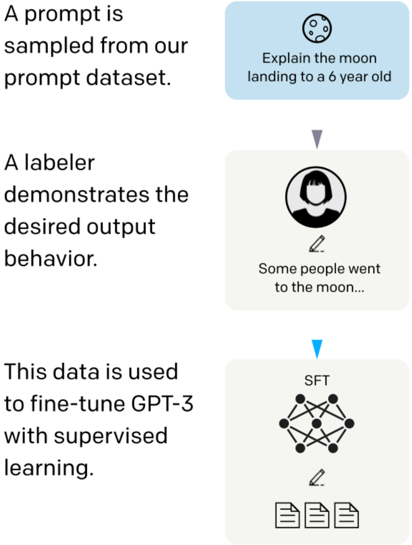
# Post-training

- **Goal**: elicit capabilities acquired during pretraining so that the model can be used directly for downstream task, e.g.,
  - A chat model that answers user queries
  - A reasoning model that solves math problems
  - A shopping assistant that answers questions about products
- **Methods**: update the model on some examples of the target task (often require annotation)
  - Supervised finetuning (SFT): input and gold output
  - Reinforcement learning (RL): input and an output judge

# Table of Contents

# Supervised finetuning

- Supervised learning on the target task
  - Input: prompt (e.g., instruction or question)
  - Output: gold response
- **Key challenge**: data collection
  *How to get the prompts and responses?*



# What kind of data do we need?

Idea 1: use existing NLP benchmarks

- **Natural language inference**:
  *Suppose "The banker contacted the professors and the athlete". Can we infer that "The banker contacted the professors"?*
- **Question answering**:
  *Given the article "The Panthers finished the regular season [...]", what team did the Panthers defeat?*
- **Sentiment analysis**:
  *What's the rating of this review on a scale of 1 to 5: We came here on a Saturday night and luckily it wasn't as packed as I thought it would be [...]*

# What kind of data do we need?

Idea 1: use existing NLP benchmarks

- **Natural language inference**:
  *Suppose "The banker contacted the professors and the athlete". Can we infer that "The banker contacted the professors"?*
- **Question answering**:
  *Given the article "The Panthers finished the regular season [...]", what team did the Panthers defeat?*
- **Sentiment analysis**:
  *What's the rating of this review on a scale of 1 to 5: We came here on a Saturday night and luckily it wasn't as packed as I thought it would be [...]*

But this is not what we ask ChatGPT to do! **distribution shift**

# What kind of data do we need?

- **Problem**: Gap between training and test data

# What kind of data do we need?

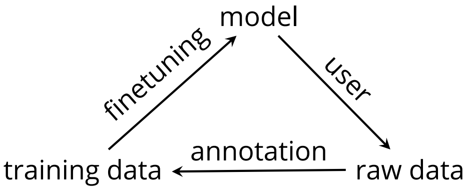
- **Problem**: Gap between training and test data
- Straightforward **solution**: collect training data that is similar to test data
  *How do we know what test data is like?*

# What kind of data do we need?

- **Problem**: Gap between training and test data
- Straightforward **solution**: collect training data that is similar to test data
  *How do we know what test data is like?*
- Get some pilot data
  *which requires a working-ish model first!*

# What kind of data do we need?

- **Problem**: Gap between training and test data
- Straightforward **solution**: collect training data that is similar to test data
  *How do we know what test data is like?*
- Get some pilot data
  *which requires a working-ish model first!*



# Data distribution from early OpenAI API

Table 1: Distribution of use case categories from our API prompt dataset.

| Use-case | (%) |
|---|---|
| Generation | 45.6% |
| Open QA | 12.4% |
| Brainstorming | 11.2% |
| Chat | 8.4% |
| Rewrite | 6.6% |
| Summarization | 4.2% |
| Classification | 3.5% |
| Other | 3.5% |
| Closed QA | 2.6% |
| Extract | 1.9% |

Table 2: Illustrative prompts from our API prompt dataset. These are fictional examples inspired by real usage—see more examples in Appendix A.2.1.

| Use-case | Prompt |
|---|---|
| Brainstorming | List five ideas for how to regain enthusiasm for my career |
| Generation | Write a short story where a bear goes to the beach, makes friends with a seal, and then returns home. |
| Rewrite | This is the summary of a Broadway play:<br>"""<br>{summary}<br>"""<br>This is the outline of the commercial for that play:<br>""" |

Figure: [Ouyang et al., 2022]

# Data distribution from early OpenAI API

Table 1: Distribution of use case categories from our API prompt dataset.

| Use-case | (%) |
|---|---|
| Generation | 45.6% |
| Open QA | 12.4% |
| Brainstorming | 11.2% |
| Chat | 8.4% |
| Rewrite | 6.6% |
| Summarization | 4.2% |
| Classification | 3.5% |
| Other | 3.5% |
| Closed QA | 2.6% |
| Extract | 1.9% |

Table 2: Illustrative prompts from our API prompt dataset. These are fictional examples inspired by real usage—see more examples in Appendix A.2.1.

| Use-case | Prompt |
|---|---|
| Brainstorming | List five ideas for how to regain enthusiasm for my career |
| Generation | Write a short story where a bear goes to the beach, makes friends with a seal, and then returns home. |
| Rewrite | This is the summary of a Broadway play:<br>"""<br>{summary}<br>"""<br>This is the outline of the commercial for that play:<br>""" |

Figure: [Ouyang et al., 2022]

What if you're not at OpenAI?

# Synthetic data

Use off-the-shelf LMs to generate prompts and responses:

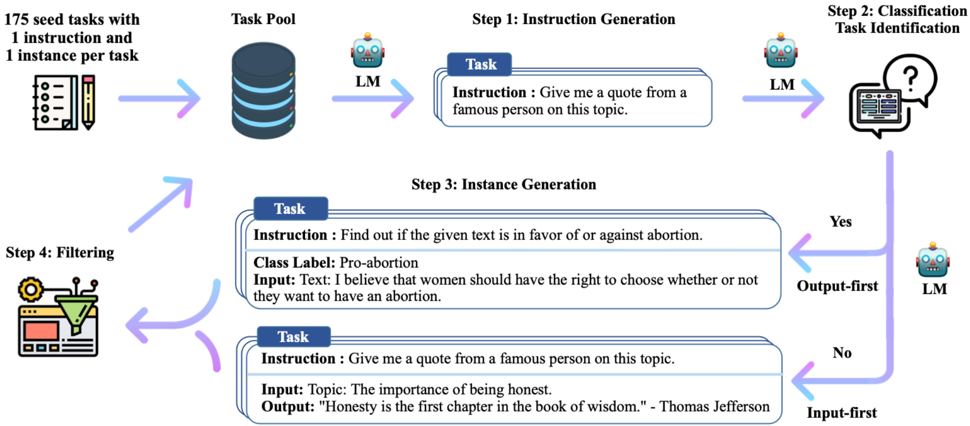



Figure: Self-instruct [Wang et al., 2023]

# The Alpaca model

- November 30, 2022: ChatGPT released

# The Alpaca model

- November 30, 2022: ChatGPT released
- February 24, 2023: LLaMA released (open-weight pretrained model)

# The Alpaca model

- November 30, 2022: ChatGPT released
- February 24, 2023: LLaMA released (open-weight pretrained model)
- March 13, 2023: Alpaca released (open-source ChatGPT like model)
  - 175 seed instruction from Self-Instruct
  - 52K prompt-response pair from text-davinci-003 (¡$500)
  - Supervised finetuning from LLaMA-7B (¡$100)

# The Alpaca model

- November 30, 2022: ChatGPT released
- February 24, 2023: LLaMA released (open-weight pretrained model)
- March 13, 2023: Alpaca released (open-source ChatGPT like model)
  - 175 seed instruction from Self-Instruct
  - 52K prompt-response pair from text-davinci-003 (¡$500)
  - Supervised finetuning from LLaMA-7B (¡$100)
- Impact:
  - Provides an open-source ChatGPT like model for research
  - Many later open-source models adopt this distillation approach

# How good are distilled models?

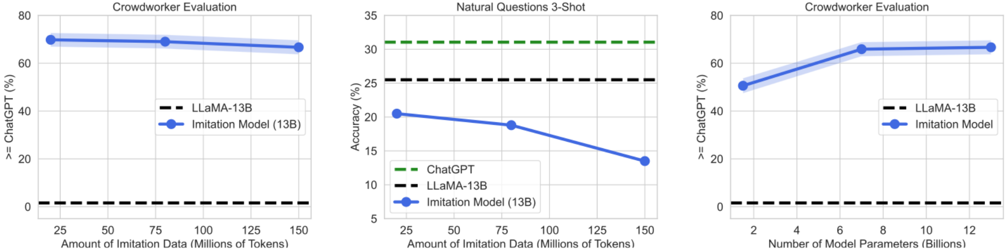



Figure: The False Promise of Imitating Proprietary LLMs [Gudibande et al., 2023]

- Increasing the amount of synthetic data doesn't keep increasing performance
- Distillation can hurt performance on tasks out of the SFT data
- Further performance gain comes from stronger pretrained models

# Parameter efficient finetuning

Finetuning all weights of a large model can be expensive (in what way?)

# Parameter efficient finetuning

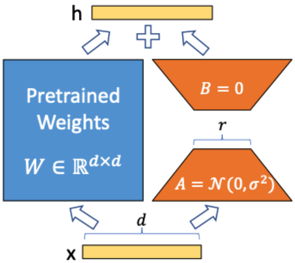
Finetuning all weights of a large model can be expensive (in what way?)

Can we finetune a smaller number of parameters to achieve performance similar to full finetuning?

- Select a subset of parameters to update
  - Last k layers [Lee et al., 2019]
  - Bias terms (BitFit) [Ben-Zaken et al., 2022]
- Add a small number of parameters to adapte the (frozen) pretrained model
  - Insert a small MLP in-between layers [Houlsby et al., 2019]

# Low-rank adaptation of LMs

**LoRA** [Hu et al., 2021]: add low-rank matrices as additional parameters




**Hypothesis**: weight matrices are low rank

Adapters: For any matrix multiplication $h = W_0 x$, we modify it to

$$h = W_0 x + \Delta W x = W_0 x + BA x$$

- $W_0 \in \mathbb{R}^{d \times k}, B \in \mathbb{R}^{d \times r}, A \in \mathbb{R}^{r \times k} (r \ll k)$
- Initialization: $BA = 0$
- Can be applied to any weight matrices, e.g., QKV projection matrices

# Low-rank adaptation of LMs

- Converges to full finetuning as $r$ increases and performance goes up consistently

# Low-rank adaptation of LMs

- Converges to full finetuning as $r$ increases and performance goes up consistently
- No additional inference latency: $W = W_0 + BA$

# Low-rank adaptation of LMs

- Converges to full finetuning as $r$ increases and performance goes up consistently
- No additional inference latency: $W = W_0 + BA$
- Main benefits:
  - Memory and storage saving (optimizer states, checkpoints): 10,000x reduction on GPT3 ($r = 4$)
  - Easy to switch between different finetuned custom models

# Summary

- Supervised finetuning: train the model on human-annotated prompt-response data
- Data consideration:
  - Desiderata: diverse, similar to test data / target task
  - Often costly to obtain; can synthesize using LMs
- Performance upperbound is still decided by the pretrained model

# Table of Contents

# Learning from outcomes

**Motivation**:

- Demonstrations are expensive to obtain—can we learn from weaker signals?
- For many tasks, humans (and animals) only get signal on whether they succeeded or not

**Example**:

- Complex physical tasks: learning to shoot a basketball
- Reasoning: learning to play the game of Go
- Decision making: learning to optimize financial portfolios
- Communication: learning to articulate your ideas to others

# Reinforcement learning

**Goal**: learning from experience by maximizing the expected cumulative reward

# Reinforcement learning

**Goal**: learning from experience by maximizing the expected cumulative reward

1. Agent takes a sequence of **actions** in a world
   *Get a degree, update CV, apply for a job*

# Reinforcement learning

**Goal**: learning from experience by maximizing the expected cumulative reward

1. Agent takes a sequence of **actions** in a world
   *Get a degree, update CV, apply for a job*
2. Agent gets **rewards** along the way indicating how well it did
   *No reponse*

# Reinforcement learning

**Goal**: learning from experience by maximizing the expected cumulative reward

1. Agent takes a sequence of **actions** in a world
   *Get a degree, update CV, apply for a job*
2. Agent gets **rewards** along the way indicating how well it did
   *No reponse*
3. Agent updates its **policy** (on what actions to take)
   *Find a connection? Get an internship? Apply for a different position?*

# Reinforcement learning

**Goal**: learning from experience by maximizing the expected cumulative reward

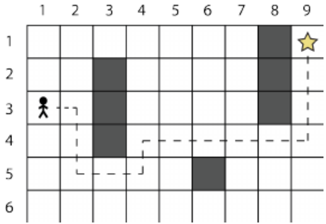
1. Agent takes a sequence of **actions** in a world
   *Get a degree, update CV, apply for a job*
2. Agent gets **rewards** along the way indicating how well it did
   *No reponse*
3. Agent updates its **policy** (on what actions to take)
   *Find a connection? Get an internship? Apply for a different position?*
4. Go back to 1

# Reinforcement learning

**Goal**: learning from experience by maximizing the expected cumulative reward

1. Agent takes a sequence of **actions** in a world — *trial*
   *Get a degree, update CV, apply for a job*
2. Agent gets **rewards** along the way indicating how well it did — *error*
   *No reponse*
3. Agent updates its **policy** (on what actions to take) — *learn*
   *Find a connection? Get an internship? Apply for a different position?*
4. Go back to 1 — *rinse and repeat*

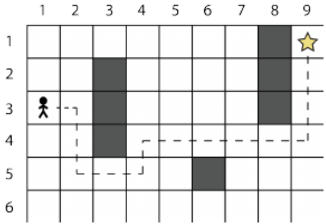
# Reinforcement learning: formalization




At each time step $t$, an agent

- is in a **state** $s_t \in \mathcal{S}$ ($\mathcal{S}$ is the **state space**)
  `cell[i][j]` in the grid world

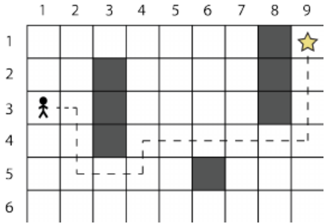
# Reinforcement learning: formalization




At each time step $t$, an agent

- is in a **state** $s_t \in \mathcal{S}$ ($\mathcal{S}$ is the **state space**)
  `cell[i][j]` in the grid world
- takes an **action** $a_t \in \mathcal{A}$ ($\mathcal{A}$ is the **action space**)
  $\{$`up, down, left, right`$\}$

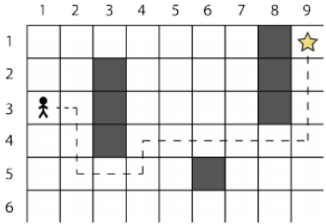
# Reinforcement learning: formalization




At each time step $t$, an agent

- is in a **state** $s_t \in \mathcal{S}$ ($\mathcal{S}$ is the **state space**)
  `cell[i][j]` in the grid world
- takes an **action** $a_t \in \mathcal{A}$ ($\mathcal{A}$ is the **action space**)
  $\{\texttt{up, down, left, right}\}$
- transitions to the next state $s_{t+1}$ according to a **transition function** $p(\cdot \mid s_t, a_t)$
  moves to the corresponding cell if there's no blocker

# Reinforcement learning: formalization




At each time step $t$, an agent

- is in a **state** $s_t \in \mathcal{S}$ ($\mathcal{S}$ is the **state space**)
  `cell[i][j]` in the grid world
- takes an **action** $a_t \in \mathcal{A}$ ($\mathcal{A}$ is the **action space**)
  $\{\texttt{up, down, left, right}\}$
- transitions to the next state $s_{t+1}$ according to a **transition function** $p(\cdot \mid s_t, a_t)$
  moves to the corresponding cell if there's no blocker
- obtains a **reward** $r(s_t, a_t)$ according to the **reward function** $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$
  1 if $s_{t+1}$ is star and 0 otherwise

# Reinforcement learning: objective

The agent uses a **policy** $\pi$ to decide which actions to take in a state:

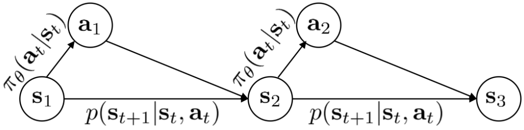
- Deterministic: $\pi(s) = a$
- Stochastic: $\pi(a \mid s) = \mathbb{P}(A = a \mid S = s)$ (our focus)

# Reinforcement learning: objective

The agent uses a **policy** $\pi$ to decide which actions to take in a state:

- Deterministic: $\pi(s) = a$
- Stochastic: $\pi(a \mid s) = \mathbb{P}(A = a \mid S = s)$ (our focus)

A policy $\pi_\theta$ defines a distribution $p_\theta(\tau)$ over **trajectories** $\tau = (a_1, s_1, \ldots, a_T, s_T)$.

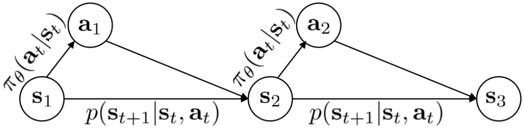


# Reinforcement learning: objective

The agent uses a **policy** $\pi$ to decide which actions to take in a state:

- Deterministic: $\pi(s) = a$
- Stochastic: $\pi(a \mid s) = \mathbb{P}(A = a \mid S = s)$ (our focus)

A policy $\pi_\theta$ defines a distribution $p_\theta(\tau)$ over **trajectories** $\tau = (a_1, s_1, \ldots, a_T, s_T)$.




The agent's **objective** is to learn a policy $\pi_\theta$ (parametrized by $\theta$) that maximizes the expected return:

$$\text{maximize } \mathbb{E}_{\tau \sim p_\theta(\tau)}\left[\sum_{t=1}^{T} r(s_t, a_t)\right]$$

# Sketch of RL algorithms

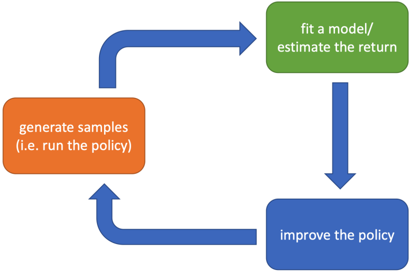



Figure: From Sergey Levine's slides

Key steps:

- **Trial**: run policy to generate trajectories
- **Error**: estimate expected return
- **Learn**: improve the policy

# Sketch of RL algorithms

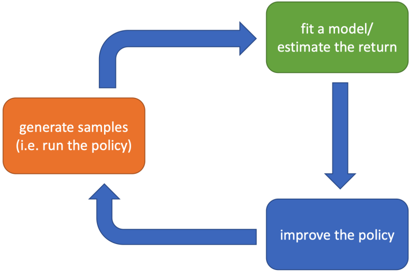



Figure: From Sergey Levine's slides

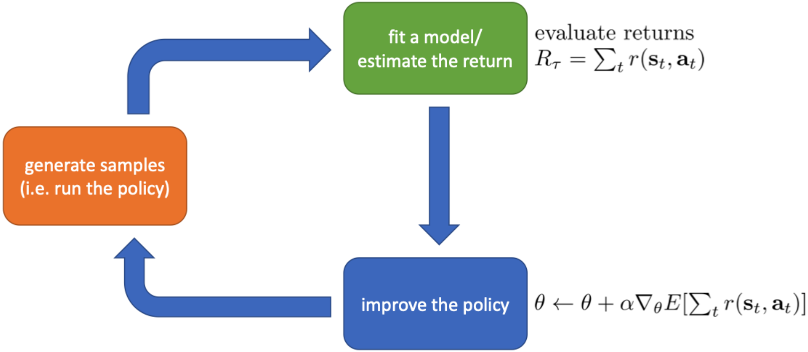
Key steps:

- **Trial**: run policy to generate trajectories
- **Error**: estimate expected return
- **Learn**: improve the policy

Challenges:

- Trials could be expensive (e.g., healthcare, education)
- Reward signal could be sparse (e.g., expert feedback)
- May need many samples to learn a good policy

# Policy gradient algorithms




While not converged

1. Sample trajectories from the current policy
2. Estimate return for each trajectories based on observed rewards
3. Take a gradient step on the expected return (w.r.t. the policy)

# How to compute the gradient?

Notation: let $r(\tau) = \sum_{t=1}^{T} r(s_t, a_t)$ be the return.

# How to compute the gradient?

Notation: let $r(\tau) = \sum_{t=1}^{T} r(s_t, a_t)$ be the return.

$$\text{Our objective:} \quad J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[ r(\tau) \right] = \sum_{\tau} p_\theta(\tau) r(\tau)$$

# How to compute the gradient?

Notation: let $r(\tau) = \sum_{t=1}^{T} r(s_t, a_t)$ be the return.

Our objective: $\quad J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} [r(\tau)] = \sum_\tau p_\theta(\tau) r(\tau)$

$$\begin{aligned}
\nabla_\theta J(\theta) &= \nabla_\theta \sum_\tau p_\theta(\tau) r(\tau) \\
&= \sum_\tau \nabla_\theta p_\theta(\tau) r(\tau) \\
&= \sum_\tau p_\theta(\tau) \nabla_\theta \log p_\theta(\tau) r(\tau) \\
&= \mathbb{E}_{\tau \sim p_\theta(\tau)} [\nabla_\theta \log p_\theta(\tau) r(\tau)]
\end{aligned}$$

**log derivative trick**

$$\begin{aligned}
& p_\theta(\tau) \nabla_\theta \log p_\theta(\tau) \\
=& p_\theta(\tau) \frac{\nabla_\theta p_\theta(\tau)}{p_\theta(\tau)} \\
=& \nabla_\theta p_\theta(\tau)
\end{aligned}$$

# How to compute the gradient?

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[ \nabla_\theta \log p_\theta(\tau) r(\tau) \right]$$ Monte Carlo estimation of the expected gradient

# How to compute the gradient?

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[ \nabla_\theta \log p_\theta(\tau) r(\tau) \right]$$ Monte Carlo estimation of the expected gradient

But what is $p_\theta(\tau)$?

$$p_\theta(\tau) = p_\theta(a_1, s_1, \ldots, a_T, s_T) = p(s_1) \prod_{t=1}^{T} \pi_\theta(a_t \mid s_t) \prod_{t=1}^{T-1} p(s_{t+1} \mid s_t, a_t)$$

# How to compute the gradient?

$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[ \nabla_\theta \log p_\theta(\tau) r(\tau) \right]$ Monte Carlo estimation of the expected gradient

But what is $p_\theta(\tau)$?

$$p_\theta(\tau) = p_\theta(a_1, s_1, \ldots, a_T, s_T) = p(s_1) \prod_{t=1}^{T} \pi_\theta(a_t \mid s_t) \prod_{t=1}^{T-1} p(s_{t+1} \mid s_t, a_t)$$

$$\log p_\theta(\tau) = \log p(s_1) + \sum_{t=1}^{T} \log \pi_\theta(a_t \mid s_t) + \sum_{t=1}^{T-1} \log p(s_{t+1} \mid s_t, a_t)$$

# How to compute the gradient?

$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[ \nabla_\theta \log p_\theta(\tau) r(\tau) \right]$ Monte Carlo estimation of the expected gradient

But what is $p_\theta(\tau)$?

$$p_\theta(\tau) = p_\theta(a_1, s_1, \ldots, a_T, s_T) = p(s_1) \prod_{t=1}^{T} \pi_\theta(a_t \mid s_t) \prod_{t=1}^{T-1} p(s_{t+1} \mid s_t, a_t)$$

$$\log p_\theta(\tau) = \log p(s_1) + \sum_{t=1}^{T} \log \pi_\theta(a_t \mid s_t) + \sum_{t=1}^{T-1} \log p(s_{t+1} \mid s_t, a_t)$$

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[ \left( \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t) \right) \left( \sum_{t=1}^{T} r(s_t, a_t) \right) \right]$$

# Putting everything together

REINFORCE algorithm:

1. Sample $N$ trajectories $\tau^1, \ldots, \tau^N$ from $\pi_\theta$
2. Estimate the gradient:

$$\nabla_\theta J(\theta) \approx \sum_{i=1}^{N} \left( \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t^i \mid s_t^i) \right) \left( \sum_{t=1}^{T} r(s_t^i, a_t^i) \right)$$

3. Update the policy with gradient ascent: $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$
4. Go back to 1

# How is all this related to LLMs?

Think of tokens as actions:

- Action space: vocabulary $a_t = x_t \in \mathcal{V}$
- State space: history / prefix $s_t = (x_1, \ldots, x_{t-1})$
- Policy: a language model $p_\theta(x_t \mid x_{<t})$
- Trajectory: a sentence / generation $x_1, \ldots, x_T$

# How is all this related to LLMs?

REINFORCE algorithm on text:

1. Sample $N$ generations from the language model $p_\theta$
2. Estimate the gradient: $\hat{g} = \frac{1}{N} \sum_{i=1}^{N} \left( \sum_{t=1}^{T} \nabla_\theta \log p_\theta(x_t^i \mid x_{<t}^i) \right) r(x_{1:T})$
3. Update the policy with gradient ascent: $\theta \leftarrow \theta + \alpha \hat{g}$
4. Go back to 1

# How is all this related to LLMs?

REINFORCE algorithm on text:

1. Sample $N$ generations from the language model $p_\theta$
2. Estimate the gradient: $\hat{g} = \frac{1}{N} \sum_{i=1}^{N} \left( \sum_{t=1}^{T} \nabla_\theta \log p_\theta(x_t^i \mid x_{<t}^i) \right) r(x_{1:T})$
3. Update the policy with gradient ascent: $\theta \leftarrow \theta + \alpha \hat{g}$
4. Go back to 1

**What is the algorithm doing?**

If $r(x_{1:T})$ is positive, take a gradient step to increase $p_\theta(x_{1:T})$.
If $r(x_{1:T})$ is negative, take a gradient step to decrease $p_\theta(x_{1:T})$.

*Supervised learning on model generations weighted by return*

# Challenges in policy gradient

$$\hat{g} = \frac{1}{N} \sum_{i=1}^{N} \left( \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t^i \mid s_t^i) \right) \sum_{t=1}^{T} r(s_t, a_t)$$

- We estimate the policy gradient based on a random sample of trajectories
  *$\hat{g}$ is a random variable*

# Challenges in policy gradient

$$\hat{g} = \frac{1}{N} \sum_{i=1}^{N} \left( \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t^i \mid s_t^i) \right) \sum_{t=1}^{T} r(s_t, a_t)$$

- We estimate the policy gradient based on a random sample of trajectories
  *$\hat{g}$ is a random variable*
- This estimator is unbiased
  *In expectation, it is the true policy gradient:* $\mathbb{E}[\hat{g}] = \nabla_\theta J(\theta)$

# Challenges in policy gradient

$$\hat{g} = \frac{1}{N} \sum_{i=1}^{N} \left( \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t^i \mid s_t^i) \right) \sum_{t=1}^{T} r(s_t, a_t)$$

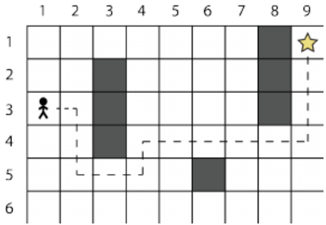
- We estimate the policy gradient based on a random sample of trajectories
  *$\hat{g}$ is a random variable*

- This estimator is unbiased
  *In expectation, it is the true policy gradient:* $\mathbb{E}[\hat{g}] = \nabla_\theta J(\theta)$

- But it has high variance
  *Depending on which trajectories we get, $\hat{g}$ can vary greatly*

# Variance of the policy gradient estimator

$$\hat{g} = \frac{1}{N} \sum_{i=1}^{N} \left( \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t^i \mid s_t^i) \right) \sum_{t=1}^{T} r(s_t, a_t)$$




- Note that every step along the trajectory is multiplied by the same return
- Reward may be sparse and delayed
- The **credit assignment** problem: how do we know which step is responsible for the good/bad outcome?

# Reducing variance: use reward-to-go

- We get a return for the full trajectory $\sum_{t=1}^{T} r(s_t, a_t)$, how to better allocate it to each step $(s_t, a_t)$?
- Future actions $(a_t)$ should not affect past rewards $(r_1, \ldots, r_{t-1})$
- $a_t$ only get rewards after $t$, i.e. **reward-to-go**:

$$\hat{g} = \frac{1}{N} \sum_{i=1}^{N} \left( \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t^i \mid s_t^i) \right) \sum_{j=t}^{T} r(a_j, s_j)$$

# Important quantities related to reward-to-go

- **Q-function**: expected return starting from $s_t$ and taking $a_t$

$$Q^\pi(s_t, a_t) = r(s_t, a_t) + \mathbb{E}_{s_{t+1:T}, a_{t+1:T}} \left[ \sum_{t'=t+1}^{T} r(s_{t'}, a_{t'}) \right]$$

# Important quantities related to reward-to-go

- **Q-function**: expected return starting from $s_t$ and taking $a_t$

$$Q^\pi(s_t, a_t) = r(s_t, a_t) + \mathbb{E}_{s_{t+1:T}, a_{t+1:T}} \left[ \sum_{t'=t+1}^{T} r(s_{t'}, a_{t'}) \right]$$

  - Reward-to-go: single trajectory estimate of the Q-value from $(s_t, a_t)$

# Important quantities related to reward-to-go

- **Q-function**: expected return starting from $s_t$ and taking $a_t$

$$Q^\pi(s_t, a_t) = r(s_t, a_t) + \mathbb{E}_{s_{t+1:T}, a_{t+1:T}} \left[ \sum_{t'=t+1}^{T} r(s_{t'}, a_{t'}) \right]$$

  - Reward-to-go: single trajectory estimate of the Q-value from $(s_t, a_t)$

- **Value function**: expected return starting from $s_t$

$$V^\pi(s_t) = \mathbb{E}_{a_t \sim \pi(\cdot|s_t)} \left[ Q^\pi(s_t, a_t) \right]$$

# Important quantities related to reward-to-go

- **Q-function**: expected return starting from $s_t$ and taking $a_t$

$$Q^\pi(s_t, a_t) = r(s_t, a_t) + \mathbb{E}_{s_{t+1:T}, a_{t+1:T}}\left[\sum_{t'=t+1}^{T} r(s_{t'}, a_{t'})\right]$$

  - Reward-to-go: single trajectory estimate of the Q-value from $(s_t, a_t)$

- **Value function**: expected return starting from $s_t$

$$V^\pi(s_t) = \mathbb{E}_{a_t \sim \pi(\cdot|s_t)}\left[Q^\pi(s_t, a_t)\right]$$

- Concept check: what is $\mathbb{E}_{s_1}\left[V^\pi(s_1)\right]$?

# Reducing variance: subtract a baseline

- Subtract a **baseline** from the return:

$$\hat{g} = \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_t^i \mid s_t^i)(r(\tau^i) - b(s_t^i))\right)$$

$b(\cdot)$: a function of the state or some constant

# Reducing variance: subtract a baseline

- Subtract a **baseline** from the return:

$$\hat{g} = \frac{1}{N} \sum_{i=1}^{N} \left( \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t^i \mid s_t^i)(r(\tau^i) - b(s_t^i)) \right)$$

  $b(\cdot)$: a function of the state or some constant

- Intuition: the return may not be due to the action you take but just because you are in a state ”closer” to the goal

# Reducing variance: subtract a baseline

- Subtract a **baseline** from the return:

$$\hat{g} = \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_t^i \mid s_t^i)(r(\tau^i) - b(s_t^i))\right)$$

  $b(\cdot)$: a function of the state or some constant

- Intuition: the return may not be due to the action you take but just because you are in a state "closer" to the goal
- By subtracting a baseline, we measure how much better the action is than the typical return

# Reducing variance: subtract a baseline

- Subtract a **baseline** from the return:

$$\hat{g} = \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_t^i \mid s_t^i)(r(\tau^i) - b(s_t^i))\right)$$

  $b(\cdot)$: a function of the state or some constant

- Intuition: the return may not be due to the action you take but just because you are in a state "closer" to the goal
- By subtracting a baseline, we measure how much better the action is than the typical return
- A simple choice is the average return

$$b(s) = \frac{1}{N}\sum_{i=1}^{N} r(\tau^i)$$

# Reducing variance: subtract a baseline

Does this change our objective?

- $\hat{g}$ is still an **unbiased** estimator, i.e.

$$\mathbb{E}\left[\nabla_\theta \log \pi_\theta(a_t \mid s_t)(r(\tau) - b(s_t))\right] = \mathbb{E}\left[\nabla_\theta \log \pi_\theta(a_t \mid s_t) r(\tau)\right]$$

# Reducing variance: subtract a baseline

Does this change our objective?

- $\hat{g}$ is still an **unbiased** estimator, i.e.

$$\mathbb{E}\left[\nabla_\theta \log \pi_\theta(a_t \mid s_t)(r(\tau) - b(s_t))\right] = \mathbb{E}\left[\nabla_\theta \log \pi_\theta(a_t \mid s_t) r(\tau)\right]$$

$$\mathbb{E}\Big[\nabla_\theta \log \pi_\theta(a_t \mid s_t)\, b(s_t)\Big] = \mathbb{E}_{s_t}\Big[b(s_t)\, \mathbb{E}_{a_t \sim \pi_\theta(\cdot \mid s_t)}\big[\nabla_\theta \log \pi_\theta(a_t \mid s_t)\big]\Big]$$

# Reducing variance: subtract a baseline

Does this change our objective?

- $\hat{g}$ is still an **unbiased** estimator, i.e.

$$\mathbb{E}\left[\nabla_\theta \log \pi_\theta(a_t \mid s_t)(r(\tau) - b(s_t))\right] = \mathbb{E}\left[\nabla_\theta \log \pi_\theta(a_t \mid s_t) r(\tau)\right]$$

$$\mathbb{E}\left[\nabla_\theta \log \pi_\theta(a_t \mid s_t)\, b(s_t)\right] = \mathbb{E}_{s_t}\left[b(s_t)\, \mathbb{E}_{a_t \sim \pi_\theta(\cdot \mid s_t)}\left[\nabla_\theta \log \pi_\theta(a_t \mid s_t)\right]\right]$$

$$\begin{aligned}\mathbb{E}\left[\nabla_\theta \log \pi_\theta(a_t \mid s_t)\right] &= \sum_{a_t} \pi_\theta(a_t \mid s_t) \nabla_\theta \log \pi_\theta(a_t \mid s_t) = \sum_{a_t} \nabla_\theta \pi_\theta(a_t \mid s_t) \\ &= \nabla_\theta \sum_{a_t} \pi_\theta(a_t \mid s_t) = \nabla_\theta 1 = 0\end{aligned}$$

- As long as $b(\cdot)$ does not depend on $a_t$, it doesn't introduce any bias.

# The advantage function

- **Advantage function**: how much better $a_t$ is compared to other actions in state $s_t$

$$A^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t)$$

reward-to-go minus a (state-dependent) baseline

# The advantage function

- **Advantage function**: how much better $a_t$ is compared to other actions in state $s_t$

$$A^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t)$$

reward-to-go minus a (state-dependent) baseline

- How to estimate advantage in policy gradient?

$$\hat{g} = \frac{1}{N} \sum_{i=1}^{N} \left( \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t^i \mid s_t^i) \right) \left( \sum_{t=1}^{T} r(s_t, a_t) - b \right)$$

high variance because it's estimated from a single trajectory

# The advantage function

- **Advantage function**: how much better $a_t$ is compared to other actions in state $s_t$

$$A^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t)$$

reward-to-go minus a (state-dependent) baseline

- How to estimate advantage in policy gradient?

$$\hat{g} = \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_t^i \mid s_t^i)\right)\left(\sum_{t=1}^{T} r(s_t, a_t) - b\right)$$

high variance because it's estimated from a single trajectory

- We can improve the estimate by sampling more rollouts from $(s_t, a_t)$

# The advantage function

- **Advantage function**: how much better $a_t$ is compared to other actions in state $s_t$

$$A^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t)$$

  reward-to-go minus a (state-dependent) baseline

- How to estimate advantage in policy gradient?

$$\hat{g} = \frac{1}{N} \sum_{i=1}^{N} \left( \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t^i \mid s_t^i) \right) \left( \sum_{t=1}^{T} r(s_t, a_t) - b \right)$$

  high variance because it's estimated from a single trajectory

- We can improve the estimate by sampling more rollouts from $(s_t, a_t)$
- Can we fit a function to predict $A^\pi$?

# The advantage function

- **Advantage function**: how much better $a_t$ is compared to other actions in state $s_t$
  $$A^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t)$$
  reward-to-go minus a (state-dependent) baseline
- How to estimate advantage in policy gradient?
  $$\hat{g} = \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_t^i \mid s_t^i)\right)\left(\sum_{t=1}^{T} r(s_t, a_t) - b\right)$$
  high variance because it's estimated from a single trajectory
- We can improve the estimate by sampling more rollouts from $(s_t, a_t)$
- Can we fit a function to predict $A^\pi$?
  - How does this solve the high variance problem?

# Fitting the value function

- Rewrite the advantage function

$$\begin{aligned} A^\pi(s_t, a_t) &= Q^\pi(s_t, a_t) - V^\pi(s_t) \\ &= r(s_t, a_t) + V^\pi(s_{t+1}) - V^\pi(s_t) \end{aligned}$$

# Fitting the value function

- Rewrite the advantage function

$$\begin{aligned} A^{\pi}(s_t, a_t) &= Q^{\pi}(s_t, a_t) - V^{\pi}(s_t) \\ &= r(s_t, a_t) + V^{\pi}(s_{t+1}) - V^{\pi}(s_t) \end{aligned}$$

- So we just need to fit the value function!

# Fitting the value function

- Rewrite the advantage function

$$
\begin{aligned}
A^\pi(s_t, a_t) &= Q^\pi(s_t, a_t) - V^\pi(s_t) \\
&= r(s_t, a_t) + V^\pi(s_{t+1}) - V^\pi(s_t)
\end{aligned}
$$

- So we just need to fit the value function!
- Estimate the value from a rollout $\tau^i$

$$
\hat{V}^\pi(s_t^i) = \sum_{t'=t}^{T} r(s_{t'}^i, a_{t'}^i)
$$

# Fitting the value function

- Rewrite the advantage function

$$\begin{aligned} A^\pi(s_t, a_t) &= Q^\pi(s_t, a_t) - V^\pi(s_t) \\ &= r(s_t, a_t) + V^\pi(s_{t+1}) - V^\pi(s_t) \end{aligned}$$

- So we just need to fit the value function!
- Estimate the value from a rollout $\tau^i$

$$\hat{V}^\pi(s_t^i) = \sum_{t'=t}^{T} r(s_{t'}^i, a_{t'}^i)$$

- This gives us a training example $(s_t^i, \hat{V}^\pi(s_t^i))$

# Fitting the value function

- Rewrite the advantage function

$$\begin{aligned} A^\pi(s_t, a_t) &= Q^\pi(s_t, a_t) - V^\pi(s_t) \\ &= r(s_t, a_t) + V^\pi(s_{t+1}) - V^\pi(s_t) \end{aligned}$$

- So we just need to fit the value function!
- Estimate the value from a rollout $\tau^i$

$$\hat{V}^\pi(s_t^i) = \sum_{t'=t}^{T} r(s_{t'}^i, a_{t'}^i)$$

- This gives us a training example $(s_t^i, \hat{V}^\pi(s_t^i))$
- Given a training set $\mathcal{D}$, train a regression model $w$ by minimizing the squared loss:

$$\sum_{s \in \mathcal{D}} \left( V_w^\pi(s) - \hat{V}^\pi(s) \right)^2$$

# Actor-Critic methods

- **Critic**: evaluate the policy
  update $w$ to improve estimates of $V_w$
- **Actor**: improve the policy
  update $\theta$ to improve the policy give feedback from the critic

# Actor-Critic methods

- **Critic**: evaluate the policy
  update $w$ to improve estimates of $V_w$
- **Actor**: improve the policy
  update $\theta$ to improve the policy give feedback from the critic

Algorithm sketch:

1. Sample a trajectory from current policy
2. Update $w$ given the estimated state values
3. Evaluate $\hat{A}^\pi(s_t, a_t) = r(s_t, a_t) + V_w^\pi(s_{t+1}) - V_w^\pi(s_t)$
4. Update the policy with gradient $\hat{g} = \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t) \hat{A}^\pi(s_t, a_t)$
5. Go back to 1

# On-Policy vs. Off-Policy

- **On-Policy** methods use samples from the policy that is currently being optimized (i.e. $\pi_\theta(a \mid s)$).
- **Off-Policy** methods use samples generated by a different behavior policy $\mu(a \mid s)$.
- Advantages of off-policy:

# On-Policy vs. Off-Policy

- **On-Policy** methods use samples from the policy that is currently being optimized (i.e. $\pi_\theta(a \mid s)$).
- **Off-Policy** methods use samples generated by a different behavior policy $\mu(a \mid s)$.
- Advantages of off-policy:
  - Can reuse old trajectories generated by past policies.
  - Allows learning from demonstrations or replay buffers.
  - Can have multiple exploration policies to generate trajectories.

# Off-Policy Policy Gradient

- Standard on-policy gradient (REINFORCE) is:

$$\nabla_\theta J(\theta) \;=\; \mathbb{E}_{\tau\sim\pi_\theta}\left[\nabla_\theta \log \pi_\theta(\tau)\, r(\tau)\right].$$

- Off-policy scenario: $\tau$ is generated by a behavior policy $\mu(\tau)$, but we want gradient w.r.t. $\pi_\theta$.

$$\nabla_\theta J(\theta) \;\neq\; \mathbb{E}_{\tau\sim\mu}\left[\nabla_\theta \log \pi_\theta(\tau)\, R(\tau)\right]$$

- Key challenge: Correctly handling the discrepancy between $\mu$ and $\pi_\theta$.

# Importance Sampling: A Quick Review

- We often want an expectation under some distribution $p(x)$, but we only have samples from a different distribution $q(x)$.
- **Importance Sampling** uses a correction ratio:

$$\mathbb{E}_{x\sim p}[f(x)] = \mathbb{E}_{x\sim q}\left[f(x)\,\frac{p(x)}{q(x)}\right].$$

- In the RL context:

$$\frac{p(\tau)}{q(\tau)} \rightarrow \frac{\pi_\theta(\tau)}{\mu(\tau)}$$

- This ratio re-weights off-policy trajectories to match the target policy distribution.

# Off-Policy Policy Gradient

- Off-policy scenario: $\tau$ is generated by a behavior policy $\mu(\tau)$, but we want gradient w.r.t. $\pi_\theta$.

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\mu}\Big[\rho(\tau)\,\nabla_\theta \log \pi_\theta(\tau)\,R(\tau)\Big], \quad \text{where} \quad \rho(\tau) = \frac{\pi_\theta(\tau)}{\mu(\tau)}.$$

- Usually we re-weight per step:

$$\hat{g} = \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\frac{\pi_\theta(a_t^i \mid s_t^i)}{\mu(a_t^i \mid s_t^i)}\nabla_\theta \log \pi_\theta(a_t^i \mid s_t^i)\right)\sum_{t=1}^{T} r(s_t^i, a_t^i)$$

- Importance weight also incurs high variance: when will the ratio blow up?
- $\pi_\theta$ and $\mu$ cannot be too different

# Summary

- Reinforcement learning: maximize return through trial and error (rollout, evaluate policy, improve policy)
- Policy gradient: increase/decrease likelihood of the trajectory proportionally to the return
- Key challenge: gradient is very noisy!
  - Use reward-to-go for each action update
  - Subtract a baseline
  - Use function approximation for the value/Q function