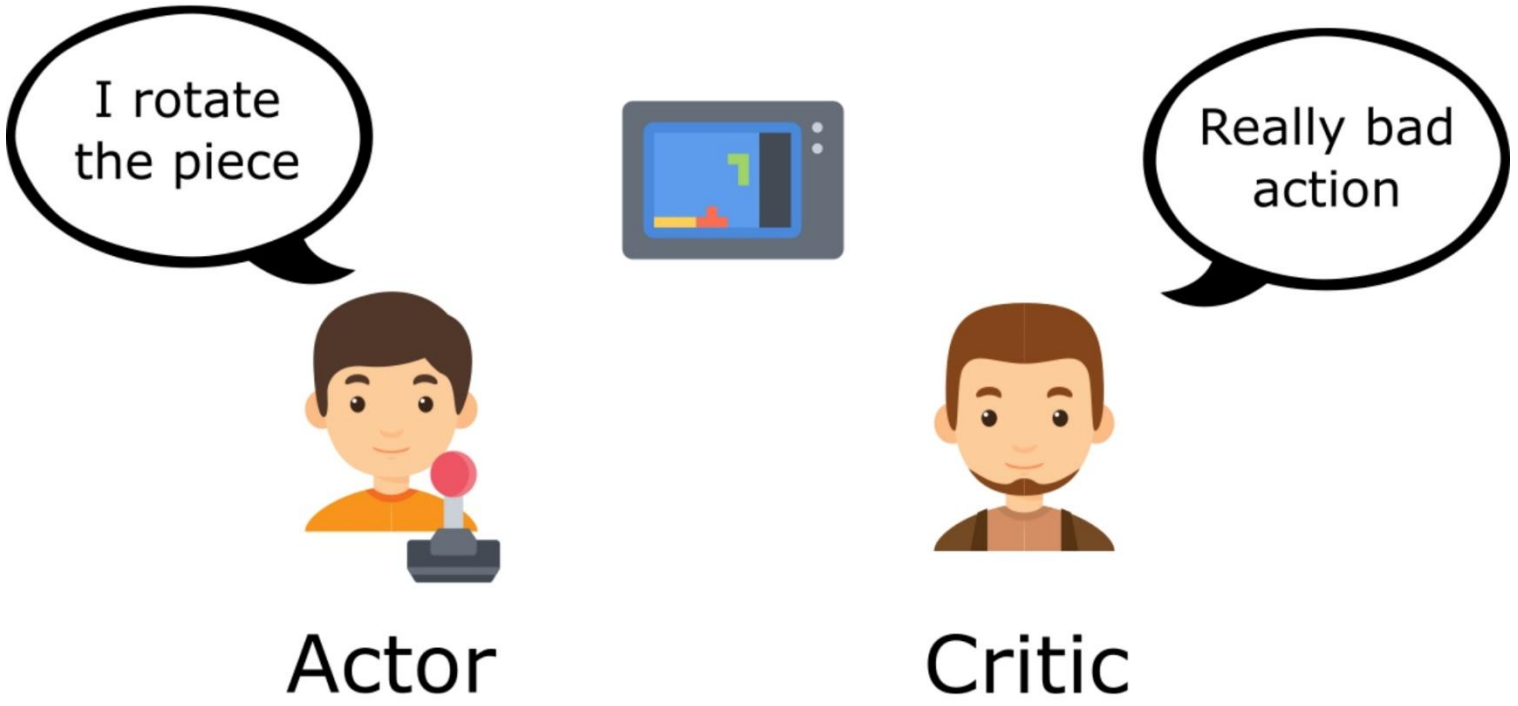
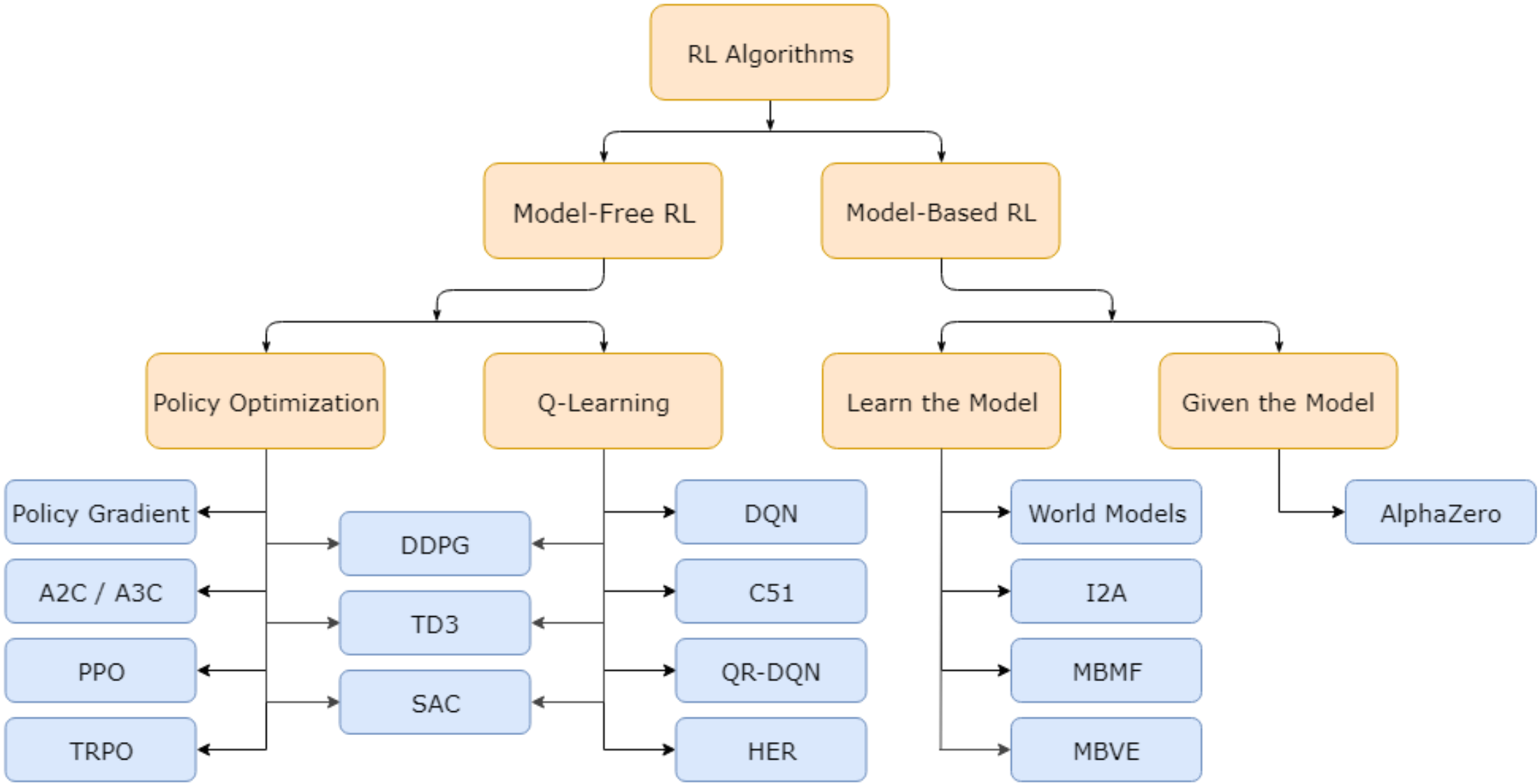
PPO and GRPO

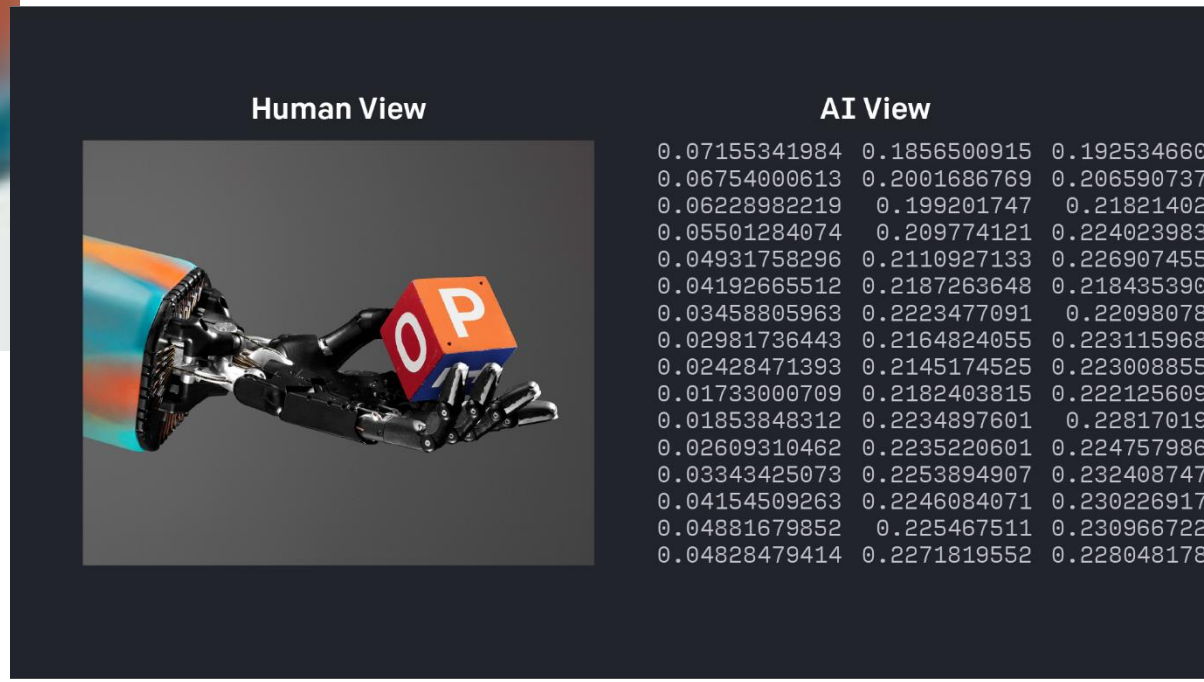


Instructor: Daniel Brown --- University of Utah

Rough Taxonomy of RL Algorithms



Dexterous Manipulation



OpenAI 5: DOTA 2

Human View

AI View

3.006	-1.386	-0.4695	0.883	1	0.84
-0.3154	-0.5425	-0.5	0.866	0	0.82
3.11	-1.36	-0.9336	0.3584	1	0.78
-2.324	2.863	0.9746	0.225	0	0.86
3.037	-1.361	-0.7773	0.6294	1	0.82
-1.387	2.951	0.988	0.1565	0	0.74
3.023	-0.9395	0.05234	-0.9985	0	0.66
2.951	-0.5747	0.01746	1	0	0.72
2.963	-1.303	0.3906	0.9204	0	0.68
2.834	-3.164	0.01746	-1	0	0.68
3.127	-1.368	0.6562	0.755	1	0.55
3.088	-1.366	0.4695	0.883	0	0.55
2.984	-1.398	-0.225	0.9746	1	0.55
3.037	-1.391	0.788	0.6157	0	0.55
3.076	-1.438	0.883	0.4695	0	0.55
-2.412	2.846	0.996	0.08716	1	0.3

RLHF in ChatGPT

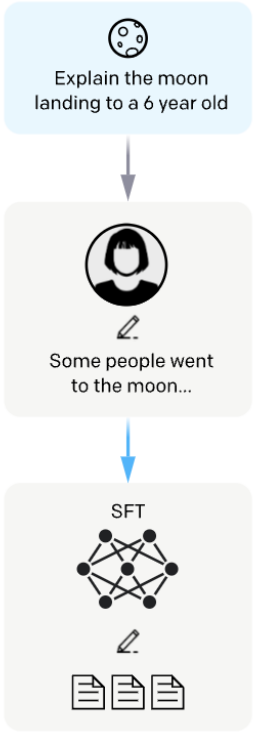
Step 1

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.



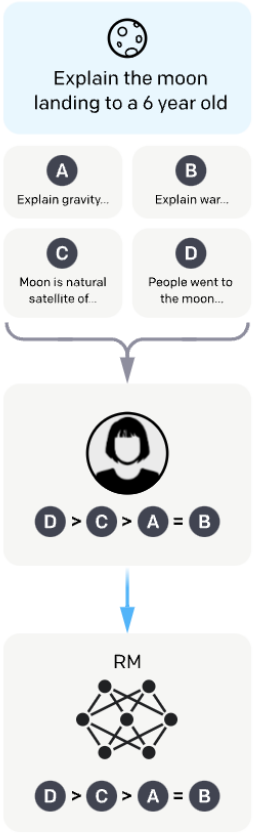
Step 2

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

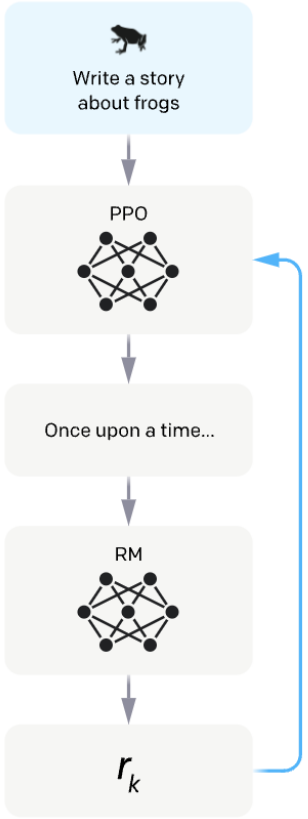
Optimize a policy against the reward model using reinforcement learning.

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.



What is the goal of RL?

- Find a policy that maximizes expected utility (discounted cumulative rewards)

$$\pi^* = arg\max_{\pi} E_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t R(s, \pi(s), s')\right]$$

The Policy Gradient (REINFORCE)

- We can now perform gradient ascent to improve our policy!

$$\theta_{k+1} \leftarrow \theta_k + \alpha \nabla_\theta J(\pi_\theta) \Big|_{\theta_k}$$

$$\nabla_\theta J(\pi_\theta) = E_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \; R(\tau) \right]$$

Estimate with a sample mean over a set D of policy rollouts given current parameters

$$\approx \frac{1}{|D|} \sum_{\tau \in D} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \; R(\tau)$$

Many forms of policy gradients

$$\nabla_\theta J(\pi_\theta) = \mathop{\mathrm{E}}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)\Phi_t\right]$$

$$\Phi_t = R(\tau), \qquad \Phi_t = \sum_{t'=t}^{T} R(s_{t'}, a_{t'}, s_{t'+1}), \qquad \Phi_t = Q^{\pi_\theta}(s_t, a_t)$$

$$\Phi_t = \sum_{t'=t}^{T} R(s_{t'}, a_{t'}, s_{t'+1}) - b(s_t)$$

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

Advantage Function

Generalized Advantage Estimation (GAE)

Published as a conference paper at ICLR 2016

HIGH-DIMENSIONAL CONTINUOUS CONTROL USING GENERALIZED ADVANTAGE ESTIMATION

John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan and Pieter Abbeel
Department of Electrical Engineering and Computer Science
University of California, Berkeley
{joschu,pcmoritz,levine,jordan,pabbeel}@eecs.berkeley.edu

Motivation

- Can we construct all possible n-step returns and average them?

$$
\begin{aligned}
\hat{A}_t^{(1)} &= r_t + \gamma V(s_{t+1}) - V(s_t) \\
\hat{A}_t^{(2)} &= r_t + \gamma r_{t+1} + \gamma^2 V(s_{t+2}) - V(s_t) \\
\cdots &= \cdots \\
\hat{A}_t^{(\infty)} &= r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots - V(s_t)
\end{aligned}
$$

Tangent: Bias and Variance

- Bias:
  - On average, how far is your estimate from the truth?

$$\text{Bias}(\hat{\theta}) = \mathbb{E}[\hat{\theta}] - \theta$$

- Variance
  - How much do your estimates fluctuate around their average?

$$\text{Var}(\hat{\theta}) = \mathbb{E}\left[(\hat{\theta} - \mathbb{E}[\hat{\theta}])^2\right]$$

Motivation

- Can we construct all possible n-step returns and average them?

$$
\begin{aligned}
\hat{A}_t^{(1)} &= r_t + \gamma V(s_{t+1}) - V(s_t) \\
\hat{A}_t^{(2)} &= r_t + \gamma r_{t+1} + \gamma^2 V(s_{t+2}) - V(s_t) \\
\cdots &= \cdots \\
\hat{A}_t^{(\infty)} &= r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots - V(s_t)
\end{aligned}
$$

$$
\hat{A}_n^{\pi}(\mathbf{s}_t, \mathbf{a}_t) = \sum_{t'=t}^{t+n} \gamma^{t'-t} r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) - \hat{V}_{\phi}^{\pi}(\mathbf{s}_t) + \gamma^n \hat{V}_{\phi}^{\pi}(\mathbf{s}_{t+n})
$$

Smaller n results in lower variance (smaller number of things we sum), but higher bias (more bootstrapping)!

Motivation

- Can we construct all possible n-step returns and average them?

$$\hat{A}_n^\pi(\mathbf{s}_t, \mathbf{a}_t) = \sum_{t'=t}^{t+n} \gamma^{t'-t} r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) - \hat{V}_\phi^\pi(\mathbf{s}_t) + \gamma^n \hat{V}_\phi^\pi(\mathbf{s}_{t+n})$$

Smaller n results in lower variance, but higher bias

$$\hat{A}_{\text{GAE}}^\pi(\mathbf{s}_t, \mathbf{a}_t) = \sum_{n=1}^{\infty} w_n \hat{A}_n^\pi(\mathbf{s}_t, \mathbf{a}_t)$$

weighted combination of n-step returns

$w_n \propto \lambda^{n-1}$ exponential falloff where $\lambda \in [0,1]$

In practice $\lambda \approx 0.95$

Just a 1-step TD error

$$\hat{A}_{\text{GAE}}^\pi(\mathbf{s}_t, \mathbf{a}_t) = \sum_{t'=t}^{\infty} (\gamma\lambda)^{t'-t} \delta_{t'} \qquad \delta_{t'} = r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) + \gamma \hat{V}_\phi^\pi(\mathbf{s}_{t'+1}) - \hat{V}_\phi^\pi(\mathbf{s}_{t'})$$

similar effect as discount!

GAE Pseudo Code

```
#predict values based on sequence of states in a trajectory

vals = predict_values(states)

# the next two lines implement GAE-Lambda advantage calculation

deltas = rews[:-1] + gamma * vals[1:] - vals[:-1]

gae = discount_cumsum(deltas, gamma * lam)
```

Proximal Policy Optimization (PPO)

Proximal Policy Optimization Algorithms

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov
OpenAI
{joschu, filip, prafulla, alec, oleg}@openai.com

Proximal Policy Optimization (PPO)

- One of the most popular deep RL algorithms
- Used to train ChatGPT and other LLMs

Motivation:

- Many Policy Gradient algorithms have stability problems.
- This can be avoided if we avoid making too big of a policy update.

https://huggingface.co/blog/deep-rl-ppo

https://huggingface.co/blog/deep-rl-ppo

We want to take multiple gradient steps

- What is the problem?

Tangent: Importance Sampling

- Problem:
  - We want to compute $\mathbb{E}_{x\sim p}[f(x)] = \int f(x)\, p(x)\, dx$
  - Without samples from p(x)
  - All we have are samples from q(x)
- Trick
  - Multiply and divide by q(x)

Importance weight

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

Tangent: Importance Sampling

- Problem:
  - We want to compute $\mathbb{E}_{x\sim p}[f(x)] = \int f(x)\, p(x)\, dx$
  - Without samples from π_θ
  - All we have are samples from π_{old}

$$r(\theta) = \frac{\pi_\theta(a|s)}{\pi_{\text{old}}(a|s)} \qquad \nabla J = \mathbb{E}_{\pi_{\text{old}}}\left[r(\theta)\nabla \log \pi_\theta(a|s) A\right]$$

  - This works, but has super high variance! Why?

Proximal Policy Iteration (PPO)

- Measure how much we are changing policy compared with previous policy using a ratio:

$$ratio_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)}$$

- Clip policy gradient update based on this ratio:

$$\theta_{k+1} = \arg\max_\theta \mathop{\mathrm{E}}_{s,a\sim\pi_{\theta_k}} [L(s, a, \theta_k, \theta)]$$

$$L(s, a, \theta_k, \theta) = \min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}}(s, a),\ \ \text{clip}\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, 1-\epsilon, 1+\epsilon\right) A^{\pi_{\theta_k}}(s, a)\right)$$

Proximal Policy Iteration (PPO)

- "Simpler" way to write clip objective:

$$L(s, a, \theta_k, \theta) = \min \left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}}(s, a), \quad g(\epsilon, A^{\pi_{\theta_k}}(s, a)) \right)$$

where

$$g(\epsilon, A) = \begin{cases} (1+\epsilon)A & A \geq 0 \\ (1-\epsilon)A & A < 0 \end{cases}$$

https://spinningup.openai.com/en/latest/algorithms/ppo.html

Proximal Policy Iteration (PPO)

- Simpler way to write clip objective:

$$L(s, a, \theta_k, \theta) = \min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}}(s, a),\quad g(\epsilon, A^{\pi_{\theta_k}}(s, a))\right)$$

where

$$g(\epsilon, A) = \begin{cases} (1+\epsilon)A & A \geq 0 \\ (1-\epsilon)A & A < 0 \end{cases}$$

What if the advantage is positive?

$$L(s, a, \theta_k, \theta) = \min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, (1+\epsilon)\right) A^{\pi_{\theta_k}}(s, a)$$

We want to increase $\pi_\theta(a|s)$, but not too much!

Once $\pi_\theta(a|s) > (1+\epsilon)\pi_{\theta_k}(a|s)$ the min kicks in and limits our policy update.

https://spinningup.openai.com/en/latest/algorithms/ppo.html

Proximal Policy Iteration (PPO)

- Simpler way to write clip objective:

$$L(s, a, \theta_k, \theta) = \min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}}(s, a),\quad g(\epsilon, A^{\pi_{\theta_k}}(s, a))\right)$$

where

$$g(\epsilon, A) = \begin{cases} (1+\epsilon)A & A \geq 0 \\ (1-\epsilon)A & A < 0 \end{cases}$$

What if the advantage is negative?

$$L(s, a, \theta_k, \theta) = \max\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, (1-\epsilon)\right) A^{\pi_{\theta_k}}(s, a)$$

We want to decrease $\pi_\theta(a|s)$, but not too much!

Once $\pi_\theta(a|s) < (1-\epsilon)\pi_{\theta_k}(a|s)$ the max kicks in and limits our policy update.

https://spinningup.openai.com/en/latest/algorithms/ppo.html

Algorithm 1 PPO-Clip

Input: initial policy parameters θ_0, initial value function parameters ϕ_0

for $k = 0, 1, 2, \ldots$ **do**

Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.

Compute rewards-to-go $\hat{R}_t$.

Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based on the current value function V_{ϕ_k}.

Update the policy by maximizing the PPO-Clip objective:

$$\theta_{k+1} = \arg\max_{\theta} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t, a_t), \quad g(\epsilon, A^{\pi_{\theta_k}}(s_t, a_t)) \right),$$

typically via stochastic gradient ascent with Adam.

Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t \right)^2,$$

typically via some gradient descent algorithm.

end for

Lots of other tricks used

- Additional advantage normalization
- Early stopping with KL-divergence
- Etc.

The 37 Implementation Details of Proximal Policy Optimization: https://iclr-blog-track.github.io/2022/03/25/ppo-implementation-details/

Group Relative Policy Optimization (GRPO)

DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning

DeepSeek-AI

research@deepseek.com

DeepSeekR1-Zero

- Directly applies RL to the base model without SFT
- Allows the model to explore chain-of-thought (CoT) for solving complex problems
- Demonstrates capabilities such as self-verification, reflection, and generating long CoTs
- First open research to validate that reasoning capabilities of LLMs can be incentivized purely through RL, without the need for SFT.

DeepSeek

- Uses rule-based reward based on correct answers.
- Works well for math, code, and STEM questions with deterministic answers.
- Also uses heuristic reward to require the following format in answers:

```
<think> ... reasoning steps ... </think>
<answer> final result </answer>
```

Group Relative Policy Optimization (GRPO)

No critic model -> massive memory and compute savings

Step 1: **Sample a group of outputs**

- For a given prompt q, the **old policy** π_{old} samples a **group** of G outputs:

$$\{o_1, o_2, \ldots, o_G\}$$

Step 2: **Evaluate each output**

- For each o_i, compute a **reward** r_i (via rule-based evaluators — correctness and formatting).

Step 3: **Compute relative advantage**

- Normalize the rewards within the group to get the **advantage** A_i:

$$A_i = \frac{r_i - \text{mean}(\{r_1, \ldots, r_G\})}{\text{std}(\{r_1, \ldots, r_G\})}$$

Just using RL leads to “learning how to think”

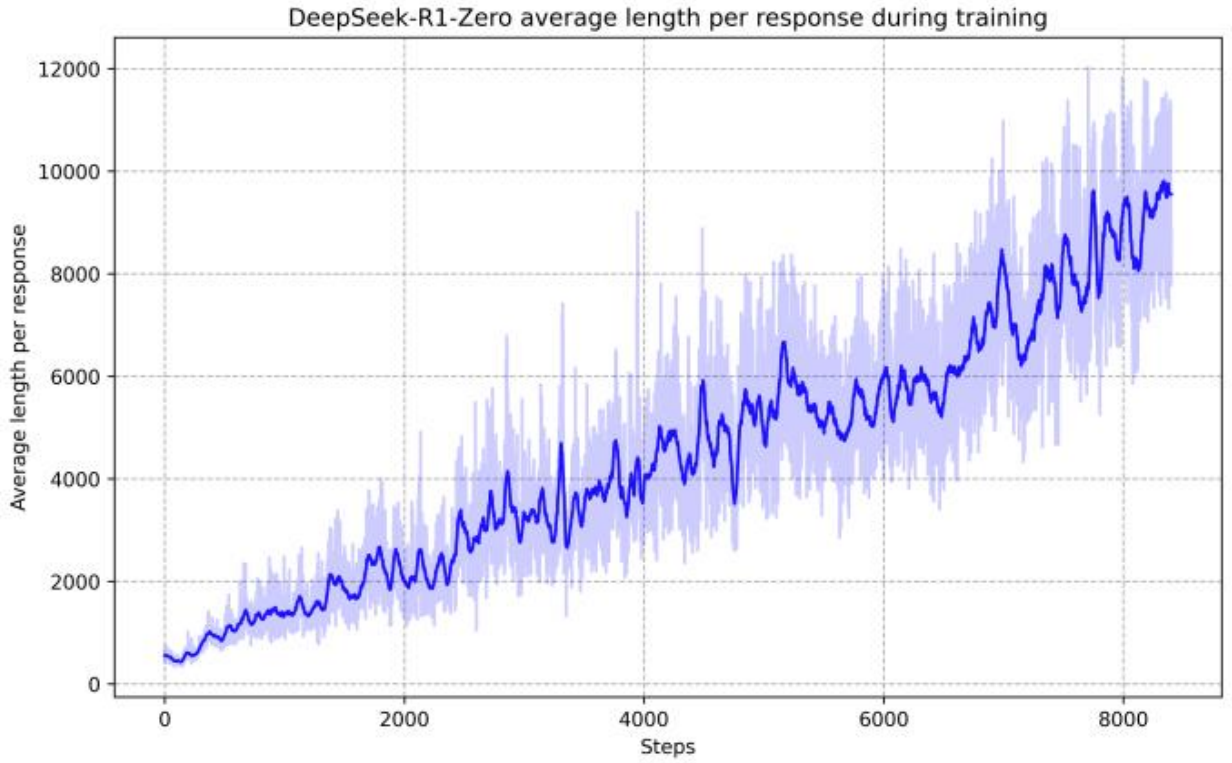


Figure 3 | The average response length of DeepSeek-R1-Zero on the training set during the RL process. DeepSeek-R1-Zero naturally learns to solve reasoning tasks with more thinking time.

Aha moments

Question: If $a > 1$, then the sum of the real solutions of $\sqrt{a - \sqrt{a+x}} = x$ is equal to
Response: <think> To solve the equation $\sqrt{a - \sqrt{a+x}} = x$, let's start by squaring both $\cdots$ $\left(\sqrt{a - \sqrt{a+x}}\right)^2 = x^2 \implies a - \sqrt{a+x} = x^2.$ Rearrange to isolate the inner square root term: $(a - x^2)^2 = a + x \implies a^2 - 2ax^2 + (x^2)^2 = a + x \implies x^4 - 2ax^2 - x + (a^2 - a) = 0$ $\ldots$ Wait, wait. Wait. That's an aha moment I can flag here. Let's reevaluate this step-by-step to identify if the correct sum can be $\cdots$ We started with the equation: $\sqrt{a - \sqrt{a+x}} = x$ First, let's square both sides: $a - \sqrt{a+x} = x^2 \implies \sqrt{a+x} = a - x^2$ Next, I could square both sides again, treating the equation: $\cdots$ $\ldots$

Table 3 | An interesting "aha moment" of an intermediate version of DeepSeek-R1-Zero. The model learns to rethink using an anthropomorphic tone. This is also an aha moment for us, allowing us to witness the power and beauty of reinforcement learning.

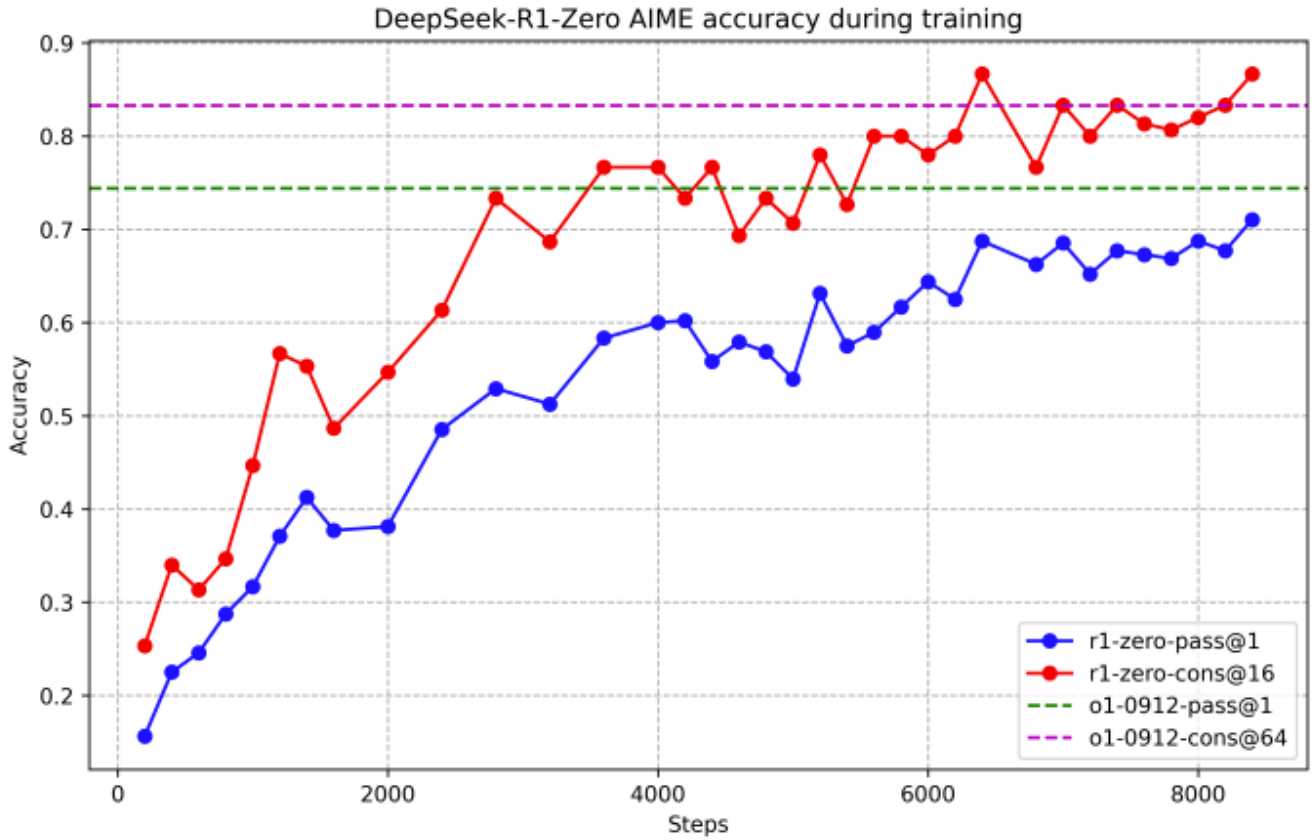


Figure 2 | AIME accuracy of DeepSeek-R1-Zero during training. For each question, we sample 16 responses and calculate the overall average accuracy to ensure a stable evaluation.

Metric	Meaning
AIME Accuracy	Accuracy on hard math problems inspired by the AIME exam
Pass@1	Is the **first** model output correct? (Strict single-sample score)
Cons@k	Is the **most common** answer across **k samples** correct? (Majority vote)

Full DeepSeek model

- Adds back SFT
- Adds a learned reward from preferences and combines that with the rule-based reward
- Uses other tricks to train using smaller GPUs and less memory