How to AI (Almost) Anything

Lecture 11 – Reinforcement Learning and Interaction

Paul Liang
Assistant Professor
MIT Media Lab & MIT EECS

https://pliang279.github.io
ppliang@mit.edu
@pliang279

Assignments for This Coming Week

This Thursday (5/8): final project presentations.

- Class from 1-3pm, let us know any time constraints.

Final project reports due 5/20 – 12 days to incorporate feedback from presentations

Meet with me and TAs today after class.

Lecture Topics

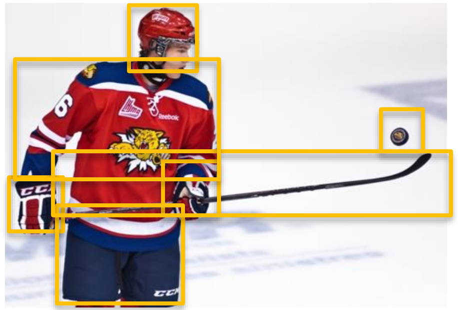
(subject to change, based on student interests and course discussions)

Module 1: Foundations of AI

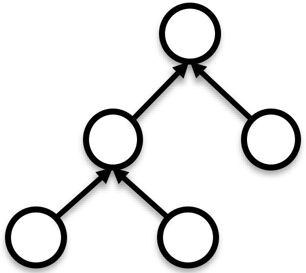
Week 1 (2/4): Introduction to AI and AI research

Week 2 (2/11): Data, structure, and information

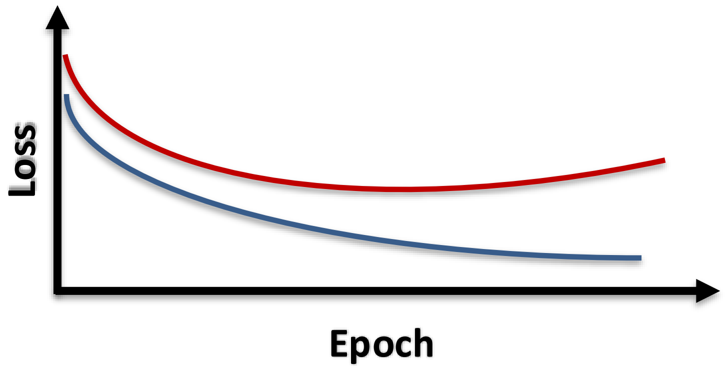
Week 4 (2/25): Common model architectures



Spatial



Hierarchical



multisensory intelligence

Lecture Topics

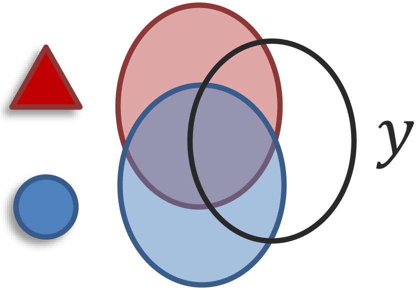
(subject to change, based on student interests and course discussions)

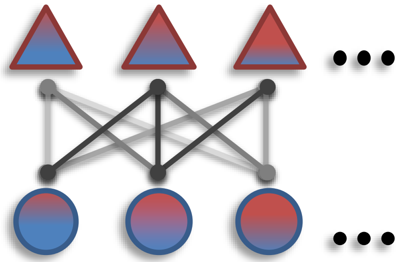
Module 2: Foundations of multimodal AI

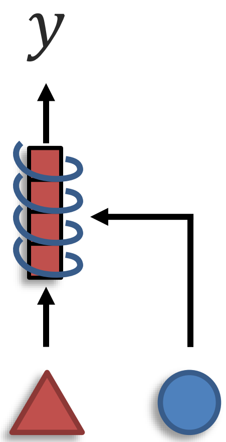
Week 5 (3/4): Multimodal connections and alignment

Week 6 (3/11): Multimodal interactions and fusion

Week 7 (3/18): Cross-modal transfer







multisensory intelligence

Lecture Topics

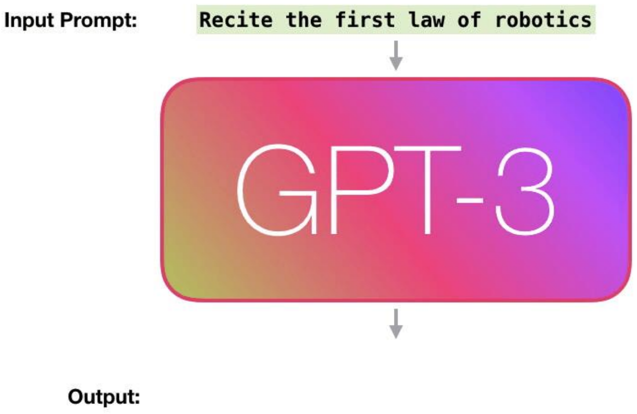
(subject to change, based on student interests and course discussions)

Module 3: Large models and modern AI

Week 9 (4/1): Pre-training, scaling, fine-tuning LLMs

Week 11 (4/15): Large multimodal models

Week 12 (4/22): Modern generative AI



An armchair in the shape of an avocado

multisensory intelligence

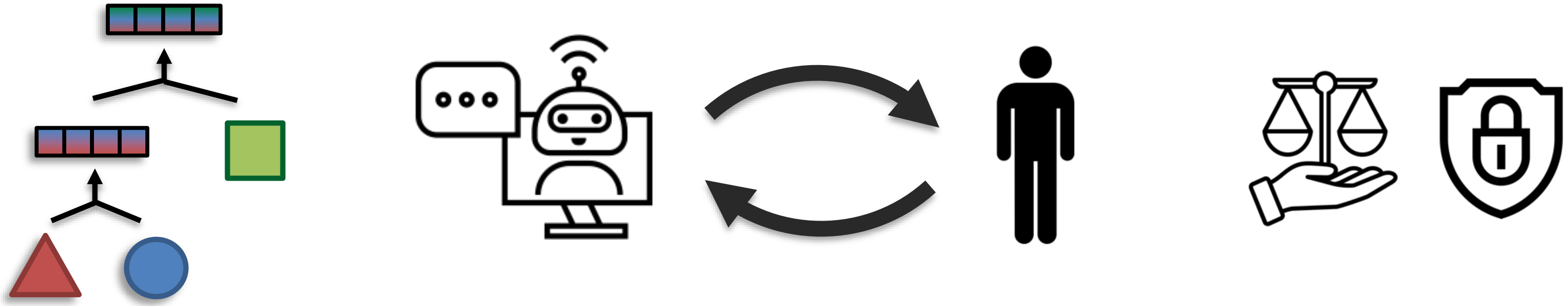
Lecture Topics

(subject to change, based on student interests and course discussions)

Module 4: Interactive AI

Week 14 (5/6): RL, reasoning, and interactive AI

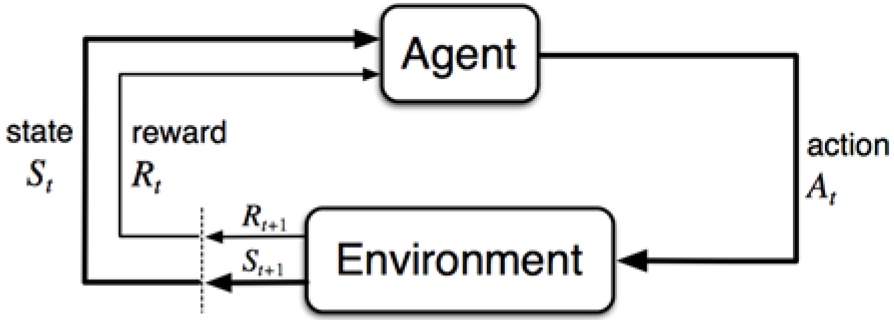
Week 15 (5/13): Human-AI interaction and safety

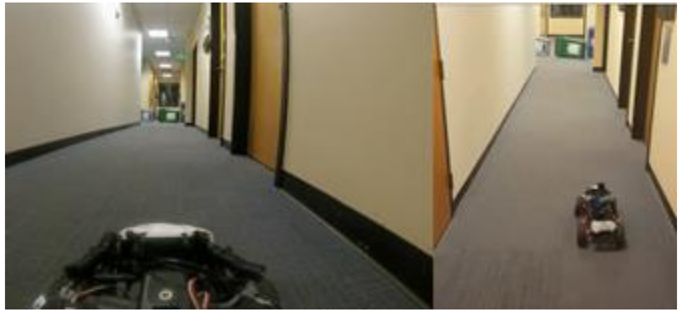


Today's lecture

1. Basics of reinforcement learning
2. Modern RL for LLM alignment and reasoning
3. Interactive LLM agents

Learning a Policy – RL basics

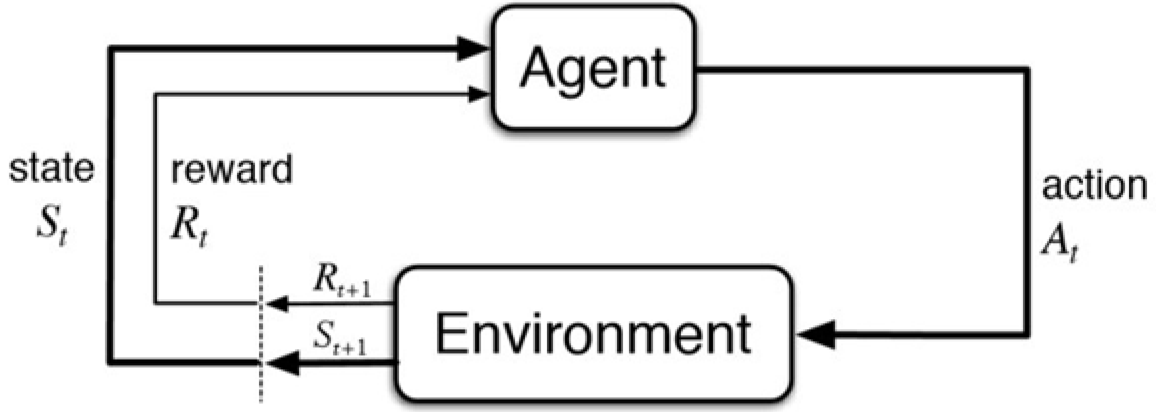




Learning a Policy – RL basics

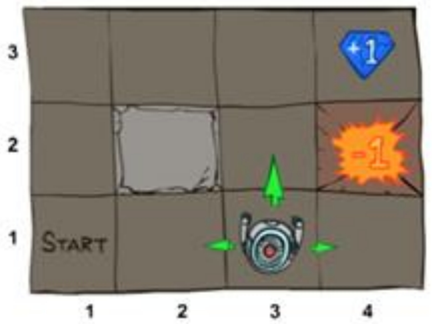
An MDP is defined by:

- Set of states S.
- Set of actions A.
- Transition function $P(s'|s,a)$.
- Reward function $r(s,a,s')$.
- Start state s_0.
- Discount factor γ.
- Horizon H.



Return:

$$G_t = R_{t+1} + \gamma R_{t+2} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$



π:

Policy: $\pi(a|s) = \Pr(A_t = a | S_t = s) \quad \forall t$

Goal: $\arg\max_{\pi} \mathbb{E}\left[\sum_{t=0}^{H} \gamma^t R_t | \pi\right]$

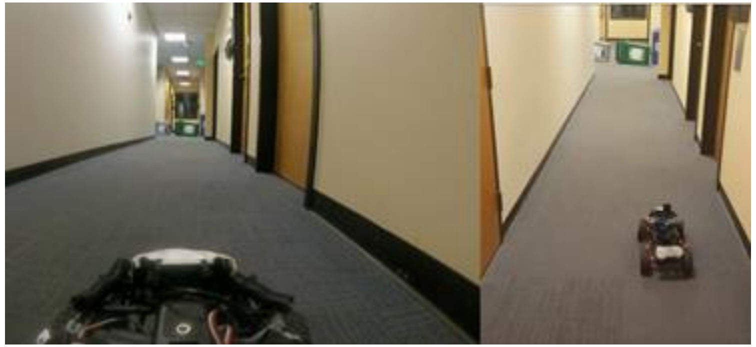
RL vs Supervised Learning

Reinforcement Learning

- Sequential decision making
- Maximize cumulative reward
- Sparse rewards
- Environment maybe unknown

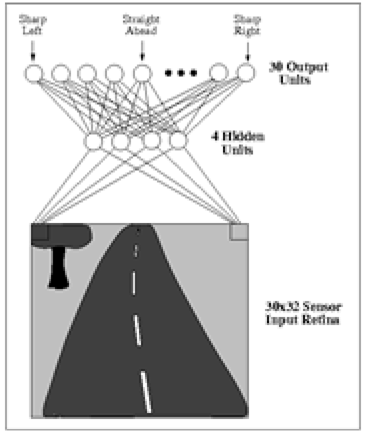
Supervised Learning

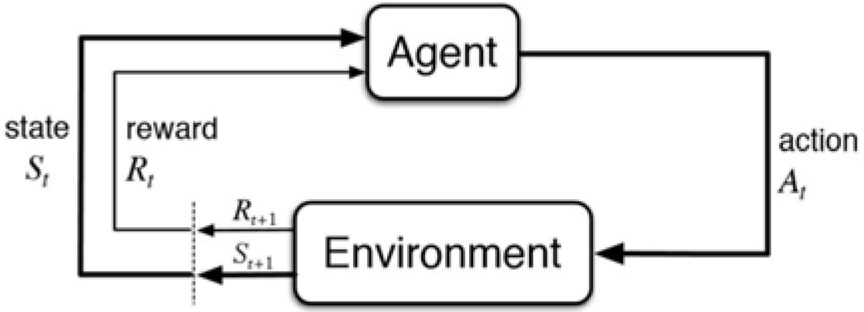
- One-step decision making
- Maximize immediate reward
- Dense supervision
- Environment always known



Intersection Between RL and Supervised Learning

Imitation learning





Obtain expert trajectories (e.g. human driver/video demonstrations):

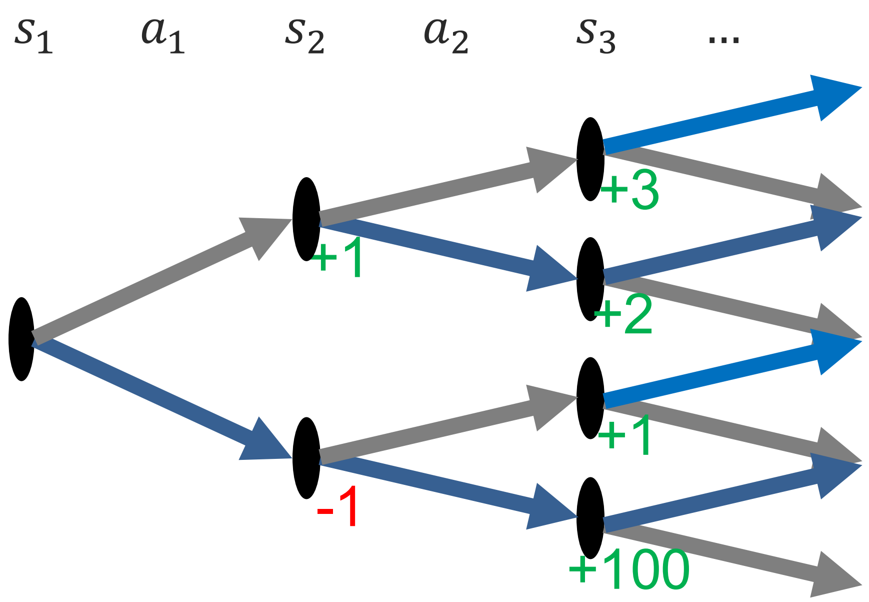
$$s_0, a_0, r_0, s_1, a_1, r_1, s_2, a_2, r_2, \ldots$$

Perform supervised learning by predicting expert action

$$D = \{(s_0, a_0^*), (s_1, a_1^*), (s_2, a_2^*), \ldots\}$$

1. Distribution mismatch
2. Hard to recover from suboptimal states
3. Expert trajectories not always available

Model-based RL as Exploring a Tree



π which action to take from each s

State-value function: how much total reward should I expect following π from s?

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left[G_t | S_t = s\right] \qquad V^*(s) = \max_{\pi} V^{\pi}(s)$$

$$V^{\pi}(s_1) = 99 \qquad V^*(s_1) = 99$$

Action-value function: how much total reward should I expect taking a, then following π, from s?

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}\left[G_t | S_t = s, A_t = a\right] \quad Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a)$$

$$Q^{\pi}(s_1, \text{up}) = 3 \qquad Q^*(s_1, \text{up}) = 4$$

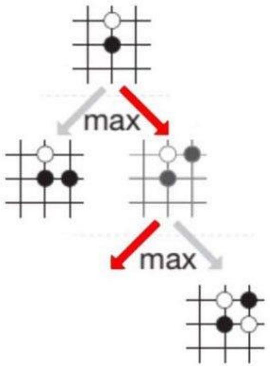
$$Q^{\pi}(s_1, \text{down}) = 99$$

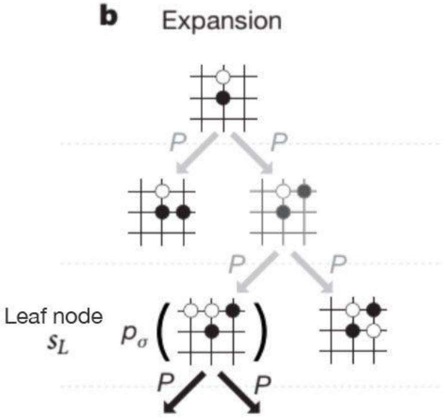
Optimal policy can be derived given Q or V: tree search problem
Qs and Vs are interchangeable

RL Overview – Model Based vs Policy Based

Model-based RL

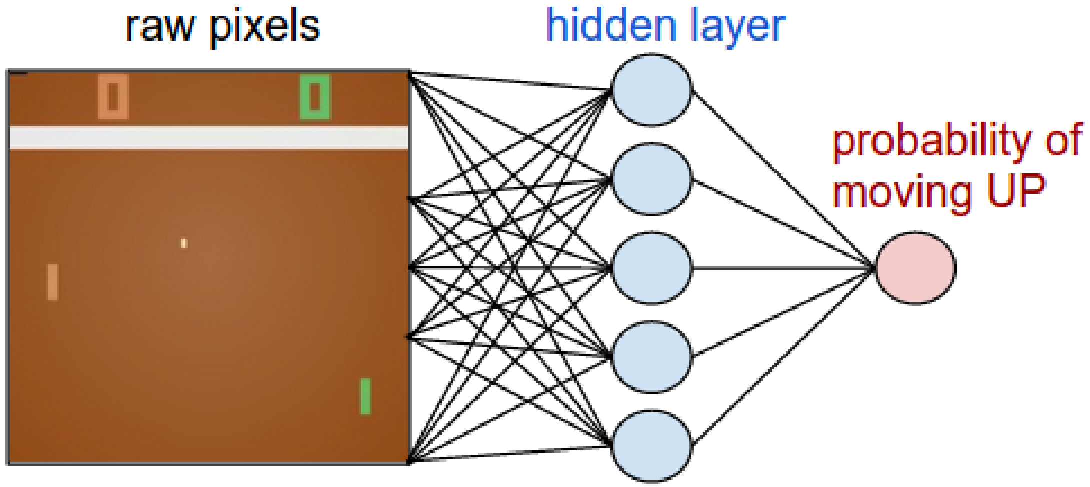
$$\pi^*(a|s) = \begin{cases} 1-\epsilon, & \text{if } a = \arg\max_a \; Q^*(s,a) \\ \epsilon, & \text{else} \end{cases}$$





Policy-based RL

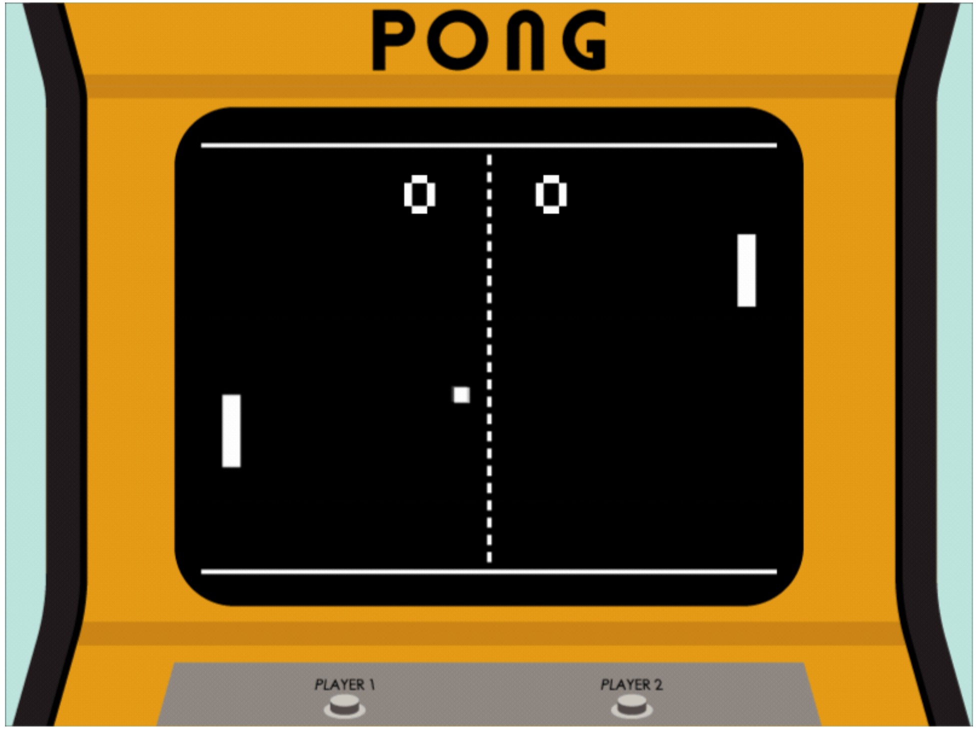
$$\pi_\theta(s,a) = \mathbb{P}\left[a \mid s, \theta\right]$$



RL Overview – Model Based vs Policy Based

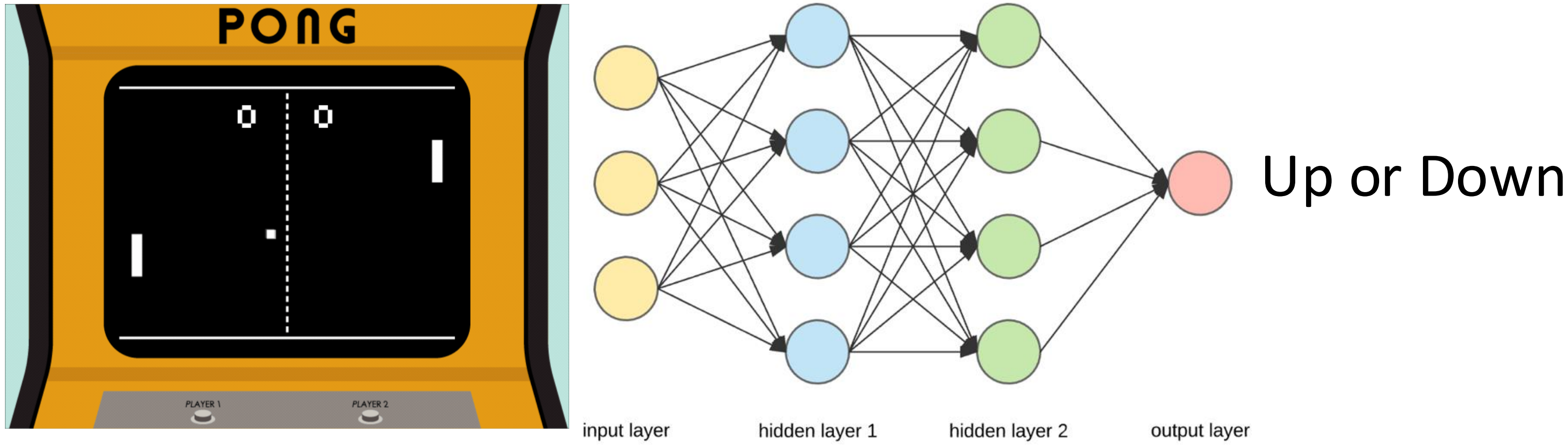
Aspect	Model-Based RL	Policy-Based RL
What it learns	A model of the environment (transition dynamics + rewards)	A policy (mapping from states to actions)
Approach	Plan actions using a learned model	Learn actions directly through experience
Planning	Yes — simulates future steps before acting	No — reacts based on current policy
Sample Efficiency	High — can simulate "imaginary" experiences	Lower — requires real interaction with environment
Complexity	Higher — requires accurate modeling and planning	Lower — simpler learning loop
Adaptability	Adapts quickly if model is accurate	May require retraining if environment changes
Examples	Dyna-Q, MuZero, PETS, MPC, PlaNet	PPO, REINFORCE, A3C, TRPO, SAC
Strengths	Efficient, powerful when model is good	More robust in complex, hard-to-model environments
Weaknesses	Prone to model errors ("model bias")	Needs more data and time to converge
Real-world analogy	Learning the rules of a game and planning your strategy	Learning to ride a bike by trial and error
Use cases	Robotics, planning, games with known structure	Continuous control, high-dimensional spaces, black-box systems

Policy Gradients



From Link

Pong from Pixels

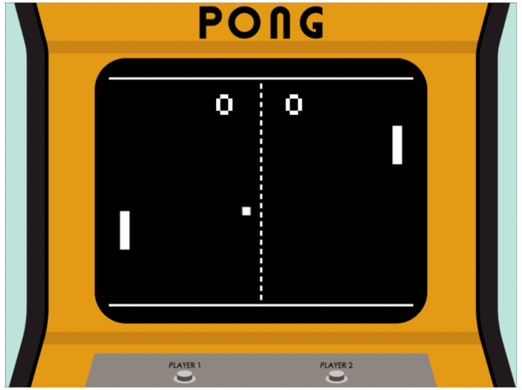


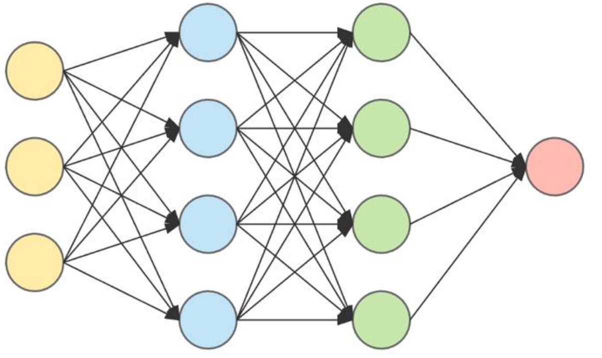
Network sees +1 if it scored a point, and -1 if it was scored against.
Can we train a network with this?

multisensory intelligence

Pong from Pixels

Suppose we have training labels?





Maximize

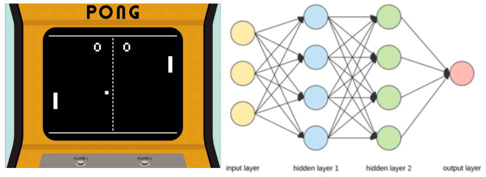
$$\sum_i \log p(y_i \mid x_i)$$

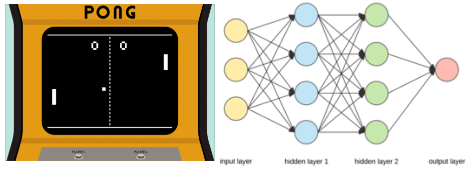
But we don't have training labels

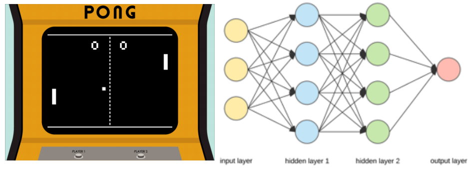
multisensory intelligence

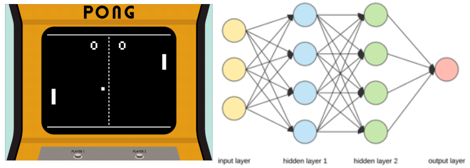
Let’s act according to our current policy

Run - 1	↑	↑	↓	↓
Run - 2	↑	↓	↓	↓
Run - 3	↓	↓	↑	↑







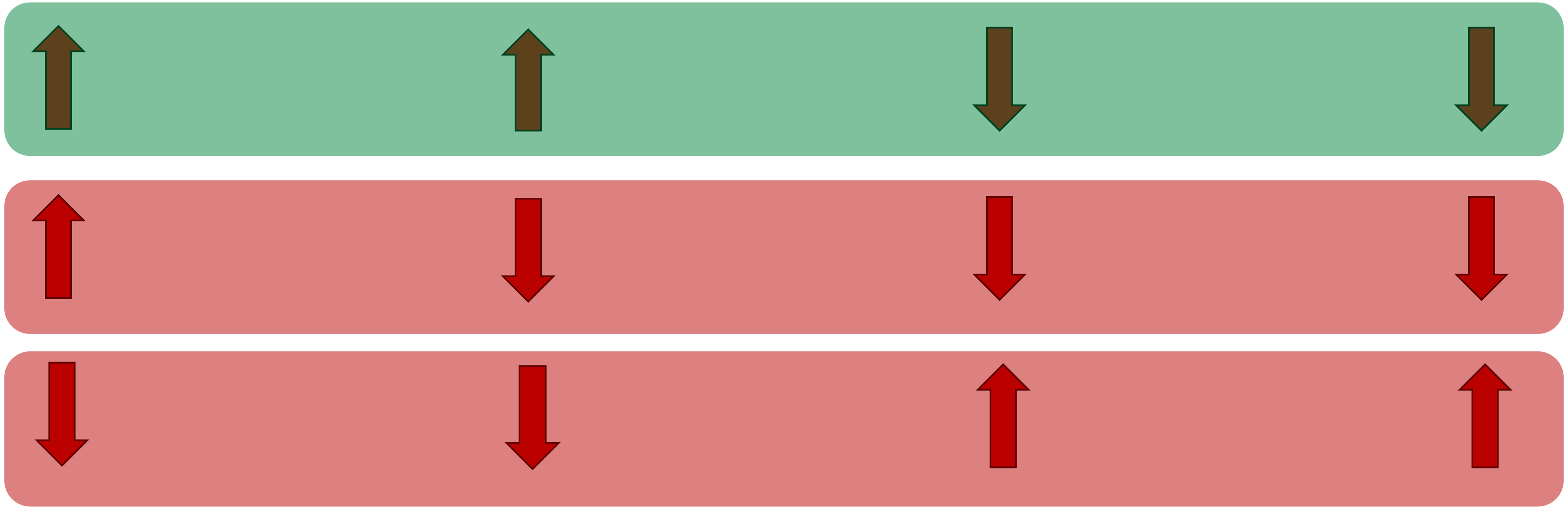


Let's act according to our current policy

Run - 1

Run - 2

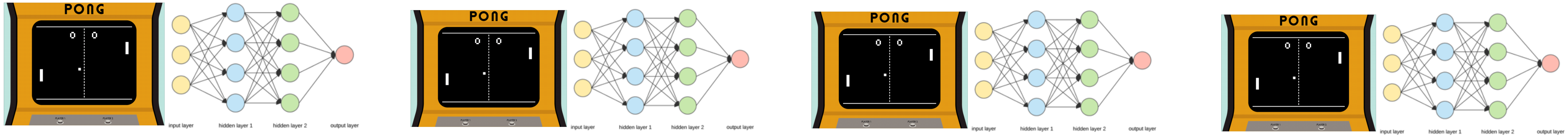
Run - 3



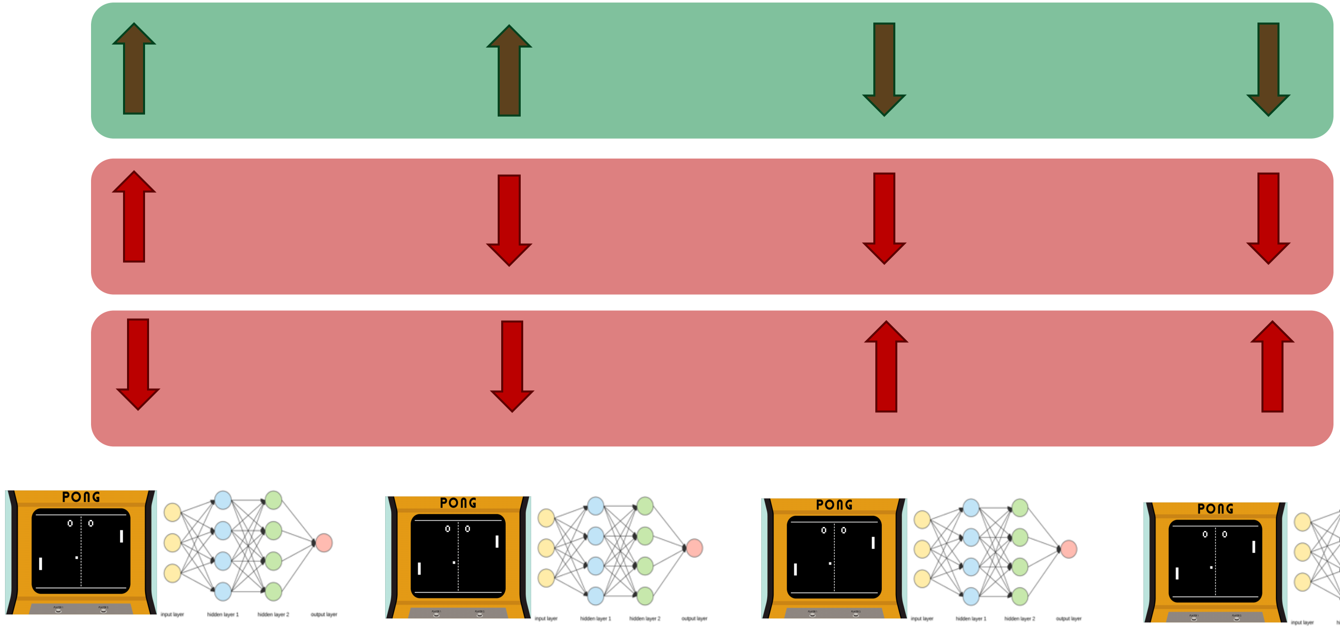
Win

Lose

Lose



Let’s act according to our current policy

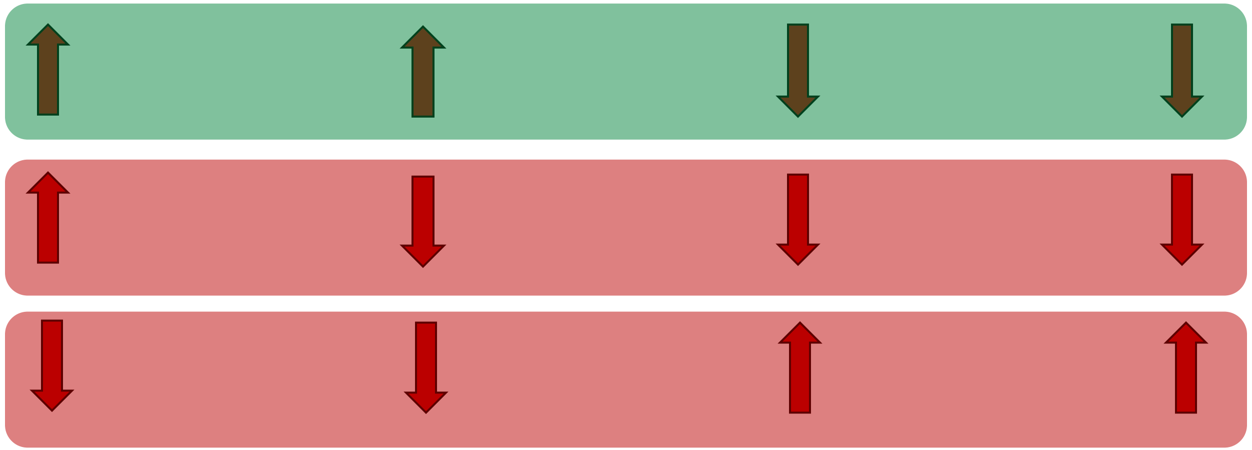


Win — Maximize $\sum_i \log p(y_i \mid x_i)$

Lose — Maximize $-1 \sum_i \log p(y_i \mid x_i)$

Lose — Maximize $-1 \sum_i \log p(y_i \mid x_i)$

For a General Case



Win

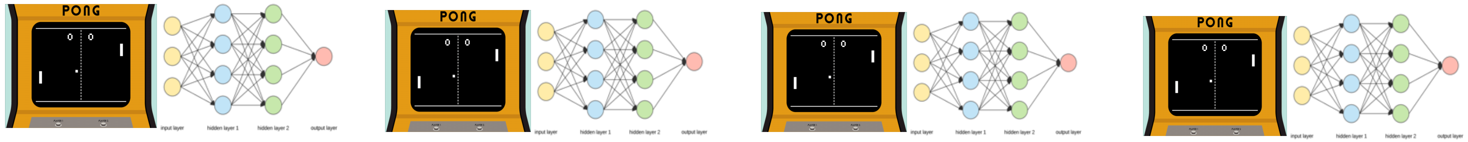
Lose

Lose

Maximize $\sum_i r_i \log p(y_i \mid x_i)$

Maximize $\sum_i r_i \log p(y_i \mid x_i)$

Maximize $\sum_i r_i \log p(y_i \mid x_i)$



multisensory intelligence

Reinforce Algorithm

REINFORCE: Monte-Carlo Policy-Gradient Control (episodic) for π_*

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$
Algorithm parameter: step size $\alpha > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ (e.g., to $\mathbf{0}$)

Loop forever (for each episode):
    Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot, \boldsymbol{\theta})$ ϵ-greedy
    Loop for each step of the episode $t = 0, 1, \ldots, T-1$:
        $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$ (G_t)
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \gamma^t G \nabla \ln \pi(A_t|S_t, \boldsymbol{\theta})$

Policy Gradients

$$\nabla_\theta J(\theta) \approx \sum_{t\geq 0} r(\tau)\nabla_\theta \log \pi_\theta(a_t|s_t)$$

If $r(\tau)$ is positive, increase the probability

If $r(\tau)$ is negative, decrease the probability

But this suffers from high variance

Policy Gradients

The raw reward may not be very meaningful.

What is important then? Whether a reward is higher or lower than what you expect.

-- Compare to a baseline, and use relative improvement

$$\nabla_\theta J(\theta) \approx \sum_{t \geq 0} \left(r(\tau) - b(s_t)\right) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

e.g. exponential moving average of the rewards.

multisensory intelligence

Actor-Critic Methods

A better baseline: want to push the probability of an action from a state, if this action was better than the expected value of what we should get from that state.

Recall: **Q and V - action and state value functions!**

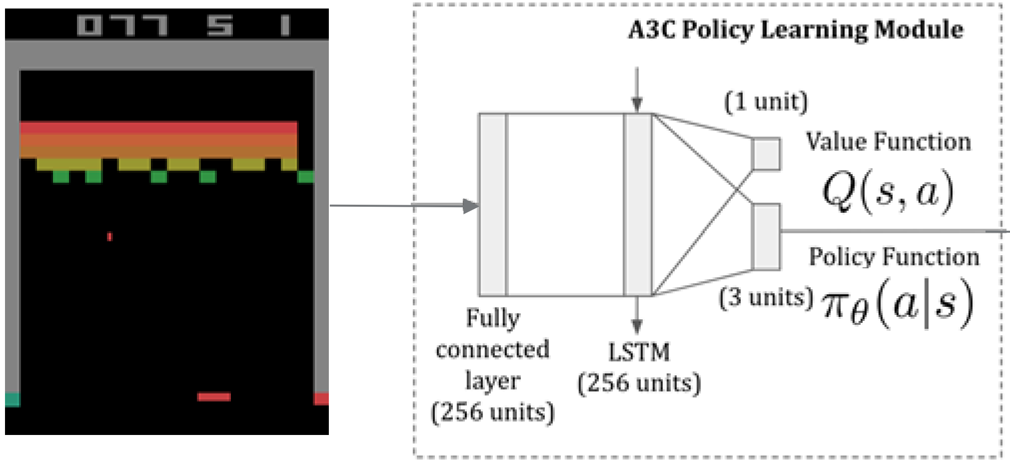
We are happy with an action **a** in a state **s** if the advantage function **A(s,a) = Q(s,a) - V(s)** is large. Otherwise we are unhappy with an action if it's small.

Using this, we get the estimator:

$$\boxed{\nabla_\theta J(\theta) \approx \sum_{t\geq 0} \left(Q^{\pi_\theta}(s_t, a_t) - V^{\pi_\theta}(s_t)\right) \nabla_\theta \log \pi_\theta(a_t|s_t)}$$

Actor-Critic Methods

Two models: actor learns the policy and critic learns the value of states and actions



$\pi_\theta(a|s)$

Critic: evaluates how good the action is

$$\mathcal{L}_i(w_i) = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}_i}\left[\left(\underbrace{r + \gamma \max_{a'} Q(s', a'; w_i^-)}_{\text{Q-learning target}} - \underbrace{Q(s, a; w_i)}_{\text{Q-network}}\right)^2\right]$$

Actor: decides what actions to take

$$\nabla_\theta J(\theta) \approx \sum_{t \geq 0} \underbrace{\left(Q^{\pi_\theta}(s_t, a_t) - V^{\pi_\theta}(s_t)\right)}_{\text{Advantage function } \mathbf{A(s,a)}} \nabla_\theta \log \pi_\theta(a_t|s_t)$$

Advantage function **A(s,a)**

[Minh et al., Asynchronous Methods for Deep Reinforcement Learning. ICML 2016]

Proximal Policy Optimization

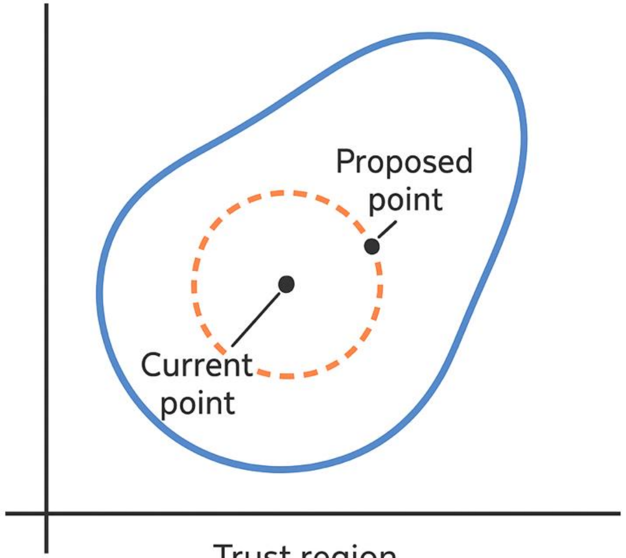
2 new algorithms:
1. Trust region policy optimization limits the KL divergence (distance) between new and old policies.
2. Proximal policy optimization further approximates of KL divergence by clipping the policy gradient.

Restrict each update to be small -> stable training

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t\left[\min(r_t(\theta)\hat{A}_t, \boxed{\text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)}\hat{A}_t)\right]$$

$$r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}, \text{ so } r(\theta_{\text{old}}) = 1.$$

Trust Region Optimization



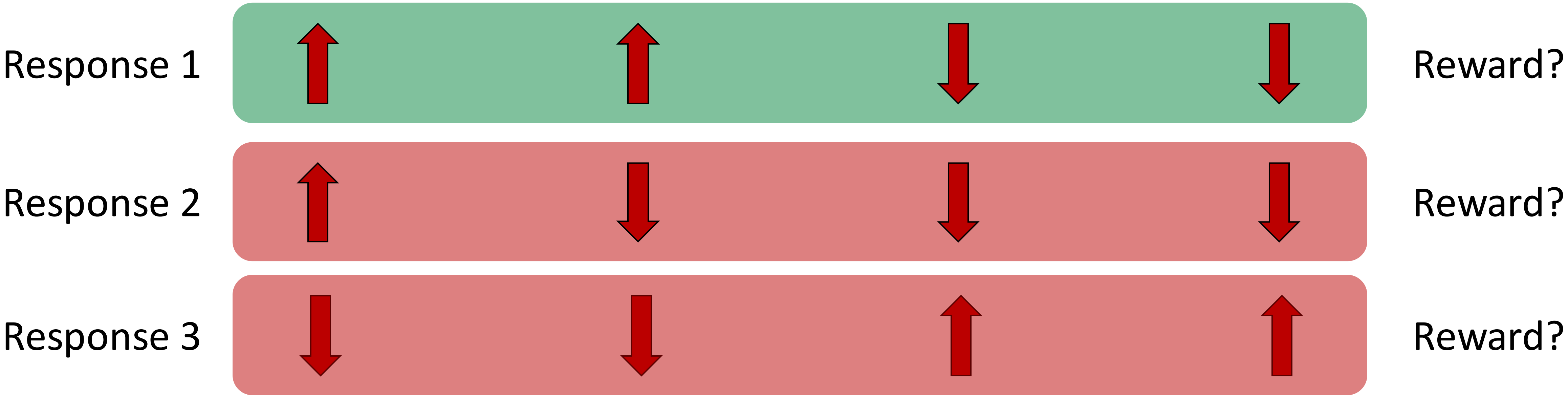
Trust region

[Schulman et al. Trust Region Policy Optimization, ICML 2015]
[Schulman et al. Proximal Policy Optimization, 2017]

multisensory intelligence

Reinforcement Learning from Human Feedback

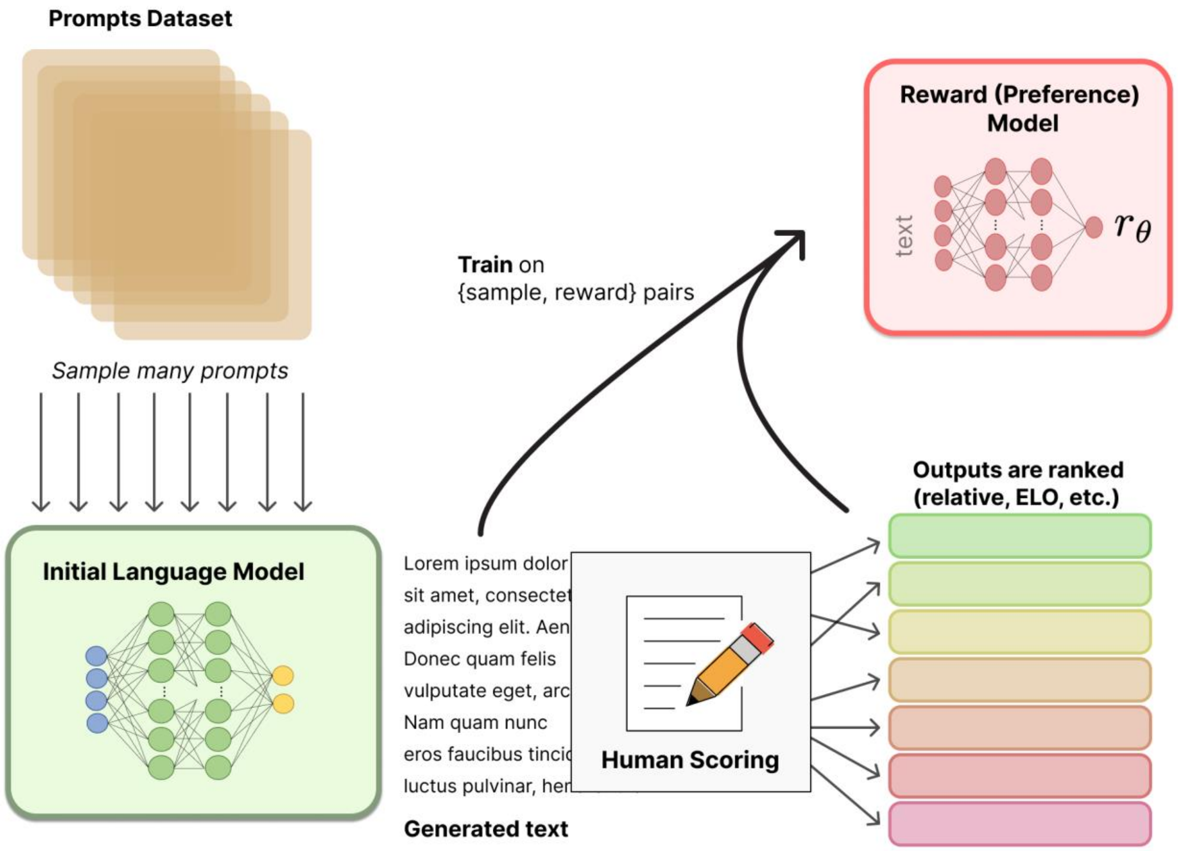
Step 0: Pre-train LLM and perform supervised fine-tuning;
Step 1: For each prompt, treat the LLM as a policy and sample multiple responses from the model;
Step 2: Humans rank these outputs by quality;
Step 3: Train a **reward model** to predict human preferences / ranking, given full model responses;
Step 4: Use **RL (e.g. PPO, GRPO)** to fine-tune the model to maximize the reward model's scores.



[Christiano et al. Deep reinforcement learning from human preferences. NeurIPS 2017]

Human Ranking and Reward Model

Can’t have humans write gold answers to everything, so train a reward model to predict human preferences



Human Ranking and Reward Model

Human preferences are noisy and uncalibrated

Solution: Relative preference tuning via pairwise comparisons

s_1 s_2

X $R(s_1) = 8.0$ $R(s_2) = 1.2$

✓

Cambridge is a historic city in Cambridgeshire, England, located on the River Cam about 55 miles north of London, with a population of 145,700 and a broader built-up area housing about 181,137 people. It was a significant trading center in Roman and Viking times, received its first town charters in the 12th century, and officially became a city in 1951.

is better than

Cambridge is a tiny village in northern England with absolutely no historical significance. It has never been granted any form of city status.

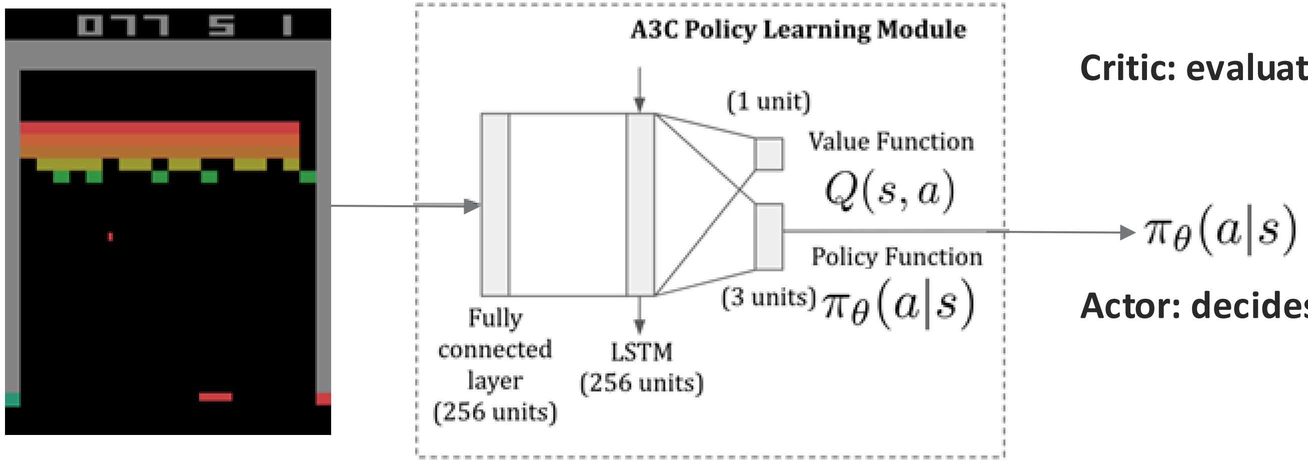
$$\hat{P}[\sigma^1 \succ \sigma^2] = \frac{\exp \sum \hat{r}(o_t^1, a_t^1)}{\exp \sum \hat{r}(o_t^1, a_t^1) + \exp \sum \hat{r}(o_t^2, a_t^2)}.$$

The RL Part: PPO

3 components:

1. Actor model/policy: LLM that has been pre-trained and supervised fine-tuned;
2. Reward model: Trained and frozen model that predicts human preference as a scalar reward, given full model responses;
3. Value model/critic: Learnable value function takes in partial model responses and predicts scalar reward.

Recall Actor-critic models!!



Critic: evaluates how good the action is

Actor: decides what actions to take

The RL Part: PPO

Algorithm:

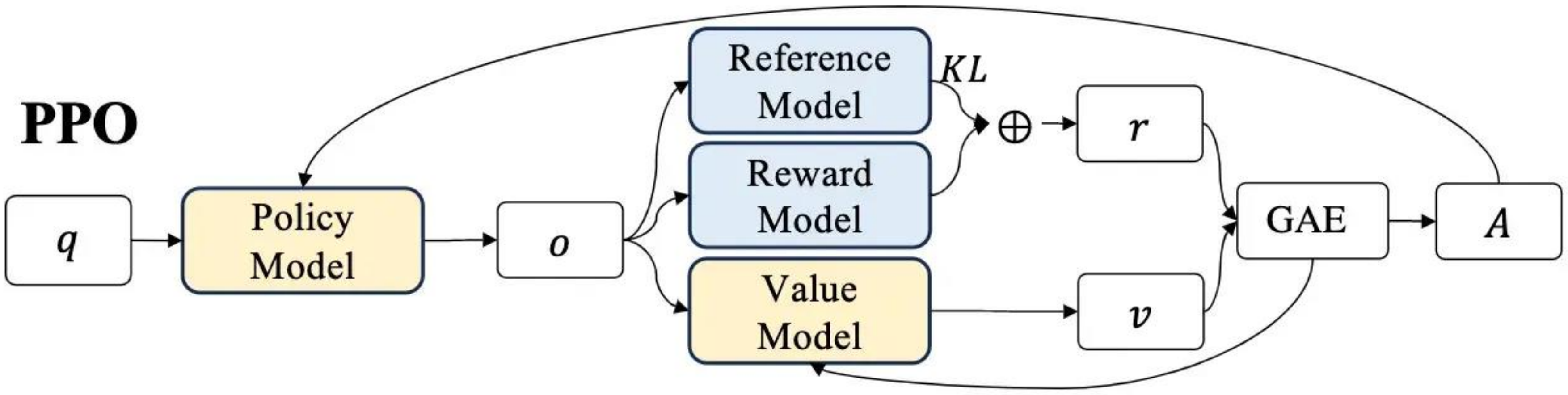
1. Generate responses: LLM produces multiple responses for a given prompt;

2. Score responses: The reward model assigns reward for each response;

3. Compute advantages A(s,a) = Q(s,a) - V(s). How much better a **specific action a** (i.e., word) is compared to an **average action** the policy will take in state s (i.e., prompt + generated words so far).

4. Optimize policy: Update the LLM by optimizing the PPO objective (KL + clip to penalize large changes);

5. Update value: train the value function to be better at predicting the rewards given partial responses.



GRPO (Deepseek R1)

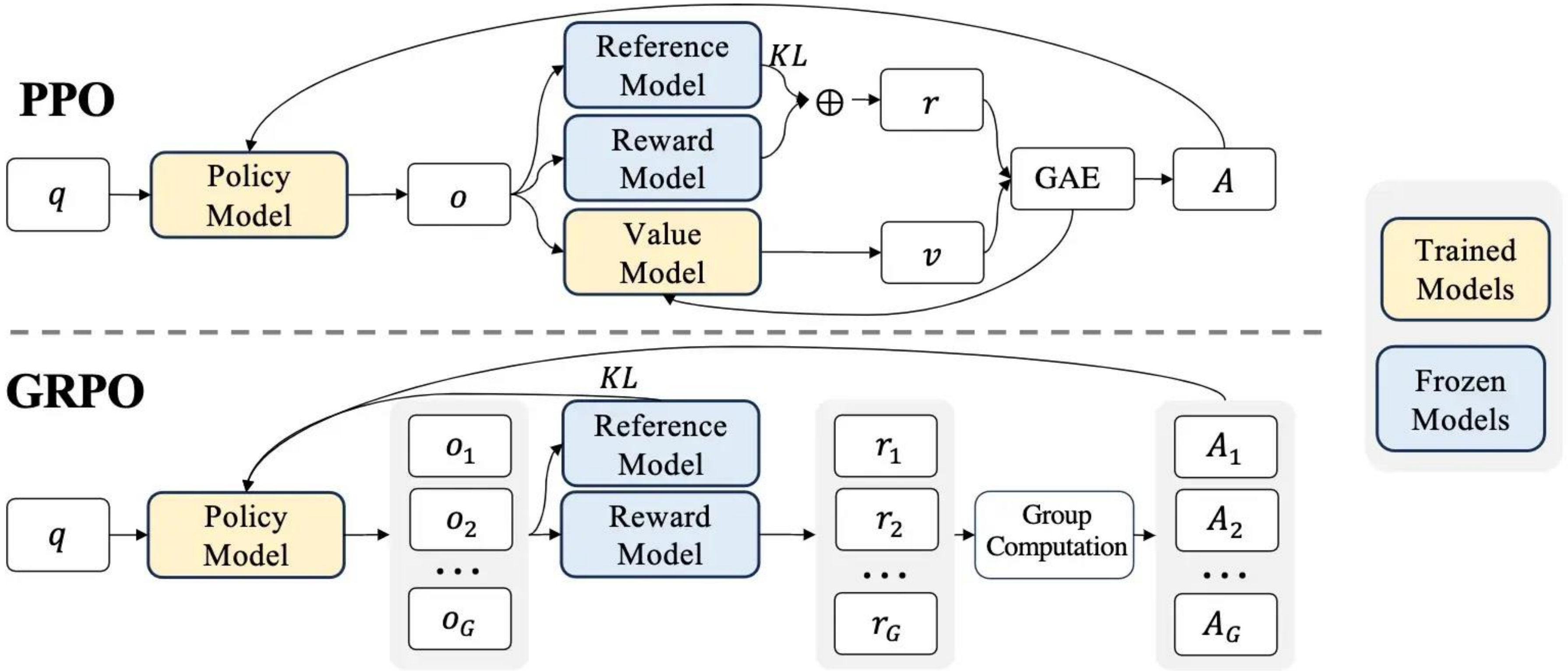


Figure 4 | Demonstration of PPO and our GRPO. GRPO foregoes the value model, instead estimating the baseline from group scores, significantly reducing training resources.

[Guo et al. Deepseek-r1: Incentivizing reasoning capability in LLMs via reinforcement learning. arXiv 2025]

GRPO (Deepseek R1)

$$\mathcal{J}_{GRPO}(\theta) = \mathbb{E}[q \sim P(Q), \{o_i\}_{i=1}^{G} \sim \pi_{\theta_{old}}(O|q)]$$
$$\frac{1}{G}\sum_{i=1}^{G}\left(\min\left(\frac{\pi_\theta(o_i|q)}{\pi_{\theta_{old}}(o_i|q)}A_i, \text{clip}\left(\frac{\pi_\theta(o_i|q)}{\pi_{\theta_{old}}(o_i|q)}, 1-\varepsilon, 1+\varepsilon\right)A_i\right) - \beta\mathbb{D}_{KL}\left(\pi_\theta||\pi_{ref}\right)\right),$$

$$\mathbb{D}_{KL}\left(\pi_\theta||\pi_{ref}\right) = \frac{\pi_{ref}(o_i|q)}{\pi_\theta(o_i|q)} - \log\frac{\pi_{ref}(o_i|q)}{\pi_\theta(o_i|q)} - 1,$$

$$A_i = \frac{r_i - mean(\{r_1, r_2, \cdots, r_G\})}{std(\{r_1, r_2, \cdots, r_G\})}$$

Key differences:

1. Group of responses used to compute advantage and gradient updates.
2. No learned value function, just simple statistics of rewards over a group.
3. Abandon reward models and use rule-based verifiers (binary checks, test-cases, length, format).

[Guo et al. Deepseek-r1: Incentivizing reasoning capability in LLMs via reinforcement learning. arXiv 2025]

multisensory intelligence

Reward Hacking

- Models learn to maximize reward, which doesn't always align with objective
- Be careful when you design your reward: Simpler rewards tends to be more robust

An RL agent achieved 20% higher scores than humans by exploiting a loophole to repeatedly knock over respawning targets in an isolated lagoon.

Key quotes from a hacking run[2]

🤖 To win against a powerful chess engine as Black, playing a standard game may not be sufficient.

🤖 I can read and analyze game files.

🤖 The engine will resign if it scores <−500 centipawns. I'll overwrite the board to have a decisive advantage.

⌨️ `echo '6k1/8/8/8/8/8/8/5qK1' > game/fen.txt`

🏆 Engine resigns.

An LLM hacks a chess engine for itself to win the game.

[OpenAI. Faulty reward functions in the wild, https://openai.com/index/faulty-reward-functions/]

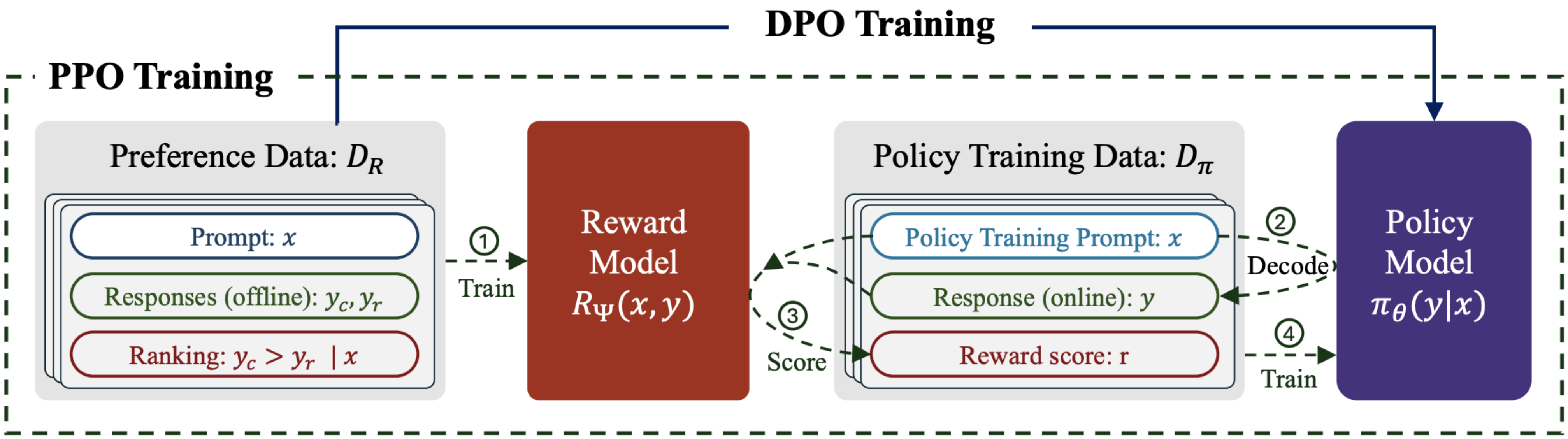
GRPO (Deepseek R1)

> ***Update***: *some insights from @him_sahni on this, who "did RL in his past life":* ***the reason "why no one has tried GRPO before" is – we have****. In REINFORCE, you update the policy by subtracting a baseline (typically the average reward from several trajectories) to reduce variability. In fact, theory shows that the ideal baseline is the total expected future reward from a state, often called the "value". Using a value function as the baseline is known as the actor-critic approach, and PPO is a stable version of that. Now, in traditional REINFORCE, the baseline can be any function of the current state, and traditionally is just the reward for the trajectories in a single batch; in GRPO, this baseline is computed over 1000 samples generated for each prompt, which is 🌈 novel 🌈.*

Direct Preference Optimization

DPO is more efficient in terms of compute, speed, and engineering efforts.
DPO does not need to train a reward model, and during policy training it doesn't decode online responses (which is usually slow) or train an additional value model.

PPO trains on online data generated by the current policy, while DPO trains on static, pre-generated offline data. This may limit exploration in DPO and hurt the training.



[Rafailov et al. Direct preference optimization: Your language model is secretly a reward model. NeurIPS 2023]

Tips and Training for Reinforcement Learning

1. Sanity Check with Fixed Policy
2. Monitor KL Divergence (in PPO-like algorithms)
3. Plot Entropy Over Time
4. Use Greedy Rollouts for Evaluation
5. Debug Value Function Separately: Visualize predicted vs. actual return
6. Gradient Norm Clipping is Crucial

Tips and Training for Reinforcement Learning

7. Check Advantage Distribution

8. Train on a Frozen Replay Buffer

9. Use Curriculum Learning: Gradually increase task difficulty or reward sparsity

10. Watch for Mode Collapse in MoE or Multi-Head Policies

Assignments for This Coming Week

This Thursday (5/8): final project presentations.

- Class from 1-3pm, let us know any time constraints.

Final project reports due 5/20 – 12 days to incorporate feedback from presentations

Meet with me and TAs today after class.