RL Foundations for Agents

From SFT to policy gradients

Carnegie Mellon University
Language Technologies Institute

Daniel Fried

11-768: AI Agents

Running example: Number Search

A hidden integer lies between 1 and 16. The agent has four guesses. After an incorrect guess, the environment replies **higher** or **lower**. Reward is 1 only when it finds the number.

Hidden number for this task instance: 11

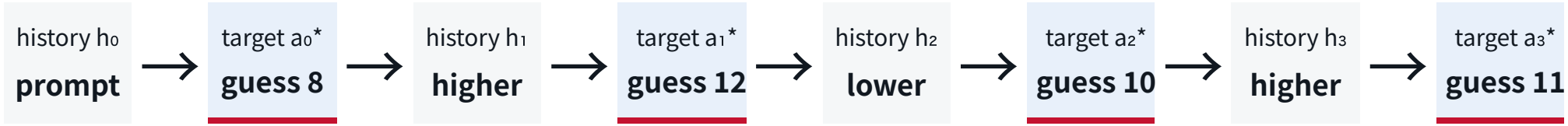
Four sample trajectories:

τ1	8	higher	12	lower	10	higher	11	correct	R = 1
τ2	4	higher	8	higher	12	lower	11	correct	R = 1
τ3	16	lower	8	higher	9	higher	10	timeout	R = 0
τ4	1	higher	2	higher	3	higher	4	timeout	R = 0

τ_1 and τ_2 different actions · same successful outcome

SFT on demonstrations

SFT trains the agent to maximize the probability of all target actions in a demonstration



$$\mathcal{L}_{\mathrm{SFT}} = -\sum_{t} \log \pi_\theta(a_t^* \mid h_{1:t})$$

Target actions may come from a person, a stronger model, or (today) a successful trajectory from this agent

Task mismatch

SFT maximizes the probability of demonstrated actions. We would like to maximize the probability that the agent carries out the task. (these are not quite the same!)

DEMONSTRATION τ1	8	higher	12	lower	10	higher	11	correct
ANOTHER SUCCESS τ2	4	higher	8	higher	12	lower	11	correct

- There are multiple ways to carry out a task.
- Some actions are worse than others (e.g. should never guess "13" after "12 -> lower").

Data mismatch

We'd like to be able to learn from sub-optimal data, too

DEMONSTRATION τ1	8	higher	12	lower	10	higher	11	correct
OTHER ALTERNATIVES τ2	4	higher	8	higher	12	lower	11	correct
τ3	16	lower	8	higher	9	higher	10	timeout
τ4	1	higher	2	higher	3	higher	4	timeout

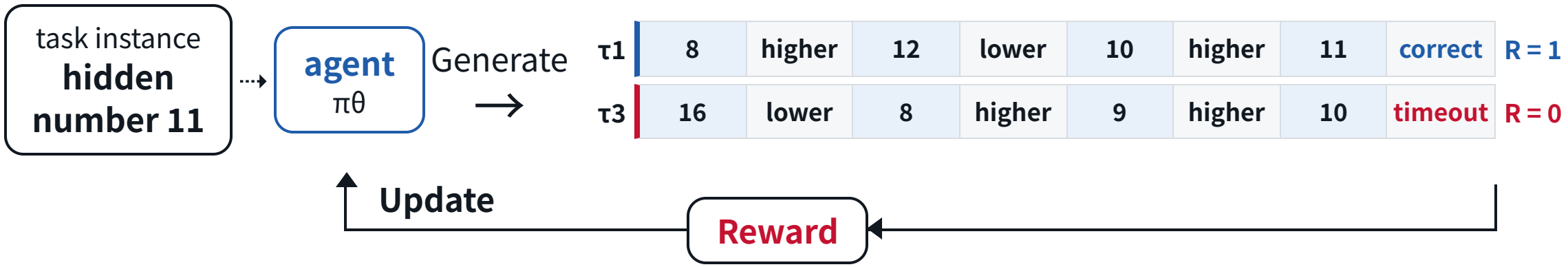
Exposure bias

The agent was not exposed to its own mistakes at training time, and might not be prepared to handle mistakes when generating

TRAINING DEMONSTRATION	8	higher	12	lower	10	higher	11	correct
GENERATION	8	higher	2	*?*				

Reinforcement learning

Generate actions/trajectories from the agent, evaluate their **reward**, and adjust the parameters of the agent to maximize expected reward.

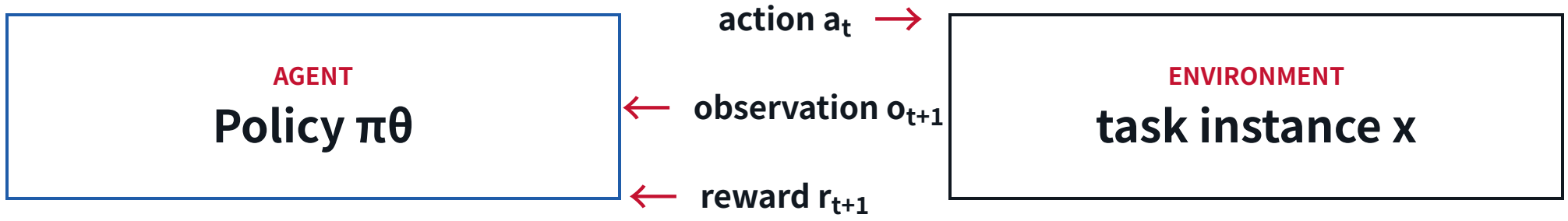


Task mismatch	**We train the agent to maximize expected reward.**
Data mismatch	**The agent can learn from all trajectories (seek high reward; avoid low reward)**
Exposure bias	**The agent generates trajectories in training (on-policy), so it learns to recover from errors**

Adapted from Sean Welleck・RL Fundamentals ↗

Interactions as trajectories

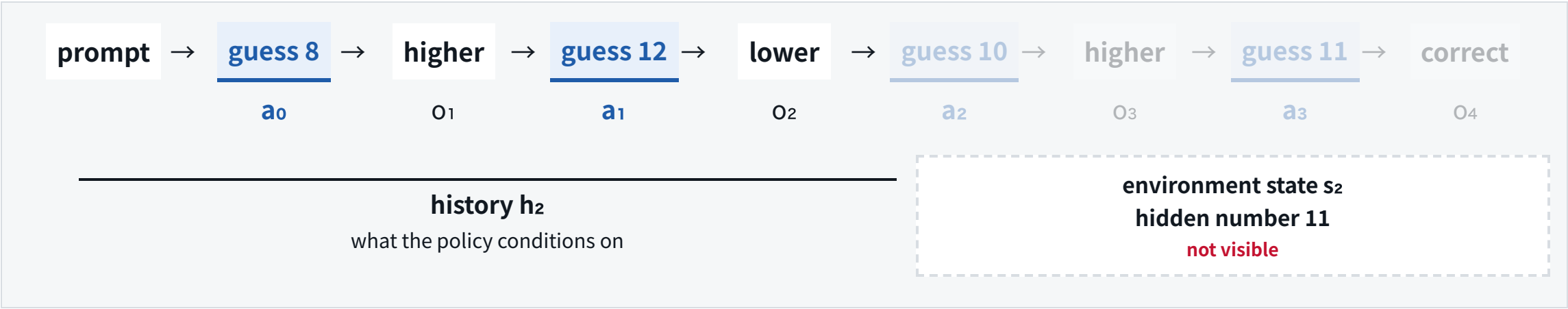
The agent–environment loop



	Task instance x	Observation o_{t+1}	Action a_t	Reward r_{t+1}
Text generation	the prompt	tokens so far	next token	score from a judge model
Web agent	the task and website	rendered page	click, type, scroll	number of sub-tasks completed
Number Search	number (hidden)	higher, lower, or correct	the next guess	1 on correct; otherwise 0

Trajectories

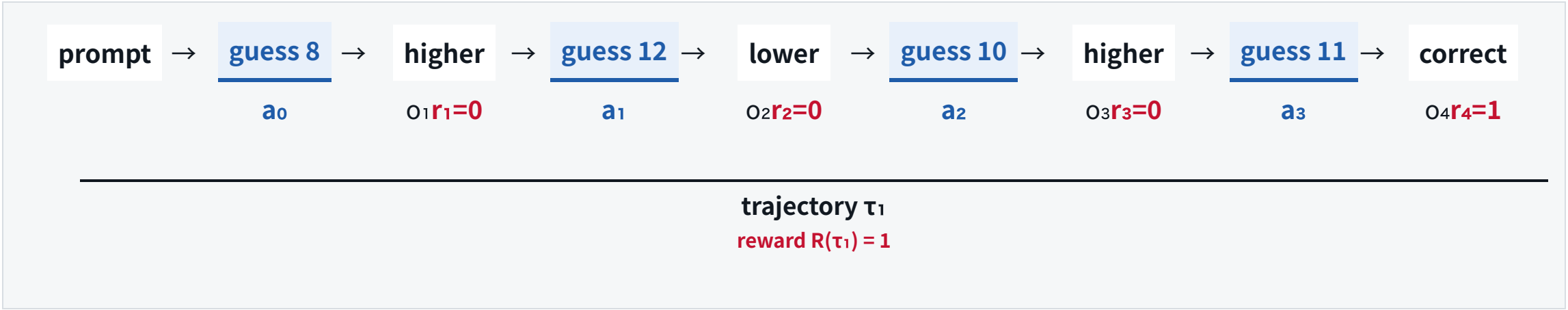
The agent conditions only on past actions and observations (not hidden environment state).



$$h_t = (o_0, a_0, o_1, \ldots, o_t) \qquad \pi_\theta(a_t \mid h_t)$$

Trajectories

Trajectories can also include reward at each step.

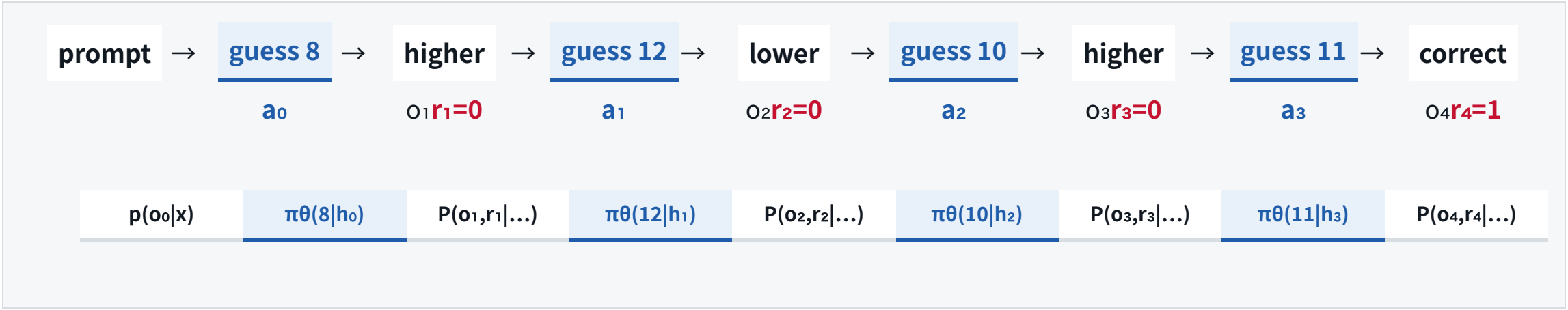


$$\tau = (o_0, a_0, r_1, o_1, \ldots, a_{T-1}, r_T, o_T) \qquad R(\tau) = \sum_{t=1}^{T} r_t$$

In number search, 1 for "correct" and 0 otherwise.

Trajectories

Trajectory probabilities depend on the policy **and** the environment.

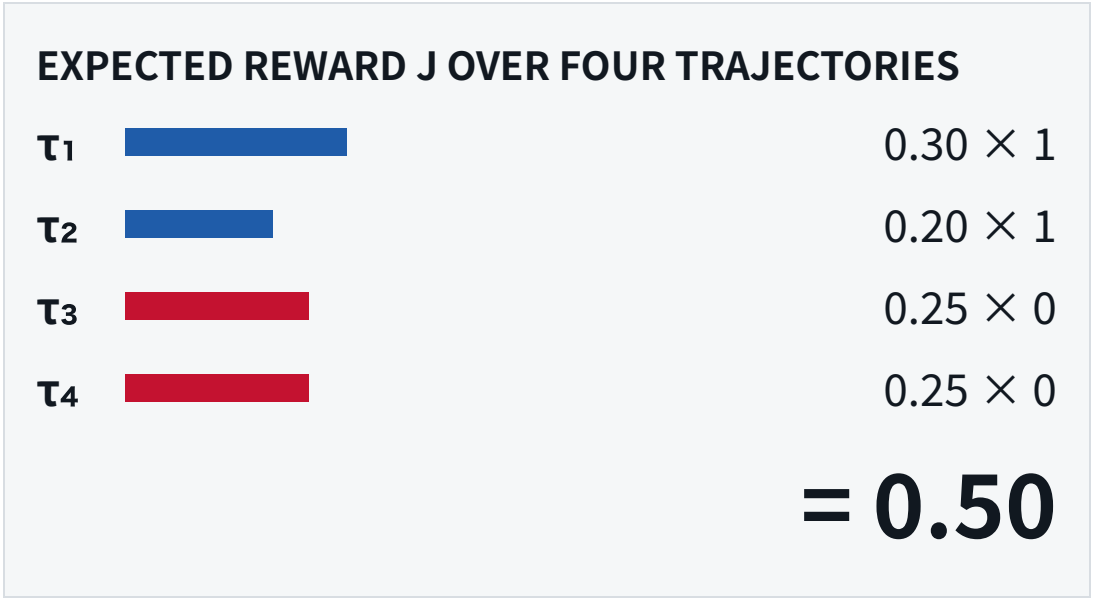


$$p_\theta(\tau \mid x) = \prod_t \pi_\theta(a_t \mid h_t)\,(o_{t+1}, r_{t+1} \mid h_t, a_t, x)$$

We don't know P, but we can sample from it by interacting with the environment.

Objective: maximize expected reward

$$J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\cdot|x)}[R(\tau)]$$



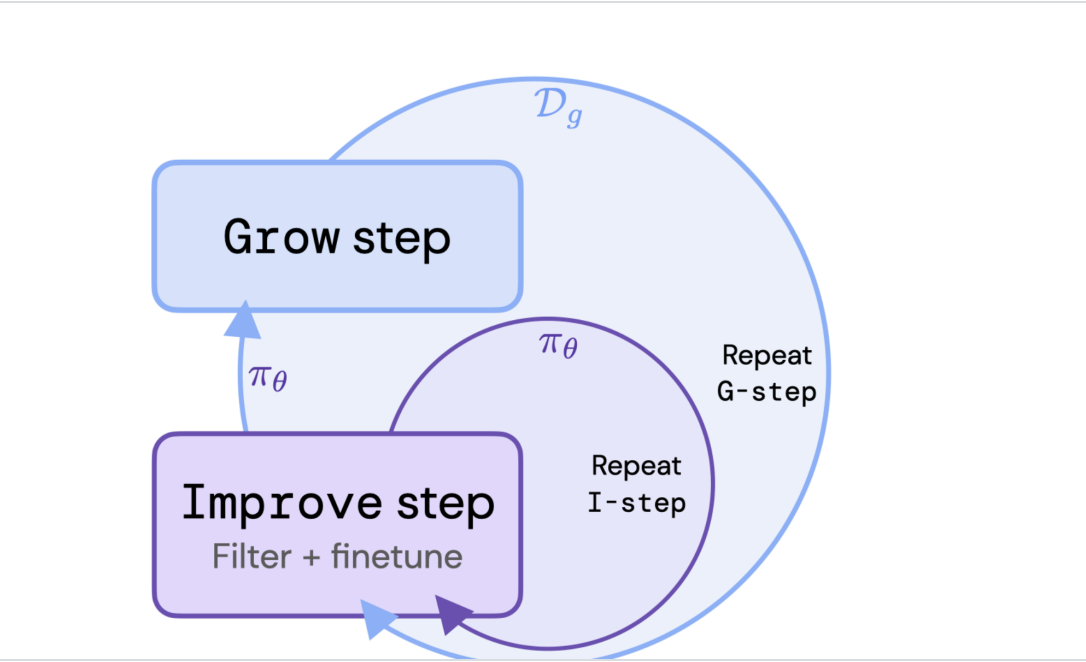
Training adjusts θ to maximize expected reward (i.e. the agent has a higher probability of high reward trajectories)

low reward trajectories — probability mass → — **high reward trajectories**

Expert iteration

Reinforced self-training

ReST generates data from the policy, and then updates the policy using successful trajectories.



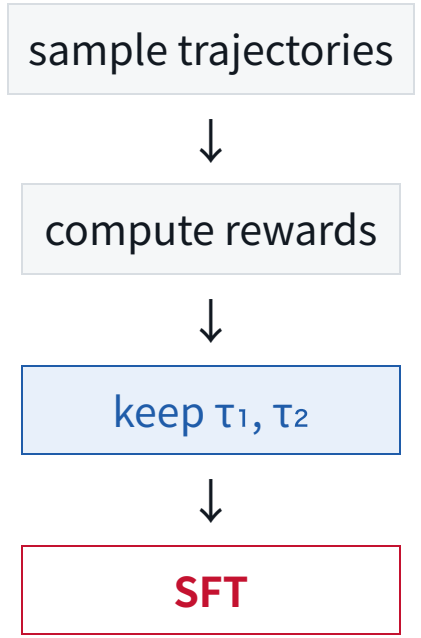
1. **Grow**
   Sample trajectories from the current policy and compute rewards.
2. **Improve**
   Use rewards to filter the data; augment the existing datasets; train the policy.
3. **Repeat**
   Use the updated policy.

Gulcehre et al. · ReST, Fig. 1 ↗ Singh et al. · ReST-EM ↗

Reinforced self-training

The loss is the standard SFT loss, on trajectories with high reward

τ1	8	higher	12	lower	10	higher	11	correct	R = 1 KEEP
τ2	4	higher	8	higher	12	lower	11	correct	R = 1 KEEP
τ3	16	lower	8	higher	9	higher	10	timeout	R = 0 DROP
τ4	1	higher	2	higher	3	higher	4	timeout	R = 0 DROP



$$\mathcal{L}_{\mathrm{ReST}} = -\sum_{\tau:\, R(\tau)=1} \sum_{t} \log \pi_\theta(a_t \mid h_t)$$

Gulcehre et al. · ReST ↗ Singh et al. · ReST-EM ↗

From rewards to gradients

Policy gradients

How to maximize expected reward?

We can't use backpropagation directly: sampling and the environment are non-differentiable

Sampled action
Changing θ changes the action probabilities, but actions are discrete and sampled.

Task environment
The world is not (usually!) differentiable. How do you take the derivative of a unit test w.r.t. code?

$$J(\theta) = \sum_{\tau} p_\theta(\tau \mid x)\ R(\tau)$$

pθ(τ|x) changes with θ

R(τ) is observed after the rollout

General solution: policy gradient (REINFORCE, Williams 1992).

Policy gradient

Maximize expected reward under the policy.

$$J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\cdot|x)}\left[\, R(\tau) \,\right] = \sum_{\tau} p_\theta(\tau \mid x)\, R(\tau)$$

Take the gradient: only the probability depends on θ.

$$\nabla_\theta J = \sum_{\tau} \nabla_\theta p_\theta(\tau \mid x)\, R(\tau)$$

Multiply and divide by trajectory probability.

$$= \sum_{\tau} p_\theta(\tau \mid x)\, R(\tau)\, \frac{\nabla_\theta p_\theta(\tau \mid x)}{p_\theta(\tau \mid x)}$$

Use $\nabla p / p = \nabla \log p$.

$$= \sum_{\tau} p_\theta(\tau \mid x)\, R(\tau)\, \nabla_\theta \log p_\theta(\tau \mid x)$$

This is an expectation we can estimate from samples.

$$= \mathbb{E}_{\tau \sim p_\theta(\cdot|x)}\left[\, R(\tau)\, \nabla_\theta \log p_\theta(\tau \mid x) \,\right]$$

REINFORCE (Williams, 1992) ↗

Policy gradient

$$\nabla_\theta J = \mathbb{E}_{\tau \sim p_\theta(\cdot|x)} \left[R(\tau)\ \nabla_\theta \log p_\theta(\tau \mid x) \right]$$

Replace the expectation by trajectories we sample.

$$\tau \sim p_\theta(\cdot \mid x), \qquad \widehat{\nabla_\theta J} = R(\tau)\, \nabla_\theta \log p_\theta(\tau \mid x) = R(\tau) \sum_t \nabla_\theta \log \pi_\theta(a_t \mid h_t)$$

$\tau \sim p_\theta(\cdot|x)$

Current-policy trajectories and histories

obtain by sampling: policy and environment

$R(\tau)$

Task-level evaluation

obtain from the environment

$\nabla_\theta \log \pi_\theta(a_t|h_t)$

Changes probabilities of sampled actions

same mechanics as in SFT

Williams · REINFORCE, 1992 ↗

Trajectory log probability

$p(o_0|x)$ • $\pi\theta(a_0|h_0)$ • $P(o_1,r_1|h_0,a_0,x)$ • $\pi\theta(a_1|h_1)$ • $P(o_2,r_2|h_1,a_1,x)$

$$\log p_\theta(\tau \mid x) = \sum_t \log P(o_t, r_t \mid h_t, a_t, x) + \sum_t \log \pi_\theta(a_t \mid h_t)$$

$$= \text{const} + \sum_t \log \pi_\theta(a_t \mid h_t)$$

$$\nabla_\theta \log p_\theta(\tau \mid x) = \sum_t \nabla_\theta \log \pi_\theta(a_t \mid h_t)$$

Same computation as SFT: sum token log probabilities. Here the tokens were sampled by the policy.

$$a_t = (u_{t,1}, \ldots, u_{t,m_t}), \qquad \nabla_\theta \log \pi_\theta(a_t \mid h_t) = \sum_{k=1}^{m_t} \nabla_\theta \log \pi_\theta(u_{t,k} \mid h_t, u_{t,<k})$$

t indexes actions; *k* indexes tokens within action a_t.

We can just multiply by R(τ) to get ∇θJ.

Computing the policy gradient

Effectively, we just do masking and rescaling of the standard SFT loss computation.

<|im_start|>user ↵ Find·the·number·in·1–16. ↵ <|im_end|> ↵ <|im_start|>assistant ↵
guess(8) ↵ <|im_end|> ↵ <|im_start|>tool ↵ higher ↵ <|im_end|> ↵
<|im_start|>assistant ↵ guess(12) ↵ <|im_end|> ↵ … ↵ <|im_start|>assistant ↵
guess(11) ↵ <|im_end|>

"·" = space, "↵" = newline · colored = sampled by the policy, gray = context only

$$\mathcal{L}_{\mathrm{PG}} = -R(\tau) \sum_{t\,:\,m_t=1} \log \pi_\theta(u_t \mid u_{<t})$$

$m_t = 1$
tokens the policy sampled, including the end-of-turn token

$m_t = 0$
prompt and observation tokens, which the policy did not choose

R(τ)
one scalar, shared by every generated token in the trajectory

Policy-gradient training loop

After an update, the new policy collects the next batch.

1 Collect

Sample task instances; sample from the agent on them.

→

2 Evaluate

Compute rewards for each trajectory.

→

3 Form loss

Weight action log probabilities by reward.

→

4 Update

Backpropagate, change θ, collect again.

↺

$$\mathcal{L}_{\mathrm{PG}} = -\sum_{i} R(\tau_i) \sum_{t} \log \pi_\theta(a_{i,t} \mid h_{i,t})$$

With binary rewards, this looks a lot like ReST

REINFORCE

$$-\sum_i R(\tau_i) \sum_t \log \pi_\theta(a_{i,t} \mid h_{i,t})$$

=

SFT loss on the successful trajectories

$$-\sum_{i:R(\tau_i)=1} \sum_t \log \pi_\theta(a_{i,t} \mid h_{i,t})$$

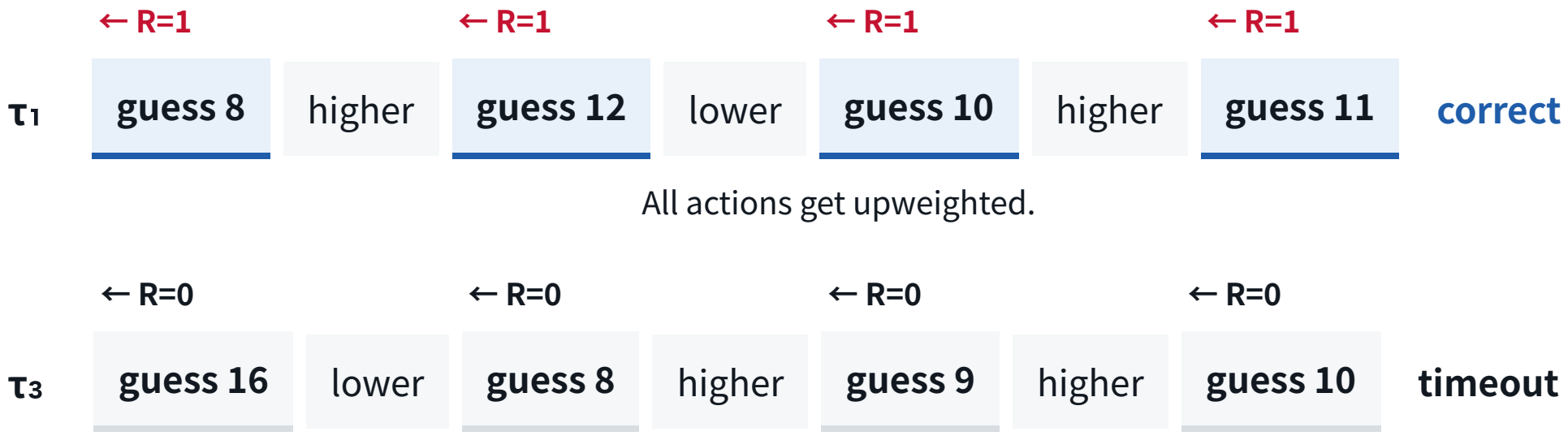
(there are some differences, e.g. ReST does batched updates and aggregates datasets)

Williams · REINFORCE ↗ Singh et al. · ReST-EM ↗

Better than expected

Baselines and advantages

Rewards should be relative



The update should depend on how well the policy usually does on this example.

Advantages

Advantage: how much better is this action, than the policy's average?

$$A^{\pi}(h,a) = \underbrace{Q^{\pi}(h,a)}_{\text{expected reward after } a} - \underbrace{V^{\pi}(h)}_{\text{expected reward from } h}$$

History h: guess 8 / higher · guess 12 / lower — the number is 9, 10 or 11, with two guesses left.

guess 11

if wrong, two candidates and one guess left

Aπ(h,11) < 0

Vπ(h)

guess 10

if wrong, one candidate and one guess left

Aπ(h,10) > 0

Neither Qπ nor Vπ is known, so we estimate advantage from the sampled reward.

$$\hat{A}_t = \underbrace{R(\tau)}_{\text{trajectory reward}} - \underbrace{b(h_t)}_{\text{an estimate of } V^{\pi}(h_t)}$$

Policy gradients with advantage estimates

Incorporate advantages by recentering the reward.

REINFORCE: one trajectory reward weights every action.

$$\widehat{\nabla_\theta J} = R(\tau) \sum_t \nabla_\theta \log \pi_\theta(a_t \mid h_t)$$

Subtract a baseline — the weight is now $\hat{A}_t$.

$$\widehat{\nabla_\theta J} = \sum_t \underbrace{\big(R(\tau) - b(h_t)\big)}_{\hat{A}_t} \nabla_\theta \log \pi_\theta(a_t \mid h_t)$$

Both lines are different ways to estimate the same ∇θJ.

$\hat{A}_t = R(\tau) - b(h_t)$. Different choices for b produce different algorithms

a constant
upcoming bandit example

a running mean of rewards
adapts as the policy improves

Vϕ(h_t)
a trained value model: actor–critic, PPO

the group mean
no extra model: GRPO (next section)

Baselines do not bias the gradient estimator

$$\widehat{\nabla_\theta J} = \sum_t \big(R(\tau) - b(h_t)\big)\, \nabla_\theta \log \pi_\theta(a_t \mid h_t)$$

$$\mathbb{E}\big[\widehat{\nabla_\theta J}\big] = \nabla_\theta J - \mathbb{E}\Big[\sum_t b(h_t)\, \nabla_\theta \log \pi_\theta\Big]$$

so this is an unbiased estimate of J's gradient if the expectation is 0

$$\mathbb{E}_{a\sim\pi_\theta(\cdot|h)}[b(h)\nabla_\theta \log \pi_\theta(a \mid h)]$$

$$= b(h) \sum_a \pi_\theta(a \mid h)\nabla_\theta \log \pi_\theta(a \mid h)$$

$$= b(h) \sum_a \nabla_\theta \pi_\theta(a \mid h)$$

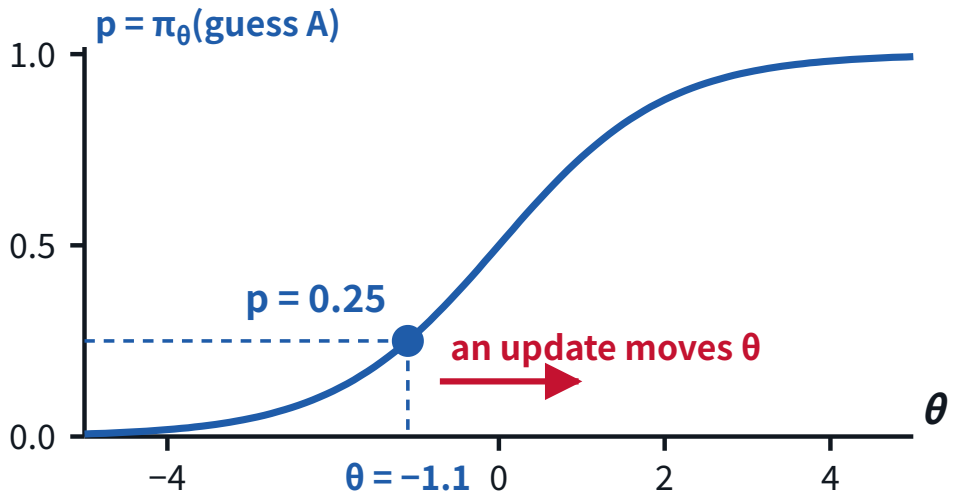
$\pi \nabla \log \pi = \nabla \pi$

$$= b(h)\nabla_\theta \sum_a \pi_\theta(a \mid h) = b(h)\nabla_\theta 1 = \mathbf{0}$$

So $\mathbb{E}[\nabla\theta J] = \nabla\theta J$ for any baseline that conditions only on the history

Simple example: two-armed bandit

With one policy parameter and two actions, we can compute gradients exactly.



The sigmoid σ turns the single real number θ into a probability.

guess A	p = σ(θ) = 0.25	reward 1
guess B	1 − p = 0.75	reward 0

Raising θ raises p(guess A) and lowers p(guess B)

sampled guess A · reward 1

reward R(τ)		slope ∂ / ∂ θ log π_θ(a)		gradient on θ
R = 1	×	1 − p = +0.75	=	+0.75

if we sample A, A becomes more likely

sampled guess B · reward 0

reward R(τ)		slope ∂ / ∂ θ log π_θ(a)		gradient on θ
R = 0	×	−p = −0.25	=	0

if we sample B, nothing happens

This gradient is correct, but a baseline/advantage can make it more useful.

Simple example: Adding a baseline

Subtracting a baseline says whether a reward was better or worse than expected.

guess A	$p = \sigma(\theta) = 0.25$	reward 1

guess B	$1 - p = 0.75$	reward 0

reward 0 — **baseline $b(h_0) = 0.5$** — reward 1

worse than expected · $\hat{A} < 0$

$\hat{A} > 0$ · better than expected

sampled guess A · reward 1

advantage $\hat{A} = R - b$ **$1 - 0.5 = +0.5$** × slope $\partial / \partial\theta \log \pi_\theta(a)$ **$1 - p = +0.75$** = gradient on θ **+0.375** without a baseline: +0.75

increases p(guess A)

sampled guess B · reward 0

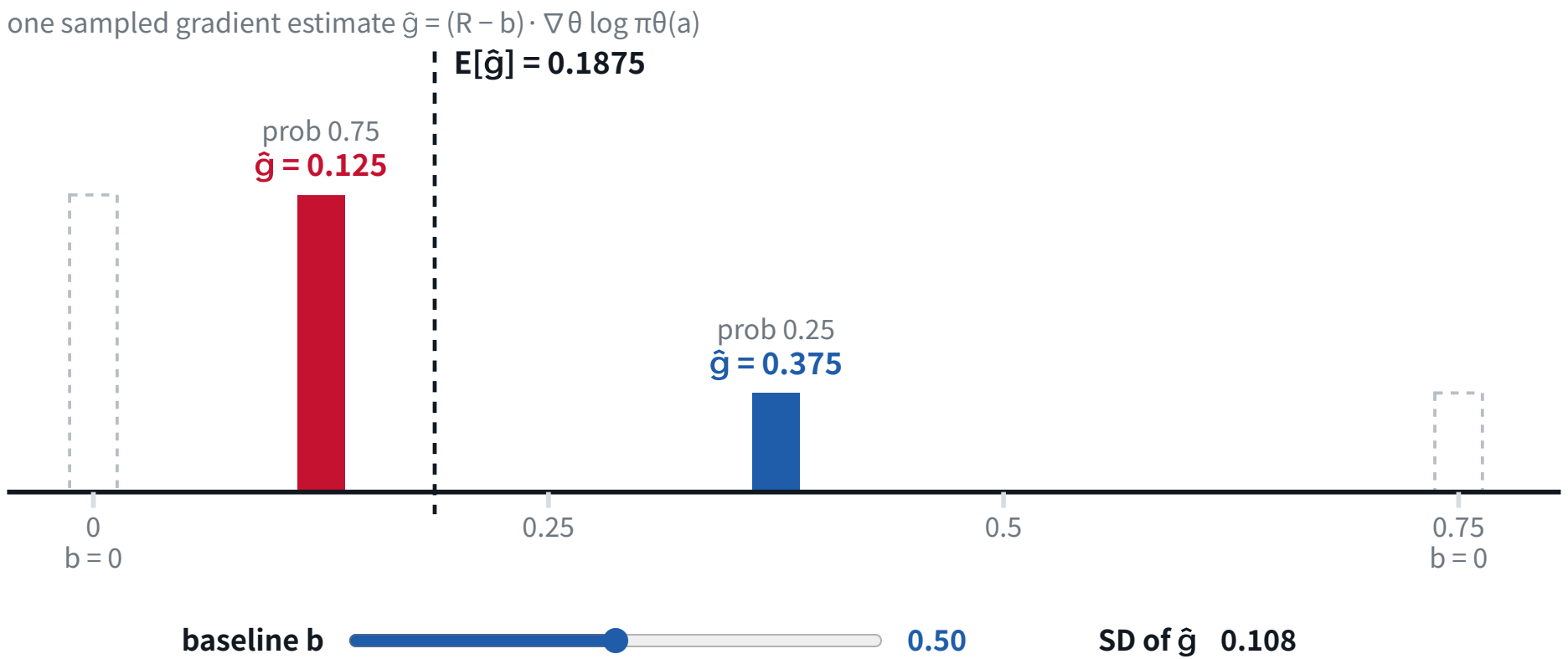
advantage $\hat{A} = R - b$ **$0 - 0.5 = -0.5$** × slope $\partial / \partial\theta \log \pi_\theta(a)$ **$-p = -0.25$** = gradient on θ **+0.125** without a baseline: 0

increases p(guess A), so decreases p(guess B)

The baseline adds a learning signal from failure while preserving the expected gradient.

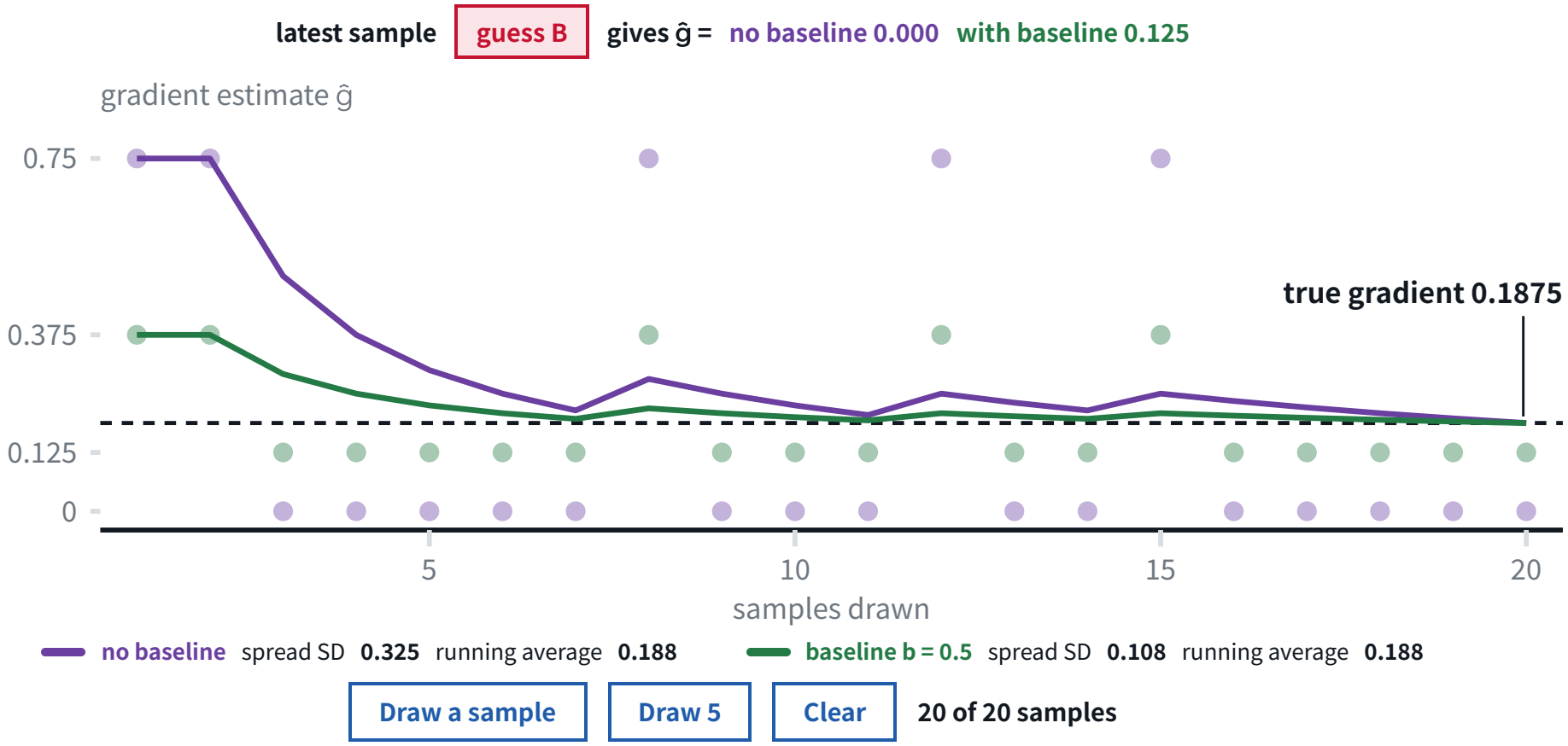
Simple example: varying the baseline

Changing the baseline changes the variance of the gradient estimate, while keeping the expectation the same



Simple example: noise in the gradient estimate

Computed a running estimate of the gradient by repeated sampling, with no baseline and with baseline 0.5.



Both averages converge to the same value, but the baseline has lower variance.

Baselines and credit assignment

What are the advantages of baselines (so far)?

Helps

Marks a trajectory as better or worse than expected.

A good baseline reduces gradient variance.

Does not solve

All actions have the same advantage $\hat{A}$.

Estimating $V^{\pi}(h)$ may require a separate value model.

$\hat{A} = +0.5$ | guess 8 | guess 12 | guess 10 | guess 11

What if we have intermediate rewards?

If the environment gives rewards before the final step, use reward-to-go to reduce variance.

$$R_t = \sum_{t'=t+1}^{T} r_{t'}$$

action → **tool reward +0.2** → action a_t → action → **terminal reward +1**

ignored — included in R_t

An action is weighted only by the rewards that come after it.

Earlier rewards do not depend on a_t, so dropping them cuts variance without adding bias.

Number Search pays out once, at the end: $R_t = R(\tau)$ for every action.

Comparing trajectories

Group-relative advantage estimates

Group-relative advantage estimates

Advantages: how much better is this trajectory than expected? We can compute this by sampling the policy multiple times. No separate model needed!

									reward R_j	centered $R_j - \bar{R}$	divide by σR	advantage $\hat{A}_j$
τ1	8	higher	12	lower	10	higher	11	correct	1	+0.5	÷0.5	+1
τ2	4	higher	8	higher	12	lower	11	correct	1	+0.5	÷0.5	+1
τ3	16	lower	8	higher	9	higher	10	timeout	0	−0.5	÷0.5	−1
τ4	1	higher	2	higher	3	higher	4	timeout	0	−0.5	÷0.5	−1
								from the group	$\bar{R} = 0.5$		$\sigma R = 0.5$	

$$b(h_t) = \bar{R} = \tfrac{1}{G} \sum_{j=1}^{G} R(\tau_j) \quad \text{for every } h_t \text{ in the group}$$

$$\hat{A}_j = \frac{R(\tau_j) - \bar{R}}{\sigma_R}, \qquad \bar{R} = b(h_t), \qquad \sigma_R = \mathrm{std}\{R(\tau_1), \ldots, R(\tau_G)\}$$

The GRPO update

Batches contain multiple task instances, each with its own group.

task instance	rewards of the G rollouts				group mean	advantage estimates $\hat{A}_{i,j}$			
x_1 · hidden 11	1	1	0	0	$\bar{R}_1 = 0.50$	+1	+1	−1	−1
x_2 · hidden 3	1	0	0	0	$\bar{R}_2 = 0.25$	+1.73	−0.58	−0.58	−0.58
x_3 · hidden 7	1	1	1	0	$\bar{R}_3 = 0.75$	+0.58	+0.58	+0.58	−1.73

policy-gradient loss · i indexes the batch

$$-\sum_i R(\tau_i) \sum_t \log \pi_\theta(a_{i,t} \mid h_{i,t})$$

→

GRPO loss · i batch, j group

$$-\sum_i \frac{1}{G} \sum_{j=1}^{G} \hat{A}_{ij} \sum_t \log \pi_\theta(a_{ij,t} \mid h_{ij,t})$$

Each task instance has its own $\bar{R}_i$ and σ_i.

Each task instance is normalized individually.

GRPO supplies group-relative weights to the same policy-gradient term.

Written for current-policy trajectories, without the paper's ratio, clipping, or KL terms.

Shao et al. · DeepSeekMath, § 4.1.2 ↗

DrGRPO's two changes

DrGRPO removes the group-standard-deviation and response-length divisions.

	GRPO	DrGRPO	Why it changes
Advantage estimate	$\frac{R_j - \bar{R}}{\sigma_R}$	$R_j - \bar{R}$	σ_R is measured inside each group, so it differs from one task instance to the next. Instances the policy almost always passes or almost always fails have a small σ_R, so they count for more.
Response aggregation	$\frac{1}{\lvert y_j \rvert} \sum_t (\cdot)$	$\frac{1}{C} \sum_t (\cdot)$	$\lvert y_j \rvert$ differs from one response to the next. A long incorrect response is penalized less per token than a short one, so the policy drifts toward longer wrong answers.

C is a global constant, the same for every response; the paper uses the generation budget.

Rewards stay fixed; the relative weights of task instances and responses change.

Liu et al. · DrGRPO, Fig. 1 and § 3.2 ↗

Token coefficients in DrGRPO

For one task instance, every generated token in trajectory j shares that trajectory's advantage estimate.

τ1
$\hat{A}_1 = +0.5$

<|im_start|>assistant ↵ guess(8) ↵ <|im_end|> ↵ <|im_start|>tool ↵ higher ↵ <|im_end|> ↵ … ↵ <|im_start|>assistant ↵ guess(11) ↵ <|im_end|>

PER TOKEN
+0.5 / C

τ3
$\hat{A}_3 = -0.5$

<|im_start|>assistant ↵ guess(16) ↵ <|im_end|> ↵ <|im_start|>tool ↵ lower ↵ <|im_end|> ↵ … ↵ <|im_start|>assistant ↵ guess(10) ↵ <|im_end|>

PER TOKEN
−0.5 / C

$$\mathcal{L}_{\mathrm{DrGRPO}} = -\frac{1}{G}\sum_{j=1}^{G}\frac{1}{C}\sum_{t\,:\,m_{j,t}=1}\hat{A}_j\,\log\pi_\theta(u_{j,t}\mid u_{j,<t})$$

$\hat{A}_j = R(\tau_j) - \bar{R}$, with no division by σR.

Simplified: the paper's objective also has a ratio and clipping.

Liu et al. · DrGRPO, § 3.2 ↗ Shao et al. · DeepSeekMath, Eq. 3 ↗

Equal rewards and zero advantage estimates

When every reward in a group matches, the group is degenerate and contributes no gradient.

ALL TRAJECTORIES FAIL

$[0, 0, 0, 0] \rightarrow [0, 0, 0, 0]$

possibly too hard for the current policy

ALL TRAJECTORIES SUCCEED

$[1, 1, 1, 1] \rightarrow [0, 0, 0, 0]$

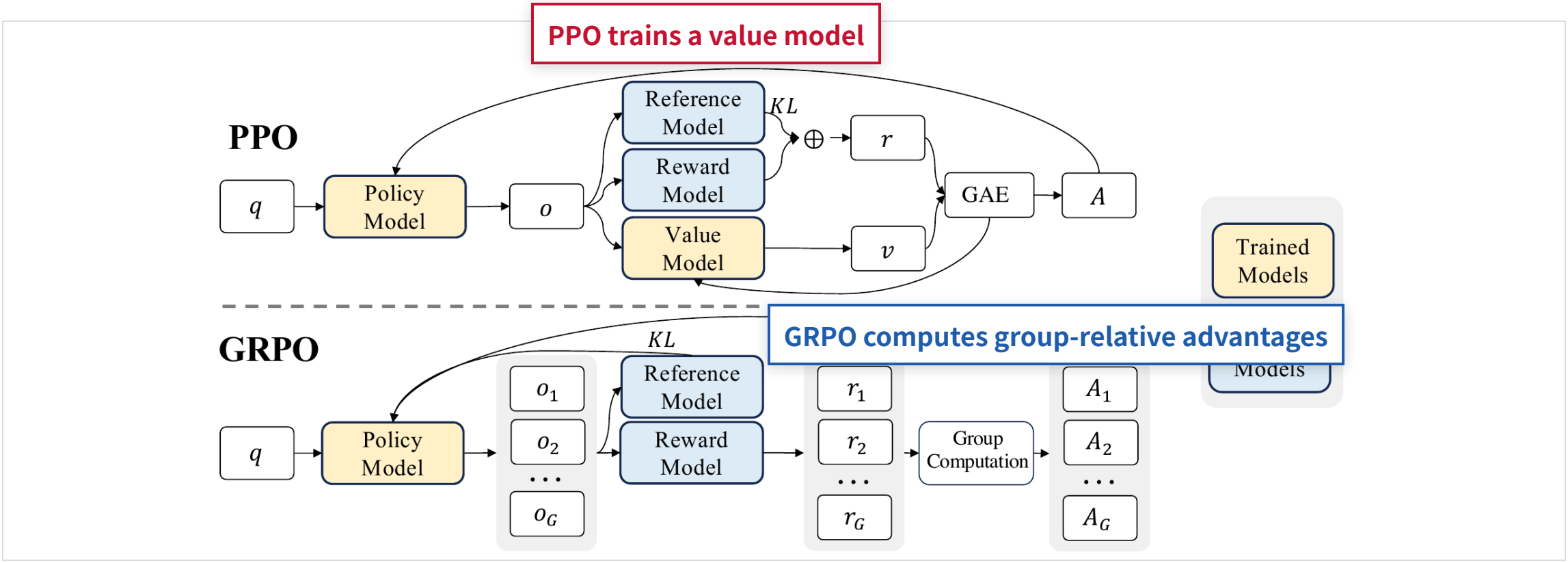
possibly too easy for the current policy

What could we change if every sampled trajectory fails?

Choose different task instances, broaden exploration, add demonstrations, or supply a more informative reward.

GRPO vs PPO

GRPO was proposed as a more efficient alternative to PPO, which uses a separate model to estimate the advantage baseline.



Shao et al. · DeepSeekMath, Fig. 4 and § 4.1.2 ↗

The advanced RL lecture covers ratios, clipping, and reference-model terms.

Summary

Comparing the methods we've seen so far

Every method today weights the actions the agent took. They (mainly) differ in the weight.

$$\widehat{\nabla_\theta J} = \sum_t w_t \nabla_\theta \log \pi_\theta(a_t \mid h_t)$$

ReST-EM	w_t = 1 on a successful trajectory, 0 on the rest (the SFT loss on what was kept)
REINFORCE	w_t = R(τ), the reward of the trajectory the action came from
GRPO/DrGRPO	w_t = $\hat{A}_j$, the reward relative to the other rollouts on the same task instance

τ1	**8**	**higher**	**12**	**lower**	**10**	**higher**	**11**	**correct**	**R = 1**
τ2	**4**	**higher**	**8**	**higher**	**12**	**lower**	**11**	**correct**	**R = 1**
τ3	**16**	**lower**	**8**	**higher**	**9**	**higher**	**10**	**timeout**	**R = 0**
τ4	**1**	**higher**	**2**	**higher**	**3**	**higher**	**4**	**timeout**	**R = 0**

Takeaways

Sampling trajectories from the policy and weighting them by reward addresses the three problems we started with.

Task mismatch

The objective is expected reward, so any trajectory that succeeds is reinforced.

Data mismatch

The agent can also learn from sub-optimal trajectories, based on their reward.

Exposure bias

The trajectories come from the current policy, so the agent is trained to recover from its own mistakes.

Questions?

Upcoming: async RL / off-policy data, importance weighting