CIS 6200: Emerging Theory of LLMs **Fall 2026**

Topic 5B: Learning from Outcome Rewards

Last updated: October 20, 2026 *Surbhi Goel*

Acknowledgements: *The policy-gradient derivation follows Williams [1]. The presentation draws on the lecture notes of Davis, Eysenbach, and Golowich [3, 4, 5]. The comparison of training objectives follows Davis and Recht [6].*

AI Disclosure: *These notes were drafted and revised with AI assistance. If you find an error, please let me know.*

In the last lecture, we used sampling and selection to improve the answer returned by a fixed model. We also saw how maximum likelihood can give that model enough coverage to find correct responses. We now use feedback on those responses to train the model itself: can we increase its probability of being correct on a single attempt?

Reasoning models and reinforcement learning. In September 2024, OpenAI introduced o1 [2], trained with reinforcement learning to use a chain of thought before answering. OpenAI reported that reasoning performance improved both with more reinforcement-learning training and with more computation at inference (Figure 1). These are two different uses of compute: training changes the model, while inference gives the trained model more time to reason through an answer. The announcement made reinforcement learning prominent in the development of reasoning models, but did not give a detailed training algorithm.

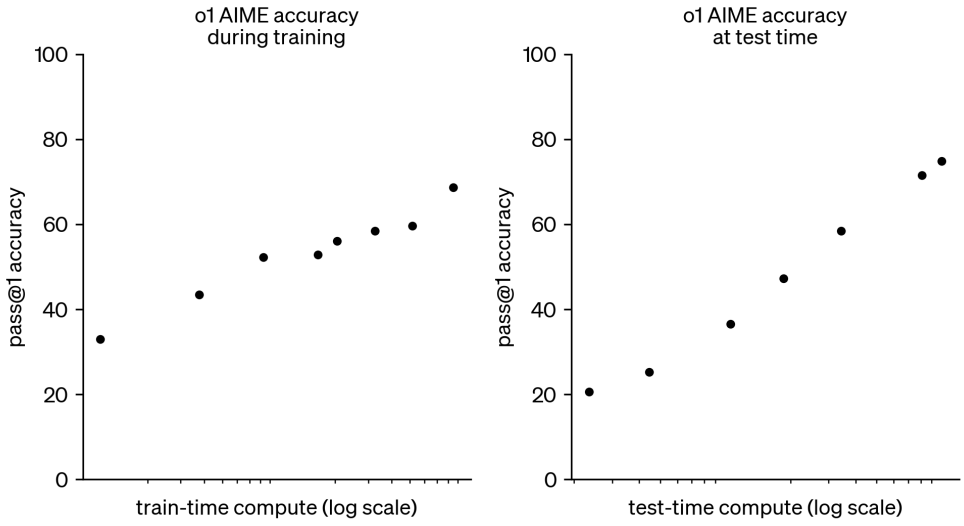

Figure 1: o1's AIME math-competition accuracy improves with more training compute (left) and inference compute (right). Both panels measure `pass@1`, or single-response accuracy. The compute axes are logarithmic. Reproduced from OpenAI [2].

DeepSeek-R1 [9] (January 2025) provided a detailed training recipe and released model weights. Its R1-Zero experiment applied reinforcement learning directly to a pretrained model, without an

intervening supervised fine-tuning stage. Correctness and formatting rewards produced substantial reasoning gains, with longer traces and self-checking behavior. Figure 2 shows the improvement during training. The final R1 model combined supervised training with reinforcement learning. Both used **group relative policy optimization (GRPO)**, introduced in DeepSeekMath [8] in February 2024.

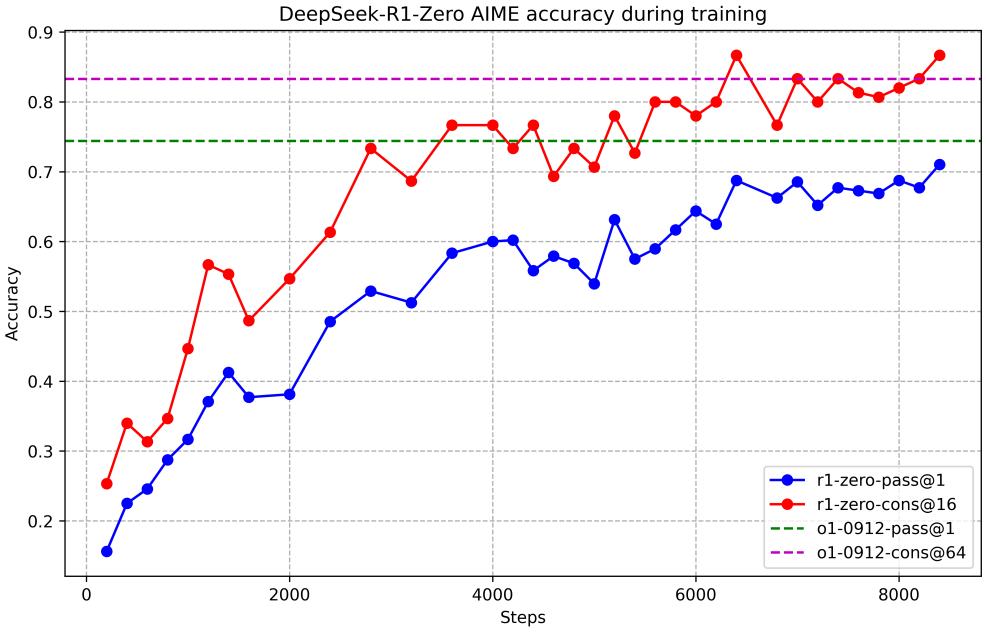

Figure 2: DeepSeek-R1-Zero's AIME 2024 accuracy during reinforcement learning. The blue curve estimates single-response accuracy by averaging over 16 responses per question. The red curve uses majority voting among 16 responses. The dashed o1 references use one response (green) and voting among 64 responses (purple). Reinforcement learning improves single-response accuracy; voting adds a gain at inference. Reproduced from DeepSeek-AI [9], Figure 2.

The underlying gradient-estimation ideas date back to Williams's REINFORCE algorithms [1] (1992). Our question is: when we train on generated responses and their rewards, what gradient are we estimating? We derive the policy gradient, explain how baselines preserve its expectation while changing its variance, and then compare sampling and weighting rules through the objectives they induce.

1 Learning from rewards

Let $x \sim \mathcal{D}$ be a prompt drawn from a fixed training distribution. As before, $\mathcal{U}$ is a finite set of complete responses, including any chain of thought, obtained by bounding response length over a finite vocabulary. We now write the generator as $q_\theta(y \mid x)$ to make its trainable parameters explicit. In reinforcement-learning terminology, a complete response is an **action** and the distribution $q_\theta(\cdot \mid x)$ is the **policy**.

Let $r(x, y) \in \mathbb{R}$ be a fixed outcome reward for a complete response. It could indicate correctness or give a graded quality score. Reinforcement learning does not require binary rewards. Define the expected reward for a prompt by

$$\alpha_\theta(x) = \mathbb{E}_{y \sim q_\theta(\cdot|x)}[r(x, y)].$$

The objective is average expected reward:

$$J(\theta) = \mathbb{E}_{x\sim\mathcal{D}}[\alpha_\theta(x)] = \mathbb{E}_{x\sim\mathcal{D},\, y\sim q_\theta(\cdot|x)}[r(x,y)]. \tag{1}$$

For a binary correctness reward, $\alpha_\theta(x)$ is the probability of a correct response, as in the previous lecture. For graded rewards, it denotes expected reward. If the reward comes from an imperfect verifier, we optimize the score it supplies. We assume finitely many training prompts and differentiable, positive response probabilities. The reward and prompt distribution are fixed as θ changes. All logarithms are natural.

1.1 The policy gradient

Fix a prompt x and suppress it in the notation:

$$\alpha_\theta = \mathbb{E}_{y\sim q_\theta}[r(y)] = \sum_{y\in\mathcal{U}} q_\theta(y) r(y).$$

The reward $r(y)$ is fixed; the parameter θ changes the probability of generating each response. Writing out the expectation makes this dependence explicit.

We can differentiate this finite sum, but there are too many responses to evaluate the gradient by enumeration. Instead, we rewrite it as an expectation under q_θ, which we can sample from:

$$\begin{aligned} \nabla_\theta \alpha_\theta &= \sum_y r(y) \nabla_\theta q_\theta(y) \\ &= \sum_y q_\theta(y) r(y) \nabla_\theta \log q_\theta(y) \\ &= \mathbb{E}_{y\sim q_\theta}[r(y) \nabla_\theta \log q_\theta(y)]. \end{aligned} \tag{2}$$

The second equality uses the **log-derivative identity** $\nabla_\theta q_\theta(y) = q_\theta(y) \nabla_\theta \log q_\theta(y)$, which follows from the chain rule. Equation (2) is the **policy-gradient identity**.

To estimate the gradient, draw B independent responses $y^{(1)}, \ldots, y^{(B)} \sim q_\theta$ and evaluate $R_i = r(y^{(i)})$. Williams's **REINFORCE** estimator is

$$\widehat{g}_{\mathrm{RF}} = \frac{1}{B} \sum_{i=1}^{B} R_i \nabla_\theta \log q_\theta(y^{(i)}), \qquad \mathbb{E}[\widehat{g}_{\mathrm{RF}}] = \nabla_\theta \alpha_\theta. \tag{3}$$

This sample average is unbiased by (2). The derivation already accounts for how θ changes the distribution of generated responses by differentiating $q_\theta(y)$. To evaluate each summand, hold the prompt, sampled token identities, and reward fixed, and differentiate the response's log probability. Gradients pass through the model's embedding and hidden-state computations to its trainable parameters, but not through the discrete sampling operation or the reward calculation.

For binary rewards, a successful response contributes its likelihood gradient; an unsuccessful response contributes zero.

Sampling the prompt from $\mathcal{D}$ as well gives an unbiased estimator of $\nabla_\theta J(\theta)$. REINFORCE is one instance of the following update, with weights $w_i = R_i$ and step size $\eta > 0$. Other choices of weights can change the estimator's variance or the objective it optimizes.

A common policy-gradient update [6]

1. Draw a prompt $x \sim \mathcal{D}$.
2. Generate B independent responses $y^{(1)}, \ldots, y^{(B)} \sim q_\theta(\cdot \mid x)$ and evaluate $R_i = r(x, y^{(i)})$.
3. Choose a weight w_i for each response.
4. Take a gradient-ascent step:

$$\theta \leftarrow \theta + \eta \widehat{g}, \qquad \widehat{g} = \frac{1}{B} \sum_{i=1}^{B} w_i \nabla_\theta \log q_\theta(y^{(i)} \mid x). \tag{4}$$

Treat the weights w_i as fixed numerical coefficients in the likelihood-gradient calculation.

1.2 The LLM update and supervised fine-tuning

Ordinary **supervised fine-tuning (SFT)** maximizes the log likelihood of demonstration responses from a fixed dataset. REINFORCE estimates the gradient of expected reward using the same likelihood derivatives, with different responses and weights:

Method	Source of the response y	Gradient weight
SFT	Fixed demonstration dataset	1
REINFORCE	Current model $q_\theta(\cdot \mid x)$	$r(x, y)$

Generating responses at the parameters where we evaluate the gradient is **on-policy sampling**. After changing θ, reusing an old batch generally no longer gives an unbiased estimate of the new reward gradient. We use fresh batches for each update.

To see how this update acts on tokens, write a complete response as $y = (y_1, \ldots, y_T)$, including the end-of-sequence token if generation stops before the length cap. Its log probability is

$$\log q_\theta(y \mid x) = \sum_{t=1}^{T} \log q_\theta(y_t \mid x, y_{<t}).$$

For a response with reward $R = r(x, y)$, its contribution to the REINFORCE update is therefore

$$R \nabla_\theta \log q_\theta(y \mid x) = \underbrace{R}_{\text{fixed weight}} \sum_{t=1}^{T} \nabla_\theta \log q_\theta(y_t \mid x, y_{<t}).$$

The same scalar outcome reward multiplies every token's gradient term. If a response receives a positive reward for its final answer, a mistaken intermediate step still receives that positive weight. The token gradient vectors differ, but their weight is shared: an outcome reward does not certify each step or identify which tokens caused success.

2 Estimating the same gradient with a baseline

An unbiased gradient estimate can still be noisy: different batches can give very different update directions. Averaging more independent REINFORCE samples reduces variance but requires more generations. We can also reduce noise by changing the weights on the sampled responses.

For example, suppose every response to a prompt receives reward 1. Then $\alpha_\theta = 1$ for every θ, so the true gradient is zero. Yet a single REINFORCE sample gives $\nabla_\theta \log q_\theta(y)$, which generally is not zero: it reinforces whichever response happened to be sampled. These updates cancel only in expectation. Subtracting 1 from the reward makes every sampled update zero, removing the noise entirely.

More generally, subtract a scalar **baseline** c from the reward, giving weight $r(y) - c$. Fix c for the current prompt and model, without using the sampled response. Its value may depend on θ, but we hold it fixed when differentiating. By the log-derivative identity,

$$\mathbb{E}_{y\sim q_\theta}[\nabla_\theta \log q_\theta(y)] = \sum_y \nabla_\theta q_\theta(y) = \nabla_\theta \sum_y q_\theta(y) = 0.$$

The last equality holds because the probabilities sum to 1. Subtracting the baseline therefore removes a term whose expectation is zero:

$$\begin{aligned}\mathbb{E}[(r(y) - c)\nabla_\theta \log q_\theta(y)] &= \mathbb{E}[r(y)\nabla_\theta \log q_\theta(y)] - c\,\mathbb{E}[\nabla_\theta \log q_\theta(y)] \\ &= \nabla_\theta \alpha_\theta.\end{aligned} \tag{5}$$

Choosing the baseline. To find the least noisy constant baseline, write $s(y) = \nabla_\theta \log q_\theta(y)$ and $g_c(y) = (r(y) - c)s(y)$. The sum of the variances of the gradient coordinates is

$$\mathbb{E}\|g_c - \mathbb{E}g_c\|^2 = \mathbb{E}[(r(y) - c)^2\|s(y)\|^2] - \|\nabla_\theta \alpha_\theta\|^2.$$

The last term does not depend on c, because the baseline leaves $\mathbb{E}g_c$ unchanged. We therefore minimize a weighted squared error in c: a response with a larger likelihood gradient contributes more to the variance. Differentiating with respect to c and setting the result to zero gives

$$\begin{aligned}2\mathbb{E}[(c - r(y))\|s(y)\|^2] &= 0, \\ c_{\text{opt}} &= \frac{\mathbb{E}[r(y)\|s(y)\|^2]}{\mathbb{E}[\|s(y)\|^2]},\end{aligned} \tag{6}$$

provided $\mathbb{E}\|s(y)\|^2 > 0$ [11]. This is a mean reward weighted by squared gradient norms. It cannot have greater variance than using $c = 0$.

Computing these individual gradient norms is more involved than estimating the mean reward $\alpha_\theta(x)$ from scalar rewards alone. The mean reward minimizes the unweighted error $\mathbb{E}[(r - c)^2]$ and agrees with c_{opt} when the gradient norms are constant. It is a convenient approximation, though it need not reduce gradient variance for every policy.

With this baseline, $r(x, y) - \alpha_\theta(x)$ measures how a response performs relative to an average response to the same prompt. This centered reward is its **advantage** [12]: positive for above-average responses and negative for below-average responses.

Estimating the mean reward. We do not know $\alpha_\theta(x)$, and it changes as the model learns. Responses generated for the same prompt give us a direct estimate, without training a separate model to predict it [13].

Fix x, draw $B \geq 2$ independent responses from the current model, and write $s_i = s(y^{(i)})$. To keep the baseline independent of response i, estimate it using the other $B - 1$ responses:

$$c_{-i} = \frac{1}{B-1}\sum_{j\neq i} R_j, \qquad \widehat{g}_{\text{LOO}} = \frac{1}{B}\sum_{i=1}^{B}(R_i - c_{-i})s_i.$$

This is **REINFORCE leave-one-out (RLOO)** [10]. Conditional on the prompt and current model, c_{-i} is independent of $y^{(i)}$. Consequently,

$$\mathbb{E}[c_{-i}s_i] = \mathbb{E}[c_{-i}]\mathbb{E}[s_i] = 0, \qquad \mathbb{E}[\widehat{g}_{\mathrm{LOO}}] = \nabla_\theta \alpha_\theta(x).$$

RLOO is therefore a way to implement REINFORCE with an estimated baseline. This calculation holds for general scalar rewards.

Including a response in its own baseline. Using the whole-group mean $\bar{R} = B^{-1}\sum_i R_i$ is nearly the same operation, but the response now affects its own baseline. For every sampled group,

$$R_i - \bar{R} = \frac{B-1}{B}(R_i - c_{-i}).$$

Thus the expected update with weights $R_i - \bar{R}$ is $(1 - 1/B)\nabla_\theta \alpha_\theta(x)$. For fixed B, this factor can be absorbed into the step size.

3 A unifying view of RL objectives

REINFORCE and RLOO estimate the same reward gradient. Changing how we sample or weight responses can change the objective itself. Following Davis and Recht [6], we compare several such methods by identifying the objective whose gradient they estimate.

For this comparison, specialize to binary correctness rewards:

$$r(x, y) \in \{0, 1\}, \qquad G(x) = \{y : r(x, y) = 1\}, \qquad \alpha_\theta(x) = q_\theta(G(x) \mid x).$$

Assume $0 < \alpha_\theta(x) < 1$ for each training prompt.

The methods below lead to objectives of the following form, where h is an increasing, differentiable function:

$$J_h(\theta) = \mathbb{E}_x[h(\alpha_\theta(x))], \qquad \nabla_\theta J_h = \mathbb{E}_x[h'(\alpha_\theta(x))\nabla_\theta \alpha_\theta(x)], \tag{7}$$

REINFORCE and RLOO correspond to $h(u) = u$, giving average accuracy. For each new sampling or weighting rule, we compute its expected update and identify h from the multiplier $h'(\alpha)$.

3.1 Fitting successful responses

With binary rewards, REINFORCE averages successful responses' likelihood gradients over the total number of samples. Another procedure is to generate until a response succeeds, then take a likelihood step on that response.

Fix a prompt, write $\alpha = \alpha_\theta$, and hold the model fixed while sampling. The first successful response is distributed as q_θ conditioned on G:

$$q_\theta(y \mid G) = \frac{q_\theta(y)\mathbf{1}\{y \in G\}}{\alpha}.$$

We fit the retained response using its original model log probability $\log q_\theta(y)$. Using (2), the expected update is

$$\mathbb{E}[\nabla_\theta \log q_\theta(y) \mid y \in G] = \frac{\mathbb{E}[r(y)\nabla_\theta \log q_\theta(y)]}{\alpha} = \frac{\nabla_\theta \alpha_\theta}{\alpha_\theta} = \nabla_\theta \log \alpha_\theta. \tag{8}$$

Thus sampling until success and fitting the retained response estimates the gradient of J_h with $h(u) = \log u$. The log-success objective arises from the sampling rule: conditioning supplies the factor $1/\alpha$.

To see what this changes, consider two equally likely prompts:

Model	Prompt x_1	Prompt x_2	Average accuracy
A	0.2	0.8	0.5
B	0.5	0.5	0.5

For Model A, one-sample REINFORCE contributes a successful response's likelihood gradient on 20% of visits to x_1 and 80% of visits to x_2. Sampling until success supplies one on every visit to either prompt. This scales their expected updates by 5 and 1.25, respectively, as in (8). Within each prompt, the relative probabilities of successful responses are unchanged.

The two models have the same average accuracy, but their average log success probabilities differ:

$$\underbrace{\tfrac{1}{2}(\log 0.2 + \log 0.8)}_{\text{Model A}} = \log 0.4 < \log 0.5 = \underbrace{\tfrac{1}{2}(\log 0.5 + \log 0.5)}_{\text{Model B}}.$$

The objective induced by fitting successful responses prefers Model B's more balanced success probabilities. The rewards remain binary; the sampling rule changes how the objective combines prompts.

The sampling cost is variable: finding a success takes $1/\alpha$ generations on average [6, Section 5]. It can be expensive for hard prompts, and there is no fixed cap on the number of attempts.

3.2 Fitting successes with a fixed budget

To control the number of attempts, generate exactly B responses per prompt. Average the likelihood gradients of its successful responses. If none succeeds, make no update and move to the next prompt without retrying. We now have a fixed number of generations, but some prompts contribute zero; this changes the expected gradient.

Fix a prompt, draw $y^{(1)}, \ldots, y^{(B)}$ independently from the current model, and let $K = \sum_{i=1}^{B} R_i$ be the number of successes. Define

$$\widehat{g}_{\text{keep}} = \begin{cases} \dfrac{1}{K} \displaystyle\sum_{i:R_i=1} \nabla_\theta \log q_\theta(y^{(i)}), & K > 0, \\ 0, & K = 0. \end{cases} \tag{9}$$

For $K > 0$, this is the boxed update with weights $w_i = BR_i/K = R_i/\bar{R}$: we divide by the number of successes rather than the batch size.

Proposition 3.1 (The expected update under a fixed budget)**.** *For $\alpha = \alpha_\theta > 0$ and $B \geq 1$,*

$$\mathbb{E}[\widehat{g}_{\text{keep}}] = \left[1 - (1-\alpha)^B\right] \nabla_\theta \log \alpha_\theta. \tag{10}$$

Proof. Condition on which sample positions succeed. For any pattern with $K > 0$, each successful response has the current model's distribution conditioned on G. By (8), each of its likelihood gradients has conditional mean $\nabla_\theta \log \alpha_\theta$. Averaging the K gradients preserves this mean. The probability that $K > 0$ is $1 - (1-\alpha)^B$; otherwise the update is zero. □

The factor in brackets is pass@B: the probability that the batch contains a success. Dividing it by α gives the multiplier of $\nabla_\theta \alpha_\theta$. It is the derivative of

$$h_B(u) = -\sum_{k=1}^{B} \frac{(1-u)^k}{k}, \qquad h_B'(u) = \frac{1-(1-u)^B}{u}. \tag{11}$$

Thus the update estimates $\nabla_\theta J_{h_B}$. This estimator and its objective were analyzed by Davis and Recht (2025), whose formula differs by an additive constant [6, Section 5]. The objective truncates the log-success objective:

$$\log u = -\sum_{k=1}^{\infty} \frac{(1-u)^k}{k}, \qquad 0 < u \le 1.$$

Keeping the first B terms gives h_B. At $B = 1$, $h_B(u) = u - 1$ gives the accuracy objective up to a constant; as B grows, $h_B(u)$ approaches $\log u$ for each $u > 0$.

The later **maximum likelihood reinforcement learning (MaxRL)** paper [7] emphasizes the following pass@k interpretation and evaluates the approach empirically. Since $\text{pass@}k_\theta(x) = 1 - (1 - \alpha_\theta(x))^k$,

$$h_B(\alpha_\theta(x)) = \sum_{k=1}^{B} \frac{\text{pass@}k_\theta(x) - 1}{k}.$$

Up to a constant, J_{h_B} is a weighted sum of success probabilities at budgets $1, \dots, B$, averaged over prompts. Increasing B therefore changes the objective being estimated, as well as the amount of data used to estimate its gradient.

3.3 Centering and normalizing rewards: GRPO

GRPO gives another choice of weights. It starts with the group baseline $R_i - \bar{R}$ from Section 2, which compares each response with the group's average reward. These centered weights have less spread in a nearly all-correct group than in a group with an equal mix of successes and failures.

GRPO divides by the group's reward standard deviation to put the weights on a common scale [8, 9]. For $B \ge 2$ independent responses, its weights are

$$w_i = \frac{R_i - \bar{R}}{\sqrt{B^{-1} \sum_j (R_j - \bar{R})^2}}, \qquad \bar{R} = \frac{1}{B} \sum_j R_j.$$

Set $w_i = 0$ when all rewards agree. Otherwise, the weights have mean zero and average square one. Successes receive positive weights and failures negative weights; the rarer outcome gets the larger magnitude. This normalization need not reduce gradient variance, since the likelihood gradients also vary.

We study this reward normalization in the response-level update (4), using fresh samples.

What population normalization optimizes. Fix x and write $\alpha = \alpha_\theta(x)$. Since the reward is binary, its mean is α and its variance is $\alpha(1-\alpha)$. Replace the sample mean and standard deviation by these population values. For $y \sim q_\theta(\cdot \mid x)$ and $R = r(x, y)$, the baseline identity gives

$$\begin{aligned} \mathbb{E}\left[\frac{R - \alpha}{\sqrt{\alpha(1-\alpha)}} \nabla_\theta \log q_\theta(y \mid x) \right] &= \frac{\nabla_\theta \alpha_\theta(x)}{\sqrt{\alpha(1-\alpha)}} \\ &= \nabla_\theta \left[2 \arcsin \sqrt{\alpha_\theta(x)} \right]. \end{aligned}$$

Thus population normalization gives J_h with $h(u) = 2 \arcsin \sqrt{u}$ [6, Section 6].

Finite groups. In a finite group, the sample mean and standard deviation depend on the same rewards as the numerator. Replacing them by population values inside the expectation is not valid: the finite-group objective depends on B, and the arcsine formula is not exact for it [6, Section 6].

What changes for non-binary rewards? Standardization also applies to graded rewards. With population moments, the expected update is $\nabla_\theta \alpha_\theta(x)/\sigma_\theta(x)$, where $\sigma_\theta(x) > 0$ is the reward standard deviation. Unlike the binary case, the variance is not determined by the mean, so this need not be the gradient of a function of mean reward alone.

Table 1 summarizes the binary-reward objectives and their gradient multipliers.

Method	$h(u)$	Multiplier $h'(u)$
REINFORCE or RLOO	u	1
Sample until success, then fit	$\log u$	$1/u$
Fixed-budget success fitting	$h_B(u)$	$[1-(1-u)^B]/u$
GRPO with population normalization	$2\arcsin\sqrt{u}$	$1/\sqrt{u(1-u)}$

Table 1: Objectives and expected-gradient multipliers for binary rewards. The GRPO row uses population moments.

For one prompt, these expected updates scale the same accuracy gradient; the multipliers alone do not determine their magnitudes. With shared parameters, $h'(\alpha)$ changes the relative contributions of prompts. Log-success fitting gives larger multipliers to hard prompts; a fixed budget limits this multiplier to B as α approaches zero. Population-normalized GRPO gives larger multipliers near both zero and one.

Figure 3 compares the methods on GSM8K, using binary correctness rewards. RLOO is REINFORCE with a leave-one-out baseline. Here MaxRL improves more slowly at first, but reaches higher pass@1 and retains more pass@32. The objective calculations above do not guarantee this ordering.

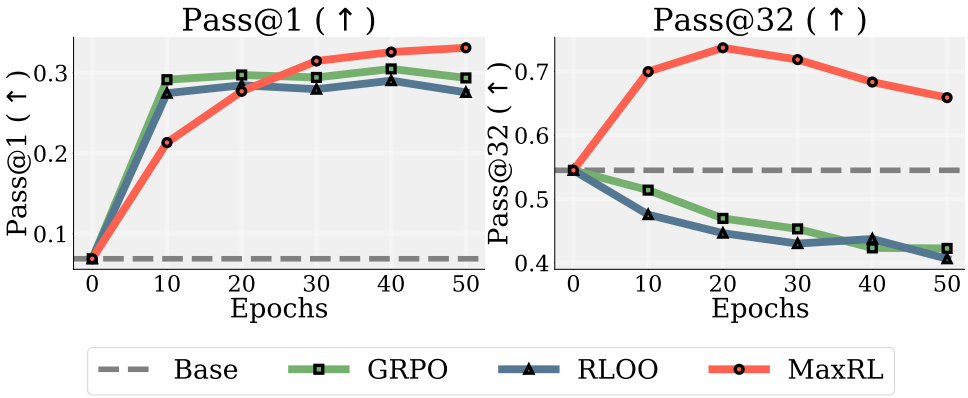

Figure 3: SmolLM2-360M-Instruct trained on GSM8K with 128 responses per prompt for each method. The MaxRL curve uses the paper's centered variant of the success-only update analyzed above. First two panels of [7, Figure 4, v3], with the original legend moved below.

We have assumed that a reward can be evaluated for each generated response. For many language tasks, we instead observe judgments comparing two responses. The next lecture studies how to learn from those comparisons, through RLHF and DPO.

References

[1] R. J. Williams. *Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning.* Machine Learning, 8:229–256, 1992. Paper.

[2] OpenAI. *Learning to Reason with LLMs.* September 12, 2024. Research announcement.

[3] D. Davis. *Reinforcement Learning for Language Models.* STAT 4830, Lecture 7. Lecture notes.

[4] B. Eysenbach. *The Policy Gradient.* Princeton COS 435/ECE 433, Lecture 6, 2025. Lecture notes.

[5] N. Golowich. *Reinforcement Learning with Verifiable Rewards.* UT Austin CS 395T, Lecture 13, 2026. Lecture notes.

[6] D. Davis and B. Recht. *What Is the Objective of Reasoning with Reinforcement Learning?* arXiv:2510.13651, 2025. Paper.

[7] F. Tajwar et al. *Maximum Likelihood Reinforcement Learning.* arXiv:2602.02710, 2026. Paper.

[8] Z. Shao et al. *DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models.* arXiv:2402.03300, 2024. Paper.

[9] DeepSeek-AI et al. *DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning.* arXiv:2501.12948, 2025. Paper.

[10] A. Ahmadian et al. *Back to Basics: Revisiting REINFORCE Style Optimization for Learning from Human Feedback in LLMs.* arXiv:2402.14740, 2024. Paper.

[11] E. Greensmith, P. L. Bartlett, and J. Baxter. *Variance Reduction Techniques for Gradient Estimates in Reinforcement Learning.* Journal of Machine Learning Research, 5:1471–1530, 2004. Paper.

[12] L. Weng. *Policy Gradient Algorithms.* Lil'Log, April 8, 2018. Blog post.

[13] S. C. Huang and A. Ahmadian. *Putting RL Back in RLHF.* Hugging Face, June 12, 2024. Blog post.