Lecture 6: Exploration in Reinforcement Learning (Part 1)

1 Introduction

Administrative

Final projects: keep pushing ahead and forming ideas. Due to many requests, HW due SUNDAY. Last extension.

- Project proposal: 11:59pm on March 20, 2025, posted publicly on Ed. The project proposal is an intermediate milestone. The deliverable is a 1 page PDF submitted on Ed.
- 1b/ 1:1 meeting with instructors: 11:59pm on Mar 31 (late deadline: April 7, 2025). During any of the office hours, have a 10 min conversation with a TA about your project.
- 1c/ Peer feedback on proposal: 11:59pm on Mar 27, 2025, submitted on Ed.

1.1 Where we are

So far in this course, we have built up two families of algorithms. On the value-based side, we developed Q-learning and DQN. On the policy optimization side, we covered REINFORCE, PPO, and GRPO. Last lecture, we brought these together in actor-critic methods — DDPG and TD3 for continuous control, or PPO with advantage estimation (or value function baselines). All of these algorithms assume that the agent can collect useful data through interaction. Recently, in LLM training land, a paper called ScaleRL explored how different variants and design decisions contributed to compute scaling with large-scale RL training for LLMs. Khatri et al. (2025) ran over 400,000 GPU-hours total experiments and found that **compute–performance curves for RL training follow a sigmoidal shape**. That is, as you increase training compute, performance follows an S-curve: slow initial progress, rapid improvement in the middle, and then diminishing returns as performance approaches an asymptote.

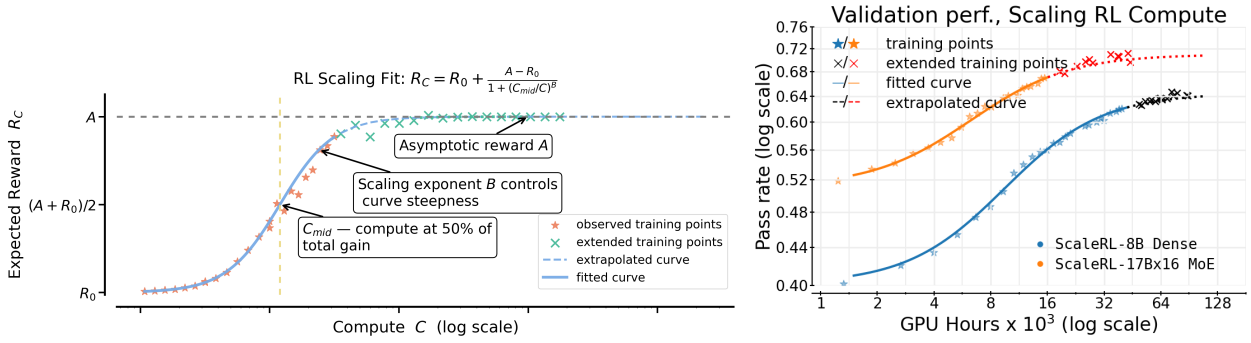

Figure 1: **Left:** The sigmoid scaling fit from Khatri et al. (2025): $R_C = R_0 + \frac{A-R_0}{1+(C_{\text{mid}}/C)^B}$.

Their study identified three key findings. First, not all training recipes yield similar asymptotic performance — some configurations are fundamentally better than others in the limit of infinite

compute. Second, design choices like loss aggregation, normalization, curriculum, and off-policy algorithms primarily modulate *compute efficiency* (how quickly you reach the asymptote) without materially shifting the ceiling. Third, stable recipes follow predictable sigmoid trajectories, enabling extrapolation: from early training data alone, they successfully predicted validation performance on a single RL run scaled to 100,000 GPU-hours.

This sigmoid pattern is not unique to LLM training. Similar curves have been observed in classic RL domains. Figure 2 shows learning curves from Atari games with sparse rewards: Montezuma's Revenge and other hard exploration games from Bellemare et al. (2016) and Burda et al. (2018).

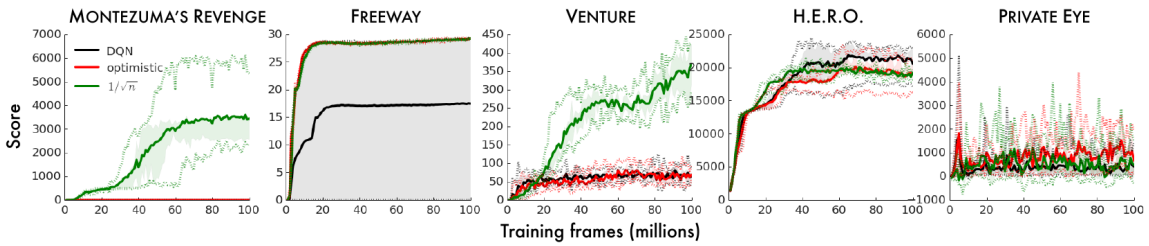

Figure 2: **Top:** Learning curves on hard Atari exploration games from Bellemare et al. (2016).

Discussion

In pretraining, scaling laws are typically power laws (Chinchilla-style). In RL training, the sigmoid means there is a *phase transition*: early training makes almost no progress, then there is a rapid takeoff. Why might this be? Look up these games and their reward functions. Think about why some might be easier to learn than others? You can find a description of the envs and their rewards on the benchmark webpage, https://ale.farama.org/environments/air_raid/. Discuss for 5 minutes. Tell me what the rewards are and why some might be harder than others. [Don't look ahead in the notes.]

1.2 Exploration as an explanation for the sigmoid

One potential explanation for the sigmoid shape is **exploration**. Consider an RL agent training on a task with sparse or delayed reward — say, solving a math problem where the reward is 1 if the answer is correct and 0 otherwise. Early in training, the policy is essentially random with respect to the task, so it rarely stumbles upon a correct solution. Without positive reward signal, policy gradient updates are uninformative. Why is that the case, even with advantage estimation?

The policy gradient with a baseline b is:

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta \log \pi_\theta(\tau^{(i)}) \left(R(\tau^{(i)}) - b \right)$$

Consider a sparse binary reward where $R(\tau) \in 0, 1$ and the policy succeeds with probability $p \ll 1$. A natural baseline is the batch mean: $b = \frac{1}{N} \sum_i R(\tau^{(i)})$.

When $Np \ll 1$, most batches contain zero successes. Then $R(\tau^{(i)}) = 0$ for all i, $b = 0$, and every advantage is zero:

$$R(\tau^{(i)}) - b = 0 - 0 = 0 \quad \forall, i$$

The gradient is exactly zero. The agent learns nothing. Even with advantage estimation, no variance in rewards means no signal. This, of course, changes if you have very good value function estimators at the risk of bias. Once the agent (by chance or by design) discovers a few successful trajectories, it can start reinforcing those behaviors. A slightly better policy produces more successful trajectories, which produce a stronger gradient signal, which yields an even better policy. That positive feedback loop is the *rapid improvement* phase.

Eventually, the easy gains are exhausted and the agent needs increasingly specific behaviors to improve further — the curve flattens to the asymptote.

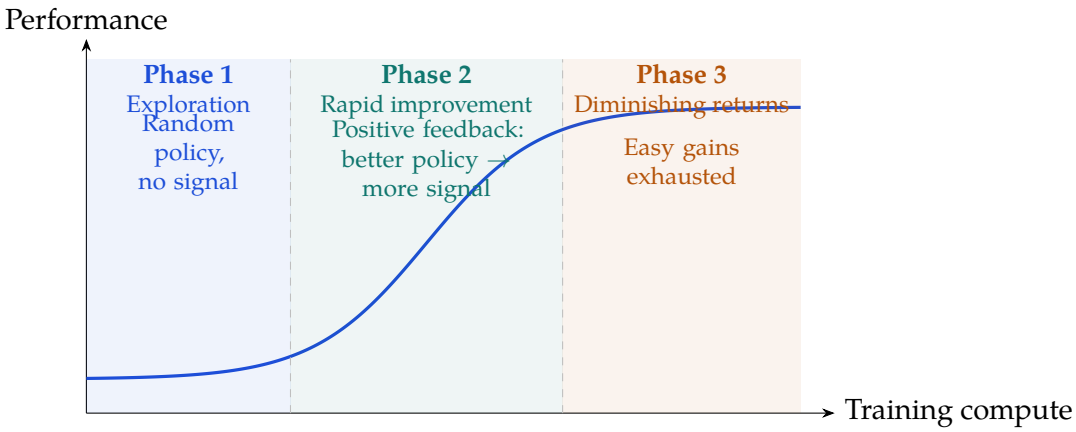

Figure 3: The three phases of RL training correspond to the sigmoid shape. Phase 1 is dominated by the exploration problem: the agent must discover successful trajectories before it can learn from them. Accelerating Phase 1 (e.g., via better exploration or a stronger initial policy from pretraining) shifts the inflection point earlier.

Implications for Frontiers

If a training batch has no variance in rewards, the information gain is 0 if you have Monte Carlo baselines and compute is wasted. Several recent systems explicitly account for this.

- GRPO (Group Relative Policy Optimization), used in DeepSeek-R1 (**?**), computes advantages by normalizing rewards within each group of sampled responses:

$$\hat{A}^{(i)} = \frac{R(\tau^{(i)}) - \mu_{\text{group}}}{\sigma_{\text{group}}}$$

  When all responses in a group receive the same reward, $\sigma_{\text{group}} = 0$ and the update is skipped entirely.

- OLMo model creators also do something like this. During RL training (using GRPO with Tulu 3), they filter out prompts where all sampled completions receive identical rewards, since these contribute zero gradient signal. This filtering can discard a substantial fraction of batches early in training, but improves compute efficiency by focusing updates on prompts where the model's behavior actually varies in quality.

- DAPO (Dynamic Sampling Policy Optimization) introduces dynamic sampling, which over-generates prompts and filters to keep only those where at least some completions succeed and some fail, ensuring every training batch contains meaningful signal. They also use token-level loss weighting to prevent longer responses from dominating the gradient.

Razin et al. (2025) show that good reward models induce more variance across batches (that is, more information signal).

Implications for Frontiers

The sigmoid scaling pattern suggests that accelerating the initial exploration phase could make RL training dramatically more compute-efficient. If better exploration shifts the inflection point earlier, then exploration methods are not just about final performance but also about sample efficiency. This also explains why SFT-then-RL pipelines work so well for LLMs. Supervised fine-tuning on diverse data implicitly solves much of the exploration problem by giving the initial policy a good starting point, pushing it past the flat phase of the sigmoid before RL training even begins. But this is unideal because we haven't even really tackled the exploration problem, not a good way to get toward an artificial general intelligence that can learn on its own.

2 Entropy Regularization

Before we get to dedicated exploration methods, we need to cover a mechanism that shows up everywhere in modern RL: **entropy regularization**. It is the simplest and most widely-used approach to maintaining some amount of exploration during policy optimization. And in theoretical convergence proofs of policy gradient methods, like PPO, it is often a necessary assumption that you have an entropy regularizer. Again, the standard RL objective maximizes expected cumulative reward:

$$\mathcal{J}(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right]. \tag{1}$$

In practice, algorithms like PPO and A2C add an **entropy bonus** to the policy loss to discourage premature collapse:

$$L(\theta) = L_{\text{policy}}(\theta) - \alpha\, \mathbb{E}_s\big[\mathcal{H}\big(\pi_\theta(\cdot \mid s)\big)\big], \tag{2}$$

where $\mathcal{H}(\pi(\cdot \mid s)) = -\sum_a \pi(a \mid s) \log \pi(a \mid s)$ is the entropy of the policy at state s, and $\alpha > 0$ is the **temperature** parameter controlling the exploration–exploitation tradeoff. The minus sign means we *subtract* entropy from the loss (equivalently, *add* entropy to the objective), so the optimizer is encouraged to keep the policy stochastic. Code snippet from stable baselines:

```
values, log_prob, entropy = self.policy.evaluate_actions(
    rollout_data.observations, actions)
...
# Entropy loss favor exploration
if entropy is None:
    # Approximate entropy when no analytical form
    entropy_loss = -th.mean(-log_prob)
else:
    entropy_loss = -th.mean(entropy)

entropy_losses.append(entropy_loss.item())

loss = policy_loss + self.ent_coef * entropy_loss + self.
    vf_coef * value_loss
```

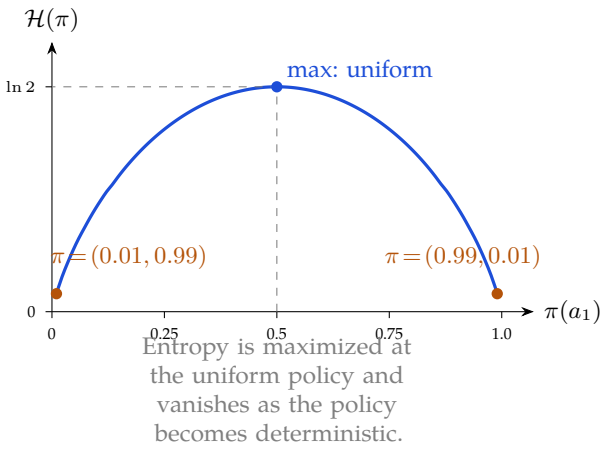

Figure 4: Entropy $\mathcal{H}(\pi) = -\pi(a_1)\log\pi(a_1) - \pi(a_2)\log\pi(a_2)$ for a 2-action policy. The bonus is highest at the uniform distribution and drops to zero at the corners. The entropy regularizer in Eq. (2) pushes the policy away from these degenerate endpoints.

Discussion

Entropy regularization discourages the policy from collapsing to a single action. A policy that maintains diversity over actions gets a higher bonus. The temperature α controls how much the agent cares about exploration versus exploitation. This is how PPO and A2C handle exploration by default. Why is this useful from a theory perspective? Think about some of the assumptions we had to make about policy/value iteration?

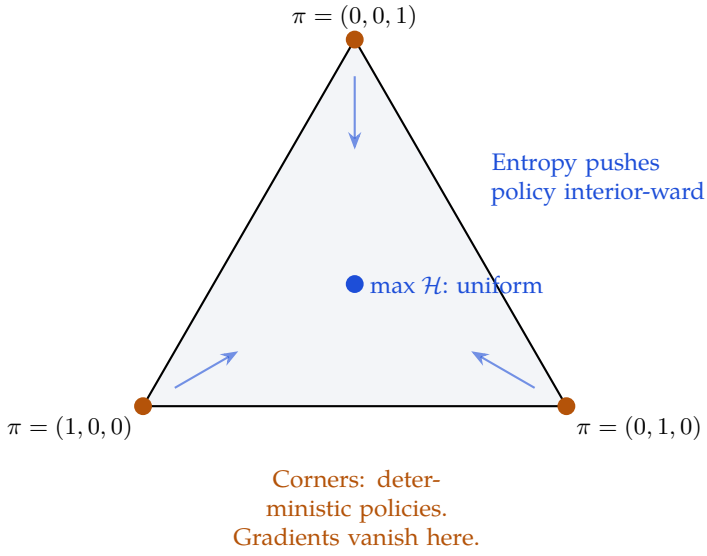

Figure 5: The probability simplex for a 3-action policy. Deterministic policies sit at the vertices (corners), where gradients degenerate. Entropy regularization pushes the policy toward the interior, maintaining well-conditioned gradients and ensuring continued exploration.

2.1 Why entropy regularization is needed: convergence of policy gradients

There is a good theoretical reasons to use entropy regularization, beyond the practical benefits. Without it, the policy gradient landscape has pathological properties near deterministic policies.

Why does entropy help so much? Because the optimal policy assigns non-zero probability to every action, the agent never stops collecting data about all state-action pairs. This ensures that value estimates continue to improve, which in turn allows the policy to improve. In the limit, this means you will eventually explore all actions in all states since everything has non-zero probability. Theoretical convergence guarantees, like Agarwal et al. (2021), often rely on this property, kind of like policy improvement operators we saw earlier in the class. There may also be additionally convenient properties for neural network training, such as convexity, curvature, etc. Ahmed et al. (2019) provide empirical confirmation, showing that entropy regularization smooths the loss landscape, helps escape poor local optima, and improves robustness to hyperparameter choices.

2.2 Connection to KL regularization

In LLM post-training, we typically see a KL penalty against a reference policy π_{ref}:

$$\mathcal{J}_{\text{KL}}(\theta) = L_{\text{policy}}(\theta) - \beta\, D_{\text{KL}}\big(\pi_\theta(\cdot \mid s_t) \| \pi_{\text{ref}}(\cdot \mid s_t)\big). \tag{3}$$

Discussion

Under what conditions does this help with exploration? Under what conditions does it hurt?

Hint: Agarwal et al. (2021) note: "our polynomial convergence rate using the KL regularizer [(relative entropy) $-\lambda\, \mathbb{E}_{s\sim\text{Unif}_\mathcal{S}}\Big[\text{KL}(\text{Unif}_\mathcal{A}, \pi_\theta(\cdot|s))\Big]$ crucially relies on the aggressive nature in which the relative entropy prevents small probabilities (the proof shows that any action, with a positive advantage, has a significant probability for any near-stationary policy of the regularized objective).

3 Exploration Methods for Deep RL

Entropy regularization helps, but it has a fundamental limitation. It is very inefficient and relies on improving exploration in the limit. The exploration problem has two faces. The first is: how can an agent discover high-reward strategies that require a temporally extended sequence of actions, each individually unrewarding? The second is the classic **exploration–exploitation tradeoff**: should the agent try new behaviors (to discover something better) or continue doing the best thing it knows to maximize return?

These are two views of the same problem. An agent that always exploits will never discover the distant rewards that require exploration. And exploration is costly: the agent has to sacrifice short-term reward for long-term information, which is precisely the exploration–exploitation tradeoff.

Why is this hard? Consider Montezuma's Revenge, an Atari 2600 game that has become the canonical benchmark for hard exploration. The player controls an explorer navigating a series of rooms in a pyramid, collecting keys to open doors while avoiding skulls, snakes, and other enemies. The game gives reward only when the agent picks up a key or opens a door. But reaching the first key requires descending a ladder, jumping across a moving platform, climbing another ladder, and precisely timing a jump over an enemy. This is roughly 100 correct actions in sequence, with zero reward along the way.

With random exploration (ϵ-greedy), the probability of stumbling on the key is astronomically small. DQN, which relies on ϵ-greedy exploration, scores can stay near zero on this game even after billions of frames of training. The agent needs a structured exploration strategy.

Should efficient exploration be possible? It depends on the structure of the problem:

- **No structure $\rightarrow$ no shortcuts.** A combination lock with three dials: testing one code tells you nothing about others. Brute force is optimal.
- In a chain environment, a random walk takes $O(|\mathcal{S}|^2)$ steps to reach the end, but a directed strategy needs only $O(|\mathcal{S}|)$.

- Wordle (or Mastermind) has $6^4 = 1296$ possible codes, yet good players solve it in five guesses or fewer (Knuth, 1976): each guess eliminates large swaths of the code space simultaneously.
- When you lose your keys, you don't search 10^8 locations uniformly. You have priors about where you last had them and prioritize them.

Exploration can be more efficient when **one observation provides information about many states**. This is the information-theoretic view, and it will guide the principled methods we develop in Lecture 7.

3.1 How existing algorithms explore

Every algorithm we have studied so far has *some* mechanism for exploration, even if it was not the focus:

Algorithm	Exploration mechanism
DQN	ϵ-greedy (random action w.p. ϵ)
REINFORCE / PPO	Stochastic policy (sample from π_θ) and entropy reg
DDPG / TD3	Gaussian action noise

All of these inject noise at the *action level*, independently at each time step. This produces **temporally uncorrelated** exploration: the agent independently decides whether to explore or exploit at every step, producing trajectories that resemble a random walk.

3.2 Parameter space noise

Plappert et al. (2017) and Fortunato et al. (2017) independently proposed a simple alternative: instead of adding noise to actions, add noise to the *policy parameters*:

$$\text{Collect data using } \pi_{\theta+\epsilon}(\cdot \mid s), \quad \epsilon \sim \mathcal{N}(0, \sigma^2 I). \tag{4}$$

The idea is that a perturbed parameter vector defines an *exploration* policy that is slightly different from the current one. When you roll out this perturbed policy for an entire episode, the exploration is temporally correlated, the agent commits to a consistent (but different) strategy.

For DDPG. Simply use $\pi_{\theta+\epsilon}(s)$ instead of $\pi_\theta(s) + \epsilon$.

For DQN. Act greedily with respect to a noisy Q-network: $\arg\max_a Q_{\theta+\epsilon}(s, a)$.

For policy gradient. We can optimize the expected return under random perturbations:

$$\max_\theta \ \mathbb{E}_{\epsilon \sim \mathcal{N}(0,\sigma^2 I)} \left[\mathbb{E}_{\pi_{\theta+\epsilon}} [R(\tau)] \right]. \tag{5}$$

Taking the gradient:

$$\nabla_\theta = \mathbb{E}_\epsilon \left[\mathbb{E}_{\pi_{\theta+\epsilon}} [R(\tau) \nabla_\theta \log \pi_{\theta+\epsilon}(\tau)] \right]. \tag{6}$$

There are two connections that might be worth keeping in mind here: (1) adding noise in parameter space is also often a way to get better generalization (see, e.g., sharpness-aware minimization, though different in respects as it is more targeted); (2) you can also consider the connection to evolutionary methods where you do this to create new evolved policies.

Parameter space noise has several open questions: (1) How much noise to add? Noise in parameter space doesn't correspond to any particular amount of noise in action space. (2) At convergence,

the perturbed policy still explores, even when it shouldn't. (3) The "drive" for exploration is still heuristic and somewhat random.

Break — 10 minutes

3.3 Intrinsic motivation and exploration bonuses

A broad class of exploration methods modify the reward function:

$$r(s, a) = r_{\text{task}}(s, a) + r_{\text{intrinsic}}(s, a). \tag{7}$$

The intrinsic reward encourages the agent to visit novel or informative states. Different choices for $r_{\text{intrinsic}}$ give different methods (Figure 6):

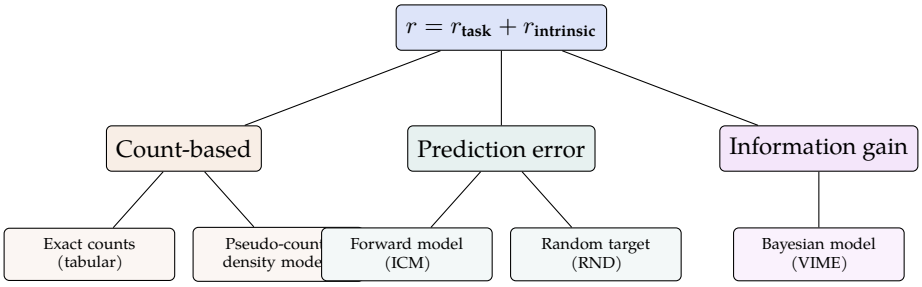

Figure 6: Taxonomy of intrinsic motivation methods. All modify the reward as $r = r_{\text{task}} + r_{\text{intrinsic}}$, but differ in how $r_{\text{intrinsic}}$ is computed. Count-based methods reward novelty via visit frequency; prediction-error methods reward surprise; information-gain methods reward learning.

Count-based exploration. The simplest idea is to reward the agent for visiting states it has not seen often. If $N(s)$ counts how many times state s has been visited, we can add a bonus that decays with the count:

$$r_{\text{intrinsic}}(s) = \frac{\beta}{\sqrt{N(s)}}, \tag{8}$$

where $\beta > 0$ controls the bonus magnitude. This is the MBIE-EB bonus (Strehl and Littman, 2008). The intuition is the same principle we will formalize shortly in the multi-armed bandit setting (Section 4.4): states that have been visited less are more uncertain, and we should be optimistic about uncertain states. The $1/\sqrt{N}$ decay mirrors the rate at which confidence intervals shrink with more samples.

In tabular settings, maintaining visit counts is straightforward. In deep RL, however, we never see the exact same state twice — pixel observations are unique — so we need **pseudo-counts**. Bellemare et al. (2016) proposed using a learned density model $\rho(s)$ to define them. This can also sometimes

be done by measuring the cosine similarity of state features if you have featurized states: [TODO: there's like some linear feature exploration thing that creates pseudo counts under this].

Discussion

As an aside, if you know about survey sampling or econometrics, you might start to make a connection to propensity weighting and making sure that propensity weights are non-zero. Optional discussion for later: can you make the connection in more depth? How can you get estimation guarantees for the reward/value leveraging ideas from econometrics? How can you improve sampling and measurement using ideas from RL if you consider potential externalities of measurements?

Prediction error as curiosity. Another approach uses the error of a predictive model as the intrinsic reward (Pathak et al., 2017; Stadie et al., 2015):

$$r_{\text{intrinsic}}(s, a, s') = \|f_\phi(s, a) - s'\|^2, \tag{9}$$

where f_ϕ is a forward dynamics model. States that are hard to predict are "surprising" and therefore worth exploring. Pathak et al. (2017) refine this by using an *inverse* dynamics model to learn a feature space, and then measuring prediction error in that space — this avoids being distracted by aspects of the observation that are unpredictable but irrelevant (e.g., random noise in the background).

This is also somewhat related to measuring information gain(Houthooft et al., 2016):

$$r_{\text{intrinsic}}(s, a, s') = D_{\text{KL}}\big(p(\phi \mid \mathcal{D} \cup \{(s, a, s')\}) \parallel p(\phi \mid \mathcal{D})\big), \tag{10}$$

where ϕ are the parameters of a dynamics model and $\mathcal{D}$ is the data collected so far. This reward is high when the new transition is maximally informative about the world model.

Random network distillation (RND). Burda et al. (2018) propose an alternative: fix a random neural network f_{θ_1} and train another network f_{θ_2} to match its outputs (Figure 7). The intrinsic reward is:

$$r_{\text{intrinsic}}(s) = \|f_{\theta_1}(s) - f_{\theta_2}(s)\|^2. \tag{11}$$

For states seen many times, f_{θ_2} will have learned to match f_{θ_1}, so the bonus is small. For novel states, the mismatch is large.

3.4 Entropy as a reward bonus: Maximum Entropy RL

All of the exploration bonuses above modify the reward: $r \to r + r_{\text{intrinsic}}$. We can do the same thing with entropy. Instead of adding entropy to the *loss* (as PPO does, Section **??**), we add it to the *reward at every timestep*:

$$\mathcal{J}_{\text{MaxEnt}}(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{\infty} \gamma^t \Big(r_t + \alpha \mathcal{H}(\pi_\theta(\cdot \mid s_t))\Big)\right]. \tag{12}$$

This is the **maximum entropy RL** objective. It looks similar to the PPO entropy bonus, but there is a crucial difference:

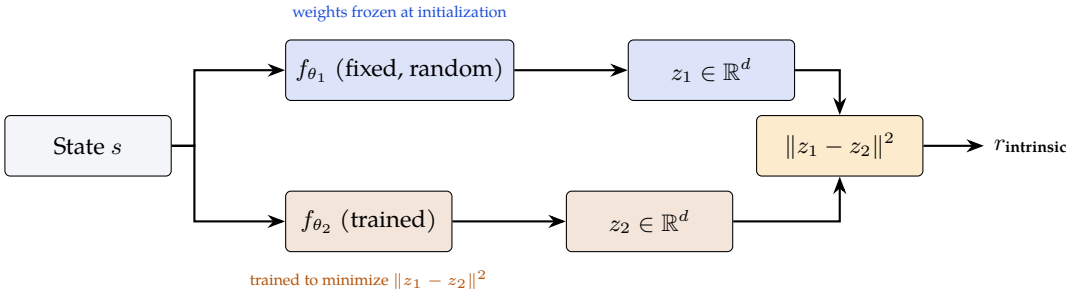

Figure 7: Random Network Distillation (RND) architecture. A fixed random network and a trainable predictor network both take the state as input. The intrinsic reward is the prediction error $\|f_{\theta_1}(s) - f_{\theta_2}(s)\|^2$. For frequently visited states, the predictor learns to match the target, so the bonus vanishes. For novel states, the mismatch is large.

Pitfall

Entropy inside the return is not the same as entropy added to the loss:

$$\underbrace{\mathbb{E}_\pi\left[\sum_t \gamma^t \Big(r(s_t, a_t) + \alpha\, \mathcal{H}\big(\pi(\cdot \mid s_t)\big)\Big)\right]}_{\text{MaxEnt RL (SAC): entropy at } \textit{every} \text{ future step}} \neq \underbrace{\mathbb{E}_\pi\left[\sum_t \gamma^t r(s_t, a_t)\right] + \alpha\, \mathbb{E}_s\big[\mathcal{H}\big(\pi(\cdot \mid s)\big)\big]}_{\text{entropy as loss regularizer (PPO, A2C)}}. \tag{13}$$

On the left, entropy appears *inside* the discounted return at every future timestep. The agent plans ahead to reach states where it can maintain high entropy, and this changes the Bellman equation. On the right, entropy is a regularizer on the current policy. It discourages the policy from collapsing but does not change value estimation: the critic still predicts standard returns $\sum_t \gamma^t r_t$. PPO uses the right. SAC uses the left.

Because entropy is inside the return, value estimation changes. Define the **soft Q-function** and **soft value function**:

$$Q^\pi_{\text{soft}}(s, a) = r(s, a) + \gamma \mathbb{E}_{s'}\big[V^\pi_{\text{soft}}(s')\big] \tag{14}$$

The value of a state equals the expected Q-value plus an entropy bonus.

The optimal policy for a given Q^*_{soft} is the **Boltzmann policy**:

$$\pi^*(a \mid s) = \frac{\exp\big(Q^*_{\text{soft}}(s, a)/\alpha\big)}{\sum_{a'} \exp\big(Q^*_{\text{soft}}(s, a')/\alpha\big)} = \text{softmax}_a\Big(\frac{Q^*_{\text{soft}}(s, \cdot)}{\alpha}\Big). \tag{15}$$

How do we get that? To find this optimal actor, we have to maximize expected value with the entropy bonus while maintaining the constraint that $\pi(a \mid s)$ represent a valid probability distribution. This entails ensuring that the probabilities sum to one and that they are non-negative. As the log function isn't defined for negative probabilities, it will be sufficient to enforce the constraint that the probabilities sum to one. Let's fix a state s, so we get the following constrained optimization problem:

$$\max_\pi \sum_a \pi(a \mid s)(Q(s, a) - \alpha \log \pi(a \mid s)) \qquad \text{s.t.} \sum_a \pi(a \mid s) = 1. \tag{16}$$

Note that this is because:

$$\max_{\pi} \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty}\gamma^t\left(r(s_t,a_t)+\alpha\mathcal{H}_\pi[a_t\mid s_t]\right)\right] = \max_{\pi} \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty}\gamma^t\left(r(s_t,a_t)+\alpha\left(-\mathbb{E}_{a_t\sim\pi(\cdot|s_t)}[\log\pi(a_t\mid s_t)]\right)\right)\right] \tag{17}$$

$$= \max_{\pi} \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty}\gamma^t\left(r(s_t,a_t)-\alpha\,\mathbb{E}_{a_t\sim\pi(\cdot|s_t)}[\log\pi(a_t\mid s_t)]\right)\right] \tag{18}$$

$$= \max_{\pi} \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty}\gamma^t\left(r(s_t,a_t)-\alpha\log\pi(a_t\mid s_t)\right)\right]. \tag{19}$$

You should think about $\pi(\cdot\mid s)$ as a big vector, where each coordinate contains the probability for a different action. We can solve this constrained optimization method using the method of Lagrange multiplies (which goes by many names, including the Euler Lagrange method; it is a special case of using the KKT conditions). We start by writing down the *relaxed* optimization problem, moving the hard constraint into the objective with a certain penalty term:

$$\mathcal{L}(\pi) = \sum_a \pi(a\mid s)(Q(s,a)-\alpha\log\pi(a\mid s)) + \lambda\left(\sum_a \pi(a\mid s)-1\right). \tag{20}$$

We now take the derivative of this term w.r.t. $\pi(a\mid s)$ (i.e., the a^{th} coordinate of that big long vector):

$$\frac{d}{d\pi(a\mid s)} = \pi(a\mid s)\left(\frac{-\alpha}{\pi(a\mid s)}\right) + Q(s,a) - \alpha\log\pi(a\mid s) + \lambda \tag{21}$$

Setting the derivative equal to zero we get:

$$-\alpha + Q(s,a) - \alpha\log\pi(a\mid s) + \lambda = 0 \tag{22}$$

$$\alpha\log\pi(a\mid s) = Q(s,a) + \lambda - \alpha \tag{23}$$

$$\pi(a\mid s) = e^{\frac{1}{\alpha}Q(s,a)}e^{\frac{\lambda}{\alpha}-1}. \tag{24}$$

Finally, we solve for the value of λ that makes $\sum_a \pi(a\mid s)=1$, we get

$$\pi(a\mid s) = \frac{e^{\frac{1}{\alpha}Q(s,a)}}{\sum_{a'} e^{\frac{1}{\alpha}Q(s,a')}} = \textsc{SoftMax}(\tfrac{1}{\alpha}Q(s,\cdot)). \tag{25}$$

The softmax is why this is called "soft" actor critic.

Discussion

Is the amount of exploration the same in all states?

3.5 Soft Actor-Critic (SAC)

Soft Actor-Critic (SAC) (Haarnoja et al., 2018) puts the MaxEnt framework into practice as a full algorithm. It is an off-policy actor-critic and one of the most widely used methods for continuous control. SAC maintains three networks:

- **Two Q-networks** Q_{ϕ_1}, Q_{ϕ_2} (as in TD3, to mitigate overestimation), trained via the soft Bellman equation:

$$\mathcal{L}(\phi_i) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}}\left[\left(Q_{\phi_i}(s,a) - \left(r + \gamma\, V_{\bar{\phi}}(s')\right)\right)^2\right], \tag{26}$$

  where $V_{\bar{\phi}}(s') = \mathbb{E}_{a'\sim\pi_\theta}\left[\min_{i=1,2} Q_{\bar{\phi}_i}(s',a') - \alpha \log \pi_\theta(a' \mid s')\right]$ uses target networks $\bar{\phi}$.

- **A policy network** π_θ, trained to maximize the soft value:

$$\mathcal{J}_\pi(\theta) = \mathbb{E}_{s\sim\mathcal{D}}\left[\mathbb{E}_{a\sim\pi_\theta}\left[\min_{i=1,2} Q_{\phi_i}(s,a) - \alpha \log \pi_\theta(a \mid s)\right]\right]. \tag{27}$$

SAC inherits the off-policy replay buffer from DDPG/TD3, uses the clipped double-Q trick from TD3 to combat overestimation, and adds entropy regularization via the $-\alpha \log \pi$ term. Compared to TD3, SAC learns a *stochastic* policy, which provides built-in exploration without needing the explicit action noise of DDPG/TD3.

3.6 Choosing an exploration method

When selecting an exploration method for a particular problem, four considerations matter:

- **Temporal correlation.** Does the method produce coherent exploration strategies, or just independent random jitter at each step?
- **Decay.** Does exploration naturally decrease as the agent learns? Count-based bonuses decay as $1/\sqrt{N}$ with experience. Parameter noise does not adapt.
- **Scalability.** Pseudo-counts and RND scale to high-dimensional observations more easily than exact density models.
- **Is there other structure and correlations we can exploit?** May want to consider other approaches that we'll talk more about next week.

Implications for Frontiers

In LLM RL training, exploration is implicit: the language model's pretraining provides a strong prior that makes most prompts "reachable" from the initial policy. But for truly novel reasoning — e.g., discovering new mathematical proofs or coding strategies — the exploration problem resurfaces. How to do principled exploration in language space is an open and exciting question!

Break — 20 minutes

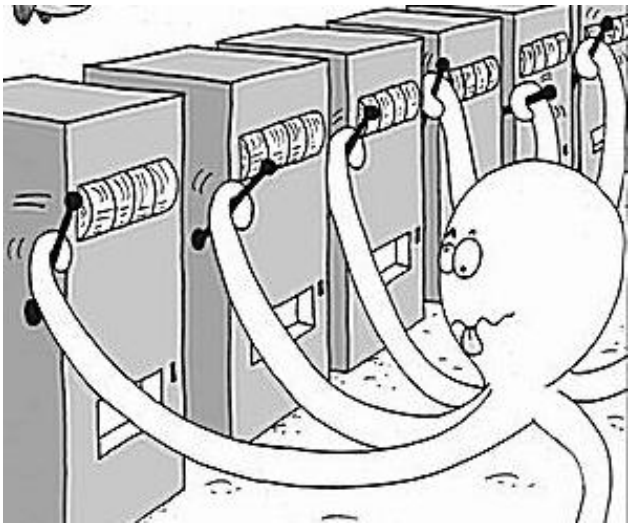

Figure 8: Source: https://blogs.mathworks.com/loren/2016/10/10/multi-armed-bandit-problem-and-exploration-vs-exploitation-trade-off/

We have now surveyed the practical landscape of exploration in deep RL: entropy regularization for local exploration, parameter-space noise for temporally coherent exploration, and intrinsic motivation for novelty-seeking. In the second half of this lecture, we develop the theoretical foundations that underpin these methods, starting with the simplest setting where exploration can be studied rigorously: the multi-armed bandit.

4 Multi-Armed Bandits

We now shift gears to study exploration in its simplest and most theoretically tractable setting: the **multi-armed bandit**. Bandits are the "drosophila of exploration problems." They are simple enough to analyze rigorously and wrap our heads around, but rich enough to capture the core exploration–exploitation tradeoff.

4.1 The bandit problem

Definition 1 (Multi-Armed Bandit)**.** A K-armed bandit is defined by K probability distributions $\nu_1, \ldots, \nu_K$ with means $\mu_1, \ldots, \mu_K$. At each round $t = 1, 2, \ldots, T$:

1. The learner selects an arm $a_t \in \{1, \ldots, K\}$.
2. The learner observes reward $r_t \sim \nu_{a_t}$.

The learner's goal is to maximize cumulative reward $\sum_{t=1}^{T} r_t$.

Notice: this is a 1-step, stateless RL problem. There is no state, no transitions, no temporal credit assignment. The only challenge is exploration vs. exploitation: should you play the arm that looks

best so far, or try a less-explored arm in hopes of finding something better? More formally, a bandit is an MDP with $|\mathcal{S}| = 1$ and horizon $H = 1$. Contextual bandits add state (but still $H = 1$). Full MDPs add sequential structure. Understanding bandits gives us a foundation for exploration in the more complex settings.

4.2 Regret

The standard performance metric for bandits is **regret**: how much worse is the learner compared to always playing the best arm?

Definition 2 (Regret)**.** Let $\mu^* = \max_a \mu_a$ be the mean reward of the best arm. The **cumulative regret** after T rounds is:

$$\text{Regret}(T) = T\mu^* - \sum_{t=1}^{T} \mu_{a_t} = \sum_{t=1}^{T} (\mu^* - \mu_{a_t}). \tag{28}$$

Equivalently, $\text{Regret}(T) = \sum_{a=1}^{K} \Delta_a \cdot \mathbb{E}[N_a(T)]$, where $\Delta_a = \mu^* - \mu_a$ is the **suboptimality gap** of arm a and $N_a(T)$ is the number of times arm a is pulled in T rounds.

4.3 ϵ-greedy

The simplest exploration strategy:

$$a_t = \begin{cases} \arg\max_a \hat{\mu}_a & \text{with probability } 1 - \epsilon, \\ \text{Uniform}(\{1, \ldots, K\}) & \text{with probability } \epsilon, \end{cases} \tag{29}$$

where $\hat{\mu}_a = \frac{1}{N_a(t)} \sum_{s:a_s=a} r_s$ is the empirical mean reward.

4.4 Upper Confidence Bound (UCB)

The UCB algorithm (Auer et al., 2002) is based on the **optimism in the face of uncertainty** principle: when uncertain about an arm's value, assume it's as good as it could plausibly be.

Algorithm 2 UCB1 (Auer et al., 2002)

Play each arm once. For $t = K+1, K+2, \ldots, T$:

$$a_t = \arg\max_{a \in \{1,\ldots,K\}} \underbrace{\hat{\mu}_a(t)}_{\text{exploitation}} + \underbrace{\sqrt{\frac{2 \ln t}{N_a(t)}}}_{\text{exploration bonus}} . \tag{30}$$

The exploration bonus $\sqrt{2 \ln t / N_a(t)}$ is an *upper confidence bound* on the true mean, derived from Hoeffding's inequality. For rewards bounded in $[0, 1]$, Hoeffding's inequality states that:

$$\Pr(\hat{\mu}_a - \mu_a \geq \epsilon) \leq \exp\left(-2 N_a(t)\, \epsilon^2\right). \tag{31}$$

Setting the right-hand side equal to $1/t^4$ (a failure probability that is summable over time) and solving for ϵ:

$$\exp(-2N_a(t)\,\epsilon^2) = t^{-4} \quad \Longrightarrow \quad \epsilon = \sqrt{\frac{2\ln t}{N_a(t)}}. \tag{32}$$

By a union bound over the two-sided case, $\mu_a \in \left[\hat{\mu}_a \pm \sqrt{2\ln t / N_a(t)}\right]$ with probability at least $1 - 2/t^4$. In particular:

$$\mu_a \le \hat{\mu}_a + \sqrt{\frac{2\ln t}{N_a(t)}} \quad \text{with high probability.} \tag{33}$$

So the UCB algorithm selects the arm with the highest *plausible* mean. Arms that have not been tried much have wide confidence intervals and get selected; as an arm is pulled more, its interval shrinks. The $\ln t$ in the numerator ensures that even well-explored arms are occasionally re-checked (the bound grows slowly with time). Intuitively, UCB explores an arm until it is *confident* the arm is suboptimal.

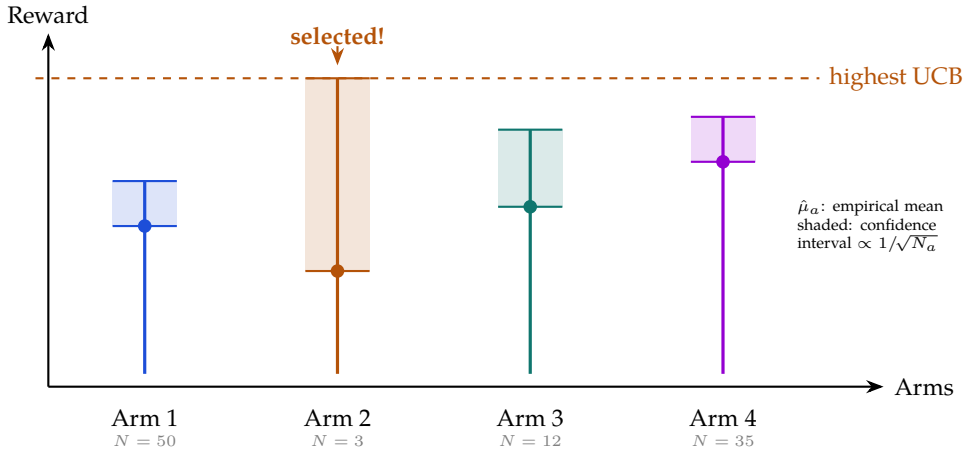

Figure 9: UCB selects the arm with the highest *upper confidence bound*, not the highest empirical mean. Arm 2 has a low empirical mean but wide confidence interval (pulled only 3 times), so its UCB is highest — it gets selected for exploration. Well-explored arms have narrow intervals.

4.5 Optimism in the face of uncertainty: the general principle

UCB instantiates a general principle: **optimism in the face of uncertainty**. Construct a confidence set $\mathcal{C}_a(t)$ for each arm's mean, and act greedily with respect to the most optimistic estimate:

$$a_t = \arg\max_a \max_{\mu \in \mathcal{C}_a(t)} \mu. \tag{34}$$

If the confidence sets are valid (contain the true means with high probability), then any arm selected is either truly good or has a wide confidence interval, meaning we have not tried it enough to know much about it. Either way, we learn something useful. Conversely, arms that are really suboptimal will eventually have narrow confidence intervals that exclude the best arm's mean, and they will stop being selected. The algorithm thus automatically balances exploration (trying

uncertain arms) and exploitation (sticking with the best known arm). Note, this can also extend to RL with exploration bonuses.

Pitfall

However, getting good confidence estimates can be challenging in more complex settings. And if you have bad estimates, things can collapse.

4.6 From bandits to RL

Why should we care about bandits when we're studying RL? Because the exploration strategies developed for bandits can be lifted to MDPs:

Bandit method	RL analogue
ϵ-greedy	ϵ-greedy in DQN
Softmax/Boltzmann	Entropy regularization / SAC
UCB (exploration bonus)	Count-based exploration / MBIE-EB
Thompson sampling	Bootstrap DQN (next lecture)
Information gain	VIME (Houthooft et al., 2016)

The main challenges in lifting these methods from bandits to full MDPs are threefold. First, **state space size**: in bandits the "state" space is trivial ($|\mathcal{S}| = 1$), but in MDPs the agent must manage uncertainty over a combinatorially large state space. Second, **temporal credit assignment**: in bandits, reward is immediate; in MDPs, an exploratory action now may only pay off many steps later, making it hard to attribute value to exploration. Third, **computational cost**: maintaining posterior distributions or confidence sets over value functions (rather than over K scalar means) is expensive, which is why the deep RL methods of Section 3 use approximations like pseudo-counts and prediction error.

Next lecture, we will develop Thompson sampling and information-theoretic approaches in more depth, expanding our toolkit.

References

Agarwal, A., Kakade, S. M., Lee, J. D., and Mahajan, G. (2021). On the theory of policy gradient methods: Optimality, approximation, and distribution shift. *Journal of Machine Learning Research*, 22(98):1–76.

Ahmed, Z., Le Roux, N., Norouzi, M., and Schuurmans, D. (2019). Understanding the impact of entropy on policy optimization. In *International Conference on Machine Learning (ICML)*, pages 151–160. PMLR.

Auer, P., Cesa-Bianchi, N., and Fischer, P. (2002). Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2):235–256.

Cover, T. M. and Thomas, J. A. (2006). *Elements of Information Theory*. Wiley-Interscience, 2nd edition.

Bellemare, M., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D., and Munos, R. (2016). Unifying count-based exploration and intrinsic motivation. In *Advances in Neural Information Processing Systems* (*NeurIPS*), volume 29.

Burda, Y., Edwards, H., Storkey, A., and Klimov, O. (2018). Exploration by random network distillation. *arXiv preprint arXiv:1810.12894*.

Chung, W., Thomas, V., Machado, M. C., and Le Roux, N. (2021). Beyond variance reduction: Understanding the true impact of baselines on policy optimization. In *International Conference on Machine Learning* (*ICML*), pages 1999–2009. PMLR.

Fortunato, M., Azar, M. G., Piot, B., Menick, J., Osband, I., Graves, A., Mnih, V., Munos, R., Hassabis, D., Pietquin, O., et al. (2017). Noisy networks for exploration. *arXiv preprint arXiv:1706.10295*.

Fu, J., Co-Reyes, J. D., and Levine, S. (2017). EX2: Exploration with exemplar models for deep reinforcement learning. In *Advances in Neural Information Processing Systems* (*NeurIPS*), volume 30.

Korbak, T., Perez, E., and Buckley, C. L. (2022). RL with KL penalties is better viewed as Bayesian inference. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 1083–1091.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. (2018). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning* (*ICML*), pages 1861–1870. PMLR.

Houthooft, R., Chen, X., Duan, Y., Schulman, J., De Turck, F., and Abbeel, P. (2016). VIME: Variational information maximizing exploration. In *Advances in Neural Information Processing Systems* (*NeurIPS*), volume 29.

Knuth, D. E. (1976). The computer as master mind. *Journal of Recreational Mathematics*, 9(1):1–6.

Lai, T. L. and Robbins, H. (1985). Asymptotically efficient adaptive allocation rules. *Advances in Applied Mathematics*, 6(1):4–22.

Mei, J., Xiao, C., Szepesvari, C., and Schuurmans, D. (2020). On the global convergence rates of softmax policy gradient methods. In *International Conference on Machine Learning* (*ICML*), pages 6820–6829. PMLR.

Osband, I., Blundell, C., Pritzel, A., and Van Roy, B. (2016). Deep exploration via bootstrapped DQN. In *Advances in Neural Information Processing Systems* (*NeurIPS*), volume 29.

Pathak, D., Agrawal, P., Efros, A. A., and Darrell, T. (2017). Curiosity-driven exploration by self-supervised prediction. In *International Conference on Machine Learning* (*ICML*), pages 2778–2787. PMLR.

Plappert, M., Houthooft, R., Dhariwal, P., Sidor, S., Chen, R. Y., Chen, X., Asfour, T., Abbeel, P., and Andrychowicz, M. (2017). Parameter space noise for exploration. *arXiv preprint arXiv:1706.01905*.

Khatri, D., Madaan, L., Tiwari, R., Bansal, R., Duvvuri, S. S., Zaheer, M., Dhillon, I. S., Brandfonbrener, D., and Agarwal, R. (2025). The art of scaling reinforcement learning compute for LLMs. *arXiv preprint arXiv:2510.13786*.

Stadie, B. C., Levine, S., and Abbeel, P. (2015). Incentivizing exploration in reinforcement learning with deep predictive models. *arXiv preprint arXiv:1507.00814*.

Strehl, A. L. and Littman, M. L. (2008). An analysis of model-based interval estimation for Markov decision processes. *Journal of Computer and System Sciences*, 74(8):1309–1331.

Tang, H., Houthooft, R., Foote, D., Stooke, A., Chen, X., Duan, Y., Schulman, J., DeTurck, F., and Abbeel, P. (2017). #Exploration: A study of count-based exploration for deep reinforcement learning. In *Advances in Neural Information Processing Systems* (*NeurIPS*), volume 30.