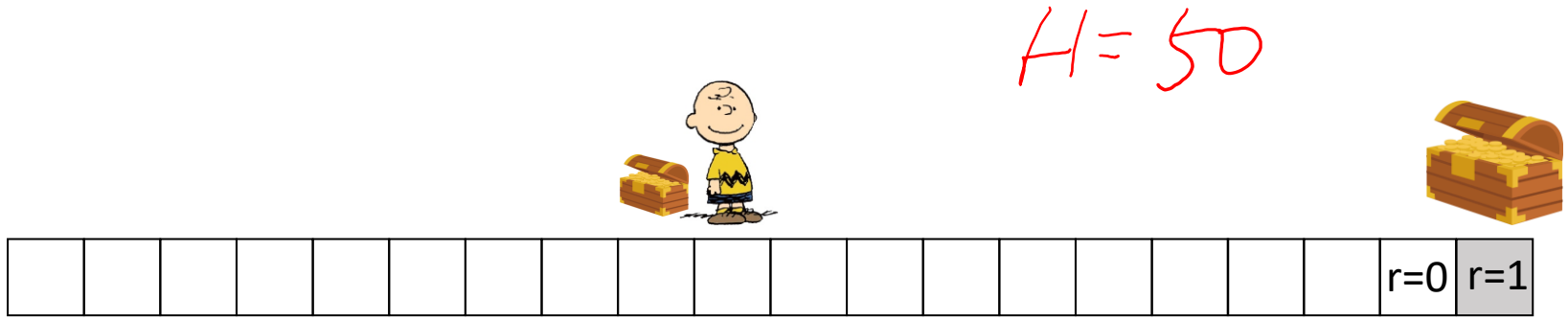
Exploration in MDPs

Chen-Yu Wei

State-Space Exploration in MDPs



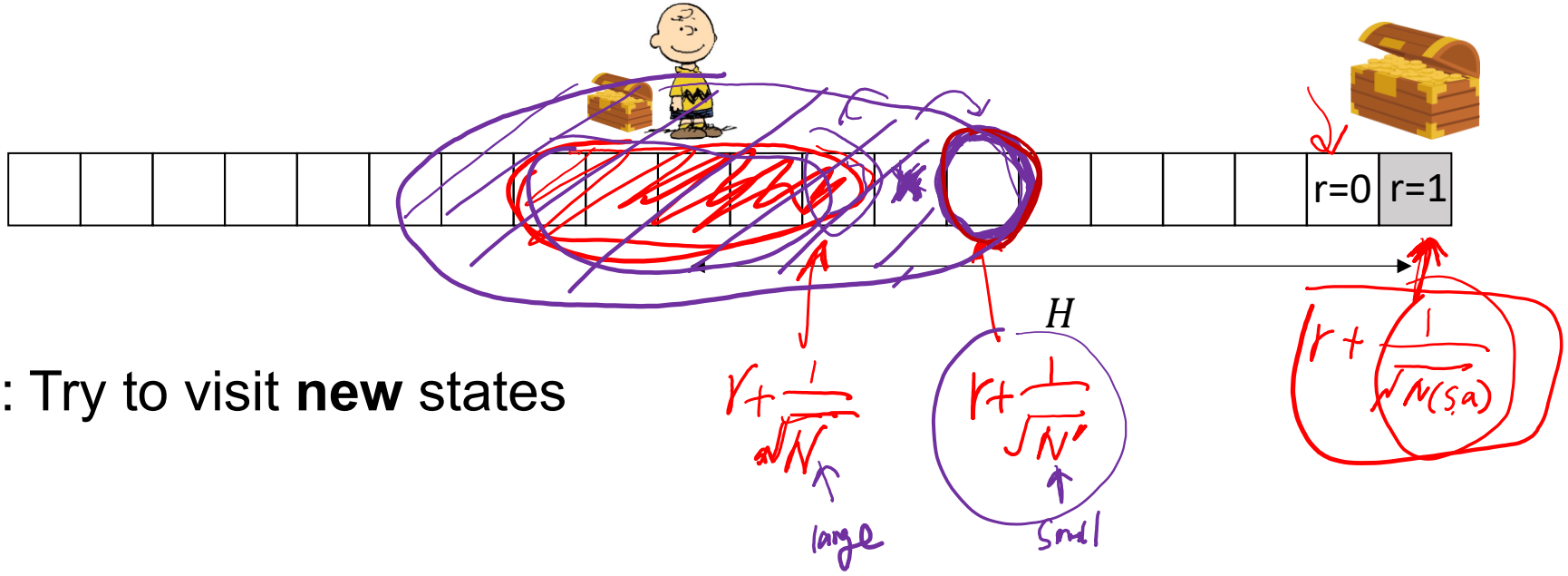
Environment:

- Fixed-horizon MDP with episode length H
- Initial state at 0
- A single rewarding state at state H
- Actions: Go LEFT or RIGHT

Randomly initialized DQN with ϵ-greedy requires **exponential (in H) many episodes** to see the reward.

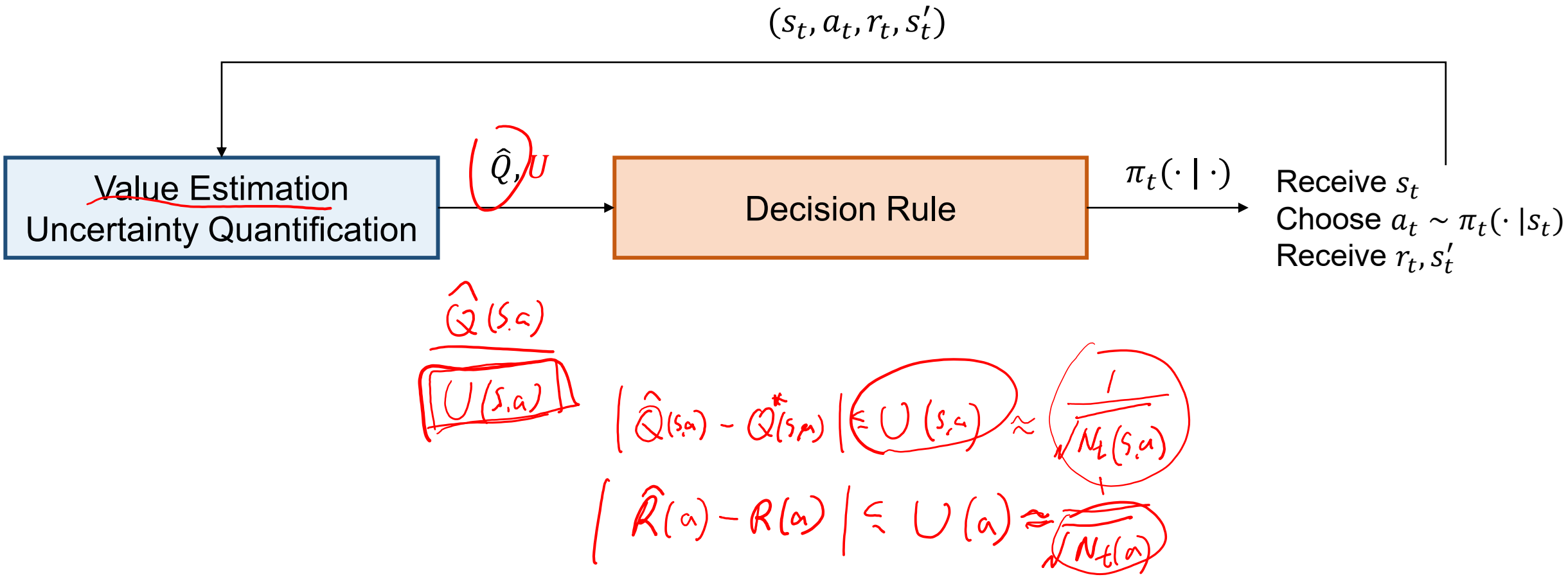
Insufficiency of algorithms we have discussed for MDPs

- Lack of **exploration over the state space** (we need **deep exploration**)
- This issue is particularly critical if
  - Local reward does not provide any information
  - Local reward provide misleading information



- Solution: Try to visit **new** states

Exploration via Uncertainty Quantification



Recall: Exploration Bonus for Bandits

- We have discussed this idea for action exploration – UCB.

Upper Confidence Bound

$$a_t = \text{argmax}_a \left[\hat{R}_t(a) + c \sqrt{\frac{1}{N_t(a)}} \right]$$

$\hat{R}_t(a)$ = the empirical mean of arm a up to time $t-1$.
$N_t(a)$ = the number of times we draw arm a up to time $t-1$.

Deep Q-Learning with UCB Bonus

Initialize $\mathcal{RB} = \{\}$

(this version is not yet practical)

For $k = 1,\ 2, \ldots$

For $i = 1, 2, \ldots, N$:

Choose action $a_i \sim \text{EG}(Q_{\theta_k}(s_i, \cdot))$ or BE

Receive reward $r_i \sim R(s_i, a_i)$ and $s_i' \sim P(\cdot\,|s_i, a_i)$

$s_{i+1} = s_i'$ if episode continues, $s_{i+1} \sim \rho$ if episode ends

Push (s_i, a_i, r_i, s_i') to $\mathcal{RB}$

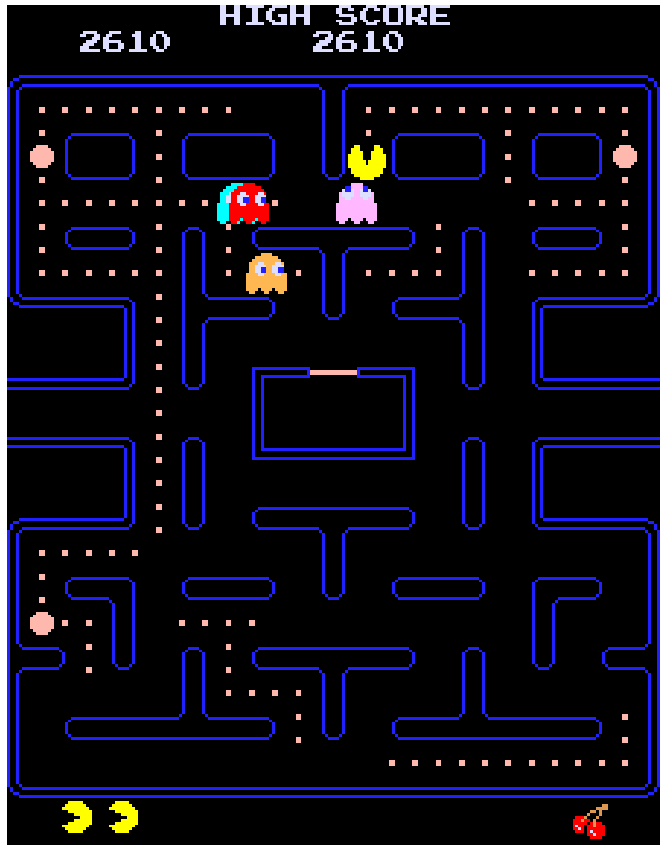
$Q_\theta(s,a) \approx r + c\sqrt{\frac{1}{N(s,a)}} + \gamma \max_{a'} Q_{\bar{\theta}}(s',a')$ (bonus)

For $m = 1, 2, \ldots, M$:

Randomly sample a batch B from $\mathcal{RB}$

$$\theta \leftarrow \theta - \alpha \frac{1}{|B|} \sum_{(s,a,r,s') \in B} \nabla_\theta \left(Q_\theta(s,a) - r - c\sqrt{\frac{1}{N_k(s,a)}} - \gamma \max_{a'} Q_{\bar{\theta}}(s',a') \right)^2$$

$$\bar{\theta} \leftarrow (1-\tau)\bar{\theta} + \tau\theta$$


HIGH SCORE
2610
2610

Common Approaches of Exploration

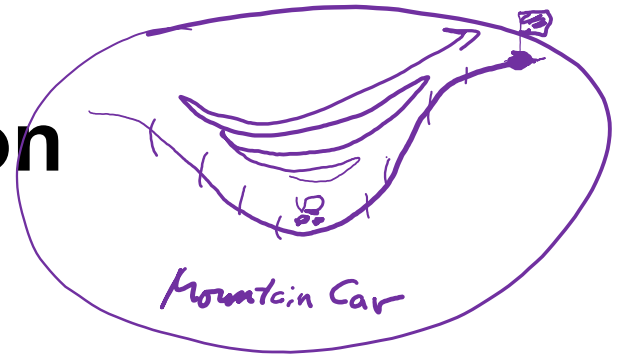
- Optimistic Exploration
  - Upper Confidence Bound
- Randomized Exploration
  - Thompson Sampling

$$+ \frac{1}{\sqrt{N(a)}}$$

$$+ \frac{1}{\sqrt{N(a)}} \mathcal{N}(0, 1)$$

Exploration in Large State Spaces

UCB / TS with State(-Action) Discretization



Partition the state-action space into a finite number of groups

Then instead of counting the #visits to individual state-action, we only count the #visits to each group

$g(s,a)$: the group (s,a) belongs to

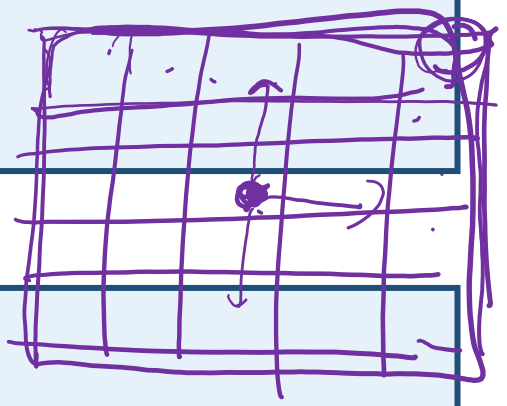
$N(g(s,a))$: how many times the learner has visited the group $g(s,a)$

$$Q(s,a) \leftarrow R(s,a) + c \cdot \frac{1}{\sqrt{N(g(s,a))}} + \sum_{s'} P(s'|s,a) \max_{a'} Q(s',a')$$

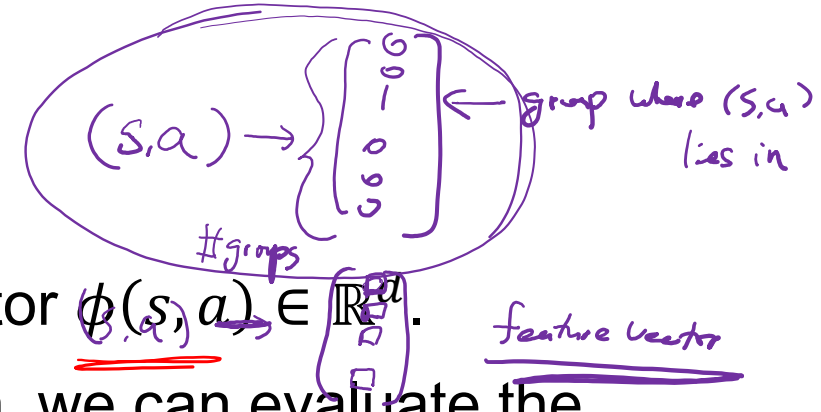
$$Q(s,a) \leftarrow R(s,a) + c \cdot \frac{\mathcal{N}(0,1)}{\sqrt{N(g(s,a))}} + \sum_{s'} P(s'|s,a) \max_{a'} Q(s',a')$$

$s \in \mathbb{R}^d$

Discretization will require 5^d



UCB / TS with State(-Action) Features



Suppose for any (s, a), we have access to a feature vector $\phi(s, a) \in \mathbb{R}^d$.

Then instead of counting the #visits to every state-action, we can evaluate the **novelty of the feature**.

$$\Lambda = \sum_{(s_i, a_i)\text{: past samples}} \phi(s_i, a_i)\phi(s_i, a_i)^\top \in \mathbb{R}^{d \times d}$$

$$\Lambda = \begin{bmatrix} N_t(g_1) & & 0 \\ & N_t(g_2) & \\ 0 & & \ddots \\ & & & N_t(g_m) \end{bmatrix}$$

$$Q(s, a) \triangleq R(s, a) + c \cdot \sqrt{\phi(s, a)\Lambda^{-1}\phi(s, a)} + \sum_{s'} P(s'|s, a) \max_{a'} Q(s', a')$$

Jin et al. Provably efficient reinforcement learning with linear function approximation. 2019.

$$Q(s, a) \triangleq R(s, a) + c \cdot \mathcal{N}(0, \phi(s, a)\Lambda^{-1}\phi(s, a)) + \sum_{s'} P(s'|s, a) \max_{a'} Q(s', a')$$

Zanette et al. Frequentist Regret Bounds for Randomized Least-Squares Value Iteration. 2019.

Ideas for Exploration

General Idea:

$$\tilde{R}(s,a) = R(s,a) + \text{bonus}(s,a)$$

where $\text{bonus}(s,a)$ quantifies the **uncertainty of the learner's estimation** for $\hat{Q}(s,a)$ (could be randomized)

$$\frac{1}{\sqrt{N_t(s,a)}}$$

After this modifications, just perform standard RL algorithm over $\tilde{R}$.

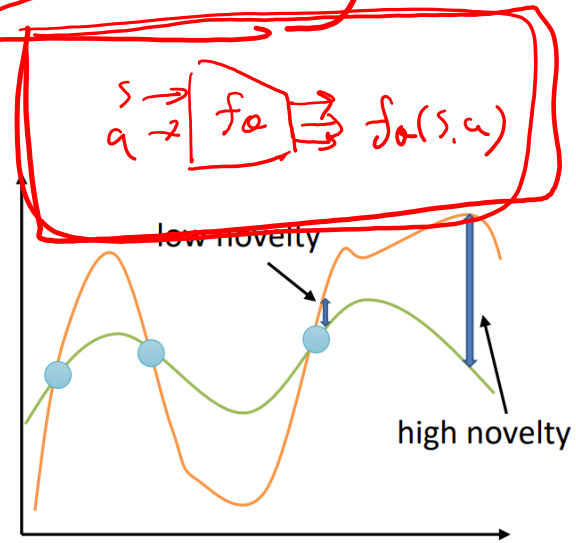
1. Bonus from Prediction Error

Random Network Distillation (RND)

Given a target function $f^\star(s,a)$ and buffer data $\mathcal{B} = \{(s_i, a_i)\}_{i=1}^n$

Minimize $\frac{1}{n}\sum_{i=1}^{n} \|f_\theta(s_i,a_i) - f^\star(s_i,a_i)\|^2$

Use $B(s,a) = \|f_\theta(s,a) - f^\star(s,a)\|^2$ as the bonus

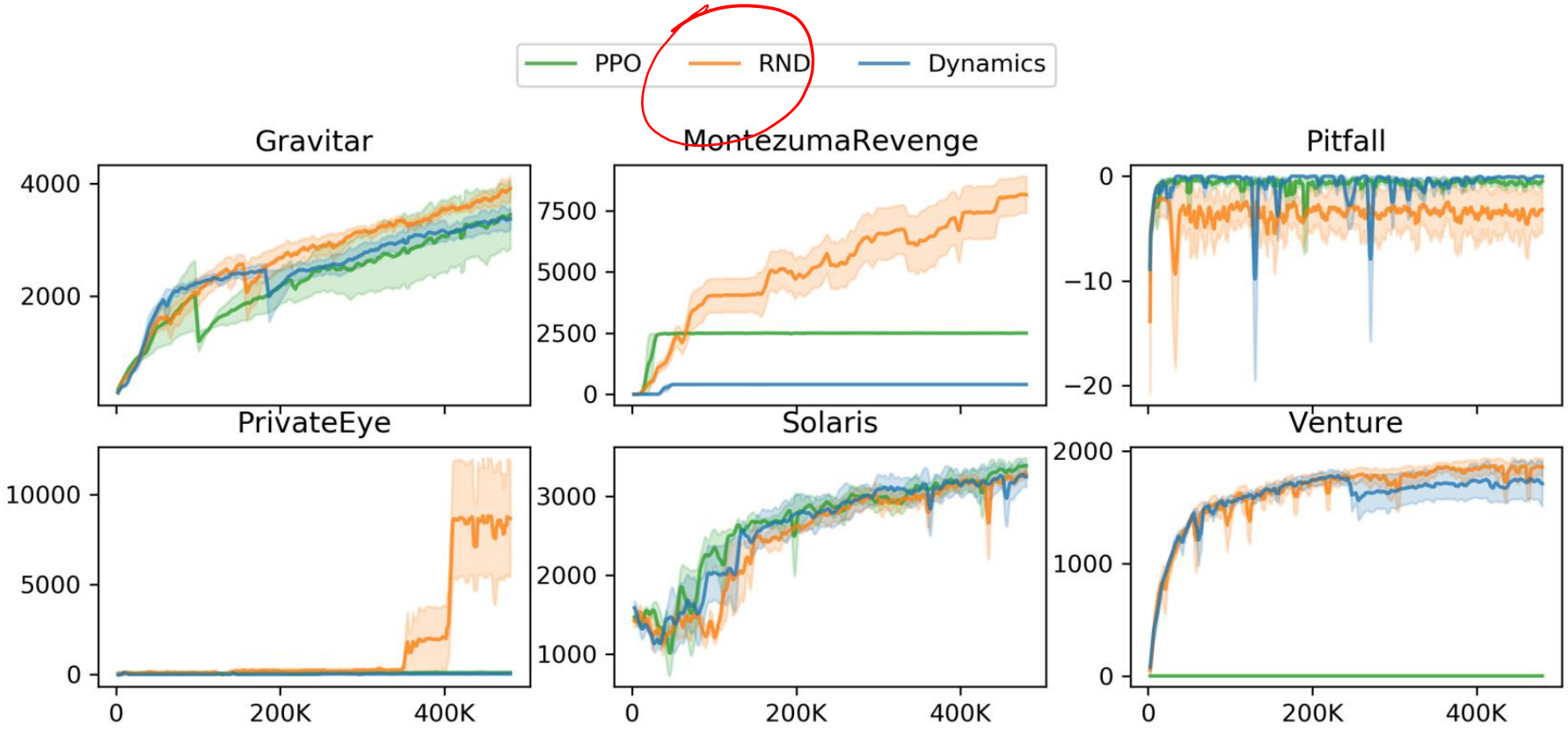


Ideally, we want $f^\star(s,a) \approx Q^\star(s,a) \in \mathbb{R}^{S\times A}$

But we can simply use a random network $f^\star(s,a) = f_\phi(s,a)$

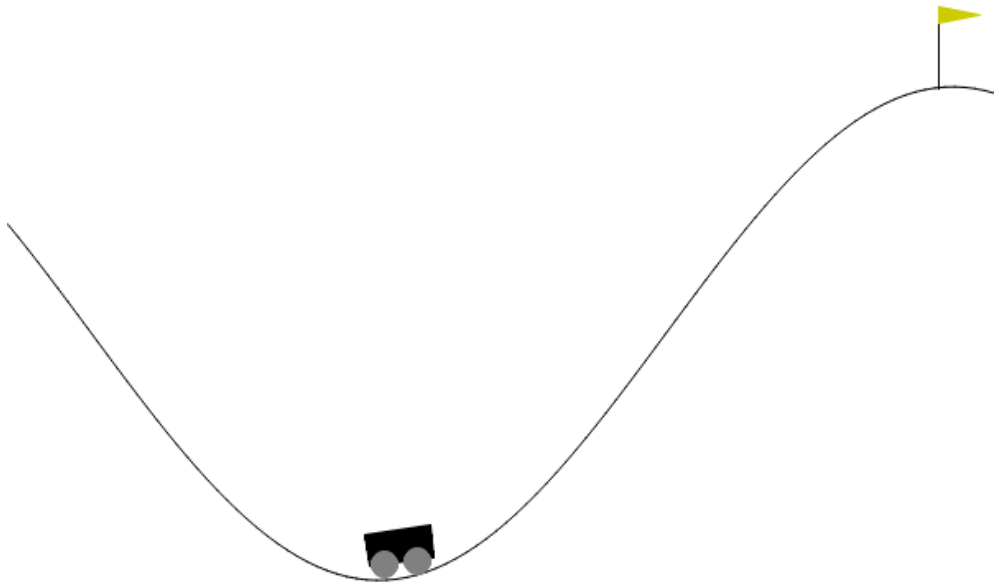
Burda et al. Exploration by Random Network Distillation. 2018.

Random Network Distillation

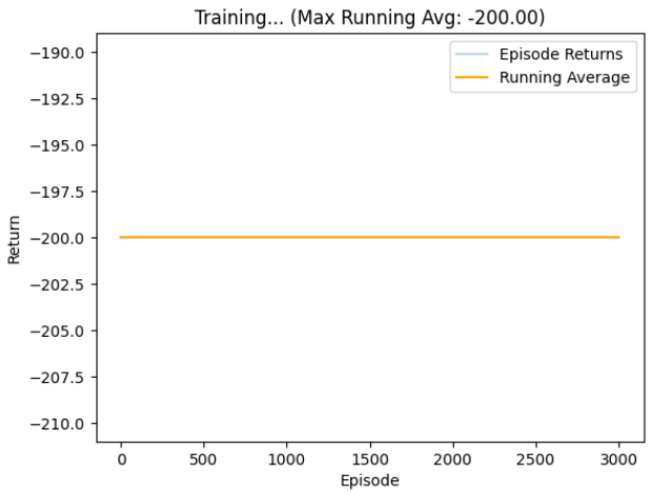


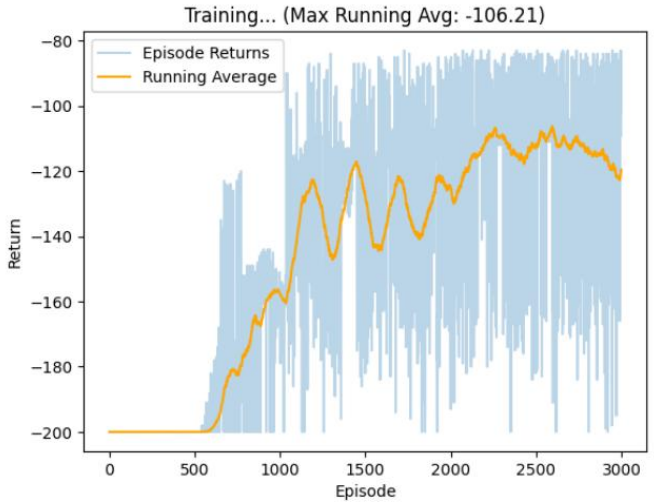
Burda et al. Exploration by Random Network Distillation. 2018.

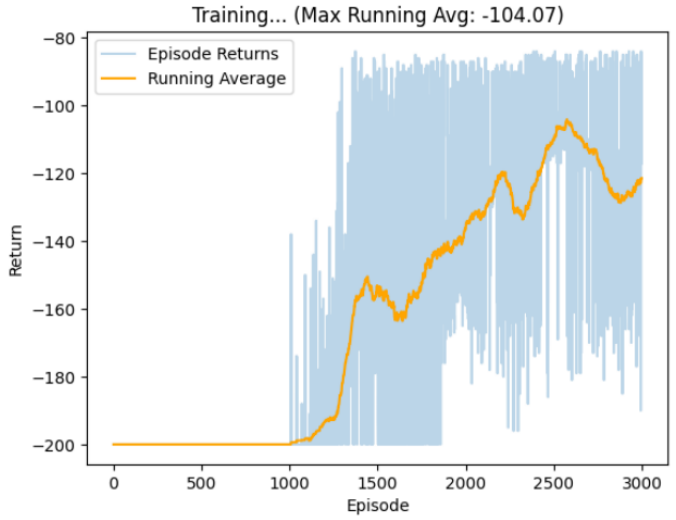
Exploration in Mountain Car

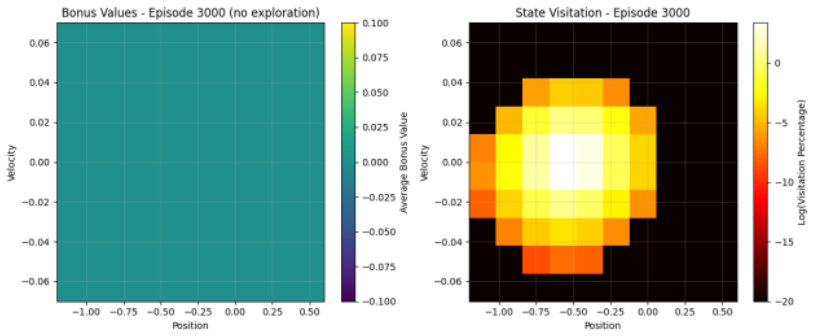


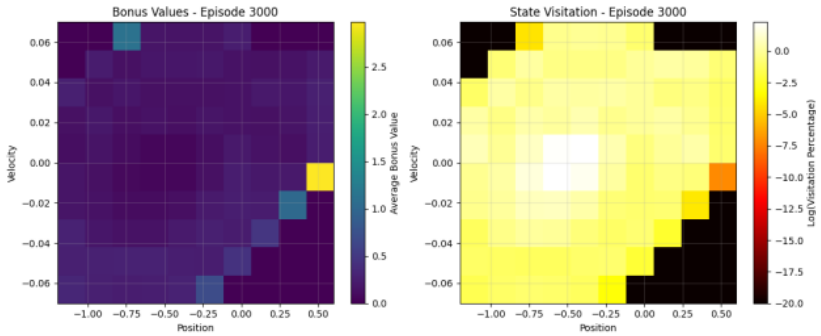
Action Space	`Discrete(3)`
Observation Space	`Box([-1.2 -0.07], [0.6 0.07], (2,), float32)`
import	`gymnasium.make("MountainCar-v0")`

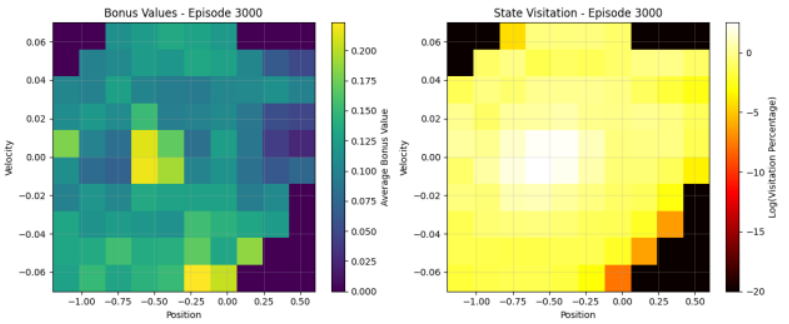












DQN + No bonus

DQN + Tabular Bonus

DQN + RND Bonus

2. Thompson Sampling

An Attempt to Adapt to Thompson Sampling

Initialize $\mathcal{RB} = \{\}$ (this is not the right algorithm)

For $k = 1,\ 2, \ldots$

For $i = 1, 2, \ldots, N$:

Choose action $a_i \sim \text{EG}(Q_{\theta_k}(s_i, \cdot))$ or BE

Receive reward $r_i \sim R(s_i, a_i)$ and $s_i' \sim P(\cdot\,|s_i, a_i)$

$s_{i+1} = s_i'$ if episode continues, $s_{i+1} \sim \rho$ if episode ends

Push (s_i, a_i, r_i, s_i') to $\mathcal{RB}$

For $m = 1, 2, \ldots, M$:

Randomly sample a batch B from $\mathcal{RB}$

$$\theta \leftarrow \theta - \alpha \frac{1}{|B|} \sum_{(s,a,r,s') \in B} \nabla_\theta \left(Q_\theta(s,a) - r - c\sqrt{\frac{1}{N_k(s,a)}}\, n(s,a) - \gamma \max_{a'} Q_{\overline{\theta}}(s', a') \right)^2$$

$$\overline{\theta} \leftarrow (1-\tau)\overline{\theta} + \tau\theta$$

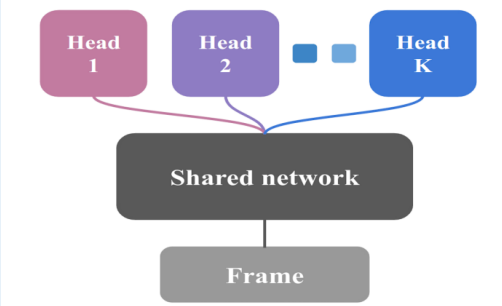
An Attempt to Adapt to Thompson Sampling

- What’s not correct in this algorithm?

Bootstrapped DQN

Osband et al. Deep Exploration via Bootstrapped DQN. 2016.
Osband et al. Randomized Prior Functions for Deep Reinforcement Learning. 2018.

Randomly initialize K instances of DQN $\theta_1, \dots, \theta_K$
(each θ_i has their own target network $\bar{\theta}_i$ and replay buffer $\mathcal{B}_i$).



(a) Shared network architecture

For each episode:

Randomly sample $i \sim \text{Unif}\{1,2,\dots,K\}$

Execute $\pi(s) = \max_a Q_{\theta_i}(s,a)$ in the whole episode.

Randomly place the obtained (s,a,r,s') in some/all replay buffers.

Update all DQN parameters.

Bootstrapped DQN

Osband et al. Deep Exploration via Bootstrapped DQN. 2016.
Osband et al. Randomized Prior Functions for Deep Reinforcement Learning. 2018.

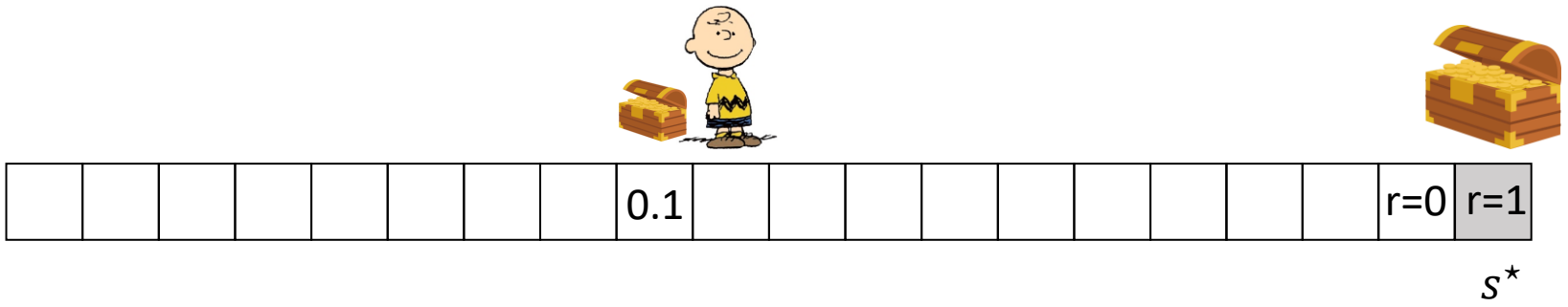
Some intuitions:

- The random initialization makes $Q_{\theta_1}(s,a), \ldots, Q_{\theta_K}(s,a)$ all very different.
  We can view them as associated with different initial noise $n_1(s,a)$.
- Over the course of training, for (s,a)'s that are more often visited, their effective magnitude of $n_t(s,a)$ decreases (because we train those DQNs without adding more noise).
- For (s,a)'s that are not often visited, their effective magnitude of $n_t(s,a)$ remains high.
- **Why does this perform deep exploration?** For a particular state s, if $\max_a Q_{\theta_i}(s,a)$ is initialized high but has not been visited many times before, the training of θ_i will propagate this high value to other state and encourage the learner to reach s from other states.

Bootstrapped DQN

Osband et al. Deep Exploration via Bootstrapped DQN. 2016.
Osband et al. Randomized Prior Functions for Deep Reinforcement Learning. 2018.

- In the toy example, as long as **one of the K DQNs** initializes $s^\star$ (or some states close to it) with a high value, then it can help the learner explore to $s^\star$.
- In this example, roughly we need $K = \mathrm{O}(\text{number of states})$ to achieve this effect.



Bootstrapped DQN

Osband et al. Deep Exploration via Bootstrapped DQN. 2016.
Osband et al. Randomized Prior Functions for Deep Reinforcement Learning. 2018.

"Deep Sea" Exploration

- **Stylized "chain" domain testing "deep exploration":**
  - *State = N x N grid, observations 1-hot.*
  - *Start in top left cell, fall one row each step.*
  - *Actions {0,1} map to left/right in each cell.*
  - ***"left" has reward = 0, "right" has reward = -0.1/N***
  - ***... but if you make it to bottom right you get +1.***
- Only one policy (out of more than 2^N) positive return.
- **ε-greedy / Boltzmann / policy gradient / are useless.**