Midterm Champ agent released.

- Midterm Champion agent released on Piazza.

- Try it out!

Chapter 9: Exploration in Deep RL

Recap: UVBVI

For $n = 1 \to N$:

1. Set $N^n(s,a) = \sum_{i=1}^{n-1} \sum_h \mathbf{1}\{(s_h^i, a_h^i) = (s,a)\}, \forall s, a$

2. Set $N^n(s,a,s') = \sum_{i=1}^{n-1} \sum_h \mathbf{1}\{(s_h^i, a_h^i, s_{h+1}^i) = (s,a,s')\}, \forall s, a, s'$

3. Estimate model: $\widehat{P}^n(s' \mid s,a) = \frac{N^n(s,a,s')}{N^n(s,a)}, \forall s, a, s'$

4. Plan: $\pi^n = VI\left(\widehat{P}^n, \{r_h + b_h^n\}_h\right)$, with $b_h^n(s,a) = cH\sqrt{\frac{\ln(SAHN/\delta)}{N^n(s,a)}}$

5. Execute π^n : $\{s_0^n, a_0^n, r_0^n, \ldots, s_{H-1}^n, a_{H-1}^n, r_{H-1}^n, s_H^n\}$

- Only work for finite S,A.
- Key insight: reward bonus helps with exploration.

Question Today:

How to perform exploration in deep RL?

Approach 1: Randomization

1. ϵ-greedy (discrete action space):
$$\pi(a|s) = (1-\epsilon)\pi_t(a|s) + \epsilon/|A|$$
2. Gaussian noise (continuous action space):
$$a = \pi(s) + N(0, \sigma I)$$
3. Entropy regularization (often in PG):
$$H(\pi) = \mathbb{E}_{s\sim\pi}[\pi(a|s)\log\pi(a|s)]$$
$$f(\pi) = J(\pi) - \alpha H(\pi)$$

- Work to some extend to find locally optimal policy.
- Provably fail in difficult settings, sparse reward, large MDP diameter, e.g. combination locks.

Approach 2: Pseudo-Counts

- Recall the UCB bonus

$$b_h^n(s,a) = cH\sqrt{\frac{\ln(SAHN/\delta)}{N^n(s,a)}}$$

- A simple generalization is the count-based bonus:
- Learning a pseudo-count function: $\widehat{N}^n(s,a) = n \cdot \rho^n(s,a)$, where $\rho^n(s,a)$ is a density function.
- Density estimation can be done in many ways:
  - Gaussian Mixture Model
  - Tree-based methods
  - Diffusion model

Approach 2: Pseudo-Counts

- In the original paper Bellemare et al., (2016), they show that count-based reward-bonus helps explore more rooms than vanilla DQN.

Approach 2: Pseudo-Counts

- In the original paper Bellemare et al., (2016), they show that count-based reward-bonus helps explore more rooms than vanilla DQN.

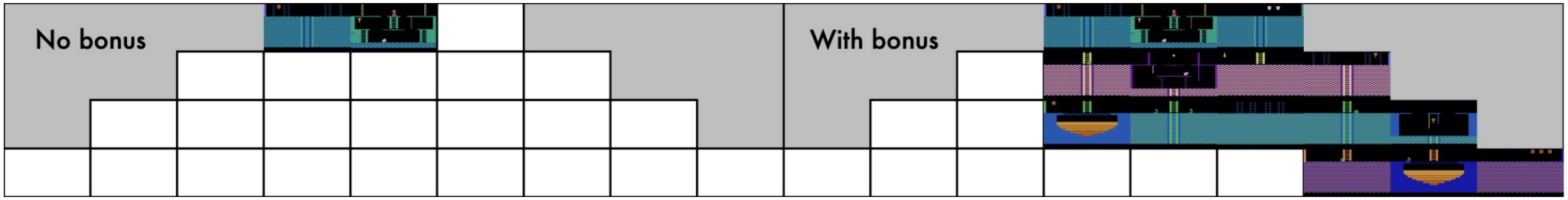


Figure 3: "Known world" of a DQN agent trained for 50 million frames with (**right**) and without (**left**) count-based exploration bonuses, in MONTEZUMA'S REVENGE.

Approach 2: Pseudo-Counts

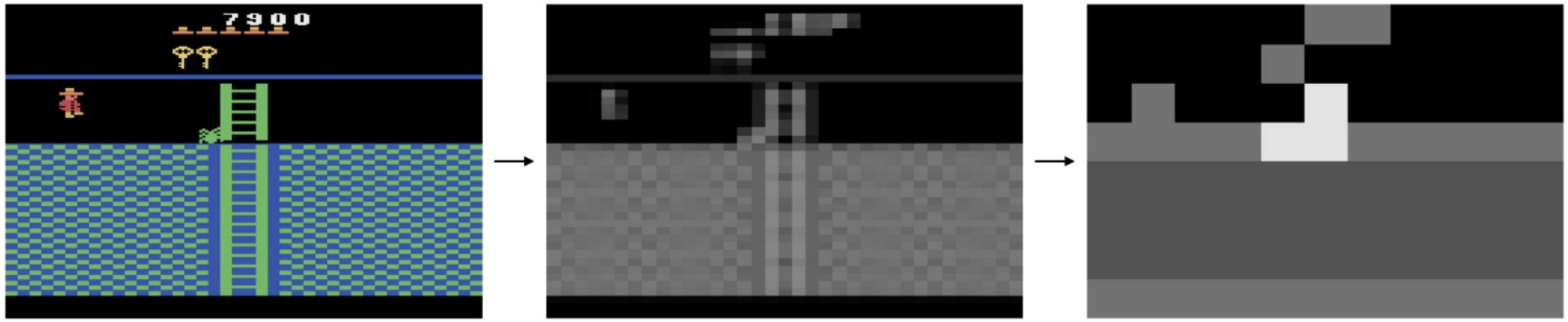
- Another variance of pseudo-counts uses hash functions:

$$\phi: S \to \mathbb{Z}$$

- i.e. mapping states to a finite set of clusters, e.g. SimHash

$$\phi(s) = \operatorname{sgn}(Ag(s)) \in \{-1, 1\}^k$$

- Then, one can count the # of occurrence of clusters instead of states.



Approach 2: Pseudo-Counts

- However, a predefined hash map may not work well in representing the similarity between states in particular environments.
- Solution? Learned hash map.
- Approach in the original paper: auto-encoder (AE)

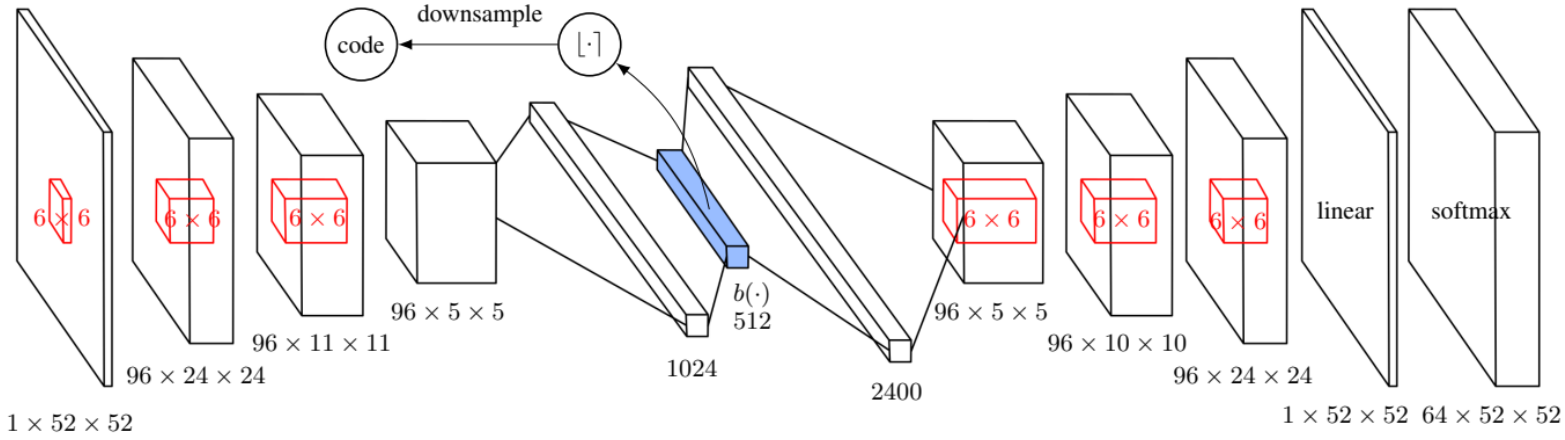


Figure 1: The autoencoder (AE) architecture for ALE; the solid block represents the dense sigmoidal binary code layer, after which noise $U(-a, a)$ is injected.

Approach 2: Pseudo-Counts

- Recall in model-based RL (e.g. DreamerV2), we learn a latent world model in the latent state space Z.

- Potentially can use the counts in this discrete latent state to calculate reward bonus.

Approach 2: Pseudo-Counts

- What's the problem with count-based reward?
- The UCB bonus is designed to upper-bound the **uncertainty of transition estimation** on each (s,a) pairs.
- It only happens to scale with $\sqrt{1/N(s,a)}$ in tabular MDPs.
- Inverse count may not be a good indicator of uncertainty in general.

Approach 3: Uncertainty Estimation

- Can we directly estimate the uncertainty in transition estimation?
- How about doing it directly? Learn a transition function f (a.k.a. model-based RL)

$$b(s_t, a_t) = ||f(s_t, a_t) - s_{t+1}|\Big|^2$$

- Recall the problem with model-based RL: learning the raw input transition is hard.
- Instead one can learn the latent transition.
- Is $||\cdot||^2$ the correct loss?

Approach 3: Uncertainty Estimation

- What if I want to do model-free RL and not learn a transition function?
- How about using the prediction error for something else, such as… a random target function?

Approach 3: Uncertainty Estimation

- Random Network Distillation (RND)
- Initialize some random target network $f: S \to \mathbb{R}^d$
- For any visited state s_t, observes $y_t = f(s_t)$.
- Run regression and learn $\hat{f}$ to minimize $\sum_t \left|\left|\hat{f}(s_t) - y_t\right|\right|_2^2$
- Then, define the reward bonus as $b(s) = \left|\left|\hat{f}(s) - f(s)\right|\right|_2^2$.

Approach 3: Uncertainty Estimation

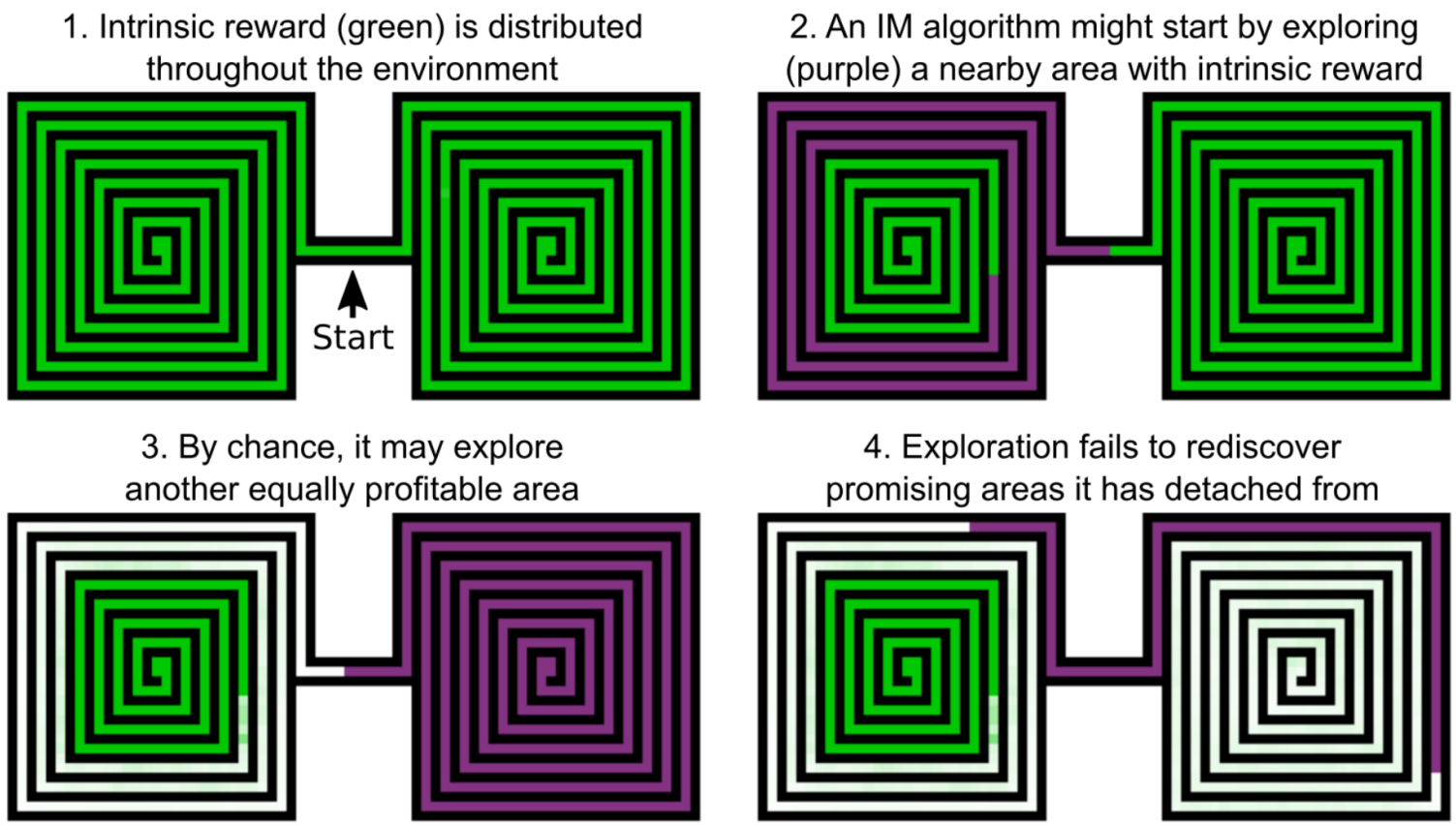
- Random Network Distillation (RND)
- It's surprising that RND works given it's predicting something irrelevant to the main task.
- However, it actually have merits, at least in the tabular setting.
- When doing regression in finite domains, the regression loss on s indeed scales $\propto \sqrt{\frac{1}{N(s)}}$.

Drawbacks of reward bonus in deep RL

- There are some common drawbacks when using reward bonus with deep RL:

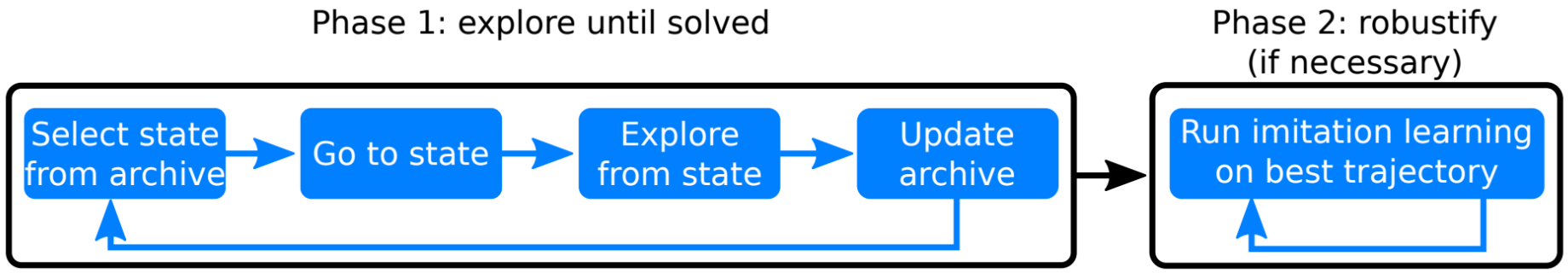
1. Function approximation is slow to catch up.
2. Exploration bonus is non-stationary, incurring non-stability in training.
3. Requires large memory buffer.

Drawbacks of reward bonus in deep RL



Approach 4: Direct Exploration

- **Go-Explore** (Ecoffet, et al., 2019)



- Maintain a selective archive of the promising states, e.g. states with high reward bonus
- Maintain a set of trajectories/goal-conditioned policies that are capable of reaching these promising states.

Approach 4: Direct Exploration