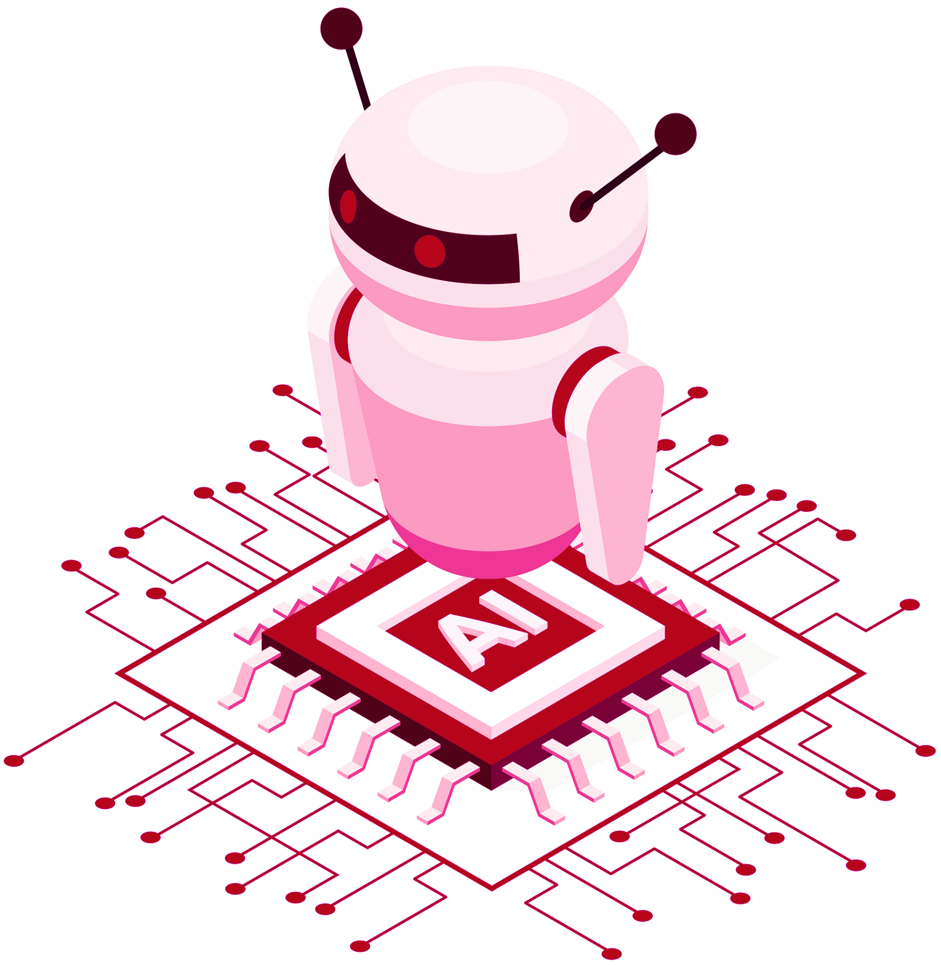

Deep Reinforcement Learning

Professor Mohammad Hossein Rohban

Homework 10:

Exploration in Deep Reinforcement Learning

By:

[Ali Ghasemzadeh]

[401106339]

Spring 2025

Contents

1 Task 1: Bootstrap DQN Variants 1

2 Task 2: Random Network Distillation (RND) 1

Grading

The grading will be based on the following criteria, with a total of 290 points:

Task	Points
Task 1: Bootstrap DQN Variants	100
Task 2: Random Network Distillation (RND)	100
Clarity and Quality of Code	5
Clarity and Quality of Report	5
Bonus 1	80

1 Task 1: Bootstrap DQN Variants

- The complete guidelines for implementing the Bootstrap DQN algorithm, including the RPF and BIV variants, are provided in the Jupyter notebook. You will find detailed instructions on how to set up the environment, implement the algorithms, and evaluate their performance.
- Make sure to read Guidelines section in the notebook carefully.

2 Task 2: Random Network Distillation (RND)

- You will implement the missing core components of Random Network Distillation (RND) combined with a Proximal Policy Optimization (PPO) agent inside the MiniGrid environment.
- **TODO:** You must complete the following parts:

File	TODO Description
`Core/model.py`	Implement the architecture of `TargetModel` and `PredictorModel`.
`Core/model.py`	Implement `_init_weights()` method for proper initialization.
`Core/ppo_rnd_agent.py`	Implement `calculate_int_rewards()` to compute intrinsic rewards.
`Core/ppo_rnd_agent.py`	Implement `calculate_rnd_loss()` to compute predictor training loss.

Table 1: Summary of required TODO implementations

- Questions:
  1. What is the intuition behind Random Network Distillation (RND)? Why does a prediction error signal encourage better exploration?
  2. Why is it beneficial to use both intrinsic and extrinsic returns in the PPO loss function?
  3. What happens when you increase the `predictor_proportion` (i.e., the proportion of masked features used in the RND loss)? Does it help or hurt learning?
  4. Try training with `int_adv_coeff=0` (removing intrinsic motivation). How does the agent's behavior and reward change?
  5. Inspect the TensorBoard logs. During successful runs, how do intrinsic rewards evolve over time? Are they higher in early training?

1. **Intuition behind Random Network Distillation (RND)**
   Random Network Distillation introduces a fixed, randomly initialized "target" neural network $f_{\text{target}}(s)$ and a "predictor" network $f_{\text{pred}}(s)$. For each state s, the predictor tries to match the target's output. The prediction error :

$$\left|\left|f_{\text{target}}(s) - f_{\text{pred}}(s)\right|\right|^2$$

   is treated as an intrinsic reward. Intuitively, states that the agent has not seen often (or at all) yield larger prediction errors—because the predictor has not yet learned to approximate the random target on those inputs. This "surprise" encourages the agent to explore novel states in order to reduce the prediction error. As learning progresses, the predictor's error on frequently visited states goes down, so the agent is driven toward truly unfamiliar states.

2. **Benefit of combining intrinsic and extrinsic returns in PPO**
   In PPO (Proximal Policy Optimization), the policy gradient is weighted by an advantage estimate. When using both extrinsic returns (from the environment's reward r^{ext}) and intrinsic returns (from RND, r^{int}), one typically forms a combined advantage:
   $$A_t \;=\; \hat{R}_t^{\text{ext}} - V_\phi(s_t) \;+\; \lambda_{\text{int}}\big(\hat{R}_t^{\text{int}} - V_\phi^{\text{int}}(s_t)\big),$$
   where λ_{int} is an intrinsic-advantage coefficient. The extrinsic return encourages the agent to maximize task performance, while the intrinsic return encourages exploration of novel states. By blending both, PPO still tends toward high extrinsic reward but does not prematurely converge to suboptimal policies due to insufficient exploration. In short, extrinsic rewards focus on exploitation, and intrinsic rewards focus on exploration, and combining them yields a more balanced update.

3. **Effect of increasing `predictor_proportion`**
   The hyperparameter `predictor_proportion` controls how many of the features (e.g., output dimensions of the random target) the predictor must match when computing the RND loss. Concretely, if `predictor_proportion` $= p$, the predictor is trained only on a random subset of $p \times D$ dimensions out of D.
   - If p is very small, the predictor sees only a few components of the random target, so the intrinsic signal becomes noisier (higher variance) because the agent is rewarded for visiting states that reduce error in only a small subset of dimensions. This can hurt learning by making intrinsic rewards unstable.
   - If p is very large (close to 1), the predictor must learn nearly the entire random target. The prediction errors then drop quickly even on moderately visited states, causing the intrinsic reward to vanish prematurely and reducing exploration.

   In practice, an intermediate p (e.g., 0.5) often strikes a balance: the predictor still has to model enough target output to give informative novelty signals, but not so much that it trivializes intrinsic rewards too early. Empirically, increasing p beyond a moderate value can hurt exploration (intrinsic reward collapses), while too small p can inject excessive noise.

4. **Training with `int_adv_coeff` $= 0$ (no intrinsic motivation)**
   Setting `int_adv_coeff` $= 0$ effectively removes the intrinsic-advantage term from the PPO objective. In that case, the agent trains purely on extrinsic rewards:
   $$A_t \;=\; \hat{R}_t^{\text{ext}} - V_\phi(s_t).$$
   - **Behavioral change:** Without intrinsic bonuses, the agent lacks any mechanism to seek out novel or "curious" states. It tends to stick to the first reward signals it discovers and often converges to a suboptimal policy if the extrinsic reward is sparse.
   - **Reward change:** Early in training, the return is typically lower because the agent may not stumble upon high-reward states often. Exploration tends to be near-random (e.g., ϵ-greedy), so it may take many more episodes to find rewarding trajectories. As training continues, extrinsic reward eventually rises, but the learning curve usually has a slower initial growth compared to runs with intrinsic motivation.

5. **Intrinsic rewards over time (from TensorBoard logs)**
   In successful RND+PPO runs, one generally observes that:
   - **Intrinsic rewards are high at the beginning.** Because nearly every state is novel, the predictor network has large errors, so $r_t^{\text{int}} = \|f_{\text{target}}(s_t) - f_{\text{pred}}(s_t)\|^2$ is large. This drives the agent to explore different parts of the state space.

- **Intrinsic rewards decay over time.** As the agent visits more states, the predictor learns to approximate the target mapping for those states, and prediction error on previously visited states decreases. Consequently, r_t^{int} declines. However, spikes in intrinsic reward often occur when the agent discovers a genuinely novel region of the environment (e.g., a new room or area).
- **Plateauing of intrinsic rewards.** After extensive exploration, intrinsic rewards may approach zero for most visited states, signaling that the predictor has "seen" almost everything the agent can reach. At that point, the intrinsic advantage is small, and the agent's learning is driven almost entirely by extrinsic returns.

In summary, the TensorBoard curve for intrinsic return usually starts high, drops quickly during early exploration, and then flattens or shows intermittent bumps when new states are encountered.