

### **Generative methods in reinforcement learning**

Enhanced Experience Replay Generation for Efficient Reinforcement Learning

Automatic Goal Generation for Reinforcement Learning Agents

Recall Traces: Backtracking Models for Efficient Reinforcement Learning

### **Exploration**

Curiosity-driven Exploration by Self-supervised Prediction

Meta-Reinforcement Learning of Structured Exploration Strategies

### **Sparse reward and Data efficiency**

Hindsight Experience Replay

Learning by Play- Solving Sparse Reward Task from Scratch

Inverse Reward Design

### **Self-supervised, unsupervised or meta methods in reinforcement learning**

Curiosity-driven Exploration by Self-supervised Prediction

Diversity is All You Need: Learning Skills Without a Reward Function

Unsupervised Meta-Learning for Reinforcement Learning

Imitating Latent Policies from Observation

### **Credit assignment for multi-agent reinforcement learning**

## Generative methods in reinforcement learning

---

### **Enhanced Experience Replay Generation for Efficient Reinforcement Learning**

This paper proposes a approach for pre-training the agentbased on the enhanced GAN data sampling to shorten the train phase.

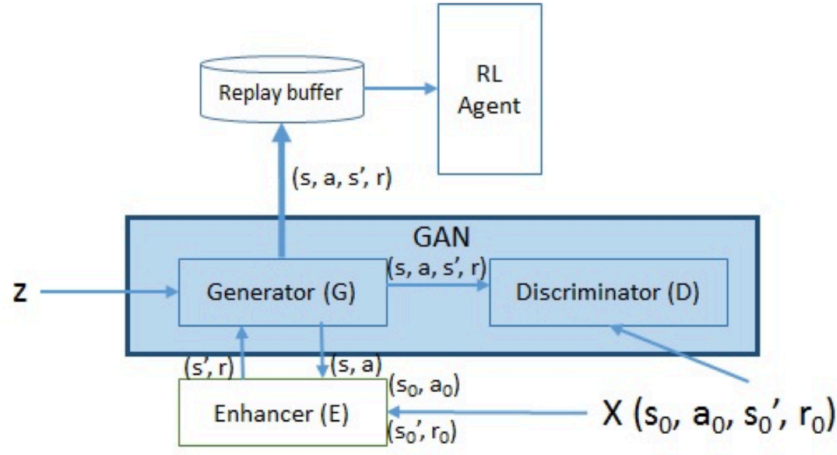


Figure 1: Enhanced GAN structure.

In short, it first collects a small set of data samples using random policy to train the GAN. Then the GAN produces some fake data to pretrain the agent.

Obviously, The Generator produces  $(s, a)$  and the Enhancer learns the relation between  $(s, a)$  and  $(s', r)$  and produces  $(s', r)$ .

## Automatic Goal Generation for Reinforcement Learning Agents

It propose a method that allows an agent to automatically discover the range of tasks that it is capable of performing in its environment. That is to say, generate different tasks at the appropriate level of difficulty for the agent.

The mothod can be broken into three parts:

1. Label a set of goals based on whether they are at the appropriate level for e currency policy.

$$GOID_i := \{g : R_{\min} \leq R^g(\pi_i) \leq R_{\max}\} \subseteq \mathcal{G}.$$

$R_{\min}$  and  $R_{\max}$  are hyperparameters and can be interpreted as the minimum and maximum probability of reaching a goal over  $T$  steps in the previous training iteration. Then the label  $y_g \in \{0, 1\}$  indicates whether  $g \in GOID_i$  for all goals  $g$ . The put the  $y_g$  in the loss function.

$$\begin{aligned}\min_D V(D) &= \mathbb{E}_{g \sim p_{\text{data}}(g)} \left[ y_g (D(g) - b)^2 + \right. \\ &\quad \left. (1 - y_g) (D(g) - a)^2 \right] + \mathbb{E}_{z \sim p_z(z)} [(D(G(z))) - a]^2 \\ \min_G V(G) &= \mathbb{E}_{z \sim p_z(z)} [D(G(z)) - c]^2\end{aligned}\quad (5)$$

The loss function boosts the generator to produce tasks *not too easy and not too hard*.

2. Use the labels to train the generator to produce new goals.
3. Use these new goals to efficiently train the policy, improving its coverage objective.

---

### Algorithm 1 Generative Goal Learning

---

**Input:** Policy  $\pi_0$   
**Output:** Policy  $\pi_N$   
 $(G, D) \leftarrow \text{initialize\_GAN}()$   
 $goals_{\text{old}} \leftarrow \emptyset$   
**for**  $i \leftarrow 1$  **to**  $N$  **do**  
     $z \leftarrow \text{sample\_noise}(p_z(\cdot))$   
     $goals \leftarrow G(z) \cup \text{sample}(goals_{\text{old}})$   
     $\pi_i \leftarrow \text{update\_policy}(goals, \pi_{i-1})$   
     $returns \leftarrow \text{evaluate\_policy}(goals, \pi_i)$   
     $labels \leftarrow \text{label\_goals}(returns)$   
     $(G, D) \leftarrow \text{train\_GAN}(goals, labels, G, D)$   
     $goals_{\text{old}} \leftarrow \text{update\_replay}(goals)$   
**end for**

---

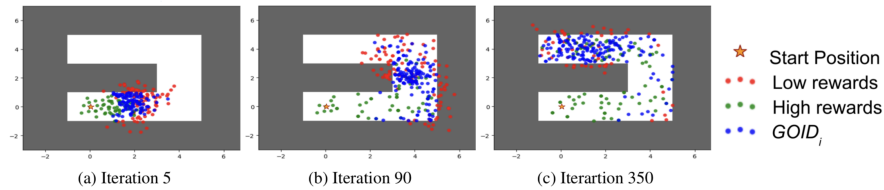
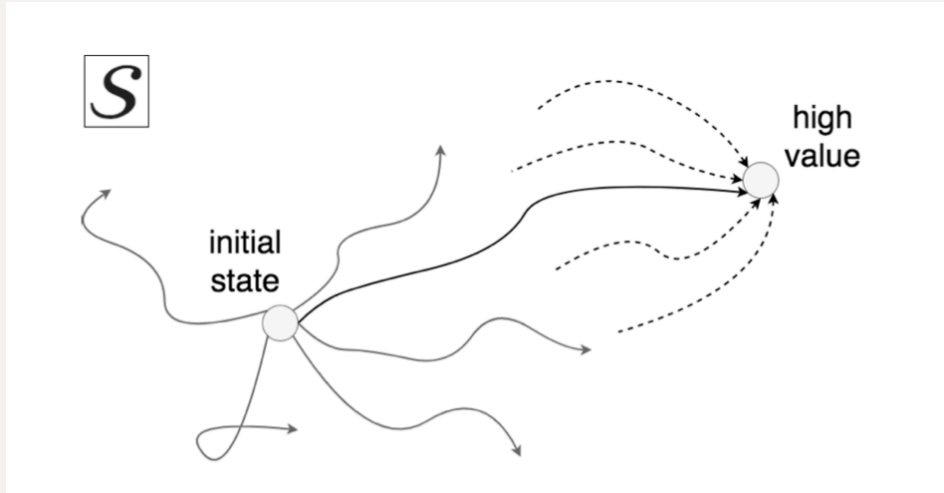


Figure 3. Goals that, at iterations  $i$ , our algorithm trains on - 200 sampled from Goal GAN, 100 from replay. Green goals satisfy  $\bar{R}^g(\pi_i) \geq R_{\max}$ . Blue ones have appropriate difficulty for the current policy  $R_{\min} \leq \bar{R}^g(\pi_i) \leq R_{\max}$ . The red ones have  $R_{\min} \geq \bar{R}^g(\pi_i)$ .

We can see the distribution of generated goals during the training. Green points represent the easy goals, blue points represent the suitable goals and red points mean hard or impossible goals. The learning process shows the agent gradually learns hard tasks.

## Recall Traces: Backtracking Models for Efficient Reinforcement Learning

Use a *backtracking* model to predict the preceding states that terminate at the given high-reward state. Use goal GAN to generate high value state.



The backtracking model consists of the backward policy  $\pi_b = q(t|s_{t+1})$  and a state generator  $q(s_t|a_t, s_{t+1})$ . To make the training more stable, we not directly predict the  $s_t$ , but the  $\Delta s_t = s_t - s_{t+1}$ .

$$q_\phi(\Delta_t, a_t|s_{t+1}) = q(\Delta s_t|a_t, s_{t+1})q(a_t|s_{t+1}).$$

With regard to how to producing high value states, first method is naive. Just pick high value sample from the replay buffer. The second method is based on the goal GAN mention above.

---

#### Algorithm 1 Produce High Value States

---

**Require:** Critic  $V(s)$

**Require:**  $D$ ; transformation 'decoder' from  $g$  to  $s$

**Require:** Experience buffer  $\mathcal{B}$  with tuples  $(s_t, a_t, r_t, s_{t+1})$

**Require:** gen\_state; boolean whether to generate states

**Require:**  $GAN$ , some generative model trained to model high-value goal states

- 1: **if** gen\_state **then**
  - 2:    $g \sim GAN$
  - 3:    $D : g \mapsto s$
  - 4:   Return  $s$
  - 5: **else**
  - 6:   Return  $\argmax(V(s)) \forall s \in \mathcal{B}$
  - 7: **end if**
- 

#### How to train

1. Training the Backtracking Model

$$\begin{aligned}
\mathcal{L}_{\mathcal{B}} &= \log q_{\phi}(\tau) = \log \prod_{t=0}^T q(\Delta s_t, a_t | s_{t+1}) \\
&= \sum_{t=0}^T \log q(\Delta s_t, a_t | s_{t+1}) \\
&= \sum_{t=0}^T \log q(a_t | s_{t+1}) + \log q(\Delta s_t | a_t, s_{t+1}),
\end{aligned}$$

## 2. Improving the Policy from the Recall Traces

RL loss and the imitation loss

$$\mathcal{L}_{\mathcal{I}} = \sum_{t=0}^T \log p(a_t | s_t) = \sum_{t=0}^T \log \pi_{\theta}(a_t | s_t),$$

---

### Algorithm 2 Improve Policy via Backtracking Model

---

**Require:** RL algorithm with parameterized policy (i.e. TRPO, Actor-Critic)

**Require:** Agent policy  $\pi_{\theta}(a|s)$

**Require:** Backtracking model  $B_{\phi} = q_{\phi}(\Delta s_t, a_t | s_{t+1})$

**Require:** Critic  $V(s)$

**Require:**  $k$  quantile of best state values used to train backtracking model

**Require:**  $N$ ; number of backward trajectories per target state

**Require:**  $\alpha, \beta$ ; forward, backward learning rates

1: Randomly initialize agent policy parameters  $\theta$

2: Randomly initialize backtracking model parameters  $\phi$

3: **for**  $t = 1$  to  $K$  **do**

4:   Execute RL algorithm to produce trajectory  $\tau$

5:   Add trajectory  $\tau = (s_1, a_1, r_1, \dots, s_T, a_T, r_T)$  in  $\mathcal{B}$

6:   Estimate  $\nabla_{\theta} R(\pi_{\theta})$  from RL algorithm

7:    $\theta \leftarrow \theta + \alpha \nabla_{\theta} R(\pi_{\theta})$

8:   Compute  $\mathcal{L}_{\mathcal{B}}$  via Equation 5 using the top  $k\%$  of valuable states in  $\mathcal{B}$

9:    $\phi \leftarrow \phi + \beta \nabla_{\phi} \mathcal{L}_{\mathcal{B}}$

10:   Algorithm 1 returns a target state  $s$

11:   Generate  $N$  traces  $\tilde{\tau}$  for  $s$  using  $B(s)$

12:   Compute imitation loss  $\mathcal{L}_{\mathcal{I}}$  via Equation 6

13:    $\theta \leftarrow \theta + \alpha \nabla_{\theta} \mathcal{L}_{\mathcal{I}}$

14:   Update sub-goals or high value states.

15: **end for**

---

# Exploration

---

## Curiosity-driven Exploration by Self-supervised Prediction

## Meta-Reinforcement Learning of Structured Exploration Strategies

## Sparse reward and Data efficiency

---

Many papers adopt the idea of imitation learning to solve the sparse reward problem. As it is less related to our research, these papers are not listed there.

## Hindsight Experience Replay

*The pivotal idea behind HER is to replay each episode with a different goal than the one the agent was trying to achieve.*

The idea is very easy. Typically, the agent can only get reward when arriving at the final state, but in HER we sample some additional goals. When the agent arrives at the goals or  $|s - g| < \epsilon$  in continuous space, it can get additional rewards. Then store the new data in the buffer and train the model via off-policy method.

---

**Algorithm 1** Hindsight Experience Replay (HER)

---

**Given:**

- an off-policy RL algorithm  $\mathbb{A}$ , ▷ e.g. DQN, DDPG, NAF, SDQN
  - a strategy  $\mathbb{S}$  for sampling goals for replay, ▷ e.g.  $\mathbb{S}(s_0, \dots, s_T) = m(s_T)$
  - a reward function  $r : \mathcal{S} \times \mathcal{A} \times \mathcal{G} \rightarrow \mathbb{R}$ . ▷ e.g.  $r(s, a, g) = -[f_g(s) = 0]$
- ▷ e.g. initialize neural networks

Initialize  $\mathbb{A}$ Initialize replay buffer  $R$ **for** episode = 1,  $M$  **do**    Sample a goal  $g$  and an initial state  $s_0$ .    **for**  $t = 0, T - 1$  **do**        Sample an action  $a_t$  using the behavioral policy from  $\mathbb{A}$ :         $a_t \leftarrow \pi_b(s_t || g)$ ▷  $||$  denotes concatenation        Execute the action  $a_t$  and observe a new state  $s_{t+1}$     **end for**    **for**  $t = 0, T - 1$  **do**         $r_t := r(s_t, a_t, g)$         Store the transition  $(s_t || g, a_t, r_t, s_{t+1} || g)$  in  $R$ ▷ standard experience replay        Sample a set of additional goals for replay  $G := \mathbb{S}(\text{current episode})$         **for**  $g' \in G$  **do**             $r' := r(s_t, a_t, g')$             Store the transition  $(s_t || g', a_t, r', s_{t+1} || g')$  in  $R$ ▷ HER        **end for**    **end for**    **for**  $t = 1, N$  **do**        Sample a minibatch  $B$  from the replay buffer  $R$         Perform one step of optimization using  $\mathbb{A}$  and minibatch  $B$     **end for****end for**

---

## Learning by Play- Solving Sparse Reward Task from Scratch

*The key idea behind our method is that active (learned) scheduling and execution of auxiliary policies allows the agent to efficiently explore its environment.*

This paper gives me a feeling of engineering. It is too complex to figure out which part really helps, although it actually outperforms baselines. The auxiliary reward needs priori knowledge and the task seem to be easy with auxiliary reward.

## Inverse Reward Design

This paper proposes a solution to inverse reward design (IRD) which infer the true reward function from a set of given reward functions.

$$P(\tilde{w}|w^*, \tilde{M}) \propto \exp \left( \beta \mathbb{E} \left[ w^{*\top} \phi(\xi) | \xi \sim \pi(\xi | \tilde{w}, \tilde{M}) \right] \right)$$

The idea that proxy reward functions are likely to the extent that they incentivize high utility behavior in the training MDP. The  $P$  is a distribution on the true utility function.

The distribution means the true utility should be high when the real reward is high. Moreover, it tends to penalize no-reward situation. I mean the situation which has less utility. So the model behaves more risk-averse. The agent learns to avoid something unforeseen because it is useless and could

brings damage. On the other hand, it also avoids potentially good things as the author mentions in the discussion.

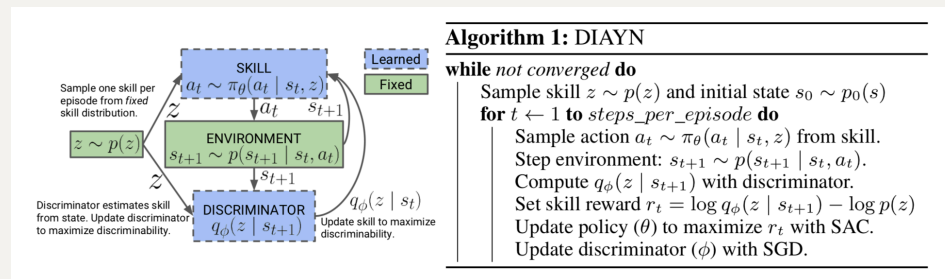
## Self-supervised, unsupervised or meta methods in reinforcement learning

Most of papers in this part are related to exploitation and sparse reward.

### Curiosity-driven Exploration by Self-supervised Prediction

### Diversity is All You Need: Learning Skills Without a Reward Function

The proposed method learns skills by maximizing an information theoretic objective using a maximum entrop policy.



Learn the skill  $z$  as the condition in policy  $\pi_\theta(a_t | s_t, z)$ . Different skills should visit different states, and hence be distinguishable. Skills with high entropy that remain discriminable must explore a part of the state space far away from other skills.

We have three parts to optimize.

1. The mutual information between skills and states.  $MI(z, s)$

It means the skills should control which states the agent visits.

2. Minimize the mutual information between skill and action given the state  $MI(z, a | s)$

It ensures that states, not actions, are used to distinguish skills.

3. Maximize the  $\mathcal{H}([a | s])$



$$\mathcal{F}(\theta) \triangleq MI(s, z) + \mathcal{H}[a | s] - MI(a, z | s)$$

1 gives intuition on how we optimize it:<sup>3</sup>

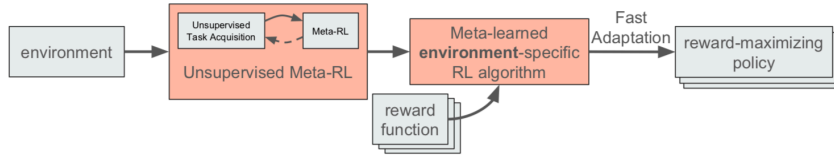
$$\begin{aligned}\mathcal{F}(\theta) &= MI(s, z) + \mathcal{H}[a | s] - MI(a, z | s) \\ &= (\mathcal{H}[z] - \mathcal{H}[z | s]) + \mathcal{H}[a | s] \\ &\quad - (\mathcal{H}[a | s] - \mathcal{H}[a | s, z]) \\ &= \mathcal{H}[z] - \mathcal{H}[z | s] + \mathcal{H}[a | s, z]\end{aligned}$$

$$\begin{aligned}\mathcal{F}(\theta) &= \mathcal{H}[a | s, z] - \mathcal{H}[z | s] + \mathcal{H}[z] \\ &= \mathcal{H}[a | s, z] + \mathbb{E}[\log p(z | s)] - \mathbb{E}[\log p(z)] \\ &\geq \mathcal{H}[a | s, z] + \mathbb{E}[\log q_\phi(z | s) - \log p(z)] \triangleq \mathcal{G}(\theta, \phi)\end{aligned}$$

So the reward function can be written as:

$$r_z(s, a) \triangleq \log q_\phi(z | s) - \log p(z)$$

## Unsupervised Meta-Learning for Reinforcement Learning




---

### Algorithm 1: Unsupervised Meta-Reinforcement Learning Pseudocode

---

**Data:**  $\mathcal{M} \setminus R$ , an MDP without a reward function

**Result:** a learning algorithm  $f : \mathcal{D} \rightarrow \pi$

Initialize  $\mathcal{D} = \emptyset$

$D_\phi \leftarrow \text{DIAYN}()$  or  $D_\phi \leftarrow \text{random}$

**while not converged do**

    Sample latent task variables  $z \sim p(z)$

    Extract corresponding task reward functions

$r_z(s)$  using  $\mathcal{D}_\phi(z|s)$

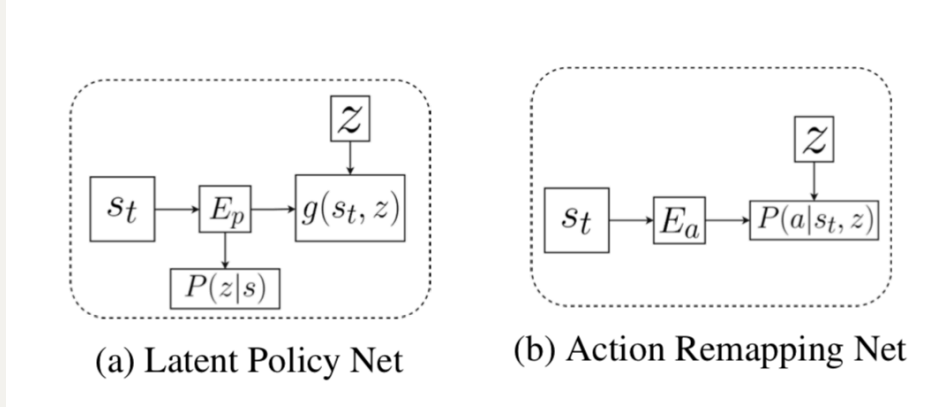
    update  $f$  using MAML with reward  $r_z(s)$

---

Use DIAYN to learn a policy and discriminator. Sample tasks by generating samples  $z \sim p(z)$  and using the corresponding task reward  $r_z(s) = \log(D_\phi(z|s))$

## Imitating Latent Policies from Observation

This paper proposed a imitation learning method to infer latent policies from expert observations. It is a two-stage model.



Step 1: learning latent policies

From the expert observation  $\{s_1, \dots, s_t\}$  learn the latent policy  $P(z|s_t)$  by fitting the  $s_t$  and  $s_{t+1}$ .

$$\mathcal{L}_{min} = \min \|\Delta_t - g(s_t, z)\|^2, \Delta_t = s_{t+1} - s_t$$

$$\mathcal{L}_{exp} = \|s_{t+1} - \hat{s}_{t+1}\|^2, \hat{s}_{t+1} = \sum_z P(z|s_t)g(s_t|z)$$

$$\mathcal{L} = \mathcal{L}_{min} + \mathcal{L}_{exp}$$

Step2 :Action remapping

Learn a mapping from the latent space to the true action space  $P(a|z, s_t)$ . The agent randomly selects  $a_t$  then we can get  $(s_t, A_t, s_{t+1})$ . Select the  $z_t$  which causes the  $s_{t+1}$  by the learned model from the Step 1. Therefore, the  $z_t$  is corresponding to the environment action  $a_t$ .

*Step 2: Action remapping*

Observe state  $s_0$

**for**  $t \leftarrow 0 \dots \#Interactions$  **do**

    Choose latent action  $z_t \leftarrow \arg \max_z P_\omega(z|s_t)$

    Take  $\epsilon$ -greedy action  $a_t \leftarrow \arg \max_a P_\xi(a|z_t, s_t)$

    Observe state  $s_{t+1}$

    Infer closest latent action  $\hat{z} = \arg \min_z \|E_p(s_{t+1}) - E_p(g(s_t, z))\|^2$

$\xi \leftarrow \xi + \nabla_\xi \log \frac{P(a_t|\hat{z}, s_t)}{\sum_a P(a|\hat{z}, s_t)}$

# Credit assignment for multi-agent reinforcement learning

---