LIIR:Learning Individual Intrinsic Rewards in Multi-agent reinforcement learning

Yali Du*, Lei Han*, Meng Fang, Ji Liu, Tianhong Dai, Dacheng Tao
33rd Conference on NeurIPS 2019

2024. 05. 23

AI Robotics KR, 에이전트브레인스토밍
Minkyoung Kim

Introduction

- **Cooperative Decentralized MARL의 great challenge**
  - 하나의 팀 보상을 받을 때, 개별 에이전트들이 다양한 행동을 생성하는 것
  - 하나의 팀 보상으로 개별 에이전트들은 팀에 대한 기여도(credit assignment)를 알기 어려움

  → MARL의 Credit Assignment Problem

- 이전 연구들은 해당 challenge를 아래 기법으로 접근함
  - Reward shaping
    - 한계1: 개별 agent에게 정확한 reward를 할당해주는데 많은 human labor가 필요함
    - 한계2: Real world task에 각 agent에 대한 정확한 보상 정의가 존재하지 않을 수 있음
  - Designing centralized critic
    - 학습 시, centralized critic이 각 agent의 state-action values를 구별할 수 있게 고려

Introduction

- LIIR(Learning Individual Intrinsic Rewards in Multi-agent reinforcement learning)은
  - Reward shaping + Critic Learning을 모두 사용
  - Centralized critic을 maximize하면서 각 agent가 parameterized individual intrinsic reward function을 학습

→ Individual intrinsic reward로 각 agent들의 기여도 문제를 해결하고자 함

→ Individual intrinsic reward는 각 agent들이 다양한 행동을 하도록 학습

Related Work

- **Focus on the architecture design of the critic**
  - COMA(Counterfactual multi-agent policy gradient)
    - Counterfactual baseline을 통해 agent의 credit assignment를 접근
  - VDN(Value Decomposition)
    - Centralized value는 individual agent value의 sum
  - QMIX
    - Centralized Q-value function은 individual Q-value function의 monotonically increasing sum

  - **<u>LIIR의 차별성: intrinsic reward를 학습</u>**
    → Value function에 대한 assumption이 없음
    → 각 agent들은 explicit immediate intrinsic reward를 timestep마다 획득하여 기여도 할당 받음

Background

- **Fully Cooperative Multi-Agent Reinforcement Learning**
- $< A, S, U, P, r, \gamma, \rho_0 >$
  - n agents $A = \{1, \dots, n\}$
  - Observation space of the agents as $S = \{S_1, S_2, \dots, S_n\}$
    - At timestep *t*, let $s_t = \{s_t^i\}_{i=1}^n$ with each $s_t^i \in S_i$ being the partial observation from agent *i*
    - $s_t \in S$, true state of the environment
  - Action space of the agent as $U = \{U_1, U_2, \dots, U_n\}$
    - At timestep t, let $u_t = \{u_t^i\}_{i=1}^n$ with each $u_t^i \in U_i$ indicating the action taken by the agent *i*
  - State transition function, $P(s_{t+1}|s_t, u_t) : S \times U \times S \to [0,1]$
  - Reward function from environment, $r(s_t, u_t): S \times U \to \mathbb{R}$
    - Extrinsic team reward, $r^{ex}(s_t, u_t)$
  - Discount factor, $\gamma \in [0,1]$
  - Distribution of the initial state $s_0, \rho_0: S \to \mathbb{R}$
  - Stochastic policy for agent *i*, $\pi_i(u_t^i|s_t^i): S_i \times U_i \to [0,1]$
    - $\pi = \{\pi_i\}_{i=1}^n$

Background

- **Fully Cooperative Multi-Agent Reinforcement Learning**
  - Aim to **find optimal $\boldsymbol{\pi}^* = \{\pi_i^*\}_{i=1}^n$**
    that achieve the maximum expected extrinsic team reward $J^{ex}(\pi^*)$

$$J^{\text{ex}}(\pi) = \mathbb{E}_{\mathbf{s}_0, \mathbf{u}_0, \dots}[R_0^{\text{ex}}] \text{ with } R_t^{\text{ex}} = \sum_{l=0}^{\infty} \gamma^l r_{t+l}^{\text{ex}}$$

$$\text{where } \mathbf{s}_0 \sim \rho_0(\mathbf{s}_0),\ u_t^i \sim \pi_i(u_t^i | s_t^i) \text{ for } i \in \mathcal{A}, \text{ and } \mathbf{s}_{t+1} \sim P(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{u}_t)$$

  - Extrinsic value function

$$V_{\pi}^{\text{ex}}(\mathbf{s}_t) = \mathbb{E}_{\mathbf{u}_t, \mathbf{s}_{t+1}, \dots}[R_t^{\text{ex}}]$$

Background

- **Centralized Learning with Decentralized Execution**
  - Learn a centralized critic to update the decentralize policy during training

  - Actor-Critic algorithms use ***n* independent parameterized policy $\boldsymbol{\pi_{\theta_i}}$** for i ∈ A
  - Update θ_i by maximizing the expected extrinsic reward

$$J^{\text{ex}}(\theta_1, \theta_2, \cdots, \theta_n) = \mathbb{E}_{\mathbf{s},\mathbf{u}}\left[R^{\text{ex}}\right]$$

$$\nabla_{\theta_i} J^{\text{ex}}(\theta_1, \theta_2, \cdots, \theta_n) = \mathbb{E}_{\mathbf{s},\mathbf{u}}\left[\nabla_{\theta_i} \log \pi_{\theta_i}(u_i|s_i) A_\pi(\mathbf{s}, \mathbf{u})\right],$$

Centralized critic

$$A_\pi(\mathbf{s}, \mathbf{u}) = r^{\text{ex}}(\mathbf{s}, \mathbf{u}) + V^{\text{ex}}(\mathbf{s}') - V^{\text{ex}}(\mathbf{s})$$

Standard advantage function

Background

- **Parameterized Intrinsic Reward**
  - Intrinsic reward function, $r_\eta^{in}(s,a)$ for a state-action pair (s,a) of the agent
  - Intrinsic reward is summed up with the extrinsic reward r^{ex} for updating the policy
  - Intrinsic reward parameter η is updated towards maximizing the expected extrinsic reward J^{ex}
  - The intuition for updating η is to find the effect that the change on η would influence the extrinsic value through the change in the policy parameter

LIIR Method

- The Objectives
  - Intrinsic reward function, $r_{\eta_i}^{in}(s_i, u_i)$ which is parameterized by η_i
    - Takes a state-action pair (s_i, u_i) of an individual agent *i* as input
  - Assign agent *i* a distinct *proxy reward*
    - At time step *t*
    - λ: hyper-parameter that balances the extrinsic team reward and the distinct intrinsic reward

$$r_{i,t}^{\text{proxy}} = r_t^{\text{ex}} + \lambda r_{i,t}^{\text{in}}$$

Extrinsic team reward Intrinsic reward

LIIR Method

- The Objectives
  - **Discounted proxy reward** for each agent *i* as

$$R_{i,t}^{\text{proxy}} = \sum_{l=0}^{\infty} \gamma^l (r_{t+l}^{\text{ex}} + \lambda r_{i,t+l}^{\text{in}}),$$

Proxy reward

$$r_{i,t}^{\text{proxy}} = r_t^{\text{ex}} + \lambda r_{i,t}^{\text{in}}$$

  - **Proxy value function** for agent *i* as

$$V_i^{\text{proxy}}(s_{i,t}) = \mathbb{E}_{u_{i,t}, s_{i,t+1}, \cdots}[R_{i,t}^{\text{proxy}}].$$

    - Do not have any physical meanings (different from extrinsic value V^{ex})
    - Only used for updating the individual policy parameter θ_i's.

LIIR Method

- **Bilevel** optimization algorithm
  - **Outer objective**: Maximize the extrinsic expected return (standard multi-agent return)
    - The intrinsic reward parameters $\boldsymbol{\eta}$ are updated to maximize the extrinsic expected return
  - **Inner objective**: Maximize the individual proxy expected return
    - Policy parameters θ_i are updated with respect to the inner proxy tasks

Outer $\max_{\boldsymbol{\eta},\boldsymbol{\theta}} \quad J^{\text{ex}}(\boldsymbol{\eta}),$

Inner $\text{s.t.} \quad \theta_i = \arg\max_{\theta} J_i^{\text{proxy}}(\theta, \boldsymbol{\eta}), \quad \forall i \in [1, 2, \cdots, n]$

$$J_i^{\text{proxy}} := \mathbb{E}_{s_{i,0}, u_{i,0}, \cdots}\left[R_{i,0}^{\text{proxy}}\right]$$

$\boldsymbol{\theta}$: $policy\ parameter\ set\ \{\theta_1, \theta_2, \ldots, \theta_n\}$
$\boldsymbol{\eta}$: intrinsic reward para$meter\ set\ \{\eta_1, \eta_2, \ldots, \eta_n\}$

LIIR Method

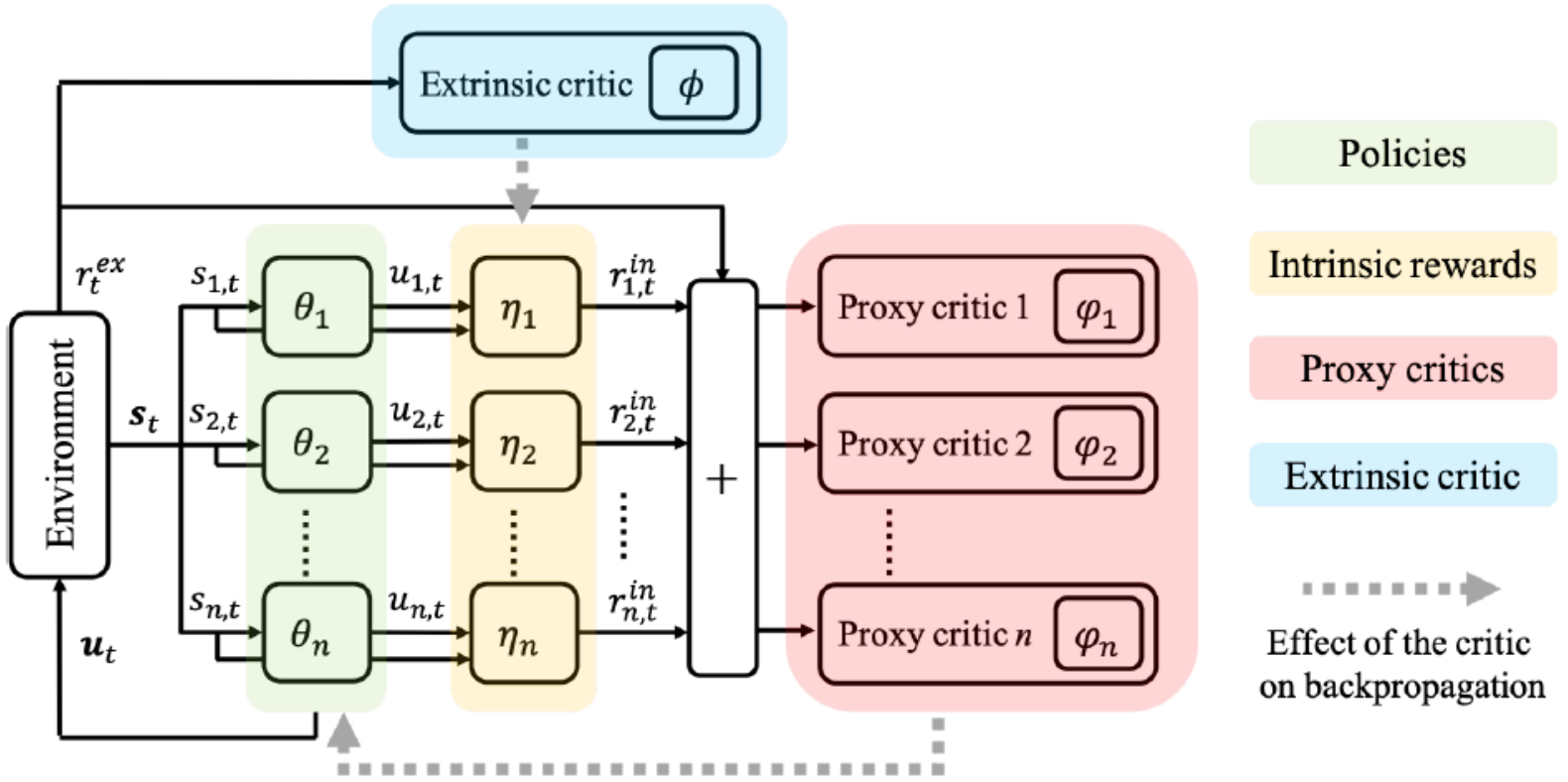


Figure 1: Architecture of the LIIR method. The architecture contains four parameter components: θ_i's for policies, η_i's for intrinsic reward, and φ_i's and ϕ's for extrinsic and proxy values respectively.

LIIR Method

$$r_{i,t}^{\text{proxy}} = r_t^{\text{ex}} + \lambda r_{i,t}^{\text{in}}$$

Extrinsic team reward Intrinsic reward

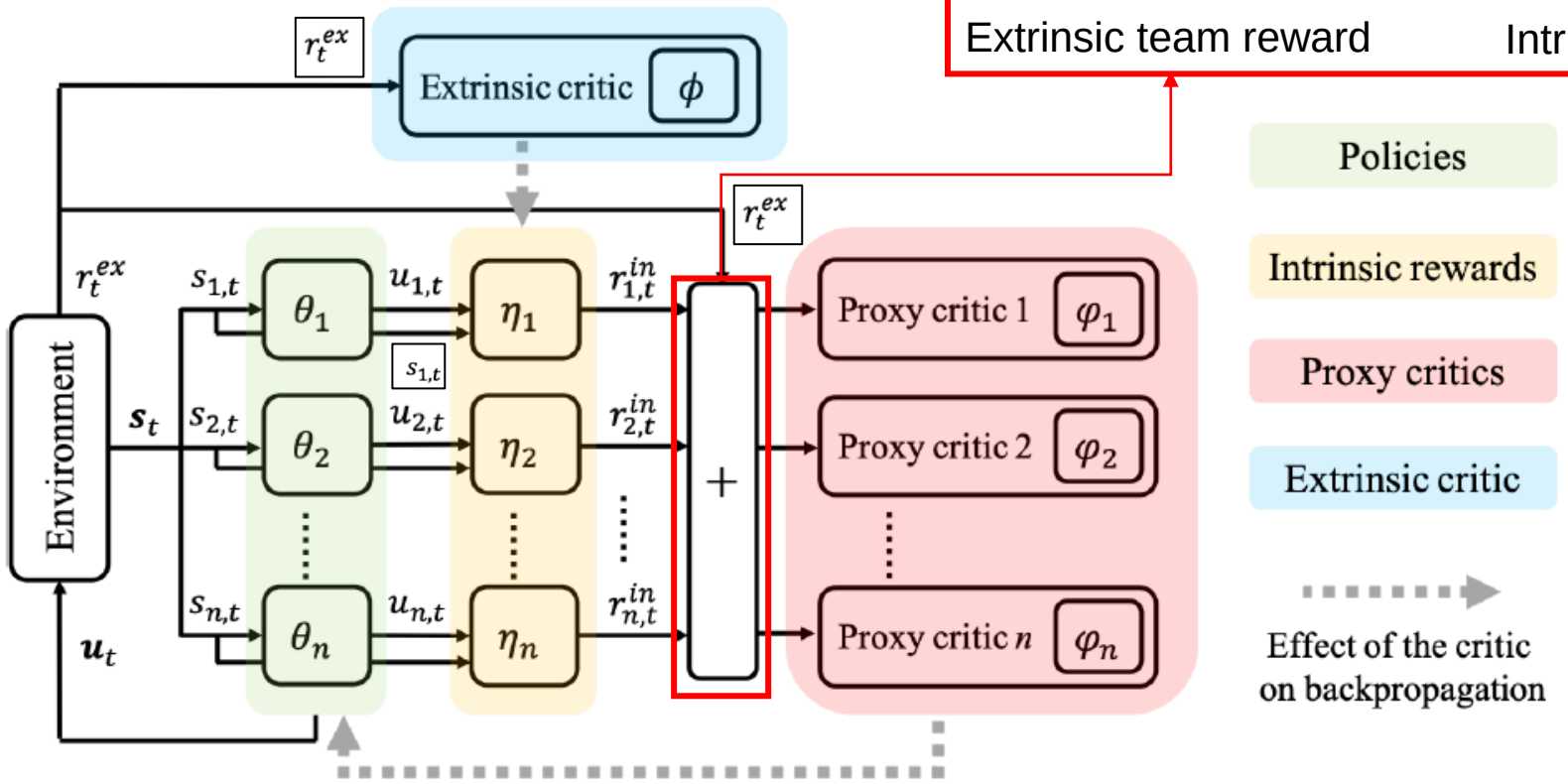


Figure 1: Architecture of the LIIR method. The architecture contains four parameter components: θ_i's for policies, η_i's for intrinsic reward, and φ_i's and ϕ's for extrinsic and proxy values respectively.

LIIR Method

$$\theta_i' = \theta_i + \alpha \nabla_{\theta_i} \log \pi_{\theta_i}(u_i|s_i) A_i^{\text{proxy}}(s_i, u_i)$$

$$A_i^{\text{proxy}}(s_i, u_i) = r_i^{\text{proxy}}(s_i, u_i) + V_{\varphi_i}^{\text{proxy}}(s_i') - V_{\varphi_i}^{\text{proxy}}(s_i)$$

$V_{\varphi_i}^{\text{proxy}}$ is the proxy value parameterized by φ_i

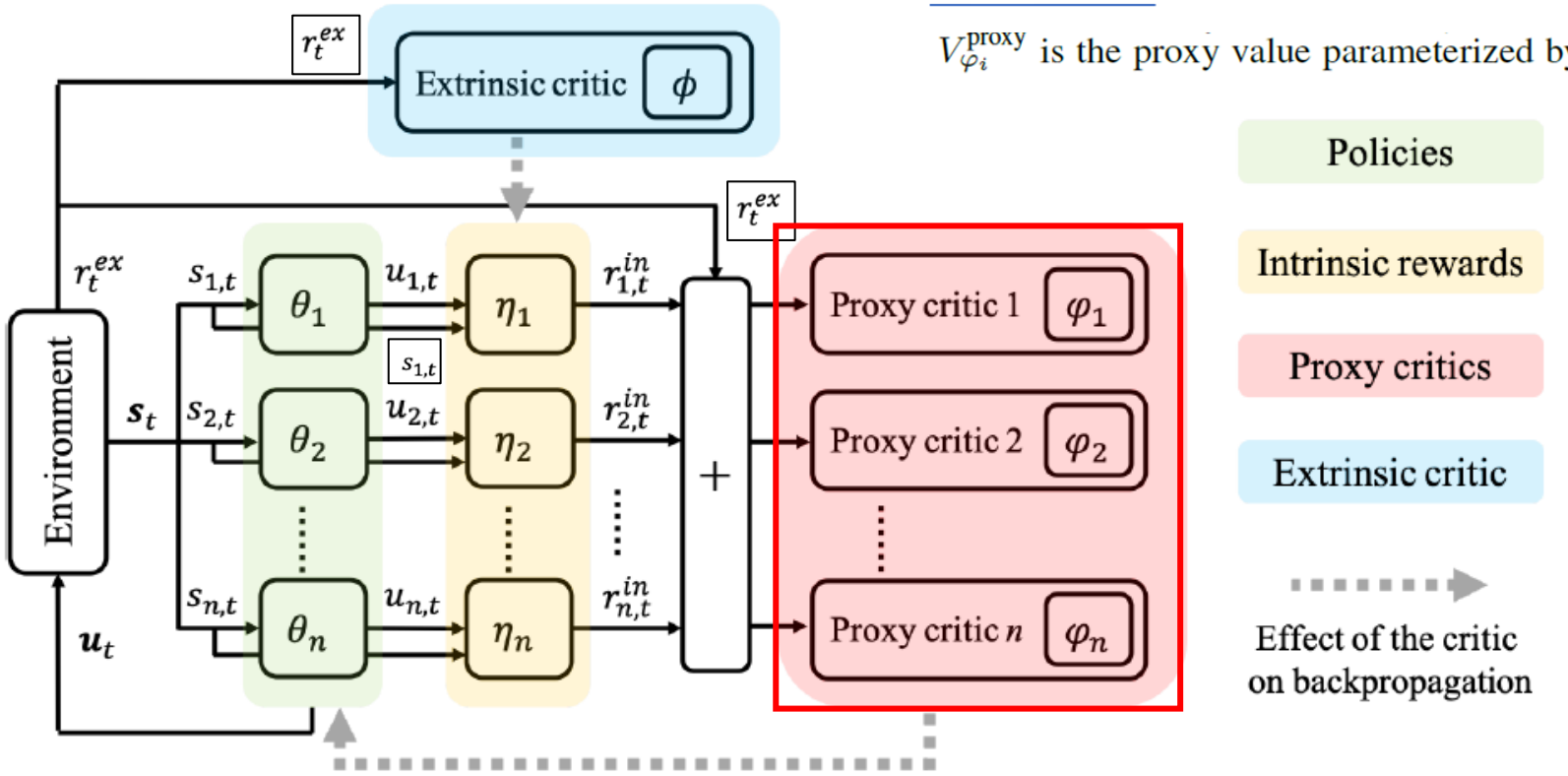


Figure 1: Architecture of the LIIR method. The architecture contains four parameter components: θ_i's for policies, η_i's for intrinsic reward, and φ_i's and ϕ's for extrinsic and proxy values respectively.

LIIR Method

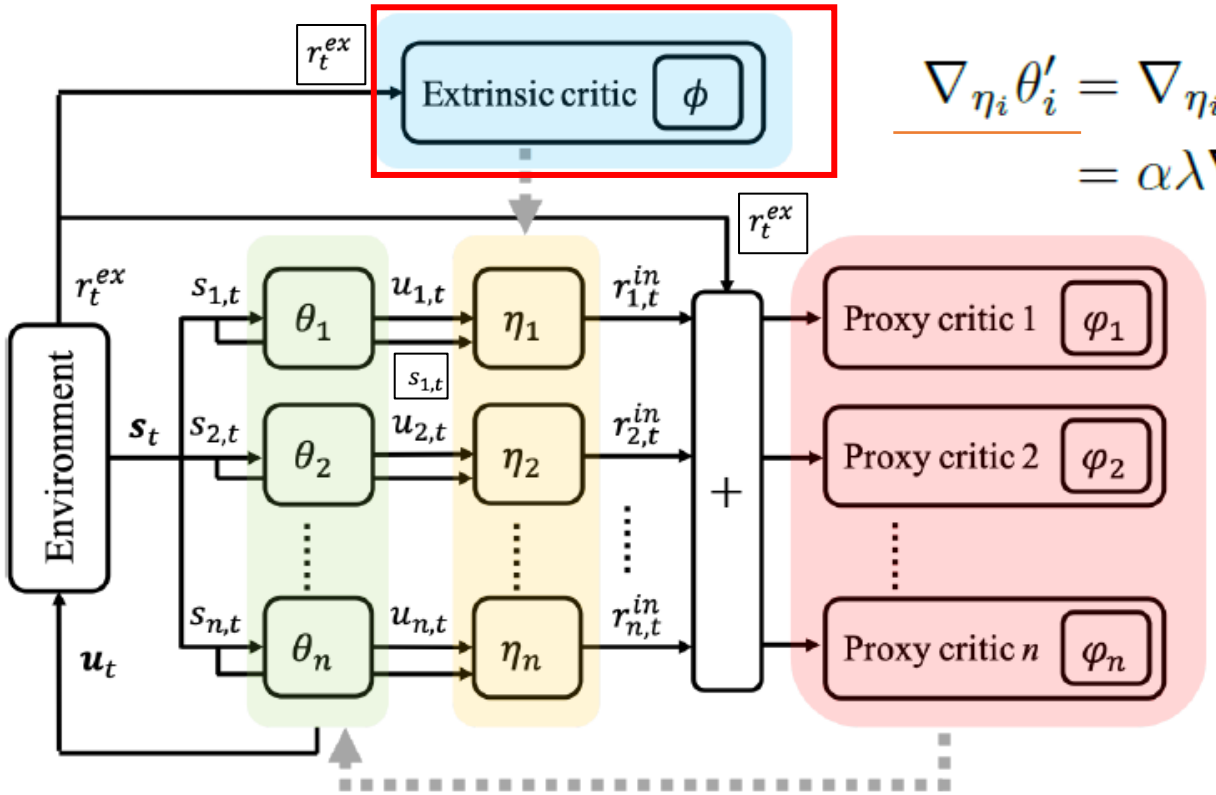
$$\nabla_{\eta_i} J^{\text{ex}} = \nabla_{\theta_i'} J^{\text{ex}} \nabla_{\eta_i} \theta_i'.$$

$$\nabla_{\theta_i'} J^{\text{ex}} = \nabla_{\theta_i'} \log \pi_{\theta_i'}(u_i|s_i) A^{\text{ex}}(\boldsymbol{s}, \boldsymbol{u})$$

$$A^{\text{ex}}(\boldsymbol{s}, \boldsymbol{u}) = r^{\text{ex}}(\boldsymbol{s}, \boldsymbol{u}) + V_\phi^{\text{ex}}(\boldsymbol{s}') - V_\phi^{\text{ex}}(\boldsymbol{s})$$

$V_\phi^{\text{ex}}(\boldsymbol{s})$ is the extrinsic value parameterized by ϕ.

$$\nabla_{\eta_i} \theta_i' = \nabla_{\eta_i} [\theta_i + \alpha \nabla_{\theta_i} \log \pi_{\theta_i}(u_i|s_i) A_i^{\text{proxy}}(s_i, u_i)]$$
$$= \alpha\lambda \nabla_{\theta_i} \log \pi_{\theta_i}(a_i|s_i) \nabla_{\eta_i} r_i^{\text{proxy}}(s_i, u_i).$$



Intrinsic rewards

Proxy critics

Extrinsic critic

Effect of the critic on backpropagation

$$A_i^{\text{proxy}}(s_i, u_i) = r_i^{\text{proxy}}(s_i, u_i) + V_{\varphi_i}^{\text{proxy}}(s_i') - V_{\varphi_i}^{\text{proxy}}(s_i)$$

$V_{\varphi_i}^{\text{proxy}}$ is the proxy value parameterized by φ_i

Figure 1: Architecture of the LIIR method. The architecture contains four parameter components: θ_i's for policies, η_i's for intrinsic reward, and φ_i's and ϕ's for extrinsic and proxy values respectively.

Method

Algorithm 1 The optimization algorithm for LIIR.

Input: policy learning rate α and intrinsic reward learning rate β.

Output: policy parameters $\boldsymbol{\theta}$ and intrinsic reward parameters $\boldsymbol{\eta}$.

1: **Init**: initialize $\boldsymbol{\theta}$ and $\boldsymbol{\eta}$;
2: **while** termination is not reached **do**
3: Sample a trajectory $\mathcal{D} = \{\mathbf{s}_0, \mathbf{u}_0, \mathbf{s}_1, \mathbf{u}_1, \cdots\}$ by executing actions with the decentralized policies $\{\pi_{\theta_1}, \cdots, \pi_{\theta_n}\}$;
4: Update $\boldsymbol{\theta}$ according to (6) with learning rate α;
5: Compute (8) using new samples from $\{\pi_{\theta'_1}, \pi_{\theta'_2}, \cdots, \pi_{\theta'_n}\}$ or reuse $\mathcal{D}$ to replace (8) with $\frac{\nabla_{\theta'_i} \pi_{\theta'_i}(u_i|s_i)}{\pi_{\theta_i}(u_i|s_i)} A^{\text{ex}}(\boldsymbol{s}, \boldsymbol{u})$;
6: Update $\boldsymbol{\eta}$ according to (7), step 5 and (9) with learning rate β;
7: **end while**

Policy parameter update 식

6 $\theta'_i = \theta_i + \alpha \nabla_{\theta_i} \log \pi_{\theta_i}(u_i|s_i) A_i^{\text{proxy}}(s_i, u_i)$

$$A_i^{\text{proxy}}(s_i, u_i) = r_i^{\text{proxy}}(s_i, u_i) + V_{\varphi_i}^{\text{proxy}}(s'_i) - V_{\varphi_i}^{\text{proxy}}(s_i)$$

$V_{\varphi_i}^{\text{proxy}}$ is the proxy value parameterized by φ_i

Method

Algorithm 1 The optimization algorithm for LIIR.

Input: policy learning rate α and intrinsic reward learning rate β.

Output: policy parameters $\boldsymbol{\theta}$ and intrinsic reward parameters $\boldsymbol{\eta}$.

1: **Init**: initialize $\boldsymbol{\theta}$ and $\boldsymbol{\eta}$;
2: **while** termination is not reached **do**
3: Sample a trajectory $\mathcal{D} = \{\mathbf{s}_0, \mathbf{u}_0, \mathbf{s}_1, \mathbf{u}_1, \cdots\}$ by executing actions with the decentralized policies $\{\pi_{\theta_1}, \cdots, \pi_{\theta_n}\}$;
4: Update $\boldsymbol{\theta}$ according to (6) with learning rate α;
5: Compute (8) using new samples from $\{\pi_{\theta'_1}, \pi_{\theta'_2}, \cdots, \pi_{\theta'_n}\}$ or reuse $\mathcal{D}$ to replace (8) with $\frac{\nabla_{\theta'_i}\pi_{\theta'_i}(u_i|s_i)}{\pi_{\theta_i}(u_i|s_i)} A^{\text{ex}}(\boldsymbol{s}, \boldsymbol{u})$;
6: Update $\boldsymbol{\eta}$ according to (7), step 5 and (9) with learning rate β;
7: **end while**

Intrinsic reward parameter update 식

Proxy critic을 사용해 업데이트된 Policy parameter θ'_i로 chain rule를 적용

7

$$\nabla_{\eta_i} J^{\text{ex}} = \nabla_{\theta'_i} J^{\text{ex}} \nabla_{\eta_i} \theta'_i.$$

8 $$\nabla_{\theta'_i} J^{\text{ex}} = \nabla_{\theta'_i} \log \pi_{\theta'_i}(u_i|s_i) A^{\text{ex}}(\boldsymbol{s}, \boldsymbol{u})$$

$V_\phi^{\text{ex}}(s)$ is the extrinsic value parameterized by ϕ.

$$A^{\text{ex}}(\boldsymbol{s}, \boldsymbol{u}) = r^{\text{ex}}(\boldsymbol{s}, \boldsymbol{u}) + V_\phi^{\text{ex}}(\boldsymbol{s}') - V_\phi^{\text{ex}}(\boldsymbol{s})$$

Method

Algorithm 1 The optimization algorithm for LIIR.

Input: policy learning rate α and intrinsic reward learning rate β.

Output: policy parameters $\boldsymbol{\theta}$ and intrinsic reward parameters $\boldsymbol{\eta}$.

1: **Init**: initialize $\boldsymbol{\theta}$ and $\boldsymbol{\eta}$;
2: **while** termination is not reached **do**
3: Sample a trajectory $\mathcal{D} = \{\mathbf{s}_0, \mathbf{u}_0, \mathbf{s}_1, \mathbf{u}_1, \cdots\}$ by executing actions with the decentralized policies $\{\pi_{\theta_1}, \cdots, \pi_{\theta_n}\}$;
4: Update $\boldsymbol{\theta}$ according to (6) with learning rate α;
5: Compute (8) using new samples from $\{\pi_{\theta'_1}, \pi_{\theta'_2}, \cdots, \pi_{\theta'_n}\}$ or reuse $\mathcal{D}$ to replace (8) with $\frac{\nabla_{\theta'_i}\pi_{\theta'_i}(u_i|s_i)}{\pi_{\theta_i}(u_i|s_i)}A^{\text{ex}}(\boldsymbol{s}, \boldsymbol{u})$;
6: Update $\boldsymbol{\eta}$ according to (7), step 5 and (9) with learning rate β;
7: **end while**

Intrinsic reward parameter update 식

7 $$\nabla_{\eta_i} J^{\text{ex}} = \nabla_{\theta'_i} J^{\text{ex}} \nabla_{\eta_i} \theta'_i.$$

9 $$\begin{aligned}\nabla_{\eta_i}\theta'_i &= \nabla_{\eta_i}[\theta_i + \alpha\nabla_{\theta_i}\log\pi_{\theta_i}(u_i|s_i)A_i^{\text{proxy}}(s_i, u_i)] \\ &= \alpha\lambda\nabla_{\theta_i}\log\pi_{\theta_i}(a_i|s_i)\nabla_{\eta_i}r_i^{\text{proxy}}(s_i, u_i).\end{aligned}$$

$$A_i^{\text{proxy}}(s_i, u_i) = r_i^{\text{proxy}}(s_i, u_i) + V_{\varphi_i}^{\text{proxy}}(s'_i) - V_{\varphi_i}^{\text{proxy}}(s_i)$$

$V_{\varphi_i}^{\text{proxy}}$ is the proxy value parameterized by φ_i

Experiments

- 1D Pursuit game → intrinsic reward의 quality를 verification
  - 2개의 agent가 각각 x, y를 가짐
  - Target value z(unknown to agents)에 도달하기 위해 action으로 {-1, +1, 0}
  - Team reward : their value(x, y)의 sum과 target value의 차이
    - + 0.01 : both agents take actions that approaching the target value
    - - 0.01 : both agents take actions that moving away from the target value
    - Otherwise 0
  - 1000개의 에피소드에서 intrinsic reward average의 분포
    - Good: approaching the target
    - Bad : moving away from target
  - Good actions일수록, intrinsic reward를 높게 받음

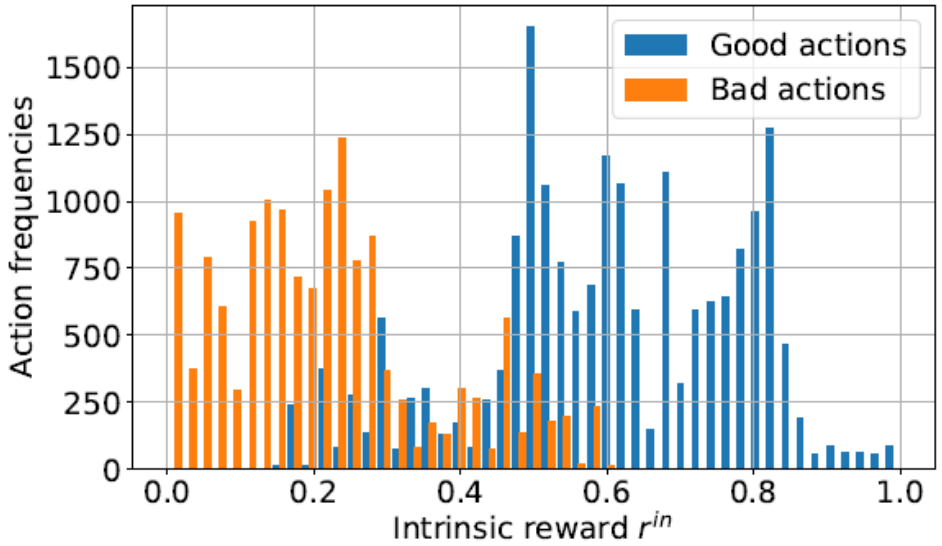


Figure 2: The distribution of the learned intrinsic rewards v.s. frequencies (counts) of taking "Good" and "Bad" actions from 1000 1D pursuit games.

Experiments

- StarCraft II Micromanagement(SMAC)
  - Winning rate
- CTDE
  - LIIR
  - COMA
  - QMIX
  - Central-V
- Independent
  - IQL
  - IAC

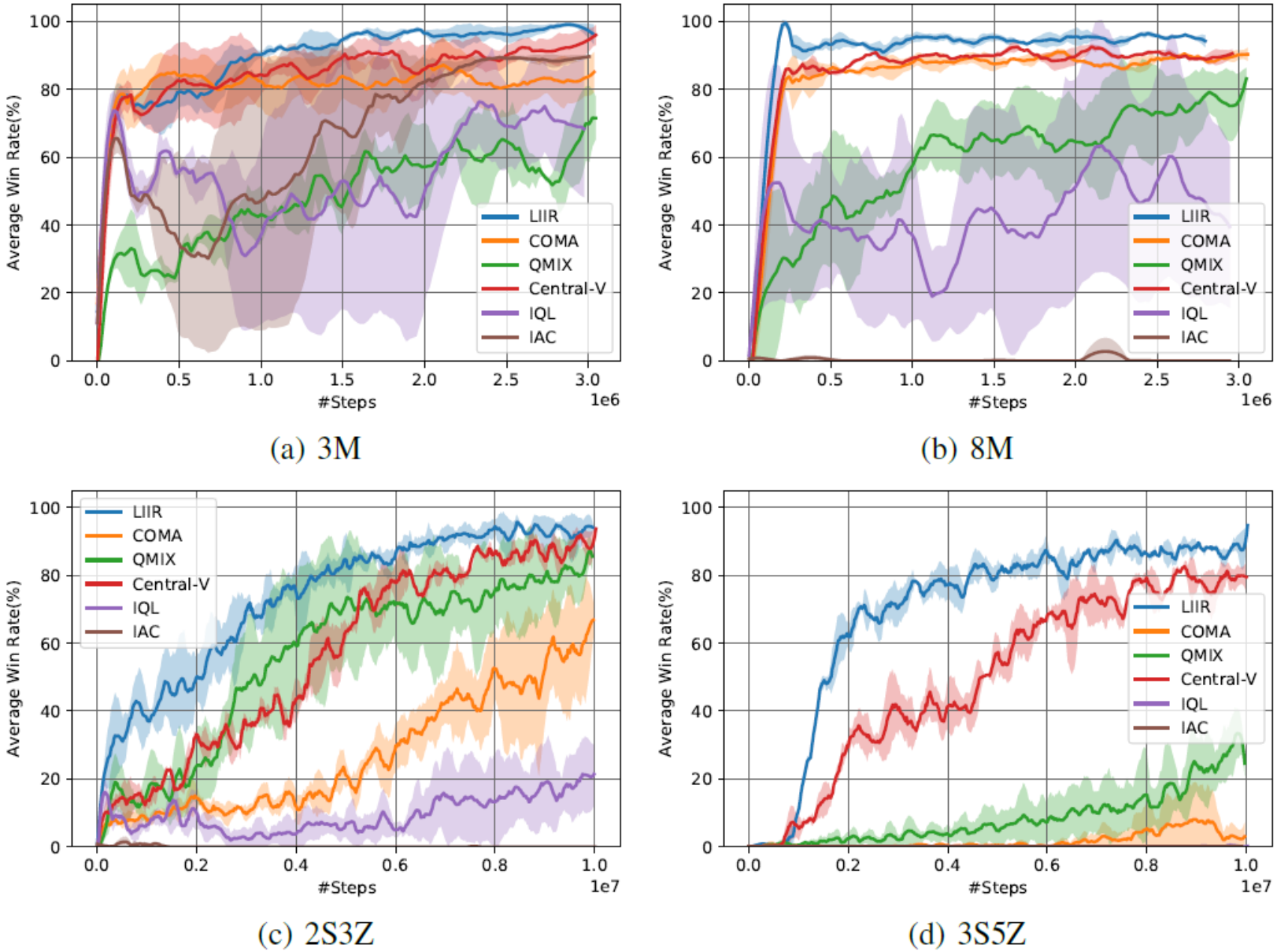


Figure 3: Test winning rates vs. training steps of various methods on all the scenarios.

Experiments

- Visualize the Learned Intrinsic reward

(a) Intrinsic reward (b) $t = 6$ (c) $t = 9$

Figure 4: An example of the intrinsic reward curves and auxiliary snapshots on 3M.

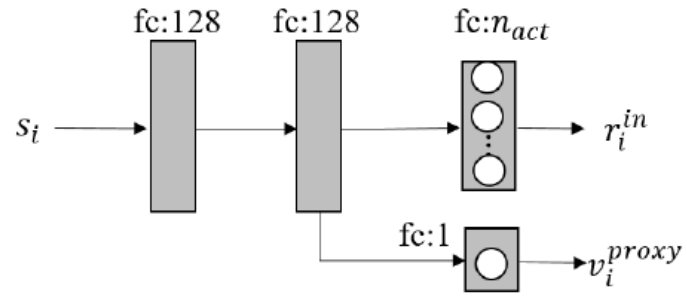
Agent 1
- t=9에서 죽음
- T=6에서 intrinsic reward가 낮음
  (hp가 낮은 상태로 공격 행동 진행)

(a) Intrinsic reward (b) $t = 27$ (c) $t = 32$

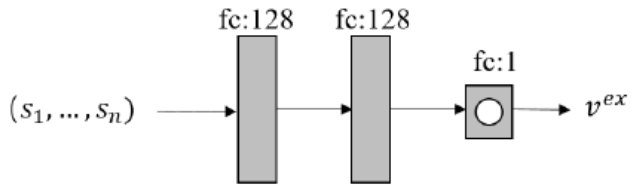
Figure 5: An example of the intrinsic reward curves and auxiliary snapshots on 2S3Z.

Agent 2
- t=27 intrinsic reward가 증가
  (hp가 낮은 상태로 도망감)

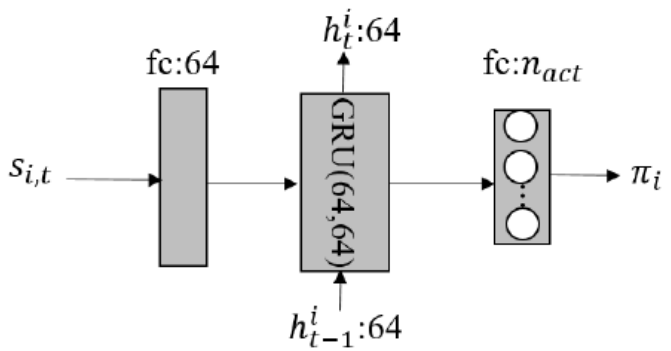
Appendix



(a) Proxy network

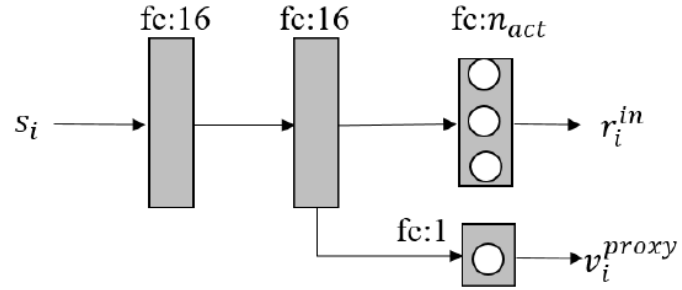


(b) Extrinsic critic

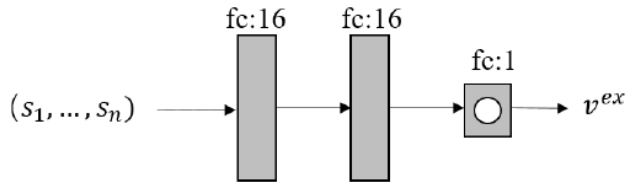


(c) Actor network

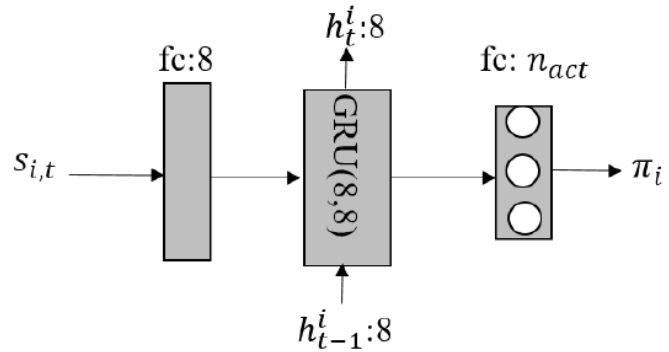
Figure 1: Neural network structures for Starcraft II.



(a) Proxy network



(b) Extrinsic critic



(c) Actor network

Figure 2: Neural network structures for *1D Pursuit*.

- independent Q-learning (IQL) [17]: IQL trains decentralized Q-functions for each agent. Since the observation and action spaces of the agents are the same within a specific environmental setting, a policy will be shared across all the agents;
- independent actor-critic (IAC) [9]: IAC is similar to IQL except that it adopts the actor-critic method;
- Central-V [9]: the method learns a centralized critic with decentralized policies. Similarly, all agents share the same policy network;
- COMA [9]: the method learns a centralized critic that is the state-action value minus a counterfactual baseline;
- QMIX [10]: the method learns decentralized Q-function for each agent with the assumption that the centralized Q-value is monotonically increasing with the individual Q-values. In the implementations, the agents share the same Q-function;
- LIIR: the proposed method. In the experiments, the agents share the same policy, intrinsic reward function and proxy critic. Since each agent has its own partial observation, sharing policy parameters does not imply that they act the same.