Multi-Agent Reinforcement Learning

Multi-Agent Deep Reinforcement Learning – Part 1

Stefano V. Albrecht, Filippos Christianos, Lukas Schäfer

The MARL Book

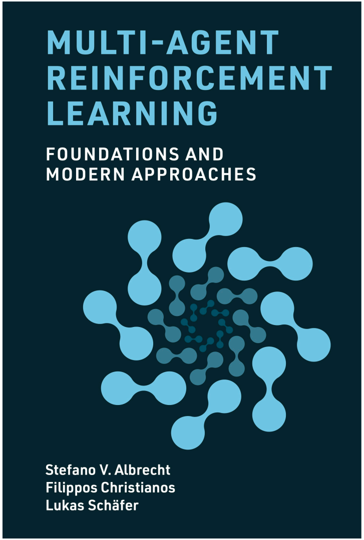
This lecture is based on

Multi-Agent Reinforcement Learning: Foundations and Modern Approaches

by Stefano V. Albrecht, Filippos Christianos and Lukas Schäfer

MIT Press, 2024

Download book, slides, and code at:
www.marl-book.com



Lecture Outline

- Training and execution modes
- Independent learning with deep reinforcement learning
- Multi-agent policy gradient algorithms
- Policy gradient algorithms
- Value decomposition in common-reward games

Training and Execution Modes

Training and Execution Modes

We often distinguish between training and execution modes in MARL:

- Training: what information is available to agents during learning?
- Execution: what information is available to agents for action selection?

Training and Execution Modes

We often distinguish between training and execution modes in MARL:

- **Training**: what information is available to agents during learning?
- **Execution**: what information is available to agents for action selection?

Reminder

We have already seen two single-agent reductions of the MARL problem:

Training and Execution Modes

We often distinguish between training and execution modes in MARL:

- **Training**: what information is available to agents during learning?
- **Execution**: what information is available to agents for action selection?

Reminder

We have already seen two single-agent reductions of the MARL problem:

- **Independent learning**: each agent learns its policy independently
  $\rightarrow$ decentralized training and decentralized execution

Training and Execution Modes

We often distinguish between training and execution modes in MARL:

- **Training**: what information is available to agents during learning?
- **Execution**: what information is available to agents for action selection?

Reminder

We have already seen two single-agent reductions of the MARL problem:

- **Independent learning**: each agent learns its policy independently
  $\rightarrow$ decentralized training and decentralized execution
- **Central learning**: learn single policy over the joint action space
  $\rightarrow$ centralized training and centralized execution

Training and Execution Modes

We often distinguish between training and execution modes in MARL:

- **Training**: what information is available to agents during learning?
- **Execution**: what information is available to agents for action selection?

Reminder

We have already seen two single-agent reductions of the MARL problem:

- **Independent learning**: each agent learns its policy independently
  $\rightarrow$ decentralized training and decentralized execution
- **Central learning**: learn single policy over the joint action space
  $\rightarrow$ centralized training and centralized execution

But we can also have centralised training with decentralised execution (CTDE)!

Independent Learning with Deep Reinforcement Learning

Independent Learning with Deep Reinforcement Learning

Reminder

In the independent learning framework, each agent i learns its policy π_i using only its local history of observations, treating the effects of other agents' actions as part of the environment.

- From the perspective of the individual agent, the environment transition function looks like this:

$$\mathcal{T}_i(s^{t+1}|s^t, a_i) \propto \sum_{a_{-i} \in A_{-i}} \mathcal{T}(s^{t+1}|s^t, \langle a_i, a_{-i} \rangle) \prod_{j \neq i} \pi_j(a_j|s^t)$$

Reminder

In the independent learning framework, each agent i learns its policy π_i using only its local history of observations, treating the effects of other agents' actions as part of the environment.

- From the perspective of the individual agent, the environment transition function looks like this:

$$\mathcal{T}_i(s^{t+1}|s^t, a_i) \propto \sum_{a_{-i} \in A_{-i}} \mathcal{T}(s^{t+1}|s^t, \langle a_i, a_{-i} \rangle) \prod_{j \neq i} \pi_j(a_j|s^t)$$

How about we do this with deep RL? We have already seen several single-agent deep RL algorithms: DQN, REINFORCE, A2C, PPO, etc.

Independent Learning with Deep Reinforcement Learning

Reminder

In the independent learning framework, each agent i learns its policy π_i using only its local history of observations, treating the effects of other agents' actions as part of the environment.

- From the perspective of the individual agent, the environment transition function looks like this:

$$\mathcal{T}_i(s^{t+1}|s^t, a_i) \propto \sum_{a_{-i} \in A_{-i}} \mathcal{T}(s^{t+1}|s^t, \langle a_i, a_{-i} \rangle) \prod_{j \neq i} \pi_j(a_j|s^t)$$

How about we do this with deep RL? We have already seen several single-agent deep RL algorithms: DQN, REINFORCE, A2C, PPO, etc.

Independent Deep Q-Networks

Algorithm Independent deep Q-networks

```
Initialize n value networks with random parameters θ_1, ..., θ_n
Initialize n target networks with parameters θ̄_1 = θ_1, ..., θ̄_n = θ_n
Initialize a replay buffer for each agent D_1, D_2, ..., D_n
for time step t = 0, 1, 2, ... do
    Collect current observations o_1^t, ..., o_n^t
    for agent i = 1, ..., n do
        With probability ε: choose random action a_i^t
        Otherwise: choose a_i^t ∈ arg max_{a_i} Q(h_i^t, a_i; θ_i)
    Apply actions (a_1^t, ..., a_n^t); collect rewards r_1^t, ..., r_n^t and next observations o_1^{t+1}, ..., o_n^{t+1}
    for agent i = 1, ..., n do
        Store transition (h_i^t, a_i^t, r_i^t, h_i^{t+1}) in replay buffers D_i
        Sample random mini-batch of B transitions (h_i^k, a_i^k, r_i^k, h_i^{k+1}) from D_i
        if s^{k+1} is terminal then
            Targets y_i^k ← r_i^k
        else
            Targets y_i^k ← r_i^k + γ max_{a'_i ∈ A_i} Q(h_i^{k+1}, a'_i; θ̄_i)
        Loss L(θ_i) ← (1/B) Σ_{k=1}^{B} ( y_i^k − Q(h_i^k, a_i^k; θ_i) )^2
        Update parameters θ_i by minimizing the loss L(θ_i)
        In a set interval, update target network parameters θ̄_i
```

Independent Deep Q-Networks

Algorithm Independent deep Q-networks

Initialize n value networks with random parameters $\theta_1, \ldots, \theta_n$
Initialize n target networks with parameters $\overline{\theta}_1 = \theta_1, \ldots, \overline{\theta}_n = \theta_n$
Initialize a replay buffer for each agent $D_1, D_2, \ldots, D_n$
for time step $t = 0, 1, 2, \ldots$ **do**
    Collect current observations $o_1^t, \ldots, o_n^t$
    for agent $i = 1, \ldots, n$ **do**
        With probability ϵ: choose random action a_i^t
        Otherwise: choose $a_i^t \in \arg\max_{a_i} Q(h_i^t, a_i; \theta_i)$
    Apply actions $(a_1^t, \ldots, a_n^t)$; collect rewards $r_1^t, \ldots, r_n^t$ and next observations $o_1^{t+1}, \ldots, o_n^{t+1}$
    for agent $i = 1, \ldots, n$ **do**
        Store transition $(h_i^t, a_i^t, r_i^t, h_i^{t+1})$ in replay buffers D_i
        Sample random mini-batch of B transitions $(h_i^k, a_i^k, r_i^k, h_i^{k+1})$ from D_i
        if s^{k+1} is terminal **then**
            Targets $y_i^k \leftarrow r_i^k$
        else
            Targets $y_i^k \leftarrow r_i^k + \gamma \max_{a_i' \in A_i} Q(h_i^{k+1}, a_i'; \overline{\theta_i})$
        Loss $\mathcal{L}(\theta_i) \leftarrow \frac{1}{B} \sum_{k=1}^{B} \left(y_i^k - Q(h_i^k, a_i^k; \theta_i) \right)^2$
        Update parameters θ_i by minimizing the loss $\mathcal{L}(\theta_i)$
        In a set interval, update target network parameters $\overline{\theta}_i$

- Almost identical to DQN from Chapter 8 but with n agents!

Independent Deep Q-Networks

Algorithm Independent deep Q-networks

Initialize n value networks with random parameters $\theta_1, \ldots, \theta_n$
Initialize n target networks with parameters $\overline{\theta}_1 = \theta_1, \ldots, \overline{\theta}_n = \theta_n$
Initialize a replay buffer for each agent $D_1, D_2, \ldots, D_n$
for time step $t = 0, 1, 2, \ldots$ **do**
  Collect current observations $o_1^t, \ldots, o_n^t$
  for agent $i = 1, \ldots, n$ **do**
    With probability ϵ: choose random action a_i^t
    Otherwise: choose $a_i^t \in \arg\max_{a_i} Q(h_i^t, a_i; \theta_i)$
  Apply actions $(a_1^t, \ldots, a_n^t)$; collect rewards $r_1^t, \ldots, r_n^t$ and next observations $o_1^{t+1}, \ldots, o_n^{t+1}$
  for agent $i = 1, \ldots, n$ **do**
    Store transition $(h_i^t, a_i^t, r_i^t, h_i^{t+1})$ in replay buffers D_i
    Sample random mini-batch of B transitions $(h_i^k, a_i^k, r_i^k, h_i^{k+1})$ from D_i
    if s^{k+1} is terminal **then**
      Targets $y_i^k \leftarrow r_i^k$
    else
      Targets $y_i^k \leftarrow r_i^k + \gamma \max_{a_i' \in A_i} Q(h_i^{k+1}, a_i'; \overline{\theta}_i)$
    Loss $\mathcal{L}(\theta_i) \leftarrow \frac{1}{B} \sum_{k=1}^{B} \left(y_i^k - Q(h_i^k, a_i^k; \theta_i) \right)^2$
    Update parameters θ_i by minimizing the loss $\mathcal{L}(\theta_i)$
    In a set interval, update target network parameters $\overline{\theta}_i$

- Almost identical to DQN from Chapter 8 but with n agents!
- Replay buffer contains off-policy experiences due to changing policies
- In MARL, the policies of all agents are changing → training can be unstable

Independent Advantage Actor-Critic

Algorithm Independent A2C with synchronous environments

Initialize n actor networks with random parameters $\phi_1, \ldots, \phi_n$
Initialize n critic networks with random parameters $\theta_1, \ldots, \theta_n$
Initialize K parallel environments
for time step $t = 0 \ldots$ **do**
  Batch of observations for each agent and environment: $\begin{bmatrix} o_1^{t,1} \ldots o_1^{t,K} \\ \ddots \\ o_n^{t,1} \ldots o_n^{t,K} \end{bmatrix}$
  Sample actions $\begin{bmatrix} a_1^{t,1} \ldots a_1^{t,K} \\ \ddots \\ a_n^{t,1} \ldots a_n^{t,K} \end{bmatrix} \sim \pi(\cdot \mid h_1^t; \phi_1), \ldots, \pi(\cdot \mid h_n^t; \phi_n)$
  Apply actions; collect rewards $\begin{bmatrix} r_1^{t,1} \ldots r_1^{t,K} \\ \ddots \\ r_n^{t,1} \ldots r_n^{t,K} \end{bmatrix}$ and observations $\begin{bmatrix} o_1^{t+1,1} \ldots o_1^{t+1,K} \\ \ddots \\ o_n^{t+1,1} \ldots o_n^{t+1,K} \end{bmatrix}$
  for agent $i = 1, \ldots, n$ **do**
    if $s^{t+1,k}$ is terminal **then**
      Advantage $Adv(h_i^{t,k}, a_i^{t,k}) \leftarrow r_i^{t,k} - V(h_i^{t,k}; \theta_i)$
      Critic target $y_i^{t,k} \leftarrow r_i^{t,k}$
    else
      Advantage $Adv(h_i^{t,k}, a_i^{t,k}) \leftarrow r_i^{t,k} + \gamma V(h_i^{t+1,k}; \theta_i) - V(h_i^{t,k}; \theta_i)$
      Critic target $y_i^{t,k} \leftarrow r_i^{t,k} + \gamma V(h_i^{t+1,k}; \theta_i)$
    Actor loss $\mathcal{L}(\phi_i) \leftarrow \frac{1}{K} \sum_{k=1}^{K} Adv(h_i^{t,k}, a_i^{t,k}) \log \pi(a_i^{t,k} \mid h_i^{t,k}; \phi_i)$
    Critic loss $\mathcal{L}(\theta_i) \leftarrow \frac{1}{K} \sum_{k=1}^{K} \left(y_i^{t,k} - V(h_i^{t,k}; \theta_i) \right)^2$
    Update parameters ϕ_i by minimizing the actor loss $\mathcal{L}(\phi_i)$
    Update parameters θ_i by minimizing the critic loss $\mathcal{L}(\theta_i)$

Independent Advantage Actor-Critic

Algorithm Independent A2C with synchronous environments

Initialize n actor networks with random parameters $\phi_1, \ldots, \phi_n$
Initialize n critic networks with random parameters $\theta_1, \ldots, \theta_n$
Initialize K parallel environments
for time step $t = 0 \ldots$ **do**
    Batch of observations for each agent and environment: $\begin{bmatrix} o_1^{t,1} \ldots o_1^{t,K} \\ \ddots \\ o_n^{t,1} \ldots o_n^{t,K} \end{bmatrix}$
    Sample actions $\begin{bmatrix} a_1^{t,1} \ldots a_1^{t,K} \\ \ddots \\ a_n^{t,1} \ldots a_n^{t,K} \end{bmatrix} \sim \pi(\cdot \mid h_1^t; \phi_1), \ldots, \pi(\cdot \mid h_n^t; \phi_n)$
    Apply actions; collect rewards $\begin{bmatrix} r_1^{t,1} \ldots r_1^{t,K} \\ \ddots \\ r_n^{t,1} \ldots r_n^{t,K} \end{bmatrix}$ and observations $\begin{bmatrix} o_1^{t+1,1} \ldots o_1^{t+1,K} \\ \ddots \\ o_n^{t+1,1} \ldots o_n^{t+1,K} \end{bmatrix}$
    for agent $i = 1, \ldots, n$ **do**
        if $s^{t+1,k}$ is terminal **then**
            Advantage $Adv(h_i^{t,k}, a_i^{t,k}) \leftarrow r_i^{t,k} - V(h_i^{t,k}; \theta_i)$
            Critic target $y_i^{t,k} \leftarrow r_i^{t,k}$
        else
            Advantage $Adv(h_i^{t,k}, a_i^{t,k}) \leftarrow r_i^{t,k} + \gamma V(h_i^{t+1,k}; \theta_i) - V(h_i^{t,k}; \theta_i)$
            Critic target $y_i^{t,k} \leftarrow r_i^{t,k} + \gamma V(h_i^{t+1,k}; \theta_i)$
        Actor loss $\mathcal{L}(\phi_i) \leftarrow \frac{1}{K} \sum_{k=1}^{K} Adv(h_i^{t,k}, a_i^{t,k}) \log \pi(a_i^{t,k} \mid h_i^{t,k}; \phi_i)$
        Critic loss $\mathcal{L}(\theta_i) \leftarrow \frac{1}{K} \sum_{k=1}^{K} \left(y_i^{t,k} - V(h_i^{t,k}; \theta_i)\right)^2$
        Update parameters ϕ_i by minimizing the actor loss $\mathcal{L}(\phi_i)$
        Update parameters θ_i by minimizing the critic loss $\mathcal{L}(\theta_i)$

- Almost identical to single-agent A2C from Chapter 8

Independent Advantage Actor-Critic

Algorithm Independent A2C with synchronous environments

Initialize n actor networks with random parameters $\phi_1, \ldots, \phi_n$
Initialize n critic networks with random parameters $\theta_1, \ldots, \theta_n$
Initialize K parallel environments
for time step $t = 0 \ldots$ **do**
  Batch of observations for each agent and environment: $\begin{bmatrix} o_1^{t,1} \ldots o_1^{t,K} \\ \ddots \\ o_n^{t,1} \ldots o_n^{t,K} \end{bmatrix}$
  Sample actions $\begin{bmatrix} a_1^{t,1} \ldots a_1^{t,K} \\ \ddots \\ a_n^{t,1} \ldots a_n^{t,K} \end{bmatrix} \sim \pi(\cdot \mid h_1^t; \phi_1), \ldots, \pi(\cdot \mid h_n^t; \phi_n)$
  Apply actions; collect rewards $\begin{bmatrix} r_1^{t,1} \ldots r_1^{t,K} \\ \ddots \\ r_n^{t,1} \ldots r_n^{t,K} \end{bmatrix}$ and observations $\begin{bmatrix} o_1^{t+1,1} \ldots o_1^{t+1,K} \\ \ddots \\ o_n^{t+1,1} \ldots o_n^{t+1,K} \end{bmatrix}$
  for agent $i = 1, \ldots, n$ **do**
    if $s^{t+1,k}$ is terminal **then**
      Advantage $Adv(h_i^{t,k}, a_i^{t,k}) \leftarrow r_i^{t,k} - V(h_i^{t,k}; \theta_i)$
      Critic target $y_i^{t,k} \leftarrow r_i^{t,k}$
    else
      Advantage $Adv(h_i^{t,k}, a_i^{t,k}) \leftarrow r_i^{t,k} + \gamma V(h_i^{t+1,k}; \theta_i) - V(h_i^{t,k}; \theta_i)$
      Critic target $y_i^{t,k} \leftarrow r_i^{t,k} + \gamma V(h_i^{t+1,k}; \theta_i)$
    Actor loss $\mathcal{L}(\phi_i) \leftarrow \frac{1}{K} \sum_{k=1}^{K} Adv(h_i^{t,k}, a_i^{t,k}) \log \pi(a_i^{t,k} \mid h_i^{t,k}; \phi_i)$
    Critic loss $\mathcal{L}(\theta_i) \leftarrow \frac{1}{K} \sum_{k=1}^{K} \left(y_i^{t,k} - V(h_i^{t,k}; \theta_i) \right)^2$
    Update parameters ϕ_i by minimizing the actor loss $\mathcal{L}(\phi_i)$
    Update parameters θ_i by minimizing the critic loss $\mathcal{L}(\theta_i)$

- Almost identical to single-agent A2C from Chapter 8
- Similar adaptation can be done for independent REINFORCE and independent PPO

Challenges of Multi-Agent Reinforcement Learning

Reminder

MARL algorithms suffer from multi-agent specific challenges:

- **Non-stationarity**: exacerbated due to changing policies of all agents
- **Equilibrium selection**: how to converge to a stable equilibrium?
- **Multi-agent credit assignment**: how to attribute rewards to agents' actions?
- **Scaling to many agents**: how to efficiently scale to large numbers of agents?

Challenges of Multi-Agent Reinforcement Learning

Reminder

MARL algorithms suffer from multi-agent specific challenges:

- **Non-stationarity**: exacerbated due to changing policies of all agents
- **Equilibrium selection**: how to converge to a stable equilibrium?
- **Multi-agent credit assignment**: how to attribute rewards to agents' actions?
- **Scaling to many agents**: how to efficiently scale to large numbers of agents?

Centralised training with decentralised execution (CTDE) can help address some of these challenges.

Multi-Agent Policy Gradient Algorithms

The Policy-Gradient Theorem

Reminder

Follow this gradient to optimise the parameters ϕ of the policy π to maximise the expected return:

$$
\begin{aligned}
\nabla_\phi J(\phi) &\propto \sum_{s \in S} \Pr(s \mid \pi) \sum_{a \in A} Q^\pi(s, a) \nabla_\phi \pi(a \mid s; \phi) \\
&= \mathbb{E}_{s \sim \Pr(\cdot \mid \pi), a \sim \pi(\cdot \mid s; \phi)} \left[Q^\pi(s, a) \nabla_\phi \log \pi(a \mid s; \phi) \right]
\end{aligned}
$$

The Policy-Gradient Theorem

Reminder

Follow this gradient to optimise the parameters ϕ of the policy π to maximise the expected return:

$$
\begin{aligned}
\nabla_\phi J(\phi) &\propto \sum_{s \in S} \Pr(s \mid \pi) \sum_{a \in A} Q^\pi(s,a) \nabla_\phi \pi(a \mid s; \phi) \\
&= \mathbb{E}_{s \sim \Pr(\cdot \mid \pi), a \sim \pi(\cdot \mid s; \phi)} \left[Q^\pi(s,a) \nabla_\phi \log \pi(a \mid s; \phi) \right]
\end{aligned}
$$

Does this also hold for MARL? Yes, with minor modifications!

The Multi-Agent Policy-Gradient Theorem

Solution

In MARL, the expected returns of agent i under its policy π_i depends on the policies of all other agents π_{-i} $\rightarrow$ the multi-agent policy gradient theorem defines an expectation over the policies of **all** agents:

The Multi-Agent Policy-Gradient Theorem

Solution

In MARL, the expected returns of agent i under its policy π_i depends on the policies of all other agents π_{-i} $\rightarrow$ the multi-agent policy gradient theorem defines an expectation over the policies of **all** agents:

$$\nabla_{\phi_i} J(\phi_i) \propto \mathbb{E}_{\hat{h} \sim \Pr(\hat{h}|\pi), a_i \sim \pi_i, a_{-i} \sim \pi_{-i}} \left[Q_i^\pi(\hat{h}, \langle a_i, a_{-i} \rangle) \nabla_{\phi_i} \log \pi_i(a_i \mid h_i = \sigma_i(\hat{h}); \phi_i) \right]$$

The Multi-Agent Policy-Gradient Theorem

Solution

In MARL, the expected returns of agent i under its policy π_i depends on the policies of all other agents π_{-i} $\rightarrow$ the multi-agent policy gradient theorem defines an expectation over the policies of **all** agents:

$$\nabla_{\phi_i} J(\phi_i) \propto \mathbb{E}_{\hat{h} \sim \Pr(\hat{h} \mid \pi), a_i \sim \pi_i, a_{-i} \sim \pi_{-i}} \left[Q_i^{\pi}(\hat{h}, \langle a_i, a_{-i} \rangle) \nabla_{\phi_i} \log \pi_i(a_i \mid h_i = \sigma_i(\hat{h}); \phi_i) \right]$$

$\rightarrow$ Derive policy update rules by finding estimators for expected returns $Q_i^{\pi}(\hat{h}, \langle a_i, a_{-i} \rangle)$.

The Multi-Agent Policy-Gradient Theorem

Solution

In MARL, the expected returns of agent i under its policy π_i depends on the policies of all other agents π_{-i} $\rightarrow$ the multi-agent policy gradient theorem defines an expectation over the policies of **all** agents:

$$\nabla_{\phi_i} J(\phi_i) \propto \mathbb{E}_{\hat{h} \sim \Pr(\hat{h}|\pi), a_i \sim \pi_i, a_{-i} \sim \pi_{-i}} \left[Q_i^{\pi}(\hat{h}, \langle a_i, a_{-i} \rangle) \nabla_{\phi_i} \log \pi_i(a_i \mid h_i = \sigma_i(\hat{h}); \phi_i) \right]$$

$\rightarrow$ Derive policy update rules by finding estimators for expected returns $Q_i^{\pi}(\hat{h}, \langle a_i, a_{-i} \rangle)$.

We have already seen independent A2C that estimates $Adv(h_i, a_i) \propto Q_i^{\pi}(\hat{h}, \langle a_i, a_{-i} \rangle)$.

The Multi-Agent Policy-Gradient Theorem

Solution

In MARL, the expected returns of agent i under its policy π_i depends on the policies of all other agents π_{-i} $\rightarrow$ the multi-agent policy gradient theorem defines an expectation over the policies of **all** agents:

$$\nabla_{\phi_i} J(\phi_i) \propto \mathbb{E}_{\hat{h}\sim \Pr(\hat{h}|\pi), a_i \sim \pi_i, a_{-i}\sim \pi_{-i}} \left[Q_i^{\pi}(\hat{h}, \langle a_i, a_{-i}\rangle) \nabla_{\phi_i} \log \pi_i(a_i \mid h_i = \sigma_i(\hat{h}); \phi_i) \right]$$

$\rightarrow$ Derive policy update rules by finding estimators for expected returns $Q_i^{\pi}(\hat{h}, \langle a_i, a_{-i}\rangle)$.

We have already seen independent A2C that estimates $Adv(h_i, a_i) \propto Q_i^{\pi}(\hat{h}, \langle a_i, a_{-i}\rangle)$.

But can we do better? Perhaps by leveraging more information?

Note

In actor-critic algorithms, only the policy/actor is used during execution and the critic is used only during training $\rightarrow$ the critic can be conditioned on centralised information z without compromising decentralised execution.

Centralized Critics

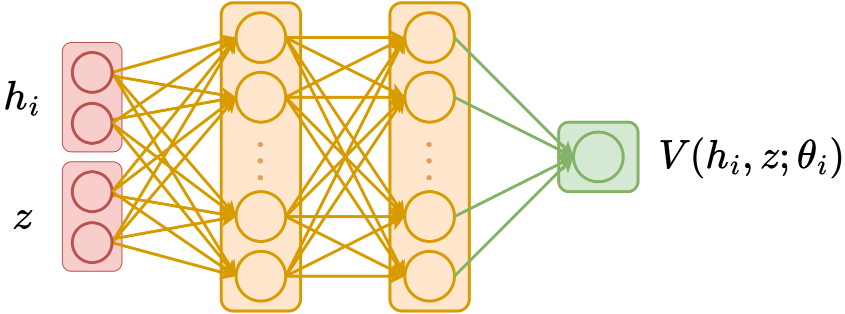
Note

In actor-critic algorithms, only the policy/actor is used during execution and the critic is used only during training $\rightarrow$ the critic can be conditioned on centralised information z without compromising decentralised execution.

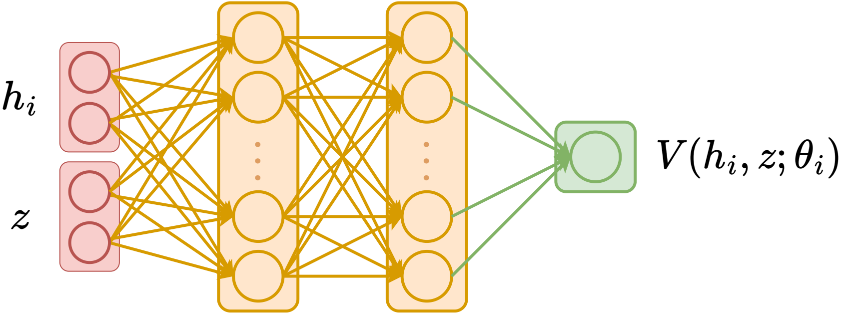
This might include:

- Global state s
- Joint action a
- Joint observation history h
- ...

Centralized Critics



Centralized Critics



Now we can integrate centralized critics into multi-agent policy gradient algorithms.

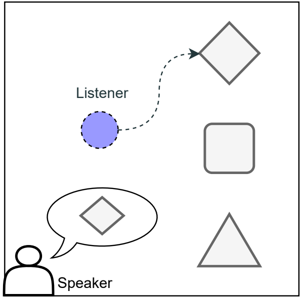
Centralized Advantage Actor-Critic

Algorithm Centralized A2C with synchronous environments

Initialize n actor networks with random parameters $\phi_1, \ldots, \phi_n$
Initialize n critic networks with random parameters $\theta_1, \ldots, \theta_n$
Initialize K parallel environments
for time step $t = 0 \ldots$ **do**
  Batch of observations for each agent and environment: $\begin{bmatrix} o_1^{t,1} \ldots o_1^{t,K} \\ \ddots \\ o_n^{t,1} \ldots o_n^{t,K} \end{bmatrix}$
  Batch of centralized information for each environment: $\begin{bmatrix} z^{t,1} \ldots z^{t,K} \end{bmatrix}$
  Sample actions $\begin{bmatrix} a_1^{t,1} \ldots a_1^{t,K} \\ \ddots \\ a_n^{t,1} \ldots a_n^{t,K} \end{bmatrix} \sim \pi(\cdot \mid h_1^t; \phi_1), \ldots, \pi(\cdot \mid h_n^t; \phi_n)$
  Apply actions; collect rewards $\begin{bmatrix} r_1^{t,1} \ldots r_1^{t,K} \\ \ddots \\ r_n^{t,1} \ldots r_n^{t,K} \end{bmatrix}$, observations $\begin{bmatrix} o_1^{t+1,1} \ldots o_1^{t+1,K} \\ \ddots \\ o_n^{t+1,1} \ldots o_n^{t+1,K} \end{bmatrix}$, and centralized information $\begin{bmatrix} z^{t+1,1} \ldots z^{t+1,K} \end{bmatrix}$
  for agent $i = 1, \ldots, n$ **do**
    if $s^{t+1,k}$ is terminal **then**
      $Adv(h_i^{t,k}, z^{t,k}, a_i^{t,k}) \leftarrow r_i^{t,k} - V(h_i^{t,k}, z^{t,k}; \theta_i)$
      Critic target $y_i^{t,k} \leftarrow r_i^{t,k}$
    else
      $Adv(h_i^{t,k}, z^{t,k}, a_i^{t,k}) \leftarrow r_i^{t,k} + \gamma V(h_i^{t+1,k}, z^{t+1,k}; \theta_i) - V(h_i^{t,k}, z^{t,k}; \theta_i)$
      Critic target $y_i^{t,k} \leftarrow r_i^{t,k} + \gamma V(h_i^{t+1,k}, z^{t+1,k}; \theta_i)$
    Actor loss $\mathcal{L}(\phi_i) \leftarrow \frac{1}{K} \sum_{k=1}^{K} Adv(h_i^{t,k}, z^{t,k}, a_i^{t,k}) \log \pi(a_i^{t,k} \mid h_i^{t,k}; \phi_i)$
    Critic loss $\mathcal{L}(\theta_i) \leftarrow \frac{1}{K} \sum_{k=1}^{K} \left(y_i^{t,k} - V(h_i^{t,k}, z^{t,k}; \theta_i)\right)^2$
    Update parameters ϕ_i by minimizing the actor loss $\mathcal{L}(\phi_i)$
    Update parameters θ_i by minimizing the critic loss $\mathcal{L}(\theta_i)$

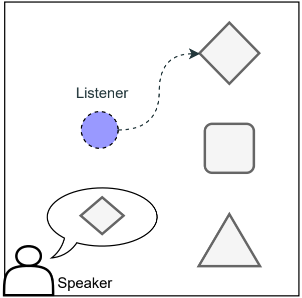
- Simple extension of independent A2C.
- Centralized information z is added to the critic input.

Centralized Critics in Action

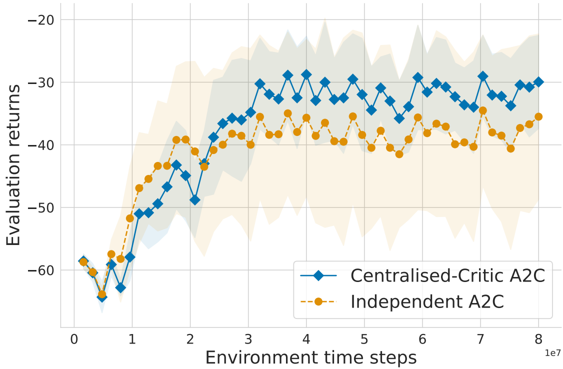


(a) Speaker-listener game

Centralized Critics in Action



(a) Speaker-listener game

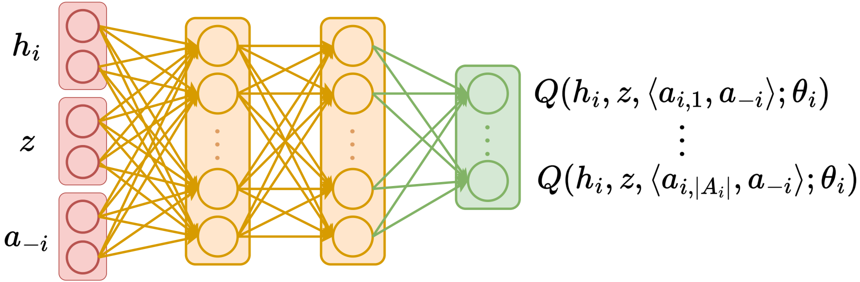


(b) Training curves

Agents with centralized critics converge to higher returns than agents with independent critics in the partially observable speaker-listener game.

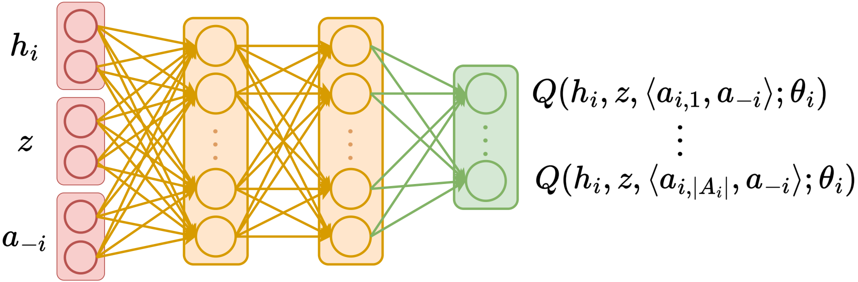
Centralized Action-Value Critics

Similarly, we can learn an action-value function that receives additional centralized information z.



Centralized Action-Value Critics

Similarly, we can learn an action-value function that receives additional centralized information z.



But what for? The centralized action-value critic can reason about the joint action space!

Counterfactual Multi-Agent Policy Gradient

For example, we can compute a counterfactual advantage for agent i

$$Adv_i(h_i, z, a) = Q(h_i, z, a; \theta) - \underbrace{\sum_{a'_i \in A_i} \pi(a'_i \mid h_i; \phi_i) Q(h_i, z, \langle a'_i, a_{-i} \rangle; \theta)}_{\text{counterfactual baseline}}$$

with the following components:

Counterfactual Multi-Agent Policy Gradient

For example, we can compute a counterfactual advantage for agent i

$$Adv_i(h_i, z, a) = Q(h_i, z, a; \theta) - \underbrace{\sum_{a'_i \in A_i} \pi(a'_i \mid h_i; \phi_i) Q(h_i, z, \langle a'_i, a_{-i} \rangle; \theta)}_{\text{counterfactual baseline}}$$

with the following components:

- $Q(h_i, z, a; \theta)$: expected returns when applying joint action $a \rightarrow$ agent i applies action a_i and all other agents apply actions a_{-i}

Counterfactual Multi-Agent Policy Gradient

For example, we can compute a counterfactual advantage for agent i

$$Adv_i(h_i, z, a) = Q(h_i, z, a; \theta) - \underbrace{\sum_{a'_i \in A_i} \pi(a'_i \mid h_i; \phi_i) Q(h_i, z, \langle a'_i, a_{-i} \rangle; \theta)}_{\text{counterfactual baseline}}$$

with the following components:

- $Q(h_i, z, a; \theta)$: expected returns when applying joint action $a \rightarrow$ agent i applies action a_i and all other agents apply actions a_{-i}
- Counterfactual baseline: expected returns when agent i samples its action a'_i from its policy $\pi(\cdot; \phi_i)$ and all other agents apply their actual actions a_{-i}

Counterfactual Multi-Agent Policy Gradient

For example, we can compute a counterfactual advantage for agent i

$$Adv_i(h_i, z, a) = Q(h_i, z, a; \theta) - \underbrace{\sum_{a'_i \in A_i} \pi(a'_i \mid h_i; \phi_i) Q(h_i, z, \langle a'_i, a_{-i} \rangle; \theta)}_{\text{counterfactual baseline}}$$

with the following components:

- $Q(h_i, z, a; \theta)$: expected returns when applying joint action $a \rightarrow$ agent i applies action a_i and all other agents apply actions a_{-i}
- Counterfactual baseline: expected returns when agent i samples its action a'_i from its policy $\pi(\cdot; \phi_i)$ and all other agents apply their actual actions a_{-i}

Identify contribution of agent i's action a_i to received rewards $\Rightarrow$ help to address the credit assignment problem in common-reward games

The Equilibrium Selection Problem

Problem

Many multi-agent games have multiple equilibria. In such games, it difficult for all agents to agree on and stably converge to a single equilibrium. This is known as the equilibrium selection problem.

The Equilibrium Selection Problem

Problem

Many multi-agent games have multiple equilibria. In such games, it difficult for all agents to agree on and stably converge to a single equilibrium. This is known as the equilibrium selection problem.

	A	B
A	4,4‡	0,3
B	3,0	2,2†

Figure: Stag Hunt

	A	B	C
A	11‡	-30	0
B	-30	7†	0
C	0	6	5

Figure: Climbing

The Stag Hunt and Climbing matrix games have multiple equilibria.

†: Pareto-dominated equilibria

‡: Pareto-optimal equilibria

Example for the Equilibrium Selection Problem

	A	B	C
A	11‡	-30	0
B	-30	7†	0
C	0	6	5

Figure: Climbing game

- Pareto-optimal equilibrium (‡): (A, A) with +11
- Pareto-dominated equilibrium (†): (B, B) with +7

Example for the Equilibrium Selection Problem

	A	B	C
A	11‡	-30	0
B	-30	7†	0
C	0	6	5

Figure: Climbing game

- Pareto-optimal equilibrium (‡): (A, A) with +11
- Pareto-dominated equilibrium (†): (B, B) with +7

- All agents prefer the Pareto-optimal equilibrium
- **But** deviation from the equilibrium by any agent results in lower returns $\rightarrow$ e.g. risk of receiving -30 if one agent deviates from A to B

Example for the Equilibrium Selection Problem

	A	B	C
A	11‡	-30	0
B	-30	7†	0
C	0	6	5

Figure: Climbing game

- Pareto-optimal equilibrium (‡): (A, A) with +11
- Pareto-dominated equilibrium (†): (B, B) with +7

- All agents prefer the Pareto-optimal equilibrium
- **But** deviation from the equilibrium by any agent results in lower returns $\rightarrow$ e.g. risk of receiving -30 if one agent deviates from A to B

Problem

Due to the risk of deviations, agents often converge to the safer Pareto-dominated equilibrium or even the suboptimal solution (C, C) with +5.

Example for the Equilibrium Selection Problem

	A	B	C
A	11‡	-30	0
B	-30	7†	0
C	0	6	5

Figure: Climbing game

- Pareto-optimal equilibrium (‡): (A, A) with +11
- Pareto-dominated equilibrium (†): (B, B) with +7

- All agents prefer the Pareto-optimal equilibrium
- **But** deviation from the equilibrium by any agent results in lower returns → e.g. risk of receiving -30 if one agent deviates from A to B

Problem

Due to the risk of deviations, agents often converge to the safer Pareto-dominated equilibrium or even the suboptimal solution (C, C) with +5.

How can we overcome this problem and robustly converge to the optimal equilibrium?

Pareto Actor-Critic for Equilibrium Selection

Both shown matrix games are no-conflict games where agents agree on the optimal policy:

$$\arg\max_{\pi} U_i(\pi) = \arg\max_{\pi} U_j(\pi) \quad \forall i, j \in I$$

Pareto Actor-Critic for Equilibrium Selection

Both shown matrix games are no-conflict games where agents agree on the optimal policy:

$$\arg\max_{\pi} U_i(\pi) = \arg\max_{\pi} U_j(\pi) \quad \forall i, j \in I$$

We can use this property! Agent i during training assumes that all other agents follow the policy π^+_{-i} that is best for agent i, i.e. $\pi^+_{-i} \in \arg\max_{\pi_{-i}} U_i(\pi_i, \pi_{-i})$.

Pareto Actor-Critic for Equilibrium Selection

Both shown matrix games are no-conflict games where agents agree on the optimal policy:

$$\arg\max_{\pi} U_i(\pi) = \arg\max_{\pi} U_j(\pi) \quad \forall i, j \in I$$

We can use this property! Agent i during training assumes that all other agents follow the policy π^{+}_{-i} that is best for agent i, i.e. $\pi^{+}_{-i} \in \arg\max_{\pi_{-i}} U_i(\pi_i, \pi_{-i})$.

We can compute π^{+}_{-i} using a centralized critic that receives the joint action a as input:

$$\pi^{+}_{-i} \in \arg\max_{a_{-i}} Q(h_i^t, z^t, \langle a_i^t, a_{-i} \rangle)$$

Pareto Actor-Critic for Equilibrium Selection

Both shown matrix games are no-conflict games where agents agree on the optimal policy:

$$\arg\max_{\pi} U_i(\pi) = \arg\max_{\pi} U_j(\pi) \quad \forall i, j \in I$$

We can use this property! Agent i during training assumes that all other agents follow the policy π^+_{-i} that is best for agent i, i.e. $\pi^+_{-i} \in \arg\max_{\pi_{-i}} U_i(\pi_i, \pi_{-i})$.

We can compute π^+_{-i} using a centralized critic that receives the joint action a as input:

$$\pi^+_{-i} \in \arg\max_{a_{-i}} Q(h_i^t, z^t, \langle a_i^t, a_{-i} \rangle)$$

During training, agent i optimises its policy π_i by minimising the following loss:

$$\mathcal{L}(\phi_i) = -\mathbb{E}_{a_i^t \sim \pi_i, a_{-i}^t \sim \pi^+_{-i}} \left[\log \pi(a_i^t \mid h_i^t; \phi_i) \left(Q^{\pi^+}(h_i^t, z^t, \langle a_i^t, a_{-i}^t \rangle; \theta_i^q) - V^{\pi^+}(h_i^t, z^t; \theta_i^v) \right) \right]$$

Pareto Actor-Critic for Equilibrium Selection in Climbing Game

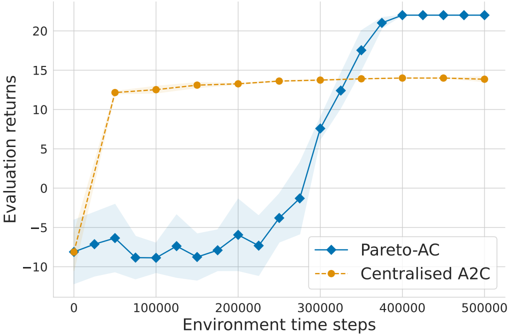


Figure: Learning curves in the Climbing game.

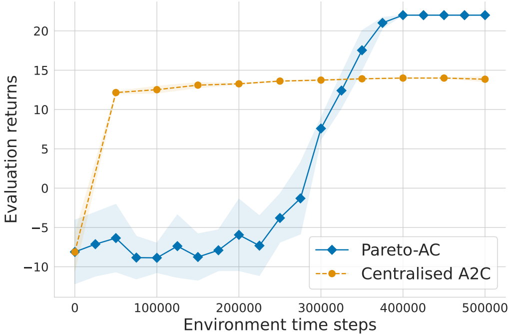


Figure: Learning curves in the Climbing game.

- A2C with centralized state-value critic converges to the Pareto-dominated equilibrium (B, B) with +7 (per agent).

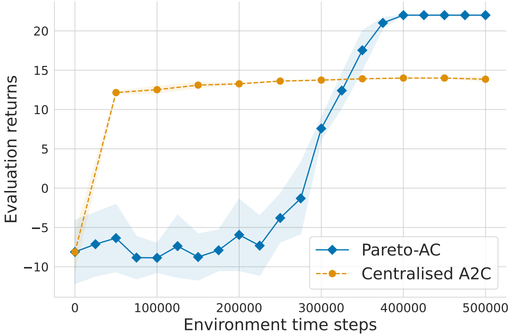


Figure: Learning curves in the Climbing game.

- A2C with centralized state-value critic converges to the Pareto-dominated equilibrium (B, B) with +7 (per agent).
- Pareto actor-critic converges to the Pareto-optimal equilibrium (A, A) with +11 (per agent).

Value Decomposition in Common-Reward Games

Centralized Value Functions in Value-Based MARL

We addressed MARL challenges in policy gradient algorithms by leveraging centralized critics. Can we also use centralized value functions in value-based MARL algorithms?

Centralized Value Functions in Value-Based MARL

We addressed MARL challenges in policy gradient algorithms by leveraging centralized critics. Can we also use centralized value functions in value-based MARL algorithms?

Problem

In value-based MARL algorithms, e.g. IDQN, agents learn value functions and derive their policy from them. However, learning and deriving a policy from centralized value functions would prevent decentralized execution.

Centralized Value Functions in Value-Based MARL

We addressed MARL challenges in policy gradient algorithms by leveraging centralized critics. Can we also use centralized value functions in value-based MARL algorithms?

Problem

In value-based MARL algorithms, e.g. IDQN, agents learn value functions and derive their policy from them. However, learning and deriving a policy from centralized value functions would prevent decentralized execution.

How can we overcome this limitation and leverage the benefits of centralized value functions in value-based MARL algorithms?

Value Decomposition

We will focus on value decomposition methods for common-reward games. These methods aim to decompose a centralized action-value function of all agents

$$Q(h^t, z^t, a^t; \theta) = \mathbb{E}\left[\sum_{\tau=t}^{\infty} \gamma^{\tau-t} r^\tau \mid h^t, z^t, a^t\right]$$

into individual utility functions of each agent: $Q(h_i, a_i; \theta_i)$ for $i \in I$

Value Decomposition

We will focus on value decomposition methods for common-reward games. These methods aim to decompose a centralized action-value function of all agents

$$Q(h^t, z^t, a^t; \theta) = \mathbb{E}\left[\sum_{\tau=t}^{\infty} \gamma^{\tau-t} r^\tau \mid h^t, z^t, a^t\right]$$

into individual utility functions of each agent: $Q(h_i, a_i; \theta_i)$ for $i \in I$

This decomposition has several benefits:

- Agents benefit from centralized information during training
- Simplify learning by decomposing the centralized value function
- Agents learn their individual utility functions to represent their contribution to the centralized value function, helping to address the **credit assignment problem**

Individual-Global-Max Property

How do we ensure that decentralized action selection with respect to the agents' individual utility functions leads to effective joint actions with respect to the decomposed centralized action-value function?

Individual-Global-Max Property

How do we ensure that decentralized action selection with respect to the agents' individual utility functions leads to effective joint actions with respect to the decomposed centralized action-value function?

Solution

Let $\hat{h}$ be a full history with joint-observation histories $h = \sigma(\hat{h})$, individual observation histories, $h_i = \sigma_i(\hat{h})$, and centralized information z. The individual-global-max (IGM) property is satisfied if and only if:

$$\forall a = (a_1, \ldots, a_n) \in A : a \in A^*(h, z; \theta) \iff \forall i \in I : a_i \in A_i^*(h_i; \theta_i)$$

with $A^*(h, z; \theta) = \arg\max_{a \in A} Q(h, z, a; \theta)$ and $A_i^*(h_i; \theta_i) = \arg\max_{a_i \in A_i} Q(h_i, a_i; \theta_i)$.

The Importance of the Individual-Global-Max Property

Upholding the IGM property has two important implications:

The Importance of the Individual-Global-Max Property

Upholding the IGM property has two important implications:

1. If all agents decentrally select actions that maximise their individual utility functions, the resulting joint action will maximise the centralized action-value function $\rightarrow$ effective decentralised execution

The Importance of the Individual-Global-Max Property

Upholding the IGM property has two important implications:

1. If all agents decentrally select actions that maximise their individual utility functions, the resulting joint action will maximise the centralized action-value function $\rightarrow$ effective decentralised execution
2. The greedy joint action with respect to the centralized action-value function can be efficiently obtained by selecting the greedy action for each agent with respect to their individual utility functions $\rightarrow$ efficient centralized training

The Importance of the Individual-Global-Max Property

Upholding the IGM property has two important implications:

1. If all agents decentrally select actions that maximise their individual utility functions, the resulting joint action will maximise the centralized action-value function $\rightarrow$ effective decentralised execution
2. The greedy joint action with respect to the centralized action-value function can be efficiently obtained by selecting the greedy action for each agent with respect to their individual utility functions $\rightarrow$ efficient centralized training

Note

It is not guaranteed that for a given environment, there exists a decomposition of the centralized action-value function that satisfies the IGM property.

Linear Value Decomposition

Value decomposition networks (VDN) uses a simple linear decomposition of the centralized action-value function:

$$Q(h^t, z^t, a^t; \theta) = \sum_{i \in I} Q(h_i^t, a_i^t; \theta_i)$$

Linear Value Decomposition

Value decomposition networks (VDN) uses a simple linear decomposition of the centralized action-value function:

$$Q(h^t, z^t, a^t; \theta) = \sum_{i \in I} Q(h_i^t, a_i^t; \theta_i)$$

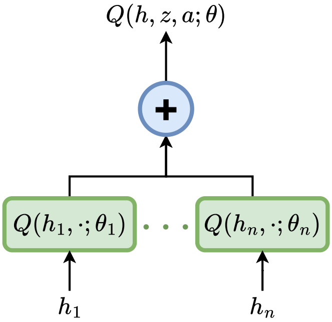
This decomposition satisfies the IGM property and we can jointly optimise the parameters of all networks by minimising the following loss on sampled batches of experiences $\mathcal{B}$:

$$\mathcal{L}(\theta) = \frac{1}{B} \sum_{(h^t, a^t, r^t, h^{t+1}) \in \mathcal{B}} \left(r^t + \gamma \sum_{i \in I} \max_{a_i \in A_i} Q(h_i^{t+1}, a_i; \bar{\theta}_i) - \sum_{i \in I} Q(h_i^t, a_i^t; \theta_i) \right)^2$$

with $\bar{\theta}_i$ denoting the parameters of agent i's target network.

Algorithm Value decomposition networks (VDN)

Initialize n utility networks with random parameters $\theta_1, \ldots, \theta_n$
Initialize n target networks with parameters $\bar{\theta}_1 = \theta_1, \ldots, \bar{\theta}_n = \theta_n$
Initialize a shared replay buffer D
for time step $t = 0, 1, 2, \ldots$ **do**
    Collect current observations $o_1^t, \ldots, o_n^t$
    for agent $i = 1, \ldots, n$ **do**
        With probability ϵ: choose random action a_i^t
        Otherwise: choose $a_i^t \in \arg\max_{a_i} Q(h_i^t, a_i; \theta_i)$
    Apply actions; collect shared reward r^t and next observations $o_1^{t+1}, \ldots, o_n^{t+1}$
    Store transition (h^t, a^t, r^t, h^{t+1}) in shared replay buffer D
    Sample mini-batch of B transitions (h^k, a^k, r^k, h^{k+1}) from D
    if s^{k+1} is terminal **then**
        Targets $y^k \leftarrow r^k$
    else
        Targets $y^k \leftarrow r^k + \gamma \sum_{i \in I} \max_{a'_i \in A_i} Q(h_i^{k+1}, a'_i; \bar{\theta}_i)$
    Loss $\mathcal{L}(\theta) \leftarrow \frac{1}{B} \sum_{k=1}^{B} \left(y^k - \sum_{i \in I} Q(h_i^k, a_i^k; \theta_i) \right)^2$
    Update parameters θ by minimizing the loss $\mathcal{L}(\theta)$
    In a set interval, update target network parameters $\bar{\theta}_i$ for each agent i



Monotonic Value Decomposition

A more general decomposition (that also ensures the IGM property) can be formulated by assuming that the centralized action-value function is a (strictly) monotonically increasing function with respect to any individual utility function:

$$\forall i \in I, \forall a \in A : \frac{\partial Q(h, z, a; \theta)}{\partial Q(h_i, a_i; \theta_i)} > 0$$

Monotonic Value Decomposition

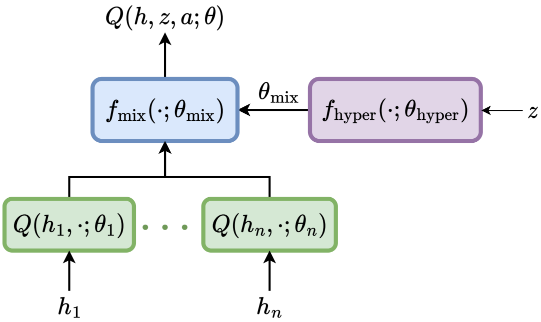
A more general decomposition (that also ensures the IGM property) can be formulated by assuming that the centralized action-value function is a (strictly) monotonically increasing function with respect to any individual utility function:

$$\forall i \in I, \forall a \in A : \frac{\partial Q(h, z, a; \theta)}{\partial Q(h_i, a_i; \theta_i)} > 0$$

The QMIX algorithm implements this assumption using a mixing function f_{mix} that aggregates individual utilities to approximate the centralized action-value function:

$$Q(h, z, a, \theta) = f_{\text{mix}}\left(Q(h_1, a_1; \theta_1), \ldots, Q(h_n, a_n; \theta_n); \theta_{\text{mix}}\right)$$

QMIX Architecture



The centralized action-value function is monotonic with respect to individual utilities if all weights of f_{mix} are positive $\rightarrow$ ensure positive weights by obtaining the parameters of the mixing function from a hypernetwork f_{hyper} conditioned on centralized information z

QMIX Optimisation

The parameters of all utility functions and the hypernetwork are jointly optimised by minimising the following loss on batches of experiences $\mathcal{B}$ sampled from a replay buffer:

$$\mathcal{L}(\theta) = \frac{1}{B} \sum_{(h^t, z^t, a^t, r^t, h^{t+1}, z^{t+1}) \in \mathcal{B}} \left(r^t + \gamma \max_{a \in A} Q(h^{t+1}, z^{t+1}, a; \bar{\theta}) - Q(h^t, z^t, a^t; \theta) \right)^2$$

with the following decomposed value estimates:

$$Q(h^t, z^t, a^t, \theta) = f_{\text{mix}} \left(Q(h_1^t, a_1^t; \theta_1), \ldots, Q(h_n^t, a_n^t; \theta_n); \theta_{\text{mix}} \right)$$

$$\max_{a \in A} Q(h^{t+1}, z^{t+1}, a; \bar{\theta}) = f_{\text{mix}} \left(\max_{a_1 \in A_1} Q(h_1^{t+1}, a_1; \bar{\theta}_1), \ldots, \max_{a_n \in A_n} Q(h_n^{t+1}, a_n; \bar{\theta}_n); \bar{\theta}_{\text{mix}} \right)$$

Value Decomposition in Matrix Games

To better understand how value decomposition works in practise, we will look at several exemplary tasks and the learned decompositions of both VDN and QMIX.

Value Decomposition in Matrix Games

To better understand how value decomposition works in practise, we will look at several exemplary tasks and the learned decompositions of both VDN and QMIX.

	A	B
A	1	5
B	5	9

Figure: Linear game

	A	B
A	0	0
B	0	10

Figure: Monotonic game

	A	B	C
A	11	-30	0
B	-30	7	0
C	0	6	5

Figure: Climbing game

Value Decomposition in Matrix Games

To better understand how value decomposition works in practise, we will look at several exemplary tasks and the learned decompositions of both VDN and QMIX.

	A	B
A	1	5
B	5	9

Figure: Linear game

	A	B
A	0	0
B	0	10

Figure: Monotonic game

	A	B	C
A	11	-30	0
B	-30	7	0
C	0	6	5

Figure: Climbing game

	Linear game	Monotonic game	Climbing game
Linearly decomposable	✓	✗	✗
Monotonically decomposable	✓	✓	✗

Value Decomposition in Linearly Decomposable Matrix Game

	A	B
A	1	5
B	5	9

(a) True rewards

	0.12	**4.12**
0.88	1.00	5.00
4.88	5.00	**9.00**

(b) VDN decomposition

	-0.21	**0.68**
0.19	1.00	5.00
0.96	5.00	**9.00**

(c) QMIX decomposition

Value Decomposition in Linearly Decomposable Matrix Game

	A	B
A	1	5
B	5	9

(a) True rewards

	0.12	**4.12**
0.88	1.00	5.00
4.88	5.00	**9.00**

(b) VDN decomposition

	-0.21	**0.68**
0.19	1.00	5.00
0.96	5.00	**9.00**

(c) QMIX decomposition

We can make several observations from the learned decompositions:

Value Decomposition in Linearly Decomposable Matrix Game

	A	B
A	1	5
B	5	9

(a) True rewards

	0.12	**4.12**
0.88	1.00	5.00
4.88	5.00	**9.00**

(b) VDN decomposition

	−0.21	**0.68**
0.19	1.00	5.00
0.96	5.00	**9.00**

(c) QMIX decomposition

We can make several observations from the learned decompositions:

- VDN and QMIX are able to learn the true centralized action-value function

Value Decomposition in Linearly Decomposable Matrix Game

	A	B
A	1	5
B	5	9

(a) True rewards

	0.12	**4.12**
0.88	1.00	5.00
4.88	5.00	**9.00**

(b) VDN decomposition

	-0.21	**0.68**
0.19	1.00	5.00
0.96	5.00	**9.00**

(c) QMIX decomposition

We can make several observations from the learned decompositions:

- VDN and QMIX are able to learn the true centralized action-value function
- The learned decompositions are not unique and can vary between different runs

Value Decomposition in Linearly Decomposable Matrix Game

	A	B
A	1	5
B	5	9

(a) True rewards

	0.12	**4.12**
0.88	1.00	5.00
4.88	5.00	**9.00**

(b) VDN decomposition

	−0.21	**0.68**
0.19	1.00	5.00
0.96	5.00	**9.00**

(c) QMIX decomposition

We can make several observations from the learned decompositions:

- VDN and QMIX are able to learn the true centralized action-value function
- The learned decompositions are not unique and can vary between different runs
- Individual utility values, in particular for QMIX, can be difficult to interpret (besides larger values indicating higher return estimates)

Value Decomposition in Monotonically Decomposable Matrix Game

	A	B
A	0	0
B	0	10

(a) True rewards

	−1.45	**3.45**
−0.94	−2.43	2.51
4.08	2.60	**7.53**

(b) VDN decomposition

	−4.91	**0.82**
−4.66	0.00	0.00
1.81	0.00	**10.00**

(c) QMIX decomposition

Value Decomposition in Monotonically Decomposable Matrix Game

	A	B
A	0	0
B	0	10

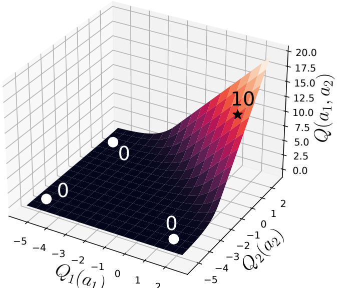
(a) True rewards

	-1.45	**3.45**
-0.94	-2.43	2.51
4.08	2.60	**7.53**

(b) VDN decomposition

	-4.91	**0.82**
-4.66	0.00	0.00
1.81	0.00	**10.00**

(c) QMIX decomposition



Only QMIX is able to represent the non-linear but monotonic relationship between the individual utility functions and the centralized action-value function.

Value Decomposition in Climbing Game

	A	B	C
A	11	-30	0
B	-30	7	0
C	0	6	5

Figure: True rewards

In the Climbing game, neither VDN nor QMIX are able to learn the true centralized action-value function and converge to sub-optimal policies.

	-4.56	-4.15	$\mathbf{3.28}$
-4.28	-8.84	-8.43	-1.00
-6.10	-10.66	-10.25	-2.82
$\mathbf{5.31}$	0.75	1.16	$\mathbf{8.59}$

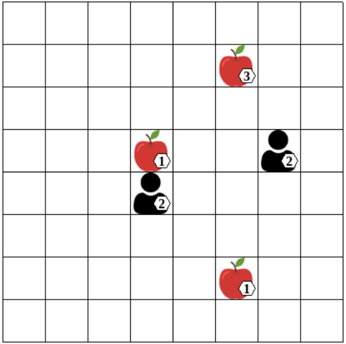
(a) VDN decomposition

	-16.60	$-\mathbf{0.24}$	-4.68
-7.44	-11.16	-11.16	-11.16
7.65	-11.15	2.34	-1.37
$\mathbf{11.27}$	-4.95	$\mathbf{8.72}$	5.01

(b) QMIX decomposition

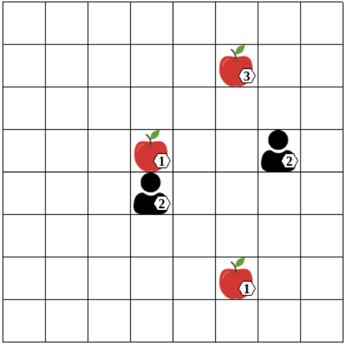
Value Decomposition in LBF

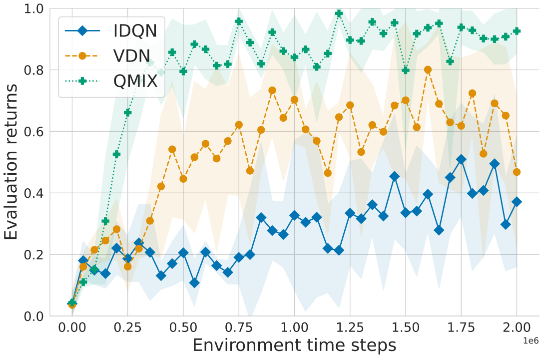
So far, we looked at simple single-step matrix games. We will now compare IDQN, VDN and QMIX in a common-reward level-based foraging task where two agents need to collect three items in a 8×8 grid world:



Value Decomposition in LBF

So far, we looked at simple single-step matrix games. We will now compare IDQN, VDN and QMIX in a common-reward level-based foraging task where two agents need to collect three items in a 8×8 grid world:





Summary

We covered:

- Training and execution modes
- Independent learning with deep reinforcement learning
- Multi-agent policy gradient algorithms
- Value decomposition in common-reward games

Next we'll cover:

- Agent modeling with deep learning
- Parameter and experience sharing
- Policy self-play in zero-sum games
- Population-based training