MARL, External Memory

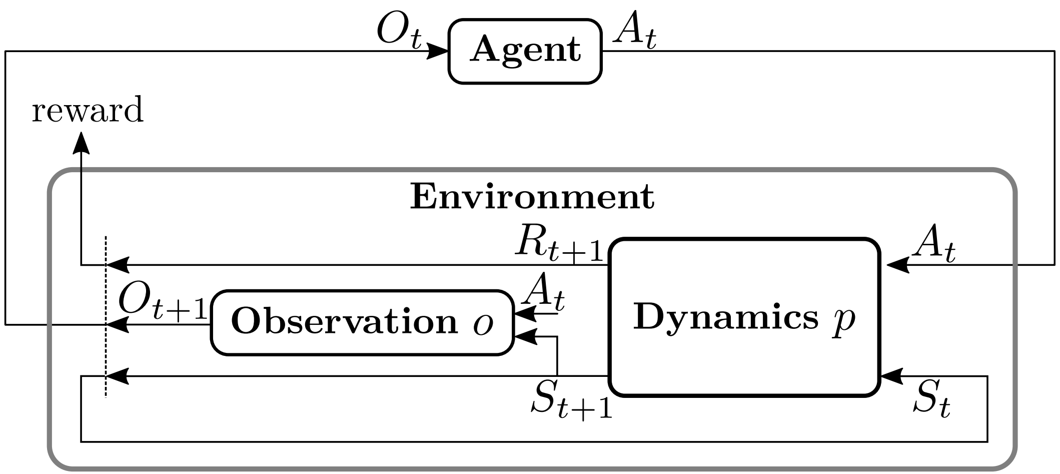
Milan Straka

May 20, 2024

EUROPEAN UNION
European Structural and Investment Fund
Operational Programme Research,
Development and Education

Charles University in Prague
Faculty of Mathematics and Physics
Institute of Formal and Applied Linguistics

Partially Observable MDPs



Recall that a **partially observable Markov decision process** extends the Markov decision process to a sextuple $(\mathcal{S}, \mathcal{A}, p, \gamma, \mathcal{O}, o)$, where the MDP components

- $\mathcal{S}$ is a set of states,
- $\mathcal{A}$ is a set of actions,
- $p(S_{t+1} = s', R_{t+1} = r | S_t = s, A_t = a)$ is a probability that action $a \in \mathcal{A}$ will lead from state $s \in \mathcal{S}$ to $s' \in \mathcal{S}$, producing a **reward** $r \in \mathbb{R}$,
- $\gamma \in [0, 1]$ is a **discount factor**,

are extended by:

- $\mathcal{O}$ is a set of observations,
- $o(O_{t+1} | S_{t+1}, A_t)$ is an observation model, where observation O_t is used as agent input instead of the state S_t.

Partially Observable Stochastic Game

A **partially observable stochastic game (POSG)** is a 9-tuple $(\mathcal{S}, N, \{\mathcal{A}^{i\in[N]}\}, \{\Omega^{i\in[N]}\}, \{R^{i\in[N]}\}, P, \{O^{i\in[N]}\}, \rho_0, \gamma)$, where

- $\mathcal{S}$ is the set of all possible *states*,
- N is the *number of agents*,
- $\mathcal{A}^i$ is the set of all possible *actions* for agent $\boldsymbol{i}$, with $\mathcal{A}^\Pi \stackrel{\text{def}}{=} \prod_i \mathcal{A}^i$,
- Ω^i is the set of all possible *observations* for agent $\boldsymbol{i}$,
- $R^i(r^i_{t+1} \in \mathbb{R} | \boldsymbol{s}_t \in \mathcal{S}, \boldsymbol{a}_t \in \mathcal{A}^\Pi, \boldsymbol{s}_{t+1} \in \mathcal{S})$ is the *reward function* for agent $\boldsymbol{i}$,
- $O^i(\omega^i_{t+1} \in \Omega^i | \boldsymbol{s}_{t+1} \in \mathcal{S}, a^i_t \in \mathcal{A})$ is the *observation model* for agent $\boldsymbol{i}$, a distribution of observing w^i_{t+1} after performing action a^i_t leading to state $\boldsymbol{s}_{t+1}$,
- $P(\boldsymbol{s}_{t+1} \in \mathcal{S} | \boldsymbol{s}_t \in \mathcal{S}, \boldsymbol{a}_t \in \mathcal{A}^\Pi)$ is the *transition model*,
- ρ_0 is the *initial state distribution*,
- $\gamma \in [0, 1]$ is a *discount factor*.

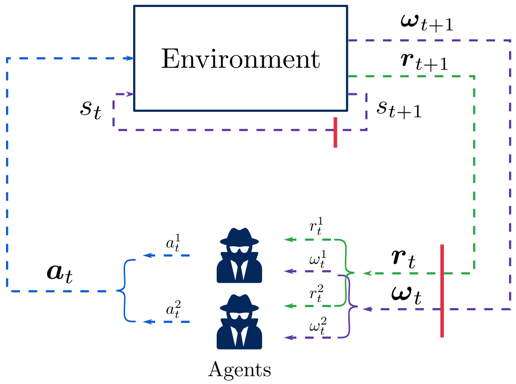


Figure 1.3: Partially Observable Stochastic Game (POSG)

Figure 1.3 of "Cooperative Multi-Agent Reinforcement Learning", https://dspace.cuni.cz/handle/20.500.11956/127431

We denote

- joint actions/policy/observation across all agents as vectors

$$\boldsymbol{a} \stackrel{\text{def}}{=} (a^1, \ldots, a^N) \in \mathcal{A}^\Pi,$$

- joint actions/policy/observation for all agents but agent i as

$$\boldsymbol{a}^{-i} \stackrel{\text{def}}{=} (a^1, \ldots, a^{i-1}, a^{i+1}, \ldots, a^N),$$

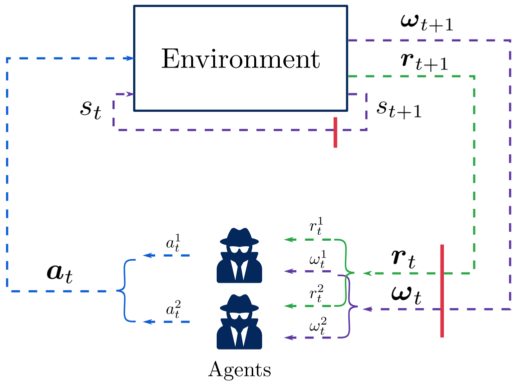


Figure 1.3: Partially Observable Stochastic Game (POSG)

Figure 1.3 of "Cooperative Multi-Agent Reinforcement Learning", https://dspace.cuni.cz/handle/20.500.11956/127431

Agent-Environment Cycle Game

However, when actually implementing POSG, various ambiguities exist in the order of execution. Therefore, **agent-environment cycle game (AECG)** has been proposed, a 12-tuple $(\mathcal{S}, N, \{\mathcal{A}^{i\in[N]}\}, \{\Omega^{i\in[N]}\}, \{R^{i\in[N]}\}, \{T^{i\in[N]}\}, P, \{O^{i\in[N]}\}, V, s_0, i_0, \gamma)$ where

- $\mathcal{S}$ is the set of all possible *states*,
- N is the *number of agents*, including 0 for "environment" agent; $[N^\cup] \stackrel{\text{def}}{=} [N] \cup \{0\}$,
- $\mathcal{A}^i$ is the set of all possible *actions* for agent i, with $\mathcal{A}^0 \stackrel{\text{def}}{=} \{\emptyset\}$, $\mathcal{A}^\cup \stackrel{\text{def}}{=} \bigcup_{i\in[N^\cup]} \mathcal{A}^i$,
- Ω^i is the set of all possible *observations* for agent i,
- $R^i(r_{t+1}^i \in \mathbb{R} | s_t \in \mathcal{S}, j \in [N^\cup], a_t^j \in \mathcal{A}^j, s_{t+1} \in \mathcal{S})$ is the *reward distribution* for agent i,
- $T^i : \mathcal{S} \times \mathcal{A}^i \to \mathcal{S}$ is the deterministic *transition function* for agent i,
- $P(s_{t+1} \in \mathcal{S} | s_t \in \mathcal{S})$ is the transition function for the environment,
- $O^i(\omega_{t+1}^i \in \Omega^i | s_{t+1} \in \mathcal{S})$ is the *observation model* for agent i,
- $V(j \in [N^\cup] | s_t \in \mathcal{S}, i \in [N^\cup], a_t^i \in \mathcal{A}^i)$ is the *next agent function*,
- $s_0 \in \mathcal{S}$ is the *initial state*,
- $i_0 \in [N^U]$ is the *initial agent*, • $\gamma \in [0, 1]$ is a *discount factor*.

Agent-Environment Cycle Game

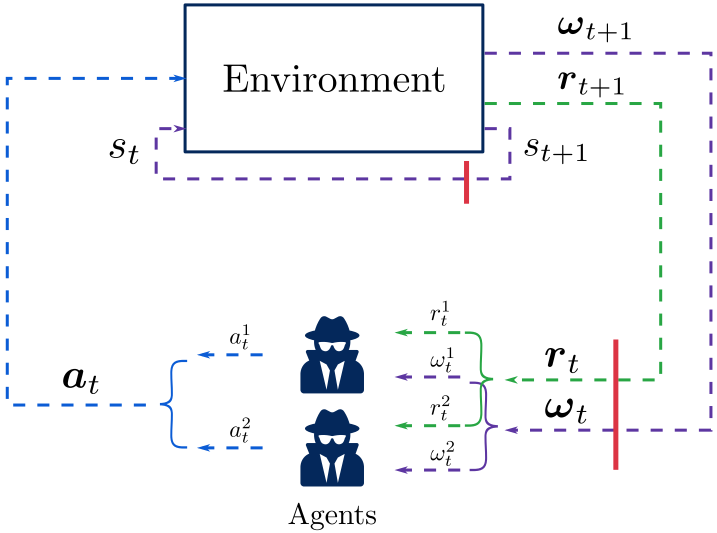


Figure 1.3: Partially Observable Stochastic Game (POSG)

Figure 1.3 of "Cooperative Multi-Agent Reinforcement Learning", https://dspace.cuni.cz/handle/20.500.11956/127431

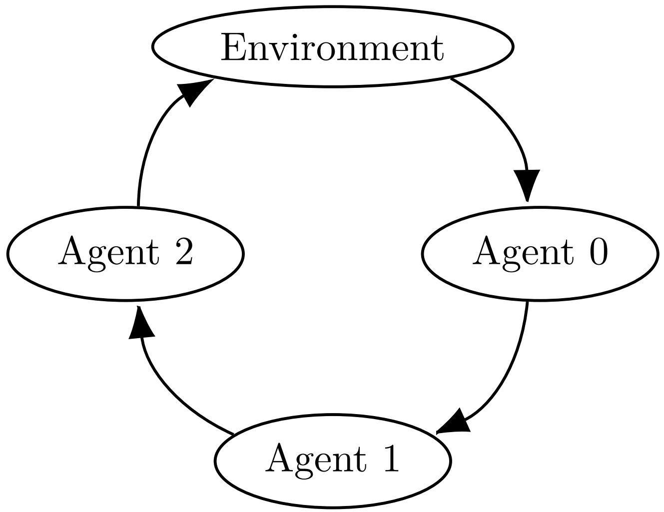


Figure 1.4: AEC diagram for MPE Simple Spread environment

Figure 1.4 of "Cooperative Multi-Agent Reinforcement Learning", https://dspace.cuni.cz/handle/20.500.11956/127431

It holds that for every POSG, there is an equivalent AECG, and vice versa.

Game Settings

Depending on the reward function, there are several game settings:

- **fully cooperative**, when $\forall i, \forall j : R^i(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1}) = R^j(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1})$,
- **cooperative**, when $\forall i, \forall j, \exists k > 0 : R^i(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1}) \geq kR^j(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1})$,
- **competitive**, when $\exists i, \exists j, \exists k < 0 : R^i(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1}) \geq kR^j(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1})$,
- **zero-sum**, when $\sum_{i \in [N]} R^i(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1}) = 0$,

The MARL Problem

We define a trajectory $\boldsymbol\tau$ as a sequence of states and actions

$$\boldsymbol\tau \stackrel{\textrm{def}}{=} (\boldsymbol s_0, \boldsymbol a_0, \boldsymbol s_1, \boldsymbol a_1, \boldsymbol s_2, \ldots),$$

where:

- $\boldsymbol s_0 \sim \rho_0$,
- $\boldsymbol a_t \sim \boldsymbol\pi(\cdot | \boldsymbol s_t)$,
- $\boldsymbol s_{t+1} \rightarrow P(\cdot | \boldsymbol s_t, \boldsymbol a_t)$.

A return for an agent i and trajectory $\boldsymbol\tau$ is

$$R^i(\boldsymbol\tau) \stackrel{\textrm{def}}{=} \sum_{t=0}^{|\boldsymbol\tau|} \gamma^t r_{t+1}^i.$$

The MARL Problem

For a given policy $\boldsymbol\pi$, the expected return for agent i is

$$J^i(\boldsymbol\pi) \stackrel{\text{def}}{=} \mathbb{E}_{\boldsymbol\tau\sim\boldsymbol\pi}\big[R^i(\boldsymbol\tau)\big],$$

where a probability of a trajetory $\boldsymbol\tau$ is

$$P(\boldsymbol\tau|\boldsymbol\pi) \stackrel{\text{def}}{=} \rho_0(s_0)\prod_{t=0}^{|\boldsymbol\tau|} P(s_{t+1}|s_t, \boldsymbol a_t)\boldsymbol\pi(\boldsymbol a_t|s_t).$$

For a given joing policy $\boldsymbol\pi^{-i}$, **best response** is

$$\hat\pi^i(\boldsymbol\pi^{-i}) \stackrel{\text{def}}{=} \arg\max_{\pi_i} J^i(\pi^i, \boldsymbol\pi^{-i})$$

The MARL Goal

It is unfortunately not clear what the goal of MARL should be, given that it is a multi-criterion optimization problem.

One possibility is to seek for **Nash equilibrium**, which is a joint policy $\boldsymbol{\pi}_*$ fulfilling

$$\forall i \in [N], \forall \pi^i : J^i(\boldsymbol{\pi}_*) \geq J^i(\pi^i, \boldsymbol{\pi}_*^{-i}).$$

In other words, π_*^i is a best response to $\boldsymbol{\pi}_*^{-i}$ for all agents i.

A Nash equilibrium exists for any finite game (finite number of players, each with a finite number of strategies). Unfortunately, there can be multiple Nash equilibria with different payoffs (Nash equilibrium is just a “local” optimum).

- Stag hunt

A\B	Stag	Rabbit
Stag	2\2	0\1
Rabbit	1\0	1\1

- Prisoner's dilemma

A\B	Stay silent	Testify
Stay silent	1\1	3\0
Testify	0\3	2\2

Centralized Scheme

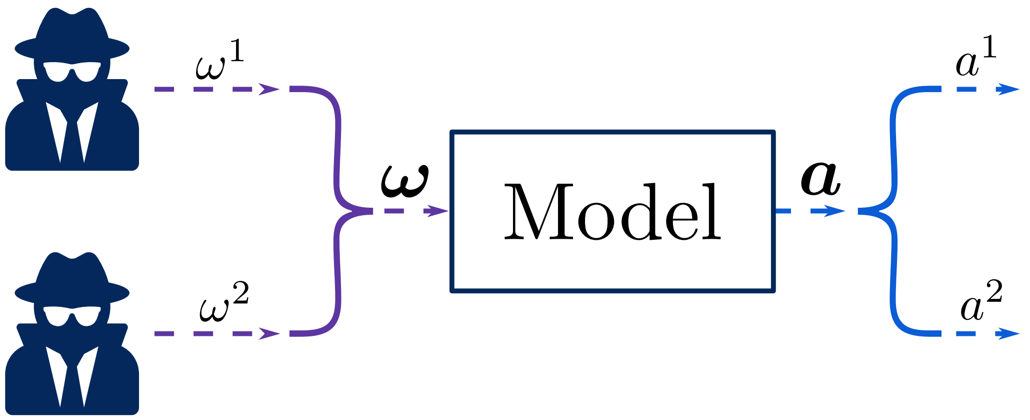


Figure 3.1: Centralized scheme

Figure 3.1 of "Cooperative Multi-Agent Reinforcement Learning", https://dspace.cuni.cz/handle/20.500.11956/127431

A joint model for all agents, a single critic.

Concurrent/Parameter-Sharing Scheme

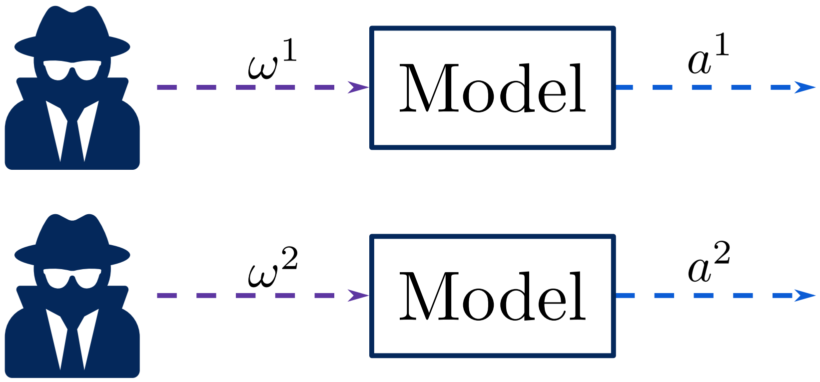


Figure 3.2: Concurrent scheme

Figure 3.2 of "Cooperative Multi-Agent Reinforcement Learning", https://dspace.cuni.cz/handle/20.500.11956/127431

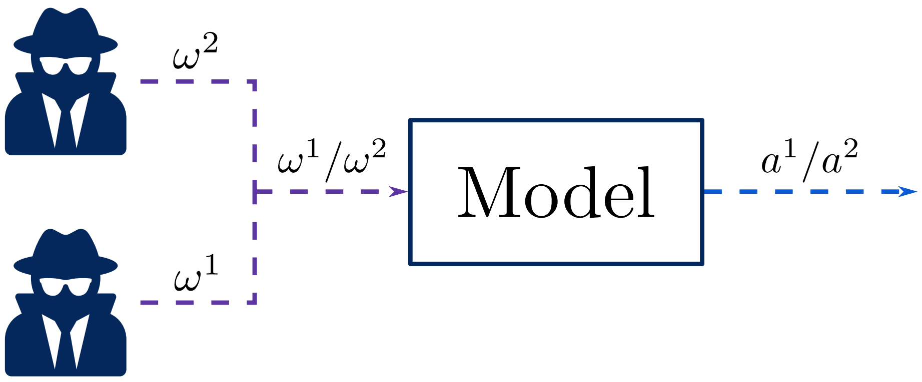


Figure 3.3: Parameter Sharing Scheme

Figure 3.3 of "Cooperative Multi-Agent Reinforcement Learning", https://dspace.cuni.cz/handle/20.500.11956/127431

Each agent is trained independently. When the agents are homogenous, their models can be optionally shared (the *parameter-sharing scheme*).

However, the environment is then non-stationary, and using a replay buffer is problematic because of changing policies of other agents.

Centralized Training with Decentralized Execution

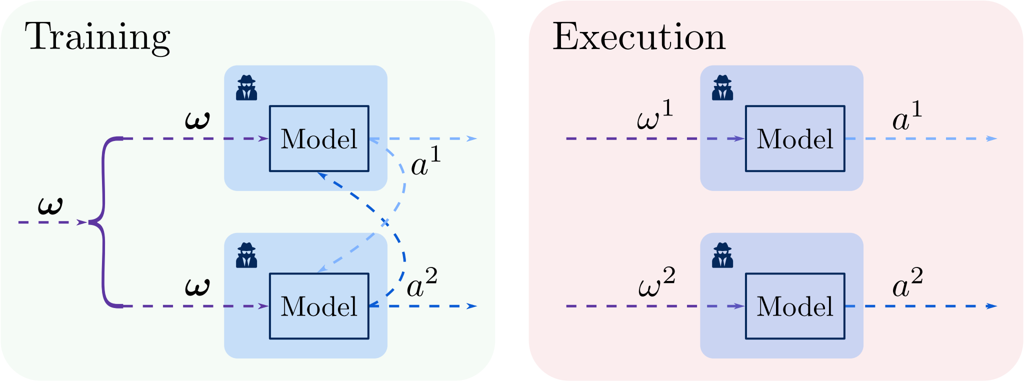


Figure 3.4: Centralized Training with Decentralized Execution (CT-DE)

Figure 3.4 of "Cooperative Multi-Agent Reinforcement Learning", https://dspace.cuni.cz/handle/20.500.11956/127431

Quite a common model, where the agents are independent, but the critics get the observations and actions of all agents.

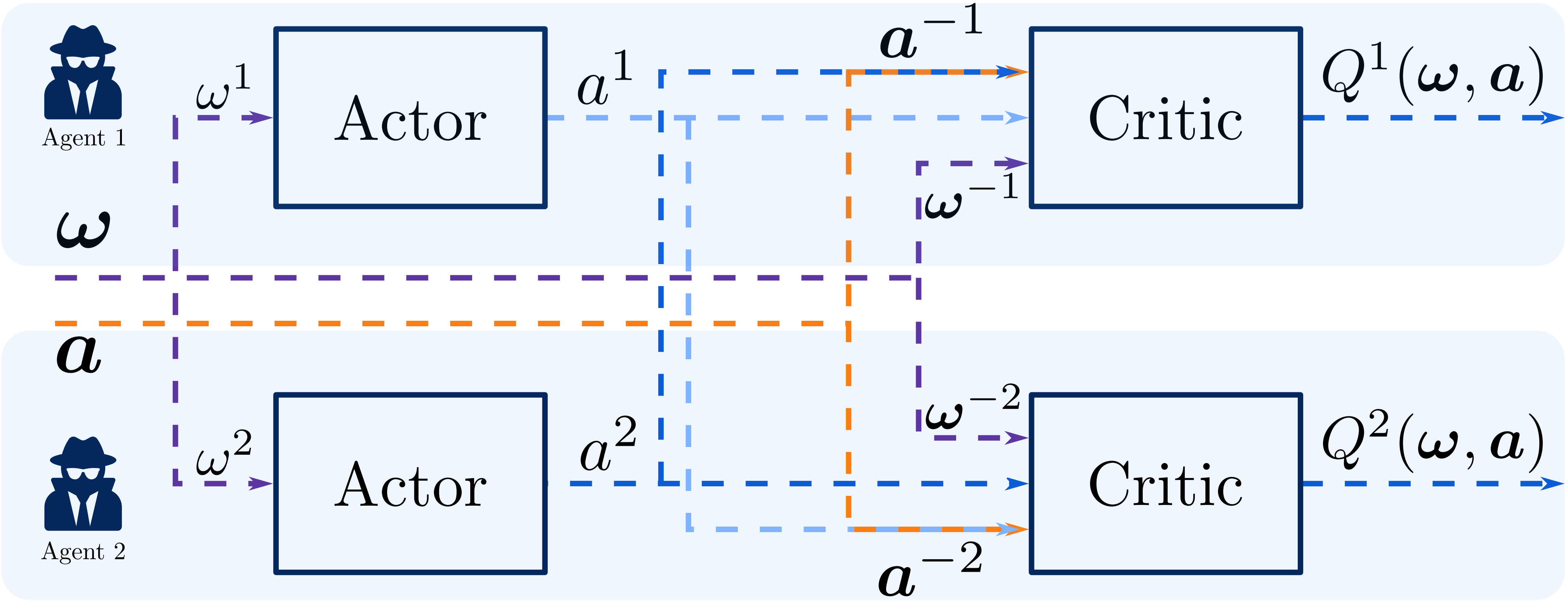


Figure 3.5: Multi-Agent Deep Deterministic Policy Gradient (MADDPG)

Figure 3.5 of "Cooperative Multi-Agent Reinforcement Learning", https://dspace.cuni.cz/handle/20.500.11956/127431

Multi-Agent Deep Deterministic Policy Gradient

Algorithm 3.1: Multi-Agent Deep Deterministic Policy Gradient

Input: initial policy parameters $\boldsymbol{\theta}$, Q-function parameters $\boldsymbol{\phi}$, empty replay buffer $\mathcal{D}$.

1 Set target parameters equal to main parameters $\boldsymbol{\theta}_{\text{targ}} \leftarrow \boldsymbol{\theta}$, $\boldsymbol{\phi}_{\text{targ}} \leftarrow \boldsymbol{\phi}$.
2 **repeat**
3 Observe joint-observation $\boldsymbol{\omega}$ and select joint-action $\boldsymbol{a} = \text{clip}(\boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{\omega}) + \boldsymbol{\epsilon}, \boldsymbol{a}_{\text{low}}, \boldsymbol{a}_{\text{high}})$, where $\boldsymbol{\epsilon} \sim \mathcal{N}$.
4 Execute $\boldsymbol{a}$ in the environment.
5 Observe next observations $\boldsymbol{\omega}'$, rewards $\boldsymbol{r}$ and done signal for each agent $\boldsymbol{d}$.
6 Store $(\boldsymbol{\omega}, \boldsymbol{a}, \boldsymbol{r}, \boldsymbol{\omega}', \boldsymbol{d})$ in replay buffer $\mathcal{D}$.
7 **if** all($\boldsymbol{d}$) is true **then**
8 Reset environment state.
9 **end if**
10 Randomly sample a batch of transitions, $B = \{(\boldsymbol{\omega}, \boldsymbol{a}, \boldsymbol{r}, \boldsymbol{\omega}', \boldsymbol{d})\}$ from $\mathcal{D}$.
11 **for** agent i in $[N]$ **do**
12 Compute targets

$$y(r^i, \boldsymbol{\omega}', d^i) = r^i + \gamma(1 - d^i) Q^i_{\phi_{\text{targ}}}(\boldsymbol{\omega}', \boldsymbol{\mu}_{\boldsymbol{\theta}_{\text{targ}}}(\boldsymbol{\omega}'))).$$

13 Update Q-function by one step of gradient descent using

$$\nabla^i_\phi \frac{1}{|B|} \sum_{(\boldsymbol{\omega}, \boldsymbol{a}, \boldsymbol{r}, \boldsymbol{\omega}', \boldsymbol{d}) \in B} \left(Q^i_\phi(\boldsymbol{\omega}, \boldsymbol{a}) - y(r^i, \boldsymbol{\omega}', d^i)\right)^2.$$

14 Update policy by one step of gradient ascent using

$$\nabla^i_\theta \frac{1}{|B|} \sum_{(\boldsymbol{\omega}, \boldsymbol{a}) \in B} Q^i_\phi(\boldsymbol{\omega}, \boldsymbol{a}^{-i}, \mu^i_\theta(\omega^i)).$$

15 Update target networks with

$$\phi^i_{\text{targ}} \leftarrow \alpha \phi^i_{\text{targ}} + (1 - \alpha) \phi^i$$
$$\theta^i_{\text{targ}} \leftarrow \alpha \theta^i_{\text{targ}} + (1 - \alpha) \theta^i.$$

16 **end for**
17 **until** convergence

Algorithm 3.1 of "Cooperative Multi-Agent Reinforcement Learning", https://dspace.cuni.cz/handle/20.500.11956/127431

Alternatively, in multi-agent settings, in some experiments it was beneficial to estimate the gradient for the policy update using the current policy instead of the action from the replay buffer; if the line 14 is changed to

$$\nabla^i_{\boldsymbol{\theta}} \frac{1}{|B|} \sum_{\boldsymbol{\omega}} Q^i_\varphi(\boldsymbol{\omega}, \boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{\omega})),$$

we talk about *Soft MADDPG*.

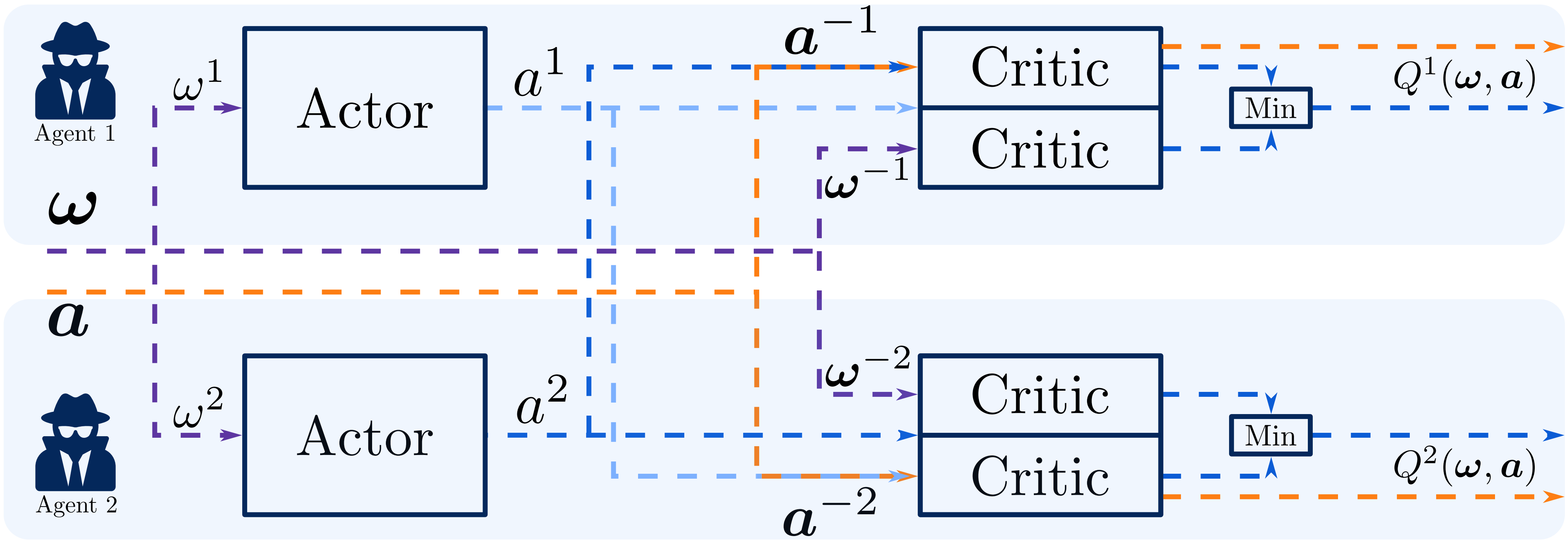


Figure 3.6: Multi-Agent Twin Delayed DDPG (MATD3)

Figure 3.6 of "Cooperative Multi-Agent Reinforcement Learning", https://dspace.cuni.cz/handle/20.500.11956/127431

Multi-Agent Twin Delayed DDPG

Algorithm 3.2: Multi-Agent Twin Delayed DDPG

Input: initial policy parameters $\boldsymbol{\theta}$, Q-function parameters $\boldsymbol{\phi}_1, \boldsymbol{\phi}_2$, empty replay buffer $\mathcal{D}$.

```
Set target parameters equal to main parameters θ_targ ← θ, φ_targ,1 ← φ_1, φ_targ,2 ← φ_2.
repeat
    Observe joint-observation ω and select joint-action
        a = clip(μ_θ(ω) + ε, a_low, a_high), where ε ~ N.
    Execute a in the environment.
    Observe next observations ω', rewards r and done signal for each agent d.
    Store (ω, a, r, ω', d) in replay buffer D.
    if all(d) is true then
        Reset environment state.
    end if
    Randomly sample a batch of transitions, B = {(ω, a, r, ω', d)} from D.
    for agent i in [N] do
        Compute target actions
```

$$\boldsymbol{a}' = \operatorname{clip}(\boldsymbol{\mu}_{\boldsymbol{\theta}_{\text{targ}}}(\boldsymbol{\omega}') + \operatorname{clip}(\boldsymbol{\epsilon}, -c, c), \boldsymbol{a}_{\text{low}}, \boldsymbol{a}_{\text{high}}), \qquad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \sigma \boldsymbol{I}).$$

```
        Compute targets
```

$$y(r^i, \boldsymbol{\omega}', d^i) = r^i + \gamma(1 - d^i) \min_{j \in \{1,2\}} Q^i_{\phi_{\text{targ},j}}(\boldsymbol{\omega}', \boldsymbol{a}').$$

```
        Update Q-function by one step of gradient descent using
```

$$\forall j \in \{1,2\}: \qquad \nabla^i_{\phi_j} \frac{1}{|B|} \sum_{(\boldsymbol{\omega},\boldsymbol{a},\boldsymbol{r},\boldsymbol{\omega}',\boldsymbol{d}) \in B} \Big(Q^i_{\phi_j}(\boldsymbol{\omega}, \boldsymbol{a}) - y(r^i, \boldsymbol{\omega}', d^i)\Big)^2.$$

```
        if time to update policy function then
            Update policy by one step of gradient ascent using
```

$$\nabla^i_\theta \frac{1}{|B|} \sum_{(\boldsymbol{\omega},\boldsymbol{a}) \in B} Q^i_{\phi_1}(\boldsymbol{\omega}, \boldsymbol{a}^{-i}, \mu^i_\theta(\omega^i)).$$

```
            Update target networks with
```

$$\begin{aligned} \phi^i_{\text{targ}} &\leftarrow \alpha \phi^i_{\text{targ}} + (1-\alpha)\phi^i \\ \theta^i_{\text{targ}} &\leftarrow \alpha \theta^i_{\text{targ}} + (1-\alpha)\theta^i. \end{aligned}$$

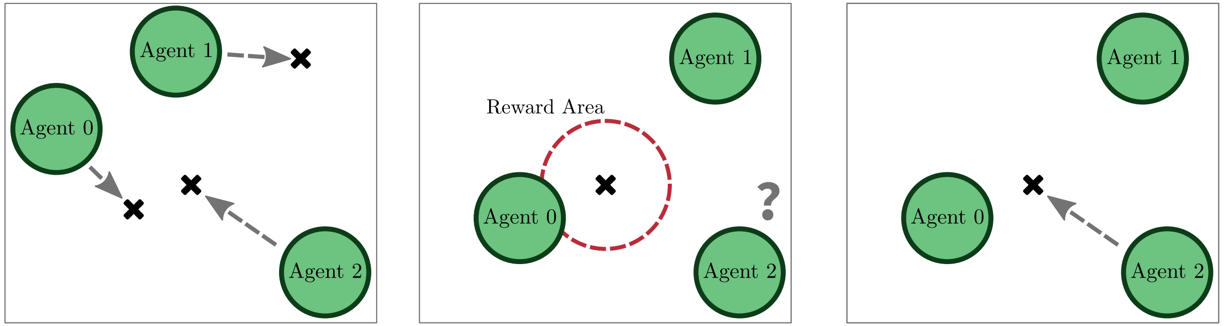
```
        end if
    end for
until convergence
```

We can again consider a *Soft MATD3* variant.

Furthermore, we can also use the minimum of both critics during policy update (shown to be beneficial by DDPG++ and SAC). The resulting algorithm is called *(Soft) MATD4*.

Algorithm 3.2 of "Cooperative Multi-Agent Reinforcement Learning", https://dspace.cuni.cz/handle/20.500.11956/127431

MARL Evaluation, Simple Target



(a) Cooperative navigation (b) MPE Simple Spread (c) MPE Simple Target

Figure 6.1: Cooperative navigation

Figure 6.1 of "Cooperative Multi-Agent Reinforcement Learning", https://dspace.cuni.cz/handle/20.500.11956/127431

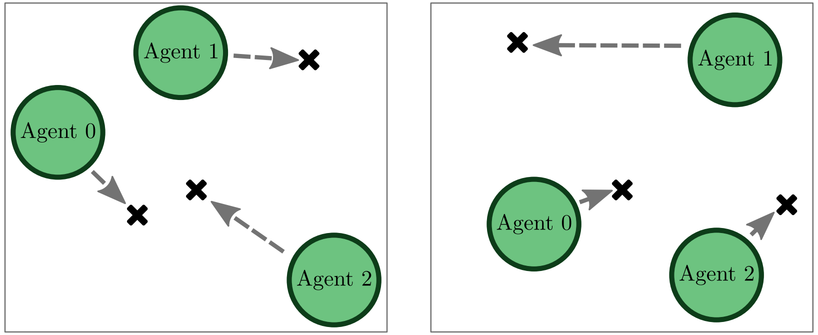


Figure 6.2: MPE Simple Collect

Figure 6.2 of "Cooperative Multi-Agent Reinforcement Learning", https://dspace.cuni.cz/handle/20.500.11956/127431

Reward is given for touching a landmark, and for unoccupied landmarks also for distance of the nearest agent (orignally any agent, but easier variant is an agent not occupying a landmark).

The agents have non-negligible size and get negative reward for colliding.

Actions can be discrete ($\varnothing$, $\leftarrow$, $\rightarrow$, $\uparrow$, $\downarrow$; ST Gumbel-softmax is used) or continuous.

In the *Simple Collect* variant, the targets disappear after being occupied for some time, and a new one appears on a random location.

MARL Evaluation, Simple Target, Continuous Actions

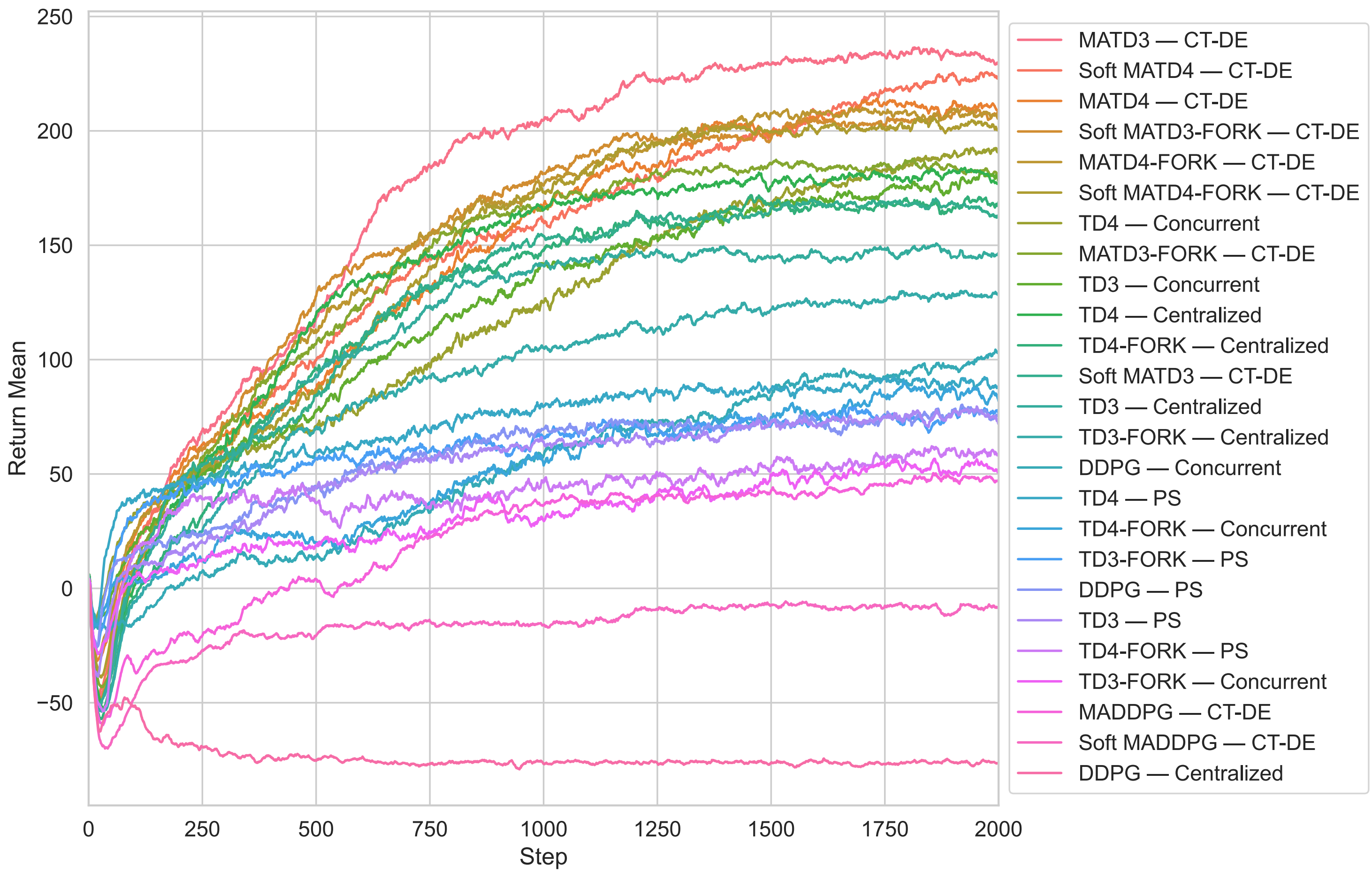


Figure 6.4: Training of MPE Simple Target Continuous

Figure 6.4 of "Cooperative Multi-Agent Reinforcement Learning", https://dspace.cuni.cz/handle/20.500.11956/127431

MARL Evaluation, Simple Target, Continuous Actions

Algorithm — Scheme	Step 1000	Step 1500	Step 2000
MATD3 — CT-DE	**204.55 ± 75.20**	**229.22 ± 61.24**	**229.13 ± 60.49**
Soft MATD4 — CT-DE	160.05 ± 91.58	198.95 ± 60.09	222.70 ± 63.45
MATD4 — CT-DE	167.36 ± 93.70	197.96 ± 83.78	209.31 ± 73.73
Soft MATD3-FORK — CT-DE	181.43 ± 120.70	196.62 ± 115.77	207.12 ± 120.15
MATD4-FORK — CT-DE	176.10 ± 103.08	207.62 ± 63.07	205.87 ± 54.62
Soft MATD4-FORK — CT-DE	173.04 ± 121.90	199.92 ± 115.03	200.56 ± 112.56
TD4 — Concurrent	124.86 ± 56.81	168.19 ± 57.67	191.26 ± 66.95
MATD3-FORK — CT-DE	165.90 ± 109.04	185.39 ± 121.30	181.14 ± 117.56
TD3 — Concurrent	141.68 ± 95.89	166.04 ± 94.70	178.08 ± 70.62
TD4 — Centralized	165.42 ± 99.01	175.77 ± 91.21	177.23 ± 94.04
TD4-FORK — Centralized	147.53 ± 158.98	168.67 ± 162.29	169.09 ± 161.63
Soft MATD3 — CT-DE	153.94 ± 90.68	166.21 ± 118.72	162.48 ± 111.84
TD3 — Centralized	140.86 ± 172.41	144.93 ± 173.38	146.84 ± 173.46
TD3-FORK — Centralized	105.11 ± 155.83	121.86 ± 128.41	127.91 ± 128.45
DDPG — Concurrent	57.89 ± 158.75	84.37 ± 156.04	102.91 ± 163.00
TD4 — PS	79.51 ± 31.69	86.90 ± 24.38	88.14 ± 25.81
TD4-FORK — Concurrent	53.72 ± 52.19	74.92 ± 71.39	81.69 ± 76.66
TD3-FORK — PS	66.37 ± 36.13	71.32 ± 41.40	76.44 ± 31.63
DDPG — PS	69.52 ± 38.57	73.21 ± 29.49	75.89 ± 32.83
TD3 — PS	64.14 ± 99.31	72.93 ± 110.74	71.72 ± 107.94
TD4-FORK — PS	46.26 ± 44.39	52.97 ± 61.03	57.93 ± 46.61
TD3-FORK — Concurrent	29.50 ± 58.41	49.89 ± 69.70	51.17 ± 79.36
MADDPG — CT-DE	36.53 ± 121.71	42.91 ± 125.80	46.68 ± 130.79
Soft MADDPG — CT-DE	-16.09 ± 68.93	-7.41 ± 69.93	-8.69 ± 71.27
DDPG — Centralized	-76.77 ± 43.85	-76.46 ± 43.56	-76.39 ± 40.30

Table 6.3: Training of MPE Simple Target Continuous

Table 6.3 of "Cooperative Multi-Agent Reinforcement Learning", https://dspace.cuni.cz/handle/20.500.11956/127431

MARL Evaluation, Simple Target, Discrete Actions

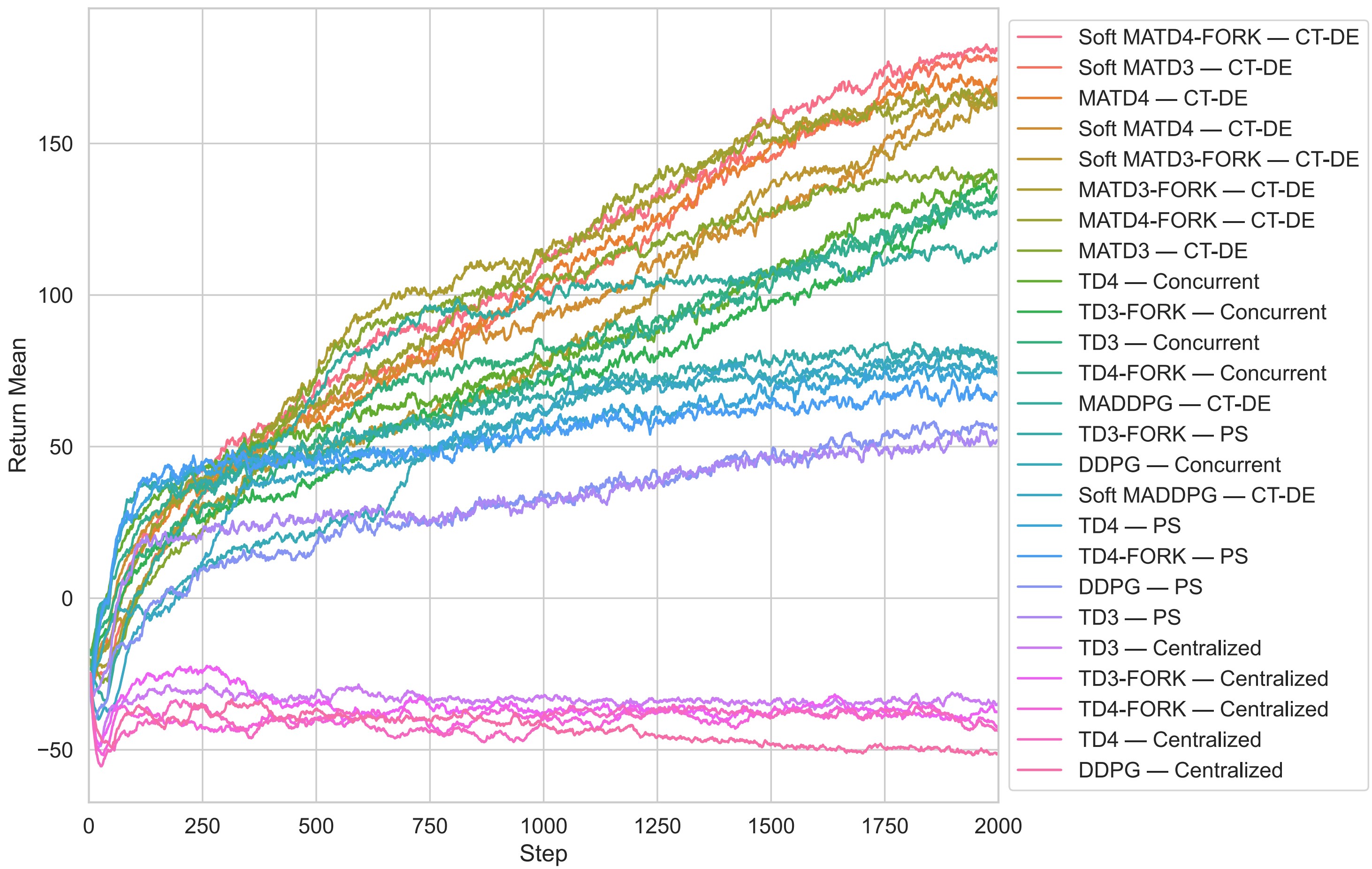


Figure 6.5: Training of MPE Simple Target Discrete

Figure 6.5 of "Cooperative Multi-Agent Reinforcement Learning", https://dspace.cuni.cz/handle/20.500.11956/127431

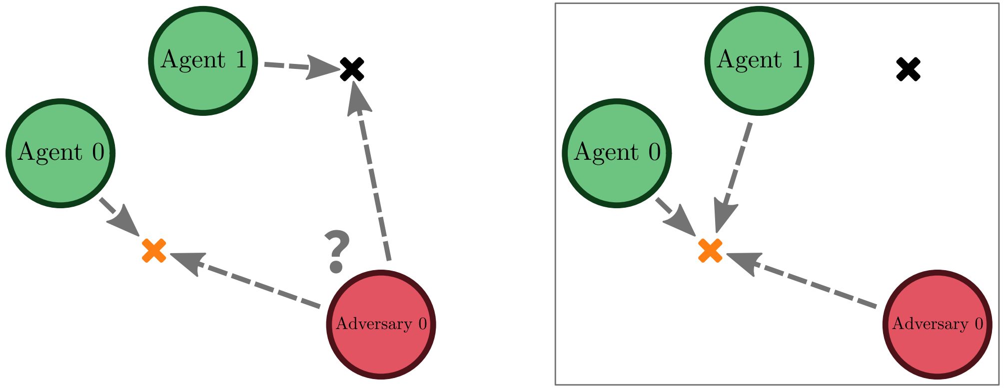
MARL Evaluation, Simple Target, Discrete Actions

Algorithm — Scheme	Step 1000	Step 1500	Step 2000
Soft MATD4-FORK — CT-DE	111.75 ± 77.96	**157.47 ± 75.75**	**181.75 ± 53.02**
Soft MATD3 — CT-DE	101.32 ± 83.43	145.85 ± 88.63	177.61 ± 89.56
MATD4 — CT-DE	104.05 ± 66.35	150.05 ± 87.94	172.31 ± 46.00
Soft MATD4 — CT-DE	92.24 ± 75.62	126.30 ± 84.29	166.52 ± 98.42
Soft MATD3-FORK — CT-DE	75.79 ± 70.77	137.09 ± 104.59	165.54 ± 84.61
MATD3-FORK — CT-DE	**112.65 ± 65.49**	156.95 ± 78.50	163.67 ± 53.54
MATD4-FORK — CT-DE	111.99 ± 64.69	150.89 ± 85.10	163.32 ± 65.31
MATD3 — CT-DE	105.28 ± 69.32	127.97 ± 67.64	138.51 ± 72.62
TD4 — Concurrent	76.77 ± 42.61	106.56 ± 50.66	137.84 ± 47.28
TD3-FORK — Concurrent	70.46 ± 61.78	98.35 ± 98.07	135.50 ± 84.69
TD3 — Concurrent	82.29 ± 59.64	109.08 ± 57.14	131.59 ± 47.93
TD4-FORK — Concurrent	74.34 ± 50.74	102.12 ± 70.92	128.22 ± 73.07
MADDPG — CT-DE	98.26 ± 92.25	107.20 ± 112.55	118.11 ± 95.47
TD3-FORK — PS	66.57 ± 36.22	77.96 ± 34.42	79.53 ± 29.01
DDPG — Concurrent	61.63 ± 67.56	71.28 ± 70.27	77.90 ± 67.64
Soft MADDPG — CT-DE	60.59 ± 100.51	71.92 ± 106.74	75.72 ± 108.28
TD4 — PS	54.58 ± 41.65	65.69 ± 36.96	74.00 ± 40.12
TD4-FORK — PS	56.13 ± 37.18	63.85 ± 44.07	66.86 ± 34.37
DDPG — PS	33.35 ± 69.91	46.41 ± 78.70	55.19 ± 82.42
TD3 — PS	32.00 ± 73.67	46.71 ± 85.10	52.29 ± 87.22
TD3 — Centralized	-32.49 ± 32.23	-34.73 ± 28.95	-34.45 ± 32.21
TD3-FORK — Centralized	-38.20 ± 34.78	-38.26 ± 31.54	-37.41 ± 32.86
TD4-FORK — Centralized	-39.79 ± 27.42	-37.66 ± 28.74	-42.60 ± 23.18
TD4 — Centralized	-38.51 ± 30.20	-38.95 ± 30.45	-42.84 ± 36.64
DDPG — Centralized	-41.75 ± 29.24	-48.40 ± 31.10	-50.95 ± 31.73

Table 6.4: Training of MPE Simple Target Discrete

Table 6.4 of "Cooperative Multi-Agent Reinforcement Learning", https://dspace.cuni.cz/handle/20.500.11956/127431

MARL Evaluation, Simple Confuse



(a) Optimal strategy (b) Suboptimal strategy

Figure 6.3: Physical Deception

Figure 6.3 of "Cooperative Multi-Agent Reinforcement Learning", https://dspace.cuni.cz/handle/20.500.11956/127431

Some number of cooperaing agents gets rewarded based on the minimum distance of any agent to the target landmark; but are penalized based on the distance of a single adversary to the target landmark.

The adversary gets rewarded based on its distance to the target landmark; however, it does not know which landmark is the target one.

Actions can be again either discrete or continuous.

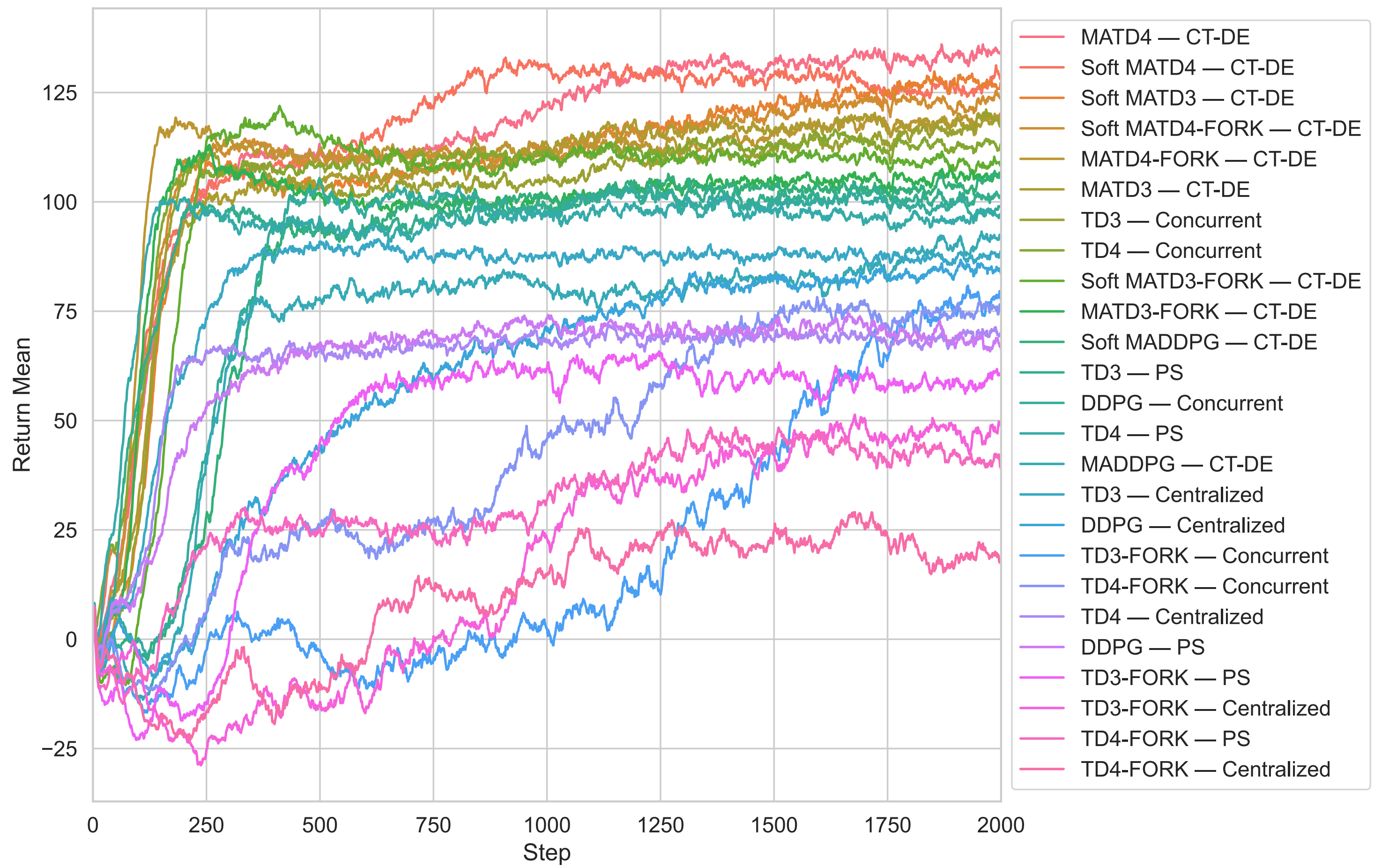


Figure 6.6: Training of MPE Simple Confuse Continuous

Figure 6.6 of "Cooperative Multi-Agent Reinforcement Learning", https://dspace.cuni.cz/handle/20.500.11956/127431

Algorithm — Scheme	**Step 1000**	**Step 1500**	**Step 2000**
MATD4 — CT-DE	122.18 ± 33.17	**131.30 ± 30.50**	**133.66 ± 27.11**
Soft MATD4 — CT-DE	**131.06 ± 37.29**	129.13 ± 31.19	127.84 ± 28.16
Soft MATD3 — CT-DE	112.32 ± 29.91	121.77 ± 28.70	126.46 ± 35.92
Soft MATD4-FORK — CT-DE	111.54 ± 21.17	120.62 ± 33.04	123.30 ± 33.84
MATD4-FORK — CT-DE	112.46 ± 31.82	115.03 ± 32.07	119.04 ± 36.35
MATD3 — CT-DE	113.12 ± 30.99	116.09 ± 30.69	118.65 ± 29.63
TD3 — Concurrent	105.01 ± 22.77	111.73 ± 31.89	116.86 ± 33.29
TD4 — Concurrent	111.86 ± 21.07	112.31 ± 21.51	112.78 ± 24.54
Soft MATD3-FORK — CT-DE	111.02 ± 27.48	110.57 ± 32.56	107.96 ± 25.47
MATD3-FORK — CT-DE	101.94 ± 19.59	105.07 ± 22.73	106.14 ± 31.57
Soft MADDPG — CT-DE	100.63 ± 20.79	103.23 ± 24.06	105.31 ± 21.57
TD3 — PS	98.97 ± 128.10	100.34 ± 123.74	100.67 ± 127.73
DDPG — Concurrent	97.02 ± 32.39	100.35 ± 26.67	98.67 ± 24.01
TD4 — PS	96.92 ± 123.85	98.33 ± 120.15	96.89 ± 124.90
MADDPG — CT-DE	80.88 ± 111.29	82.77 ± 105.02	91.45 ± 119.99
TD3 — Centralized	87.51 ± 109.69	87.82 ± 108.53	88.24 ± 102.05
DDPG — Centralized	70.40 ± 109.99	82.32 ± 109.73	83.46 ± 115.35
TD3-FORK — Concurrent	2.43 ± 85.80	42.99 ± 69.73	77.41 ± 37.24
TD4-FORK — Concurrent	45.92 ± 84.69	72.06 ± 90.34	76.21 ± 74.87
TD4 — Centralized	68.44 ± 118.04	68.00 ± 120.47	70.01 ± 116.58
DDPG — PS	73.56 ± 118.83	72.27 ± 124.91	66.27 ± 125.84
TD3-FORK — PS	62.00 ± 114.66	61.09 ± 113.96	60.19 ± 119.57
TD3-FORK — Centralized	22.97 ± 98.41	43.69 ± 93.67	49.01 ± 102.98
TD4-FORK — PS	31.90 ± 118.33	44.34 ± 122.45	38.96 ± 115.48
TD4-FORK — Centralized	14.61 ± 101.07	24.73 ± 104.00	18.10 ± 105.13

Table 6.5: Training of MPE Simple Confuse Continuous

Table 6.5 of "Cooperative Multi-Agent Reinforcement Learning", https://dspace.cuni.cz/handle/20.500.11956/127431

MARL Evaluation, Simple Confuse, Discrete Actions

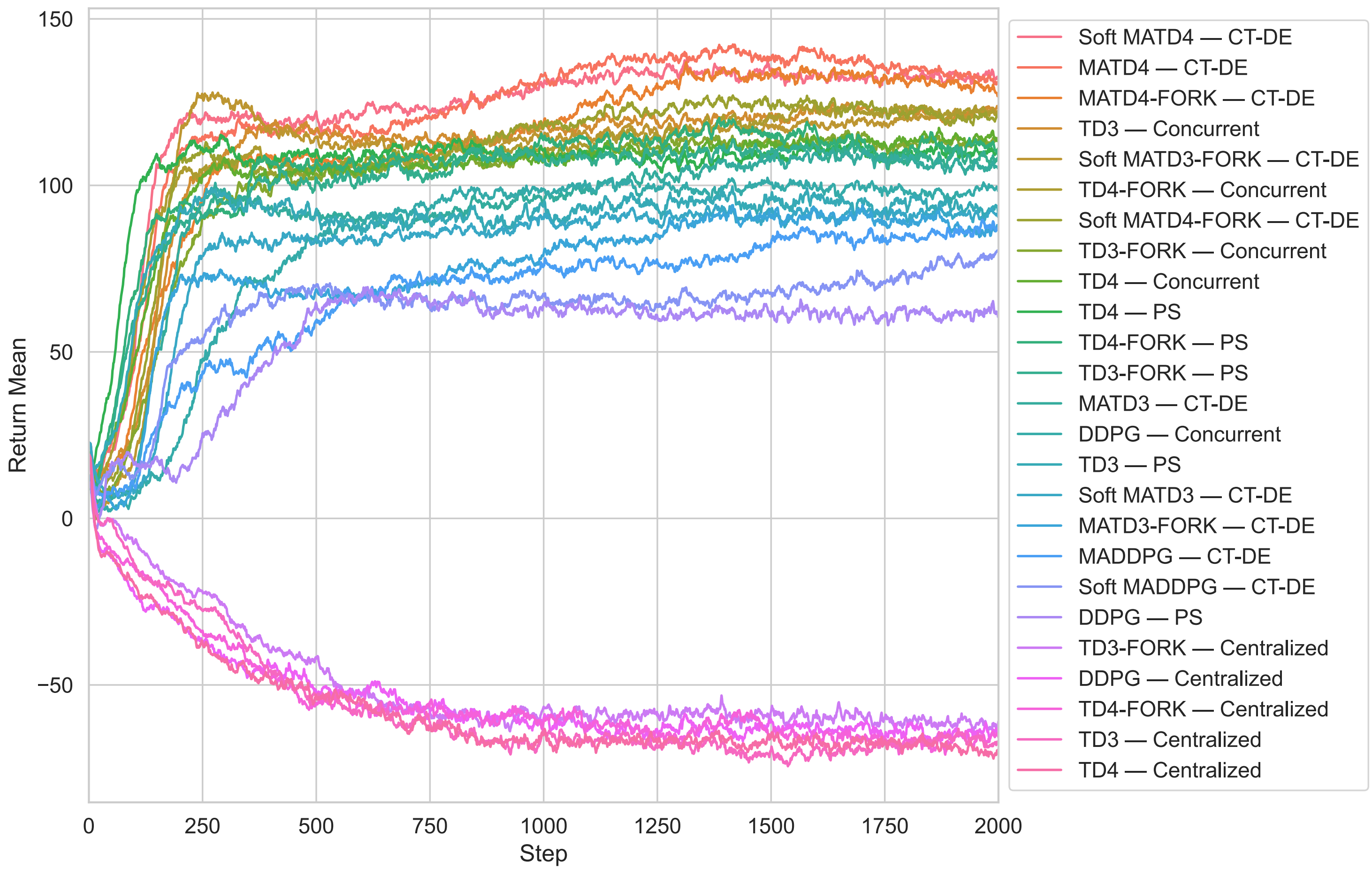


Figure 6.7: Training of MPE Simple Confuse Discrete

Figure 6.7 of "Cooperative Multi-Agent Reinforcement Learning", https://dspace.cuni.cz/handle/20.500.11956/127431

MARL Evaluation, Simple Confuse, Discrete Actions

Algorithm — Scheme	Step 1000	Step 1500	Step 2000
Soft MATD4 — CT-DE	130.73 ± 64.02	134.33 ± 67.35	**131.67 ± 60.95**
MATD4 — CT-DE	**133.16 ± 62.79**	**137.66 ± 60.18**	130.77 ± 62.34
MATD4-FORK — CT-DE	118.88 ± 30.88	133.05 ± 35.41	126.85 ± 31.52
TD3 — Concurrent	117.66 ± 70.83	121.21 ± 68.82	122.93 ± 73.97
Soft MATD3-FORK — CT-DE	114.68 ± 76.54	117.71 ± 75.47	121.95 ± 73.92
TD4-FORK — Concurrent	110.27 ± 25.70	120.18 ± 32.67	121.65 ± 33.89
Soft MATD4-FORK — CT-DE	117.52 ± 36.84	122.71 ± 40.63	120.39 ± 33.28
TD3-FORK — Concurrent	109.80 ± 28.23	112.24 ± 31.18	113.66 ± 31.04
TD4 — Concurrent	107.70 ± 25.13	112.05 ± 30.07	113.13 ± 32.46
TD4 — PS	110.53 ± 93.17	108.77 ± 95.13	111.26 ± 90.47
TD4-FORK — PS	113.83 ± 136.90	114.20 ± 129.01	108.52 ± 115.81
TD3-FORK — PS	107.67 ± 93.65	109.83 ± 93.05	107.64 ± 96.69
MATD3 — CT-DE	95.42 ± 91.09	107.58 ± 103.40	105.22 ± 104.55
DDPG — Concurrent	98.94 ± 27.85	100.38 ± 26.36	98.39 ± 25.54
TD3 — PS	94.36 ± 148.68	95.72 ± 151.30	92.01 ± 153.11
Soft MATD3 — CT-DE	89.92 ± 105.15	91.97 ± 109.25	91.08 ± 108.21
MATD3-FORK — CT-DE	79.58 ± 108.00	90.22 ± 110.49	87.85 ± 109.62
MADDPG — CT-DE	76.21 ± 79.41	82.52 ± 86.87	87.00 ± 92.56
Soft MADDPG — CT-DE	67.05 ± 78.25	67.61 ± 75.00	80.64 ± 65.77
DDPG — PS	64.00 ± 155.87	60.61 ± 157.34	60.46 ± 159.04
TD3-FORK — Centralized	-56.55 ± 34.65	-58.02 ± 33.39	-60.90 ± 40.31
DDPG — Centralized	-59.94 ± 37.19	-60.30 ± 36.79	-64.07 ± 38.17
TD4-FORK — Centralized	-60.94 ± 33.82	-65.66 ± 32.61	-65.76 ± 32.89
TD3 — Centralized	-63.76 ± 30.93	-69.51 ± 28.80	-67.00 ± 32.73
TD4 — Centralized	-65.01 ± 34.23	-65.31 ± 34.02	-70.42 ± 35.32

Table 6.6: Training of MPE Simple Confuse Discrete

Table 6.6 of "Cooperative Multi-Agent Reinforcement Learning", https://dspace.cuni.cz/handle/20.500.11956/127431

Multi-Agent Hide-and-Seek

As another example, consider https://openai.com/blog/emergent-tool-use/.

MERLIN

In a partially-observable environment, keeping all information in the RNN state is substantially limiting. Therefore, *memory-augmented* networks can be used to store suitable information in external memory (in the lines of NTM, DNC, or MANN models).

We now describe an approach used by Merlin architecture (*Unsupervised Predictive Memory in a Goal-Directed Agent* DeepMind Mar 2018 paper).

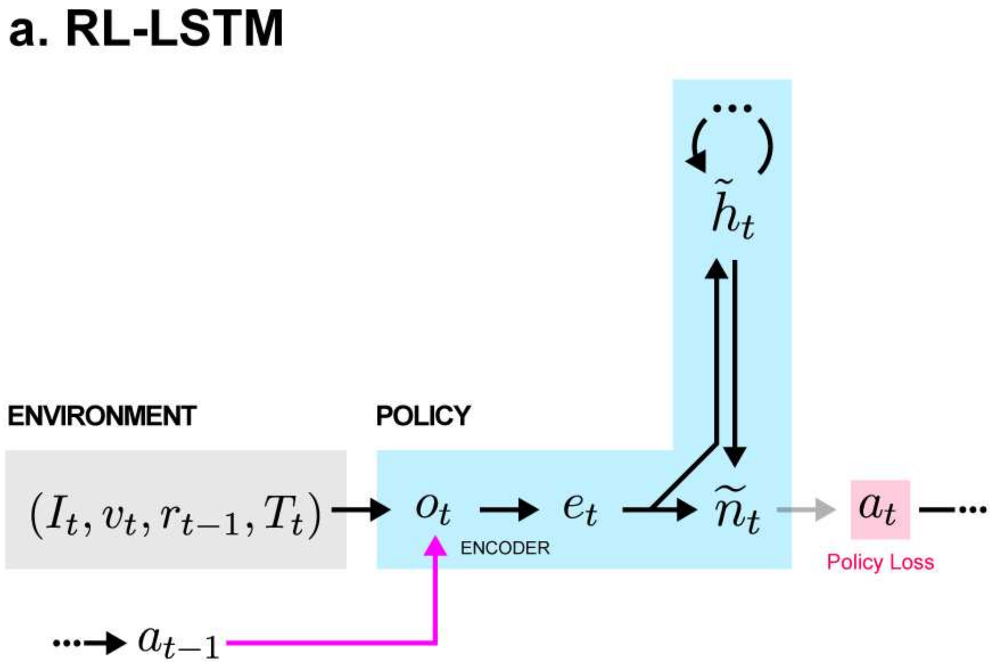


Figure 1a of "Unsupervised Predictive Memory in a Goal-Directed Agent", https://arxiv.org/abs/1803.10760

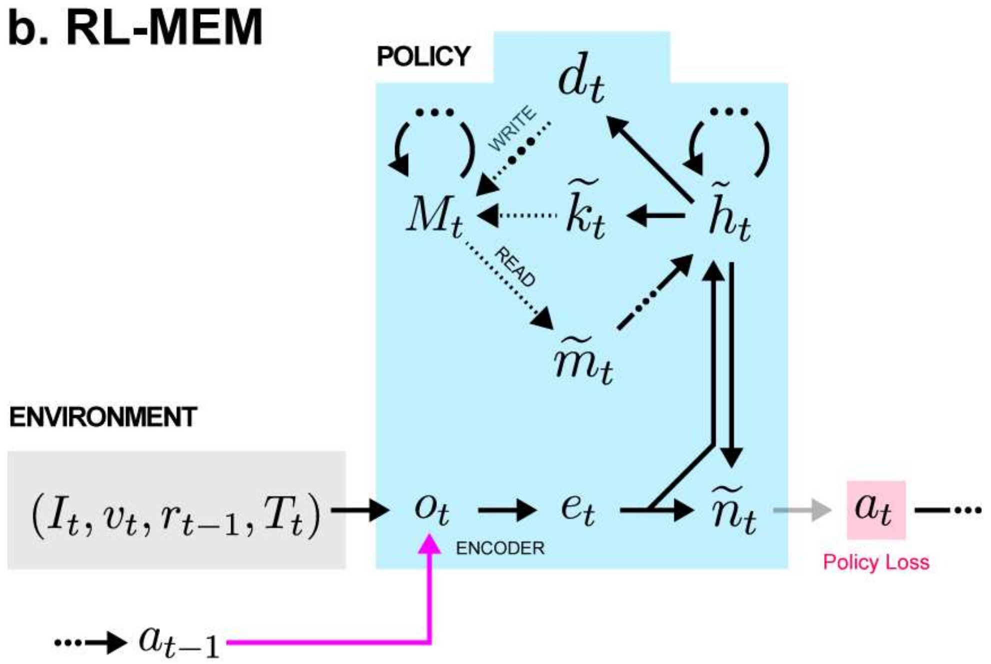


Figure 1b of "Unsupervised Predictive Memory in a Goal-Directed Agent", https://arxiv.org/abs/1803.10760

MERLIN – Memory Module

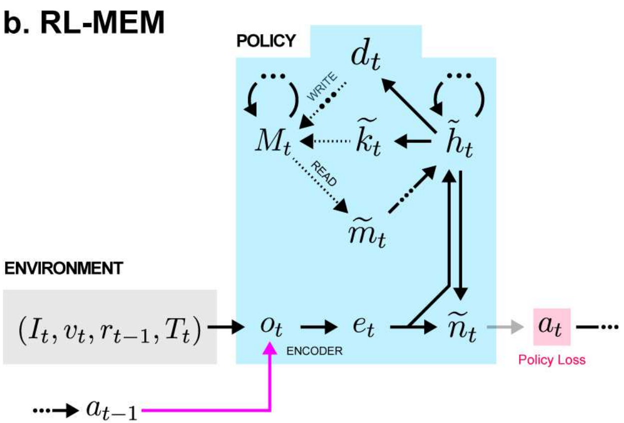


Figure 1b of "Unsupervised Predictive Memory in a Goal-Directed Agent", https://arxiv.org/abs/1803.10760

Let $\boldsymbol{M}$ be a memory matrix of size $N_{mem} \times 2|\boldsymbol{e}|$.

Assume we have already encoded observations as $\boldsymbol{e}_t$ and previous action a_{t-1}. We concatenate them with K previously read vectors and process them by a deep LSTM (two layers are used in the paper) to compute $\boldsymbol{h}_t$.

Then, we apply a linear layer to $\boldsymbol{h}_t$, computing K key vectors $\boldsymbol{k}_1, \ldots, \boldsymbol{k}_K$ of length $2|\boldsymbol{e}|$ and K positive scalars $\beta_1, \ldots, \beta_K$.

Reading: For each i, we compute cosine similarity of $\boldsymbol{k}_i$ and all memory rows $\boldsymbol{M}_j$, multiply the similarities by β_i and pass them through a $\operatorname{softmax}$ to obtain weights $\boldsymbol{\omega}_i$. The read vector is then computed as $\boldsymbol{M}\boldsymbol{\omega}_i$.

Writing: We find one-hot write index $\boldsymbol{v}_{wr}$ to be the least used memory row (we keep usage indicators and add read weights to them). We then compute $\boldsymbol{v}_{ret} \leftarrow \gamma \boldsymbol{v}_{ret} + (1-\gamma)\boldsymbol{v}_{wr}$, and retroactively update the memory matrix using $\boldsymbol{M} \leftarrow \boldsymbol{M} + \boldsymbol{v}_{wr}[\boldsymbol{e}_t, 0] + \boldsymbol{v}_{ret}[0, \boldsymbol{e}_t]$.

However, updating the encoder and memory content purely using RL is inefficient. Therefore, MERLIN includes a *memory-based predictor (MBP)* in addition to policy. The goal of MBP is to compress observations into low-dimensional state representations $\boldsymbol{z}$ and storing them in memory.

We want the state variables not only to faithfully represent the data, but also emphasise rewarding elements of the environment above irrelevant ones. To accomplish this, the authors follow the hippocampal representation theory of Gluck and Myers, who proposed that hippocampal representations pass through a compressive bottleneck and then reconstruct input stimuli together with task reward.

In MERLIN, a (Gaussian diagonal) *prior* distribution over $\boldsymbol{z}_t$ predicts next state variable conditioned on history of state variables and actions $p(\boldsymbol{z}_t^{\text{prior}}|\boldsymbol{z}_{t-1}, a_{t-1}, \ldots, \boldsymbol{z}_1, a_1)$, and *posterior* corrects the prior using the new observation $\boldsymbol{o}_t$, forming a better estimate $q(\boldsymbol{z}_t|\boldsymbol{o}_t, \boldsymbol{z}_t^{\text{prior}}, \boldsymbol{z}_{t-1}, a_{t-1}, \ldots, \boldsymbol{z}_1, a_1) + \boldsymbol{z}_t^{\text{prior}}$.

MERLIN — Prior and Posterior

To achieve the mentioned goals, we add two terms to the loss.

- We try reconstructing input stimuli, action, reward and return using a sample from the state variable posterior, and add the difference of the reconstruction and ground truth to the loss.
- We also add KL divergence of the prior and the posterior to the loss, to ensure consistency between the prior and the posterior.

c. MERLIN

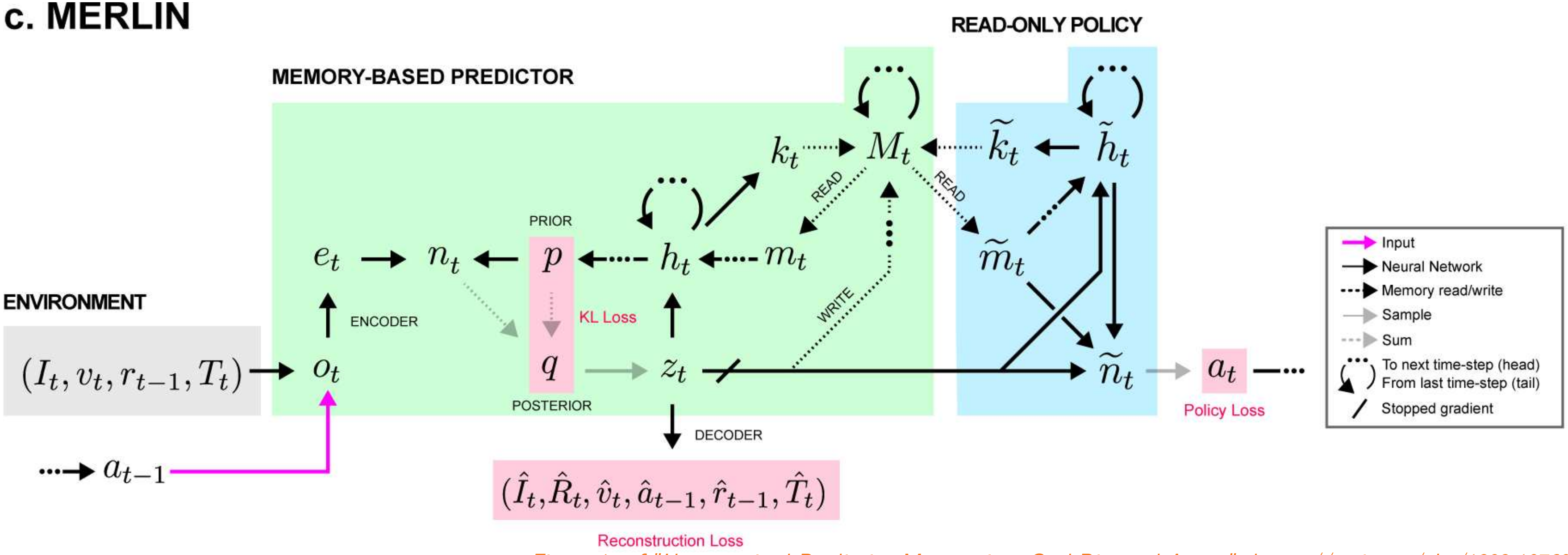


Figure 1c of "Unsupervised Predictive Memory in a Goal-Directed Agent", https://arxiv.org/abs/1803.10760

Algorithm 1 MERLIN Worker Pseudocode

// Assume global shared parameter vectors θ for the policy network and χ for the memory-based predictor; global shared counter $T := 0$
// Assume thread-specific parameter vectors θ', χ'
// Assume discount factor $\gamma \in (0, 1]$ and bootstrapping parameter $\lambda \in [0, 1]$
Initialize thread step counter $t := 1$
repeat
  Synchronize thread-specific parameters $\theta' := \theta; \chi' := \chi$
  Zero model's memory & recurrent state if new episode begins
  $t_\text{start} := t$
  repeat
    Prior $\mathcal{N}(\mu_t^\text{p}, \log \Sigma_t^\text{p}) = p(h_{t-1}, m_{t-1})$
    $e_t = \text{enc}(o_t)$
    Posterior $\mathcal{N}(\mu_t^\text{q}, \log \Sigma_t^\text{q}) = q(e_t, h_{t-1}, m_{t-1}, \mu_t^\text{p}, \log \Sigma_t^\text{p})$
    Sample $z_t \sim \mathcal{N}(\mu_t^\text{q}, \log \Sigma_t^\text{q})$
    Policy network update $\tilde{h}_t = \text{rec}(\tilde{h}_{t-1}, \tilde{m}_t, \text{StopGradient}(z_t))$
    Policy distribution $\pi_t = \pi(\tilde{h}_t, \text{StopGradient}(z_t))$
    Sample $a_t \sim \pi_t$
    $h_t = \text{rec}(h_{t-1}, m_t, z_t)$
    Update memory with z_t by Methods Eq. 2
    $R_t, o_t^r = \text{dec}(z_t, \pi_t, a_t)$
    Apply a_t to environment and receive reward r_t and observation o_{t+1}
    $t := t + 1; T := T + 1$
  until environment termination or $t - t_\text{start} == \tau_\text{window}$
  If not terminated, run additional step to compute $V_\nu^\pi(z_{t+1}, \log \pi_{t+1})$ and set $R_{t+1} := V^\pi(z_{t+1}, \log \pi_{t+1})$ // (but don't increment counters)
  Reset performance accumulators $\mathcal{A} := 0; \mathcal{L} := 0; \mathcal{H} := 0$
  for k from t down to t_start **do**
    $$\gamma_t := \begin{cases} 0, & \text{if } k \text{ is environment termination} \\ \gamma, & \text{otherwise} \end{cases}$$
    $R_k := r_k + \gamma_t R_{k+1}$
    $\delta_k := r_k + \gamma_t V^\pi(z_{k+1}, \log \pi_{k+1}) - V^\pi(z_k, \log \pi_k)$
    $A_k := \delta_k + (\gamma\lambda) A_{k+1}$
    $\mathcal{L} := \mathcal{L} + \mathcal{L}_k$ (Eq. 7)
    $\mathcal{A} := \mathcal{A} + A_k \log \pi_k[a_k]$
    $\mathcal{H} := \mathcal{H} - \alpha_\text{entropy} \sum_i \pi_k[i] \log \pi_k[i]$ (Entropy loss)
  end for
  $d\chi' := \nabla_{\chi'} \mathcal{L}$
  $d\theta' := \nabla_{\theta'} (\mathcal{A} + \mathcal{H})$
  Asynchronously update via gradient ascent θ using $d\theta'$ and χ using $d\chi'$
until $T > T_\text{max}$

Algorithm 1 of "Unsupervised Predictive Memory in a Goal-Directed Agent", https://arxiv.org/abs/1803.10760

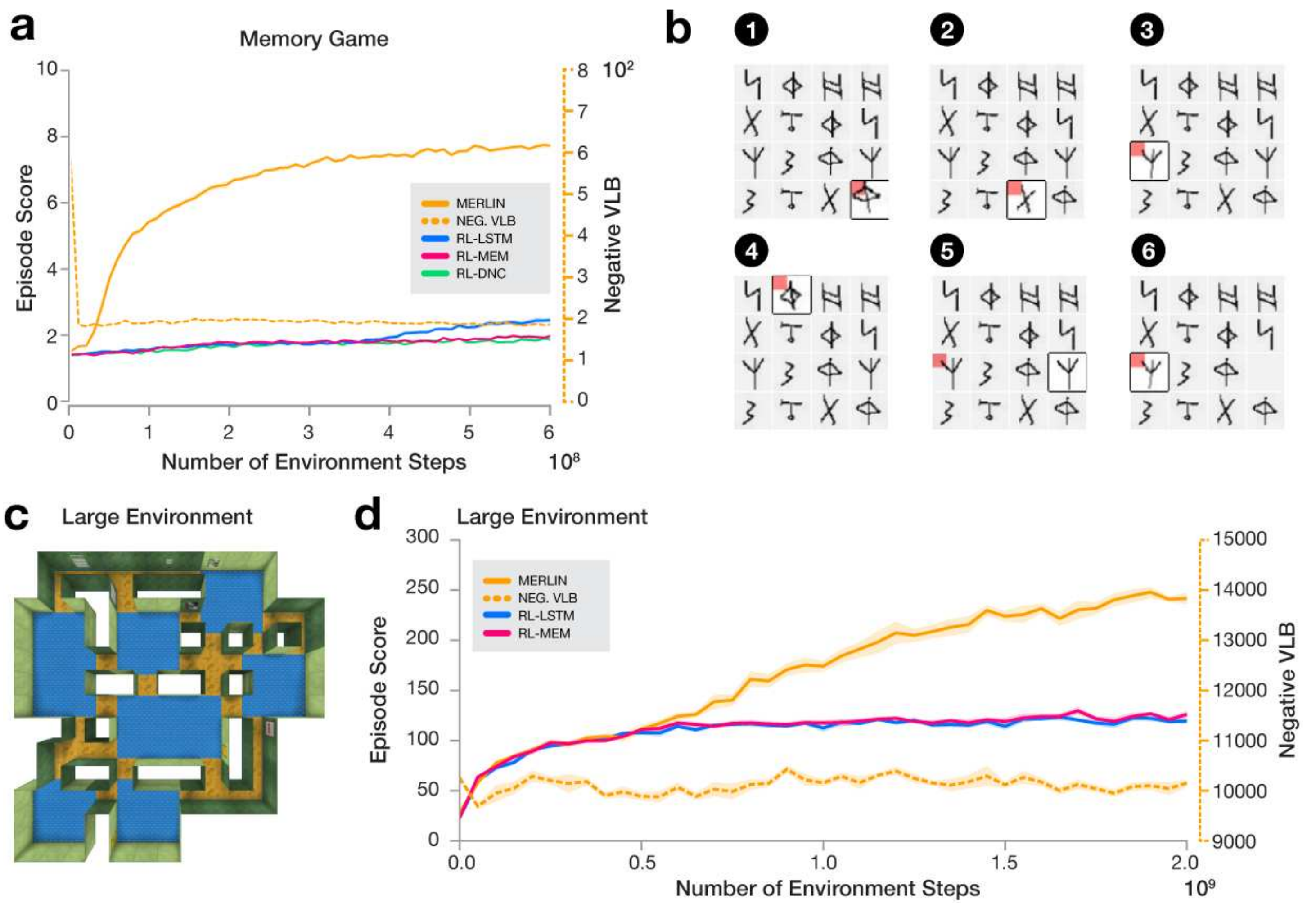


Figure 2 of "Unsupervised Predictive Memory in a Goal-Directed Agent", https://arxiv.org/abs/1803.10760

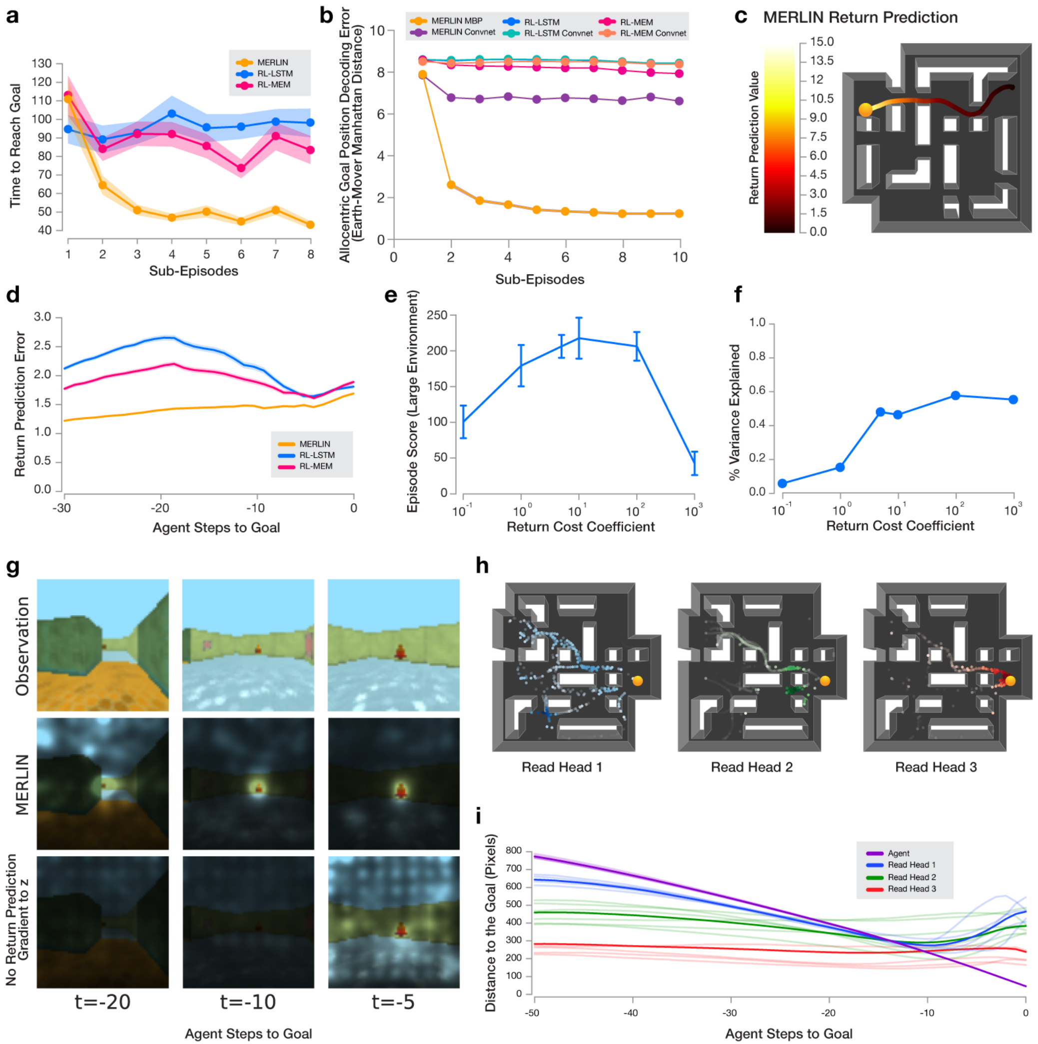
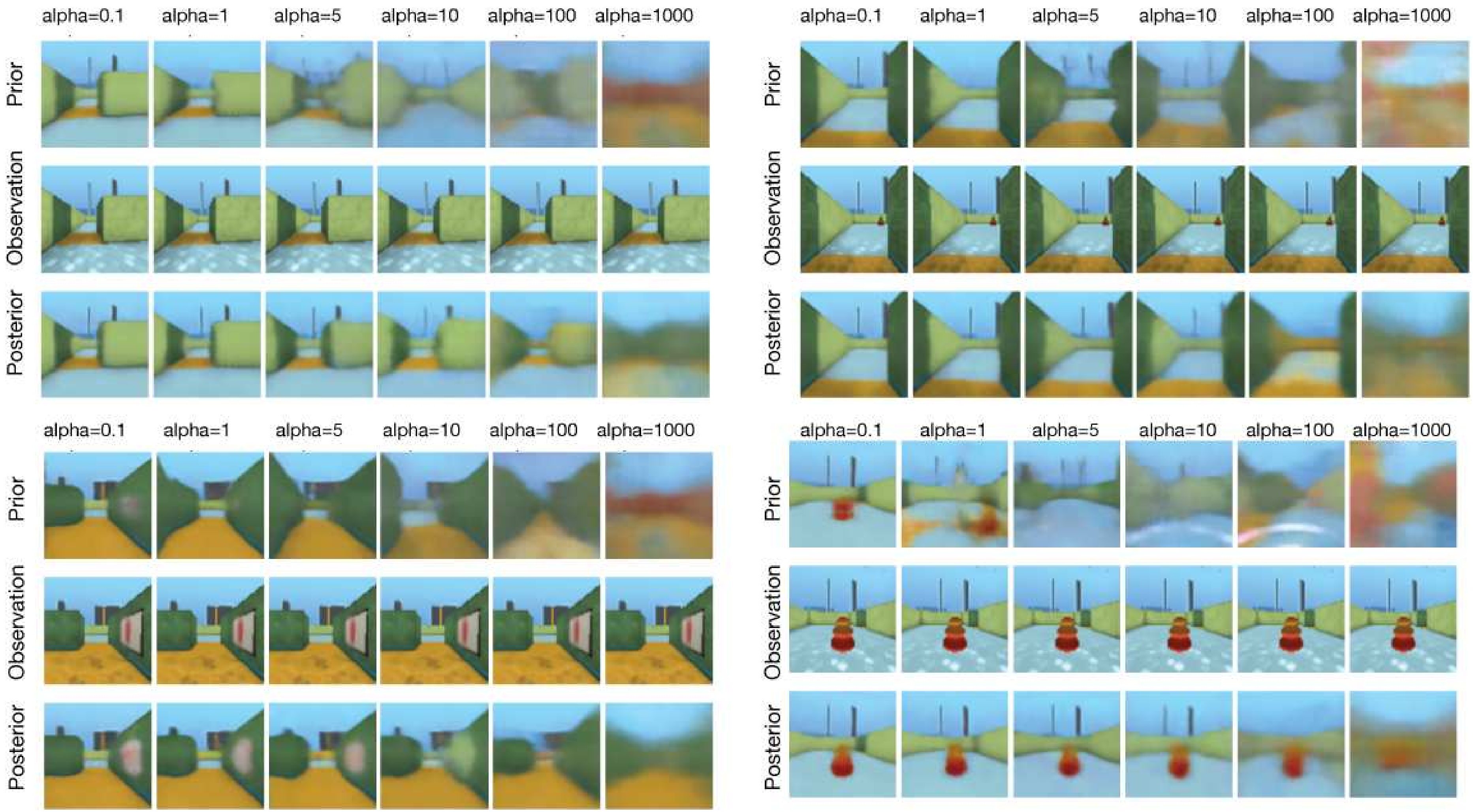


Figure 3 of "Unsupervised Predictive Memory in a Goal-Directed Agent", https://arxiv.org/abs/1803.10760



Extended Figure 3 of "Unsupervised Predictive Memory in a Goal-Directed Agent", https://arxiv.org/abs/1803.10760

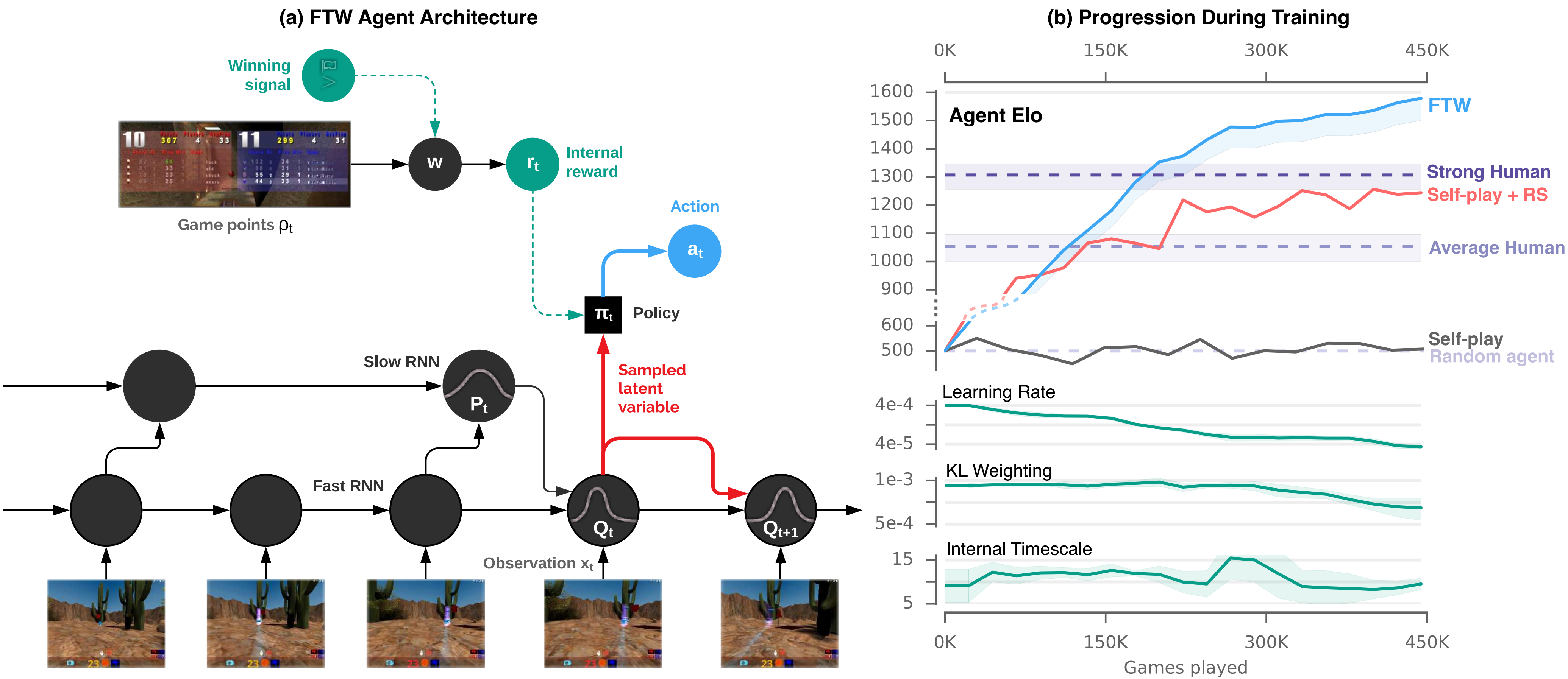


Figure 2 of "Human-level performance in first-person multiplayer games with population-based deep reinforcement learning" by Max Jaderber et al.

For the Win agent for Capture The Flag

- Extension of the MERLIN architecture.
- Hierarchical RNN with two timescales.
- V-Trace with both clipping factors set to 1 is used.
- Rewards for 13 pre-defined events (picking a flag, returning a flag, tagging/being tag with/without a flag, …) are learned by the agent.
- Population based training controlling KL divergence penalty weights, internal dense rewards, slow ticking RNN speed, and gradient flow factor from fast to slow RNN.

  In every game, teams of similarly skilled agents were selected, and the authors state it is crucial to employ several agents instead of just one (30 simultaneously trained agents are used).

For the Win agent for Capture The Flag

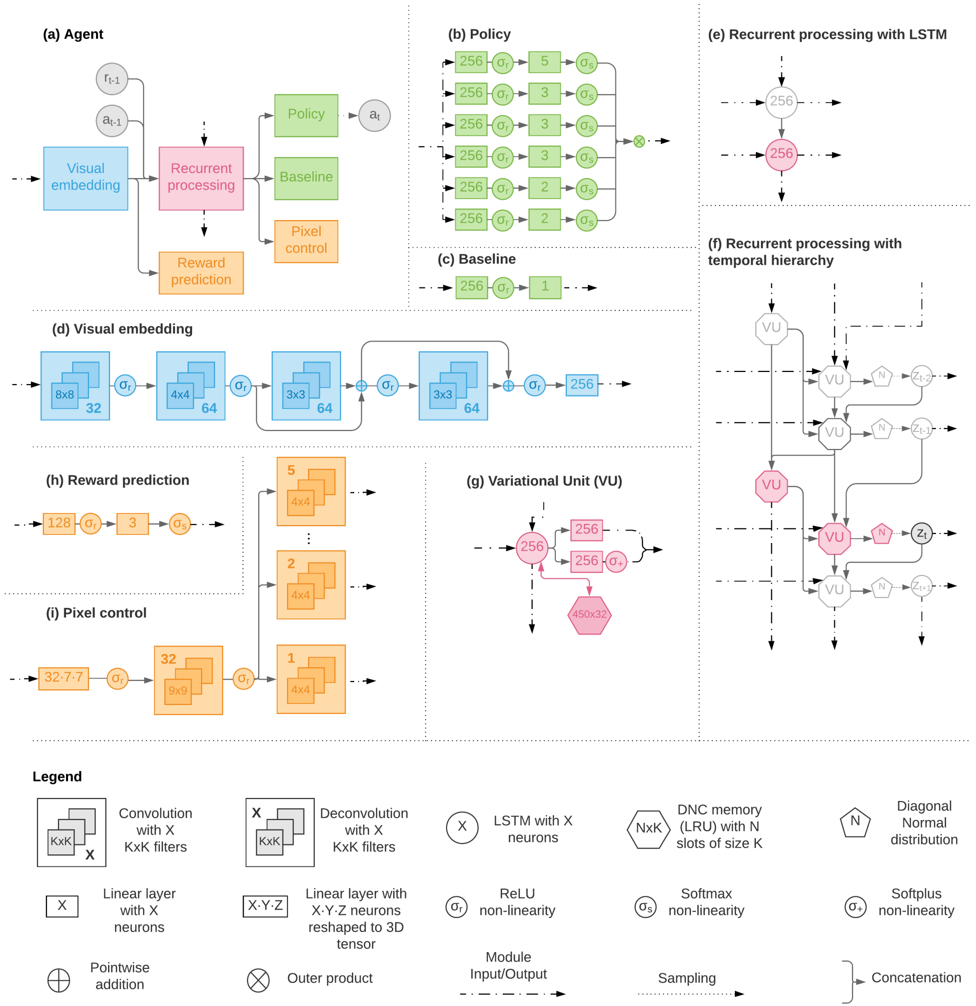


Figure S10 of "Human-level performance in first-person multiplayer games with population-based deep reinforcement learning" by Max Jaderber et al.

For the Win agent for Capture The Flag

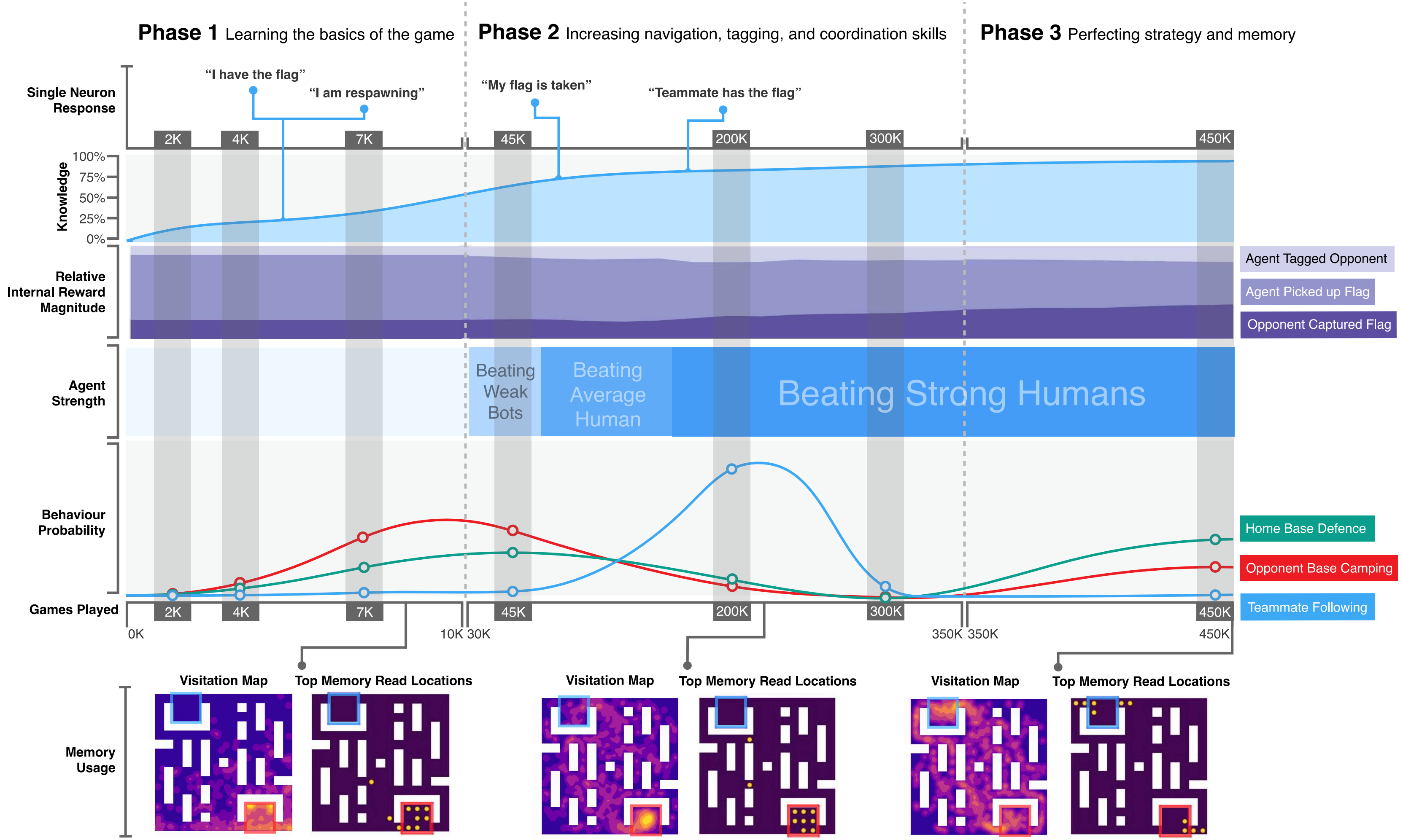


Figure 4 of "Human-level performance in first-person multiplayer games with population-based deep reinforcement learning" by Max Jaderber et al.