Deep Reinforcement Learning

Lecture 6: Multi-Agent Reinforcement Learning

Pascal Leroy

pleroy@uliege.be

university of
groningen

Multi-Agent RL stories:

- Reinforcement basics (SARL)
- Multi-agent RL framework
- Cooperative scenarios
- Communication
- Competitive scenarios
- Adversarial attacks
- References

RL basics: MDP

A Markov decision process (MDP) is defined by:

- A set of states $s \in \mathcal{S}$.
- A set of actions $u \in \mathcal{U}$.
- Transition function: $s_{t+1} \sim P(s_{t+1}|s_t, u_t)$.
- Reward function: $r_t = R(s_{t+1}, s_t, u_t)$.
- Policy: $\pi(u_t|s_t)$

The agent goal is to maximize its total expected sum of (discounted) rewards:

$$\mathbb{E}_\pi \left[\sum_t \gamma^t r_t\right] \text{ with } \gamma \in [0, 1)$$

RL basics: DQN

Q-learning with a neural network parametrised by θ:

$$\mathcal{L}(\theta) = \mathbb{E}_{\langle s_t, u_t, r_t, s_{t+1} \rangle \sim B} \left(r_t + \gamma \max_{u \in \mathcal{U}} Q(s_{t+1}, u; \theta') - Q(s_t, u_t; \theta) \right)^2$$

- The replay buffer B is a collection of transitions.
- Sampling transitions allows to update the network.
- θ' denotes the parameters of the target network, a copy of θ that is periodically updated.
- To play Atari games, θ is a CNN.
- When the environment is partially observable (POMDP), θ is a recurrent network (DRQN) and B stores sequences of transitions.

RL basics: Actor-Critic

Advantage Actor-Critic, the baseline is the Value function:

Actor θ, learns the policy:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\left(\sum_t^T A(s_t, u_t; \phi) \nabla_\theta \log \pi_\theta(u_t, s_t) \right) \right]$$

Critic ϕ, learns the advantage $A(s_t, u_t) = Q(s_t, u_t) - V(s_t)$:

1. $$A(s_t, u_t; \phi) = r_t + \gamma V(s_{t+1}; \phi) - V(s_t; \phi)$$
2. $$A(s_t, u_t; \phi) = Q(s_t, u_t; \phi) - \sum_u \pi_\theta(s_t, u) Q(s_t, u; \phi)$$

Multi-Agent Reinforcement Learning

Markov Game

Markov Game (also referred to as stochastic Game) $[n, \mathcal{S}, O, \mathcal{Z}, \mathcal{U}, r, P, \gamma]$:

- A set of n agents, each one is represented by a or $a_i, i \in \{1, ..., n\}$.
- A set of states $s \in \mathcal{S}$.
- An observation function $O : \mathcal{S} \times 1, ..., n \rightarrow \mathcal{Z}$.
- A set of action spaces $\mathcal{U} = \mathcal{U}_1 \times ... \times \mathcal{U}_n$, one per agent $u_t^{a_i} \in \mathcal{U}_i$.
- A transition function: $s_{t+1} \sim P(s_{t+1}|s_t, \mathbf{u_t})$ with $\mathbf{u_t} = (u_t^{a_1}, ..., u_t^{a_n})$.
- A reward function per agent: $r_t^{a_i} = R^{a_i}(s_{t+1}, s_t, \mathbf{u_t})$.
- Agents sometimes store their history $\tau_t^a \in (\mathcal{Z} \times \mathcal{U})^t$.
- The goal of each agent a_i is to maximize its total expected sum of (discounted) rewards

$$\mathbb{E}_\pi \left[\sum_t \gamma^t r_t^{a_i}\right] \text{ with } \gamma \in [0, 1)$$

In Multi-agent settings, the goal of each agent may differ:

1. Cooperative setting: all agents share a common goal.

   Examples: traffic control, robotics teams,...

2. Competitive setting: the gain of an agent equals the loss of other agents.

   Often referred as zero-sum setting, because the sum of rewards of all agents sums to zero.

   Examples: 1v1 board games, 1v1 video games,...

3. General sum setting: lies in between the two others.

   Examples: everything else that is not cooperative or competitive, 5v5 video games,...

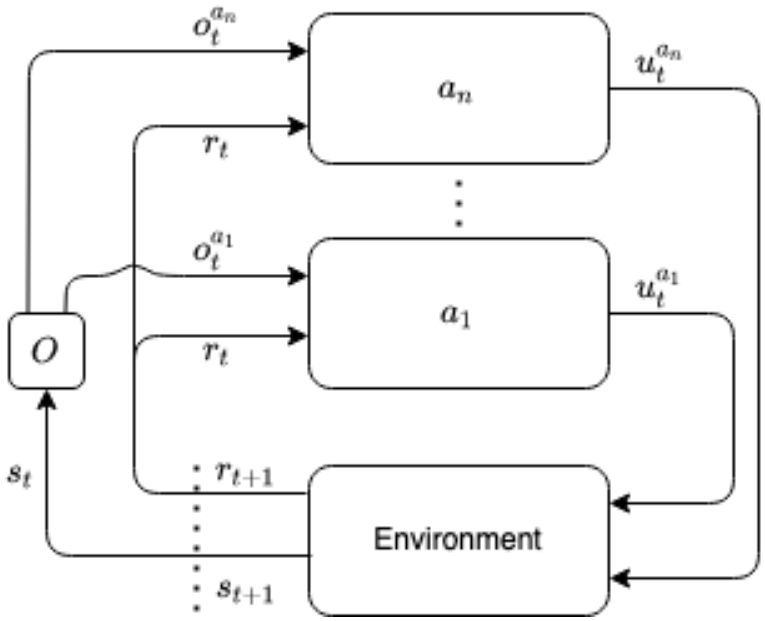
Cooperative setting

Dec-POMDP

In a cooperative setting, it is possible to have a single reward function, each agent receives a same global reward:

$$r_t^{a_1} = r_t^{a_n} = r_t = R(s_{t+1}, s_t, \mathbf{u_t}) : \mathcal{S}^2 \times \mathcal{U} \to \mathbb{R}$$

Such Markov Games are called Decentralised-POMDP.



StarCraft multi-agent challenge

- SMAC is a Dec-POMDP environment based on StarCraft 2.
- All agents learn to cooperate against the built-in AI.
- This is not a competitive setting because the built-in AI is stationary.

Centralised controller

Centralised controller:

- One agent controls all actions.
- A single joint actions space $\mathcal{U}_1 \times ... \times \mathcal{U}_n$.

Problems:

Centralised controller

Centralised controller:

- One agent controls all actions.
- A single joint actions space $\mathcal{U}_1 \times \ldots \times \mathcal{U}_n$.

Problems:

- Joint actions space scales exponentially with n.
- What about the partial observability? $\rightarrow$ Not possible to centralise.

Solutions:

Centralised controller

Centralised controller:

- One agent controls all actions.
- A single joint actions space $\mathcal{U}_1 \times \ldots \times \mathcal{U}_n$.

Problems:

- Joint actions space scales exponentially with n.
- What about the partial observability? $\rightarrow$ Not possible to centralise.

Solutions:

- Decentralised controller.
  - Naive learner: train each agent with SARL methods.
- Centralised training with decentralised execution (CTDE).
  - Benefit from supplementary information during training, such as the entire state of the game.

Naive learner

Naive learning:

- Ignore the fact that there are multiple learning agents.
- Provide a first baseline to compare algorithms.
- Easy to implement.
- Not so young: a tabular version with IQL (Tan 1993).

Challenges:

Naive learner

Naive learning:

- Ignore the fact that there are multiple learning agents.
- Provide a first baseline to compare algorithms.
- Easy to implement.
- Not so young: a tabular version with IQL (Tan 1993).

Challenges:

- Non-stationarity:
  - Other agents are also learning and their policy changes over time.
- Credit assessment:
  - How an agent learns whether its actions is the one that lead to good (or bad) reward?
  - How an agent maximises the joint actions reward knowing only its action?

Value-based methods in CTDE

Independent Q-Learning (IQL):

- Each agent learns its individual $Q_a(\tau_t^a, u_t^a)$ independently.

Problem:

Value-based methods in CTDE

Independent Q-Learning (IQL):

- Each agent learns its individual $Q_a(\tau_t^a, u_t^a)$ independently.

Problem:

- How to ensure that $\mathbf{arg\,max}_{u_t^a} Q_a(\tau_t^a, u_t^a)$ maximises $Q(\boldsymbol{s}_t, \mathbf{u_t})$?

Solution:

Value-based methods in CTDE

Independent Q-Learning (IQL):

- Each agent learns its individual $Q_a(\tau_t^a, u_t^a)$ independently.

Problem:

- How to ensure that $\mathbf{arg\,max}_{u_t^a} Q_a(\tau_t^a, u_t^a)$ maximises $Q(\boldsymbol{s}_t, \mathbf{u_t})$?

Solution:

- Learn $Q(\boldsymbol{s}_t, \mathbf{u_t})$ as a function of all $Q_a(\tau_t^a, u_t^a)$ during training.

Individual Global Max

Learn $Q(s_t, \mathbf{u_t})$ as a function of all $Q_a(\tau_t^a, u_t^a)$ during training.

Condition: Individual Global Max (IGM)

$$\arg\max_{\mathbf{u_t}} Q(s_t, \mathbf{u_t}) = \begin{pmatrix} \arg\max_{u_t^{a_1}} Q_1(\tau_t^{a_1}, u_t^{a_1}) \\ . \\ . \\ . \\ \arg\max_{u_t^{a_n}} Q_n(\tau_t^{a_n}, u_t^{a_n}) \end{pmatrix}$$

Q_a is not a Q function anymore, but a utility function used to select actions.

Question: How to satisfy IGM?

VDN

How to satisfy IGM?

Value Decomposition Network:

$$Q(\boldsymbol{s}_t, \mathbf{u_t}) = \sum_{i=1}^{n} Q_{a_i}(\tau_t^{a_i}, u_t^{a_i})$$

Problems:

VDN

How to satisfy IGM?

Value Decomposition Network:

$$Q(\boldsymbol{s}_t, \mathbf{u_t}) = \sum_{i=1}^{n} Q_{a_i}(\tau_t^{a_i}, u_t^{a_i})$$

Problems:

- Addition does not allow to build complex functions.
- Current state information $\boldsymbol{s}_t$ is not considered.

How can we build non-linear factorisation satisfying IGM?

QMIX

How to build non-linear factorisation satisfying IGM?

QMIX enforces monotonicity:

$$\frac{\partial Q(\boldsymbol{s}_t, \mathbf{u_t})}{\partial Q_a(\tau_t^a, u_t^a)} \geq 0 \; \forall a \in a_1, .., a_n$$

How to build a non-linear monotonic factorisation of $Q(\boldsymbol{s}_t, \mathbf{u_t})$ as a function of every $Q_a(\tau_t^a, u_t^a)$ and $\boldsymbol{s}_t$ with neural networks?

QMIX

How to build non-linear factorisation satisfying IGM?

QMIX enforces monotonicity:

$$\frac{\partial Q(s_t, \mathbf{u_t})}{\partial Q_a(\tau_t^a, u_t^a)} \geq 0 \; \forall a \in a_1, .., a_n$$

How to build a non-linear monotonic factorisation of $Q(s_t, \mathbf{u_t})$ as a function of every $Q_a(\tau_t^a, u_t^a)$ and s_t with neural networks?

In QMIX, monotonicity is ensured by constraining a hypernetwork that computes the weights of a second neural network.

QMIX architecture

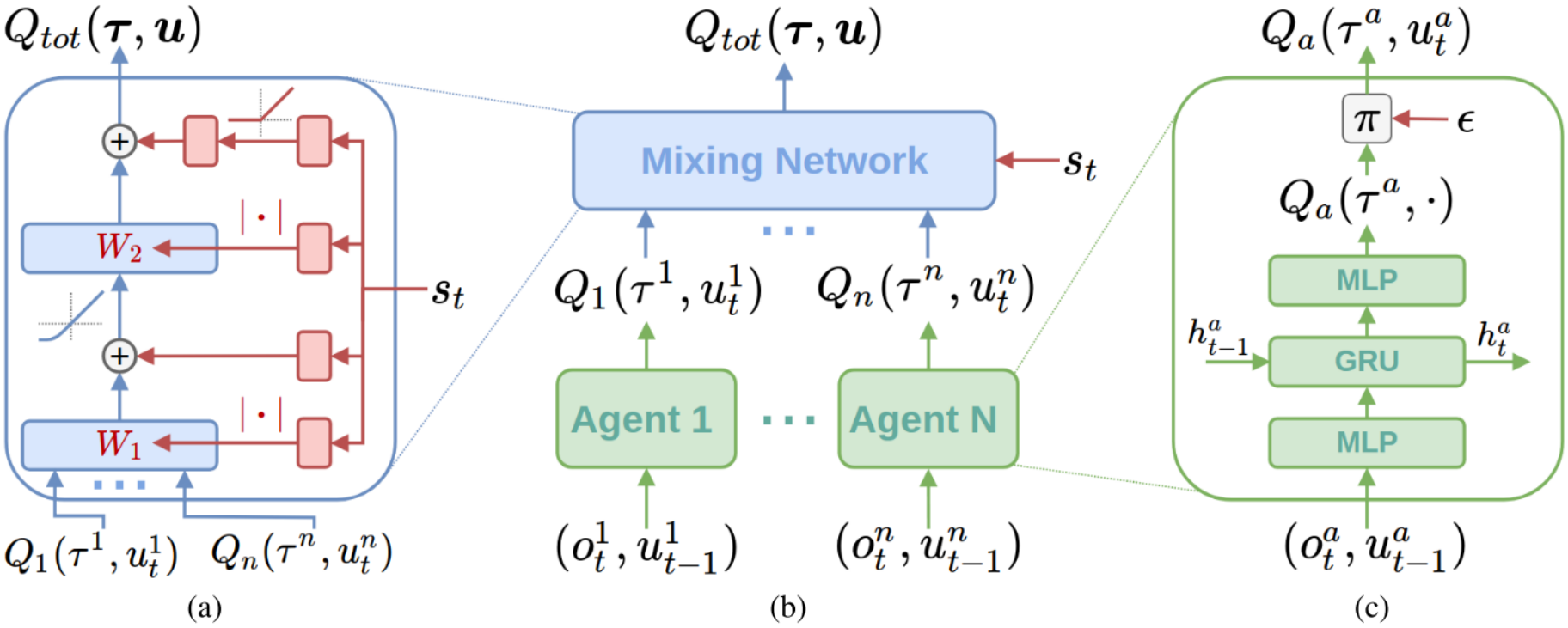


Figure 2. (a) Mixing network structure. In red are the hypernetworks that produce the weights and biases for mixing network layers shown in blue. (b) The overall QMIX architecture. (c) Agent network structure. Best viewed in colour.

QMIX

In QMIX:

- A hypernetwork h_p takes the state s_t as input and computes the weights W1 and W2 of a second neural network.
- These weights are constrained to be positive and then used in a feed forward network h_o to factorise $Q(s_t, \mathbf{u_t})$ with the individual Q_a.
- A neural network made of monotonic functions and strictly positive weights is monotonic with respect to its inputs.

$$\rightarrow Q_{mix}(s_t, \mathbf{u_t}) = h_o\left(Q_{a_1}(), .., Q_{a_n}(), h_p(s_t)\right)$$

The optimisation procedure follows the same principles of DQN algorithm:

$$\mathcal{L}(\theta) = \mathbb{E}_{\langle . \rangle \sim B}\left[\left(r_t + \gamma \max_{\mathbf{u} \in \mathcal{U}} Q_{mix}(s_{t+1}, \mathbf{u}; \theta') - Q_{mix}(s_t, \mathbf{u_t}; \theta)\right)^2\right]$$

Parameters of individual networks are commonly shared to speed up learning.

QMIX results

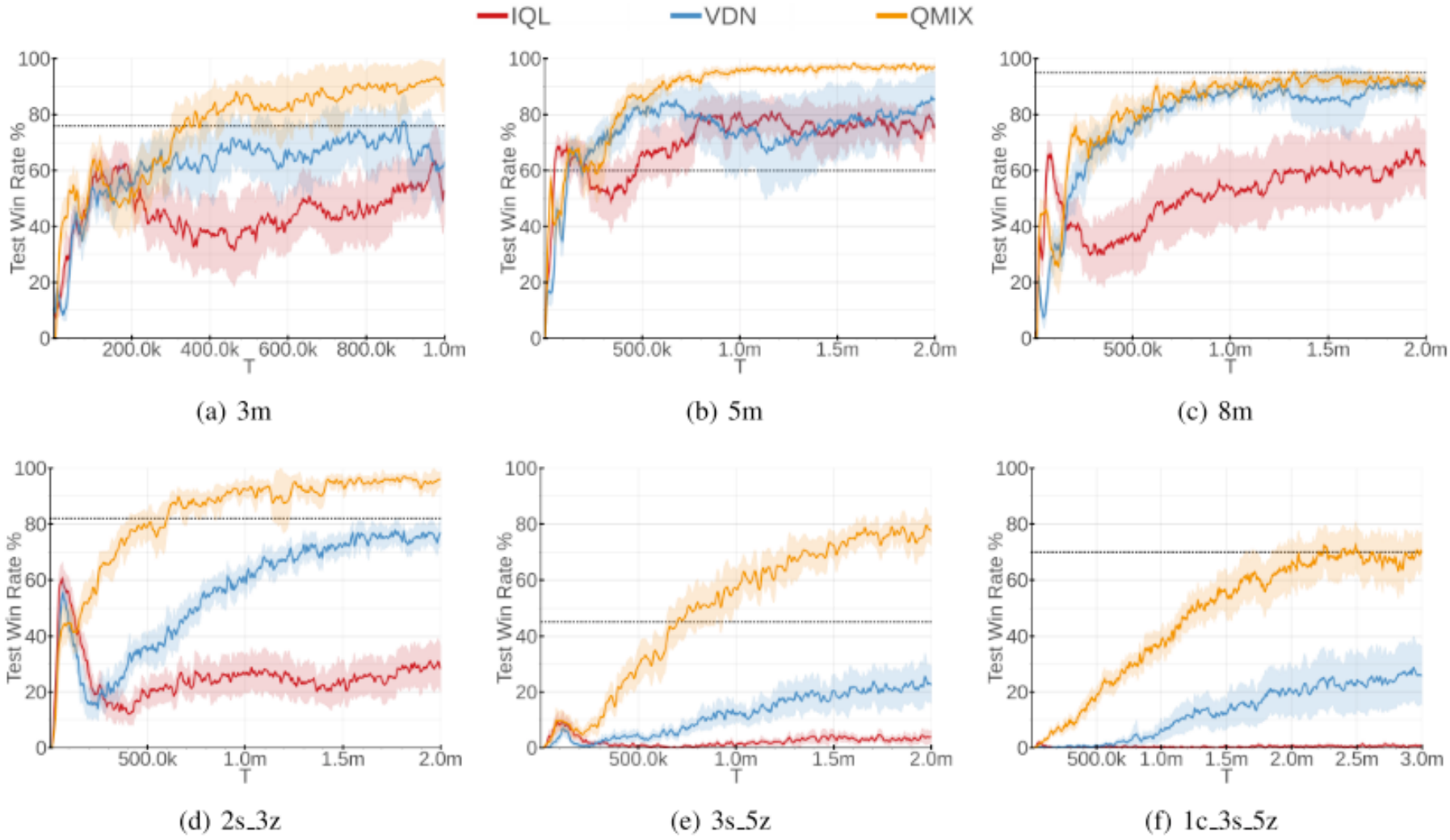


Figure 3. Win rates for IQL, VDN, and QMIX on six different combat maps. The performance of the heuristic-based algorithm is shown as a dashed line.

DQV

Deep-Quality Value: learn $Q(.;\theta)$ and $V(.;\phi)$ at the same time.

$$\mathcal{L}(\theta) = \mathbb{E}_{\langle s_t, u_t, r_t, s_{t+1} \rangle \sim B} \left[\left(r_t + \gamma V(s_{t+1}; \phi') - Q(s_t, u_t; \theta) \right)^2 \right]$$

$$\mathcal{L}(\phi) = \mathbb{E}_{\langle s_t, u_t, r_t, s_{t+1} \rangle \sim B} \left[\left(r_t + \gamma V(s_{t+1}; \phi') - V(s_t; \phi) \right)^2 \right]$$

Benefit of DQV is to reduce the overestimation problem of DQN, linked to the max operator.

DQN loss reminder:

$$\mathcal{L}(\theta) = \mathbb{E}_{\langle s_t, u_t, r_t, s_{t+1} \rangle \sim B} \left[\left(r_t + \gamma \max_{u \in \mathcal{U}} Q(s_{t+1}, u; \theta') - Q(s_t, u_t; \theta) \right)^2 \right]$$

QVMix

QVMix and QVMix-Max are extensions of the Deep-Quality value family of algorithms to cooperative multi-agent.

QVMix:

$$\mathcal{L}(\theta) = \mathbb{E}_{\langle . \rangle \sim B}\left[\left(r_t + \gamma V(\boldsymbol{s}_{t+1}; \phi') - Q(\boldsymbol{s}_t, \mathbf{u_t}; \theta)\right)^2\right]$$

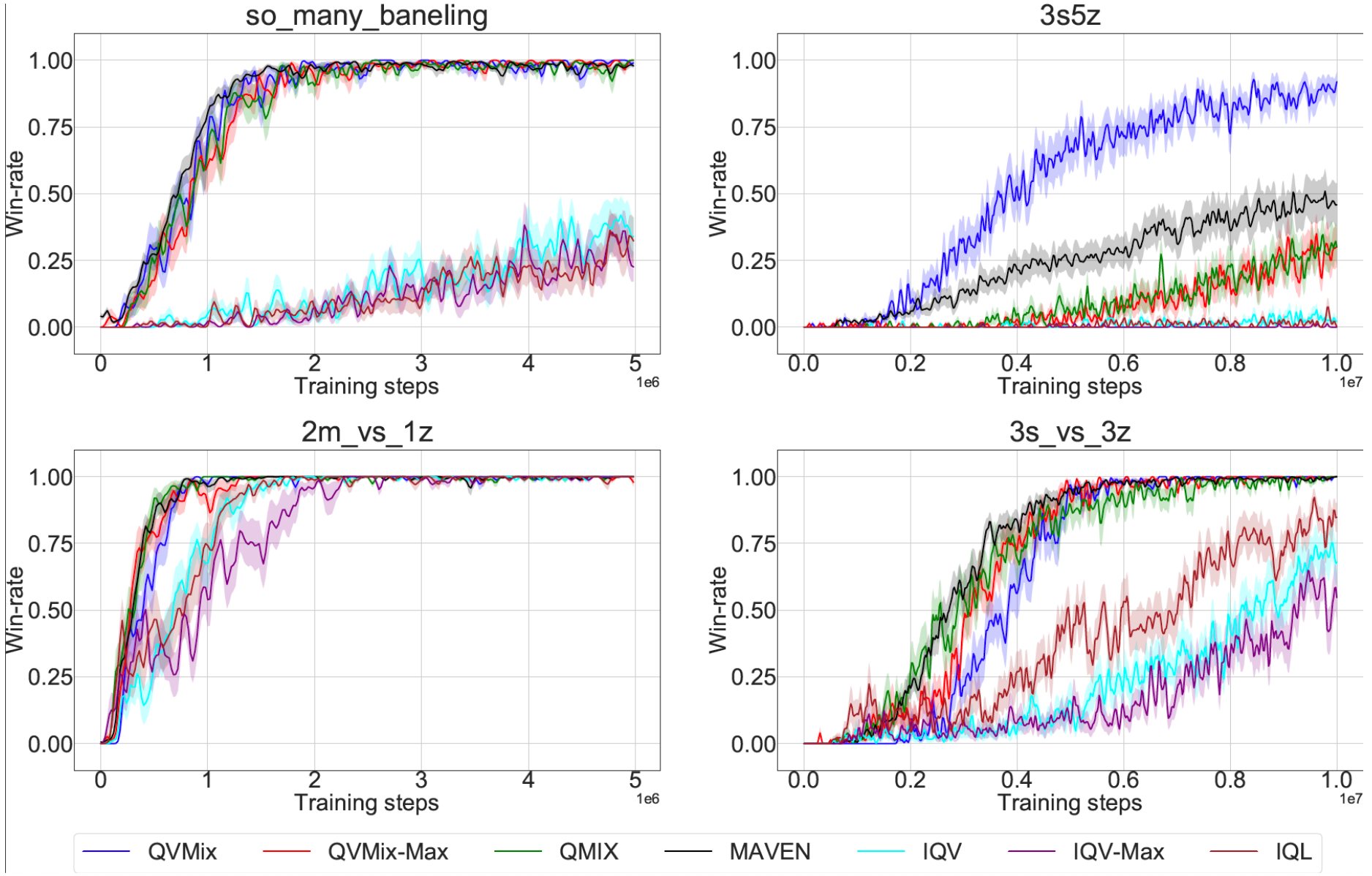
$$\mathcal{L}(\phi) = \mathbb{E}_{\langle . \rangle \sim B}\left[\left(r_t + \gamma V(\boldsymbol{s}_{t+1}; \phi') - V(\boldsymbol{s}_t; \phi)\right)^2\right]$$

QVMix-Max:

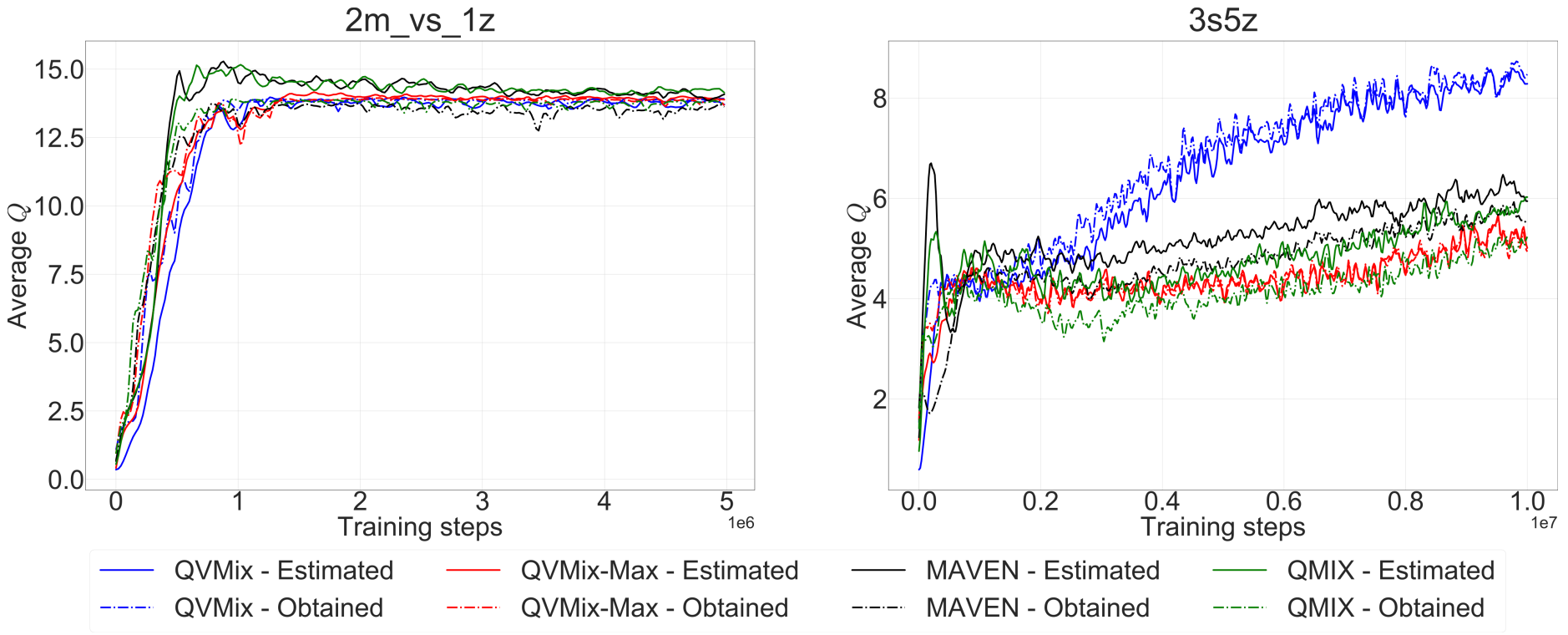
$$\mathcal{L}(\phi) = \mathbb{E}_{\langle . \rangle \sim B}\left[\left(r_t + \gamma \max_{\mathbf{u} \in \mathcal{U}} Q(\boldsymbol{s}_{t+1}, \mathbf{u}; \theta') - V(\boldsymbol{s}_t; \phi)\right)^2\right]$$

In QVMix, the architecture of V and Q are the same as that of Q in QMIX, except that V has a single output.

QVMix results



QVMix overestimation bias



Policy-based methods in CTDE

Naive learner: Independent Actor-Critic (IAC)

- Each agent learns its actor and critic independently.

Two possible critics:

- IAC-V: $A(\tau_t, u_t; \phi) = r + \gamma V(\tau_{t+1}; \phi) - V(\tau_t; \phi)$.
- IAC-Q: $A(\tau_t, u_t; \phi) = Q(\tau_t, u_t; \phi) - \sum_{u^a} \pi_\theta(\tau_t, u^a) Q(\tau_t, u^a; \phi)$.

Problem: How to benefit from centralised information such as $\boldsymbol{s}_t$?

COMA

Problem: How to benefit from centralised information such as $\boldsymbol{s}_t$?

Solutions proposed by Foerster, et. al. (2018) (COMA):

- Critic is only used during training.
    - Centralised critic that computes the advantage based on $\boldsymbol{s}_t$.

$$A(\boldsymbol{s}_t, u_t^a; \phi) = r_t + \gamma V(\boldsymbol{s}_{t+1}; \phi) - V(\boldsymbol{s}_t; \phi)$$

Problems:

COMA

Problem: How to benefit from centralised information such as $\boldsymbol{s}_t$?

Solutions proposed by Foerster, et. al. (2018) (COMA):

- Critic is only used during training.
    - Centralised critic that computes the advantage based on $\boldsymbol{s}_t$.

$$A(\boldsymbol{s}_t, u_t^a; \phi) = r_t + \gamma V(\boldsymbol{s}_{t+1}; \phi) - V(\boldsymbol{s}_t; \phi)$$

Problems:

- Based on global rewards $\boldsymbol{r}_t$.
- The centralised critic does not solve the credit assignment problem.

COMA

Solution: use a counterfactual baseline.

- Inspired from difference reward:

$$D^a = r(\boldsymbol{s}, \mathbf{u}) - r(\boldsymbol{s}, (\mathbf{u}^{-a}, c^a))$$

- The common reward $r(\boldsymbol{s}, \mathbf{u})$ is compared to a reward obtained when agent a executes a default action c^a, actions of other agents ($\mathbf{u}^{-a}$) unchanged.
- Any action u^a that maximises $r(\boldsymbol{s}, \mathbf{u})$ also maximises D^a.
- Limitations:

COMA

Solution: use a counterfactual baseline.

- Inspired from difference reward:

$$D^a = r(\boldsymbol{s}, \mathbf{u}) - r(\boldsymbol{s}, (\mathbf{u}^{-a}, c^a))$$

- The common reward $r(\boldsymbol{s}, \mathbf{u})$ is compared to a reward obtained when agent a executes a default action c^a, actions of other agents ($\mathbf{u}^{-a}$) unchanged.
- Any action u^a that maximises $r(\boldsymbol{s}, \mathbf{u})$ also maximises D^a.
- Limitations:
    1. A simulator is required to obtain $r(\boldsymbol{s}, (\mathbf{u}^{-a}, c^a))$.
        - But these can be approximated.
    2. Decide which action is c^a.

COMA

Use a centralised critic that computes difference rewards by learning $Q(s, \mathbf{u})$.

For each agent a, the advantage is:

$$A^a(s, \mathbf{u}) = Q(s, \mathbf{u}) - \sum_{u'^a} \pi^a(u'^a|\tau^a) Q(s, (\mathbf{u}^{-\mathbf{a}}, u'^a))$$

Problem: $(|\mathcal{U}_1| * ... * |\mathcal{U}_n|)$ Q values must be computed.

- In practice, a Q network has one ouput for each possible action.
- Here, this leads to $|\mathcal{U}_1| * ... * |\mathcal{U}_n|$ outputs which is impractical.
- Solution: the critic takes as input u^{-a} and computes only $|\mathcal{U}_a|$ outputs.
- This method is only possible for discrete action spaces!
- It is possible to evaluate $\sum_{u'^a} \pi^a(u'^a|\tau^a) Q(s, (\mathbf{u}^{-\mathbf{a}}, u'^a))$
    - Monte Carlo
    - Gaussian policies in continuous action spaces.

COMA architecture

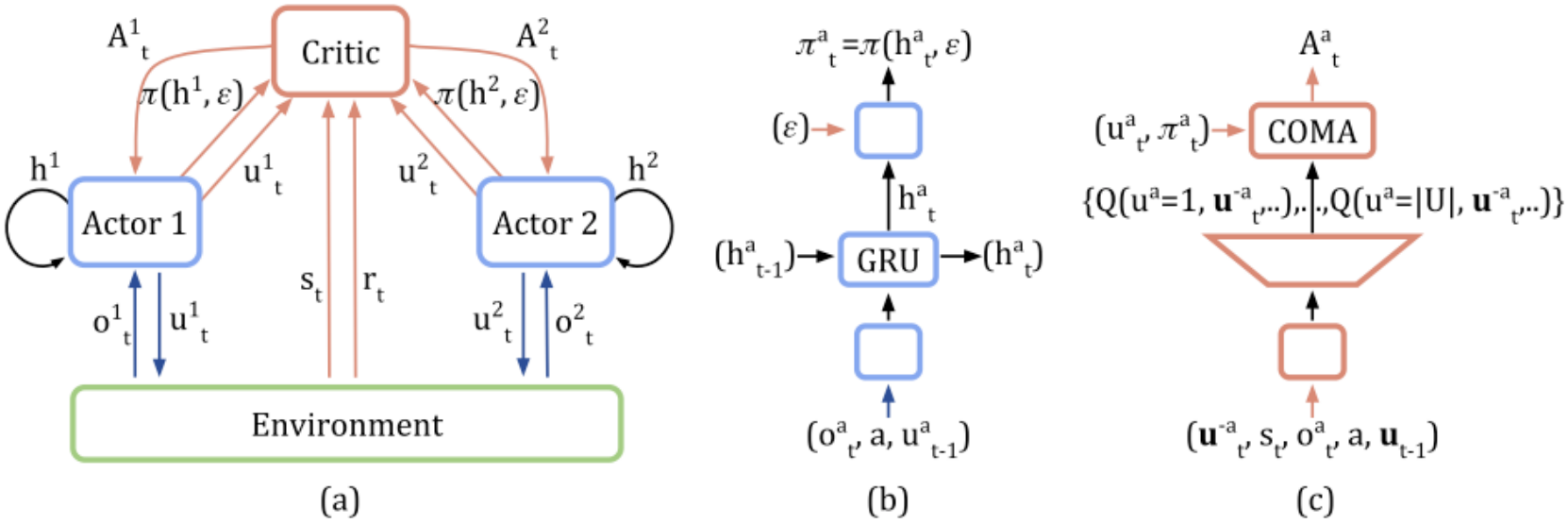


Figure 1: In (a), information flow between the decentralised actors, the environment and the centralised critic in COMA; red arrows and components are only required during centralised learning. In (b) and (c), architectures of the actor and critic.

COMA results

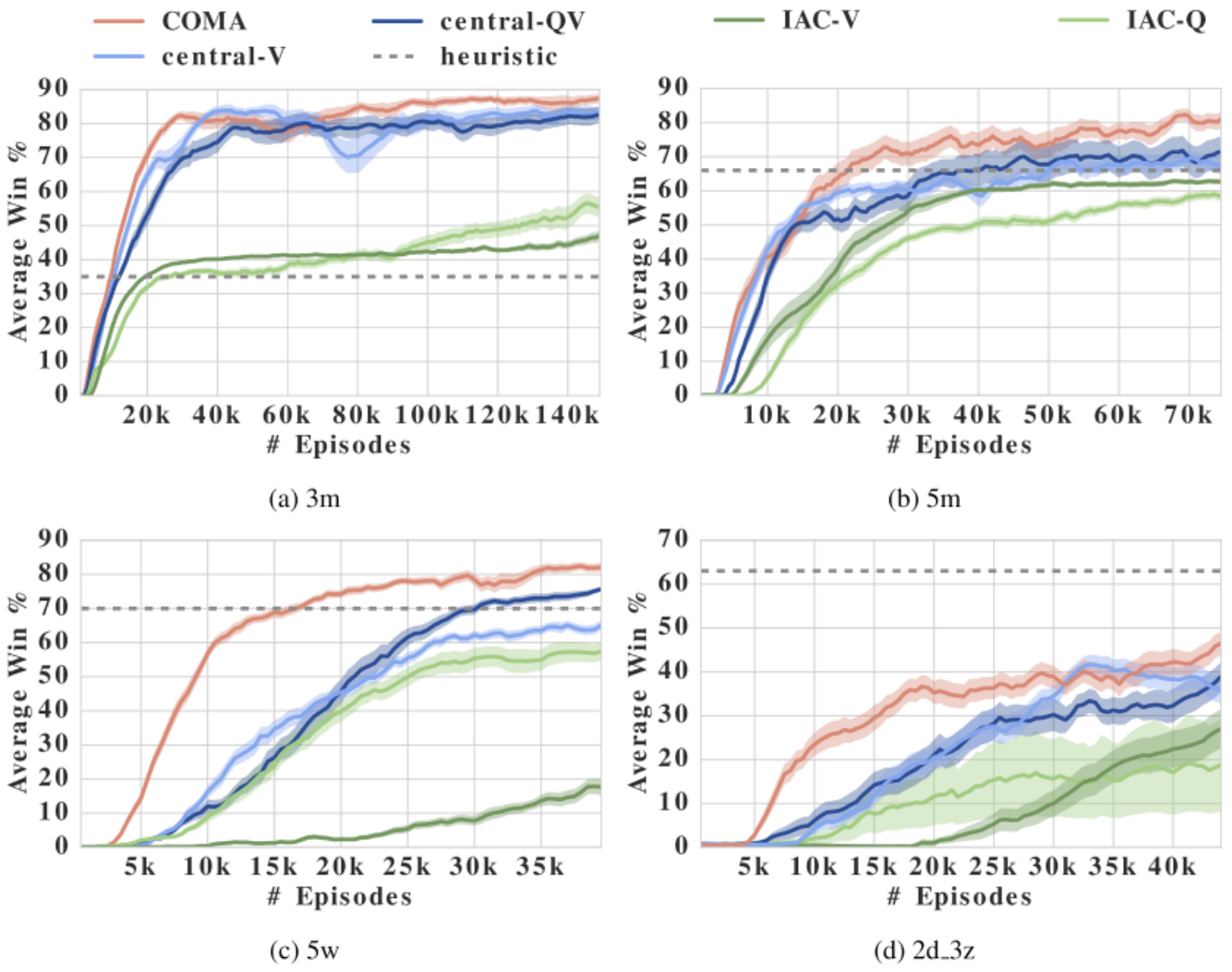
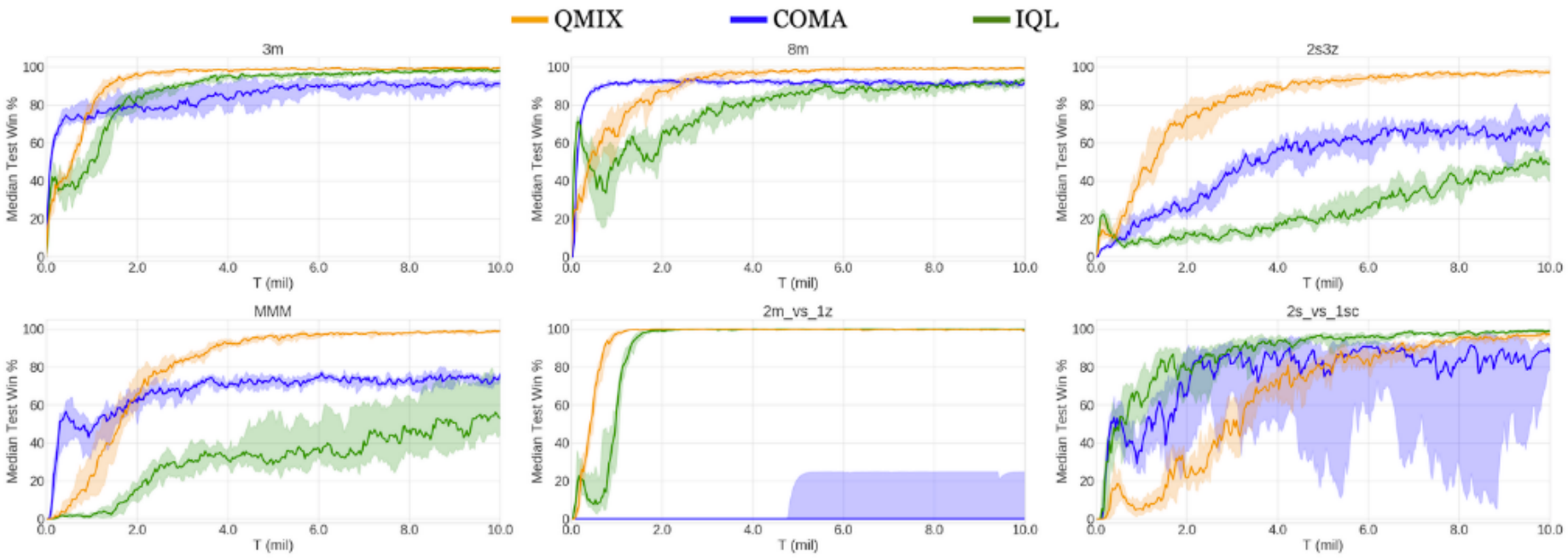


Figure 3: Win rates for COMA and competing algorithms on four different scenarios. COMA outperforms all baseline methods. Centralised critics also clearly outperform their decentralised counterparts. The legend at the top applies across all plots.

COMA vs QMIX


QMIX
COMA
IQL
3m
8m
2s3z
MMM
2m_vs_1z
2s_vs_1sc
Median Test Win %
T (mil)

More methods

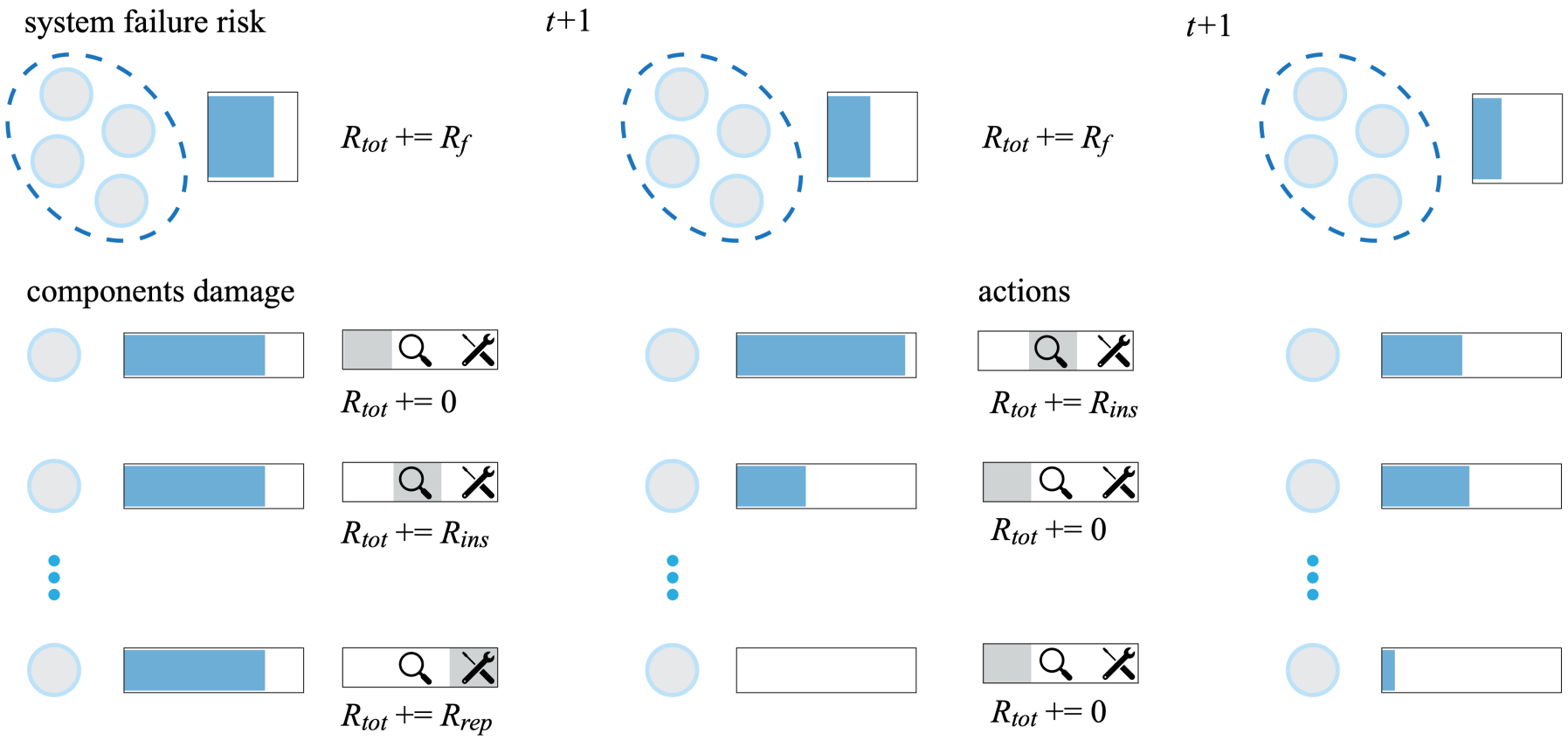
- MAVEN: Multi-agent variational exploration.
- QTRAN: Learning to factorize with transformation for cooperative multi-agent reinforcement learning.
- QPLEX: Duplex Dueling Multi-Agent Q-Learning.
- LIIR: Learning individual intrinsic reward in multi-agent reinforcement learning.
- MADDPG: Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments.
- MAAC: Actor-Attention-Critic for Multi-Agent Reinforcement Learning
- FACMAC: Factored Multi-Agent Centralised Policy Gradients
- HATRPO-HAPPO: Trust Region Policy Optimisation in Multi-Agent Reinforcement Learning

Infrastructure Management Planning

An application for CTDE methods.

- Planning the inspection and maintenance of a structural system can be performed with RL and MARL.
- Joint work in progress with Pablo Morato from the reliability engineering community.
- Goal: Decide of inspection and repair actions over time to reduce costs and risks to maintain a system.
- This can be applied to any type of system: civil, maritime, transportation, and urban management setting.
- In our current work: simulated deterioration and off-shore wind turbines.

IMP sketch


system failure risk
R_{tot} += R_f
components damage
R_{tot} += 0
R_{tot} += R_{ins}
R_{tot} += R_{rep}
t+1
R_{tot} += R_f
actions
R_{tot} += R_{ins}
R_{tot} += 0
R_{tot} += 0
t+1

IMP definition

State and observations:

- Belief on the deterioration state of the part.
- Belief is updated with precision depending weither or not you inspected a component of the system.

Actions:

1. Do-nothing / 2. Inspect / 3. Replace

Reward:

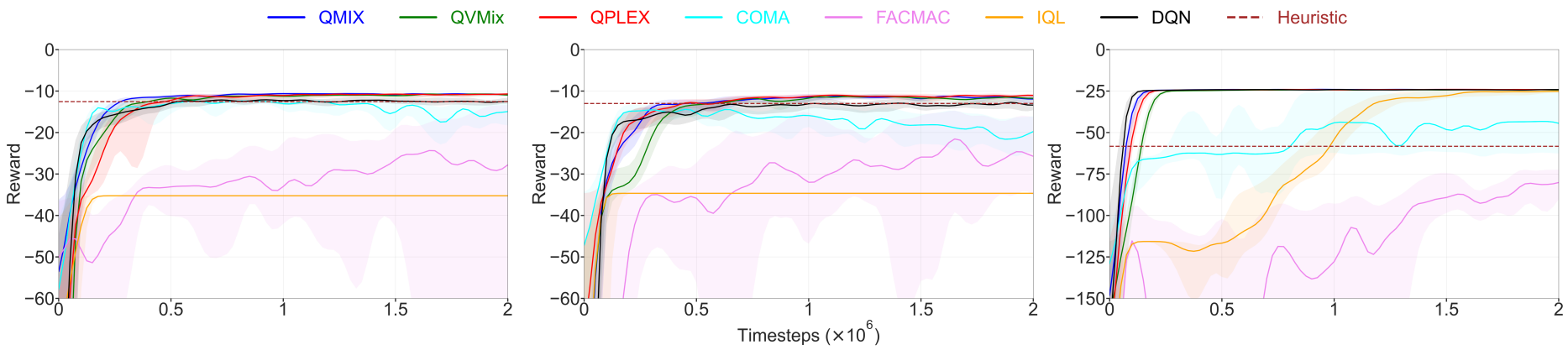
$$R_{tot} = \sum_{t=0}^{T-1} \gamma^t \left[R_{t,f} + \left[\sum_{a=1}^{n} R^a_{t,ins} + R^a_{t,rep} \right] \right]$$

Failure cost is $R_f = c_F \times p_{Fsys}$ encompassing economic, environmental, and societal losses.

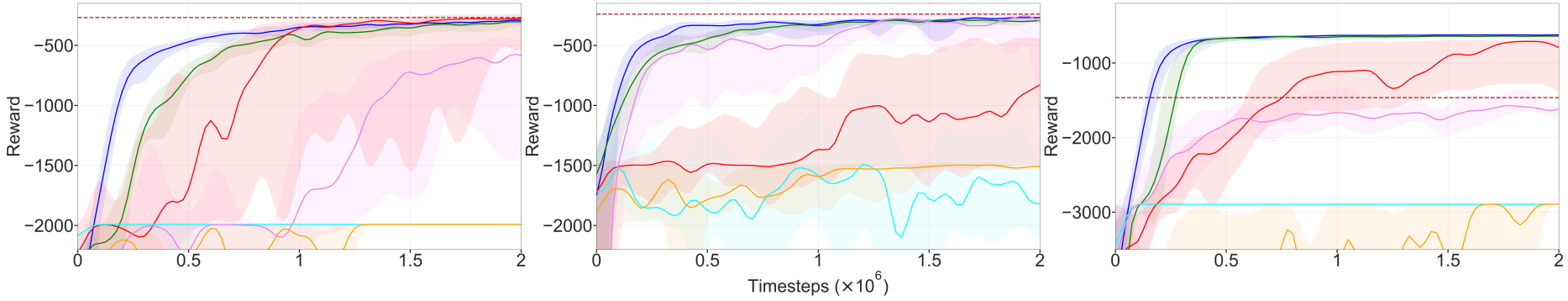
IMP results

Preliminary results on three different systems with 2 or 3 and 50 agents:

k-out-of-n 3 & k-out-of-n correlated 3 & off shore wind farm 3



k-out-of-n 50 & k-out-of-n correlated 50 & off shore wind farm 50

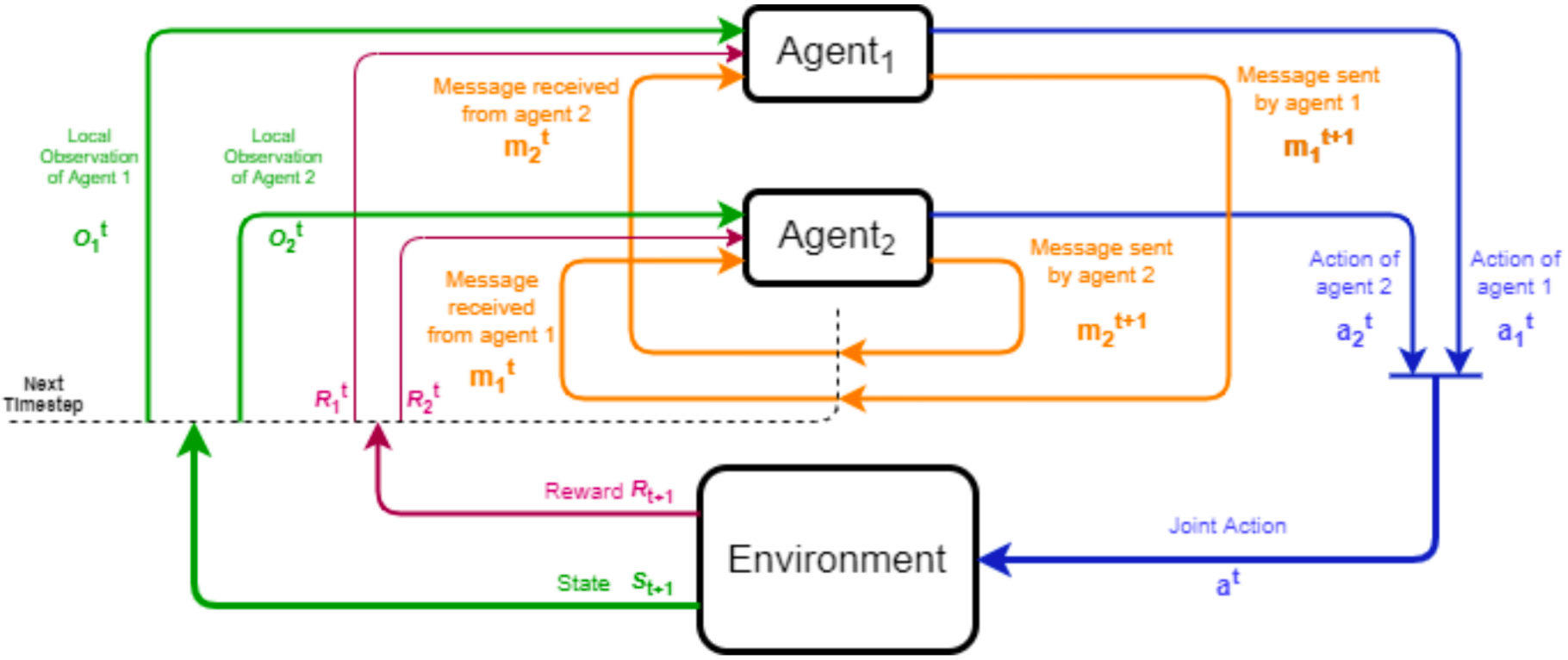


Recap

- Dec-POMDP
- Value-based methods
  - IQL: each agent learns without considering other agents.
  - QMIX: factorise the $Q(\boldsymbol{s}, \mathbf{u})$ as a function of Q^a and monotonicity.
  - QVMIX: learn both Q and V to reduce overestimation of $Q(\boldsymbol{s}, \mathbf{u})$.
- Policy-based methods
  - IAC-V, IAC-Q: each agent learns without considering other agents and with different critics.
  - COMA: centralised critic that computes a counterfactual baseline.
- Inspection and maintenance planning

Communication

Communication



Communication

Communication can be part of the feedback loop.

- Agents take action and send message based on local observations and messages received.

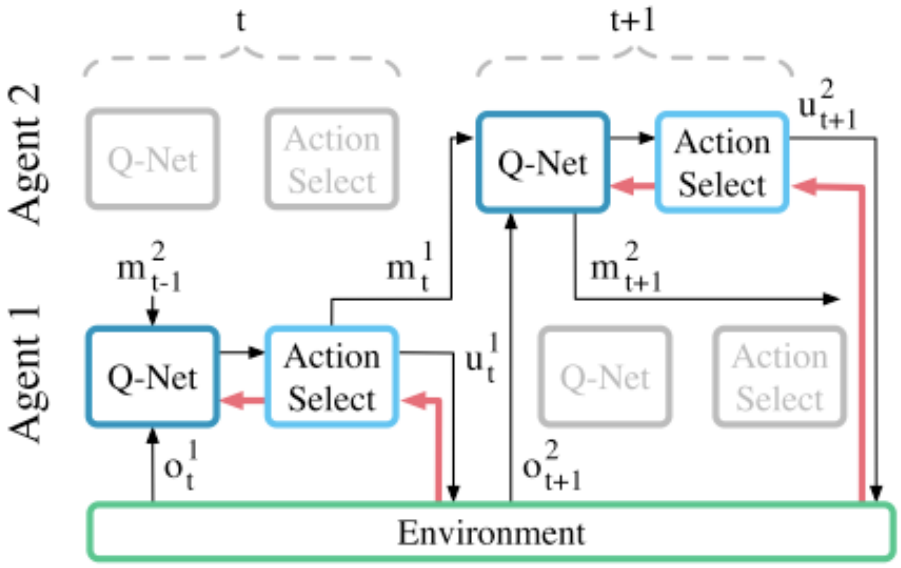
The two "first" methods: RIAL and DIAL:

- Same as a Markov Game.
- No communication protocol: agents must learn it (difficult).
- Agents select discrete communication action $m \in \mathcal{M}$.
- Many details in the paper.

RIAL

Reinforced inter-agent learning (RIAL) is a first approach:

- Agents learn independently $Q^a(o_t^a, m_{t-1}^{-a}, u_t, h_{t-1})$.
- The network outputs is divided in $|\mathcal{U}|$ Q_u^a and $|\mathcal{M}|$ Q_m^a to avoid computing $|\mathcal{U}||\mathcal{M}|$ outputs.
- Actions and messages are chosen separately from Q_u^a and Q_m^a by an action selector.
- Gradient path in red.



(a) RIAL - RL based communication

Limitation: Agents do not provide feedbacks on messages sent by others.

Communication: DIAL

Differentiable inter-agent learning (DIAL).

- Integrate the gradient through the communication channel.
- In RIAL, each agent is trained separately.
- In DIAL, CTDE as training is performed across agents.

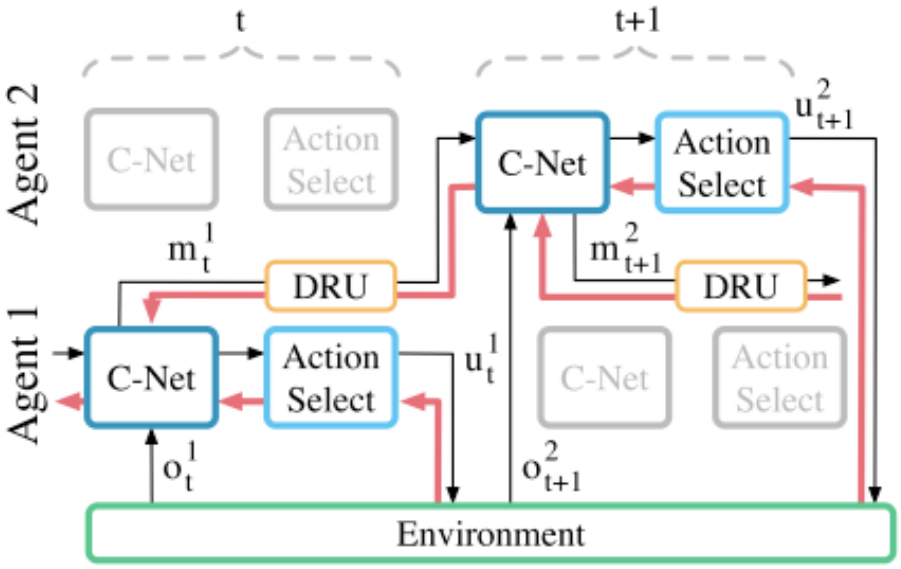
Q-Net is now called C-Net and outputs:

1. The Q value, which is fed to the action selector.
2. A real-valued message.

DIAL

Constraint: the real-valued message cannot be used during execution.

- DIAL introduces a discretise regularise unit (DRU).
- Training: DRU regularises m.
- Execution: DRU discretises m.
- DIAL extends to continuous messages easily.
- DIAL perform better than RIAL.
- Results in the paper.



(b) DIAL - Differentiable communication

Challenges in communication

RIAL and DIAL is the first attempt to learn to communicate with deep RL.

Successors to RIAL and DIAL tackle other challenges with new frameworks.

Challenges and some associated papers

- Multi-stage communication (several messages before taking action).
    - Send and receive messages before taking their action.
        - CommNet
- Adapting to various number of agents:
    - CommNet and BiCNet.
- Targeted communication:
    - IC3Net, TarMac and ATOC.
- Limit the number of messages:
    - SchedNet and GACML.

Competition

Minimax-Q

In a two player competition, gain of one agent is equal loss of the other.

We can define a new reward function $r_t^{a_i} = R^{a_i}(s_{t+1}, s_t, u_t^{a_i}, u_t^{a_{-i}})$.

Goal:

- Agent a_i maximises $r_t^{a_i}$.
- Its opponent a_{-i} minimises $r_t^{a_i}$.

Minimax-Q (Littman 1994) ideas:

$$V^{\pi *}(s) = \max_{\pi} \min_{u^{a_{-i}}} \sum_{u^{a_i} \in \mathcal{U}_i} Q^{\pi *}(s, u^{a_i}, u^{a_{-i}}) \pi(a_i|s)$$

$$Q^{\pi *}(s, u^{a_i}, u^{a_{-i}}) = R^{a_i}(.) + \gamma \sum_{s'} P(s'|s, u^{a_i}, u^{a_{-i}}) V^{\pi *}(s')$$

AlphaGo Zero

- No details as already covered in Model-Based lecture04.
- Nice blog
- Trained with self-play!
  - The trained agent acts as both player during training.
- Reminder:
  - Policy and value networks trained to guide MCTS search.
  - AlphaGo Zero:
    - No supervised learning.
    - No human data.
  - Some numbers:
    - 4.9 million games generated.
    - 1600 games for each MCTS (~ 0.4s).
    - 700, 000 minibatches of 2, 048 states.

Hide and Seek

Hide and Seek

Some details:

- Time limit: 240
- Observation space:
  - Position, velocity and size (or objects), in a 135 degree cone in front of the agent.
  - "LIDAR": 30 range sensors around the agent.
- Team based reward
  - Hiders: +1 if hidden, -1 if seen.
  - Seekers: opposite.
  - 0 if no one is seen by the seekers.
- Preparation phase:
  - Only the hiders can perform action in the beginning.
- Action space:
  - Move: discretized forces along x and y axis and torque around their z axis.
  - Grab or lock the closest object in front of them. 2 binaries.
    - Grab: the object is bound to the agent while boolean is True.
    - Lock: the object is locked and cannot be moved. Unlocking is available if the agent is part of the agent team that has locked the object.

More in the blog!

Hide and Seek

Training:

- PPO and GAE.
- Centralize training, decentralized execution (CTDE).
  - At training: the critic has access to the full state to learn a centralized value function.
  - At execution: the policy network is used normally.
  - No counterfactual baseline.
- Self-play: each agent acts independently and shares the same network parameters, playing against itself.
- 5% chance of using a past policy version to improve robustness.

Architecture in the blog!

Hide and Seek

Evaluation:

- Reward in Multi-Agent is not sufficient to evaluate agents.
- Are agents improving evenly or have they stagnated?
    - ELO score (or Trueskill) allows to establish a ranking between policies.
    - Elo score does not differentiate adaptation and improvement of already learned skills.
- Their propose two evaluations scheme.
    - Comparison to intrinsic motivation:
        - Count based exploration: agent receives reward when visiting states that has not been visited much.
    - Inteligence tests: transfer and fine-tuning.

Goal: Compare behavior learned in Hide and Seek with common unsupervised exploration techniques.

Results in the blog!

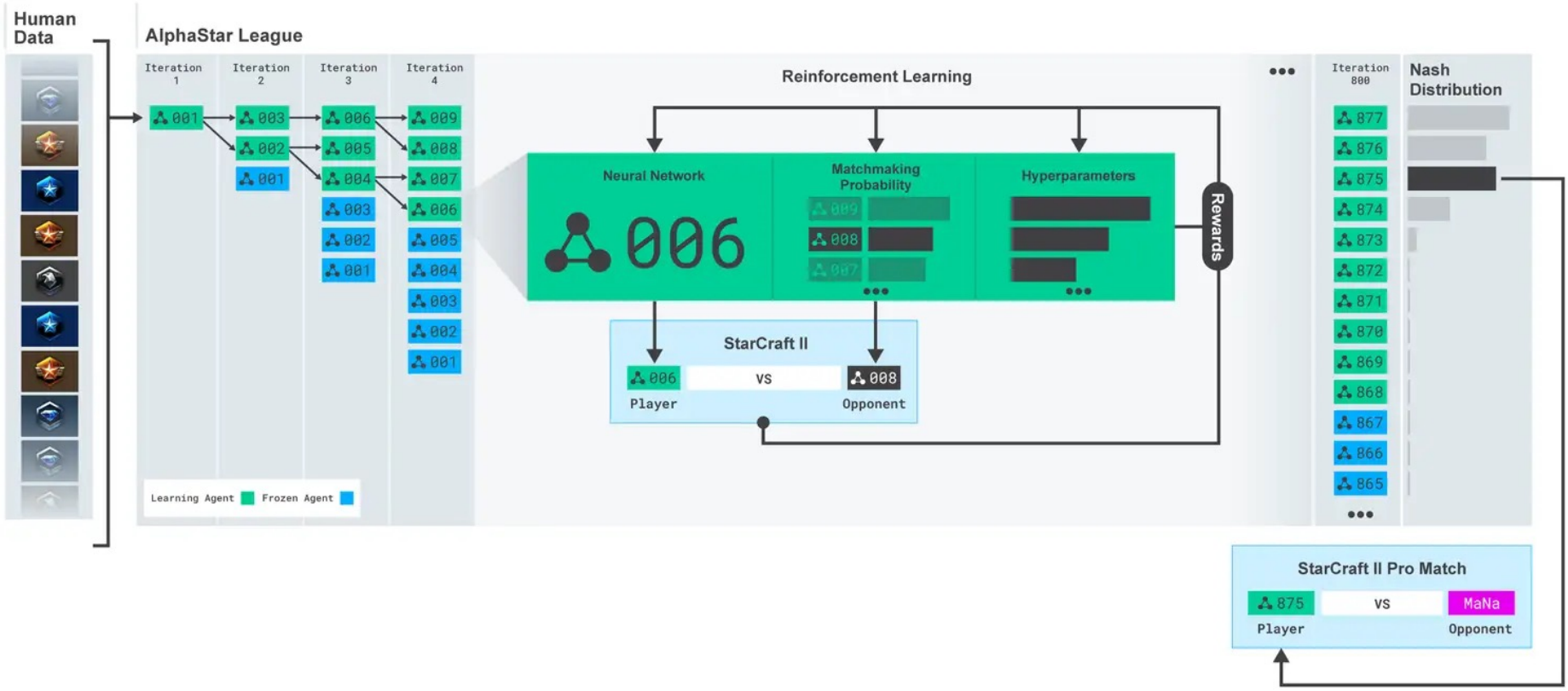
Competitive: Other works

1. Dota 2, OpenAI Five (2019): Dota 2 with large scale deep reinforcement learning.
2. StarCraft 2, AlphaStar (2019): Grandmaster level in StarCraft II using multi-agent reinforcement learning.
3. Quake 3 capture the flag (2019): Human-level performance in 3D multiplayer games with population-based reinforcement learning.
4. Stratego (10^{535} possible states vs 10^{360} possible states in Go): Deep Nash (2022) Mastering the game of Stratego with model-free multiagent reinforcement learning.

AlphaStar

Again, a nice blog!

Population based training.



Two team Markov game

What if we have 2 symmetric teams?

- A set of n agents, each represented by a or $a_{i,j}, i \in 1, ..., n, j \in 1, 2$.
- A set of states $\boldsymbol{s} \in \mathcal{S}$.
- An observation function $O : \mathcal{S} \times 1, ..., n \to \mathcal{Z}$.
- A set of action spaces $\mathcal{U} = \mathcal{U}_1 \times ... \times \mathcal{U}_n$, one per agent $u_t^{a_{i,j}} \in \mathcal{U}_i \forall j$.
- A transition function: $\boldsymbol{s}_{t+1} \sim P(\boldsymbol{s}_{t+1} | \boldsymbol{s}_t, \mathbf{u_t})$, $\mathbf{u_t} = (u_t^{a_{i,j}})_{i \in 1,..,n; j \in 1,2}$.
- A reward function per team: $r_t^j = R^j(\boldsymbol{s}_{t+1}, \boldsymbol{s}_t, \mathbf{u_t})$.
- Agents store their history $\tau_t^a \in (\mathcal{Z} \times \mathcal{U})^t$.
- The goal of each agent a_i is to maximize its total expected sum of (discounted) rewards $\sum_{t=0}^{T} \gamma^t r_t^{a_i}$.

Two team Markov game

How to train a team of agents to compete?

Train against multiple evolving strategies!

- Teams are trained with CTDE methods: QMIX, QVMix and MAVEN.
- Teams are trained with three different learning scenarios:
    1. Against a stationary strategy (heuristic),
    2. Self-play,
    3. Within a population of training teams.
- New Competitive StarCraft Multi-Agent Challenge.
- Teams trained with 10^7 timesteps in 2 environments ($3m$ and $3s5z$).
- Each method/scenario pair has been executed 10 times.
- Evaluation is performed with Elo scores and win rates.

Elo score

The Elo rating system assign each player of a population with a rating R to rank them.

- We can compute the probability that a player will win against B.
- Let R_A and R_B be the ELO scores of player A and B, E_A = proba A wins.

$$E_A = \frac{10^{R_A/400}}{10^{R_A/400} + 10^{R_B/400}} \text{ and } E_B = \frac{10^{R_B/400}}{10^{R_A/400} + 10^{R_B/400}}$$

- 400 is a parameter: if the Elo score of player A is 400 points above that of B, it has a ten-times greater chance of defeating B.
- New score where cst is a constant that defines the maximum possible update of the Elo score (10 in our paper, typically 32).

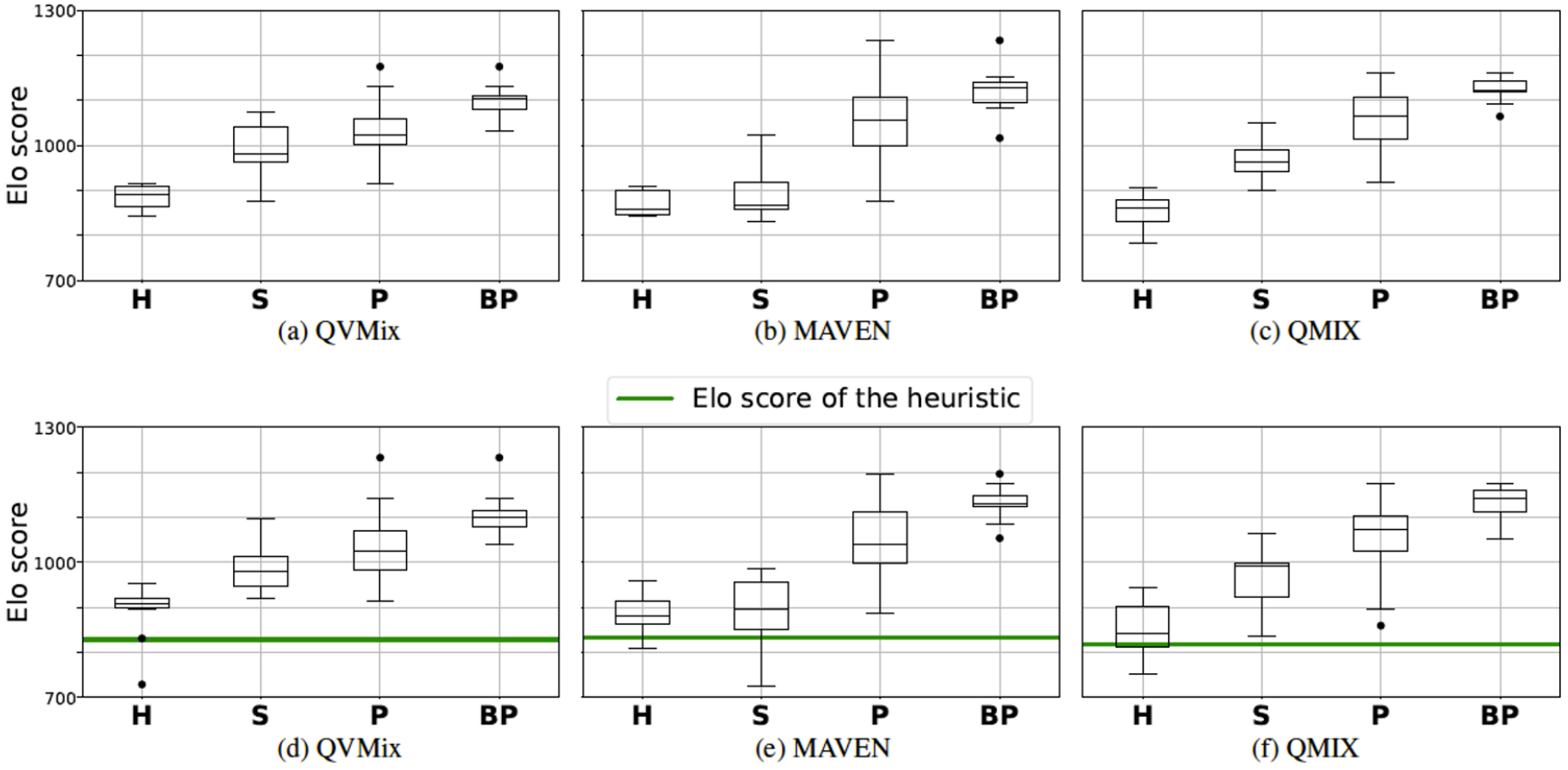
$$R'_A = R_A + cst * (S_A - E_A)$$

- S_A is equal to 1 for a win, 0 for a loss and 0.5 for a draw.

Two team Markov game: results

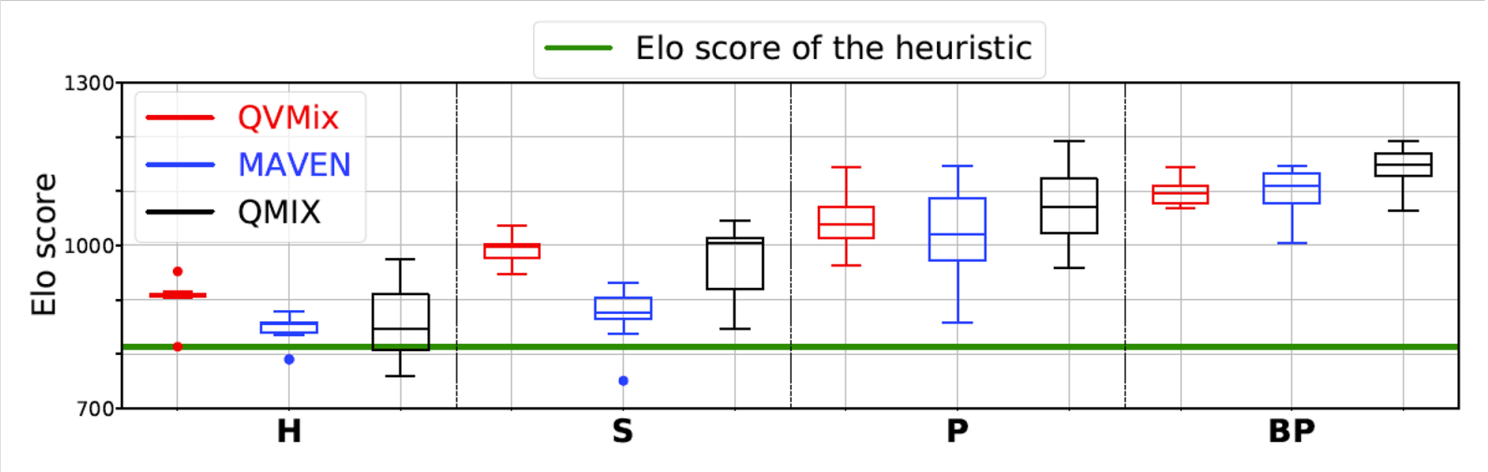
H = trained against heuristic, S = self-play, P = within a population, BP = the 10 bests of each population.

Elo after training (Elo score obtained in different test populations):



Two team Markov game: results

All together:



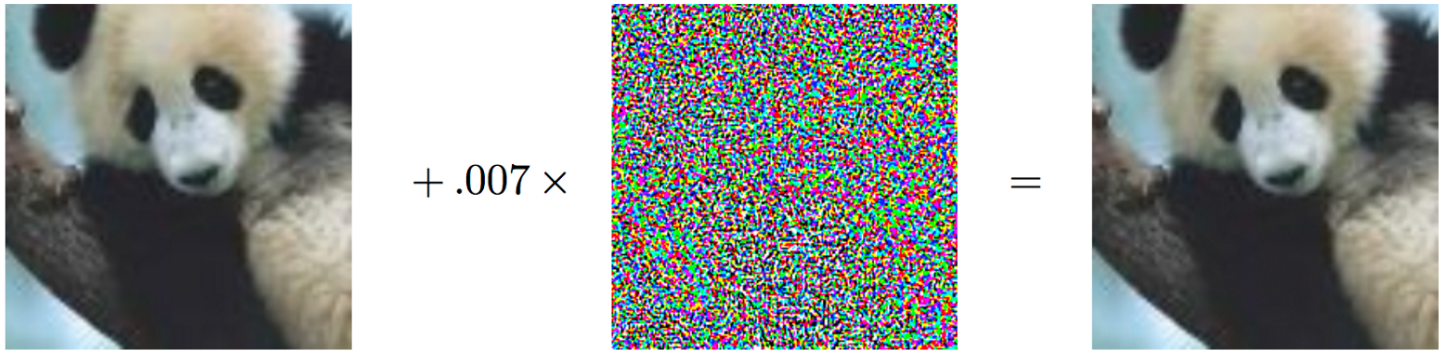
Conclusions:

- Training teams within a population of learning teams is the best learning scenario, when each team plays the same number of timesteps for training purposes.
- This is irrespective of whether or not the stationary strategy was better than all trained teams.
- A selection procedure is required in the same training population.

Adversarial Attack

Adversarial Attack

It is possible to trick a neural network with small perturbations.



"panda"

57.7% confidence

noise

"gibbon"

99.3% confidence

Adversarial Attack

"Adversarial policies: Attacking deep reinforcement learning".

- It is not possible to modify the observation of an other agent.
- But can we attack them with adversarial observation?
- This is the goal of this paper:
- Learn to win by not directly playing the game but by performing adversarial policies.

Framework: Zero sum Markov Game.

Goal: black box attack by learning adversary policies.

adversarialpolicies.github.io

Adversarial Attack

Adversarial attacks

How to attack a trained agent?

- Victims are trained with self-play.
- In the paper, they take pre-trained networks between 680 and 1360 millions timesteps.
- By fixing the victim policy π_ν, an adversarial policy π_α is trained.
- Note that the agent is now in a MDP since π_ν is now stationary.
- Adversarial policy is trained for 20 millions timesteps.

Attacks are validated by masking the position of the attacker from the victim which now wins:

- The attacker did not learn a strong policy.

Fine-tuning against the attacker allows to defend against these attacks.

References

- Bible: Richard S. Sutton and Andrew G. Barto. (2018) Reinforcement Learning: An Introduction.
- DQN: Mnih, V., Kavukcuoglu, K., Silver, D. et al. (2015). Human-level control through deep reinforcement learning.
- DRQN: Hausknecht, M., Stone, P. (2015). Deep recurrent q-learning for partially observable mdps.
- Actor-critic: Konda, V., & Tsitsiklis, J. (1999). Actor-critic algorithms.
- Markov Game and Minimax Q: Littman, M. L. (1994). Markov games as a framework for multi-agent reinforcement learning.
- Dec-POMDP: Oliehoek, F. A., Amato, C. (2016). A concise introduction to decentralized POMDPs.
- IQL: Tan, M. (1993). Multi-agent reinforcement learning: Independent vs. cooperative agents.

- IGM and QTRAN: Son, K., Kim, D., Kang, W.J., Hostallero, D.E. , Yi, Y.. (2019). QTRAN: Learning to Factorize with Transformation for Cooperative Multi-Agent Reinforcement Learning.
- VDN: Sunehag, P., Lever, G., Gruslys, A., Czarnecki, W. M., Zambaldi, V., Jaderberg, M., ... Graepel, T. (2017). Value-decomposition networks for cooperative multi-agent learning.
- QMIX: Samvelyan, M., Rashid, T., De Witt, C. S., Farquhar, G., Nardelli, N., Rudner, T. G., ... Whiteson, S. (2019). The starcraft multi-agent challenge.
- COMA: Foerster, J., Farquhar, G., Afouras, T., Nardelli, N., Whiteson, S. (2018, April). Counterfactual multi-agent policy gradients.
- DQV: Sabatelli, M., Louppe, G., Geurts, P., & Wiering, M. A. (2020, July). The deep quality-value family of deep reinforcement learning algorithms.
- QVMix: Leroy, P., Ernst, D., Geurts, P., Louppe, G., Pisane, J., Sabatelli, M. (2020). QVMix and QVMix-Max: Extending the Deep Quality-Value Family of Algorithms to Cooperative Multi-Agent Reinforcement Learning.

- MAVEN: Mahajan, A., Rashid, T., Samvelyan, M., & Whiteson, S. (2019). Maven: Multi-agent variational exploration.
- QPLEX: Wang, J., Ren, Z., Liu, T., Yu, Y., & Zhang, C. (2020). Qplex: Duplex dueling multi-agent q-learning.
- LIIR: Du, Y., Han, L., Fang, M., Liu, J., Dai, T., & Tao, D. (2019). Liir: Learning individual intrinsic reward in multi-agent reinforcement learning.
- MADDPG: Lowe, R., Wu, Y. I., Tamar, A., Harb, J., Pieter Abbeel, O., & Mordatch, I. (2017). Multi-agent actor-critic for mixed cooperative-competitive environments.
- MAAC: Iqbal, S., & Sha, F. (2019). Actor-attention-critic for multi-agent reinforcement learning.
- FACMAC: Peng, B., Rashid, T., Schroeder de Witt, C., Kamienny, P. A., Torr, P., Böhmer, W., & Whiteson, S. (2021). Facmac: Factored multi-agent centralised policy gradients.
- HATRPO-HAPPO: Kuba, J. G., Chen, R., Wen, M., Wen, Y., Sun, F., Wang, J., & Yang, Y. (2021). Trust region policy optimisation in multi-agent reinforcement learning.

- RIAL/DIAL: Foerster, J. N., Assael, Y. M., De Freitas, N., Whiteson, S. (2016). Learning to communicate with deep multi-agent reinforcement learning.
- CommNet: Sukhbaatar, Sainbayar, Arthur Szlam, and Rob Fergus (2016). Learning multiagent communication with backpropagation.
- BiCNet: Peng, Peng et al. (2017). Multiagent Bidirectionally-Coordinated Nets: Emergence of Human-level Coordination in Learning to Play StarCraft Combat Games.
- IC3Net: Singh, Amanpreet, Tushar Jain, and Sainbayar Sukhbaatar (2019). Learning when to communicate at scale in multiagent cooperative and competitive tasks.
- TarMac: Das, Abhishek et al. (2019). TarMAC: Targeted multi-agent communication.
- ATOC: Jiang, Jiechuan and Zongqing Lu (2018). Learning attentional communication for multi-agent cooperation.
- SchedNet: Kim, Daewoo et al. (2019). Learning to schedule communication in multi-agent reinforcement learning.
- GACML: Mao, Hangyu et al. (2019). Learning Agent Communication under Limited Bandwidth by Message Pruning.

- AlphaGo: Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., ... Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search.
- AlphaGo Zero: Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., ... Hassabis, D. (2017). Mastering the game of go without human knowledge.
- OpenAI Five: Berner, C., Brockman, G., Chan, B., Cheung, V., ... Zhang, S. (2019). Dota 2 with large scale deep reinforcement learning.
- AlphaStar: Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., ... Silver, D. (2019). Grandmaster level in StarCraft II using multi-agent reinforcement learning.
- Capture the Flag: Jaderberg, M., Czarnecki, W. M., Dunning, I., Marris, L., Lever, G., Castaneda, A. G., ... Graepel, T. (2019). Human-level performance in 3D multiplayer games with population-based reinforcement learning.

- Hide and Seek: Baker, B., Kanitscheider, I., Markov, T., Wu, Y., Powell, G., McGrew, B., Mordatch, I. (2019). Emergent tool use from multi-agent autocurricula.
- Stratego: Perolat, J., De Vylder, B., Hennes, D., Tarassov, E., Strub, F., de Boer, V.,..., Tuyls, K. (2022). Mastering the game of Stratego with model-free multiagent reinforcement learning. Science.
- Two-team Markov Game: Leroy, P., Pisane, J., Ernst, D. (2022). Value-based CTDE Methods in Symmetric Two-team Markov Game: from Cooperation to Team Competition. In Deep Reinforcement Learning Workshop, NeurIPS.
- Adversarial policies: Gleave, A., Dennis, M., Wild, C., Kant, N., Levine, S., Russell, S. (2019). Adversarial policies: Attacking deep reinforcement learning.

Thanks!