INFO8003-1
Optimal decision making for complex problems

Multi-Agent Reinforcement Learning

Pascal Leroy

April 2022

Content

Today, an overview of multi-agent reinforcement learning (MARL):

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

RL basics

RL basics: MDP

Single-agent reinforcement learning (SARL)

Markov decision process (MDP)

Defined by:

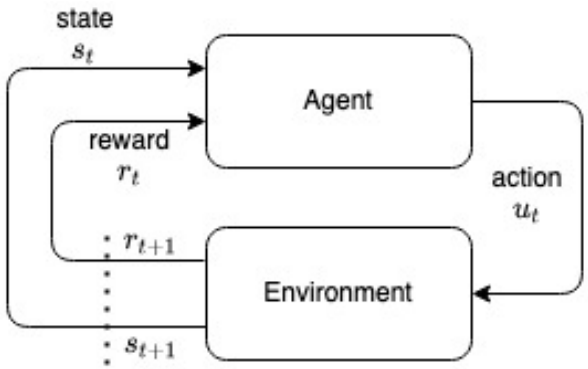


Figure: RL environment.

- 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 maximize its total expected sum of (discounted) rewards $\sum_{t=0}^{T} \gamma^t r_t$ with $\gamma \in [0, 1)$, obtained with the optimal policy π^*.

Richard S. Sutton and Andrew G. Barto. 2018. Reinforcement Learning: An Introduction. A Bradford Book, Cambridge, MA, USA.

RL basics: Value-based methods

Value-based methods:

- State Value of a policy π:

$$V^{\pi}(s_t) = \mathbb{E}_{\pi}\left[r_t + \gamma V^{\pi}(s_{t+1})|s_t\right]$$

- State-Action Value of a policy π:

$$Q^{\pi}(s_t, u_t) = \mathbb{E}_{\pi}\left[r_t + \gamma Q^{\pi}(s_{t+1}, u_{t+1})|s_t, u_t\right]$$

- The optimal policy is:

$$\pi^*(s_t) = \underset{u}{\text{argmax}}\, Q^{\pi^*}(s_t, u)$$

RL basics: DQN

DQN: Q-learning with a neural network parametrised by θ:

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

- 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: Policy-based methods

We denote the discounted sum of reward of a trajectory by

$$G_t(\tau) = \sum_{j=0}^{T} \gamma^j r_{t+j} \qquad \text{where } \tau = (s_t, u_t, r_t, s_{t+1}, u_{t+1}, .., s_T)$$

Reinforce:

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

RL basics: How to improve reinforce?

Smaller variance with a baseline b:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\left(\sum_t^T \left(G_t(\tau) - b \right) \nabla_\theta \log \pi_\theta(a_t, s_t) \right) \right]$$

Or Q Actor-Critic:

$$Q(s_t, u_t,) = \mathbb{E}_{(r_t, s_{t+1}, \ldots, s_T)} \left[G_t(\tau) \right]$$

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

RL basics: Advantage 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 $Q(s_t, a_t) - V(s_t)$:

$$A(s_t, u_t; \phi) = r_t + \gamma V(s_{t+1}; \phi) - V(s_t; \phi)$$

RL basics: Recap

- Markov Decision Process
- Value-based methods
  - DQN
  - DRQN
- Policy-based methods
  - Reinforce + improvements
  - Advantage Actoc-Critic

Multi-Agent Reinforcement Learning

MARL: Framework

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: $s_{t+1} \sim P(s_{t+1}|s_t, \boldsymbol{u_t})$ with $\boldsymbol{u_t} = \bigcup_{i \in \{1,..,n\}} u_t^{a_i}$.
- A reward function per agent: $r_t^{a_i} = R^{a_i}(s_{t+1}, s_t, \boldsymbol{u_t})$, sometimes shorten $r(s_t, \boldsymbol{u_t})$.
- 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 $\sum_{t=0}^{T} \gamma^t r_t^{a_i}$.

MARL: Different settings

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 is equivalent 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

Cooperative: environment example

StarCraft multi-agent challenge (SMAC).
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.

Figure: SMAC example (3s5z).

`https://github.com/oxwhirl/smac`
`https://youtu.be/VZ7zmQ_obZ0`

Samvelyan, M., Rashid, T., De Witt, C. S., Farquhar, G., Nardelli, N., Rudner, T. G., ... Whiteson, S. (2019). The starcraft multi-agent challenge.

Cooperative: Dec-POMDP

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

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

Such Markov Games are called Decentralised-POMDP.

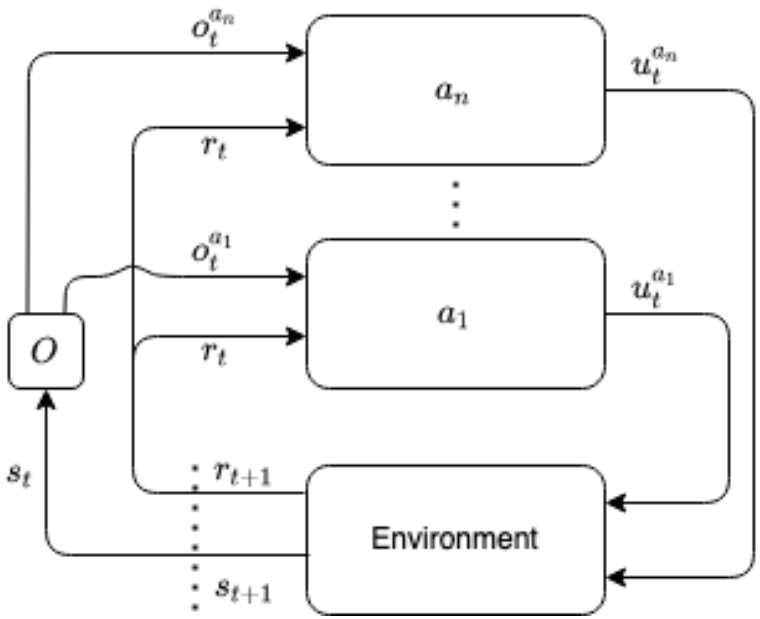


Figure: Dec-POMDP.

Cooperative: Centralised controller

Centralised controller:

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

Problems?

Cooperative: Centralised controller

Centralised controller:

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

Problems?

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

Solutions?

Cooperative: Centralised controller

Centralised controller:

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

Problems?

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

Solutions?

1. Decentralised controller.
   $\rightarrow$ Naive learner: train each agent with SARL algorithms.
2. Centralised training with decentralised execution (CTDE).
   $\rightarrow$ Benefit from supplementary information during training, such as the entire state of the game.

Cooperative: 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:

Cooperative: 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?

Cooperative: Value-based methods in CTDE

Naive learner: Independent Q-Learning (IQL)
$\rightarrow$ Each agent learns its individual $Q_a(\tau_t^a, u_t^a)$ independently.

Problem: How to ensure that $\arg\max_{u_t^a} Q_a(\tau_t^a, u_t^a)$ maximises $Q(s_t, \boldsymbol{u_t})$?

Cooperative: Value-based methods in CTDE

Naive learner: Independent Q-Learning (IQL)
$\rightarrow$ Each agent learns its individual $Q_a(\tau_t^a, u_t^a)$ independently.

Problem: How to ensure that $\arg\max_{u_t^a} Q_a(\tau_t^a, u_t^a)$ maximises $Q(s_t, \boldsymbol{u_t})$?

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

Q_a are not anymore Q function but utility function used to select actions.

Condition: Individual Global Max (IGM):

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

Cooperative: VDN and QMIX

How to satisfy IGM?
Value Decomposition Network:

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

Problems:

- Addition does not allow to build complex functions.
- Current state information s_t is not considered.

Cooperative: QMIX

How can we build non-linear factorisation satisfying IGM?

QMIX idea is to enforce monotonicity:

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

How to build a non-linear monotonic factorisation of $Q(s_t, \boldsymbol{u_t})$ 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. This is a neural network that computes the weights of a second neural network.

Cooperative: QMIX

In QMIX:

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

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

The optimisation procedure follows the same principles of DQN algorithm and is applied to $Q_{mix}(s_t, \boldsymbol{u_t})$.
Parameters of individual network are shared to speed up learning.

Cooperative: QMIX architecture

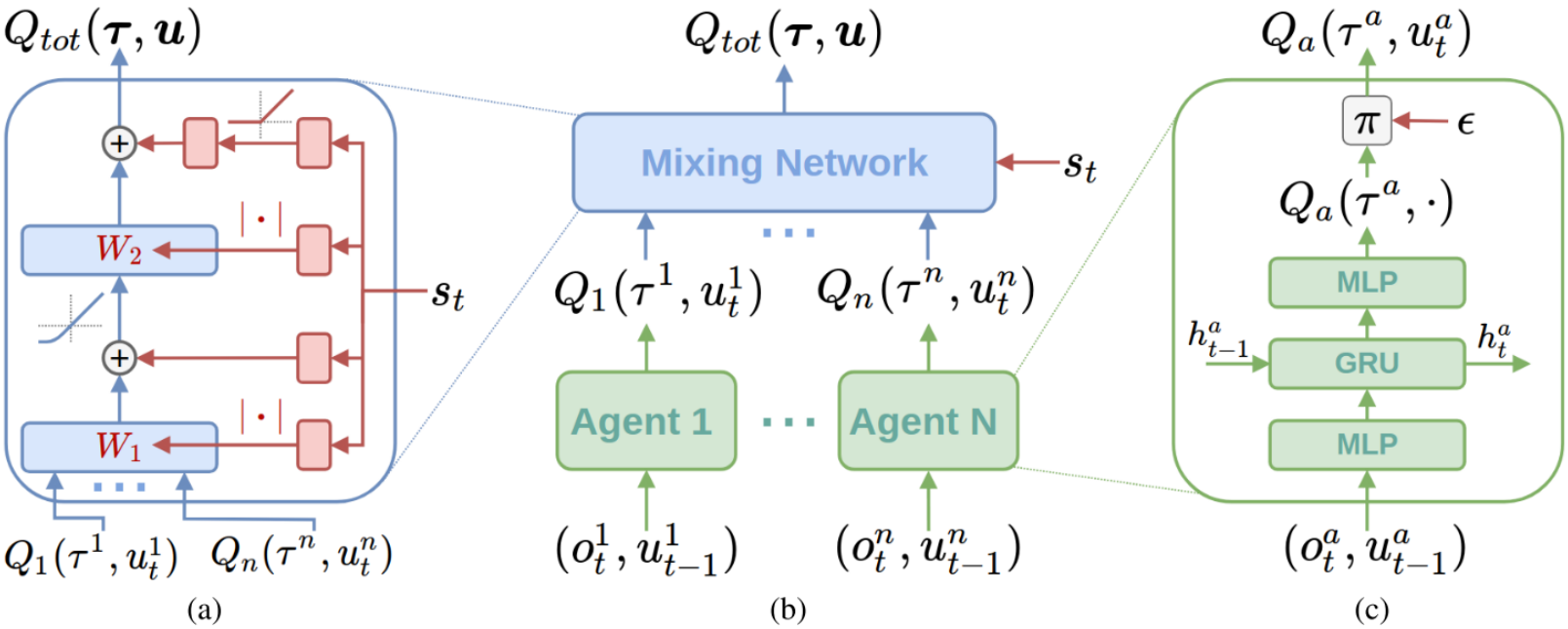


Figure 2. (a) Mixing network structure. In red are the hypernetworks that produce the weights and biases for mixing network layers shown in blue. (b) The overall QMIX architecture. (c) Agent network structure. Best viewed in colour.

Figure: QMIX architecture.

Rashid, T., Samvelyan, M., Schroeder, C., Farquhar, G., Foerster, J., Whiteson, S. (2018). Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning.

Cooperative: QMIX results

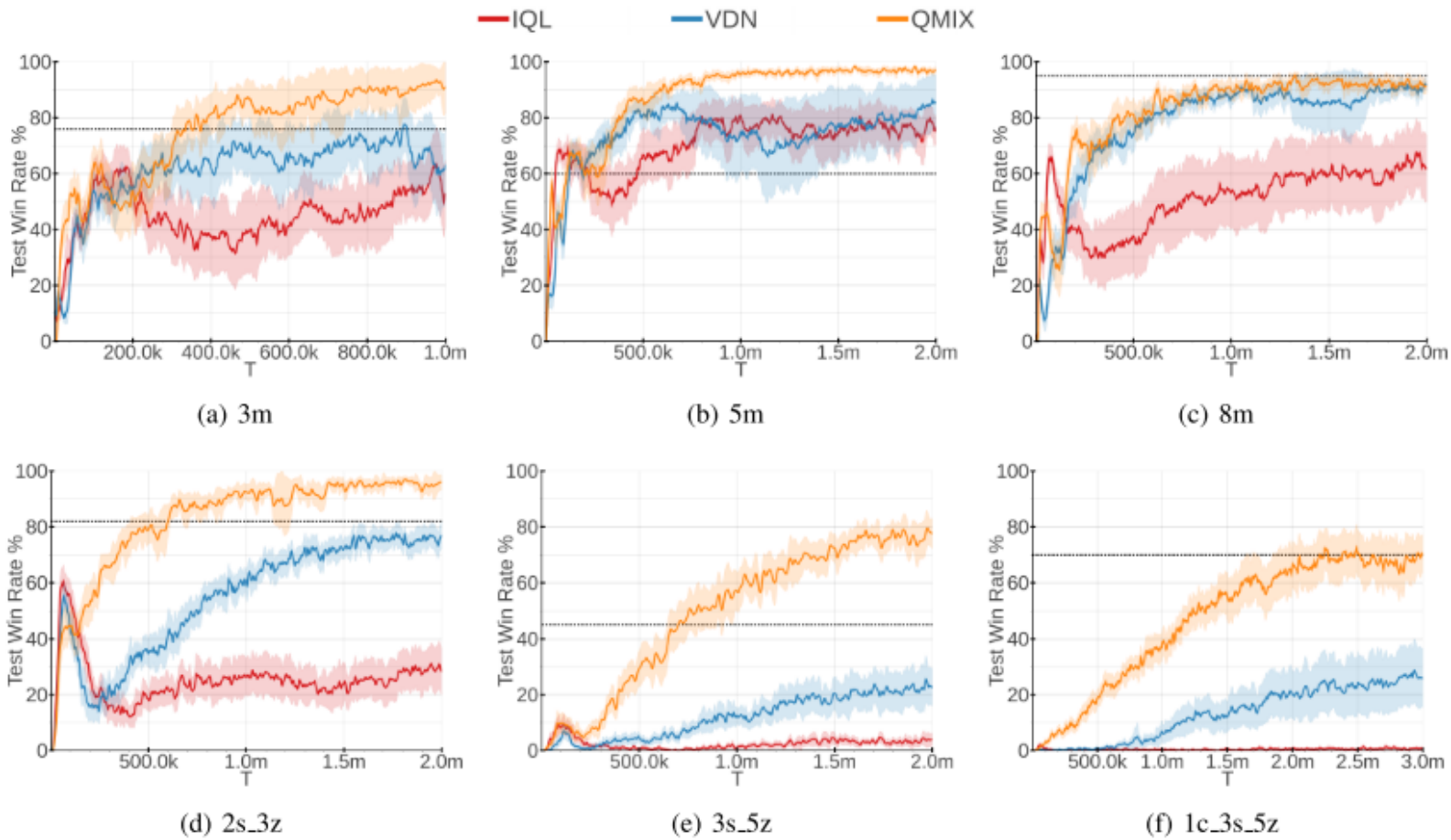


Figure 3. Win rates for IQL, VDN, and QMIX on six different combat maps. The performance of the heuristic-based algorithm is shown as a dashed line.

Rashid, T., Samvelyan, M., Schroeder, C., Farquhar, G., Foerster, J., Whiteson, S. (2018). Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning.

Cooperative: Policy-based methods in CTDE

Naive learner: Independent Actor-Critic (IAC)
$\rightarrow$ 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 s_t?

Foerster, J., Farquhar, G., Afouras, T., Nardelli, N., Whiteson, S. (2018, April). Counterfactual multi-agent policy gradients.

Cooperative: COMA

Problem: How to benefit from centralised information such as s_t?

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

- Critic is only used during training.
- $\rightarrow$ Centralised critic that computes the advantage based on s_t.

$$A(s, u_t^a; \phi) = r + \gamma V(s_{t+1}; \phi) - V(s_t; \phi)$$

Problems:

- Based on global rewards r_t.
- The centralised critic does not solve the credit assignment problem.

Cooperative: COMA

Solution: use a counterfactual baseline.

- Inspired from difference reward:

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

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

- Any action u^a that maximises $r(s, \boldsymbol{u})$ also maximises D^a.

- Problems:
  1. A simulator is required to obtain $r(s, (\boldsymbol{u}^{-a}, c^a))$, but these can be approximated.
  2. One needs to decide which action is c^a.

Cooperative: COMA

Foerster, et. al. (2018) overcomes these problems with a centralised critic that computes difference rewards by learning $Q(s, \boldsymbol{u})$.

For each agent a, the advantage is:

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

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.
- COMA solution: the critic takes as input u^{-a} and computes only $|\mathcal{U}_a|$ outputs.

Note that this method is only possible for discrete action spaces, while it is possible to evaluate $\sum_{u'^a} \pi^a(u'^a|\tau^a) Q(s, (\boldsymbol{u}^{-\boldsymbol{a}}, u'^a))$ with Monte Carlo or Gaussian policies in continuous action spaces.

Cooperative: COMA architecture

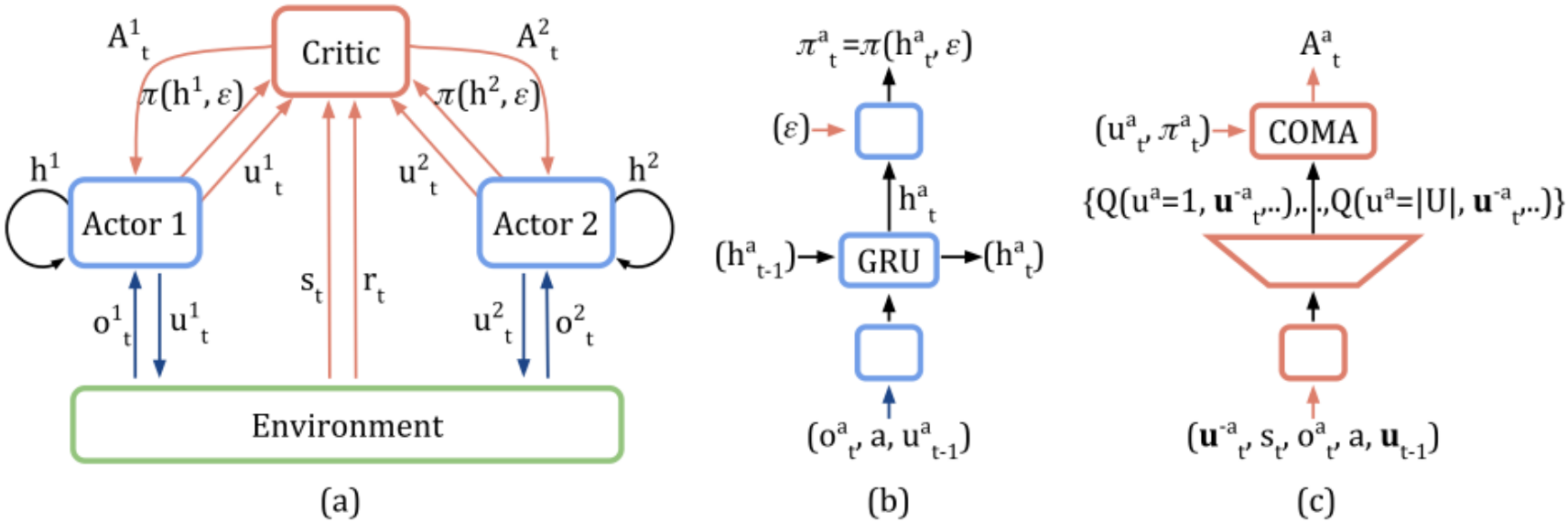


Figure 1: In (a), information flow between the decentralised actors, the environment and the centralised critic in COMA; red arrows and components are only required during centralised learning. In (b) and (c), architectures of the actor and critic.

Figure: COMA architecture.

Foerster, J., Farquhar, G., Afouras, T., Nardelli, N., Whiteson, S. (2018, April). Counterfactual multi-agent policy gradients.

Cooperative: COMA results

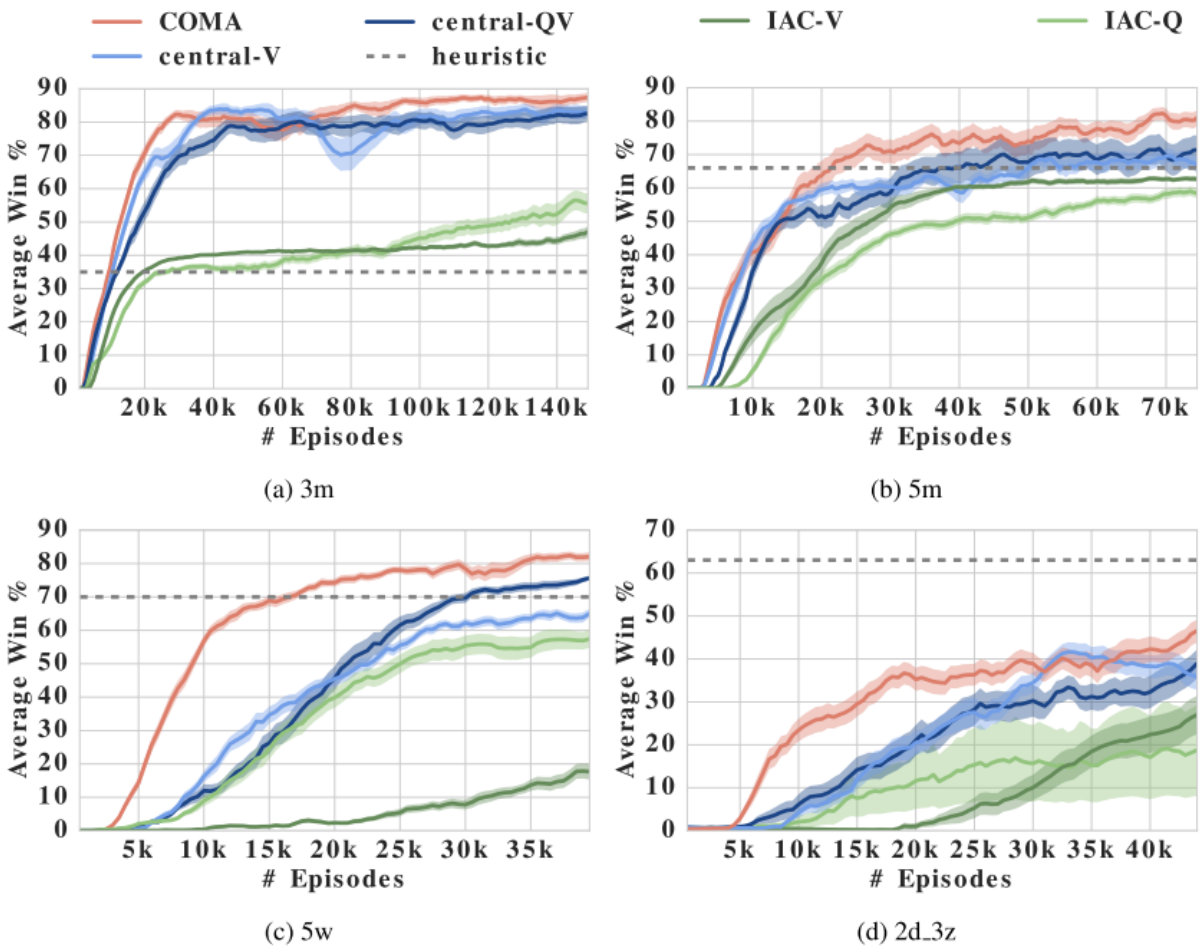


Figure 3: Win rates for COMA and competing algorithms on four different scenarios. COMA outperforms all baseline methods. Centralised critics also clearly outperform their decentralised counterparts. The legend at the top applies across all plots.

Foerster, J., Farquhar, G., Afouras, T., Nardelli, N., Whiteson, S. (2018, April). Counterfactual multi-agent policy gradients.

Cooperative: COMA vs QMIX

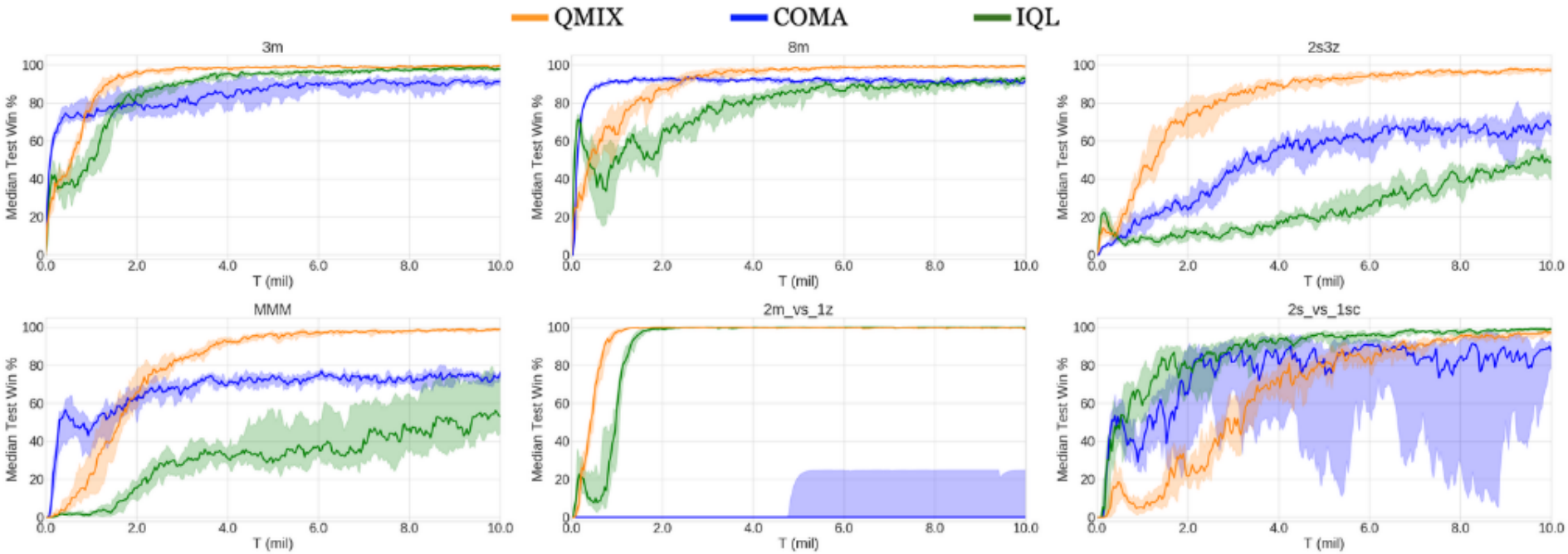


Figure: Median win rates for QMIX, COMA, and IQL on easy SMAC scenarios.

Samvelyan, M., Rashid, T., De Witt, C. S., Farquhar, G., Nardelli, N., Rudner, T. G., ... Whiteson, S. (2019). The starcraft multi-agent challenge.

Cooperative: QVMix

QVMix is an extension of the Deep-Quality value family of algorithms to multi-agent.

Principles of DQV: learn $Q(.;\theta)$ and $V(.;\phi)$ at the same time.

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

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

Overcome the overestimation problem of Q-Learning.

In QVMix, the architecture of V is the same as Q in QMIX, except that there is a single output since actions are ignored.

Cooperative: QVMix results

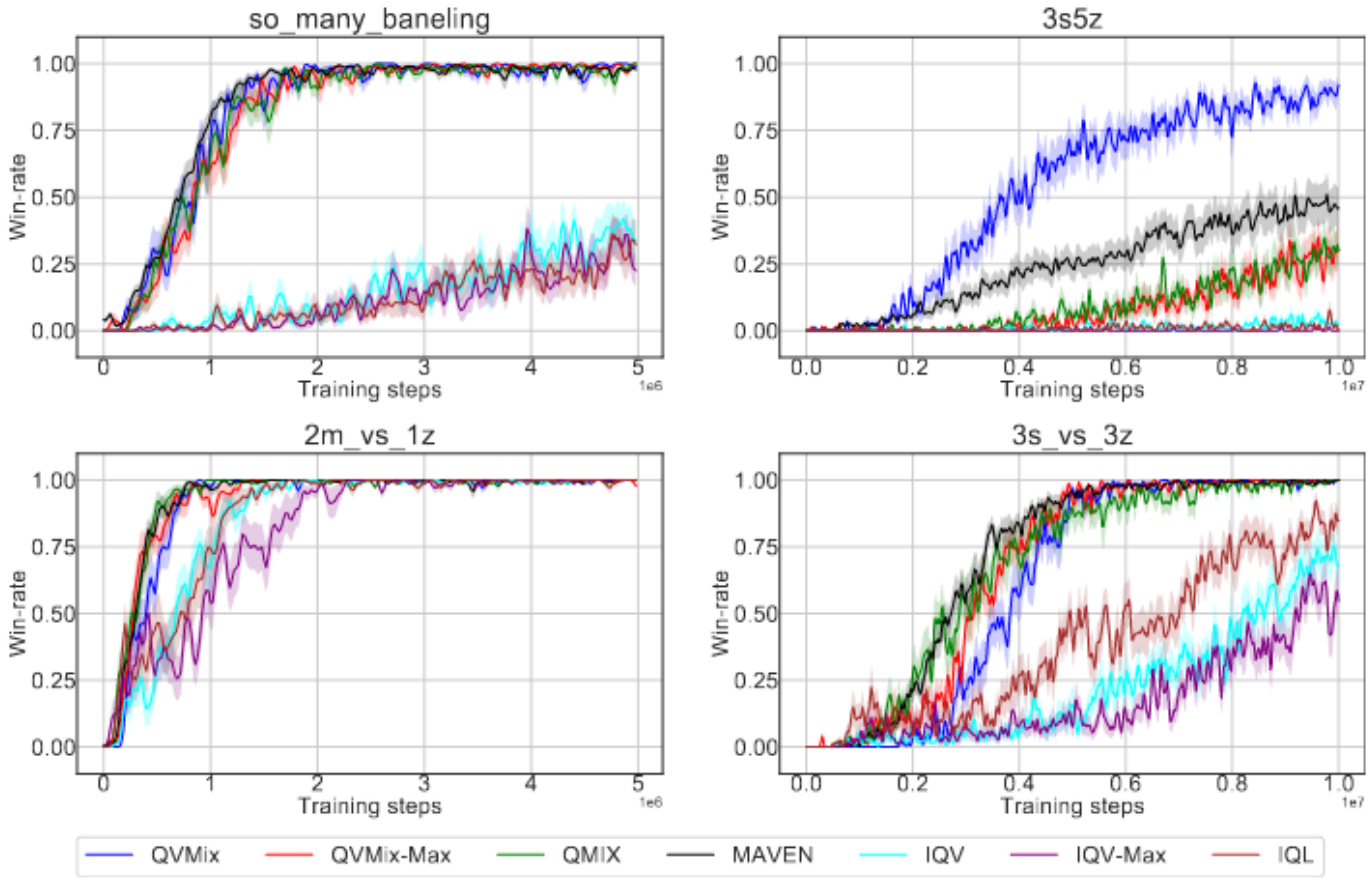


Figure 3: Means of win-rates achieved by QVMix, QVMix-Max, QMIX, MAVEN, IQV, IQVMax and IQL in eight scenarios. Top to bottom, left to right, the scenarios are `3m`, `8m`, `so_many_baneling`, `2m_vs_1z`, `MMM`, `2s3z`, `3s5z` and `3s_vs_3z`. The error band is proportional to the variance of win-rates.

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

Cooperative: QVMix overestimation bias

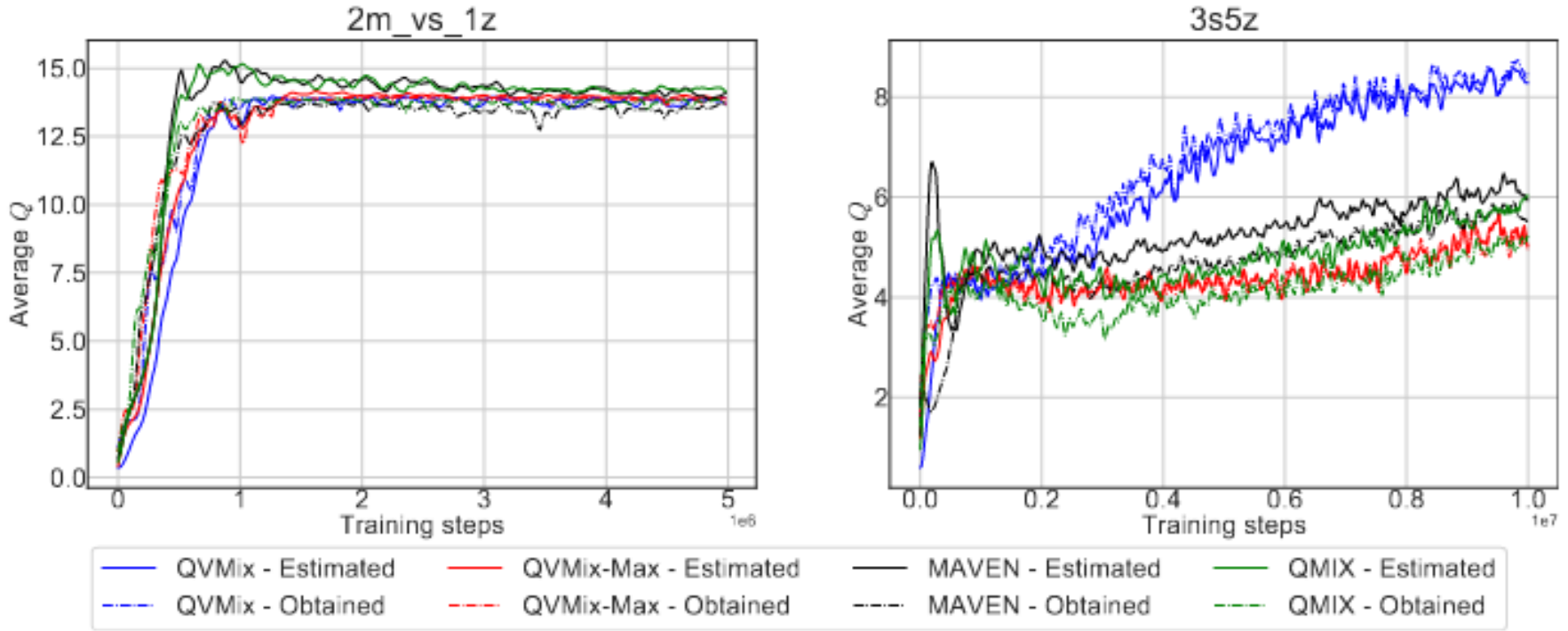


Figure 4: Q values obtained and estimed when training QVMix, QVMix-Max, MAVEN and QMIX. Dash-dotted lines represent the obtained Q values while solid lines represent the estimated ones.

Figure: QVMix overestimation comparison.

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

Cooperative: 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

Cooperative: Recap

- Dec-POMDP
- Value-based methods
  - IQL: each agent learns without considering other agents.
  - QMIX: factorise the $Q(s, \boldsymbol{u})$ as a function of Q^a and monotonicity.
  - QVMIX: learn both Q and V to not overestimate $Q(s, \boldsymbol{u})$.
- 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.

Communication

Communication

How evolves Markov Game with communication?
Communication can be part of the feedback loop: agents take action and send message based on local observations and messages received.

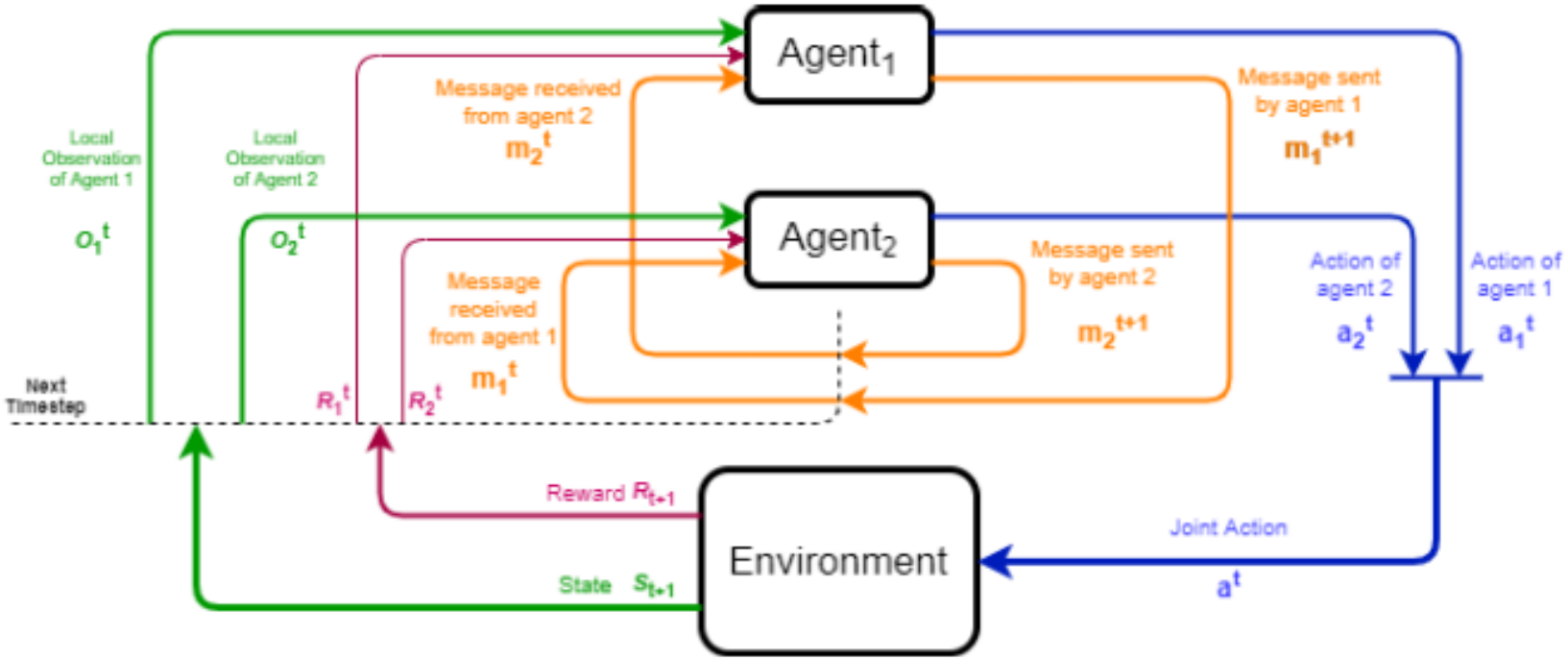


Figure: Communication framework (Fombellida, A. (2020) Battlefield Coordination using Multi-Agent Reinforcement Learning)

Communication: RIAL

The two first methods: RIAL and DIAL ("Learning to communicate with deep multi-agent reinforcement learning").

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

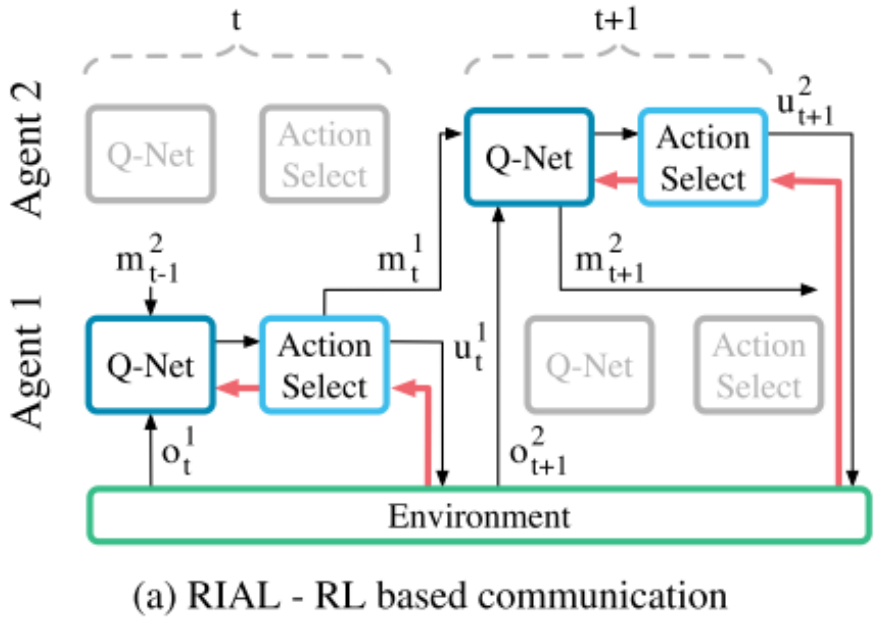
Reinforced inter-agent learning (RIAL) is a first approach:

- Each agent learns $Q^a(o_t^a, m_{t-1}^{-a}, u_t, h_{t-1})$ (called Q-Net).
- 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.

Communication: RIAL

Implementation tricks:

- No more replay buffer because of non-stationarity.
- Previous action and message are inputs at next timestep.
- Parameter sharing to speed up training.
- Gradient path in red.



(a) RIAL - RL based communication

Figure: RIAL architecture.

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

Foerster, J. N., Assael, Y. M., De Freitas, N., Whiteson, S. (2016). Learning to communicate with deep multi-agent reinforcement learning.

Communication: DIAL

Second approach: differentiable inter-agent learning (DIAL) address the feedback limitation by integrating the gradient through the communication channel.

In RIAL, each agent is trained separately while with DIAL, CTDE is now exploited since training is performed across agents.

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

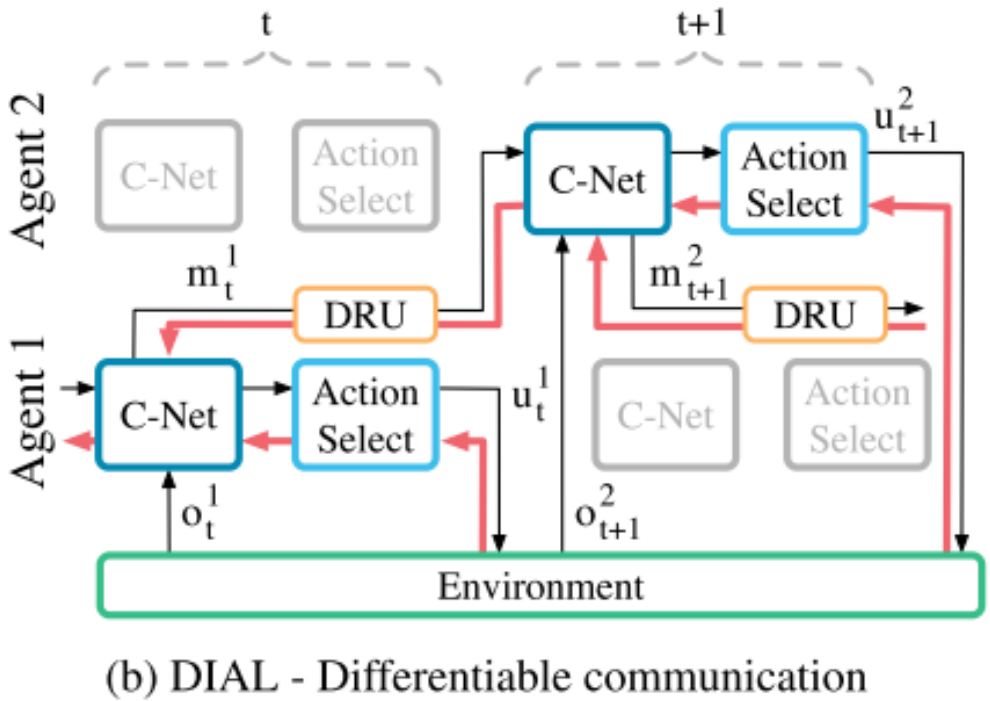
1. The Q value, which is fed to the action selector.
2. A real-valued message.

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

- DIAL introduces a discretise/regularise unit (DRU).
- At training, the DRU regularises messages.
- At execution, the DRU discretises messages.

Communication: DIAL

- DIAL extends to continuous messages easily.
- DIAL perform better than RIAL.
- Results in the paper.
- This is the first attempt to learn to communicate with deep RL.



(b) DIAL - Differentiable communication

Figure: DIAL architecture

Foerster, J. N., Assael, Y. M., De Freitas, N., Whiteson, S. (2016). Learning to communicate with deep multi-agent reinforcement learning.

Other communication challenges

Successor methods to DIAL and RIAL tackle other challenges with a new framework.

Agents send and receive messages before taking their action, allowing multi-stage communication (several messages before taking action).

Challenges and some associated papers:

- Adapting to various number of agents:
  CommNet and BiCNet.
- Targeted communication:
  IC3Net, TarMac and ATOC.
- Limit the number of messages:
  SchedNet and GACML.

Competitive setting

Competitive

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

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

The goal of agent a_i is to maximise this reward while for its opponent a_{-i}, the goal is to minimise it.

Minimax-Q (Littman 1994) ideas:

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

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

Competitive: Minimax-Q

Littman has shown that is possible to learn these Q and V:

Initialize:

For all s in S, a in A, and o in O,
  Let Q[s,a,o] := 1
For all s in S,
  Let V[s] := 1
For all s in S, a in A,
  Let pi[s,a] := 1/|A|
Let alpha := 1.0

Choose an action:

With probability explor, return an action uniformly at random.
Otherwise, if current state is s,
  Return action a with probability pi[s,a].

Learn:

After receiving reward rew for moving from state s to s'
    via action a and opponent's action o,
  Let Q[s,a,o] := (1-alpha) * Q[s,a,o] + alpha * (rew + gamma * V[s'])
  Use linear programming to find pi[s,.] such that:
    pi[s,.] := argmax{pi'[s,.], min{o', sum{a', pi[s,a'] * Q[s,a',o']}}}
  Let V[s] := min{o', sum{a', pi[s,a'] * Q[s,a',o']}}
  Let alpha := alpha * decay

Figure 1: The minimax-Q algorithm.

Littman, M. L. (1994). Markov games as a framework for multi-agent reinforcement learning

Competitive: AlphaGo

From AlphaGo to AlphaGo Zero: an overview of how DeepMind mastered the game of Go.

Reminder of AlphaGo:

- Two networks are trained:
  1. A policy network that predicts best moves (trained in supervised learning with expert moves and improved with reinforcement learning).
  2. A value network that predicts the winner of the game.
- These two are combined with MCTS to provide a lookahead search and led to the first version of AlphaGo.
- "Mastering the game of Go with deep neural networks and tree search".

AlphaGo zero blog: `https://deepmind.com/blog/article/alphago-zero-starting-scratch`

Competitive: AlphaGo Zero

"Mastering the game of Go without human knowledge".

Main differences with AlphaGo:

1. No supervised learning and no human data.
2. A single neural network is trained.
3. Input features number is reduced (see paper for details).
4. Simpler tree search that relies only on the trained network.

Competitive: AlphaGo Zero

The neural network f_θ:

- Input s: raw game board actual position and 7 previous positions.
- Outputs $(\boldsymbol{p}, v)$: a vector of probabilities and a value:
  - The probability of selecting each move.
  - An estimation of the probability to win from s.
- Architecture details in the paper.

The neural network $f_\theta(s) = (\boldsymbol{p}, v)$ is trained in self-play: the agent plays lot of games against itself.

Competitive: AlphaGo Zero

To select action from a state s, AlphaGo Zero uses $(\boldsymbol{p}, v)$ to perform MCTS search to obtain the best moves:

1. Perform a MCTS search to obtain probabilities π of selecting moves.
2. They showed that π provides better actions than $\boldsymbol{p}$.
3. Game ends with a winner z that provides a sample of value v.
4. The network is trained so that $(\boldsymbol{p}, v)$ gets closer to (π, z) to improve MCTS search.

Note that the notation has changed: an action is represented by a.

Competitive: AlphaGo Zero self-play

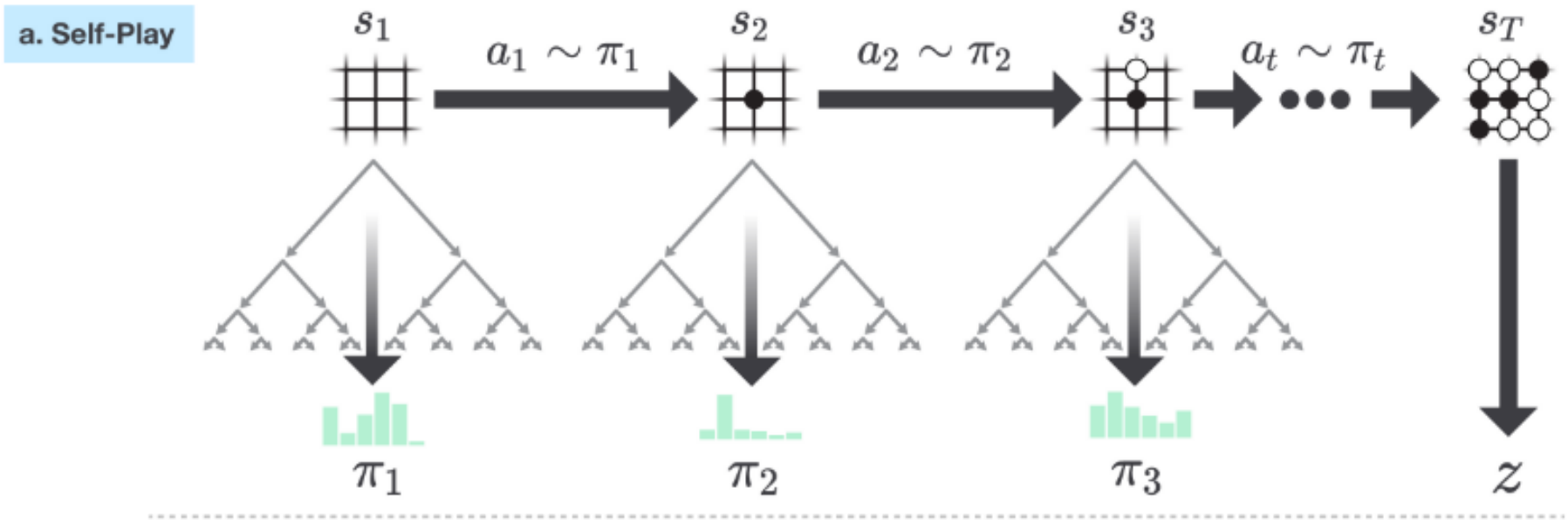


Figure: How an episode is conducted in AlphaGo Zero.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., ... Hassabis, D. (2017). Mastering the game of go without human knowledge.

Competitive: AlphaGo Zero MCTS

How $f_\theta(s)$ guides the MCTS search in AlphaGo Zero to select an action?

- Each edge (s, a) stores:
  1. A prior probability $P(s, a)$.
  2. A visit count $N(s, a)$.
  3. An action-value $Q(s, a)$.
- From the root, iteratively select move that maximises:

$$Q(s, a) + U(s, a)\,;\; U(s, a) \propto \frac{P(s, a)}{1 + N(s, a)}$$

- When a leaf s' is reached, the network generates $f_\theta(s') = (P(s', .), V(s'))$.
- Each traversed edge is updated:
  1. $N(s, a) = N(s, a) + 1$.
  2. $Q(s, a) = 1/N(s, a) \sum_{s'|s,a\to s'} V(s')$.
- Finally, $\pi = \alpha_\theta(s) \propto N(s, a)^{1/\tau}$ (τ is a temperature parameter here).

Competitive: AlphaGo Zero MCTS

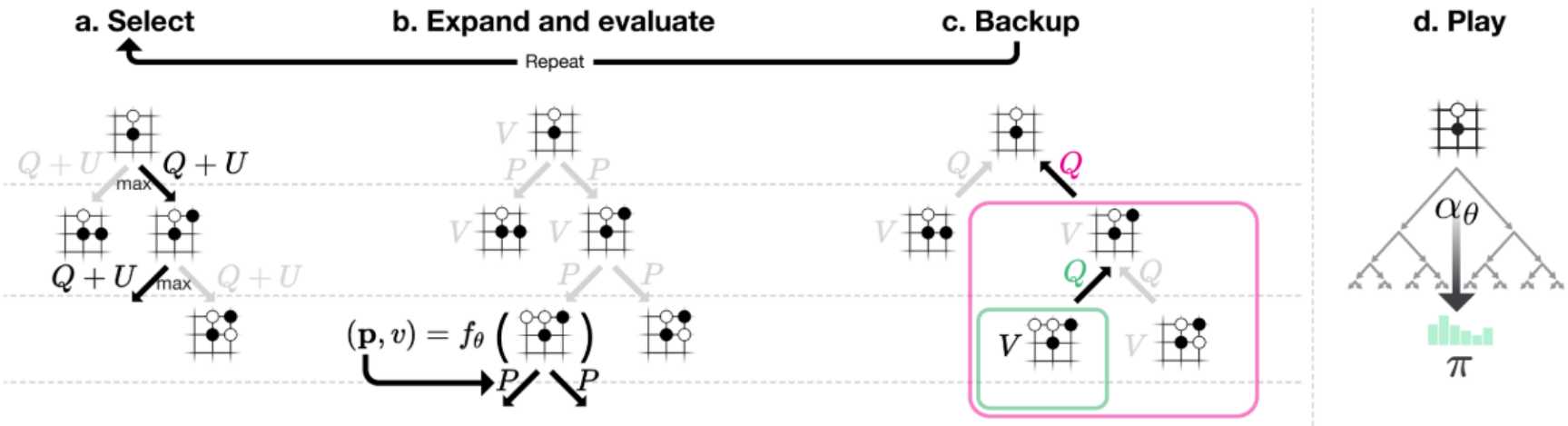


Figure: MCTS in AlphaGo Zero.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., ... Hassabis, D. (2017). Mastering the game of go without human knowledge.

Competitive: AlphaGo Zero training

$$loss = (z - v)^2 - \boldsymbol{\pi}^T log \boldsymbol{p} + c||\theta||^2$$

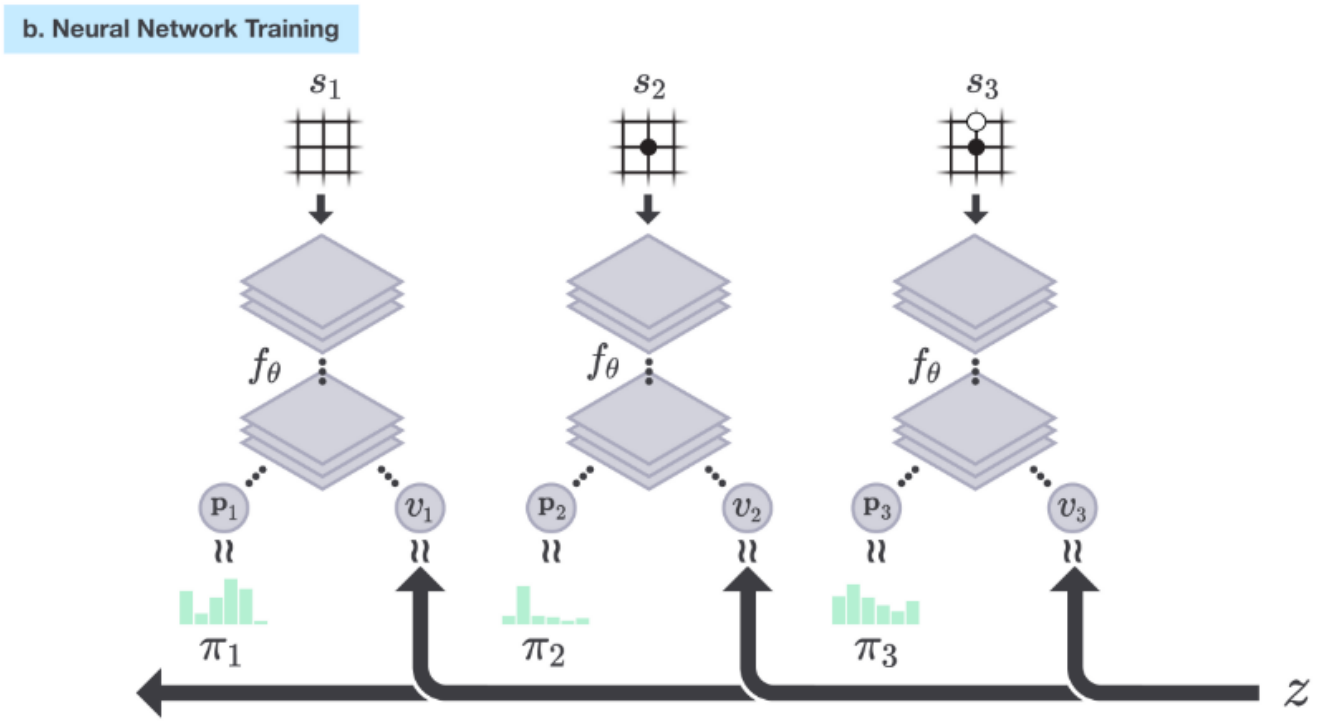


Figure: Training in AlphaGo Zero.

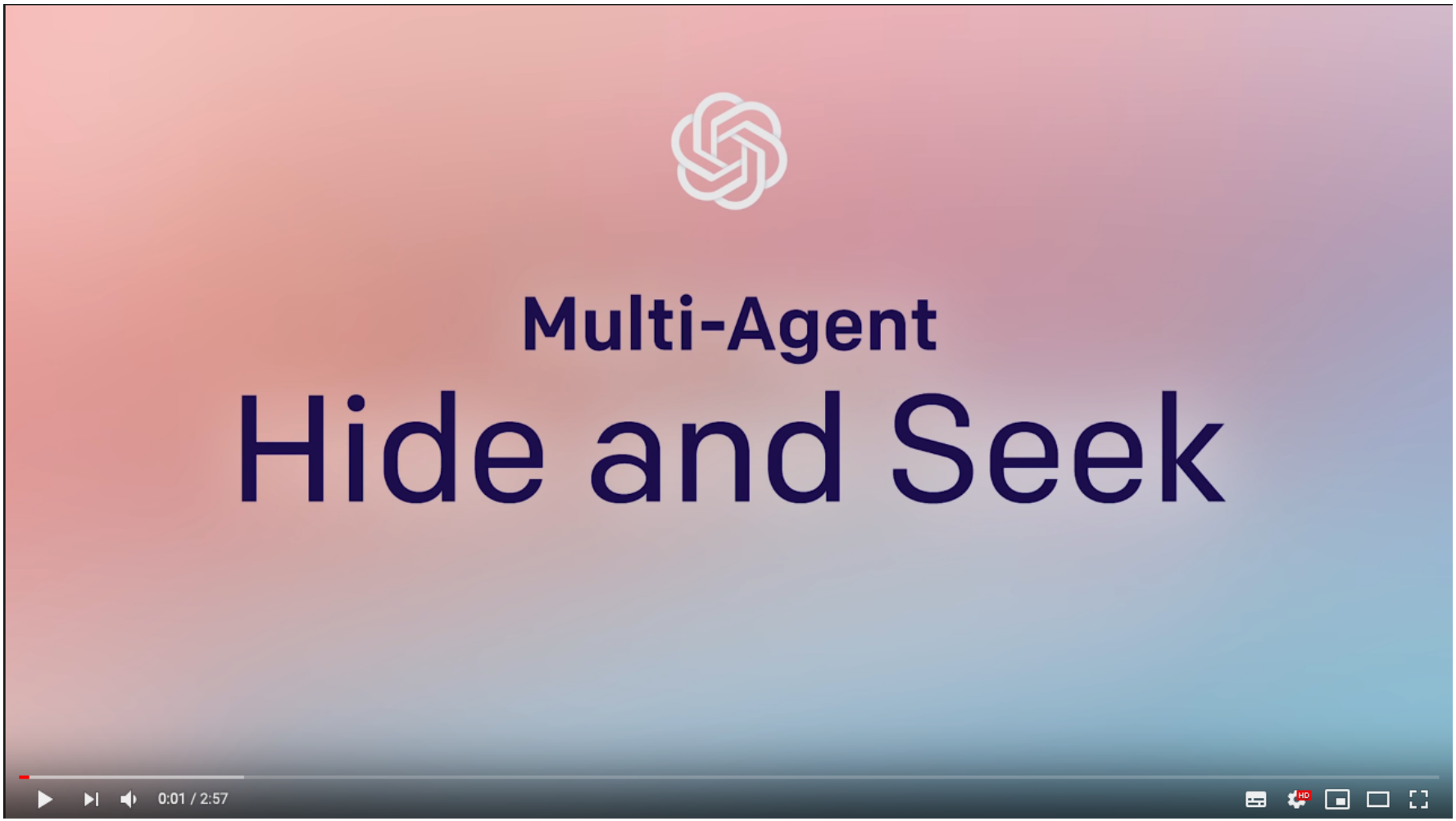
Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., ... Hassabis, D. (2017). Mastering the game of go without human knowledge.

Competitive: AlphaGo Zero numbers

Some random numbers after 3 days of training:

- 4.9 million games generated.
- 1600 games for each MCTS ($\sim 0.4s$).
- $700,000$ minibatches of $2,048$ states.
- Outperform AlphaGo Lee after 36 hours.
- AlphaGo Zero on 4 TPUs and AlphaGo Lee on 48 TPUs.

Teaser

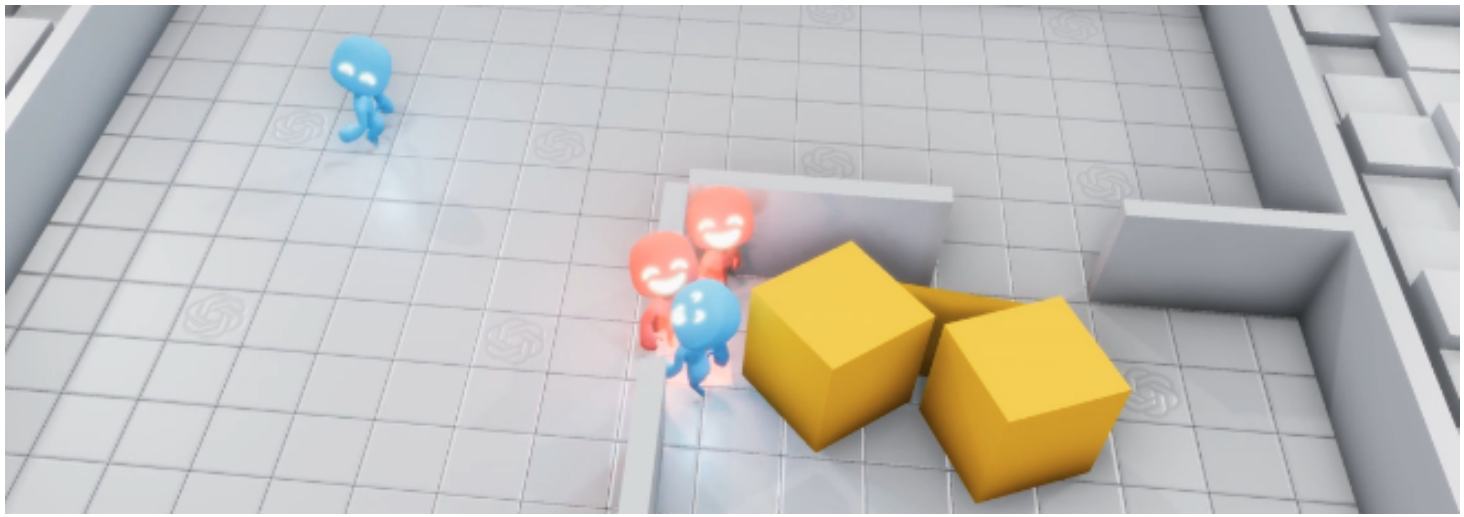


https://www.youtube.com/watch?v=kopoLzvh5jY

Hide and Seek environment

- Hiders (blue) trie to avoid line of sight of the Seekers (red).
- Objects can be grabbed or locked.
- Preparation phase: only the hiders can perform action in the beginning.

- Team based reward
- Hiders: +1 if hidden, -1 if seen.
- Seekers: opposite.
- 0 if no one is seen by the seekers.
- Time limit: 240



Hide and Seek environment

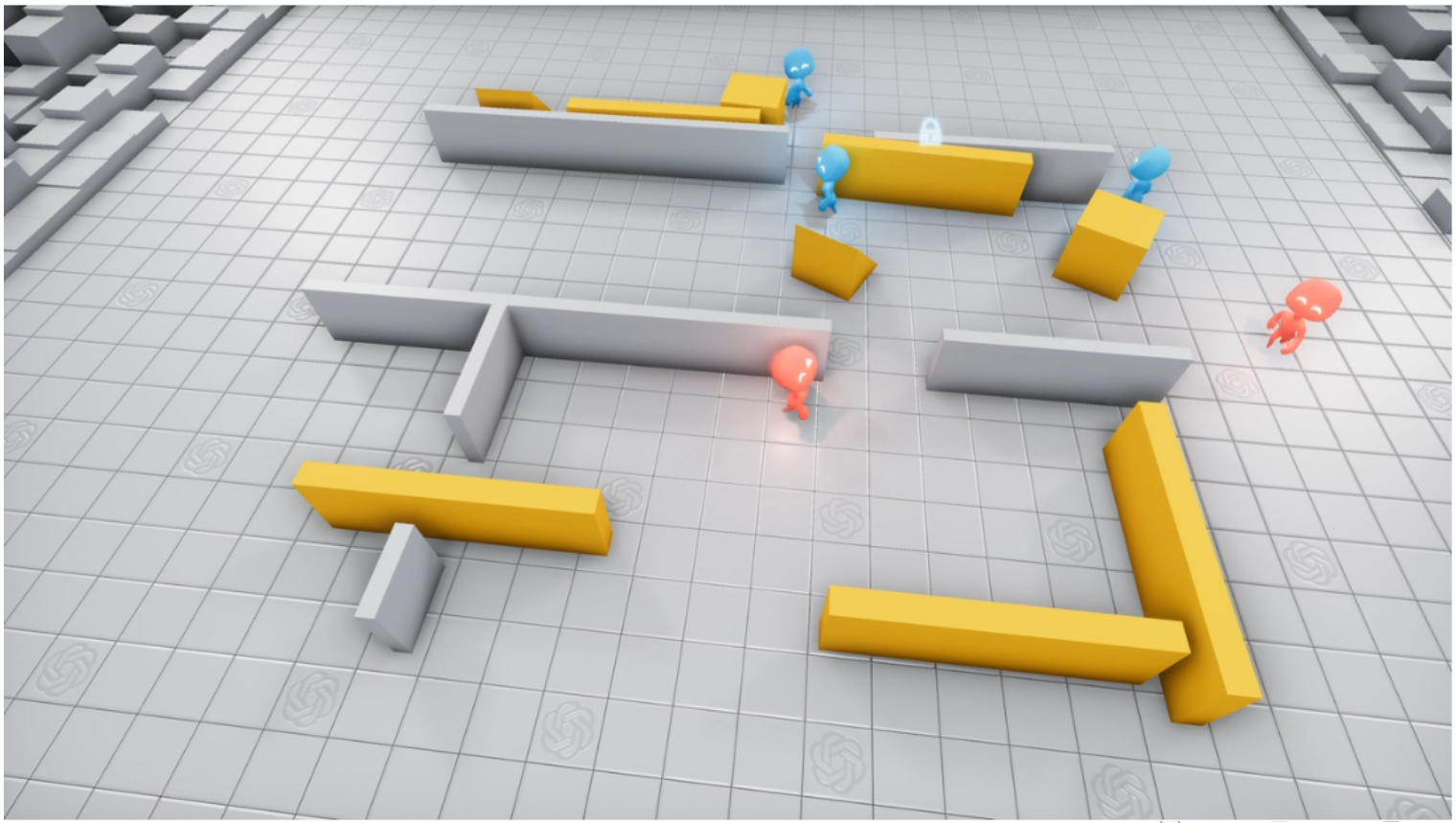
- Action space:
  - Move: discretized forces along x and y axis and torque around their z axis.
  - Grab or lock the closest object. 2 binaries. (Only object in front of them within a small range)
    - Grab: the object is bound to the agent while boolean is True.
    - Lock: the object is locked and cannot be moved. Unlocking is available if the agent is part of the agent team that has locked the object.
- 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.
- Go to the blog!

Figure: `https://openai.com/blog/emergent-tool-use/`

Hide and Seek scenarios

Normal environment:

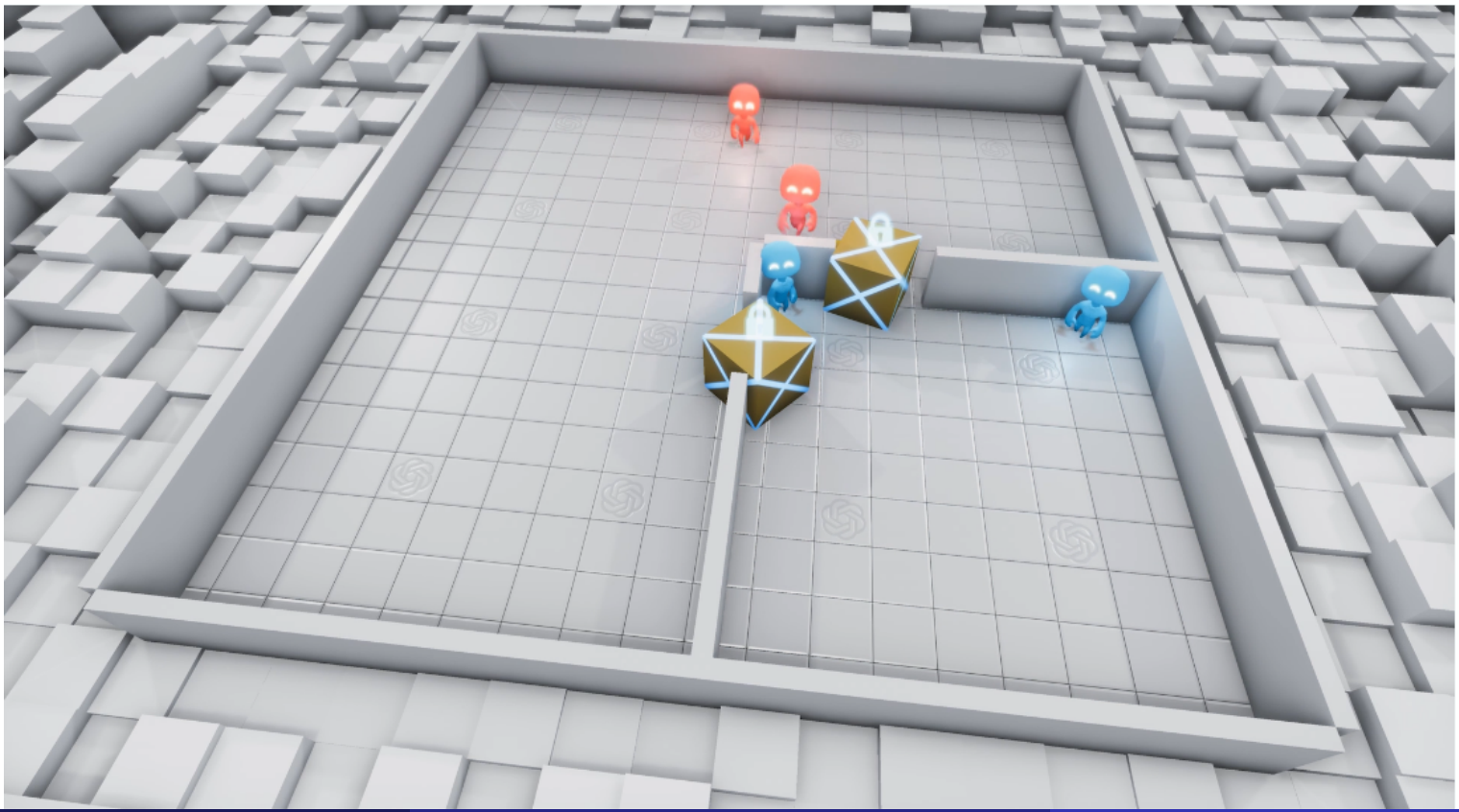
- 1 to 3 hiders, 1 to 3 seekers
- 3 to 9 boxes (at least 3 elongated)
- 2 movable ramps
- random walls and rooms
- random initial position



Hide and Seek scenarios

Quadrant environment:

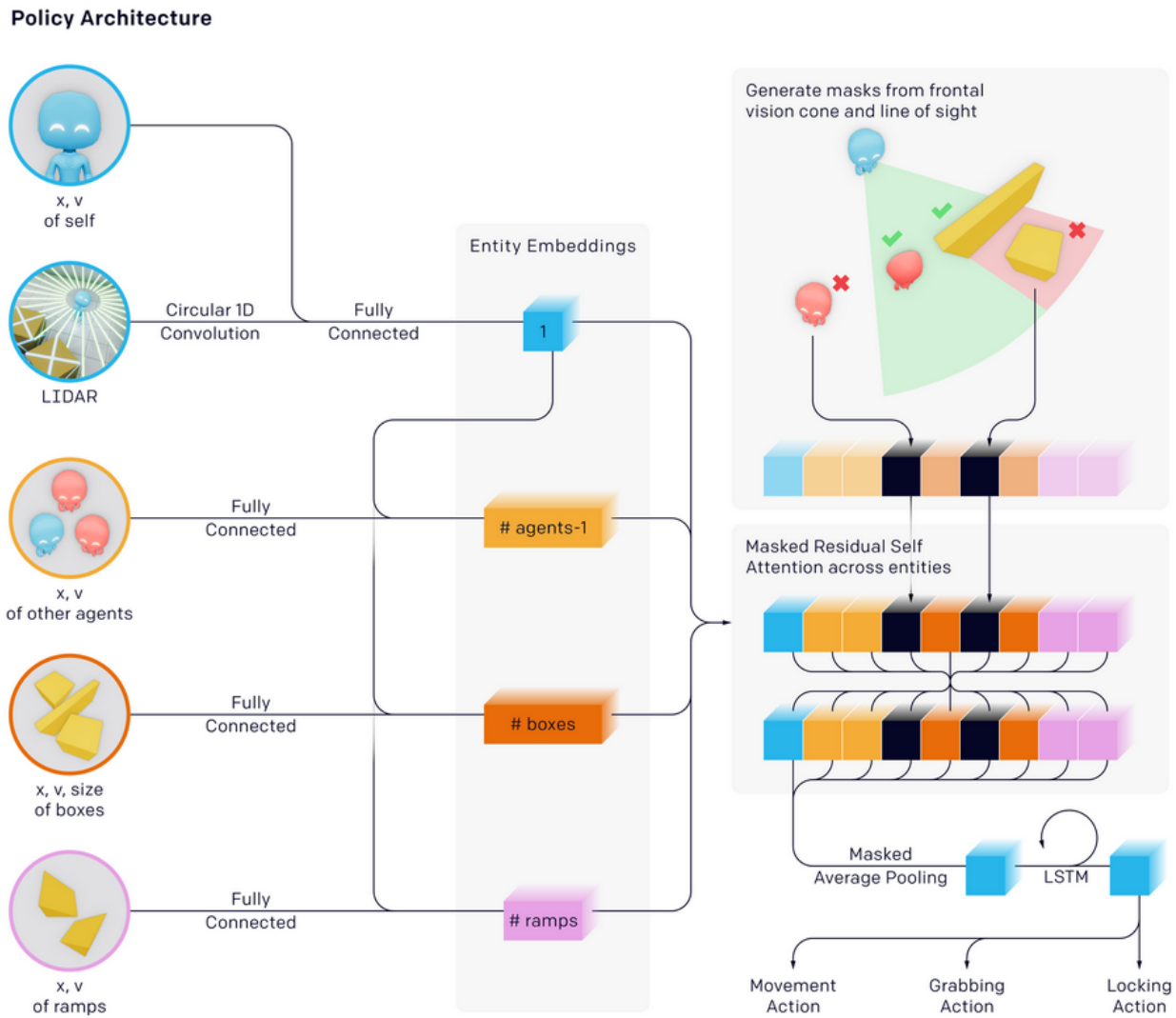
- 2 hiders, 2 seekers
- 2 boxes inside the room
- Seekers spawn outside the room
- 1 movable ramp that cannot be blocked
- 1 fixed room with 1 or 2 random doors



Hide and Seek optimization

- Actor-Critic $\rightarrow$ 2 sets of parameters.
- The actor network, optimised with PPO and GAE, produces an action distribution based on the agent observations.
- The critic network that predicts the discounted future returns.
- Use centralize training, decentralized execution (CTDE).
  - At training time: the critic has access to the full state to learn a centralized value function.
  - At execution time: 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.

Hide and Seek architecture



Hide and Seek entity-centric observations

- The observation is composed of different entities: agents, boxes, ramps.
- **The first step** is to embed each entity. Same objects are embedded with the same parameters.
- Self information: the concatenation of Lidar (that went through a circular 1-D convolution) with position and velocity of the agent are embedded with a fully connected layer.
- There are three other entities. Each one is concatenated with the embed of the self information before their own embed:
  1. Other agents information ($N-1$): position and velociity.
  2. Box information (Number of boxes): position, velocity and size.
  3. Ramp information (Number of ramps): position and velocity.

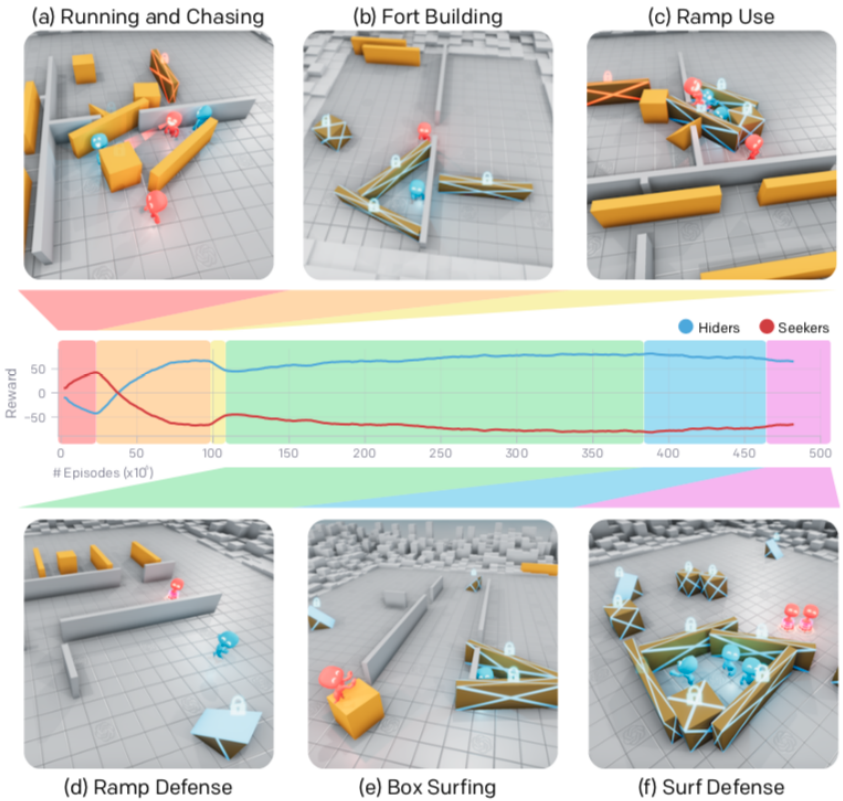
  The number of these entities varies depending on the scenario.

Hide and Seek Policy optimization

- **The second step** is to pass all embedded entities through a residual self-attention block (unobservable entities are masked away).
- Attention mechanisms allow to capture object-level information.
- "A self-attention module takes in n inputs, and returns n outputs. The self-attention mechanism allows the inputs to interact with each other ("self") and find out who they should pay more attention to ("attention"). The outputs are aggregates of these interactions and attention scores."
- **The third step** is to concatenate the average-pool entities embedding with self information.
- **The fourth step** is to pass through a LSTM.
- **The last step** is to pass through the three sperate heads (one for each action type)
- Full parameter values are detailed in the paper.

Hide and Seek Emergent behavior

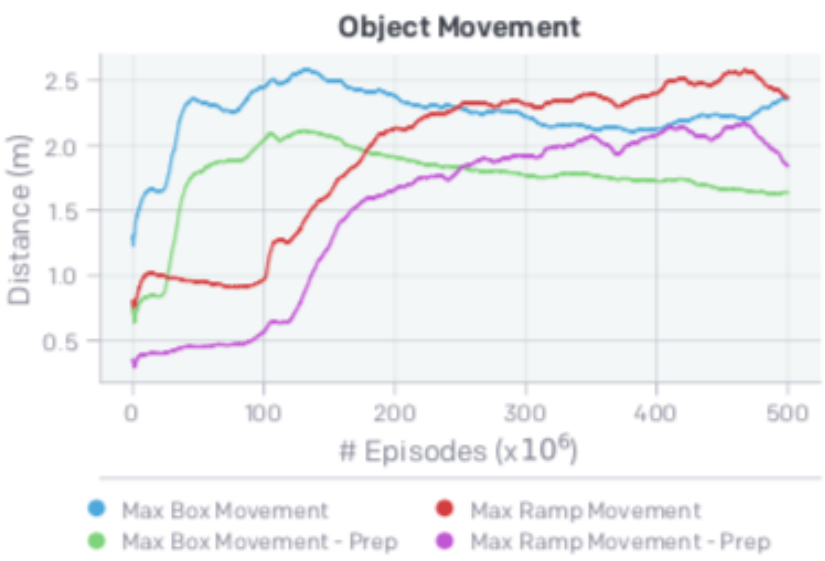
- In both scenarios, different strategies emerge as the agents train → autocurriculum. They can be seen on the blog.
- In the paper, they insist on the fact that there is no incentives for agents to interact with objects, this is only a result of autocurriculum induced by the competition.

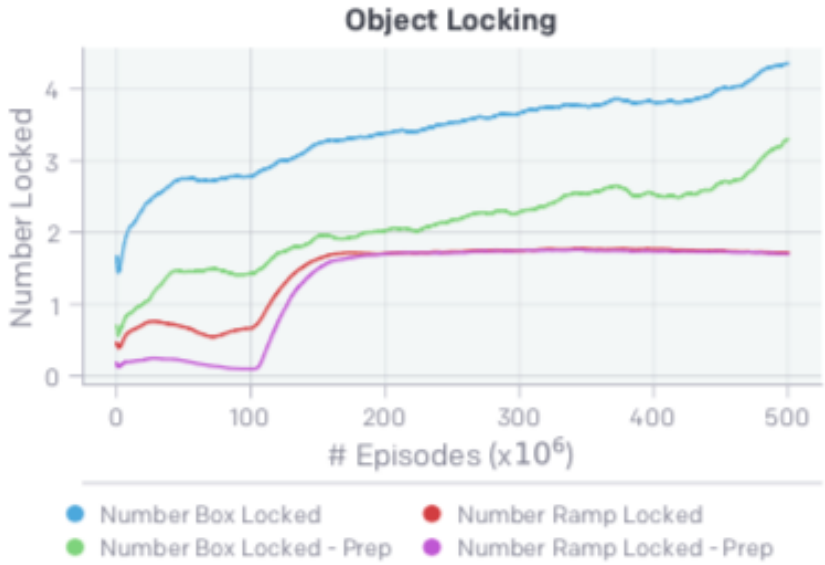


Hide and Seek Emergent behavior

Tool interaction

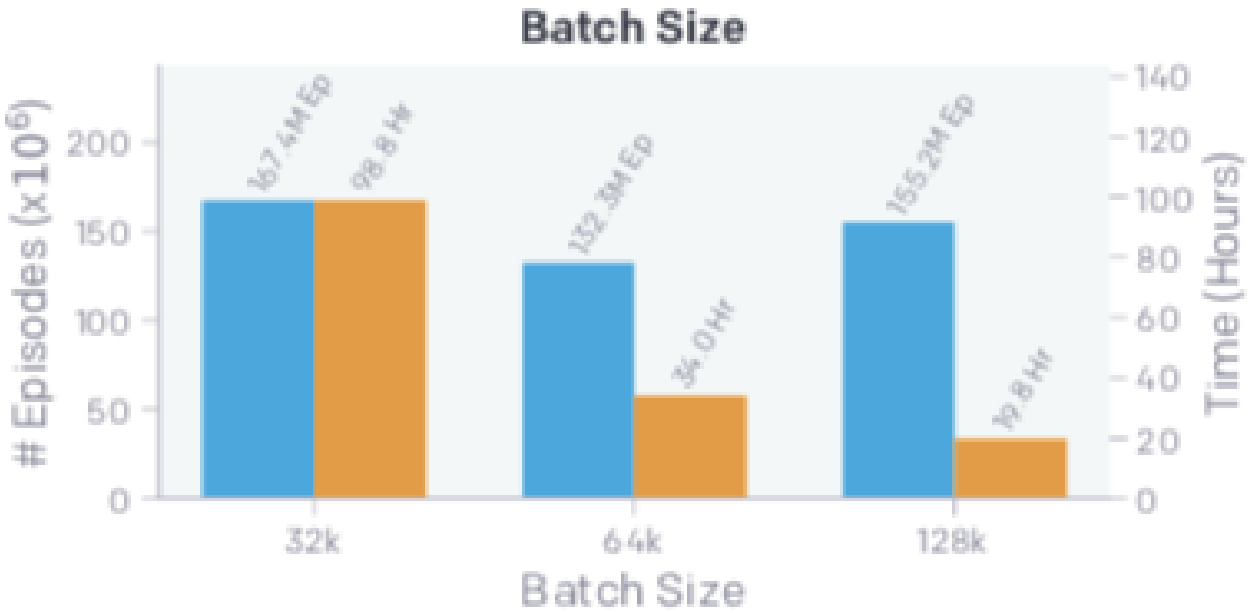
- Agents learn to divide the labor: Against box surfing, 2 and 3 hiders lock 25% and 36% more boxes than a single hider.
- It is possible to track the stages of emergent policy with the interaction of tools in the environment.





Hide and Seek Emergent behavior

- Scale matters!
- In the Figure, the number of episode (blue) and time (orange) **to reach stage 4** (ramp defense) in term of batch size.
- The size of batch size is the number of transition chunks.
- The model uses 1.6 million parameters.



Hide and Seek Evaluation

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 and measures improvement compared to other policies. However, it does not differentiate adaptation and improvement of already learned skills.
- Their propose two evaluations scheme.
  1. Comparison to intrinsic motivation.
  2. Inteligence tests: transfer and fine-tuning.

Hide and Seek Intrinsic motivation

- **Goal?** Compare behavior learned in Hide and Seek with common unsupervised exploration techniques.
- Underlying idea: Can agent behave like human with these unsupervised exploration methods.
- Count based exploration: the agent receives reward when visiting states that has not been visited much.
- Very dependent of state representation.
- **Result**: The learned behavior is very not human-like.
- See blog results.

Hide and Seek Intelligence tests

- **Goal?** Use transfer learning to evaluate network parameters on five new tasks.
- Three configuration:
  1. Trained from scratch
  2. Pre-trained with Hide and Seek and fine tuned.
  3. Pre-trained with count based and fine tuned.
- These tests include supervised learning and reinforcement learning but are single-agent:
  1. Object counting
  2. Lock and return
  3. Sequential lock
  4. Construction from blueprint
  5. Shelter construction
- Notes:
  - Same spaces and observation (fake hiders)
  - Object counting task: the action heads are replaced by a classification head (7-classes representing whether 0 through 6 boxes have gone to the left)

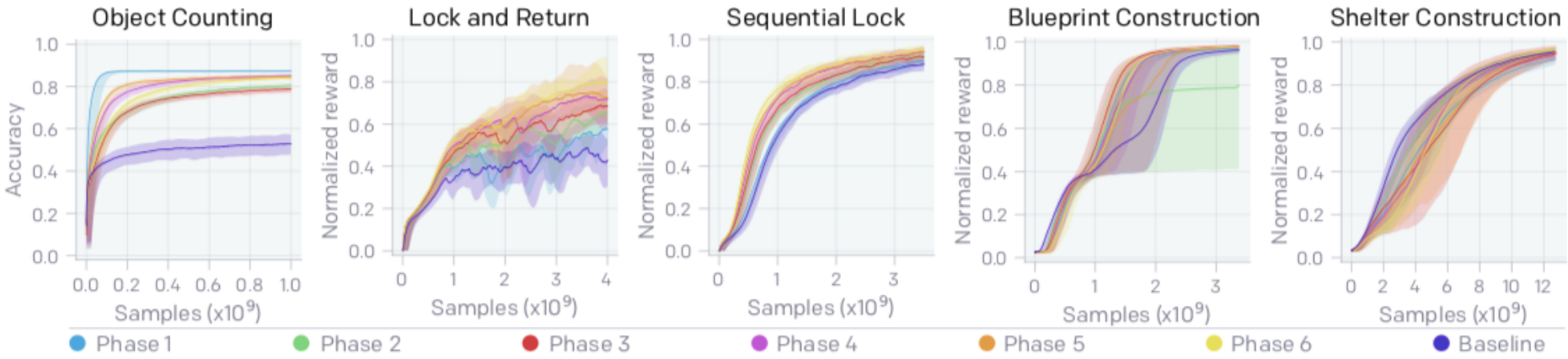
Hide and Seek Intelligence tests

Results:

- Pre-trained with Hide and Seek configuration is better on Lock and return, Sequential lock and Construction from blueprint.
- Pre-trained with count-based configuration is better on Object counting.
- Pre-trained with Hide and Seek configuration achieves the same results as the trained from scratch configuration on Shelter construction but sligthly slower.

More intelligence tests

Intelligence tests are performed at different phases of emergence.



- Improves on navigation and memory as it progresses.
- Object counting is transient.
- Performance on manipulation tasks (constructions) is uncorrelated to the phases. The most surprising: policy from phase 1 performs comparably well to others.

Alternative games

Hide and Seek with food reward.

- Can eat food if:
  - after preparation phase.
  - hiders not seen by seekers.
  - hider close and visible to the food.

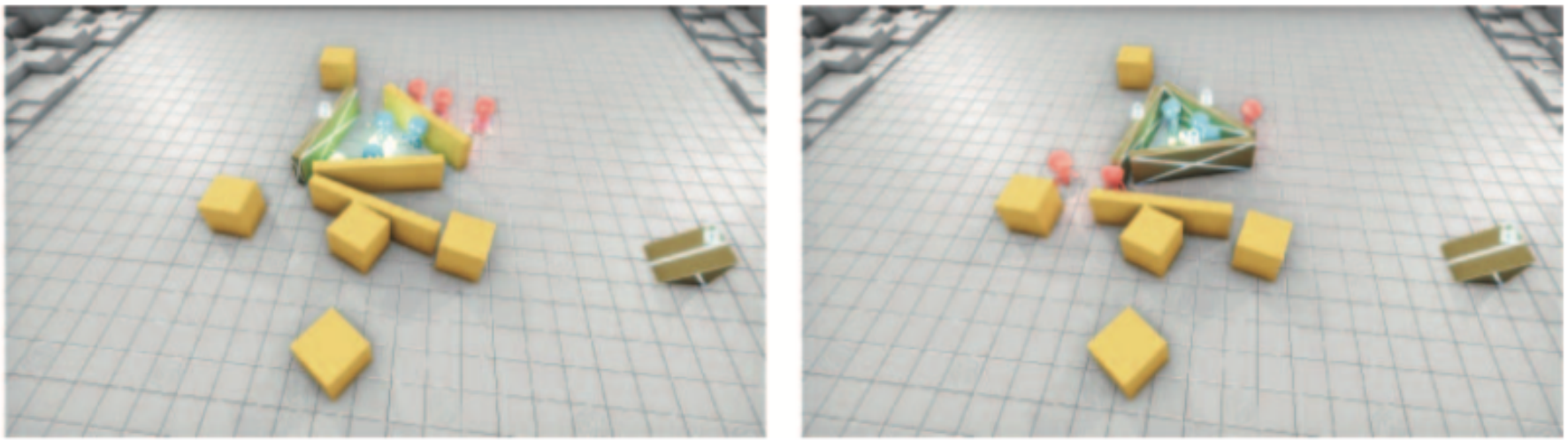


Figure: Hide and Seek with food reward.

Alternative games

Hide and Seek with food reward: Results.

- Incentivized to build forts around food.
- Four levels of skill progression (see Figure)

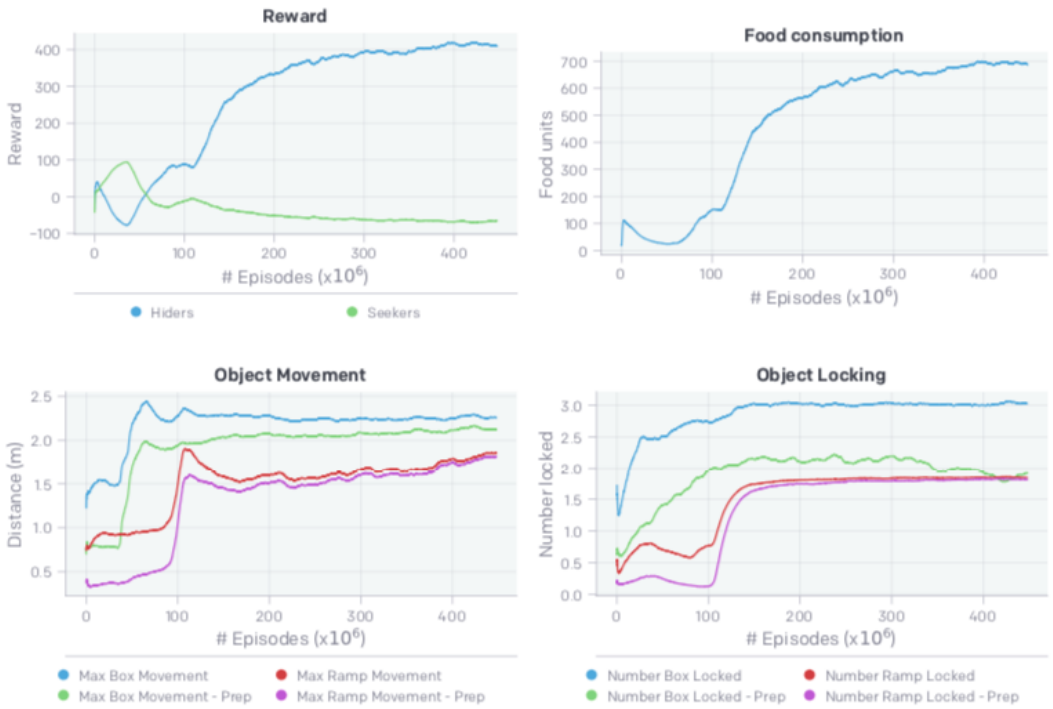


Figure: Hide and Seek with food reward results

Alternative games

Hide and Seek with dynamic food.

- Only one food that disappear when eaten.
- It reappears in the center of the map (in a square of length 1/5 of game area size).
- The hidders need to learn to build large forts.
- emerge after 45 billions samples.
- if food region ratio is 1/6, emerge after 15 billions samples.
- if food region ratio is 1/4, the hiders ignore food and protect themselves.

Conclusions

- Simple game rules and multi-agent competition can induce agents to learn complex strategies and skills.
- Intelligent tests, using transfer learning, can be used to evaluate learning progress and to compare agents in a same domain.
- Multi-agent autocurricula can lead to physically grounded and human-relevant behavior (in opposition to unsupervised exploration techniques).

Competitive: Other works

Recent work has been done in competitive games:

- Dota 2, OpenAI Five (2019): Dota 2 with large scale deep reinforcement learning.
- StarCraft 2, AlphaStar (2019): Grandmaster level in StarCraft II using multi-agent reinforcement learning.
- Quake 3 capture the flag (2019): Human-level performance in 3D multiplayer games with population-based reinforcement learning.

Adversarial attacks

Adversarial attacks

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

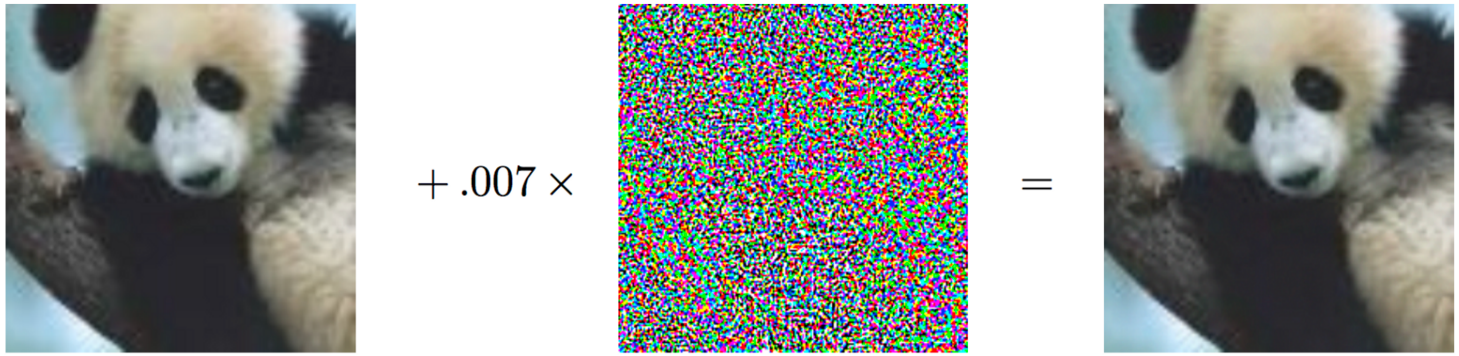


Figure: Adversarial attack in classification.

Goodfellow, I. J., Shlens, J., Szegedy, C. (2014). Explaining and harnessing adversarial examples.

Adversarial attacks

"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.

Examples: `https://adversarialpolicies.github.io/`

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.

Summary

A non-exhaustive overview of multi-agent reinforcement learning.

- Cooperative setting with partial observations:
  1. QMIX
  2. COMA
  3. QVMIX
- Communication:
  1. RIAL
  2. DIAL
- Competitive setting:
  1. Minimax-Q
  2. AlphaGo Zero
  3. Hide and seek
- Adversarial policies

Questions?

References

- 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.
- 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.

References

- 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.
- 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.

References

- RIAL and 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.

References

- 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.

References

- 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.
- Adversarial policies: Gleave, A., Dennis, M., Wild, C., Kant, N., Levine, S., Russell, S. (2019). Adversarial policies: Attacking deep reinforcement learning.