# INFO8003-1
# Optimal decision making for complex problems

## Multi-Agent Reinforcement Learning

Pascal Leroy

April 2021

# 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)

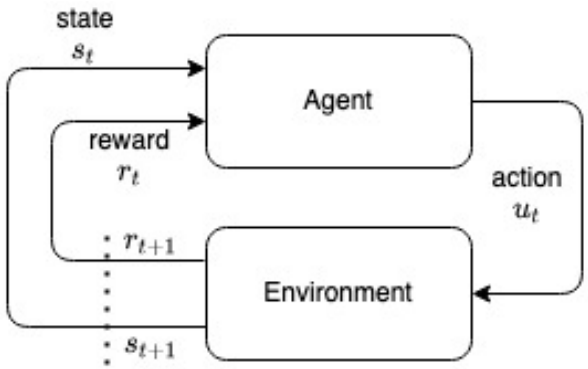



Figure: RL environment.

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 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 $\pi^*$.

---

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 $\pi$:

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

- State-Action Value of a policy $\pi$:

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

- 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 $\theta$:

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

- The replay buffer $B$ is a collection of transitions.
- Sampling transitions allows to update the network.
- $\theta'$ denotes the parameters of the **target network**, a copy of $\theta$ that is periodically updated.

- To play Atari games, $\theta$ is a CNN.
- When the environment is partially observable, $\theta$ is a recurrent network(DRQN) and $B$ stores sequences of transitions.

# RL basics: Policy Gradient

Policy Gradient:

We denote the discounted sum of reward of a trajectory by $G_t(\tau) = \sum_{j=0}^{T} \gamma^j r_{t+j}$ where $\tau = (s_t, u_t, r_t, s_{t+1}, u_{t+1}, .., s_T)$.

Reinforce:

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

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

Smaller variance with a baseline $b$:

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

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

- Actor $\theta$, 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 $\phi$, learns the advantage $Q(s_t, a_t) - V(s_t)$:

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

# MARL: Framework

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

- 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 for each agent: $r_t^{a_i} = R^{a_i}(s_{t+1}, s_t, \boldsymbol{u_t})$.
- 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 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: board games, video games,...

3. General sum setting: lies in between the two others.
   Examples: everything else that is not cooperative or competitive.

# Cooperative setting

# Cooperative: Dec-POMDP

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

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

Such Markov Games are called Decentralised-POMDP.

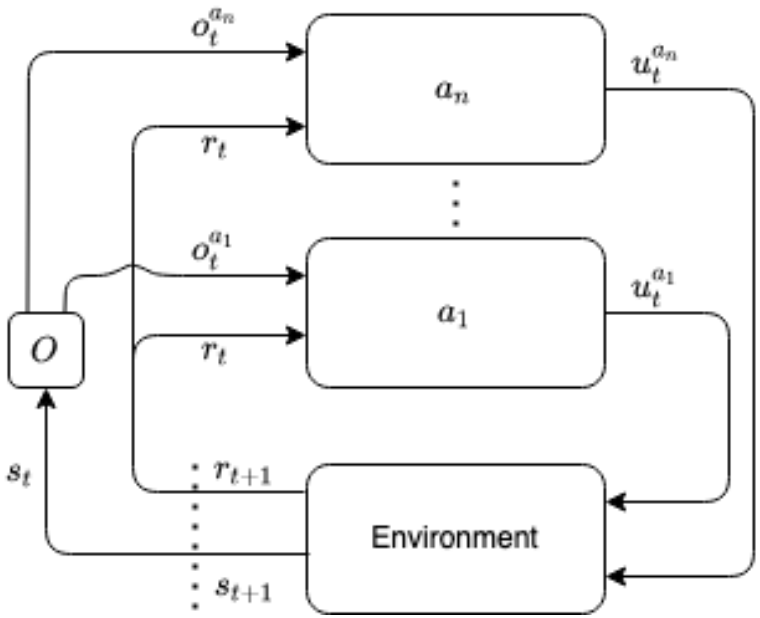



Figure: Dec-POMDP.

# Cooperative: SMAC

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).

`http://whirl.cs.ox.ac.uk/blog/smac/`

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

# Cooperative: Centralised vs decentralised

Centralised controller?

- It is possible to train a single agent to control all agents.
- Problems?
  - Joint action space scale exponentially with $n$.
  - What about the partial observability?
    $\rightarrow$ Not possible to centralise.
- Solutions?
  - Decentralised controller.
    $\rightarrow$ Naive learner: train with SARL algorithms.
  - Centralised training with decentralised execution (CTDE).
    $\rightarrow$ Use 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:

- Non-stationarity, the other agents are also learning: How an agent maximises the joint-action reward knowing only its action?
- Credit assessment: How an agent learns whether its actions is the one that lead to good (or bad) reward?

# Cooperative: Value-based methods in CTDE

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

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

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

Condition: Individual Global Max (IGM):

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

# Cooperative: VDN and QMIX

How to satisfy IGM?
Value Decomposition Network:

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

Problems:

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

How can we build non-linear factorisation satisfying IGM?
QMIX idea is to enforce monotonicity:

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

# Cooperative: QMIX

How to build a non-linear monotonic factorisation of $Q(s_t, \boldsymbol{u_t})$ as a function of every $Q_a(s_t, u_t^a)$ and $s_t$ with neural networks?
In QMIX, this is done with a hypernetwork which is a neural network that compute the weight of a second neural network.
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 used by the DQN algorithm but applied to $Q_{mix}(s_t, \boldsymbol{u_t})$.
Parameters of individual network are shared to speed up learning.

# Cooperative: QMIX architecture

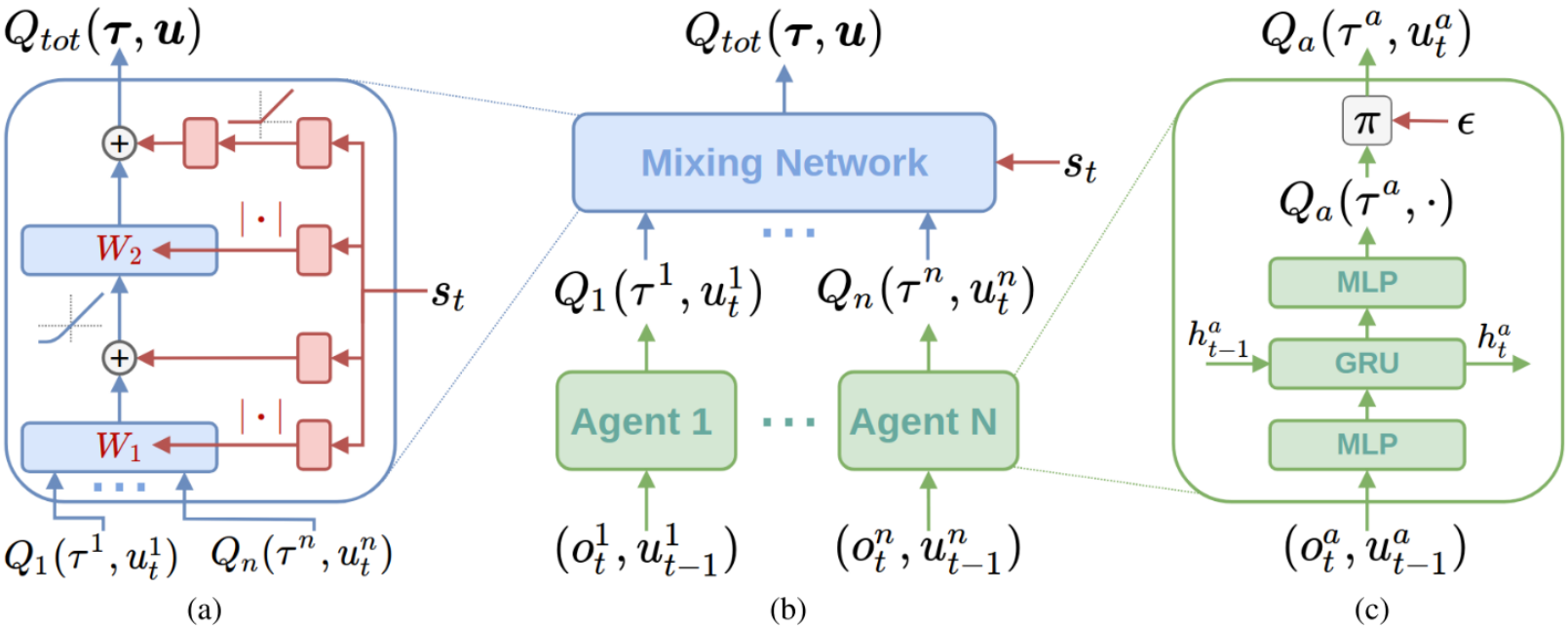



*Figure 2.* (a) Mixing network structure. In red are the hypernetworks that produce the weights and biases for mixing network layers shown in blue. (b) The overall QMIX architecture. (c) Agent network structure. Best viewed in colour.

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

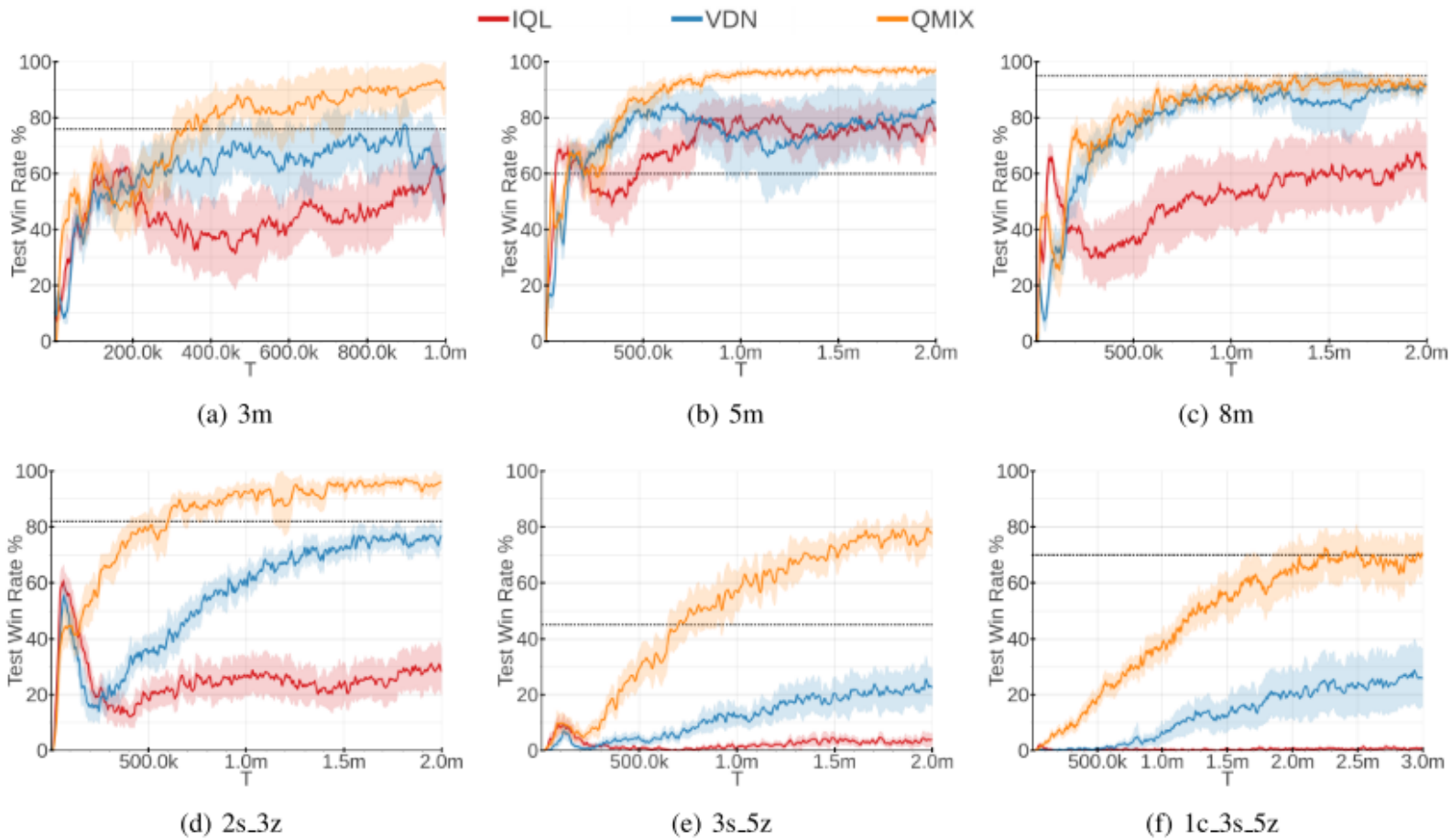



*Figure 3.* Win rates for IQL, VDN, and QMIX on six different combat maps. The performance of the heuristic-based algorithm is shown as a dashed line.

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.

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

Solutions proposed in COMA (counterfactual multi-agent):

- Critic is only used during training.
- It is possible to have a centralised critic that computes the advantage based on $s_t$ instead of $\tau_t$.

In COMA, they compared two different Critic (centralised and IAC):

- Central-V, learns V: $A(s_t, u_t; \phi) = r_t + \gamma V(s_{t+1}; \phi) - V(s_t; \phi)$.
- Central-QV, learns Q:
  $A(s_t, u_t; \phi) = Q(s_{t+1}, \boldsymbol{u}; \phi) - \sum_{u_a} \pi_\theta(u_t^a, s_t) Q(s_t, \boldsymbol{u}; \phi)$.

---

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

# Cooperative: COMA

Problem of central-V and central-QV:

- These advantages are based on global rewards.
- The centralised critic does not solve the credit assignment problem.

Solution: use a counterfactual baseline.

- Inspired from difference reward: $D^a = r(s, \boldsymbol{u}) - r(s, (\boldsymbol{u}^{-a}, c^a))$ where the common reward is compared to a reward obtained when agent $a$ executes a default action $c^a$, other agent actions $(\boldsymbol{u}^{-a})$ 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

COMA 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

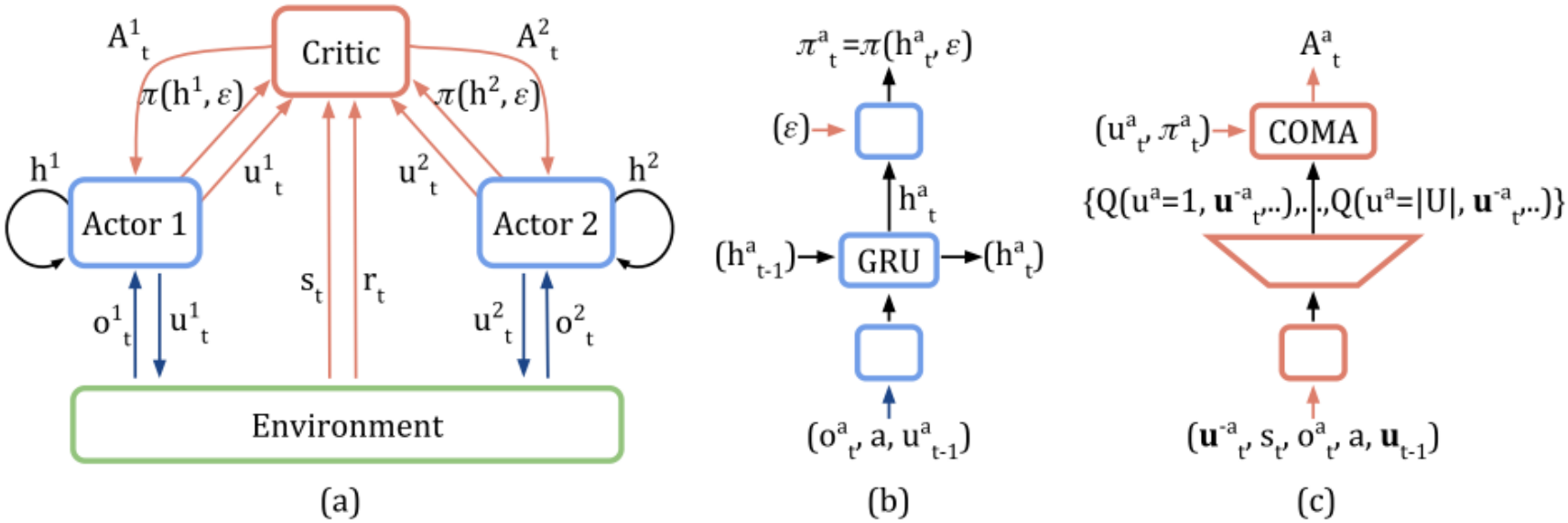



Figure 1: In (a), information flow between the decentralised actors, the environment and the centralised critic in COMA; red arrows and components are only required during centralised learning. In (b) and (c), architectures of the actor and critic.

Figure: COMA architecture.

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

# Cooperative: COMA results

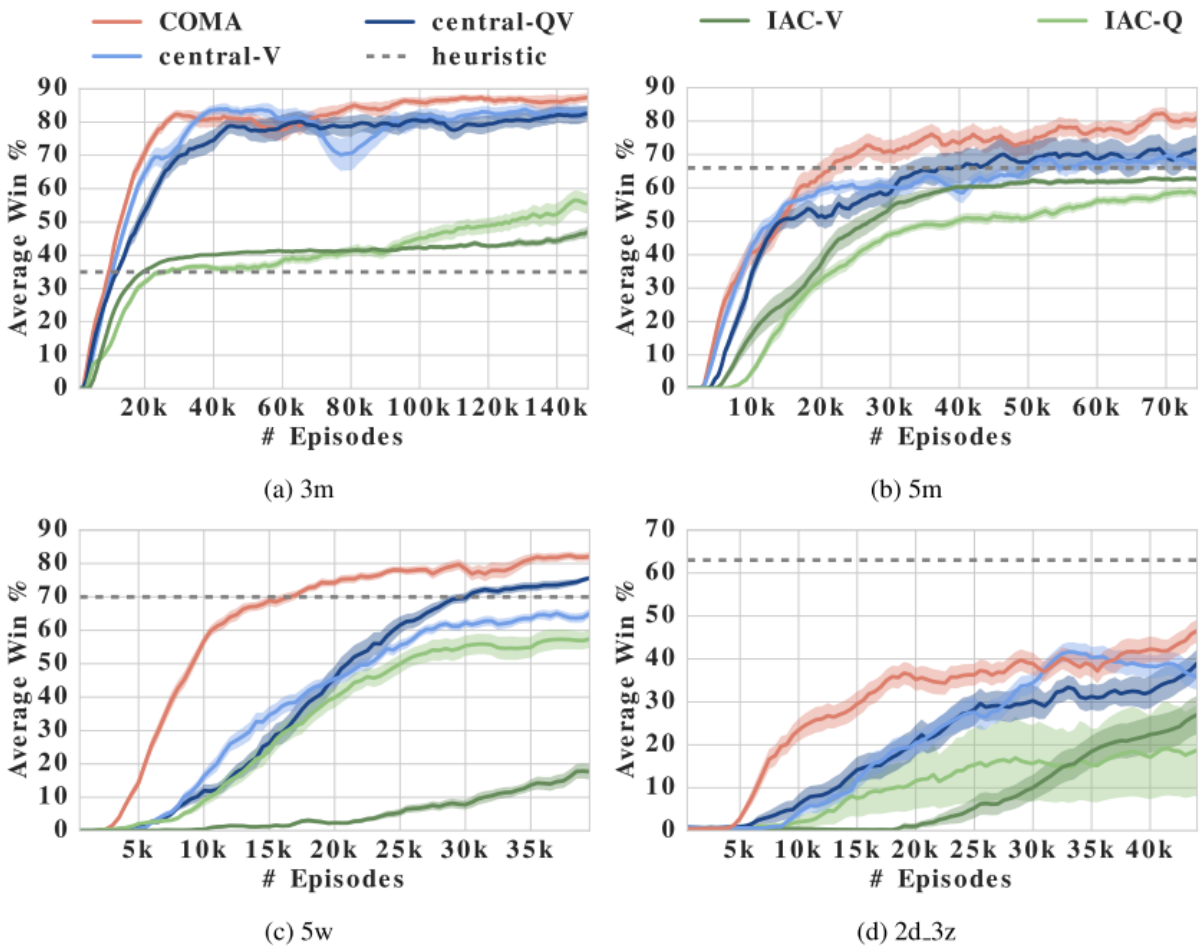



Figure 3: Win rates for COMA and competing algorithms on four different scenarios. COMA outperforms all baseline methods. Centralised critics also clearly outperform their decentralised counterparts. The legend at the top applies across all plots.

---

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

# Cooperative: COMA vs QMIX

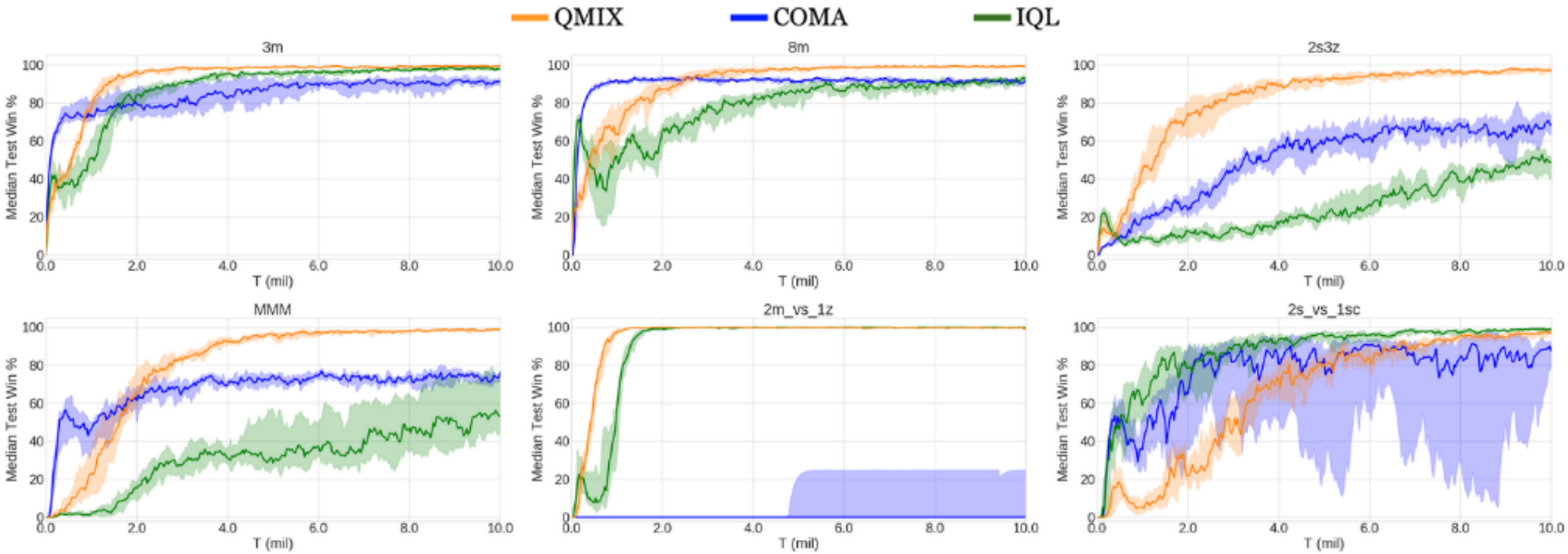



Figure: Median win rates for QMIX, COMA, and IQL on easy SMAC scenarios.

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

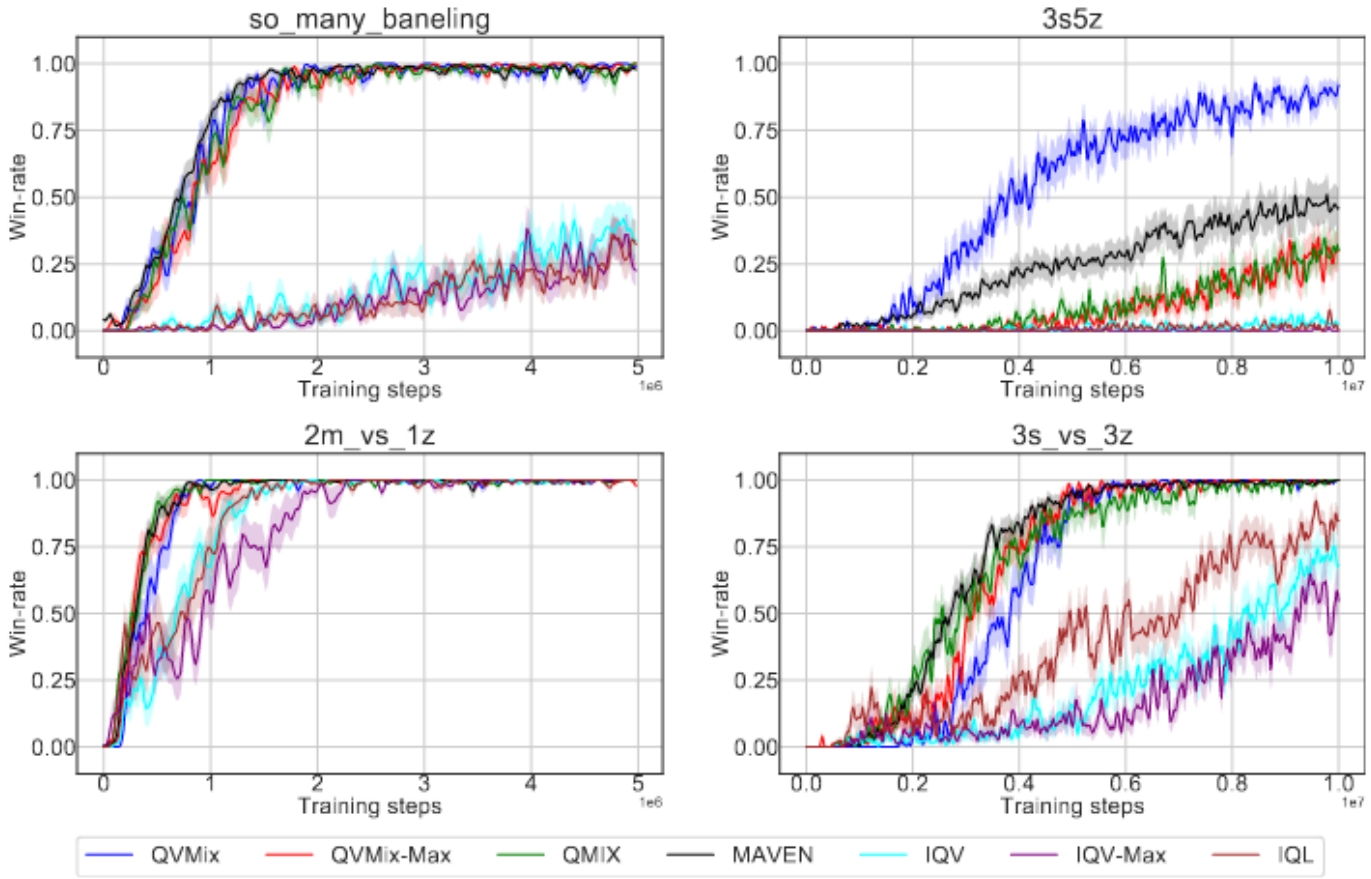



Figure 3: Means of win-rates achieved by QVMix, QVMix-Max, QMIX, MAVEN, IQV, IQVMax and IQL in eight scenarios. Top to bottom, left to right, the scenarios are `3m`, `8m`, `so_many_baneling`, `2m_vs_1z`, `MMM`, `2s3z`, `3s5z` and `3s_vs_3z`. The error band is proportional to the variance of win-rates.

---

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

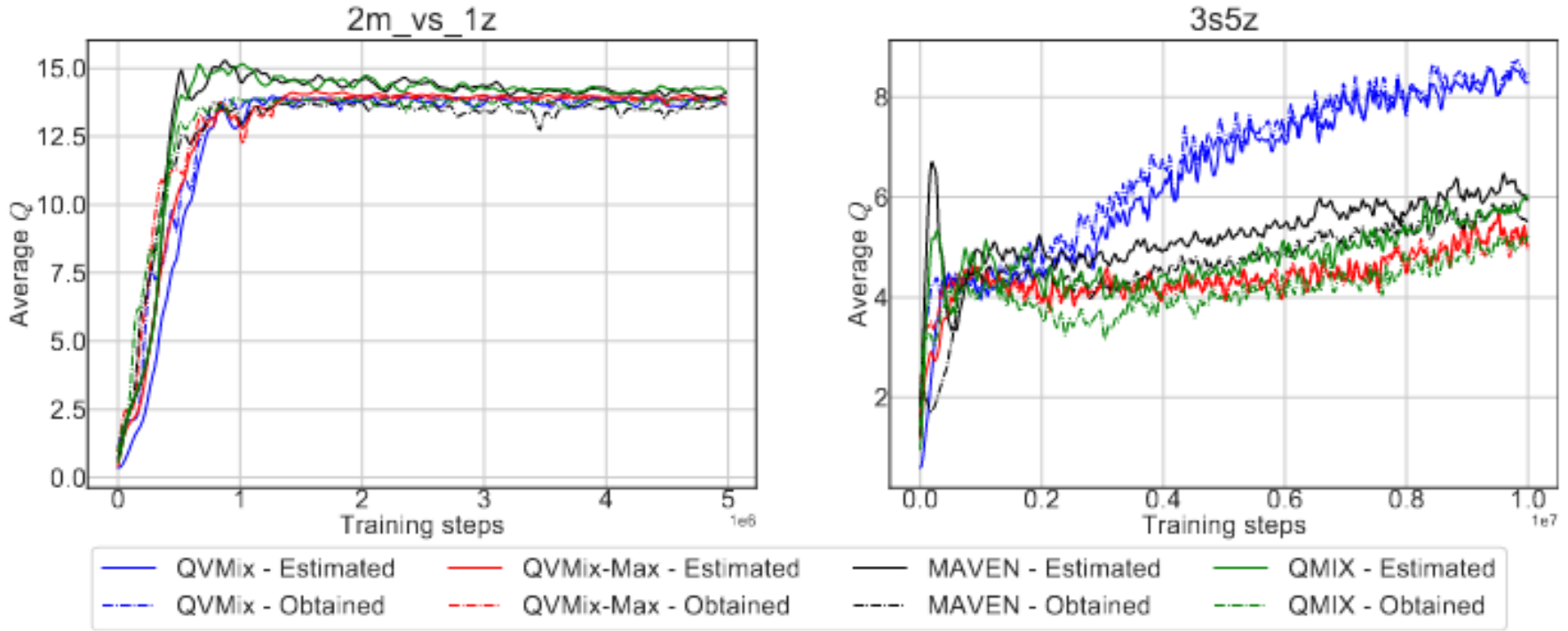



Figure 4: $Q$ values obtained and estimed when training QVMix, QVMix-Max, MAVEN and QMIX. Dash-dotted lines represent the obtained $Q$ values while solid lines represent the estimated ones.

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 Learnin

# 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.

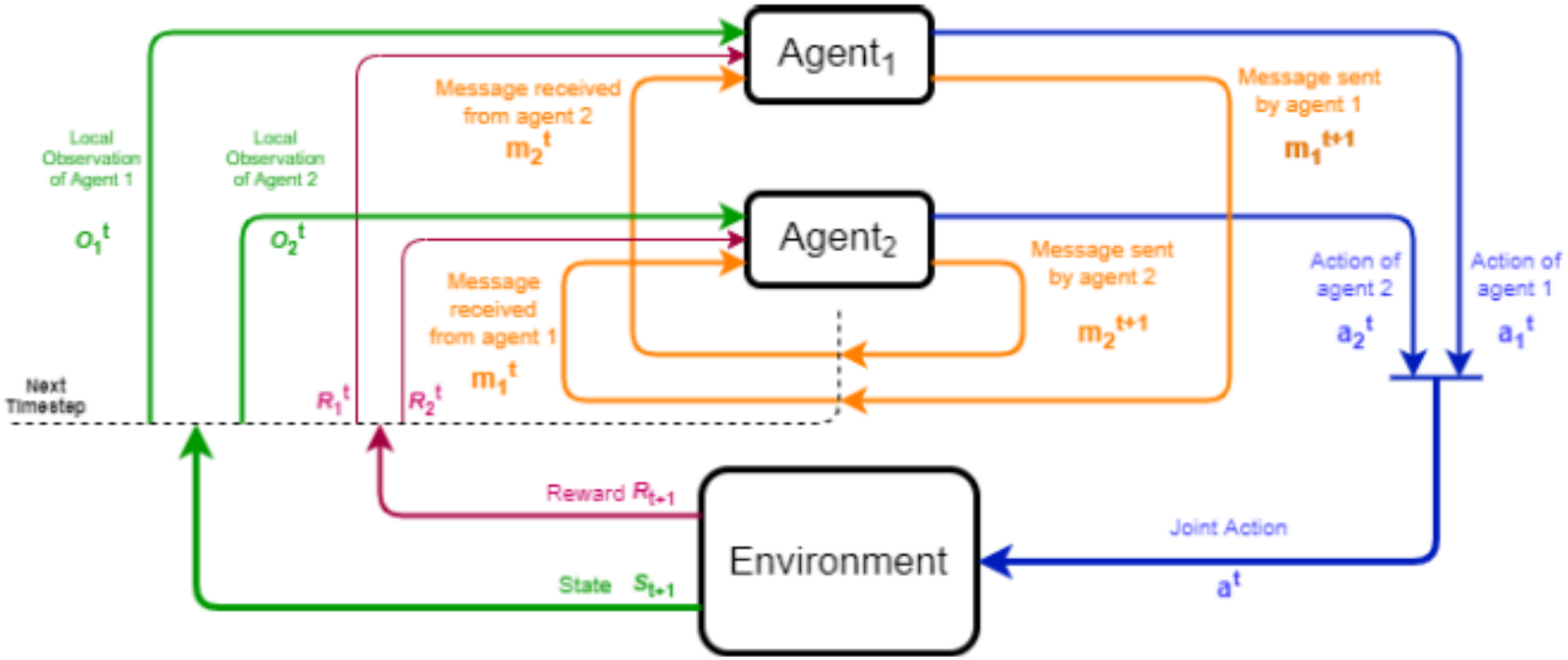



Figure: Communication framework (Fombellida, A. (2020) Battlefield Coordination using Multi-Agent Reinforcement Learning)

# 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.

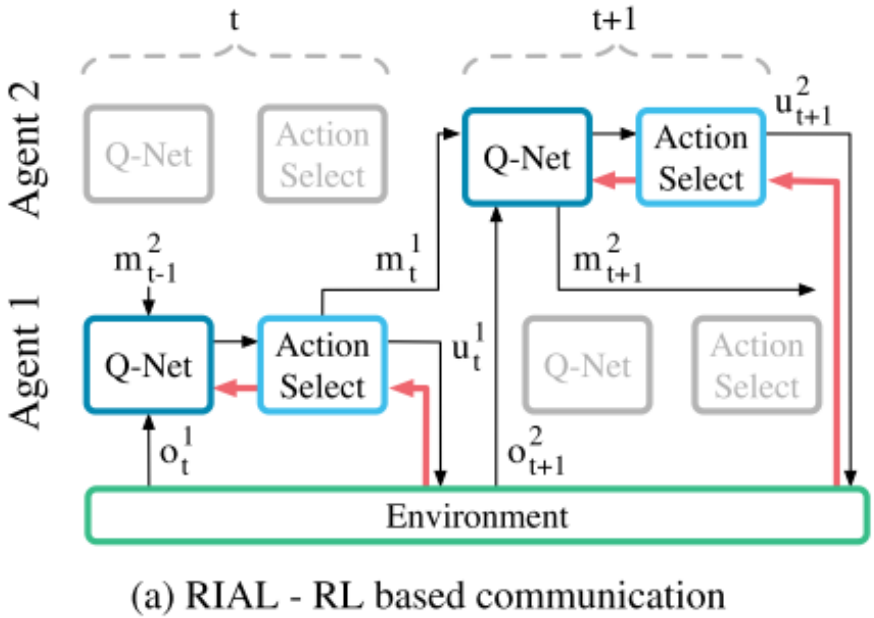
Reinforced Inter-Agent Learning (RIAL) is a first approach:

- 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.




(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:

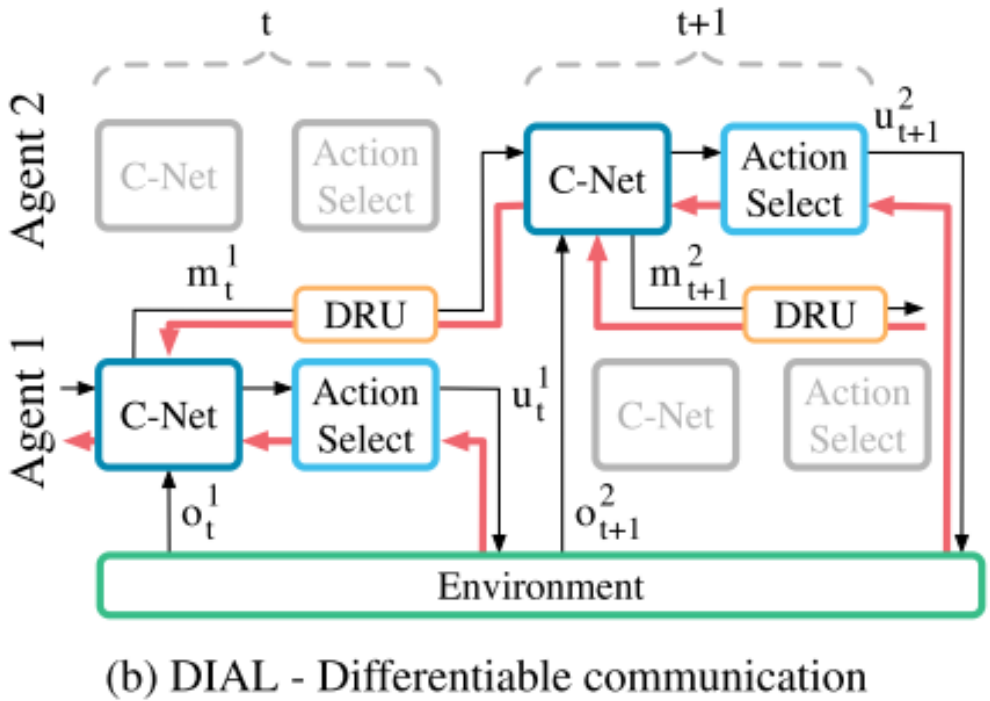
1. The $Q$ value, which is fed to the action selector.
2. A real-valued message.

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.

The neural network $f_\theta$:

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

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

# Competitive: AlphaGo Zero self-play

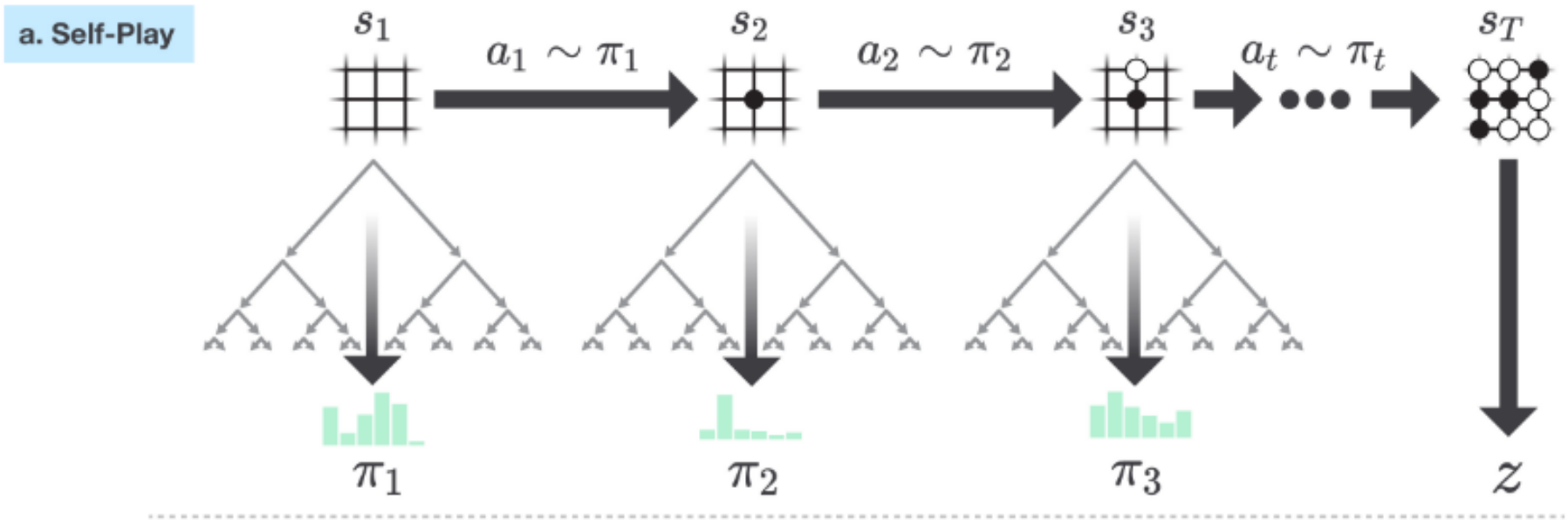



Figure: How an episode is conducted in AlphaGo Zero.

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, $\pi = \alpha_\theta(s) \propto N(s, a)^{1/\tau}$ ($\tau$ is a temperature parameter here).

# Competitive: AlphaGo Zero MCTS

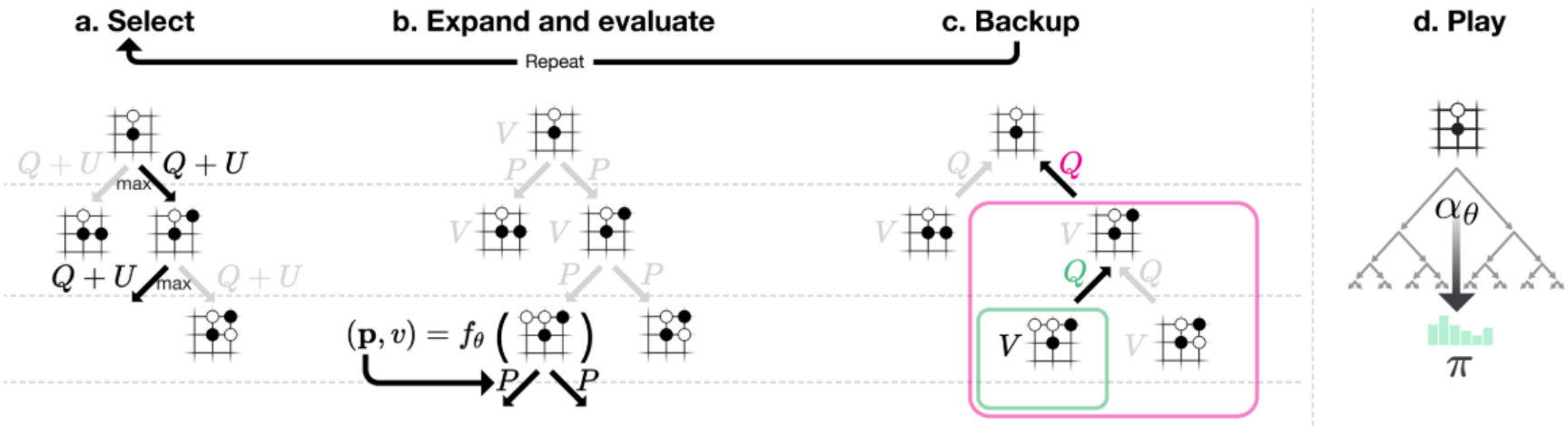



Figure: MCTS in AlphaGo Zero.

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

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

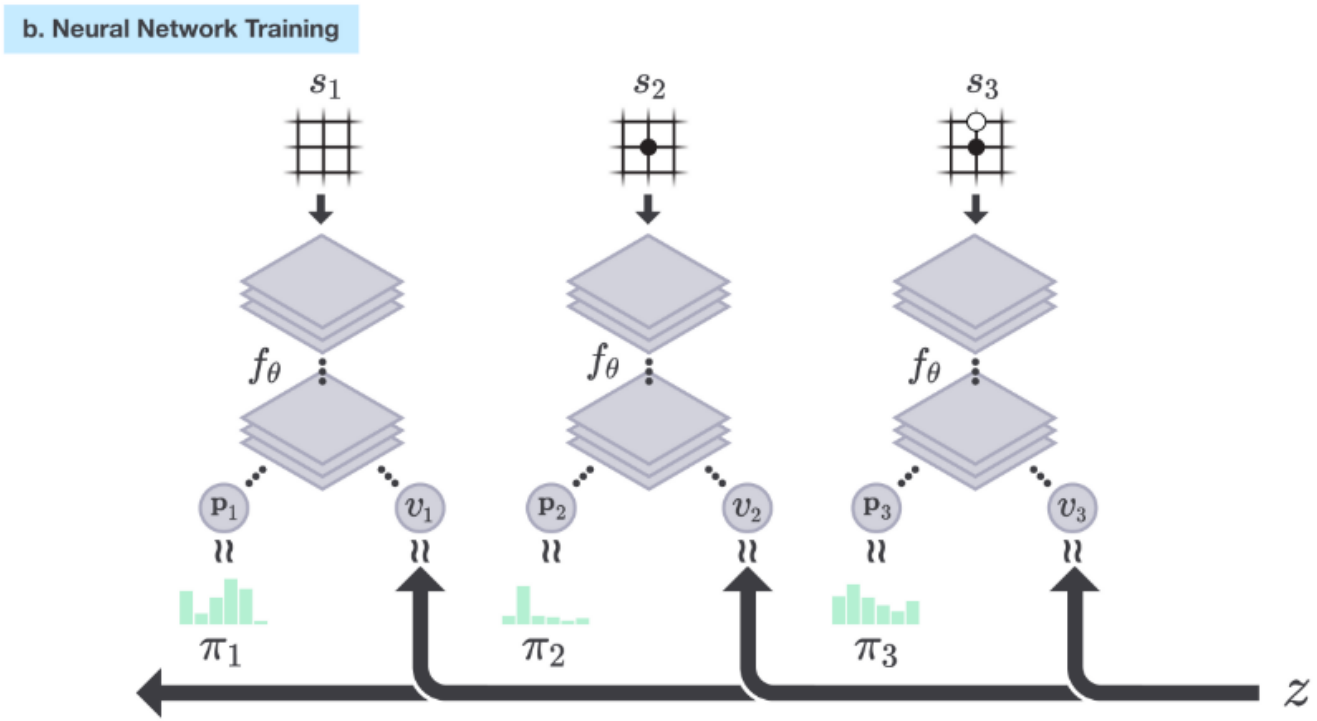



Figure: Training in AlphaGo Zero.

---


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.

# 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.
- Hide and Seek (2019): Emergent tool use from multi-agent autocurricula. (Presentation in INFO8004 - Advanced Machine Learning: `https://glouppe.github.io/info8004-advanced-machine-learning/pdf/pleroy-hide-and-seek.pdf`)

# Adversarial attacks

# Adversarial attacks

It is possible to trick neural network with small perturbations.

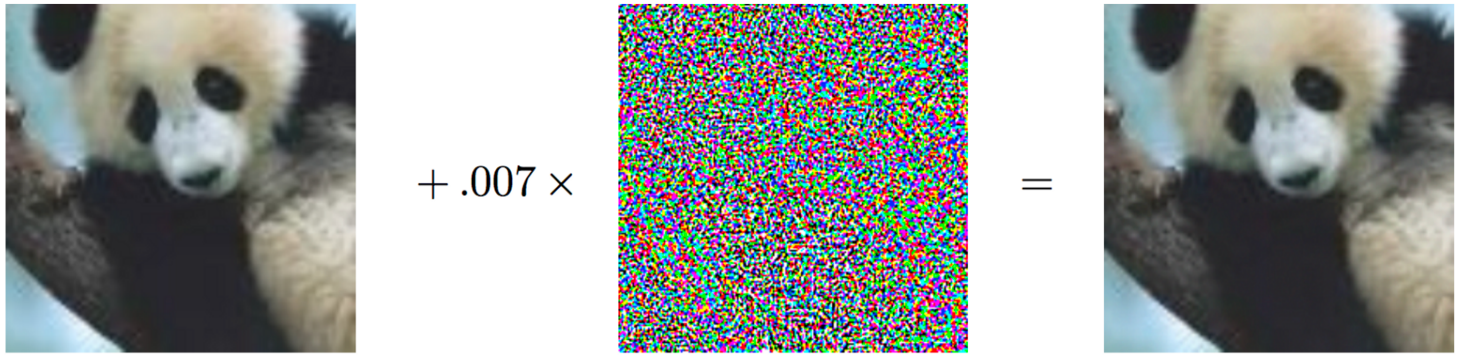



Figure: Adversarial attack in classification.

---

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 (trained between 680 and 1360 millions timesteps.
- By fixing the victim policy $\pi_\nu$, an adversarial policy $\pi_\alpha$ is trained.
- Note that the agent is now in a MDP since $\pi_\nu$ 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 observation:
  1. QMIX
  2. COMA
  3. QVMIX
- Communication:
  1. RIAL
  2. DIAL
- Competitive setting:
  1. Minimax-Q
  2. AlphaGo Zero
- Adversarial policies

# Advertisement

Internship at John Cockerill Defense to conduct a Master Thesis:

- Domain adaptation in classification (supervised learning).
- Multi-agent reinforcement learning.

Contact me pleroy[at]uliege.be to discuss about it!

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.