Cooperative Multi-agent Deep Reinforcement Learning in a 2 Versus 2 Free-Kick Task

Jim Martin Catacora Ocana(✉), Francesco Riccio, Roberto Capobianco, and Daniele Nardi

Department of Computer, Control and Management Engineering, Sapienza University of Rome, Via Ariosto 25, 00185 Rome, Italy
{catacora,riccio,capobianco,nardi}@diag.uniroma1.it

Abstract. In multi-robot reinforcement learning the goal is to enable a team of robots to learn a coordinated behavior from direct interaction with the environment. Here, we provide a comparison of the two main approaches to tackle this challenge, namely independent learners (IL) and joint-action learners (JAL). IL is suitable for highly scalable domains, but it faces non-stationarity issues. Whereas, JAL overcomes non-stationarity and can generate highly coordinated behaviors, but it presents scalability issues due to the increased size of the search space. We implement and evaluate these methods in a new multi-robot cooperative and adversarial soccer scenario, called 2 versus 2 free-kick task, where scalability issues affecting JAL are less relevant given the small number of learners. In this work, we implement and deploy these methodologies on a team of simulated NAO humanoid robots. We describe the implementation details of our scenario and show that both approaches are able to achieve satisfying solutions. Notably, we observe joint-action learners to have a better performance than independent learners in terms of success rate and quality of the learned policies. Finally, we discuss the results and provide conclusions based on our findings.

1 Introduction

Multi-agent reinforcement learning is concerned with the application of reinforcement learning (RL) techniques to situations having multiple agents learning at the same time in the same environment. Multi-agent reinforcement learning is important because it could provide solutions for challenging domains involving robot teams or robot swarms, see [2,6,9,11].

Two main MARL approaches have been proposed for handling multi-agent domains. In independent learners (IL), every agent performs standard RL, but in the presence of other agents. IL has the drawback that each individual sees the environment as non-stationary; and hence, guarantees of single-agent RL do not longer hold. Meanwhile, in joint-action learners (JAL), the state and action spaces of all agents are merged together, and a single policy is learned

S. Chalup et al. (Eds.): RoboCup 2019, LNAI 11531, pp. 44–57, 2019.
https://doi.org/10.1007/978-3-030-35699-6_4

that maps joint-observations to joint-actions. Any single-agent RL algorithm can readily be used to learn such joint-policy. JAL overcomes the non-stationarity problem, but it presents scalability issues as the joint state and action spaces can grow exponentially larger with the number of agents. Additionally, from a practical perspective, communication range/bandwidth and memory capacity could also limit JAL's scalability.

In this work, we implement IL and JAL and compare them on a robotic task. We consider a simplified version of soccer, referred to as 2 versus 2 offensive free-kick task. In soccer domains, the number of agents is normally small (up to eleven agents per team in a full soccer scenario); hence, it is conceivable for JAL to converge within a reasonable amount of time, contrary to domains with hundreds or more agents where JAL is impractical. JAL may still be slower than IL in such soccer scenarios, but it is also more likely to find better policies. Therefore, these special domains provide a motivating opportunity for investigating the tradeoff between optimality and convergence rate across MARL approaches.

A previous work, Hausknecht [3], compares parameter/memory sharing IL and JAL within a partially observable Robocup 2D domain, called 2-vs-1 half-field offense. This study reports that neither approach succeeds in the multi-agent sense. With IL, agents learn to unreliably score single-handedly; meanwhile, JAL finds no competent policies. By contrast, we manage to generate successful solutions grounded on cooperation for both approaches and within a similar domain. Furthermore, our results reveal that on average IL converges slightly faster than JAL; on the other hand, JAL discovers more synchronized strategies than IL, which may allow the attacking team to score goals within a shorter period of time. On top of that, JAL shows a higher success rate than IL, 60% versus 20% respectively, where the success rate indicates the percentage of runs in which a perfect scoring solution was found by the learning algorithm.

Summarizing, our contributions are: (1) we provide satisfactory results from IL and JAL in a new domain; namely, in the 2-vs-2 offensive free-kick task, (2) we compare the performances of IL and JAL, confirming that JAL should not be overlooked when facing analogous domains, and (3) we carry out experiments on a physically realistic 3D simulator, whereas, as far as we know, all previous studies concerning MARL in soccer domains involved 2D environments.

1.1 Markov Decisions Processes (MDPs)

MDPs [4] are a formalism for modeling sequential decision problems that concisely encode the interactions between agents and their environments. An MDP is a tuple $\langle S, A, T, R, \gamma \rangle$, comprising:

- The set S of all states in the world,
- The set A containing every action a learning agent can execute,
- A probability distribution over transitions $T : S \times A \times S \rightarrow [0, 1]$ from a current state to a next state given the action performed by the agent in the starting state,
- A reward function $R : S \times A \rightarrow \mathbb{R}$ that quantifies the immediate goodness of the action performed in the current state,

– And a discount factor γ that weights down the value of future rewards.

An MDP must satisfy the Markov property [16], such that the next state and expected next reward depend, in general, probabilistically only on the state and action representations at the current time step, which means that the following equality holds true for all r and s':

$$\Pr\{R_{t+1}{=}r, S_{t+1}{=}s' | S_t, A_t\} = \Pr\{R_{t+1}{=}r, S_{t+1}{=}s' | S_0, A_0, R_0, ..., S_t, A_t\} \quad (1)$$

Given an MDP, the goal is to find a policy π, i.e. a generally stochastic mapping from states to actions that fully encodes the decision-making behavior of an agent.

1.2 Markov Games

Markov Games [14], also called Stochastic Games, are the straightforward extension of MDPs to situations where there are multiple agents capable of making decisions in the same environment. In a Markov Game, all agents perceive the complete state of the world, i.e. they have full observability. In addition, in the general case, each agent has a distinct reward function, but for strictly cooperative tasks the entire team can be granted a single reward. The general formulation of a Markov Game is a tuple $\langle S, \mathcal{A}, T, R, \gamma \rangle$, comprising:

– The set S of all states in the world,
– The collection $\mathcal{A}$ of action sets pertaining to each agent $A^1, ..., A^k$,
– A probability distribution over transitions $T : S \times \mathcal{A} \times S \rightarrow [0, 1]$ from a current state to a next state given the joint-action performed simultaneously by all agents in the starting state,
– A reward function, possibly different for each agent i, $R^i : S \times A^1 \times ... \times A^k \rightarrow \mathbb{R}$ that quantifies the immediate goodness of the joint-action performed by the agents in the current state,
– And a discount factor γ that weights down the value of future rewards.

1.3 Reinforcement Learning (RL)

Reinforcement Learning is concerned with learning in MDPs without having an a priori model of the domain (transition and reward functions). The main characteristics of RL is that learning is accomplished through raw experience by executing actions and receiving rewards; and that learning is synonymous with the maximization of the sum of rewards collected over time [16]. The core objective of RL is to find the optimal policy π^* that yields the highest sum of rewards, which is usually defined in terms of the cumulative discounted reward. Formally, if an agent with policy π is at state S_t after t timesteps, and thereafter it receives a sequence of future rewards of $\{R_{t+1}, R_{t+2}, ...\}$, then the expected cumulative discounted reward G_t is expressed as:

$$E_\pi[G_t | S_t = s] = E_\pi\Big[\sum_{k=0} \gamma^k R_{t+k+1} | S_t = s\Big] \quad (2)$$

The term $E_\pi[G_t|S_t = s]$ is also called the value of s under π. A function $v_\pi(s)$ that encodes this information for every state in S is known as a value function. This definition can be extended to every state-action pair in $S \times A$, yielding an action-value function, also called a q-function, which is formalized as:

$$q_\pi(s,a) = E_\pi\Big[\sum_{k=0} \gamma^k R_{t+k+1} | S_t = s, A_t = a\Big] \tag{3}$$

1.4 Deep Reinforcement Learning

Deep RL relies on deep neural networks for representing the value functions or policies that are learned. These networks are trained through some efficient variant of stochastic gradient descent (SGD), which minimizes a loss function that measures the difference between the current predictions of the network and the target values it must approximate to. For example, for a network with parameters $\boldsymbol{\theta}$, prediction vector $\mathbf{y}_\theta$ and target vector $\mathbf{d}$, the L2-loss function is defined as:

$$\mathcal{L}(\mathbf{d}, \mathbf{y}_\theta) = (\mathbf{d} - \mathbf{y}_\theta)^T \cdot (\mathbf{d} - \mathbf{y}_\theta) \tag{4}$$

The update rule of SGD given a learning rate α and a uniformly random perturbation $\mathbf{e}_t$ is then computed as:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha \nabla_\theta \mathcal{L}(\mathbf{d}, \mathbf{y}_\theta)|_{\boldsymbol{\theta}_t} + \mathbf{e}_t \tag{5}$$

Deep Deterministic Policy Gradient (DDPG) was developed by Lillicrap et al. [8]. It employs two separate deep neural networks, denoted as critic and actor. The critic learns a parametrized q-function (Q_θ). Meanwhile, the actor learns and executes a parametrized policy (π_θ). During training, at each time step t the learning agent collects $[S_t, A_t, R_{t+1}, S_{t+1}]$. Naïve updates for the critic and actor can be computed as follows. In the case of the critic, we feed the network with the most recent state-action pair (S_t, A_t) to obtain a prediction $Q_\theta(S_t, A_t)$, then we set the corresponding target (see Eq. (6)), by computing an approximation through temporal differences [15] and relying on the current actor network in order to estimate the next action $A_{t+1} = \pi_\theta(S_{t+1})$.

$$R_{t+1} + \gamma Q_\theta(S_{t+1}, \pi_\theta(S_{t+1})) \tag{6}$$

When training the actor, we take advantage of the gradients of the critic with respect to its action inputs [1] to set the following direction of improvement, which is then used to update the actor's parameters as in Eq. (5):

$$\nabla_a Q_\theta(S_t, a = \pi_\theta(S_t)) \nabla_\theta \pi_\theta(S_t) \tag{7}$$

Stable DDPG. Naïve updates will lead to instabilities during training. Two extra mechanisms proposed by Mnih et al. [10] need to be enforced to overcome this problem.

First of all, SGD requires that samples are independent and identically distributed. To break the ordering of the collected samples, an experience replay buffer is introduced that gathers samples $[S_t, A_t, R_{t+1}, S_{t+1}]$ from several episodes. Afterwards, off-policy updates are performed by drawing a batch of random experiences from this buffer.

Second, in Eq. (6) targets depend on the current critic and actor networks. This could lead to learning oscillations and divergence, since the targets themselves will also change after each update. To deal with this issue, two target networks Q'_θ and π'_θ are introduced. Their architectures are identical to the critic and actor networks but their parameters are updated less frequently. Given these target networks, the critic's targets are computed as:

$$R_{t+1} + \gamma Q'_\theta(S_{t+1}, \pi'_\theta(S_{t+1})) \tag{8}$$

1.5 Independent Learners Approach

In independent learners, each agent applies RL on its own. There is no direct communication among agents. Furthermore, agents are not explicitly aware of the actions executed by other learners. Because of this, from the perspective of each individual the world appears non-stationary, i.e. the reward signal each agent receives depends on hidden, random and time-varying variables, which are the unknown policies of other agents. As a result, directly applying single-agent RL techniques is no longer guaranteed to produce good results.

One approach to diminish the negative effects of non-stationarity is homogeneous learners, applicable only when all agents have identical state and action spaces. In this scenario, it is possible to reduce the problem to one in which we learn a single shared q-function or policy from the combined experiences of all agents. Previous works, such as [5] and [7], report satisfactory results from such approach.

1.6 Joint-Action Learners Approach

The key idea of JAL is to reformulate the learning problem such that instead of having as many independent MDPs as the number of agents, only a single Markov Game is defined over the joint-action space corresponding to the group of agents. Moreover, when a single reward is handed to the team of agents in cooperative tasks, the Markov Game formulation can effectively be treated as a single-agent problem; completely avoiding non-stationarity.

On the downside, JAL presents scalability issues, as the joint-state and joint-action spaces as well as the size of any parametrized structure must increase. Notoriously, this growth is only linear for approximate methods that explicitly learn a policy, such as DDPG. However, even a linear growth can be quite restrictive compared to the constant complexity of IL for domains involving hundreds of agents or more.

2 Evaluation

2.1 Implementation

Task Specification. Experiments were carried out on the 2 vs. 2 offensive free-kick task. It involves an offensive team (2 attackers) and a defending team (defender and goalkeeper). The game takes place in the half-field belonging to the latter team. An episode starts with the offensive team being granted a free kick. From the beginning of a match and until an attacker makes contact with the ball, the players of the defending team are forbidden to remain less than 0.75 m away from the ball. The attackers' goal is to score a goal within a time limit and without losing control of the ball.

In this work, we made use of a RoboCup Standard Platform League [12] simulator, called the B-Human framework [13]. This framework allows teams of NAO humanoid robots to compete against each other in a physically grounded and fairly realistic virtual environment. Therefore, the offensive free-kick task is played by four NAO robots with identical sensing and actuation capabilities.

World Settings. Figure 1 shows a random initialization of the task. At the beginning of each episode, the ball is placed at position $[1.0, 2.0]$ m. The defender is positioned 0.75 m away from the ball and in between the ball and the center of the goal line; hence, it blocks a direct kick to goal. The goalkeeper is placed at the center of the goal line.

Moreover, one attacker is initially placed 0.3 m from the ball and according to the angular variable β with range $[0, \pi/4]$ rad. The other attacker is fixed to -2.5 m on the y-axis, whereas its position on the x-axis varies uniformly at random within $[1.0, 2.0]$ m at each initialization.

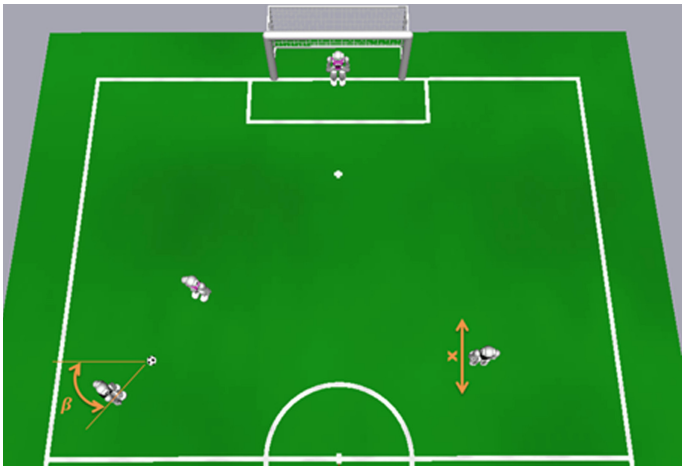


Fig. 1. Random initial world state.

The defensive players follow handcrafted policies. The defender's policy considers two conditions: (1) before the ball is touched, it stands in-place, and (2) after the ball is touched, it approaches it as fast as it can. The goalkeeper is only allowed to move from post to post along the goal line. It moves towards the intersection of the goal line and the ball's velocity vector or until it reaches a post, as long as the ball moves forward.

An episode terminates if:

- The offensive team has scored a goal;
- a sufficient time limit of 20 s has expired;
- one player of the defending team has made contact with the ball;
- the ball has left the field.

MDP Formulation for IL. Since both attackers are identical, an homogeneous learners strategy is enforced for IL; hence, effectively only one MDP is formulated. From the point of view of each agent, its associated MDP comprises an 18-dimensional state vector, consisting of the following high-level information:

- The agent's absolute pose, $[x^A, y^A, \theta^A] \in \mathbb{R}^3$, which encodes the current Cartesian position $\mathbf{p}^A = [x^A, y^A]$ and 2D orientation of the agent;
- the absolute pose of its teammate, $[x^T, y^T, \theta^T] \in \mathbb{R}^3$, which likewise encodes position $\mathbf{p}^T = [x^T, y^T]$;
- the ball's absolute position, $\mathbf{p}^B = [x^B, y^B] \in \mathbb{R}^2$;
- the ball's absolute velocity, $[v_x^B, v_x^B] \in \mathbb{R}^2$;
- the defender's absolute position, $[x^D, y^D] \in \mathbb{R}^2$;
- the goalkeeper's spot over the goal line, $y^G \in \mathbb{R}$, which indicates a location on the y-axis of the world's frame;
- a 5-bit timestamp, $[t^4, t^3, t^2, t^1, t^0] \in \{0, 1\}^5$, representing the current time step of the episode. The timer increases by one step every 1 s of game play.

This timestamp is essential to satisfy the Markov Property, as otherwise, the next state could either be a new configuration of players and ball or the failure state depending on the time remaining.

In order to perform fast simulations, location information is gathered directly from the framework's oracle. Nevertheless, the B-human framework also provides the functionality to preprocess the same high-level information from images, considering a simulated camera located over the field, such that full observability is achieved as well.

Each agent is permitted to execute only one of two macro actions, namely: walk and kick, which come as pre-built functions with the B-Human framework. The walk function requires three arguments (relative Cartesian and rotational velocities), it loops indefinitely step after step, taking a robot approximately 0.5 s to make a single step. Wrapped around this function, we declare the walk action to be a sequence of a couple of steps and we implement it by forcing termination of the walk function after 1 s according to a clock.

On the other hand, the kick function lasts a bit less than 5 s and requires no extra arguments (an agent always kicks with the same leg). We define the

kick action given the kick function plus some idle time such that it lasts exactly 5 s. The action space of the MDP is represented by a 5-dimensional real-valued vector, composed of:

- Two selectors in the range $[0.0, 1.0]$. The agent executes the action whose associated selector is the highest.
- Three walking arguments taking values in $[-1.0, 1.0]$.

It is worth mentioning that each agent receives observations and makes decisions only during the first time step of an episode and time steps immediately after a robot terminates an action (time instances when the walk and kick functions release control back to the main execution), all other intermediate time steps are ignored by the agent. Thus, when the kick action is called, the agent then waits 5 s (time steps) before making its next decision.

As the free-kick task is fairly complex, being unlikely that the attacking team scores as a result of random actions, and since we choose to use algorithms that are not optimized for handling sparse rewards, in order for learning to happen it was necessary to design a shaped reward function that reinforces the agents in a continuous manner for getting closer to scoring a goal.

Therefore, at each time step t, both agents are independently given shaped rewards based on the same function, which is shown below. Said function is similar to the one used in [3], except that we add the term D_B^T. In the following equations, $\mathbf{p}$ corresponds to a 2D position with components x and y; furthermore, scripts A, T, B, L, R and P refer to the agent, its teammate, the ball, the left goal post, the right goal post and either post, respectively.

$$R = D_A^B + max(D_B^T, D_B^G) + G \tag{9}$$

Where:

$$D_A^B = d(\mathbf{p}_t^B, \mathbf{p}_{t-1}^A) - d(\mathbf{p}_t^B, \mathbf{p}_t^A) \tag{10}$$

$$D_B^T = \begin{cases} \text{if the agent pushed the ball, } d(\mathbf{p}_{t-1}^B, \mathbf{p}_t^T) - d(\mathbf{p}_t^B, \mathbf{p}_t^T) \\ \text{else,} \qquad\qquad 0 \end{cases} \tag{11}$$

$$D_B^G = \begin{cases} \text{if the agent pushed the ball, } d_{goal}(\mathbf{p}_{t-1}^B) - d_{goal}(\mathbf{p}_t^B) \\ \text{else,} \qquad\qquad 0 \end{cases} \tag{12}$$

$$G = \begin{cases} \text{if goal, } +20 \\ \text{else,} \quad\ 0 \end{cases} \tag{13}$$

$$d(\mathbf{p}, \mathbf{q}) = \sqrt{(x_p - x_q)^2 + (y_p - y_q)^2} \tag{14}$$

$$d_{goal}(\mathbf{p}^B) = \begin{cases} \text{if } y^B \geq y^L, \ d(\mathbf{p}^B, \mathbf{p}^L) \\ \text{if } y^B \leq y^R, \ d(\mathbf{p}^B, \mathbf{p}^R) \\ \text{else,} \qquad |x^B - x^P| \end{cases} \tag{15}$$

This reward function integrates the following terms:

- D_A^B: Rewards an agent for approaching the ball.

- D_B^T: Promotes cooperation, since it incentivizes an agent for passing the ball to a teammate.
- D_B^G: Rewards an agent for hitting the ball to goal.
- G: Assigns a large positive reward to all agents of the attacking team when a goal is scored, and assigns zero otherwise.

Markov Game Formulation for JAL. Since full observability is assumed in both formulations, the state space of the Markov Game associated with the free-kick task is also 18-dimensional, embedding the same components as the previous MDP. Because a parametrized policy will be learned, the joint-action space (10D) is constructed simply by concatenating each agent's 5-dimensional action vector, as presented in the precedent sub-section. Likewise, the team makes decisions only during the first time step and time steps following the culmination of an action executed by anyone of the attackers.

For instance, if during time step 0 the team decides one attacker to walk and the other to kick, then, the next team decision will be made during time step 1, but it will only have a meaningful effect over the next action performed by the first attacker as the second one still has to complete the kick action. Assuming that the next action of the first attacker is to kick, then the next team decision will occur during time step 5 (skipping time steps from 2 to 4), when the next action to be executed by the second attacker will have to be selected.

A shaped reward is also considered for JAL, for the same reasons as previously stated. In this case, the offensive team as a whole is rewarded given the following function that employs the notation found in the previous sub-section, but introduces subscript A_i denoting the i-th agent in the attacking team.

$$R = \sum_i D_{A_i}^B + max(\{D_B^{A_i}, \forall i\}, D_B^G) + G \tag{16}$$

Where:

$$D_{A_i}^B = d(\mathbf{p}_t^B, \mathbf{p}_{t-1}^{A_i}) - d(\mathbf{p}_t^B, \mathbf{p}_t^{A_i}) \tag{17}$$

$$D_B^{A_i} = d(\mathbf{p}_{t-1}^B, \mathbf{p}_t^{A_i}) - d(\mathbf{p}_t^B, \mathbf{p}_t^{A_i}) \tag{18}$$

$$D_B^G = d_{goal}(\mathbf{p}_{t-1}^B) - d_{goal}(\mathbf{p}_t^B) \tag{19}$$

The G term and functions $d(\mathbf{p}, \mathbf{q})$, $d_{goal}(\mathbf{p}^B)$ are computed according to Eqs. (13), (14) and (15), respectively. The terms of this function convey the following information:

- $\sum_i D_{A_i}^B$: Rewards the team for every attacker that approaches the ball.
- $\{D_B^{A_i}, \forall i\}$: Rewards the team whenever the ball is passed to any member.
- D_B^G: Rewards the team if any agent shoots to goal.
- G: Assigns a large positive reward to the attacking team when a goal is scored, and assigns zero otherwise.

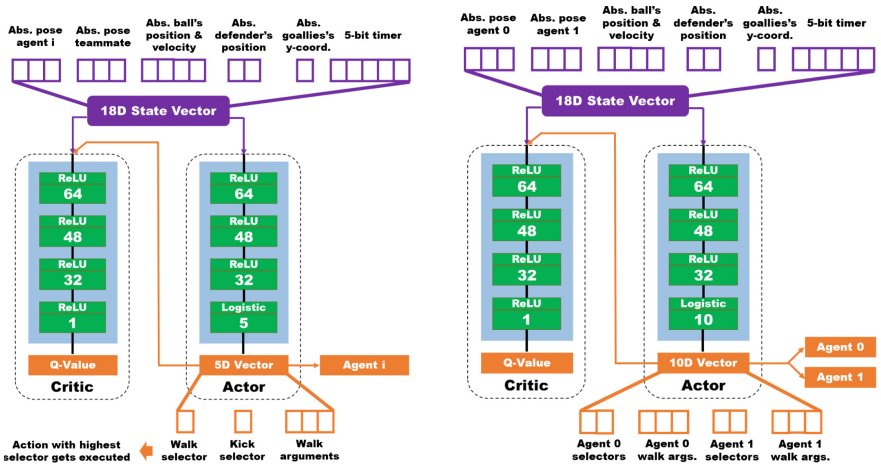


Fig. 2. Neural architectures. On the left, the architecture for independent learners; while on the right, the architecture for joint-action learners.

Deep RL Settings. The DDPG algorithm is used in IL as well as in JAL. Accordingly, four separate deep neural networks are maintained during training: critic, actor, target critic and target actor.

All four networks are represented by dense networks with 3 hidden layers (Fig. 2). This choice was taken because the offensive free-kick task has a low dimensional state and it does not require a memory of past states to be solved; thus, convolutional or recurrent networks are not necessary. Critics receive as input the concatenation of state and action vectors of the corresponding MDP (18D + 5D) or Markov Game (18D + 10D) and predict a single Q-value. Actors are fed only state vectors (18D) and produce an output of size equal to the cardinality of the action (5D) or joint-action (10D) vector corresponding to IL or JAL.

An ϵ-greedy strategy is followed during training. By setting $\epsilon = 0.5$ and fixing this value for the entire run, the action that will be executed by the agent or the team will agree exactly with the output of the corresponding actor half the time; and the other half, macro actions will be selected uniformly at random, while a small uniform noise of ± 0.05 will be added to the walking velocities initially determined by said actor.

After each iteration (single match/episode), all networks are updated once given a mini-batch of experiences, randomly selected from the replay buffer. Training of the critic and actor relies on the L2-loss and Adam optimizer. In turn, target networks are updated smoothly based on the recently-updated parameters of their references (represented as vectors $\boldsymbol{\theta}^Q$ and $\boldsymbol{\theta}^\pi$) and on the scalar hyper-parameter τ, as follows:

$$\boldsymbol{\theta}'^Q_{t+1} = \tau \boldsymbol{\theta}^Q_{t+1} + (1 - \tau)\boldsymbol{\theta}'^Q_t \tag{20}$$

$$\boldsymbol{\theta}'^{\pi}_{t+1} = \tau\boldsymbol{\theta}^{\pi}_{t+1} + (1-\tau)\boldsymbol{\theta}'^{\pi}_{t} \tag{21}$$

Hyper-parameters of DDPG are set as shown in Table 1:

Table 1. DDPG hyper-parameter setting

Parameter	Approach	Symbol	Value
Hidden units/layer critic/actor	Both	-	64,48,32 (RELU)
Output units in critic	Both	–	1 (linear)
Output units in actor	IL	–	5 (logistic)
Output units in actor	JAL	–	10 (logistic)
Size of replay buffer	Both	–	100000
Training batch size	Both	–	4000
ϵ-greedy control parameter	Both	ϵ	0.5
Discount rate	Both	γ	0.9
Learning rate critic/actor	Both	α	0.001
Update rate target networks	Both	τ	0.01

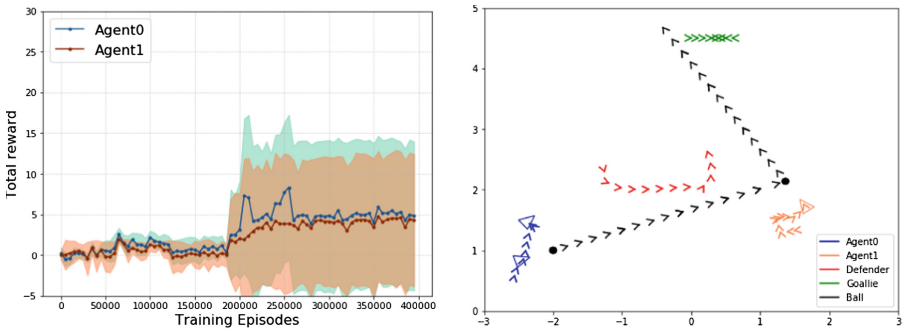
2.2 Results

IL and JAL were executed 10 runs each, over the offensive free-kick task. Each run continued for 400K iterations, i.e. 20-s matches. Policies were validated on 50 randomly initialized episodes after every 200 iterations, where the ϵ hyper-parameter is set to zero.

Independent Learners. IL achieved successful team strategies in two out of ten runs. In the remaining runs, IL scored zero goals at every validation step. Figure 3(a) shows that initially IL gets stuck in a bad local minimum, but it eventually manages to converge to a satisfactory policy after 200K iterations on average. In this local minimum, the attacker further away from the ball, only approaches the ball but never kicks it; likely because it is seldom close enough to the ball to perform a kick with positive effects.

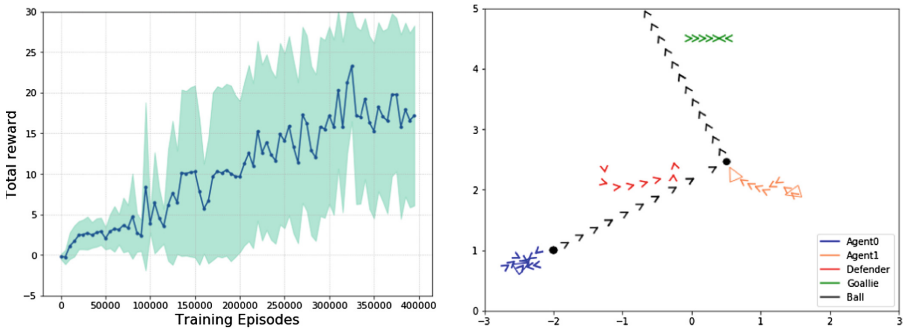
Figure 3(b) reveals the typical behavior of a successful strategy attained by IL. In particular, it shows the trajectories followed by attackers, defenders and ball at different time steps, where the chevrons' apices point to the direction of motion, triangles denote a kick action and circles indicate ball at rest. We can notice that the attacker that receives the ball displays a reactive behavior, i.e. it does not know where to go until its teammate makes a pass; as a result, it spends extra time readjusting and searching for the ball.

Joint-Action Learners. JAL accomplished a perfect goal percentage in validation in six out of ten runs. Another two ended with percentages of 0.52 and 0.78, and the last two stayed at zero the entire time. Figure 4(a) reveals that in



(a) (b)

Fig. 3. Independent Learners. On the left, the average total reward per episode; while on the right, an execution of the learned policy.



(a) (b)

Fig. 4. Joint-Action Learners. On the left, the average total reward per episode; while on the right, an execution of the learned policy.

some cases JAL suffers from the same local minimum encountered by IL. However, in other cases, JAL can discover effective policies quickly, but it does not truly converge until several steps later (after 300K iterations on average).

Figure 4(b) shows the response of a successful policy usually derived by JAL. It can be observed that when executing a pass, there is a beforehand understanding between the agents about where the ball will be directed, such that the receiver moves straight to intersect the ball without wasting much time. Needless to say, this strategy shows more coordination than the one learned by IL.

2.3 Discussion

By comparing the results obtained from IL and JAL, it is evident that the latter approach outperforms the former. JAL learned well-performing strategies much more consistently (60% of runs) than IL (20% of runs). In addition, final policies

derived from JAL revealed a higher degree of coordination between agents, which learned to pass the ball with a more precise timing among them. Not even in terms of convergence rate was IL able to surpass JAL, since they both converged after a comparable number of iterations, which is a direct consequence of having full observability.

Notwithstanding, the performance of both approaches was far from ideal. First of all, both methods suffered from bad local minima partly caused by the use of shaped rewards, which gave rise to unwanted behaviors such as: dribbling to goal instead of shooting, shooting instead of passing or passing instead of shooting. This hints that there is a need for dismissing shaped rewards altogether, relying instead in new techniques for promoting exploration in sparse settings.

A second shortcoming is that DDPG as well as most others basic Deep RL algorithms are extremely sample-inefficient. Certainly, there are several sources of inefficiency; yet, the most fundamental is the lack of generalization throughout the state-action space. Deep networks indeed provide some generalization, but it is limited to a small neighborhood around a given point in the space. Therefore, basic algorithms still require examples gathered from all around the underlying space in order to learn an overarching policy.

Specifically, they do not exploit world symmetries, such that an optimal action learned for one small region would be instantly replicated to other regions, which are equivalent under some transformation, such as a translation, rotation or reflection. That would be a game-changer in robotic domains. Moreover, their learning is state-action specific instead of abstract, i.e. their goal is to discover an optimal action for every patch of state space, instead of learning universal rules such as "move in the direction of the ball until reaching it".

3 Conclusions

This work achieved successful implementations of both independent learners and joint-action learners in the 2-vs-2 offensive free-kick task and within a complex 3D simulator.

Joint-action learners were clearly superior to independent learners. They both converged after a comparable number of iterations. However, JAL discovered good strategies more consistently than IL; furthermore, policies found by JAL reveal a greater inter-agent coordination than those found by IL. Hence, we conclude that for similar robotic domains JAL constitutes a MARL alternative that should not be ignored.

Despite its effectiveness in discovering a solution, the learning procedure was quite inefficient. It suffered from bad local minima; in addition, there were no mechanisms for exploiting symmetries or abstractions found in the world in order to make use of fewer training examples. These issues will be addressed in future work.

References

1. Hafner, R., Riedmiller, M.: Reinforcement learning in feedback control. Mach. Learn. **84**(1–2), 137–169 (2011)
2. Haksar, R.N., Schwager, M.: Distributed deep reinforcement learning for fighting forest fires with a network of aerial robots. In: IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, pp. 1067–1074 (2018)
3. Hausknecht, M.J.: Cooperation and communication in multiagent deep reinforcement learning. Ph.D. thesis, University of Texas at Austin, USA (2016)
4. Howard, R.A.: Dynamic Programming and Markov Processes. MIT Press, Cambridge (1960)
5. Kalyanakrishnan, S., Stone, P.: Learning complementary multiagent behaviors: a case study. In: Baltes, J., Lagoudakis, M.G., Naruse, T., Ghidary, S.S. (eds.) RoboCup 2009. LNCS (LNAI), vol. 5949, pp. 153–165. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-11876-0_14
6. Knopp, M., Aykın, C., Feldmaier, J., Shen, H.: Formation control using GQ(λ) reinforcement learning. In: 26th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN), pp. 1043–1048, August 2017
7. Kurek, M.: Deep reinforcement learning in keepaway soccer. Master's thesis, Poznan University of Technology, Poland (2015)
8. Lillicrap, T.P., et al.: Continuous control with deep reinforcement learning. CoRR abs/1509.02971 (2015). http://arxiv.org/abs/1509.02971
9. Liu, Y., Nejat, G.: Multirobot cooperative learning for semiautonomous control in urban search and rescue applications. J. Field Rob. **33**(4), 512–536 (2016)
10. Mnih, V., et al.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529–533 (2015)
11. Pham, H.X., La, H.M., Feil-Seifer, D., Nguyen, L.V.: Cooperative and distributed reinforcement learning of drones for field coverage. CoRR abs/1803.07250 (2018). http://arxiv.org/abs/1803.07250
12. RoboCup Technical Committee: Robocup standard platform league (nao) rule book. Rules2018.pdf, August 2018. https://spl.robocup.org/downloads/
13. Röfer, T., et al.: B-human: team report and code release 2017. Technical report, Deutsches Forschungszentrum für Künstliche Intelligenz, Universität Bremen (2017)
14. Shapley, L.S.: Stochastic games. Proc. Natl. Acad. Sci. **39**(10), 1095–1100 (1953)
15. Sutton, R.S.: Learning to predict by the methods of temporal differences. Mach. Learn. **3**, 9–44 (1988)
16. Sutton, R.S., Barto, A.G.: Introduction to Reinforcement Learning, 1st edn. MIT Press, Cambridge (1998)