Multi-agent Double Deep Q-Networks

David Simões[1,2,3(✉)], Nuno Lau[1,3], and Luís Paulo Reis[1,2,4]

[1] IEETA - Institute of Electronics and Informatics Engineering of Aveiro, University of Aveiro, Aveiro, Portugal
[2] LIACC - Artificial Intelligence and Computer Science Lab, Porto, Portugal
[3] DETI/UA - Electronics, Telecommunications and Informatics Department, University of Aveiro, Aveiro, Portugal
`david.simoes@ua.pt`
[4] DSI/EEUM - Information Systems Department - School of Engineering, University of Minho, Braga, Portugal

Abstract. There are many open issues and challenges in the multi-agent reward-based learning field. Theoretical convergence guarantees are lost, and the complexity of the action-space is also exponential to the amount of agents calculating their optimal joint-action. Function approximators, such as deep neural networks, have successfully been used in single-agent environments with high dimensional state-spaces. We propose the Multi-agent Double Deep Q-Networks algorithm, an extension of Deep Q-Networks to the multi-agent paradigm. Two common techniques of multi-agent Q-learning are used to formally describe our proposal, and are tested in a Foraging Task and a Pursuit Game. We also demonstrate how they can generalize to similar tasks and to larger teams, due to the strength of deep-learning techniques, and their viability for transfer learning approaches. With only a small fraction of the initial task's training, we adapt to longer tasks, and we accelerate the task completion by increasing the team size, thus empirically demonstrating a solution to the complexity issues of the multi-agent field.

1 Introduction

In recent years, there have been many successes using deep representations in reinforcement learning. One such example is Deep Q-Networks (DQN) [12], a reinforcement learning algorithm that allowed a single agent to achieve human-level performance across many Atari games. However, in most situations, agents interact with other agents in order to solve a given problem. Multi-agent Systems research focuses on building a system with multiple independent agents, and how to coordinate them [16]. We propose to extend the DQN algorithm to the multi-agent scenario and describe two variants for cooperative scenarios. We show how these variants behave in different tasks and how they can generalize to similar tasks, with more agents or more goals to accomplish.

The remainder of this article is structured as follows: Sect. 2 introduces Markov Decision Processes, Q-learning, Double Deep Q-Networks, and the multi-agent reward-based learning paradigm. Section 3 describes similar work in the

E. Oliveira et al. (Eds.): EPIA 2017, LNAI 10423, pp. 123–134, 2017.
DOI: 10.1007/978-3-319-65340-2_11

field, and Sect. 4 describes our proposed algorithms. Section 5 shows the results obtained, and Sect. 6 draws our conclusions and lists future research options.

2 Background

In single-agent reinforcement learning, the environment of an agent is commonly modeled by a Markov Decision Process (MDP). A Decentralized Partially-Observable Markov Decision Process (Dec-POMDP) is a multi-agent generalization of a MDP. It is defined by the tuple $\mathcal{E} = (\mathcal{S}, \{\mathcal{A}_i^n\}, F, \{\mathcal{O}_i^n\}, O, R)$, where n is the number of agents, $\mathcal{S} \in \mathbb{N}^d$ is a finite set of possible d-dimensional states of the environment $\mathcal{E}$, $\mathcal{A}_i, i = 1, \ldots, n$ is a set of possible actions for agent i, $\mathcal{A}$ is the joint-action set $\mathcal{A} = \mathcal{A}_1 \times \ldots \times \mathcal{A}_n$, $F : \mathcal{S} * \mathcal{A} * \mathcal{S} \rightarrow [0,1]$ is a state transition probability function $F(s'|s,a)$ for transitioning to state s' after taking joint-action a in state s, $\mathcal{O}_i, i = 1, \ldots, n$ is the set of observations for agent i, $\mathcal{O}$ is the joint-observation set $\mathcal{O} = \mathcal{O}_1 \times \ldots \times \mathcal{O}_n$, $O : \mathcal{S} * \mathcal{A} * \mathcal{O} \rightarrow [0,1]$ is a state observation probability function $O(o|s,a)$ for observing o after taking joint-action a in state s, and $R : \mathcal{S} * \mathcal{A} * \mathcal{S} \rightarrow \mathbb{R}$ is the associated reward function of all agents. The state transitions and observations are the result of the joint-action $a_k \in \mathcal{A}$. The joint policy π is gathered from the policies $\pi_i : \mathcal{S} * \mathcal{A}_i \rightarrow [0,1], i = 1, \ldots, n$. At time-step t, agent i observes observation $o_{t,i} \in \mathcal{O}_i$ from the environment and the reward r_t. In Dec-MDP, the state can be computed from the set of agent observations $\{\mathcal{O}_0, \ldots, \mathcal{O}_n\}$, i.e., $O(o|s,a) = \{0,1\}$.

Q-learning [19] is a model-free MDP-based algorithm, from which many multi-agent reinforcement learning algorithms are derived [2]. The action-value function, also known as Q-function, $Q_\pi : \mathcal{S} * \mathcal{A} \rightarrow \mathbb{R}$, is the expected return of policy π based on the current state and action performed. A discount factor $\gamma \in [0,1[$ represents the importance of future rewards. Discount factors closer to 1 increase the amount of fore-planning the agent takes into account. The optimal Q-function is defined as $Q^*(s,a) = \max_\pi \ Q^\pi(s,a)$. The agent determines Q^* and then uses a greedy policy to choose its actions, which is optimal when applied to Q^*. Q-learning is an iterative approximation procedure, where the current estimate of Q^* is updated using samples collected as rewards from actual experiences with the task, i.e.,

$$Q_{t+1}(s_t, a_t) = (1 - \eta_t)\{Q_t(s_t, a_t)\} + \eta_t\{r_{t+1} + \gamma \max_{a'} \ Q_t(s_{t+1}, a')\}, \tag{1}$$

where $\eta_t \in [0,1]$ represents a learning rate. In single-agent systems, the Q-function provably converges to Q^* as long as the agent tries all actions in all states with non-zero probability (usually by using an ϵ-greedy exploration method). In multi-agent systems which are not fully cooperative, due to the non-stationarity of the environment and the conflict of interests between agents, such theoretical guarantees are lost.

2.1 Double Deep Q-Networks

In single-agent systems, this basic approach is also impractical in high dimensional tasks, because the action-value function is estimated separately for each state, without any generalization. The Deep Q-Networks (DQN) algorithm [12] uses a deep neural network with weights θ as a function approximator, where $Q(s,a;\theta) \approx Q^*(s,a)$. Agents compute a processed state $\phi_t = \Phi(o_t)$ (dependent on the problem; Mnih et al. [12] used a stack of the last 4 frames to compensate for partial observability) and, under the policy $\pi(\phi,a)$, the sequence of loss functions in iteration i with target y_i, denoted by

$$L_i(\theta_i) = \mathbb{E}_{\phi,a\sim\pi(.)}[(y_i - Q(\phi,a;\theta_i))^2], \tag{2}$$

is minimized, in order to train the Q-network. A target network is kept, whose parameters θ^-_{i-1} for iteration i are fixed, and only updated (copied from the on-line network) every τ time-steps. The target of the Double DQN [7] extension is

$$y_i = \mathbb{E}_{\phi'\sim\mathcal{E}}[r + \gamma \max_{a'} Q(\phi', \arg\max_{a'} Q(\phi,a';\theta_{i-1});\theta^-_{i-1})|\phi,a], \tag{3}$$

which decouples action selection and evaluation over the on-line and target networks. This successfully reduces overoptimism, and results in a more stable and reliable learning process. The gradient of the loss function with respect to the weights can be optimized by stochastic gradient descent. The agent's experience at each time-step $e_t = (\phi_t, a_t, r_t, \phi_{t+1})$ is stored in a data-set $\mathcal{D} = e_1, \ldots, e_N$, known as a replay memory. Updates are performed with mini-batches (k samples of experience $e \sim \mathcal{D}$) drawn uniformly from the replay memory.

2.2 Multi-agent Reinforcement Learning

Reinforcement Learning in the multi-agent paradigm (MARL) has several advantages over the single-agent counterpart. Parallel computation leads to speedups in the learning phase when agents exploit the decentralized structure of the task [2]. MARL also benefits from the multi-agent systems' advantages, such as robustness and scalability.

Q-learning has been widely adopted by the scientific community as a viable option to find good strategies in cooperative multi-agent environments. Claus and Boutilier [3] describe multi-agent Q-learning with Independent Learners (IL) and Joint-Action Learners (JAL). In IL, each learner's Q-values are based solely on its own actions. In JAL, on the other hand, the learner's Q-values are based on the joint-action of all the agents. In practice, this implies that JAL agents require knowledge about the other agent's actions during the learning phase to form their Q-tables, while IL agents do not.

3 Related Work

There are many approaches to multi-agent coordination, both with hand-made policies and with multi-agent learning techniques. Hand-made policies usually rely on domain-knowledge, and use both situation- and role-based mechanisms [10,14,15] to achieve flexibility among cooperative agents. Multi-agent learning techniques, like multi-agent Q-learning [3] have shown good results in fully cooperative or fully competitive scenarios. However, since state-action representations are stored in tables, without any generalization to unseen states, these algorithms suffer from the curse of dimensionality, and require specific techniques to decrease the state- and action-spaces. Other techniques [8,11] worsen these problems by relying on state-based or agent-based lists and counters to improve upon the agents' coordination. Other algorithms focus on Decentralized MDPs and POMDPs, but assume unrealistic assumptions, such as complete independence from other agents [1] or the ability to observe other agents' actions [13]. Most of these have also only been evaluated in the context of single-stage games.

Multi-agent adaptations of deep-learning algorithms, on the other hand, have recently emerged as a solution to multi-agent learning. Tampuu et al. [17] use the *Pong* video-game and adjust the rewarding schemes of the game to range from cooperative to competitive behaviors. The authors use the Independent Learners approach (which would be necessary in non-cooperative scenarios), and report great results, such as cooperative agents learning to hit the ball parallel to the x-axis. Egorov [4] has also adopted the original DQN algorithm to a multi-agent scenario using the Pursuit environment and the Independent Learners technique. The author demonstrates how the algorithm can generalize to similar tasks, with a different number of agents, or different obstacles. Finally, the author uses a Transfer Learning technique, by transferring the network weights between similar scenarios, to speed-up learning. Foerster et al. [5] propose a set of techniques for multi-agent DQN, such as inter-agent weight sharing, where the same network is used by all agents, instead of several separate ones. The authors also suggest feeding each agent's last action to its input and disabling the experience replay feature of DQN, and demonstrate how multi-agent riddles can be solved by learning communication protocols.

4 Multi-agent Double Deep Q-Networks

Our proposal, which we call Multi-agent Double Deep Q-Networks (MaDDQN) is a multi-agent version of Double DQN, using the Joint-Action Learners (JAL) and Independent Learners (IL) strategies for fully cooperative scenarios. JAL is a technique where agents learn the value of joint-actions, instead of just their own, where Q-values represent the expected return of the joint-action $\mathcal{A}$. In fully cooperative scenarios, the optimal action for each agent is their respective component of the joint-action with the largest Q-value. IL is a technique where agents learn the value of their own actions, and thus do not require knowledge about the actions performed by other agents. When the number of agents $n = 1$, the algorithm is the original Double DQN.

The full JAL algorithm is described in Algorithm 1. Each agent interacts with the environment and executes a random action with probability ϵ. Agents observe the new state and the obtained reward, and store the joint-action transition in a replay memory. For JAL, a global processed state ϕ_t must always be computable by any agent, a requirement satisfied in all POMDP and fully observable Dec-MDP, but not in all Dec-POMDP. In the simplest case, the processed state is

```
Input: Learning rate $\eta$, mini-batch size $k$, discount factor $\gamma$, network update period $\tau$, replay
    memory $\mathcal{D}$ with capacity $N$, action-value function $Q$ with random weights $\theta$
1:  for iteration = $1, M$ do
2:      for agent $p = 1, P$ do
3:          Sample state $s_{1,p}$
4:      end for
5:      Compute $\phi_1$
6:      for step $t = 1, T$ do
7:          for agent $p = 1, P$ do
8:              Select random action $a_{t,p}$ with probability $\epsilon$, otherwise best action
                $a_{t,p} = \max_a Q^*(\phi(s_t), a; \theta)$
9:              Execute $a_{t,p}$
10:             Sample state $s_{t+1,p}$ and reward $r_t$
11:         end for
12:         Compute $\phi_{t+1}$
13:         Store transition $(\phi_t, a_{t,1}, ..., a_{t,p}, r_t, \phi_{t+1})$ in $\mathcal{D}$
14:         Sample random mini-batch of $k$ transitions $(\phi_j, a_{j,1}, ..., a_{j,b}, r_t, \phi_{j+1})$ from $\mathcal{D}$
15:         for transition $i = 1, k$ do
16:             Update $\theta \leftarrow \theta + \eta \nabla_{\theta_i} L_i(\theta_i)$
17:         end for
18:         Update network weights $\theta_{target} \leftarrow \theta$ every $\tau$ time-steps
19:     end for
20: end for
```

Algorithm 1. Joint-Action Multi-agent Double DQN

simply the latest observed state stripped of any agent-specific information (such as agent perspective). The full IL algorithm is described in Algorithm 2. In this case, the processed state ϕ_t can be different for any agent, thus relaxing the requirements of JAL. Transitions are stored by each agent and no longer contain the joint-action, but simply the action of their corresponding agent. Unlike Tampuu et al. [17], who focus on the diversity of coordinated behaviors between two agents through different rewarding schemes, we assume fully cooperative agents and instead focus on their adaptability to similar tasks and to larger team sizes. We also train all our agents at the same time, as opposed to Egorov [4], who fixes the network weights of all-but-one agents during training and periodically distributes the learned weights to the remaining agents.

The input of the network is the processed state ϕ_t (which is usually dependent on the observation o_t), implying that larger observations (like larger maps) will require a change in network architecture. The output of each network is the optimal action for IL, and optimal joint-action for JAL. Since the joint-action is dependent on the number of agents, the JAL variant of MaDDQN cannot generalize for different numbers of agents, as opposed to the IL variant.

5 Results

Our proposal was tested in two multi-agent environments: a foraging task and a predator-prey game. The foraging task is a task where agents are tasked with foraging items and bringing them back to specific places. Our implementation of the Foraging Task provides homogeneous agents with complete vision, but no communication. An exemplary map is shown in Fig. 1(a). The starting positions of agents and berries are randomized across the lower and upper parts of the

```
Input: Learning rate $\eta$, mini-batch size $k$, discount factor $\gamma$, network update period $\tau$, replay
    memory $\mathcal{D}$ with capacity $N$, action-value function $Q$ with random weights $\theta$
1: for iteration $= 1, M$ do
2:     for agent $p = 1, P$ do
3:         Sample state $s_{1,p}$ and compute $\phi_{1,p}$
4:     end for
5:     for step $t = 1, T$ do
6:         for agent $p = 1, P$ do
7:             Select random action $a_{t,p}$ with probability $\epsilon$, otherwise best action
               $a_{t,p} = \max_a Q^*(\phi(s_t), a; \theta)$
8:             Execute $a_{t,p}$
9:             Sample state $s_{t+1,p}$ and reward $r_t$
10:            Compute $\phi_{t+1,p}$
11:            Store transition $(\phi_{t,p}, a_{t,p}, r_t, \phi_{t+1,p})$ in $\mathcal{D}$
12:        end for
13:        Sample random mini-batch of $k$ transitions $(\phi_{j,b}, a_{j,b}, r_t, \phi_{j+1,b})$ from $\mathcal{D}$
14:        for transition $i = 1, k$ do
15:            Update $\theta \leftarrow \theta + \eta \nabla_{\theta_i} L_i(\theta_i)$
16:        end for
17:        Update network weights $\theta_{target} \leftarrow \theta$ every $\tau$ time-steps
18:    end for
19: end for
```

Algorithm 2. Independent Multi-agent Double DQN

map, respectively. Each agent can only carry one berry at a time, and can only release it in the base. Agents move simultaneously and, if they would collide, no movement occurs. All state transitions and rewards are deterministic, and agents observe the map from a global perspective. A time limit of 200 steps was set for each simulation. The Predator-Prey game is also known as the Pursuit task, where a team of homogeneous predators must capture the elements of the team of semi-randomly moving prey in a toroidal grid. Communication was disabled and predators must on top of a prey to capture it. An exemplary map is shown in Fig. 1(b). The starting positions of predators and prey are randomized across the map. Movements occur alternatively for predators and prey, but are simultaneous for members of the same team. Predators are penalized if they collide, and set randomly on the map. The prey are intelligent and move randomly away from the closest predator. We use unreliable communication channels, and the prey move semi-randomly, so state transitions and rewards are stochastic. Agents observe a local perspective of the relative positions of other elements in the map, and there is no time limit. For a single agent, catching the prey is impossible, as both move at the same speed.

Based on the DQN algorithm, an average network value $\mathcal{V}$ was used to determine the learning performance for our tests, which corresponds to the average Q-value of the best action in all steps of a fixed simulation. Tests were performed on a 7 by 7 grid, whose size is small enough for a policy with Q-tables to be learned, and on a 15 by 15 grid, too large for table-based methods. For the small maps, a fully connected neural network with 2 hidden layers of 50 neurons was deployed. The hidden layer units are activated with a rectified linear function $f(x) = \max(0, x)$, and the output layer units with the identity function $f(x) = x$. In our tests, the Replay Memory had a capacity $N = 5000$ transitions, and the discount factor, network update period, mini-batch size, and learning rate took

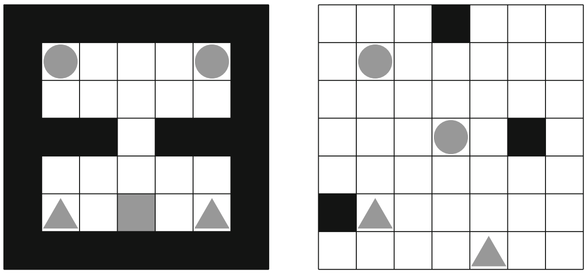


Fig. 1. The initial state of a Foraging map (left) and of a random toroidal Pursuit map (right). The berries and prey are represented in circles, the agents and predators in triangles, the base in squares, and obstacles in black.

values $\gamma = 0.9$, $\tau = 500$, $k = 32$, $\eta = 0.001$, respectively. The exploration rate ϵ was annealed from 1 to 0.1 during the first 75% steps. We use the Adam Optimizer [9] for optimization and Xavier initialization [6] for the weights' initial values. In order to compare our proposal with table-based Q-learning, we also learned standard Q-learning policies with a discount factor and learning rate of $\gamma = 0.9$ and $\eta = 0.5$, respectively. For tests on 15 by 15 environments, 2 hidden layers of 250 neurons were used. Training took five times as long, and a trainer was used to guide the initial exploration phase, by guiding the agents or handicapping the prey with some probability.

5.1 Joint-Action Learners and Independent Learners

The Foraging Task and the Pursuit game were used to compare the learning performance of the JAL and IL approaches. Both environments provide agents with a categorical grid of the observed world, which is converted to a 1-hot grid. The input layer consists of $w * h * m$ neurons, where w is the map width, h the map height, and m the number of map cell categories (empty, obstacle, base, berry or prey, agent). The output layer, for JAL, contains a^n neurons, where a is the number of possible actions, and n is the number of agents in the team. In the Forager Task, $a = 7$ (move in any of four directions, pick up, drop, do nothing), while in the Pursuit Game, $a = 5$ (move in any of four directions, do nothing). For IL, the output layer contains simply a neurons. Foerster et al.'s inter-agent weight sharing [5] was used for a faster and more consistent learning, by providing each agent with its own network, but using the same weights across all networks, which takes advantage of the fact that our agents are homogeneous. In a 7×7 map, with only a narrow gap between both halves of the map, we expected the IL approach to be harder to learn than its JAL counterpart. The results in Fig. 2(a) show, however, that both approaches have very similar learning curves. We conclude that this is due to the homogeneity of the agents and the inter-agent weight sharing technique, which allows the independent agents to quickly learn a homogeneous cooperative policy. In the Pursuit game, agents observe a local perspective of the map, and their perspectives were concatenated in the

JAL approach. Figure 2(b) shows how the learning process evolved, for a team of 2 agents attempting to catch a team of 2 prey. We see that JAL is now easier to learn, due to the strong coordination requirements of the environment. The agents learn to converge and surround each prey, until they eliminate it, before moving on to the next. We also trained our networks in large 15 by 15 maps, with up to 6 agents and up to 10 berries or prey. In the Pursuit environment, we added some random obstacles for increased complexity. Using regular Q-learning, the amount of states was intractable for the learning process and no policy was learned. However, MaDDQN managed to learn highly successful policies for the both environments (over 80% perfect score in less than 200 time steps).

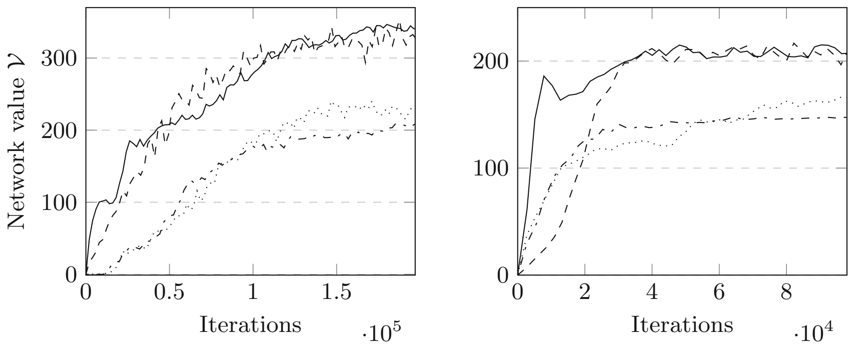
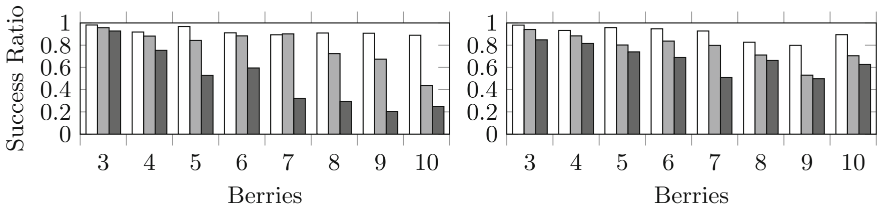


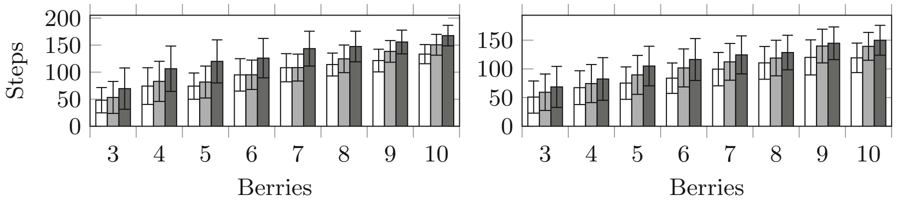
Fig. 2. The average and standard deviation of $\mathcal{V}$ in fixed Foraging (left) and Pursuit (right) simulations during the learning process. IL (solid) and JAL (dashed) strategies with MaDDQN, and IL (dotted) and JAL (dash dotted) strategies with Q-tables, are averaged over 3 trials.

5.2 Generalization

We also evaluated the generality of our proposal, by directly measuring the task performance when the task conditions were changed. Transfer learning approaches [18] allow learning in one task to improve the learning performance in a related, but different, task. We used 7 by 7 maps with 2 agents for the Foraging task, to demonstrate how MaDDQN behaves in narrow complex global-perspective environments, and 15 by 15 maps with 4 agents for the Pursuit games, to contrastingly demonstrate MaDDQN's behavior in large toroidal local-perspective environments. After training both MaDDQN and standard Q-learning (using Q-tables in 7 by 7 maps), the policies were then progressively re-trained for a fraction of the original training for different numbers of berries or prey and their performance was re-evaluated. The fractions of the original training for these tests, which we call Generalization Fractions (GF), ranged from 10% to 2.5%. The results of DQN generalization for the Foraging environment are shown in Fig. 3, where the ratio of successful simulations represents simulations where the agents captured all the berries within the time limit. The average



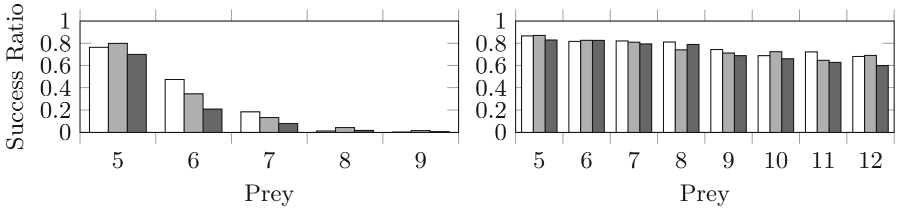
(a) Success rate of JAL (left) and IL (right) adaptation.



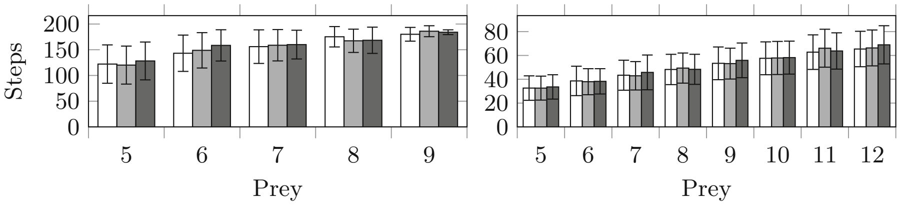
(b) Steps taken for JAL (left) and IL (right) adaptation.

Fig. 3. Adapting to Foraging scenarios with progressively more berries, over 1000 test simulations, using the JAL and IL approaches, with a GF of 10% (white), 5% (gray), and 2.5% (dark gray). (a) The ratio of successful attempts. (b) The average and standard deviation of the steps taken in successful attempts.

amount of steps per successful simulation is also shown, with the simulation time limit being kept at 200 steps. Both figures show how DQN-based algorithms are generalizable to more complex scenarios with only a limited amount of training of the network's weights, by achieving a success rate of over 80%, with at least 10% training. Using Q-tables, however, despite achieving a near perfect success rate on the original scenario, generalization achieved only a 12% and a 5% success rate with a GF of 10% for 3 berries in JAL and IL, respectively, and could not generalize any further. In some of our tests, we saw that policies actually improved, despite the increased complexity of the tasks. One of the initial policies consisted on only one of the agents picking up berries, while the other simply tried to avoid collisions. With 10% training, however, the agents learned a better policy, where both alternate behaviors and both carry berries. The same approach was tried in the Pursuit domain as well, as shown in Fig. 4, where a simulation is considered successful when all the prey were caught within 200 time-steps and without any agent collisions. Despite the increased map size, great results can be observed, with a success rate of over 60% with a GF of only 2.5% on the IL approach. The JAL approach, however, on large or complex maps, has issues generalizing. In fact, we used only 2 predators (for a simpler network output) and prey who only escaped with a 50% probability, and still the results show that after an increase of only 2 prey, predators already have an unsatisfactory behavior. In the Pursuit game, even with the small 7 by 7 map, Q-tables also failed to generalize with positive results. Generalization with a GF

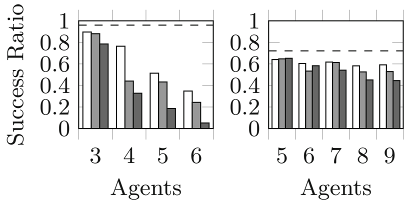


(a) Success rate of JAL (left) and IL (right) adaptation.

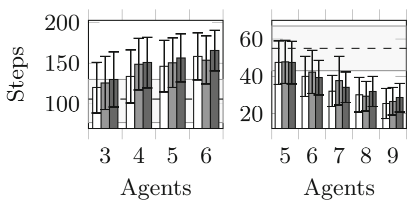


(b) Steps taken for JAL (left) and IL (right) adaptation.

Fig. 4. Adapting to Pursuit scenarios with progressively more prey, over 1000 simulations, using the JAL and IL approaches, with a GF of 10% (white), 5% (gray), and 2.5% (dark gray). (a) The ratio of successful attempts. (b) The average and standard deviation of the steps taken in successful attempts.



(a) Success rate of Foraging (left) and Pursuit (right) adaptation.



(b) Steps taken for Foraging (left) and Pursuit (right) adaptation.

Fig. 5. Adapting to Foraging and Pursuit scenarios with progressively more agents, over 1000 simulations, using the IL approach, with a GF of 10% (white), 5% (gray), and 2.5% (dark gray). The dashed line represents the original training's baseline, with 200000 training steps for Foraging and 400000 for Pursuit. (a) The ratio of successful attempts. (b) The average and standard deviation of the steps taken in successful attempts.

of 10% achieved only a 17% and a 4% success rate for 3 prey in JAL and IL, respectively, and could not generalize further than 4 prey.

Unlike the JAL counterpart, the IL approach also allows the generalization of the number of agents performing the task. The scenarios were trained with 6 berries and only 2 agents, and 4 predators and 10 prey. The network was then re-trained, as before, with up to 6 agents and up to 9 predators, and re-evaluated. The results for the Foraging and Pursuit tasks are shown in Fig. 5. In the Foraging narrow map, a larger amount of agents are not expected to help parallelize the task performance. Indeed, the results show that agents disturb each other and conflict in an attempt to cross the narrow bridge. On the other hand, we expect the same generalization to improve the performance of tasks where a broad map allows more agents to speed up the task completion. In the Pursuit environment, given a toroidal map, increasing the amount of agents with a small fraction of training speeds up the completion of the task, while decreasing its success rate (due to a larger amount of predator collisions).

6 Conclusion

We formally extend the Double DQN algorithm to the multi-agent paradigm. The Independent Learners and Joint-Action Learners techniques are described, and their viability for learning complex policies in multi-agent scenarios is shown, in comparison with regular Q-learning. The algorithm code, experiments, and demonstrations have been published on-line at https://github.com/bluemoon93/Multi-agent-Double-Deep-Q-Networks.git.

We demonstrate how MaDDQN can generalize and corroborate previous results, where transfer learning is shown to speed up convergence. With only a small fraction of the original training, IL can generalize not only on more complex tasks, with a higher success rate, but also with a larger number of agents. On the other hand, JAL has issues adapting its policy to more complex environments, due to the increased complexity of the network. We conclude that the independent approach with implicit coordination is not only more generalizable, but also a more realistic solution than a centralized approach, on highly complex environments. Transfer learning techniques allow for gradually harder tasks to be learned, whose complexity might be prohibitively high without any previous knowledge, as well as an increase in the speed of a parallel task's completion, by increasing the team's size. We believe that these are stepping stones to solving the complexity problems commonly associated with multi-agent systems.

In the future, we plan on extending MaDDQN to make use of other multi-agent adaptations of Q-learning, apart from IL and JAL, which allow for non-deterministic policies and non-cooperative scenarios. Finally, other deep reinforcement learning techniques, aside from DQN, must also be extended to the multi-agent paradigm and compared against MaDDQN.

Acknowledgements. The first author is supported by FCT (Portuguese Foundation for Science and Technology) under grant PD/BD/113963/2015. This research was partially supported by IEETA and LIACC. The work was also funded by project EuRoC, reference 608849 from call FP7-2013-NMP-ICT-FOF.

References

1. Becker, R., Zilberstein, S., Lesser, V., Goldman, C.V.: Transition-independent decentralized markov decision processes. In: Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS 2003, pp. 41–48. ACM, New York (2003)

2. Busoniu, L., Babuska, R., De Schutter, B.: A comprehensive survey of multiagent reinforcement learning. Trans. Syst. Man Cybern. Part C **38**(2), 156–172 (2008)
3. Claus, C., Boutilier, C.: The dynamics of reinforcement learning in cooperative multiagent systems. In: Innovative Applications of Artificial Intelligence, IAAI 1998, pp. 746–752. American Association for Artificial Intelligence (1998)
4. Egorov, M.: Multi-agent deep reinforcement learning. University of Stanford, Department of Computer Science, Technical report (2016)
5. Foerster, J.N., Assael, Y.M., de Freitas, N., Whiteson, S.: Learning to communicate to solve riddles with deep distributed recurrent q-networks. CoRR abs/1602.02672 (2016)
6. Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. In: AISTATS, vol. 9, pp. 249–256 (2010)
7. van Hasselt, H., Guez, A., Silver, D.: Deep reinforcement learning with double q-learning. CoRR abs/1509.06461 (2015)
8. Kapetanakis, S., Kudenko, D.: Reinforcement learning of coordination in cooperative multi-agent systems. In: Eighteenth National Conference on Artificial Intelligence, Menlo Park, CA, USA, pp. 326–331. American Association for Artificial Intelligence (2002)
9. Kingma, D.P., Ba, J.: Adam: a method for stochastic optimization. CoRR abs/1412.6980 (2014)
10. Lau, N., Reis, L.P.: FC Portugal - high-level coordination methodologies in soccer robotics. InTech Education and Publishing, Vienna, December 2007
11. Lauer, M., Riedmiller, M.: An algorithm for distributed reinforcement learning in cooperative multi-agent systems. In: Proceedings of the Seventeenth International Conference on Machine Learning, pp. 535–542. Morgan Kaufmann (2000)
12. Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M.: Playing atari with deep reinforcement learning. CoRR abs/1312.5602 (2013)
13. Nair, R., Tambe, M., Yokoo, M., Pynadath, D., Marsella, S., Nair, R., Tambe, M.: Taming decentralized pomdps: towards efficient policy computation for multiagent settings. In: IJCAI, pp. 705–711 (2003)
14. Reis, L.P., Lau, N., Oliveira, E.C.: Situation based strategic positioning for coordinating a team of homogeneous agents. BRSDMAS 2000. LNCS, vol. 2103, pp. 175–197. Springer, Heidelberg (2001). doi:10.1007/3-540-44568-4_11
15. Stone, P.: Layered Learning in Multiagent Systems: A Winning Approach to Robotic Soccer. MIT Press, Cambridge (2000)
16. Stone, P., Veloso, M.: Multiagent systems: a survey from a machine learning perspective. Auton. Robot. **8**(3), 345–383 (2000)
17. Tampuu, A., Matiisen, T., Kodelja, D., Kuzovkin, I., Korjus, K., Aru, J., Aru, J., Vicente, R.: Multiagent cooperation and competition with deep reinforcement learning. CoRR abs/1511.08779 (2015)
18. Taylor, M.E., Stone, P.: Transfer learning for reinforcement learning domains: a survey. J. Mach. Learn. Res. **10**(1), 1633–1685 (2009)
19. Watkins, C.J., Dayan, P.: Q-learning. Mach. Learn. **8**(3–4), 279–292 (1992)