Multi-Agent Deep Reinforcement Learning

Maxim Egorov
Stanford University
megorov@stanford.edu

Abstract

This work introduces a novel approach for solving reinforcement learning problems in multi-agent settings. We propose a state reformulation of multi-agent problems in $\mathcal{R}^2$ that allows the system state to be represented in an image-like fashion. We then apply deep reinforcement learning techniques with a convolution neural network as the Q-value function approximator to learn distributed multi-agent policies. Our approach extends the traditional deep reinforcement learning algorithm by making use of stochastic policies during execution time and stationary policies for homogenous agents during training. We also use a residual neural network as the Q-value function approximator. The approach is shown to generalize multi-agent policies to new environments, and across varying numbers of agents. We also demonstrate how transfer learning can be applied to learning policies for large groups of agents in order to decrease convergence time. We conclude with other potential applications and extensions for future work.

1. Introduction

Multi-agent systems arise in a variety of domains from robotics to economics. In general, decision-making in multi-agent settings is intractable due to the exponential growth of the problem size with increasing number of agents. The goal of this work is to study multi-agent systems using deep reinforcement learning (DRL). In particular, we explore if effective multi-agent policies can be learned using DRL in a stochastic environment. In deep reinforcement learning, a neural network is used to estimate the Q-value function in a stochastic decision process (e.g. Markov decision process or MDP). Recent work has shown that deep Q-Networks can be used to achieve human-level performance in the Atari video game domain [13]. However, little work has been done in the area of multi-agent DRL. The most closely related work examines two agents in the Atari environment playing Pong [15]. That work only considers two agents in a domain with deterministic dynamics. This work will examine more general systems with an arbitrary number of agents who may be self-interested. The DRL approach to computing multi-agent policies is promising because many of the traditional approaches for solving multi-agent MDPs fail due to their enormous size and complexity. For example, Dec-MDPs [1] are not scalable past a few number of agents and small state spaces. If the agents are self-interested (e.g. they do not share a single reward function), the problem becomes more difficult and approaches such as I-POMDPs [5] must be used to allow agents to reason about other self-interested agents in the environment. This work explores if DRL can alleviate some of the scalability issues encountered in the literature for multi-agent systems.

In this work, we introduce a novel reformulation of the multi-agent control problem. Specifically, we consider discrete state systems in $\mathbb{R}^2$ with self-interested agents (see Figure 1 for an example). The reformulation involves decomposing the global system state into an image like representation with information encoded in separate channels. This reformulation allows us to use convolutional neural networks to efficiently extract important features from the image-like state. We extend the state-of-the-art approach for solving DRL problems [13] to multi-agent systems with this state representation. In particular, our algorithm uses the image-like state representation of the multi-agent system as an input, and outputs the estimated Q-values for the agent in question. We describe a number of implementation contributions that make training efficient and allow agents to learn directly from the behavior of other agents in the system. Finally, we evaluate this approach on the problem of pursuit-evasion. In this context, we show that the DRL policies can generalize to new environments and to previously unseen numbers of agents.

2. Prior Work

The sub-field of deep reinforcement learning has been quickly growing over the last few years. However, attempts to use non-linear function approximators in the context of reinforcement learning have been unsuccessful for a long time, primarily due to possibility of divergence when up-

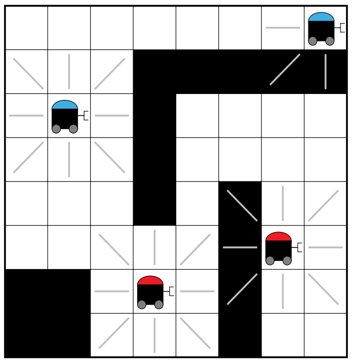

Figure 1. Example instance of a pursuit-evasion game, where the two pursuers (red agents) are attempting to catch the two evaders (blue agents). The agents are self interested, i.e. the pursuers and the evaders have different objectives. The filled in grid points indicate obstacles through which the agents cannot pass.

date schemes are not based on bound trajectories [16]. Recent work has demonstrated that using neural networks for approximating Q-value functions can lead to good results in complex domains by using techniques like experience replay, and target networks during updates [13], giving birth to deep reinforcement learning. Deep reinforcement learning has been successfully applied to continuous action control [9], strategic dialogue management [4]and even complex domains such as the game of Go [14]. Also, a number of techniques have been developed to improve the performance of deep reinforcement learning including double Q-Learning [17], asynchronous learning [12], and dueling networks [19] among others. However, work on extending deep reinforcement learning to multi-agent settings has been limited. The only prior work known to the author involves investigating multi-agent cooperation and competition in Pong using two deep Q-Network controllers [15]. This work aims to fill a part of that gap in literature.

3. Methods

This section describes the approaches taken in this work. We first online how deep reinforcement learning works in single agent systems. We then propose our approach to extending deep reinforcement learning to multi-agent systems.

3.1. Deep Reinforcement Learning

In reinforcement learning, an agent interacting with its environment is attempting to learn an optimal control policy. At each time step, the agent observes a state s, chooses an action a, receives a reward r, and transitions to a new state s'. Q-Learning is an approach to incrementally estimate the utility values of executing an action from a given state by continuously updating the Q-values using the following rule:

$$Q(s,a) = Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s',a') - Q(s,a)). \quad (1)$$

Where $Q(s,a)$ denotes the utility of taking action a from state s. Q-learning can be directly extended to DRL frameworks by using a neural network function approximator $Q(s,a|\theta)$ for the Q-values, where θ are the weights of the neural network that parametrize the Q-values. We update the neural network weights by minimizing the loss function:

$$\mathcal{L}(s,a|\theta_i) = (r + \gamma \max_{a} Q(s',a|\theta_i) - Q(s,a|\theta_i))^2. \quad (2)$$

The backpropogation algorithm is used to update the network weights at iteration $i+1$ by performing the computation: $\theta_{i+1} = \theta_i + \alpha \nabla_\theta \mathcal{L}(\theta_i)$. In this work the ADAM update rule [7] was used. The use of neural networks a Q-value function approximators was an open research challenge for a long time. The two insights that significantly improve convergence and training rates are the use of experience real dataset and the use of a target Q-network for computing the loss. The experience replay dataset contains a fixed number of transition tuples in it that contain (s,a,r,s') where r is the reward obtained by performing action a from state s, and s' is the state the agent transitions to after performing that action. The experience tuples are sampled uniformly from the experience replay dataset in mini batches, and are used to update the network. The addition of experience replay helps prevent correlation between training samples, which improves convergence. The use of a target network in the loss function calculation helps convergence as well. The target network in the loss function $Q(s',a|\theta_i)$ is kept fixed for a certain number of iterations, and is updated periodically.

3.2. Multi-Agent Deep Reinforcement Learning

This section outlines an approach for multi-agent deep reinforcement learning (MADRL). We identify three primary challenges associated with MADRL, and propose three solutions that make MADRL feasible. The first challenge is problem representation. Specifically, the challenge is in defining the problem in such a way that an arbitrary number of agents can be represented without changing the architecture of the deep Q-Network. To solve this problem, we make a number of simplifying assumptions: (i) two dimensional representation of the environment, (ii) discrete time and space, and (iii) two types of agents. Because, we limit ourselves to two agent types, we will refer to the two agent types as allies and opponents (assuming competing agents). These assumptions allow us to represent the global system state as an image-like tensor, with each channel of the image containing agent and environment specific information (see Figure 2). This representation allows us to take advantage of Convolutional Neural Networks which have been shown to work well for image classification tasks [8]. The image tensor is of size $4 \times \mathrm{W} \times \mathrm{H}$, where W and H are the height and width of our two dimensional domain and four is the number of channels in the image. Each channel encodes a different set of information from the global state in its pixel values (shown in Figure 2). The channels can be broken down in the following way:

- **Background Channel:** contains information about any obstacles in the environment
- **Opponent Channel:** contains information about all the opponents
- **Ally Channel:** contains information about all the allies
- **Self Channel:** contains information about the agent making the decision

Note that channels in the image-like representation are sparse. In both the opponent and ally channels, each non-zero pixel value encodes the number of opponents or allies in that specific position.

The second challenge is multi-agent training. When multiple agents are interacting in an environment, their actions may directly impact the actions of other agents. To that end, agents must be able to reason about one another in order to act intelligently. In order to incorporate multi-agent training, we train one agent at a time, and keep the policies of all the other agents fixed during this period. After a set number of iterations the policy learned by the training agent gets distributed to all the other agents of its type. Specifically, an agent distributes its policy to all of its allies (see Figure 3). This process, allows one set of agents to incrementally improve their policy over time. The learning process itself is not distributed, since one agent must share its policy with all of its allies. However, the policy execution is distributed, because each agent has their own neural network controller. Each agent must be able to sense the locations of all the other agents, but does not need to explicitly communicate with the other agents about its intent.

The last challenge is dealing with agent ambiguity. Consider a scenario where two ally agents are occupying the same position in the environment. The image-like state representation for each agent will be identical, so their policies will be exactly the same. To break this symmetry, we enforce a stochastic policy for our agents. The actions taken by the agent are drawn from a distribution derived by taking a softmax over the Q-values of the neural network. This allows allies to take different actions if they occupy the same state and break the ambiguity. In previous work, the policy used in evaluation was ϵ-greedy [13]. However, a more principled approach would be to use a stochastic policy generated by taking a softmax over the Q-values as proposed here.

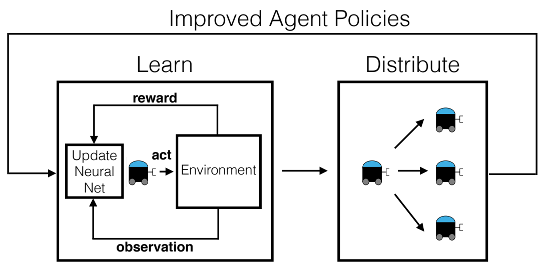

Figure 3. Schematic of the multi-agent centralized training process

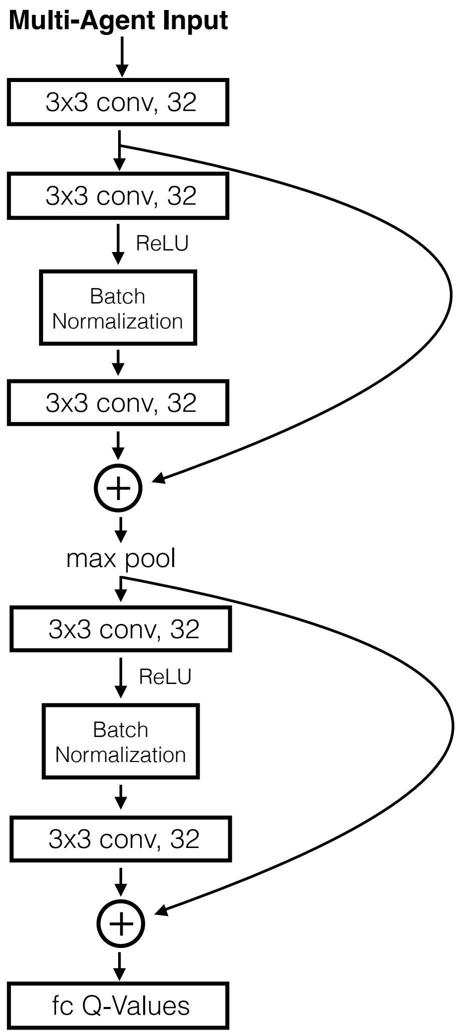

Figure 5. The multi-agent deep Q-Network (MADQN) architecture

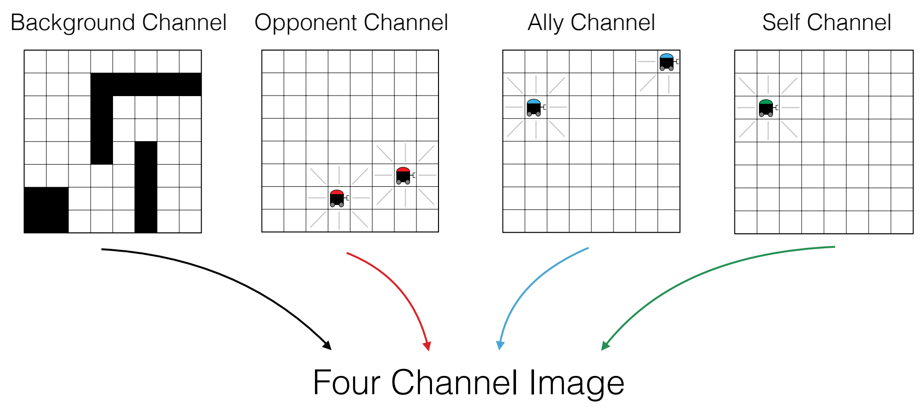

Figure 2. Four channel image-like representation of the multi-agent state used as input to the deep Q-Network. The background channel encodes the locations of the obstacles in its pixel values, while the ally and opponent channels encode the locations of the allies and opponents. The self channel encodes the location of the agent that is making the decision.

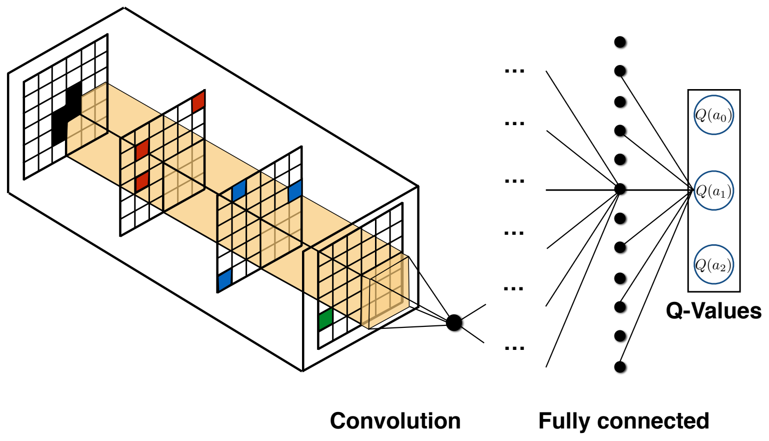

Figure 4. An example schematic showing the multi-agent image-like state being used as an input to the convolutional neural network.

4. Problem Formulation and Network Architecture

The primary goal of this work is to evaluate the approach outlined in the previous section on a simple problem and quantitively identify interesting multi-agent behavior such as cooperation. To accomplish this, we pick the pursuit-evasion problem [3] as our evaluation domain. In pursuit-evasion, a set of agents (the pursuers) are attempting to chase another set of agents (the evaders). The agents in the problem are self-interested (or heterogeneous), i.e. they have different objectives. In particular, we focus on computing pursuit policies using the MADRL approach. Specifically, the evaders follow a simple heuristic policy, where they move in the direction that takes them furthest from the closest pursuer. The pursuers are attempting to learn a distributed neural network controller as outlined in the previous section. Since we limit the problem to a discrete state space, the pursuers receive a reward anytime they occupy the same grid cell as an evader. In this work, we modify the pursuit-evasion task such that the terminal state of the system is reached only when all the evaders are tagged by a pursuer. The pursuers receive a large reward in the terminal state. Thus, in order to reach the high rewarding terminal state, the pursuers must be able to make decisions in a cooperative manner.

To train the neural network controller, the data set containing the image-like state representations was generated through simulation. Specifically, each data sample contained an experience tuple of the form (s, a, r, s'), where s is the starting image-like state, a is the action taken, r is the reward received, and s' is the new state of the system. As with other deep reinforcement learning approaches, the experience replay dataset from which the network was trained, was continuously updated during training.

The neural network architecture used in this work follows many of the same principles as in previous work [13], with some modifications. Since our state representation is image-like, we can use a convolutional neural network architecture. A schematic of the multi-agent image-like state being used as an input to a convolutional neural network is shown in Figure 4. In the figure, the image-like representation captures all the state information, while the output layer of the network represents the Q-values for all the actions the agent can take from that state. The network architecture used in this work is shown in Figure 5. The differences between this architecture and the architectures used in previous works is that we adopt the Residual Network [6] architecture or skip connections to improve gradient flow throughout the network, and we use batch normalization. The name of this architecture is multi-agent deep Q-Network or MADQN.

5. Results

For the majority of the evaluation, we focused on a 10×10 instance of the pursuit-evasion problem with a single rectangular obstacle in the middle of the environment. The image-like states were generated by simulating the motion of the agents, and converting the state into a three dimensional tensor. In this section we provide three experimental results that test the capabilities of the MADQN controller. We first demonstrate the ability of the MADQN to generalize to previously unseen environments. Next, we show that the MADQN can generalize across different numbers of agents. Specifically, we show that MADQN controllers that have been trained on 2 agent instances can perform well in 3 agent settings. Lastly, we demonstrate a transfer learning approach for MADQN that speeds up training convergence. During all the evaluations, the initial conditions for each simulation were set by using rejection sampling. The state space was uniformly sampled, and states were rejected if either agent was inside of an obstacle or if a pursuers was too close to an evader (start at least one grid cell away).

Figure 6 shows the experimental set-up for MADQN generalization over new environments. The figure depicts the training and evaluation scenarios we used for this experiment. We trained the MADQN on two instances of the pursuit-evasion problem and evaluated on a new instance to see how well the controller can generalize. Namely, we considered a scenario with a single evader and a single pursuer, on three configurations (Figure 6). The difference between each instance is the obstacle configuration. The MADQN was trained on the two scenarios shown on the left of Figure 6, and evaluated on the scenario shown on the right of the figure. The results are outlined in Table 1. The table shows evaluation results of both the split policy (policy obtained form training on the two problem instances at once) and the combined policy (policy obtained from training on the problem instance where the obstacles are combined). We see that the split policy performs just as well as the combined policy on the scenario where the obstacles are combined. We also see that the combined policy can generalize to scenario where the obstacles are split. While this result is preliminary, and could be studied with more rigor, it demonstrates that MADQN controllers can generalize well to new environments.

Next, we evaluated MADQN policies on varying numbers of agents. The results are shown in Figure 8. The figure shows the average reward for a given policy evaluated on different scenarios. In this evaluation, we consider three types policies:

- **N vs N:** MADQN policies obtained through training on a problem instance with N evaders and N pursuers (e.g. 1 vs 1, 2 vs 2, etc).
- **Heuristic:** A heuristic policy in which the pursuer moves towards the evader closest to it.
- **Value iteration:** The optimal policy obtained by solving the multi-agent MDP (tractable only for a scenario with 1 evader and 1 pursuer)

For the N vs N policies, we considered the following training scenarios: 1 vs 1, 2 vs 2 and 3 vs 3. Both the N vs N and the heuristic policies were evaluated on the following scenarios: 1 vs 1, 2 vs 2, 3 vs 3. We also evaluated the value iteration policy on the 1 vs 1 scenario. There are three main things to take away form the figure. First, all of the policies perform near optimally on the 1 vs 1 scenario. Since the value iteration policy servers as an upper bound on performance, we can directly infer that all policies evaluated on the 1 vs 1 scenario have ned-optimal performance. This indicates that the MADQN network can learn near optimal policies for the simple 1 vs 1 scenario. Second, the performance of the 1 vs 1 and the heuristic policies on the 2 vs 2 and the 3 vs 3 scenarios is nearly identical. Since the heuristic is just a greedy strategy, this implies that the policy learned by the MADQN when trained on the 1 vs 1 scenario is similar to the greedy heuristic. This is an important point to consider when evaluating policies that were trained for more than one agent. In particular, the 1 vs 1 policy has no knowledge of cooperation. This can be seen in the evaluation results for the 2 vs 2 and the 3 vs 3 scenarios. Lastly, we compare the performance of the policies trained on the 2 vs 2 and the 3 vs 3 scenarios against each other. While the two policies perform best on the scenarios that they were trained on, both outperform the 1 vs 1 policy on scenario that they were not trained on. For example, the 2 vs 2 policy does nearly as well as the 3 vs 3 policy on the 3 vs 3 scenario. This implies that the MADQN controllers are able to generalize the notion of cooperation between agents to different numbers of agents fairly well.

In practice training these networks on large problems is time consuming. We explored a transfer learning approach to alleviate some of the computational burden. Figure 8

Table 1. Average rewards per time-step for the pursuit policies evaluated over 50000 time-steps

	Maps		
	Flat	Upright	Combined
Split Policy	0.164	0.159	0.143
Combined Policy	0.152	0.149	0.145

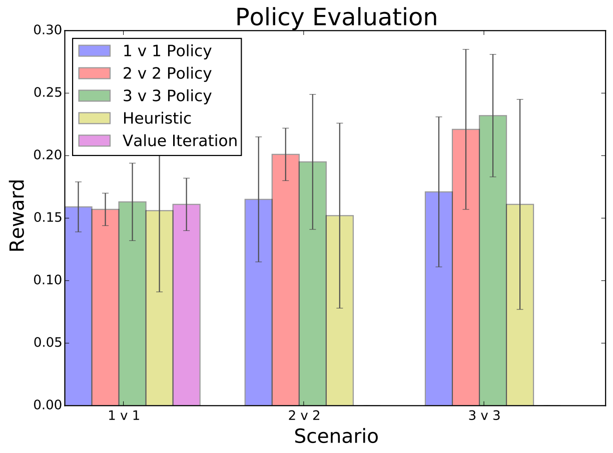

Figure 7. Average reward per time step for various policies across three different pursuit-evasion scenarios.

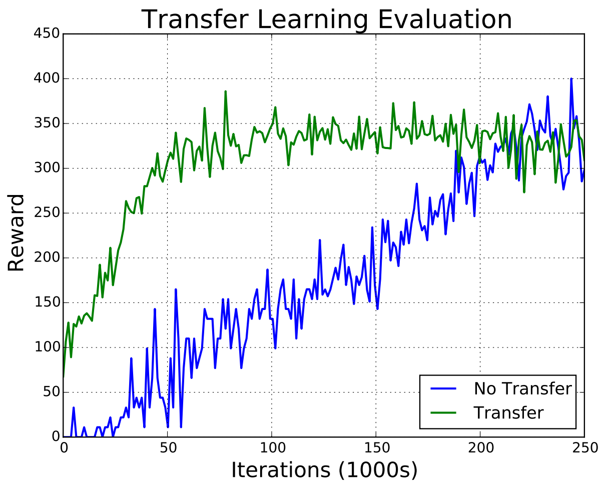

Figure 8. Evaluation reward for the MADQN during training time for a 2 vs 2 pursuit-evasion scenario. The green line shows a MADQN initialized with the same weights as a converged 1 vs 1 MADQN, while the blue line shows the MADQN initialized with random weights.

shows convergence plots for two to MADQNs training on the 2 vs 2 scenario. The plot shows the average reward obtained by evaluating each MADQN after a given number of training iterations. The blue curve was generated by training a network with random weight initialization (from scratch). The green curve used the network trained until convergence on the 1 vs 1 scenario (transfer network). The transfer learning approach of using a network trained on a different but related environment allows us to speed up convergence in multi-agent systems. This indicates that we can train a single network on a specific problem domain, and if we want to fine tune it for a specific number of agents, we can use transfer learning to quickly improve its performance.

6. Conclusion

Despite many of the recent advances in optimal control and automated planning, multi-agent systems remain an open research challenge. The difficulties arise from the curse of dimensionality that makes systems with large numbers of agents intractable. In this work, we demonstrated that a deep reinforcement learning approach can be used to solve decision making problems that are intractable for classical algorithms such as value iteration. The three primary contributions of this work are: (i) demonstrated generalization across environments in multi-agent systems with MADRL, (ii) demonstrated generalization across a varying number of agents using MADRL, and (iii) showed that transfer learning can be used in multi-agent systems to speed up convergence.

There are a number of potential directions for future work. While this work considered self-interested agents implicitly, their policies were kept stationary. The next step would be to train networks that control heterogenous agents. This could be done by having a network for each type of agent, and alternating training between the networks. This would allow the networks to learn better policies as the behavior of other types of agents changes in the environment. Another area of future work would involve extending this approach to more complicated domains. One potential application is robotic soccer. Specifically, selective coordination approaches have been shown to work well for robot soccer [11]. Future work will explore if MADRL can be used to learn policies that perform as well as the selective coordination approaches and if it can outperform them. Another part of future work is to do a more through comparison of the MADRL approach to other multi-agent reinforcement learning techniques such as the minimax Q-algorithm [2]. It may be of value to compare this approach to game theoretic [18] and other reinforcement learning [10] techniques.

7. Appendix: Multi-Agent Deep Q-Network Hyperparameters

The hyperparameters used for training the MADQN can be found in 2.

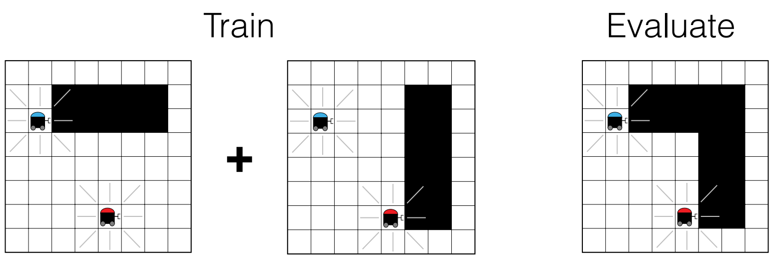

Figure 6. Generalization across environments: the multi-agent deep Q-Network was trained on the two pursuit-evasion scenarios on the right, and was evaluated on the scenario on the left

Table 2. Deep Q-Network hyperparameters

Hyperparameter	Value	Decription
Max train iterations	250000	The maximum number of training samples generated
Minimbatch size	32	Number of training samples per update
Replay size	100000	Size of the experience replay dataset
Learning rate	0.0001	Rate used by the optimizer
Update rule	ADAM	The parameter update rule used by the optimizer
Initial exploration	1.0	Initial ϵ value in ϵ greedy exploration policy
ϵ decay	5e-5	Rate at which ϵ decreases
Target network update	5000	Frequency of updating the target network
Agent network update	50000	Frequency of updating the networks of other agents

References

[1] D. S. Bernstein, R. Givan, N. Immerman, and S. Zilberstein. The complexity of decentralized control of Markov decision processes. *Mathematics of Operations Research*, 27(4):819–840, 2002.

[2] L. Busoniu, R. Babuska, and B. De Schutter. A comprehensive survey of multiagent reinforcement learning. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 38(2):156–172, 2008.

[3] T. H. Chung, G. A. Hollinger, and V. Isler. Search and pursuit-evasion in mobile robotics. *Autonomous robots*, 31(4):299–316, 2011.

[4] H. Cuayáhuitl, S. Keizer, and O. Lemon. Strategic dialogue management via deep reinforcement learning. *arXiv preprint arXiv:1511.08099*, 2015.

[5] P. J. Gmytrasiewicz and P. Doshi. A framework for sequential planning in multi-agent settings. *J. Artif. Intell. Res. (JAIR)*, 24:49–79, 2005.

[6] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[7] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[8] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, page 2012, 2012.

[9] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[10] J. Liu, S. Liu, H. Wu, and Y. Zhang. A pursuit-evasion algorithm based on hierarchical reinforcement learning. In *Measuring Technology and Mechatronics Automation, 2009. ICMTMA'09. International Conference on*, volume 2, pages 482–486. IEEE, 2009.

[11] J. P. Mendoza, J. Biswas, P. Cooksey, R. Wang, S. Klee, D. Zhu, and M. Veloso. Selectively reactive coordination for a team of robot soccer champions. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2016.

[12] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *arXiv preprint arXiv:1602.01783*, 2016.

[13] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 02 2015.

[14] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[15] A. Tampuu, T. Matiisen, D. Kodelja, I. Kuzovkin, K. Korjus, J. Aru, J. Aru, and R. Vicente. Multiagent cooperation and competition with deep reinforcement learning. *CoRR*, abs/1511.08779, 2015.

[16] J. N. Tsitsiklis and B. Van Roy. An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, 42(5):674–690, 1997.

[17] H. Van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. *arXiv preprint arXiv:1509.06461*, 2015.

[18] R. Vidal, O. Shakernia, H. J. Kim, D. H. Shim, and S. Sastry. Probabilistic pursuit-evasion games: theory, implementation, and experimental evaluation. *Robotics and Automation, IEEE Transactions on*, 18(5):662–669, 2002.

[19] Z. Wang, N. de Freitas, and M. Lanctot. Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581*, 2015.