Reducing Overestimation Bias in Multi-Agent Domains Using Double Centralized Critics

Johannes Ackermann *
Technical University of Munich
johannes.ackermann@tum.de

Volker Gabler
Technical University of Munich
v.gabler@tum.de

Takayuki Osa
Kyushu Institute of Technology, RIKEN
osa@brain.kyutech.ac.jp

Masashi Sugiyama
RIKEN, The University of Tokyo
sugi@k.u-toyo.ac.jp

Abstract

Many real world tasks require multiple agents to work together. Multi-agent reinforcement learning (RL) methods have been proposed in recent years to solve these tasks, but current methods often fail to efficiently learn policies. We thus investigate the presence of a common weakness in single-agent RL, namely value function overestimation bias, in the multi-agent setting. Based on our findings, we propose an approach that reduces this bias by using double centralized critics. We evaluate it on six mixed cooperative-competitive tasks, showing a significant advantage over current methods. Finally, we investigate the application of multi-agent methods to high-dimensional robotic tasks and show that our approach can be used to learn decentralized policies in this domain.

1 Introduction

In recent years, many real world problem settings have been modeled as multi-agent systems, be it smart-grid applications [10], package routing [28], or road transportation [1].

While it is possible to regard these problems as a centralized single agent, with a large state and action space, and apply methods from single-agent reinforcement learning (RL), this leads to an action space that increases exponentially with the number of agents [15]. Another approach is to assume independent learners [24], in which agents regard the influence of other agents as part of the environment. However, due to the behavior of other agents changing over time, the transition probabilities change, leading to the Markov assumption being violated. Therefore, recent research has focused on decomposing these systems into individual, decentralized agents during execution, while updating them in a centralized training phase, allowing to maintain the Markov property during training [18]. Although recent research has introduced powerful multi-agent reinforcement learning (MARL) techniques based on this principle, such as counterfactual multi-agent (COMA) policy gradients [4], or the multi-agent deep deterministic policy gradient (MADDPG) method [14], their performance has not been studied as thoroughly as approaches in single-agent RL. Motivated by findings in the single-agent case [26, 5], which have shown it to generally suffer from an overestimation bias of the value function, we thus investigate this issue in MARL on the example of the popular MADDPG method.

Similarly to multi-agent tasks, complex robotic systems face the challenge of high-dimensional and continuous state-action spaces. A popular way to approach these tasks is decentralized control

Preprint. Accepted for the NeurIPS Deep RL Workshop 2019.

*Work performed while at the University of Tokyo

[7], which requires a high amount of model knowledge. Recently [19] presented a method to learn decentralized policies with RL, but it still relies on a shared or centralized meta-policy. We propose a way to learn truly decentralized policies, by modeling robotic systems as multi-agent systems, eliminating the need for a centralized controller.

The main contribution of our work is a new method for MARL, that addresses overestimation bias and outperforms previous methods in most of the evaluated cooperative-competitive tasks. Furthermore, we provide an approach to learn decentralized policies for high-dimensional robotic tasks, based on MARL. We show that our method is able to learn decentralized policies on a simulated task and outperforms existing MARL methods.

2 Background

In this section, we explain the relevant background for our work, first explaining general methods in RL, then addressing more recent policy gradient algorithms.

2.1 Markov Games

In our work, we focus on Markov games [13], an extension of Markov decision processes to multi-agent domains. A Markov game with N agents consists of a state set $\boldsymbol{S}$, a collection of action sets, $\boldsymbol{A}_1, \ldots, \boldsymbol{A}_N$, a transition function $T : \boldsymbol{S} \times \boldsymbol{A}_1 \times \cdots \times \boldsymbol{A}_N \rightarrow \mathrm{Dist}(\boldsymbol{S})$, with $\mathrm{Dist}(\boldsymbol{S})$ being a distribution over states. Each agent has its own reward function $r_i : \boldsymbol{S} \times \boldsymbol{A}_1 \times \cdots \times \boldsymbol{A}_N \rightarrow \mathbb{R}$, which depends on the actions of all agents. Since we regard decentralized agents, they each posses a different observation set $\boldsymbol{O}_i$, which is available to them during execution, and choose actions according to a policy $\pi_i : \boldsymbol{O}_i \rightarrow \mathrm{Dist}(\boldsymbol{A}_i)$.

The agents each aim to maximize their own total expected return $R_i = \sum_{t=0}^{t=T} \gamma^t r_i$, with a discount factor $0 < \gamma \leq 1$ and time horizon T. If $r_i = k r_j$, $i \neq j$, the interaction is cooperative for $k > 0$ and competitive for $k < 0$.

2.2 Q-Learning

Q-learning [22] is an off-policy algorithm that learns the value of executing action a in state s in form of the expected return $Q^\pi(s, a) = \mathbb{E}[R|s_t = s, a_t = a]$, which can be recursively obtained as $Q^\pi(s, a) = \mathbb{E}_{s'}[r(s, a) + \gamma \mathbb{E}_{a' \sim \pi(s')}[Q^\pi(s', a')]]$. Assuming a greedy policy $\pi(s) = \arg\max_a (Q^\pi(s, a))$, Q-learning can be used to learn an optimal policy.

Mnih et al. [16] proposed an approach that approximates the Q-function with multi-layer perceptrons (MLPs), called deep Q-networks (DQN). It uses a target network $Q^\pi_{\theta'}$, whose parameters θ' slowly follow the network parameters of Q^π_θ, to update the parameters of the Q-network. Additionally, transitions (s, a, r, s') are stored in a replay buffer $\boldsymbol{D}$.

Double Q-learning [25] found that Q-learning often overestimates the Q-value in stochastic environments, leading to a failure to learn an efficient policy. To remove this positive bias, they proposed to learn two Q-functions Q_1, Q_2, which are updated using the value of the respective other function. For Q_1, this resolves to $Q_1(s, a) = Q_1(s, a) + \alpha(r + \gamma Q_2(s', a') - Q_1(s, a))$, with $a' = \arg\max_a Q_1(s', a)$.

2.3 Policy Gradient Methods

Sutton et al. [23] took a different approach to optimizing the behavior of the agent, by directly using gradient descent on the parameters of the policy. Their target $J(\theta) = \mathbb{E}_{s \sim p^\pi, a \sim \pi_\theta}[\sum_{t=0}^{\infty} \gamma^t r_t]$ is defined as the expected total reward over a policy dependent state distribution p^π and the gradient resolves to

$$\nabla_\theta J(\theta) = \mathbb{E}_{s \sim p^\pi, a \sim \pi_\theta} \left[\nabla_\theta \log \pi_\theta(a|s) Q^\pi(s, a) \right] . \tag{1}$$

In 2014, Silver et al. [20] derived a formulation of the policy gradient theorem for deterministic policies $\mu : \boldsymbol{S} \rightarrow \boldsymbol{A}$ called deterministic policy gradient (DPG). They also showed that deterministic policies tend to learn significantly quicker than stochastic policies in some domains.

An algorithm for RL in continuous control problems, based on DPG, was presented in [12], called the deep deterministic policy gradient (DDPG). It performs off-policy updates using transitions from a replay buffer $\boldsymbol{D}$ and utilizes a target network, as in DQN. Using this, the gradient in (1) becomes

$$\nabla_\theta J(\theta) = \mathbb{E}_{s\sim D}\left[\nabla_\theta \mu_\theta(s) \nabla_a Q^\mu(s,a)|_{a=\mu_\theta(s)}\right] . \tag{2}$$

2.4 TD3

The twin delayed deep deterministic policy gradient (TD3) [5] improves on DDPG by addressing the overestimation bias of the Q-function, similarly to double Q-learning. They find that due to approximation errors of the MLP, combined with gradient descent, DDPG tends to overestimate the Q-value of state-action pairs, leading to a slower convergence. TD3 addresses this by using two Q-networks $Q_{\theta_1}, Q_{\theta_2}$, along with two target networks. The Q-functions are updated with the target $y = r_t + \gamma \min_{1,2} Q_{\theta'_i}(s', a')$, while updating the policy with Q_{θ_1}. Additionally, they introduce target policy smoothing by adding noise in the determination of the next action for the critic target $a' = \mu_{\theta'_\pi}(s') + \epsilon$, with ϵ being clipped Gaussian noise $\epsilon = \texttt{clip}(\mathcal{N}(0, \sigma), -c, c)$, where c is a tunable parameter.

Additionally they use delayed upolicy updates, and only update the policy π and target network parameters θ'_π, θ'_Q once every d critic updates.

2.5 Multi-Agent Deep Deterministic Policy Gradient

MADDPG [14] is an extension of DDPG to the multi-agent setting. It uses the decentralized execution with a centralized training setting, learning a centralized critic that has access to the policies of all agents. This centralized Q-function, representing the expected future reward of agent i, is then learned with

$$Q_i^\pi(\mathbf{x}, a_1, ..., a_N) = \mathbb{E}_{r,\mathbf{x}'}[r_i + \gamma Q_i'^\pi(\mathbf{x}', \mu_1(o'_1), ..., \mu_N(o'_N)] \tag{3}$$

Using this Q-function, the deterministic policy of agent i can be optimized by gradient descent:

$$\nabla_{\theta_i} J(\mu_i) = \mathbb{E}_{\mathbf{x}, a_{j\neq i}\sim \boldsymbol{D}}\left[\nabla_{\theta_i} \mu_i(a_i|o_i) \nabla_{a_i} Q_i^\pi(\mathbf{x}, a_1, ..., a_N)|_{a_i=\mu_i(o_i)}\right] . \tag{4}$$

In this work we denote the observation received at runtime by agent i as o_i, and the full state information as $\mathbf{x}$, from which the observations o_i are derived. The replay buffer $\boldsymbol{D}$ here contains transitions $(\mathbf{x}, a_1, ..., a_N, r_1, ..., r_N, \mathbf{x}')$ of all agents.

3 Overestimation Bias in a Centralized Critic

Motivated by related work [5, 25, 26], that found an overestimation bias in other methods, we investigate whether this effect persists in the multi-agent domain on the example of the popular MADDPG approach.

As we are using deterministic policies, we can, in the short-term, approximate the environment as stationary from the view-point of each agent, so that the we can regard the transition probability as $P(s'|s, a_1, ..., a_N) \approx P(s'|s, a_i)$.

Under this assumption, we can approximate the value of our centralized critic as $Q_i^\pi(\mathbf{x}, a_1, ..., a_N) \approx Q_i^\pi(\mathbf{x}, a_i)$, reducing the setting to the one regarded in [5], in which they have shown that overestimation occurs in DDPG.

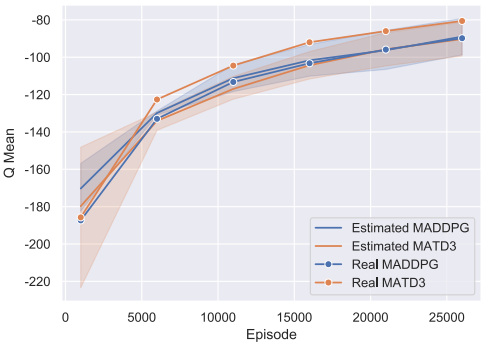

Figure 1: Empirical evaluation of overestimation in MARL. The Q-values estimated by the Q-network and the true Q-values are shown. The results are averaged across 5 runs and 95 % CIs of the mean are shown for the estimated values. We can see, that MADDPG overestimates the Q-values, while MATD3 underestimates them and achieves higher real values.

Empirical Evaluation: To test whether this overestimation also appears in practice, we evaluated MADDPG on the "Cooperative Navigation" task as outlined in [14]. We increased the number

of time-steps per episode to 200 and determine the true and estimated Q-values by sampling states and actions from the replay buffer, that were saved since the last evaluation time-step. From those states, we perform 200 rollouts, with 100 steps each, and save the discounted reward. We then compare the mean of the discounted rewards with the value of the Q-function approximator. The results are shown in Figure 1. They show that MADDPG tends to overestimate the Q-values, especially during earlier episodes. Looking at single runs, we can see that this overestimation does not always occur, but when it happens it leads to a significantly worse final performance.

It should also be noted that the evaluated domains are deterministic. In contrast to that, most real world applications are stochastic. Stochasticity has been shown to lead to a higher value function overestimation, because it adds to the noise from function approximator errors [25].

4 Multi-Agent TD3

Our proposed approach, called multi-agent TD3 (MATD3), extends TD3 to the multi-agent domain in a similar manner to the extension of DDPG to MADDPG. We use the centralized training with decentralized execution setting, in which we assume that during training we have access to the past actions, observations and rewards, as well as policies, of all agents. We use this information to learn two centralized critics $Q^{\pi}_{i,\theta_{1,2}}(\mathbf{x}, a_1, ..., a_N)$ for each agent i. In order to reduce the overestimation bias, we update them with the minimum of both critics: $y_i = r_i + \gamma \min_{j=1,2} Q^{\pi}_{i,\theta_j}(\mathbf{x}', a'_1, ..., a'_N)$. This may lead to underestimation, however, this is preferable to overestimation: In the case of overestimation, actions with an overestimated value are chosen with a higher probability, due to the policy update. When then updating the critic, the overestimated value of the next action a' is used $Q^{\pi}(\mathbf{x}', \mu'(o'))$, which propagates the error to the update target y. If it is underestimated, the probability of choosing this action is reduced in the policy update. It is thus not used to update the Q-values and the error does not propagate further.

In addition, we use target policy smoothing, adding clipped Gaussian noise $\epsilon = \texttt{clip}(\mathcal{N}(0, \sigma), -c, c)$ to the actions of all agents in the critic update: $a'_j = \mu_{\theta'_j}(o'_j) + \epsilon$. This serves as a regularization, based on the assumption that similar actions should have similar values. The complete target for the critic resolves to

$$y_i = r_i + \gamma \min_{j=1,2} Q^{\pi}_{i,\theta'_j}(\mathbf{x}', \mu'_1(o'_1) + \epsilon, ..., \mu'_N(o'_N) + \epsilon)\,, \tag{5}$$

with μ'_i being short for $\mu_{\theta'_i}$. The policies are updated similar to (4), but using Q_{i,θ_1} instead of Q_i. We also employ delayed policy updates, only updating the target networks θ'_Q, θ'_π and policies π_i after every d critic updates. This is motivated in [5] by the need to have an accurate critic before using it to update the policy, thus updating the critic more often. This is especially crucial in multi-agent domains, as the change of the critic values has to reflect not only small changes in the own policy, but also in the policies other agents. However, in adversarial settings this is not always beneficial, as it can slow the adaption to the policy of an adversary. The full algorithm is shown in the Appendix.

5 Evaluation in Particle Environments

We evaluate the efficacy of our approach on the particle environments proposed by [17] and used to evaluate MADDPG in [14]. They are shown in Figure 2. The particle environments consist of two-dimensional continuous state spaces, in which agents can exert a force on themselves. Additionally, the agents may have access to a discrete communication channel to each of the other agents.

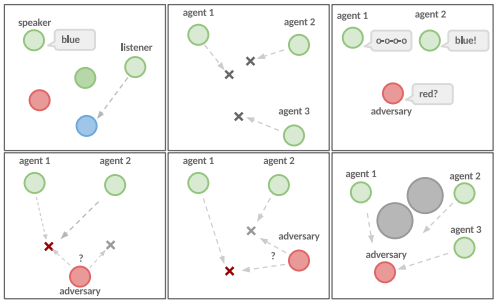

Figure 2: Illustration of the particle environment tasks used in our evaluation. Left to right, top to bottom: "Cooperative Communication", "Cooperative Navigation", "Covert Communication", "Keep-away", "Physical Deception", "Predator-Prey". The figure is based on [14].

The particle environments consist of a set of six tasks: Two cooperative tasks called "Cooperative Navigation" and "Cooperative Communication", in addition to four adversarial tasks "Covert

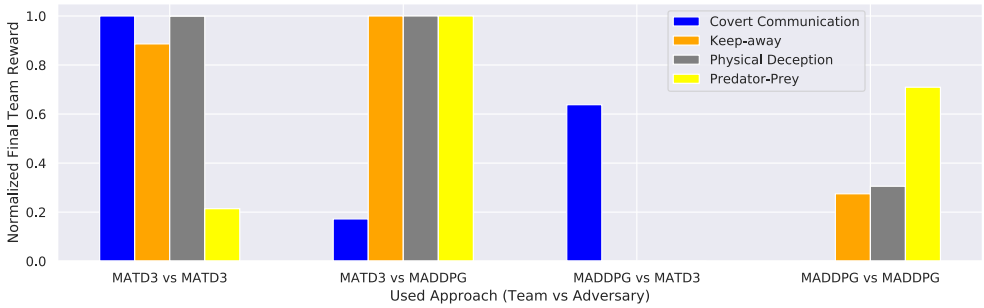

Figure 4: Evaluation in the adversarial domains (Team vs Single Agent). Shown is the 0-1 normalized final reward of the team, in all combinations of MATD3 and MADDPG, averaged across 20 trials each. In the domains where it is necessary to learn a stable winning strategy ("Keep away", "Physical-Deception", "Predator-Prey") MATD3 outperforms MADDPG in the direct comparison. However, in the "Covert Communication" domain, where quick adaption to the policy of the adversary is advantageous, MADDPG outperforms MATD3.

Communication", "Keep-Away", "Physical Deception" and "Predator-Prey". In all of the adversarial tasks, there is a team of "Agents" and a single "Adversary". Most of the tasks require a team of agents to learn a cooperative strategy, which can deal with most behaviors of the adversarial agent. An exception is the "Covert Communication" task, in which the adversary has to decode a message the agents are sending to each other. In this task a good result can be achieved by quickly changing the communication scheme, without learning a more complex behavior.

5.1 Results

We implement our approach, named MATD3, and compare its performance to MADDPG.[2]

Hyper-parameters are chosen by grid-search over learn rate $\alpha = [0.01, 0.003, 0.001]$, mini-batch size $b = [256, 1000]$ and policy update frequency $d = [1, 2, 3]$. The parameters we found to work best are $\alpha = 0.01$, $b = 1000$, $d = 2$. We use the same set of parameters for all tasks, to ensure a fair comparison.

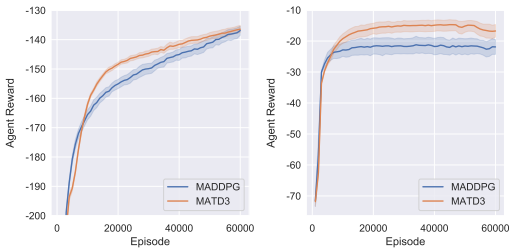

Figure 3: Evaluation in the cooperative domains used in [14], "Cooperative Navigation" (left) and "Cooperative Communication" (right). We can see, that MATD3 significantly outperforms MADDPG. Shown is the mean episodic reward over the last 1000 episodes, shaded areas are the 95 % confidence intervals of the mean, averaged across 20 trials.

For MADDPG we use the hyper-parameters and implementation provided in [14], which were tuned for the same tasks. For both approaches we approximate the Q-functions and policies with MLPs with two hidden layers with 64 units each. As activation function we use the Rectified Linear Unit (ReLU) function, and as optimizer we use use Adam [9] in all experiments, as well as the Gumbel-Softmax estimator [8].

Cooperative Environments The results in the cooperative tasks are shown in Figure 3. On the cooperative navigation task both achieve a similar final performance, while MATD3 learns a better policy significantly faster. On the cooperative communication task MATD3 achieves a significantly better final performance.

Competitive Environments Results in the competitive environments are shown in Figure 4, as 0-1 normalized, final rewards, averaged across 20 trials each. They show that, in direct comparison, MATD3 outperforms MADDPG in three out of four environments.

In the task where MADDPG significantly outperforms our proposed approach, 'Covert Communication', a team of agents has to learn a communication strategy, that the single adversary has to decode. Due to the delayed policy updates, MATD3 is slower at adapting to its opponent's behavior, thus

[2]The source code will be available after the review period, to ensure anonymity.

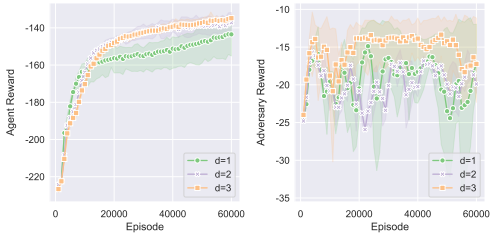

Figure 5: Effect of the policy update rate d on performance in "Cooperative Navigation" (left) and "Covert Communication" (right). We can see, that a less frequent policy update is beneficial in the cooperative task, while in the adversarial task it leads to a better performance of the Adversary, i.e., it being better at decrypting the communication of the agent team.

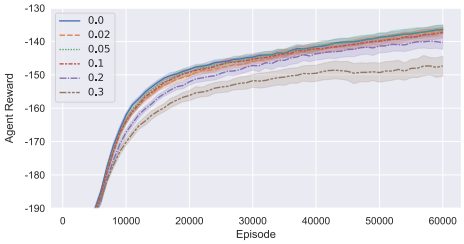

Figure 6: Evaluation of target policy smoothing on the "Cooperative Navigation" task. Shown is the mean reward for different values for ϵ, averaged across 10 runs each. Note the zoomed in axis. In our evaluation we did not find a significant advantage of target policy smoothing.

being outperformed by MADDPG. In the other tasks a consistent winning strategy, that can beat all behaviors of the single agent, can be learned, at which our proposed approach succeeds.

Delayed Policy Updates Delayed policy updates are intended to ensure a sufficiently converged critic before using it to update the policy. To investigate their effect in MARL, we evaluated different policy update rates and show the results in Figure 5. Less frequent policy updates showed to be beneficial in all tasks, leading to a lower variance in results and a higher final performance, with the exception being the "Covert Communication" task. In this task, the team of agents is made slower at changing it's communication policy, leading to the adversary being better at decrypting it.

Target Policy Smoothing We evaluate the effect of target policy smoothing in Figure 6 for different levels of added noise ϵ. We do not find target policy smoothing to improve the performance unlike in [5]. We assume that this is due to the policies of the other agents used in the critic target being updated frequently, and thus implicitly introducing a similar randomness.

We also evaluated the relative overestimation for different numbers of agents, but did not find a significant difference.

6 Learning Fully Decentralized Controllers for Robotic Systems

The particle environments regarded in the previous section are mostly fully observable, with the actions of one agent often not strongly affecting the other agents. Thus, in many tasks, a high reward can be achieved without much cooperation. We therefore also evaluate our approach in a new, challenging setting: Learning decentralized controllers for robotic systems. In high-dimensional robotic tasks decentralized control has been shown to be an effective method, enabling efficient locomotion [27]. It commonly functions by virtually subdividing the robot into multiple parts. These parts are then coordinated by a central controller. Additionally, in many cases this kind of decentralization is required, for example due to band-width limitations or the structure of the robot.

We propose to eliminate the need for a centralized or shared control policy by regarding the robot as a multi-agent system. Furthermore, this does not require the additional model knowledge to design a decentralized controller. We thus partition the robotic system into multiple agents, which only have access to partial information about the state of the other agents. The agents learn to coordinate their actions based on the reward signal they receive and the centralized critic.

The reduction to a partial observation for each agent leads to the task becoming a partially observable stochastic game [6], that requires a large degree of cooperation between agents. We therefore do not aim to outperform the current state-of-the-art methods proposed for single-agent continuous tasks, as they have access to the full-state, but see this as a new, challenging task for MARL.

6.1 Decomposition to Multiple Agents

We evaluate the functionality of our proposed approach on the OpenAI Gym [2] "Ant-v2" task. The ant consists of a spherical torso and four legs. The legs each have two actuated joints, one at the

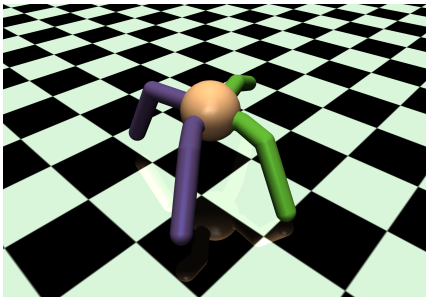

Figure 7: The "Ant-v2" task, split into two agents, visualized as the green and blue part. The observation of each agent consists of full information of its side, but only includes the joint positions of the other side, without acting forces or velocities.

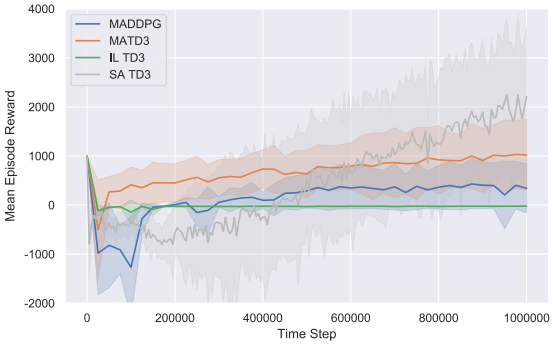

Figure 8: Performance on the "Ant-v2" task. Mean reward of deterministic evaluation episodes, averaged over 6 seeds per approach. The shaded area is a standard deviation. Shown are MATD3, MADDPG and independent learner (IL) TD3 on the decentralized task. For comparison we also show the performance of a single agent (SA) using TD3, with full state information. Note that this is a significantly less difficult setting.

attachment to the torso and one at the knee. The state information provided in the standard Gym task consists of all joint positions and angular velocities, position and velocity of the torso as well as contact forces with the floor. This is then used to generate a reward consisting of the distance traveled in a set direction and a cost term for actuator force and contact with the surface. Additionally, a positive reward is obtained for every time-step that the agent does not reach a terminal position, which occurs when the torso falls below a threshold height.

We separate the ant into two halves, as shown in Figure 7. The action-space of one agent comprises the two left legs, and the action-space of the other consists of the other two legs. As observation each agent receives all information about their respective legs, that is provided in the "Ant-v2" task - position, angular velocity and external forces - but only the position of the other legs. In addition, both agents receive the location and velocity of the torso of the ant.

6.2 Implementation

We implemented our approach, MATD3, for the ant experiments, along with a MADDPG and independent learner (IL) TD3 version. To our knowledge this is the first investigation of the efficacy of MADDPG in high-dimensional, continuous action spaces.

The hyperparameters for MATD3 and MADDPG were selected by grid-search over learn rate $\alpha = [0.01, 0.003, 0.001]$, batch-size $b = [100, 300]$ and $\tau = [0.005, 0.01]$. The hyper-parameters we found to be working best for both approaches are $\alpha = 0.001$, $b = 100$ as batch-size and $\tau = 0.01$, and $d = 2$ for MATD3. For single agent (SA) TD3 and IL TD3 we are using the hyper-parameters suggested in the original paper [5].

For all Q-functions and policies we use MLPs with two hidden layers with 400 and 300 units respectively. As activation function we use ReLU, except at the output, which uses a sigmoid activation. The output is then scaled linearly to the range of the respective action space.

6.3 Results

The results of our trials are shown in Figure 8. They show that MATD3 performs better than MADDPG. Both approaches usually first find a policy that remains in place, however, our approach, unlike MADDPG, usually recovers from this local optimum and achieves locomotion in the required direction. As a baseline we also show the performance of two independent learner (IL) TD3 agents, which receive the same observation as the MATD3 and MADDPG agents, but do not use a centralized critic. They failed to learn a successful policy in all trials.

Finally, we show the performance of SA TD3 in the standard, fully observable "Ant-v2" task. Unsurprisingly, due to the SA task being significantly easier, the final performance of SA TD3 outperforms all MA approaches. However it only starts to do so after a high number of time-steps, showing promise for further work.

7 Related Work

Regarding the improvement of MADDPG, Minimax Multi-Agent DDPG (M3DDPG) [11] has to be noted. They approximate a minimax training objective by adding adversarial perturbations to the actions of other agents when updating the critic and policy. However, their improvements are limited to adversarial tasks, while we also address cooperative ones. Additionally, it should be possible to combine their approach with ours. An approach that aims to reduce overestimation bias in MARL by using Double Deep Q Networks (DDQN) is presented in [21]. However, they do not investigate whether overestimation does indeed occur in MARL and if their approach reduces it. Further, their work focuses on discrete state and action spaces in a grid-world, while our work focuses on more complex, continuous domains. Furthermore, DDQN has been shown to not be effective in the actor-critic setting [5].

In the field of robotics, learning decentralized controllers via RL has been studied by [3]. Instead of using a centralized critic, they use independent critics which are augmented by certain additional observations of the other agents. In addition, they use value iteration to learn the policies, which does not scale to high-dimensional tasks, and only study comparatively simple tasks.

8 Conclusion and Future Work

We have shown, that overestimation occurs in multi-agent domains and significantly hinders convergence. We used this finding to propose a new approach for multi-agent reinforcement learning (MARL), called multi-agent TD3 (MATD3). It is based on the decentralized execution, centralized training setting and addresses the overestimation bias by using double centralized critics. We have shown that our proposed approach significantly outperforms MADDPG on most of the particle-domain tasks. In addition, we propose a new method of learning decentralized controllers for robotic tasks, by regarding them as multi-agent systems and using methods from MARL. We showed that we can use this approach to learn decentralized policies for the popular "Ant" task, and that our proposed approach also outperforms MADDPG in this domain.

For future work, we plan to investigate a hybrid approach, that combines the initial benefits of multi-agent TD3 (MATD3) with the later performance of TD3.

Algorithm 1: Multi-Agent TD3

Initialize replay buffer $\boldsymbol{D}$ and network parameters
for $t = 0$ to T_{max} **do**
  Select actions $a_i \sim \mu_i(o_i) + \epsilon$
  Execute actions $(a_1, ..., a_N)$, observe r_i, $\mathbf{x}'$
  Store transition $(\mathbf{x}, a_1, ..., a_N, r_1, ..., r_N, \mathbf{x}')$ in $\boldsymbol{D}$
  $\mathbf{x} \leftarrow \mathbf{x}'$
  for agent $i = 1$ to N **do**
    Sample a random minibatch of S samples $(\mathbf{x}^b, a^b, r^b, \mathbf{x}'^b)$ from $\boldsymbol{D}$
    $y^b \leftarrow r_i^b + \gamma \min_{j=1,2} Q_{i,j}^{\mu'}(\mathbf{x}'^b, a_1, ..., a_N) \mid_{a_k = \mu'_k(o'^b_k)+\epsilon}$
    Minimize Q-function loss for both critics $j = 1, 2$
    $\mathcal{L}(\theta_j) = \frac{1}{S} \sum_b (Q_{i,j}^{\mu}(\mathbf{x}^b, a_1^b, ..., a_N^b) - y^b)^2$
    if t mod $d = 0$ **then**
      Update policy μ_i with gradient

$$\nabla_{\theta_{\mu,i}} J \approx \frac{1}{S} \sum_b \nabla_\theta \mu_{\theta_{\mu,i}}(o_i^b) \nabla_{a_i} Q_{i,1}^{\mu}(\mathbf{x}^b, a_1^b, ..., \mu_{\theta_{\mu,i}}(o_i), ...a_N^b)$$

      Update the target networks $\theta' \leftarrow \tau\theta + (1-\tau)\theta'$

References

[1] J. L. Adler, G. Satapathy, V. Manikonda, B. Bowles, and V. J. Blue. A multi-agent approach to cooperative traffic management and route guidance. *Transportation Research Part B: Methodological*, 39(4):297 – 318, 2005.

[2] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym, 2016.

[3] L. Buşoniu, B. De Schutter, and R. Babuška. Decentralized reinforcement learning control of a robotic manipulator. In *ICARCV-06*, pages 1347–1352, 2006.

[4] J. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson. Counterfactual Multi-Agent Policy Gradients. In *AAAI*, 2018.

[5] S. Fujimoto, H. van Hoof, and D. Meger. Addressing Function Approximation Error in Actor-Critic Methods. In *ICML*, pages 1587–1596, 2018.

[6] E. A. Hansen, D. S. Bernstein, and S. Zilberstein. Dynamic Programming For Partially Observable Stochastic Games. In *AAAI*, pages 709–715, 2004.

[7] A. J. Ijspeert. Central pattern generators for locomotion control in animals and robots: A review. *Neural Networks*, 21(4):642–653, 2008.

[8] E. Jang, S. Gu, and B. Poole. Categorical Reparameterization with Gumbel-Softmax. In *ICLR*, 2017.

[9] D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. In *ICLR*, 2015.

[10] F. D. Li, M. Wu, Y. He, and X. Chen. Optimal control in microgrid using multi-agent reinforcement learning. *ISA Trans.*, 51(6):743–751, 2012.

[11] S. Li, Y. Wu, F. Fang, and S. Russell. Robust Multi-Agent Reinforcement Learning via Minimax Deep Deterministic Policy Gradient. In *AAAI*, 2019.

[12] 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. In *ICLR*, 2016.

[13] M. L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *ICML*, pages 157–163, 1994.

[14] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch. Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments. *NeurIPS*, pages 6379–6390, 2017.

[15] N. Mehta, P. Tadepalli, and C. Science. Multi-Agent Shared Hierarchy Reinforcement Learning. In *ICML Work. Richer Represent. Reinf. Learn.*, 2005.

[16] V. Mnih, K. Kavukcuoglu, D. Silver, A. a. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, pages 529–533, 2015.

[17] I. Mordatch and P. Abbeel. Emergence of Grounded Compositional Language in Multi-Agent Populations. In *AAAI*, pages 1495–1503, 2018.

[18] F. A. Oliehoek, M. T. Spaan, and N. Vlassis. Optimal and approximate Q-value functions for decentralized POMDPs. *J. Artif. Intell. Res.*, 32:289–353, 2008.

[19] G. Sartoretti, Y. Shi, W. Paivine, M. Travers, and H. Choset. Distributed learning for the decentralized control of articulated mobile robots. In *IEEE ICRA*, pages 1–6, May 2018.

[20] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller. Deterministic Policy Gradient Algorithms. *ICML*, pages 387–395, 2014.

[21] D. Simões, N. Lau, and L. P. Reis. Multi-agent Double Deep Q-Networks. In *Progress in Artificial Intelligence, EPIA 2017*, pages 123–134, 2017.

[22] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. Number 1. MIT press Cambridge, 1998.

[23] R. S. Sutton, D. Mcallester, S. Singh, and Y. Mansour. Policy Gradient Methods for Reinforcement Learning with Function Approximation. *NeurIPS*, pages 1057–1063, 1999.

[24] M. Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *ICML*, pages 330–337, 1993.

[25] H. van Hasselt. Double Q-Learning. In *NeurIPS*, pages 2613–2621, 2010.

[26] H. van Hasselt, A. Guez, and D. Silver. Deep Reinforcement Learning with Double Q-learning. In *AAAI*, pages 2094–2100, 2016.

[27] J. Whitman, F. Ruscelli, M. Travers, and H. Choset. Shape-based compliant control with variable coordination centralization on a snake robot. In *IEEE CDC*, pages 5165–5170, 2016.

[28] D. Ye, M. Zhang, and Y. Yang. A multi-agent framework for packet routing in wireless sensor networks. *Sensors (Switzerland)*, 15(5):10026–10047, 2015.