Cooperative Multi-Agent Control Using Deep Reinforcement Learning

Jayesh K. Gupta
Stanford University
jkg@cs.stanford.edu

Maxim Egorov
Stanford University
megorov@stanford.edu

Mykel Kochenderfer
Stanford University
mykel@stanford.edu

ABSTRACT

This work considers the problem of learning cooperative policies in complex, partially observable domains without explicit communication. We extend three classes of single-agent deep reinforcement learning algorithms based on policy gradient, temporal-difference error, and actor-critic methods to cooperative multi-agent systems. We introduce a set of cooperative control tasks that includes tasks with discrete and continuous actions, as well as tasks that involve hundreds of agents. The three approaches are evaluated against each other using different neural architectures, training procedures, and reward structures. Using deep reinforcement learning with a curriculum learning scheme, our approach can solve problems that were previously considered intractable by most multi-agent reinforcement learning algorithms. We show that policy gradient methods tend to outperform both temporal-difference and actor-critic methods when using feed-forward neural architectures. We also show that recurrent policies, while more difficult to train, outperform feed-forward policies on our evaluation tasks.

1. INTRODUCTION

Learning to cooperate between several interacting agents has been well studied [39, 30, 6]. While the problem of cooperation can be formulated as a decentralized partially observable Markov decision process (Dec-POMDP), exact solutions are intractable [1, 5]. A number of approximation methods for solving Dec-POMDPs have been developed recently that adapt techniques ranging from reinforcement learning [3] to stochastic search [28]. However, applying these methods to real-world problems is challenging because they are typically limited to discrete actions and require carefully designed features.

On the other hand, recent work in single agent reinforcement learning has enabled learning in domains that were previously thought to be too challenging due to their large and complex observation spaces. This line of work combines ideas from deep learning with earlier work on function approximation [40, 22], giving rise to the field of deep reinforcement learning. Deep reinforcement learning has been successfully applied to complex real-world tasks that range from playing Atari games [24] to robotic locomotion [20]. The recent success of the field leads to a natural question—how well can ideas from deep reinforcement learning be applied to cooperative multi-agent systems?

In this work, we focus on problems that can be modeled as Dec-POMDPs. We extend three classes of deep reinforcement learning algorithms: temporal-difference learning using Deep Q Networks [24], policy gradient using Trust Region Policy Optimization [33], and actor-critic using Deep Deterministic Policy Gradients [21]. We compare their performance on three benchmark tasks. The benchmark tasks were chosen to represent a diverse variety of complex environments with discrete and continuous actions and observations.

Our empirical evaluations show that using a decentralized parameter sharing neural network policy with an appropriate training protocol and choice of reward function leads to emergent cooperative behavior without explicit communication between agents. We also show that the policy gradient method scales to large multi-agent control tasks with dozens of agents required to complete collaborative tasks and hundreds of agents present in the environment. To our knowledge, this work presents the first cooperative reinforcement learning algorithm that can successfully scale in continuous action spaces. We call this algorithm PS-TRPO.

2. RELATED WORK

Multi-agent reinforcement learning has a rich literature [8, 30]. A number of algorithms involve value function based cooperative learning. [39] compared the performance of cooperative agents to independent agents in reinforcement learning settings. [29] identified modularity as a useful prior to simplify the application of reinforcement learning methods to multiple agents. [13] later extended this idea and factored the joint value function into a linear combination of local value functions and used message passing to find the joint optimal actions. [19] tried distributing the value function into learning multiple tables but failed to scale to stochastic environments.

On the other hand, policy search methods have found better success in partially observable environments [34]. [32] studied gradient based distributed policy search methods. Our solution approach can be considered a direct descendant of the techniques introduced in their work. However, instead of using finite state machines, our model uses deep neural networks to control the agents. This approach allows us to extend neural network controllers to tasks with continuous actions, use deep reinforcement learning optimization techniques, and consider more complex observation spaces.

Relatively little work on multi-agent reinforcement learning has focused on continuous action domains. A few notable approaches include those of [11] who focus on discretization and [37] who used a normalized Gaussian Network as a function approximator to learn continuous action policies. Many of these approaches only work in fairly restricted settings and fail to scale to high dimensional raw observations or continuous actions. Moreover, their computational complexity grows exponentially with the number of agents.

Multi-agent control has also been studied in extensive detail from the dynamical systems perspective in problems like formation control [10], coverage control [9], and consensus [27]. The limitations of the dynamical systems approach lie in its requirement for hand-engineered control laws and problem specific features. While the approach allows for development of provable characteristics about

the controller, it requires extensive domain knowledge and hand engineering. Overall, deep reinforcement learning provides a more general way to solve multi-agent problems without the need for hand-crafted features and heuristics by allowing the neural network to learn those properties of the controller directly from raw observations and reward signals.

Recent research has applied deep reinforcement learning to multi-agent problems. [38] extended the DQN framework to independently train multiple agents. Specifically, they demonstrate how collaborative and competitive behavior can arise with the appropriate choice of reward structure in a two-player Pong game. More recently, [12] and [36] train multiple agents to learn a communication protocol to solve tasks with shared utility. They demonstrate end-to-end differentiable training using novel neural architectures. However, these examples work with either relatively few agents or simple observations and do not share our focus on decentralized control problems with high dimensional observations and continuous action spaces.

3. BACKGROUND

In this work, we consider multi-agent domains that are fully cooperative and partially observable. All agents are attempting to maximize the discounted sum of joint rewards. No single agent can observe the state of the environment. Instead, each agent receives a private observation that is correlated with that state. We assume the agents cannot explicitly communicate and must learn cooperative behavior only from their observations.

Formally, the problems considered in this work can be modeled as Dec-POMDPs defined by the tuple $(\mathcal{I}, \mathcal{S}, \{\mathcal{A}_i\}, \{\mathcal{Z}_i\}, T, R, O)$, where $\mathcal{I}$ is a finite set of agents, $\mathcal{S}$ is a set of states, $\{\mathcal{A}_i\}$ is a set of actions for each agent i, $\{\mathcal{Z}_i\}$ is a set of observations for each agent i, and T, R, O are the joint transition, reward, and observation models, respectively. In this work, we consider problems where $\mathcal{S}$, $\mathcal{A}$, and $\mathcal{Z}$ can be infinite to account for continuous domains. In the reinforcement learning setting, we do not know T, R, or O, but instead have access to a generative model. It is natural to also consider a centralized model known as a multi-agent POMDP (MPOMDP), with joint action and observation models. The centralized nature of MPOMDPs makes them less effective at scaling to systems with many agents.

We briefly describe three single-agent deep reinforcement learning algorithms, including temporal-difference, actor-critic, and policy gradient approaches.

3.0.1 Deep Q-Network

The deep Q-network (DQN) algorithm [24] is a temporal-difference method that uses a neural network to approximate the state-action value function:

$$Q(s,a) = \max_{\pi} \mathbb{E}\left[r_t + \gamma r_{t+1} + \ldots \mid s_t = s, a_t = a, \pi\right]$$

DQN relies on an experience replay dataset $\mathcal{D}_t = \{e_1, \ldots, e_t\}$, which stores the agent's experiences $e_t = (s_t, a_t, r_t, s_{t+1})$ to reduce correlations between observations. The experience consists of the current state s_t, the action the agent took a_t, the reward it received r_t, and the state it transitioned to s_{t+1}. The learning update at each iteration i uses a loss function based on the temporal-difference update:

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}} \left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i)\right)^2\right]$$

where θ_i and θ_i^- are the parameters of the Q-networks and the target network respectively at iteration i, and the experience samples (s, a, r, s') are sampled uniformly from $\mathcal{D}$. In partially observable domains where only observations o_t are available at time t instead of the entire state s_t, the experience takes the form $e_t = (o_t, a_t, r_t, o_{t+1})$. One of the limitations of DQN is that it cannot easily handle continuous action spaces.

3.0.2 Deep Deterministic Policy Gradient

Deep deterministic policy gradient (DDPG) combines the actor-critic and DQN approaches to learn policies in continuous action spaces. The goal in DDPG is to maintain a parameterized actor function $\mu(s \mid \theta_\mu)$, which deterministically maps states to actions while learning a critic $Q(s, a)$ that estimates the value of state-action pairs. The actor can be updated with the following optimization step:

$$\nabla_{\theta_\mu} J \approx \mathbb{E}_{s_t\sim\rho_\pi}[\nabla_a Q(s, a \mid \theta_Q)|_{s=s_t, a=\mu(s_t)} \nabla_{\theta_\mu}\mu(s \mid \theta_\mu)|_{s=s_t}]$$

where ρ_π are transitions generated from a stochastic behavior policy π, typically represented with a Gaussian distribution centered at $\mu(s \mid \theta^\mu)$.

3.0.3 Trust Region Policy Optimization

Trust region policy optimization (TRPO) [33] is a policy gradient method that allows precise control of the expected policy improvement during the optimization step. At each iteration k, TRPO aims to solve the following constrained optimization problem by optimizing the stochastic policy π_θ:

$$\begin{aligned} \underset{\theta}{\text{Maximize}} \quad & \mathbb{E}_{s\sim\rho_{\theta_k}, a\sim\pi_{\theta_k}}\left[\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A_{\theta_k}(s,a)\right] \\ \text{subject to} \quad & \mathbb{E}_{s\sim\rho_{\theta_k}}\left[D_{KL}(\pi_{\theta_k}(\cdot|s) \| \pi_\theta(\cdot|s))\right] \le \Delta_{KL} \end{aligned}$$

where $\rho_\theta = \rho_{\pi_\theta}$ are the discounted state-visitation frequencies induced by π_θ. $A_{\theta_k}(s, a)$ is the advantage function, which can be estimated by the difference between the empirical returns and the baseline. We use a linear value function baseline in our experiments. D_{KL} is the KL divergence between the two policy distributions, and Δ_{KL} is a step size parameter that controls the maximum change in policy per optimization step. The expectations in the expression can be evaluated using sample averages, and the policy can be represented by non-linear function approximators such as neural networks. The stochastic policy π_θ can be represented by a categorical distribution when the actions of the agent are discrete and by a Gaussian distribution when the actions are continuous.
We also found it useful to supplement these ideas from single agent reinforcement learning with reward transformations and curriculum learning.

3.0.4 Reward Structure

There can be a benefit in assigning rewards to agents that does not necessarily match the actual objective. The concept of reward shaping [26] involves modifying rewards to accelerate learning without changing the optimal policy and approximate Bayesian methods that add bonus reward to the objective reward to achieve optimism under uncertainty [17, 35]. When modeling a multi-agent system as a Dec-POMDP, rewards are shared jointly by all agents. In a centralized representation, the reward signal cannot be decomposed into separate components, and is equivalent to the joint reward in a Dec-POMDP. However, decentralized representations can allow us an alternative local reward representation allowing us to

assign credit in a more fine-grained manner. Local rewards can restrict the reward signal to only those agents that are involved in the success or failure at a task. [2] have shown that such local information can help reduce the number of samples required for learning. As we will note later, this decomposition can drastically improve training time. The results are, however, still evaluated using the global reward.

3.0.5 *Curriculum Learning*

Many reinforcement learning tasks are difficult to learn from scratch. It is often easier to learn a simple task first, and then build on that knowledge to solve the difficult task. This idea is known as curriculum learning [4], and it can be extended to reinforcement learning problems with multiple cooperating agents. Formally, a curriculum $\mathcal{T}$ is an ordered set of tasks organized by increasing difficulty. In cooperative multi-agent settings, the tasks in the curriculum become more difficult as the number of cooperating agents required to complete the task increases.

4. MULTI-AGENT DEEP REINFORCEMENT LEARNING

This section presents three training schemes for reinforcement learning that are applicable to multi-agent domains. We outline the advantages and disadvantages of each approach, and describe how each can be combined with deep reinforcement learning.

4.1 Centralized

The centralized learning approach assumes a joint model for the actions and observations of all the agents. A centralized policy maps the joint observation of all the agents to a joint action, and is equivalent to an MPOMDP policy. A major drawback of this approach is that it is centralized in both training and execution, and leads to an exponential growth in the observation and actions spaces with the number of agents. We address this intractability in part by factoring the action space of centralized multi-agent systems.

We first assume that the joint action can be factored into individual components for each agent. The factored centralized controller can then be represented as a set of independent sub-policies that map the joint observation to an action for a single agent. In the policy gradient approach this reduces to factoring the joint action probability as $P(\vec{a}) = \prod_i P(a_i)$ where a_i are the individual actions of an agent. In practice, this means that the output of our neural network policy has to capture only the action distributions for each individual agent rather than the joint action distributions for all the agents. In systems with discrete actions, this reduces the size of the action space from $|\mathcal{A}|^n$ to $n|\mathcal{A}|$, where n is the number of agents and $\mathcal{A}$ is the action space for a single agent (we assume homogeneous agents for simplicity). While this is a significant reduction in the size of the action space, the exponential growth in the observation spaces ultimately makes centralized controllers impractical for complex cooperative tasks.

4.2 Concurrent

In concurrent learning, each agent learns its own individual policy. Concurrent policies map an agent's private observation to an action for that agent. Each agent's policy is independent. In the policy gradient approach, this means optimizing multiple policies simultaneously from the joint reward signal. One of the advantages of this approach is that it makes learning of heterogeneous policies easier. This can be beneficial in domains where agents may need to take on specific roles in order to coordinate and receive reward.

There are two major downsides to the concurrent training approach. First, training unique policies does not scale to large numbers of agents. Because the agents do not share experience with each other, this approach adds additional sample complexity to the reinforcement learning task. During training, the approach requires a policy for each agent, which can add significant computational and memory burdens when the policies are represented by complex models like neural networks. Second, as the agents are learning and adjusting their policies, the change in the policies make the environment dynamics non-stationary. This could lead to instability, and is particularly problematic in deep reinforcement learning approaches based on experience replay. Stored experiences can be quickly rendered obsolete due to the changing dynamics of other agents.

Algorithm 1 PS-TRPO

Input: Initial policy parameters Θ_0, trust region size Δ
for $i \leftarrow 0, 1, \ldots$ **do**
  Rollout trajectories for all agents $\vec{\tau} \sim \pi_{\theta_i}$
  Compute advantage values $A_{\pi_{\theta_i}}(o^m, m, a^m)$ for each agent m's trajectory element.
  Find $\pi_{\theta_{i+1}}$ maximizing Eq. (1)
    subject to $\overline{D}_{KL}(\pi_{\theta_i} \| \pi_{\theta_{i+1}}) \leq \Delta$

4.3 Parameter Sharing

If the agents are homogeneous, their policies may be trained more efficiently using parameter sharing. In the parameter sharing approach, we allow all the agents to share the parameters of a single policy. This allows the policy to be trained with the experiences of all agents simultaneously. However, it still allows different behavior between agents because each agent receives different observations, which includes their respective index. In this work, we focus on the decentralized parameter sharing training protocol because we found it to be the most scalable approach out of the three described in this section. In this protocol, the control is decentralized but the learning is not. In the remainder of the paper, all training schemes are assumed to be parameter sharing unless stated otherwise.

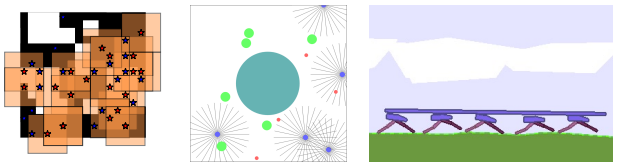
Algorithm 1 describes a policy gradient version of the parameter sharing training approach based on single agent TRPO [33]. We first initialize the policy network and set the step size parameter. At each iteration of the algorithm, the decentralized policy is used to sample trajectories from each agent. The batch of trajectories from all the agents is used to compute the advantage value and maximize the following objective:

$$L(\theta) = \mathbb{E}_{o \sim \rho_{\theta_k}, a \sim \pi_{\theta_k}} \left[\frac{\pi_\theta(a \mid o, m)}{\pi_{\theta_k}(a \mid o, m)} A_{\theta_k}(o, m, a) \right] \quad (1)$$

where m is the agent index. The results of the optimization are used to compute the parameter update for the policy. So long as the agents can execute decentralized policies with shared parameters, both DDPG and DQN can be extended to multi-agent systems in a similar manner.

5. EXPERIMENTS

This section presents empirical results that compare the performance of multi-agent extensions of TRPO, DDPG, and DQN in cooperative settings. We present three tasks that aim to demonstrate coordination in different domains including tasks with both discrete and continuous control: pursuit-evasion, waterworld, and coordinating bipedal walkers. For discrete action tasks, we compare TRPO to DQN, and for continuous action tasks we compare TRPO to DDPG. For the policy gradient approach, we examine a few neural network architectures including feed-forward MLP and



(a) Pursuit (b) Waterworld (c) Multi-Walker

Figure 1: Examples of three cooperative domains.

CNN for reactive policies and GRU for recurrent policies. We also examine the effects of centralized, concurrent, and decentralized training schemes as well as two reward mechanisms that are relevant to multi-agent domains. The results are compared against each other and against a heuristic hand-crafted baseline for each task. We also compare the performance of PS-TRPO to a traditional Dec-POMDP dynamic programming approach on a small, discrete problem.

We consider three types of neural network architectures in this work — fully connected, recurrent, and convolutional. With policy gradients, for both discrete and continuous control tasks we use a fully connected multi-layer perceptron (MLP) network with three hidden layers consisting of 100, 50, and 25 hidden units with tanh nonlinearities. For discrete tasks, the network outputs a categorical distribution over actions while for continuous tasks, it outputs the mean of the Gaussian distribution. We also consider recurrent neural networks with 32 gated recurrent units (GRU). Although recurrent policies are more difficult to train than MLP policies, they improve performance in partially observable domains by learning a hidden state vector that represents information about the history of observations seen by the agent. We also use the MLP network described above as an observation feature extractor. These extracted features are then passed into the GRU network. For tasks with observations that can be represented with spatially correlated images, we also show the use of convolutional neural networks (CNNs).

The DDPG and DQN benchmarks train a deterministic policy represented by a feed-forward network with two hidden layers, consisting of 400 and 300 hidden units with rectified linear units as activations. In the case of DDPG, the same network architecture is used for the policy and the state-action value function.

In all our experiments, we use the discount factor $\gamma = 0.99$. For PS-TRPO, we set the step size parameter to $\Delta = 0.01$, and constrain the size of each batch to a maximum of 24000 time-steps. For DDPG and DQN, we used batch sizes of 32, learning rate of 1×10^{-3} for the state-action value function and 1×10^{-4} for the policy network.

5.1 Discrete Control Task

Pursuit is a standard task for benchmarking multi-agent algorithms [41]. The pursuit-evasion domain consists of two sets of agents: evaders and pursuers. The evaders are trying to avoid pursuers, while the pursuers are trying to catch the evaders. The action and observation spaces in this problem are discrete. The agents interact on a two-dimensional grid, and an evader is considered caught if it is surrounded by pursuers on four sides. In order to catch the evaders, the pursuers must learn to cooperate by trapping the evaders on all sides. When the pursuers catch an evader, they receive a reward. This reward was set to 5. We also needed a shaping reward of 0.01 for encountering an evader to ease exploration. For simplicity, the evaders follow a uniform random policy. The domain contains obstacles through which the agents cannot pass. Each pursuer receives a range-limited observation of its surroundings, and must choose between five actions Stay, Go East, Go West, Go South, Go North. The observations contain information about the agent's surroundings, including the location of nearby pursuers, evaders, and obstacles. The example in Fig. 1a shows a 32×32 grid world with randomly generated obstacles, 20 pursuers (denoted by red stars), and 20 evaders (denoted by blue stars). The square box surrounding the pursuers indicates their observation range.

We first compared the performance of the three training schemes with a feed-forward MLP policy using TRPO as the policy gradient approach. The results are shown in Fig. 2a for a 16×16 grid, 8 pursuers with an observation range of 7, and 30 evaders. While only 4 cooperating agents are needed to complete the task, our policy learns to break up the pursuers into teams of 4 when there are more than 4 pursuers interacting with each other. The plot shows the returns from the heuristic policy which serves as a baseline for our experiments. The heuristic policy moves the pursuer towards the closest evader if they are within observation range, otherwise it selects a random action. From Fig. 2a, it is clear that for a policy parameterized by an MLP, decentralized and concurrent training do better than centralized. Because the observation is image-like (see Fig. 5) with spatial correlations present in each observation dimension, we can leverage the ability of convolutional neural networks to better capture these correlations. In the pursuit task, the CNN policy tends to outperform the MLP policy. We also compared this behavior with a GRU network parameterized policy and found a similar trend. Decentralized training consistently outperformed centralized training. Visualizations also showed consistent attempts by the agents to herd together the evaders in a coordinated manner.

We then compared the training behavior of global and local rewards. We found that using local rewards consistently improved convergence during training. An example of this difference for the pursuit evasion problem is shown in Fig. 4.

We compared the behavior of DQN against our policy gradient approach PS-TRPO. As can be seen from Fig. 3, DQN does not perform as well as the policy gradient based PS-TRPO. We hypothesize that an important factor in DQN's inability to learn a good controller in the decentralized setting is the non-stationarity of the problem.

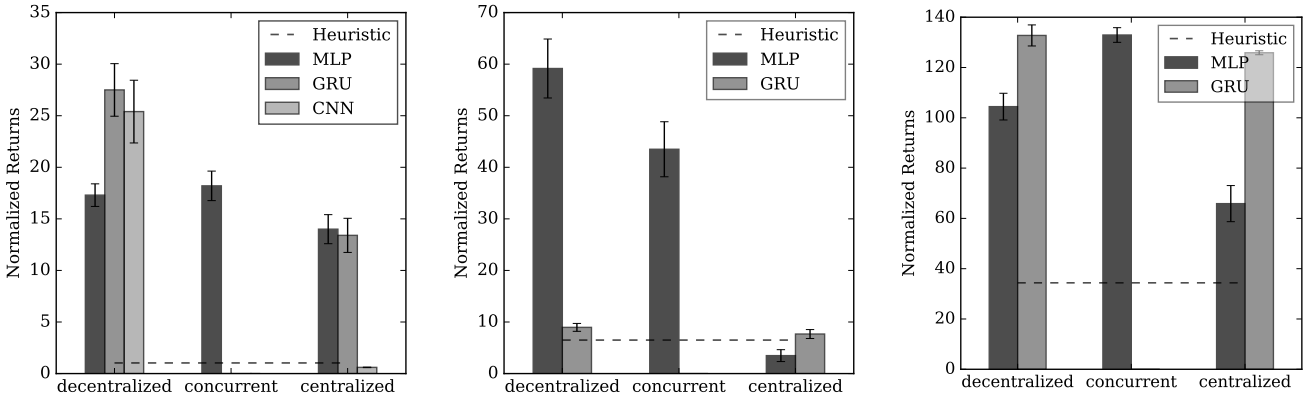
5.2 Comparison to Traditional Method

Traditional reinforcement learning and Dec-POMDP approaches find it difficult to handle problems with continuous action spaces and scale to problems with large numbers of agents. We compare the performance of PS-TRPO against a traditional approach for solving Dec-POMDPs based on Joint Equilibrium search for policies (JESP) [25] on a small-scale pursuit problem. The approach we use as comparison resembles JESP in that it finds a policy that maximizes the joint expected reward for one agent at a time, while keeping the policies of all the other agents fixed. The process is repeated until an equilibrium is reached. In our approach, we use the fast informed bound (FIB) algorithm [14] to perform the policy optimization of a single agent.

The pursuit problem is set on a 5×5 grid with a square obstruction in the middle. There is a single evader and two pursuers. Both of the pursuers must occupy the same location as the evader in order to catch it and obtain a reward. This problem has a total of

Table 1: Average returns on small-scale pursuit problem

	PS-TRPO	FIB
Average Returns	9.36 ± 0.52	9.29 ± 0.65



(a) Pursuit (b) Waterworld (c) Multi-Walker

Figure 2: Normalized average returns for multi-agent policies. A random policy has zero normalized average return. Error bars represent standard error. Results from Wilcoxon test suggest that these differences are significant ($p < 0.05$) except for the difference between centralized GRU and decentralized GRU for the waterworld domain.

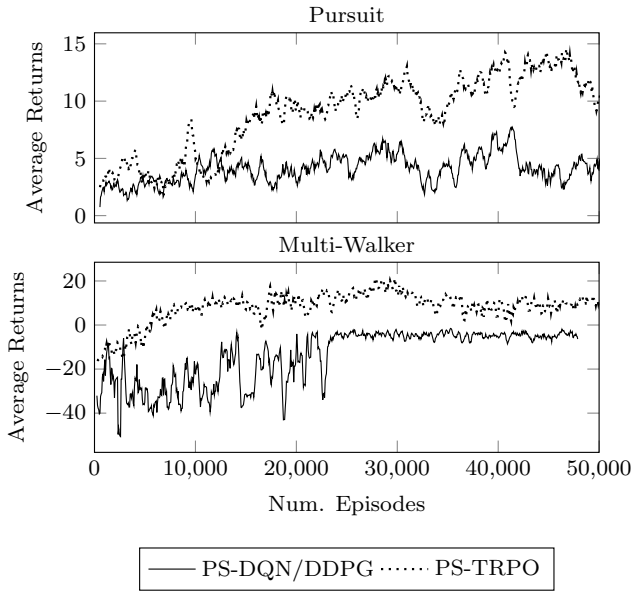


Figure 3: Average returns comparing PS-TRPO and DQN/DDPG in Pursuit and Multi-Walker Domains.

15625 states and 729 observations. The results comparing the average performance and their standard errors of PS-TRPO and FIB policies averaged over 100 simulations are shown in Table 1. The results demonstrate that PS-TRPO performs as well as the traditional approaches on the small problem, and has the ability to scale to large and continuous spaces.

5.3 Continuous Control Tasks

We next evaluated the algorithm's performance on continuous control tasks. First, we extended the above mentioned pursuit-evasion problem to a continuous domain. The extension is based on the single agent waterworld domain used by [15] . In this task, agents need to cooperate to capture moving food targets while avoiding poison targets. Both the observation and action spaces are continuous, and the agents move around by applying a two-dimensional force. Each agent has 30 range-limited sensors facing outward with uniform angular spacing. The sensors are used to make distance and velocity measurements of other agents, food targets and poison targets, resulting in a 212-dimensional observation for each agent. The agents receive a reward each time they collaborate to capture a food target, and are penalized when they collide with a poison target. They also receive an action penalty defined as the square norm of the force applied. In our experiments, the food reward was set to 10 and the poison penalty was -1. We also added a reward of 0.01 for encountering the food targets to ease exploration.

We used policy gradients to compare the proposed training schemes and found that decentralized and concurrent approaches outperformed centralized training for continuous tasks as well (Fig. 2b). The policies were trained in environments with 16 agents, where at least 8 agents need to cooperate to catch a food target. The waterworld heuristic policy moves the agents towards a food target or an ally and away from the poison target. We also found that a reactive policy performed better than a recurrent one. We believe this is caused by the difficulty of training recurrent networks compared to simpler feedforward ones with high-dimensional observations.

Finally, we tested our algorithm on a more difficult continuous control locomotion task that is based on the BipedalWalker environment from OpenAI gym [7]. The domain consists of multiple bipedal walkers that can actuate the joints in each of their legs. At the start of each simulation, a package is placed on top of the walkers. The package is large enough that it stretches across all of the walkers, and is too large for a single walker to move on its own. The walkers must learn how to move forward and to coordinate with other agents in order to keep the package balanced while navigating a complex terrain. An example environment with five walkers is shown in Fig. 1c.

The multi-walker domain also consists of continuous observation and action spaces. The walkers control torques in the two joints on each of their legs for a total of 4 action variables. Each walker receives noisy lidar measurements of the terrain as well as displacement information about its two neighboring walkers and the package for a total of 32 observation variables. The agents are rewarded for moving the package forward, and are penalized when the package touches the ground. An episode ends when the walkers reach the edge of the terrain or when the package touches the ground. In our experiments, dropping the package had a penalty of

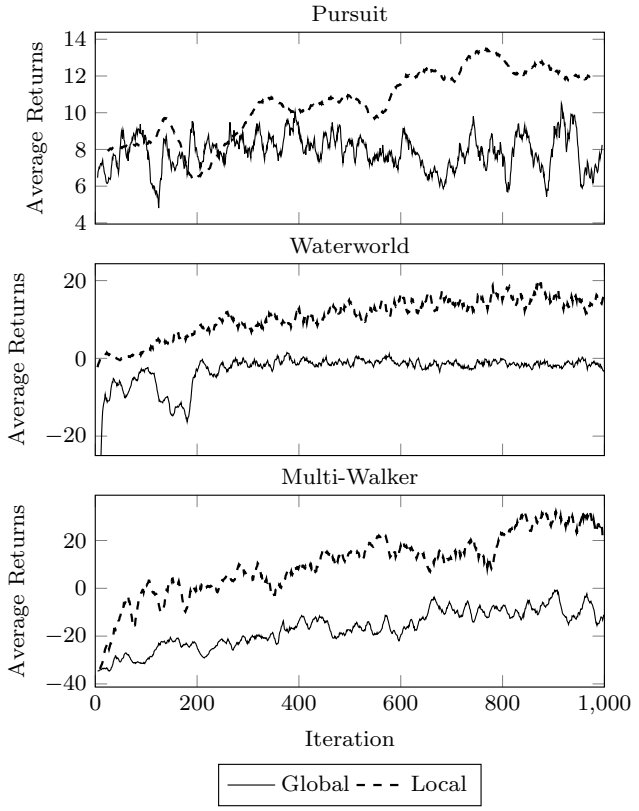


Figure 4: Average returns for decentralized multi-agent policies with global and local rewards.

-100 while moving forward had a reward of 1.

We set the number of agents in the environment to 3 and use TRPO to learn stochastic policies parameterized by either MLP or GRU with different training schemes (see Fig. 2c). Our experiments again confirmed that decentralized and concurrent schemes outperform the centralized training scheme. We see a remarkable improvement in performance between the GRU and MLP policy using the decentralized behavior, indicating that taking the observation history into account is important in the multi-walker domain. Visualizing the policies showed consistent intelligent behavior in the agents coordinating and pushing the box forward without letting it fall down.

We also compared DDPG against our PS-TRPO approach and again found that PS-TRPO based on policy gradient worked better

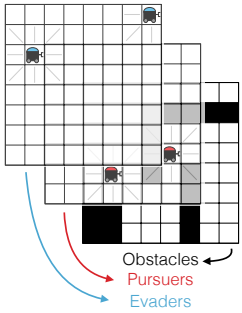


Figure 5: Image like representation of an observation in the pursuit evasion domain. The locations of each entity (pursuers, evaders, and obstacles) are represented as bitmaps in their respective channels.

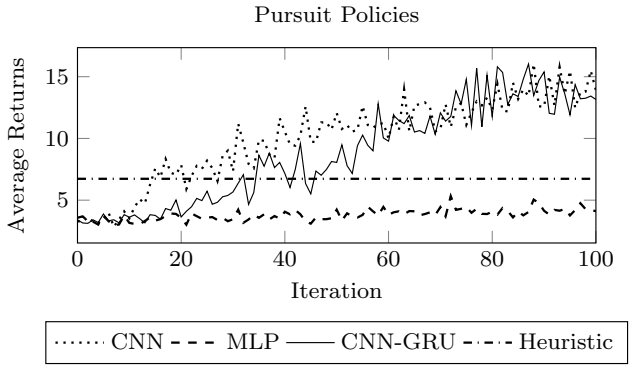


Figure 6: Performance as a function of the number of iteration for different neural architectures in the pursuit domain with 200 agents. At least 16 agents need to occupy the same cell to capture an evader.

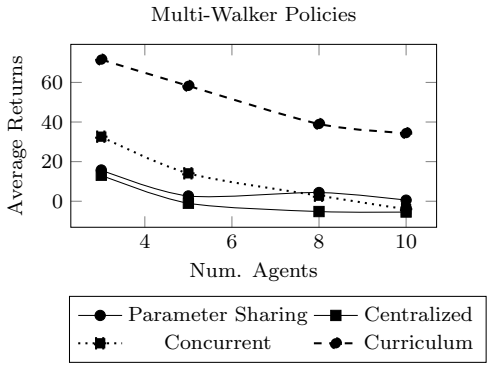


Figure 7: Performance of multi-walker policies as a function of the number of agents during training. Each data point in the decentralized, centralized, and concurrent curves was generated by training and evaluating a policy with a fixed number of agents. The curriculum curve was generated by evaluating a single policy with varying number of agents.

than DDPG, as can be seen in Fig. 3.

5.4 Scaling

We next studied how well the decentralized method scales to larger observation spaces and many agents.

For the discrete pursuit-evasion problem, we first test PS-TRPO on a large 128×128 environment with 200 pursuers and evaders with at least 16 agents required to capture an evader. While a large number of agents are present in the environment, only 16 of them need to cooperate to achieve the capture task. Each observation is a four channel 21×21 image, making the observation space 1764 dimensional. To test the behavior when a large number of agents need to cooperate, we modified the task so that agents receive a reward when the cooperating agents all occupy the same cell to capture an evader. However, we found the reactive policy learned with an MLP representation to be unsatisfactory. Therefore, to leverage the ability of a convolutional neural network to capture spatial locality in the input space, we represent the local observations of an agent as an image with nearby pursuers, evaders, and obstacles presented as different channels in the image. As can be seen in Fig. 6, this representation allows us to scale our method to a fairly large number of cooperating agents and results in intelligent behavior.

We also studied how the performance of cooperative policies scales with increasing number of agents for different training schemes. Figure 7 shows the degrading performance of PS-TRPO

Algorithm 2 Curriculum Training

Input: Curriculum $\mathcal{T}$, Iteration n, Policy π_Θ, $r_{\text{threshold}}$
$\alpha_\mathcal{T} \leftarrow [\text{length}(\mathcal{T}), 1, 1, \ldots]$
while $r_{\text{min}} < r_{\text{threshold}}$ **do**
  {Sample task from the task distribution.}
  $w \sim \text{Dirichlet}(\alpha_\mathcal{T})$
  $i \sim \text{Categorical}(w)$
  {Apply optimization step for a few iterations.}
  PS-TRPO $(\mathcal{T}_i, \pi_\theta, n)$
  {e_{curr} is the task with the highest weight $\alpha_\mathcal{T}$.}
  $r_{e_{\text{curr}}} \leftarrow \text{Evaluate}(\pi_\theta, e_{\text{curr}})$
  if $r_{e_{\text{curr}}} > r_{\text{threshold}}$ **then**
    Circular shift $\alpha_\mathcal{T}$ weights to the next task
  {Find the minimum average reward across tasks.}
  $r_{\text{min}} \leftarrow \min_\mathcal{T} \mathbb{E} r_\mathcal{T}$

policies as the number of agents increases. The decreases in performance are in part due to the increasing difficulty of the reinforcement learning task as the number of cooperating agents grows.

We investigated how a curriculum learning scheme can help scale the multi-walker problem in the number of agents. An intuitive curriculum for this problem is over the number of agents, and so we define a curriculum with the number of agents in the environment ranging from 2 to 10. Because the policies are decentralized, they can be evaluated on tasks with any number of cooperating agents regardless of the number of cooperating agents present during training. Unfortunately, we found that decentralized policies trained on a few agents often fails to generalize to larger numbers of agents. We therefore define a Dirichlet distribution for this range of tasks with higher probability assigned to the simplest task (with 2 agents). We then sample an environment from this distribution over the tasks in the curriculum and optimize the policy with PS-TRPO for a few iterations. Once the expected reward for the most likely environment reaches a threshold, we change the distribution such that the next environment is most likely. We continue this curriculum until the expected reward in all environments reaches the defined threshold. Algorithm 2 describes this process. As can be seen in Fig. 7, the resulting policy outperforms policies trained without the curriculum. We believe this improvement in performance is due to two reasons: a) The distribution over environments provides a regularization effect, helping avoid local minima during optimization, and b) It partially addresses the exploration problem by smoothly increasing the difficulty of the policy to be learned.

One potential issue with this experiment is that the curriculum scheme observed more episodes than the ones without curriculum. However, we believe that it's still an improvement because we do see convergence, albeit to a worse optima, while learning without curriculum as well.

6. CONCLUSION

Despite the advances in decentralized control and reinforcement learning over recent years, learning cooperative policies in multi-agent systems remains a challenge. The difficulties lie in scalability to high-dimensional observation spaces and to large numbers of agents, accommodating partial observability, and handling continuous action spaces. In this work, we extended three deep reinforcement learning algorithms to the cooperative multi-agent context, and applied them to three high-dimensional, partially observable domains with many agents.

Our empirical evaluations show that PS-TRPO policies have substantially better performance than DQN and DDPG in collaborative multi-agent domains. We suspect that DQN and DDPG perform poorly in systems with multiple learners due to the non-stationarity of the system dynamics caused by the changing policies of the agents. The non-stationary nature of the system makes experience replay samples obsolete and negatively impacts training. One potential way to address this problem is by disabling experience replay and instead relying on asynchronous training [23]. A more detailed investigation is left for future work. Finally, we presented empirical results on three multi-agent learning problems, demonstrating that PS-TRPO can scale to systems with large numbers of agents and perform well in domains with continuous action spaces.

There are several areas for future work. To improve scalability of the proposed approach for larger numbers of cooperating agents further work is needed. Two major challenges in multi-agent systems are accommodating reward sparsity through intelligent domain exploration and incorporating high-level task abstractions and hierarchy [31]. These are acute forms of similar challenges in the single agent learning. Recently, curiosity based information gain maximizing exploration strategy was explored by [16]. Similar ideas could be adapted to maximize information gain not only about the environment's dynamics, but the dynamics of an agent's behavior as well. Correspondingly, hierarchical value functions were integrated with deep reinforcement learning [18]. Incorporating task hierarchies in a multi-agent system would allow us to tackle learning specialization and heterogeneous behavior.

The source code for this work can be found at `https://github.com/sisl/MADRL`.

7. ACKNOWLEDGEMENTS

This work was supported by Army AHPCRC grant W911NF-07-2-0027. The authors would like to thank the anonymous reviewers for their helpful comments.

REFERENCES

[1] C. Amato, G. Chowdhary, A. Geramifard, N. K. Ure, and M. J. Kochenderfer. Decentralized control of partially observable Markov decision processes. In *IEEE Conference on Decision and Control (CDC)*, Florence, Italy, 2013.

[2] D. Bagnell and A. Y. Ng. On local rewards and scaling distributed reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 91–98, 2005.

[3] B. Banerjee, J. Lyle, L. Kraemer, and R. Yellamraju. Sample bounded distributed reinforcement learning for decentralized pomdps. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2012.

[4] Y. Bengio, J. Louradour, R. Collobert, and J. Weston. Curriculum learning. In *International Conference on Machine Learning (ICML)*, pages 41–48, 14 June 2009.

[5] D. S. Bernstein, S. Zilberstein, and N. Immerman. The complexity of decentralized control of Markov decision processes. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 32–37. Morgan Kaufmann, 2000.

[6] D. Bloembergen, K. Tuyls, D. Hennes, and M. Kaisers. Evolutionary dynamics of multi-agent learning: a survey. *Journal of Artificial Intelligence Research*, 53:659–697, 2015.

[7] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. OpenAI Gym, 2016.

[8] L. Busoniu, R. Babuska, and B. D. Schutter. Multi-Agent reinforcement learning: A survey. In *International Conference on Control, Automation, Robotics and Vision*, volume 527, pages 1–6, Dec. 2006.

[9] J. Cortes, S. Martinez, T. Karatas, and F. Bullo. Coverage control for mobile sensing networks. In *IEEE International Conference on Robotics and Automation (ICRA)*, volume 2, pages 1327–1332. IEEE, 2002.

[10] A. K. Das, R. Fierro, V. Kumar, J. P. Ostrowski, J. Spletzer, and C. J. Taylor. A vision-based formation control framework. *IEEE transactions on robotics and automation*, 18(5):813–825, 2002.

[11] F. Fernández and L. E. Parker. Learning in large cooperative multi-robot domains. *International Journal of Robotics and Automation*, 16(4):217–226, 2001.

[12] J. N. Foerster, Y. M. Assael, N. de Freitas, and S. Whiteson. Learning to communicate with deep multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems (NIPS)*, 2016.

[13] C. Guestrin, M. Lagoudakis, and R. Parr. Coordinated reinforcement learning. In *International Conference on Machine Learning (ICML)*, volume 2, pages 227–234, 2002.

[14] M. Hauskrecht. Incremental methods for computing bounds in partially observable Markov decision processes. In *AAAI Conference on Artificial Intelligence (AAAI)*, 1997.

[15] J. Ho, J. K. Gupta, and S. Ermon. Model-free imitation learning with policy optimization. In *International Conference on Machine Learning (ICML)*, 2016.

[16] R. Houthooft, X. Chen, Y. Duan, J. Schulman, F. De Turck, and P. Abbeel. Variational information maximizing exploration. *arXiv preprint arXiv:1605.09674*, 2016.

[17] J. Z. Kolter and A. Y. Ng. Near-Bayesian exploration in polynomial time. In *International Conference on Machine Learning (ICML)*, pages 513–520, 2009.

[18] T. D. Kulkarni, K. R. Narasimhan, A. Saeedi, and J. B. Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. *arXiv preprint arXiv:1604.06057*, 2016.

[19] M. Lauer and M. Riedmiller. An algorithm for distributed reinforcement learning in cooperative multi-agent systems. In *International Conference on Machine Learning (ICML)*, pages 535–542. Morgan Kaufmann, 2000.

[20] S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. *Journal of Machine Learning*, 17(39):1–40, 2016.

[21] 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.

[22] L.-J. Lin. *Reinforcement Learning for Robots Using Neural Networks*. PhD thesis, Carnegie Mellon University, 1992.

[23] 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.

[24] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[25] R. Nair, M. Tambe, M. Yokoo, D. Pynadath, and S. Marsella. Taming decentralized POMDPs: Towards efficient policy computation for multiagent settings. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2003.

[26] A. Y. Ng, D. Harada, and S. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *International Conference on Machine Learning (ICML)*, volume 99, pages 278–287, 1999.

[27] R. Olfati-Saber, J. A. Fax, and R. M. Murray. Consensus and cooperation in networked multi-agent systems. *Proceedings of the IEEE*, 95(1):215–233, 2007.

[28] S. Omidshafiei, A.-a. Agha-mohammadi, C. Amato, S.-Y. Liu, J. P. How, and J. Vian. Graph-based cross entropy method for solving multi-robot decentralized POMDPs. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2016.

[29] N. Ono and K. Fukumoto. A modular approach to multi-agent reinforcement learning. In G. Weiß, editor, *Distributed Artificial Intelligence Meets Machine Learning Learning in Multi-Agent Environments*, Lecture Notes in Computer Science, pages 25–39. Springer Berlin Heidelberg, 1997.

[30] L. Panait and S. Luke. Cooperative multi-agent learning: The state of the art. *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 11(3):387–434, 2005.

[31] R. Parr and S. Russell. Reinforcement learning with hierarchies of machines. *Advances in Neural Information Processing Systems (NIPS)*, pages 1043–1049, 1998.

[32] L. Peshkin, K.-E. Kim, N. Meuleau, and L. P. Kaelbling. Learning to cooperate via policy search. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 489–496, 30 June 2000.

[33] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. In *International Conference on Machine Learning (ICML)*, 2015.

[34] S. P. Singh, T. S. Jaakkola, and M. I. Jordan. Learning without State-Estimation in partially observable Markovian decision processes. *International Conference on Machine Learning (ICML)*, 1994.

[35] J. Sorg, S. P. Singh, and R. L. Lewis. Variance-based rewards for approximate Bayesian reinforcement learning. *CoRR*, abs/1203.3518, 2010.

[36] S. Sukhbaatar, A. Szlam, and R. Fergus. Learning multiagent communication with backpropagation. In *Advances in Neural Information Processing Systems (NIPS)*, 2016.

[37] H. Tamakoshi and S. Ishii. Multiagent reinforcement learning applied to a chase problem in a continuous world. *Artificial Life and Robotics*, 5(4):202–206, 2001.

[38] 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.

[39] M. Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *International Conference on Machine Learning (ICML)*, pages 330–337, 1993.

[40] G. Tesauro. Extending Q-learning to general adaptive multi-agent systems. In *Advances in Neural Information Processing Systems (NIPS)*, 2003.

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