Swarm Intelligence manuscript No.
(will be inserted by the editor)

Using M-Embeddings to Learn Control Strategies for Robot Swarms

Gregor H. W. Gebhardt · Maximilian Hüttenrauch · Gerhard Neumann

Received: date / Accepted: date

Abstract Neural networks usually have a predefined structure which requires that the number of inputs and outputs is known in advance. In the case of swarms this is a severe limitation, as we might not always have the same number of agents in the swarm. However, also in other situations we might have to deal with variable numbers of homogeneous observations, as for example point clouds. Furthermore, such data has usually no ordering (i.e., if we exchange two swarm agents, we still have semantically the same state of the swarm, if we exchange two points in a pointcloud, it still represents the same 3D structure) which cannot be exploited by standard neural network architectures. In this paper, we present a structure, called the deep M-embeddings which are inspired by the kernel mean embeddings and allow for a compact representation of a variable set of homogeneous inputs as a fixed size feature vector. In experimental evaluations, we show that this representation allows to learn complex policies in a multi-agent environment outperforming a standard multi-layer perceptron both in the achieved average episode return and in sample efficiency.

Keywords swarm control · reinforcement learning · policy search · representation learning

Acknowledgements Calculations for this research were conducted on the Lichtenberg high performance computer of the TU Darmstadt. This research was supported by grants from NVIDIA and the NVIDIA DGX Station.

Gregor H.W. Gebhardt
Computational Learning for Autonomous Systems
Technische Universität Darmstadt, Germany
E-mail: gebhardt@ias.tu-darmstadt.de
ORCID iD: 0000-0002-8575-069X

Maximilian Hüttenrauch
Lincoln Centre for Autonomous Systems
University of Lincoln, UK
E-mail: mhuettenrauch@lincoln.ac.uk

Gerhard Neumann
Lincoln Centre for Autonomous Systems
University of Lincoln, UK
E-mail: gneumann@lincoln.ac.uk

1 Introduction

Swarms in nature are groups of often simple animals which, as a collective, exposes a complex behavior. Examples are bees, ants and termites, or fish schools. As a collective they achieve much higher goals than an individual would be able to, such as building large structures, defending against predators, or foraging in environments of sparse nutrition (Bonabeau et al., 1999). Swarm robotics aims to build similar collectives of simple agents which can achieve higher goals if they act as a collective. Such agent are usually quite limited in their sensory and motor skills. Also the computational power of these agents is often restricted to a small micro-processor and little memory. A strength, however, is their inherent redundancy which adds robustness against failures and a high parallelization of the tasks. For the application of learning control policies, these limitations pose challenges such as an upper limit on the complexity of the learned controller. However, learning also profits from the large number of agents which allows for parallel exploration by either running slightly different control routines on each agent or by gathering different experiences for the same policy.

In this work, we want to address the problem of learning swarm policies for object manipulation using reinforcement learning (RL). Learning policies for swarms is a hard task because it usually implies a large state space and also a large action space. Both spaces need to be represented and explored in RL to find a good policy for controlling the agents. Recent development in reinforcement learning which leverages from the power of deep neural networks in learning compact feature representations allow for learning controllers in such high dimensional settings with large state and action spaces. Deep reinforcement learning (DRL) has proven to successfully learn polices directly from large dimensional observations such as images and for high-dimensional action spaces such as humanoid robots. In DRL, neural networks have been applied to either approximate a state(-action) value functions (Mnih et al., 2013, 2015; Hasselt et al., 2016) and/or the policy itself (Schulman et al., 2015a, 2017; Arulkumaran et al., 2017; Lillicrap et al., 2015). In this paper, we use an actor-critic variant of trust region policy optimization (TRPO) (Schulman et al., 2015b) to learn the swarm policy, which we discuss in more detail in Section 2.1.

Learning policies for swarms of robots to act in environments cluttered with objects using deep networks as function approximators is challenging because of two reasons. First, networks usually have a predefined structure which requires that the number of inputs and outputs is known in advance. For swarms this would require that we know the exact number of agents in the swarm before we learn the policy and later we can also apply the learned policy only to swarms with exactly the same size. Second, while a neural network itself is already highly redundant structure (the nodes in a layer are exchangeable), the state space of a swarm introduces a lot more redundancy into the problem, i.e., having agent a at position p_1 and agent b at position p_2 is for homogeneous agents equivalent to a being at p_2 and b at p_1. Furthermore, both of these issues also apply for other observations such as the objects in the environment.

In this work, we use a network structure that allows to compute a fixed size representation from a set of observations. With this network structure, we can learn functions that can take a variable number of observations as inputs. In this structure, each observation is processed individually by an arbitrary feature network, before the outputs of all observations are combined into a single feature representation. This combination of feature activations is inspired by the kernel mean embeddings of distributions Smola et al. (2007). In prior work, Hüttenrauch et al. (2018) have presented the deep mean embeddings on which we build this work. We present different variants which involve a *mean*, a *max*, or a *softmax* operation and, hence, we call them the *Deep M-Embeddings*.

While the deep M-embeddings solve the issue of a variable number of agents in the swarm or objects in the environment, we still need to compute an action for each agent. To this end, we centrally learn a decentralized policy that acts locally on the agent using a parameter sharing approach (Gupta et al., 2017). That is, we learn a single policy for all agents which takes the local observations of an agent as input and outputs the local action for the agent. Thus, the learned policy could be run locally on the individual swarm agents. The reward signal for learning this policy, however, is a global reward signal to the whole swarm. A centralized critic is learned from this reward signal and used for updating the local policy.

In experimental evaluations, we compare the variants of the deep M-embeddings with each other and to a standard multi-layer perceptron which has been used in state-of-the-art RL literature. We further show, that using the proposed network structure of the deep M-embeddings enables us to learn complex control strategies for swarms including multi-modal problems (sorting objects of different types).

1.1 Related Work

After the latest successes of DRL in single-agent learning problems, also the application of deep learning methods to multi-agent reinforcement learning (MARL) has become an area of more and more interest in the recent years. The prevalent problems of MARL is the non-stationarity of the environment which often leads to instabilities in the learning algorithms that prevent the learner to find good solutions, as well as, depending on the problem, the partial observability of the system. While most of the contributions in multi-agent DRL focus on these problems, they usually neglect the problem of the growing dimensionality and the interchangeability in the state space.

The deep mean embeddings—on which we build our work—are presented in Hüttenrauch et al. (2018). The authors successfully employ the deep mean embeddings in the policy and value function networks that are learned with TRPO in the rendezvous and pursuit evasion task. In Gebhardt et al. (2018) a swarm kernel based on the kernel mean embedding of the swarm state is presented. The kernel is used with actor-critic REPS Kupcsik et al. (2015) to learn policies for guiding the swarm with a common input signal to manipulate objects.

Zheng et al. (2017) present a simulation environment for massive multi-agent RL which uses images as representation with multiple layers for storing different entities of information. The environment is a grid world with discrete states and, thus, a representation of continuous states, which would be a limitation of images, is not necessary. Yang et al. (2018) apply mean-field theory to approach the problem of increasing dimensionality in settings with many agents. Instead of modeling the interactions with all other agents, each individual agent only considers the average effect of its local neighborhood. The derived mean-field Q-function is applied to actor-critic learning with deterministic policy gradients (Silver et al., 2014).

Many approaches aim for robustly learning the Q-function in the multi-agent setting. Lauer and Riedmiller (2000) introduce distributed Q-learning where optimistic agents only update their Q-values for positive TD-errors. Matignon et al. (2007) follow this direction but instead of neglecting negative updates of the Q-function, they introduce a hysteresis to the update. Omidshafiei et al. (2017) approach the problem of multi-task multi-agent reinforcement learning by combining hysteretic Q-learning with deep recurrent Q-networks (DRQN) using concurrent experience replay memory. DRQNs (Hausknecht and Stone, 2015) extend the deep Q-networks (Mnih et al., 2015, 2013) to the partially observable setting by adding a recurrent LSTM (Hochreiter and Schmidhuber, 1997) layer to the network

architecture. By further applying policy distillation (Rusu et al., 2015), Omidshafiei et al. (2017) can combine expert Q-networks of each agent into a multi-task policy. A problem of this approach is that the trajectories stored in the experience replay memory get outdated because of the changing behavior of the other agents. To approach this problem, Palmer et al. (2018) introduces leniency (Panait et al., 2006) for controlling the influence of negative policy updates from the experience replay memory.

Sunehag et al. (2017) learns agent policies with joint reward signal using a decomposition of the swarm value function into agent value functions. The centralized Q-function is the additive composition of the agent Q-functions. Thus, by learning the centralized Q-function, the agent Q-functions are learned. Rashid et al. (2018): builds on this work, but instead of additive composition only requires monotonicity in the selection of the optimal action. Thus allowing for more complex agent value functions.

Gupta et al. (2017) investigate the application of prominent DRL methods (DQN, DDPG, A3C, TRPO) to the multi-agent setting by introducing parameter sharing. In this work, we use this approach with TRPO, but since we assume homogeneous agents, we omit the agent index. Lowe et al. (2017) present the multi-agent deep deterministic policy gradient (MADDPG), an extension of the actor-critic DDPG (Lillicrap et al., 2015) to the multi-agent scenario by learning a centralized Q-function as critic, while updating the policies locally. Similarly, Foerster et al. (2017) introduce the counterfactual multi-agent (COMA) policy gradients. COMA uses a centralized critic that takes the joint action and uses a counterfactual baseline. This baseline is separate for each agent and uses counterfactuals in which only the agents action changes to improve the assessment of the impact of the agents action on the reward signal. Peng et al. (2017) introduce a multiagent bidirectionally-coordinated network (BiCNet) with a recurrent structure that allows for information sharing between agents. However, the proposed learning method requires individual rewards instead of a global reward signal.

Grover et al. (2018) present a framework for learning representations of policies in a multi-agent setting using an encoder-decoder structure. The representations are learned from observed trajectories and are used to characterize, imitate, and adapt to the other agent's behavior. Similarly, Al-Shedivat et al. (2018) introduce a gradient-based meta-learning algorithm based on (Finn et al., 2017) to quickly adapt the agent's policy to the opponent's behavior in a non-stationary competitive scenario. Other approaches use genetic algorithms such as particle swarm optimization to learn the policies for robot swarms (Pugh and Martinoli, 2006, 2007).

2 Preliminaries

We use deep reinforcement learning for obtaining the control policies. In the following paragraphs we shortly discuss the policy gradient method we applied, and the network structure we used for approximating policy and value function.

2.1 Trust Region Policy Optimization

In reinforcement learning (RL), the problems which we want to optimize are usually given as a Markov decision process (MDP). An MDP is defined by the tuple $(\mathcal{S}, \mathcal{A}, P, r, \rho_0, \gamma)$, where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, P is the state transition model (usually defined as a probability distribution $P(s'|s,a)$), r is the reward function, ρ_0 is the initial

state distribution, and γ is the discount factor. In each state $s_t \in \mathcal{S}$, an agent choses an action according to a (stochastic) policy $\pi(a_t|s_t)$. Applying this action results in a transition from state s_t to s_{t+1} according to $P(s_{t+1}|s_t, a_t)$ for which the agent receives the reward $r(s_t, a_t, s_{t+1})$. Reinforcement learning (RL) aims to find the policy π^* that maximizes the expected discounted reward of the agent

$$\mathbb{E}_{a_{0:T}, s_{0:T}} \left[\sum_{t=0}^{T} \gamma^t r(s_t, a_t, s_{t+1}) \right]. \tag{1}$$

If we assume a policy function parameterized by the parameter vector $\vec{\theta}$, we can find the optimal policy by applying a policy gradient method. Policy gradient methods try to optimize a parametric policy by following the gradient of the expected return with respect to the policy parameters. Directly following the gradient, however, can lead to disastrous behavior as small changes in the parameter space might lead to large changes in the state-action distribution $p(s, a)$. To circumvent this problem, the update step is usually subject to constraints on the divergence of the policy to ensure that the policy changes slowly and gracefully (Kakade, 2001; Peters et al., 2005). In this work, we use Trust Region Policy Optimization (TRPO) (Schulman et al., 2015a). By applying a constraint on the Kullback-Leibler (KL) divergence between the old policy $\pi_{\theta_{\text{old}}}$ and the new policy π_θ and doing further approximations to the optimization problem, TRPO optimizes the objective

$$\begin{aligned} J(\theta) &= \mathbb{E}\left[\frac{\pi_\theta(s, a)}{\pi_{\theta_{\text{old}}}(s, a)} \hat{A}(s, a) \right] \\ \text{s.t.} \quad & \mathbb{E}\left[D_{\text{KL}}\left(\pi_\theta(s, a) || \pi_{\theta_{\text{old}}}(s, a) \right) \right] \leq \delta. \end{aligned} \tag{2}$$

Here, $\pi_{\theta_{\text{old}}}(s, a)$ is the policy with the old set of parameters, $\hat{A}(s, a)$ denotes the advantage function and D_{KL} is the KL divergence which is bound by a hyper-parameter δ.

The advantage of an action a in state s is the difference between the expected future return for taking a in state s and from then following the policy π over directly following π from state s. It can be expressed as the difference between the state-action value function $Q^\pi(s, a)$ and the state value function $V^\pi(s)$, i.e.,

$$A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s) \tag{3}$$

The estimator of the advantage function $\hat{A}(s, a)$ can be obtained from samples collected during roll-outs of the policy with the old parameters using *generalized advantage estimation* (GAE) (Schulman et al., 2015b). Similar to TD(λ), a parameter λ allows to scale between temporal-difference (TD) ($\lambda = 0$) and Monte-Carlo ($\lambda = 1$) estimates. While TD updates have less variance, they are highly biased by the current estimate of the value function. In contrast, Monte-Carlo estimates are unbiased but have a large variance (Sutton et al., 1999; Sutton and Barto, 2018).

TRPO solves the constraint optimization problem by linearizing the objective and uses the Hessian of the KL as approximation to the covariance of the gradients. As neural networks are considered as approximator of the policy function, the parameter vector $\vec{\theta}$ is usually very high dimensional. Using the conjugate gradient algorithm alleviates this problem. Additionally, a line search along the found gradient ensures that the objective improves and that the constraints are met. GAE requires the estimation of a value function $V(s)$ for estimating the advantages. This value functions is optimized using a similar objective as in (2) (Schulman et al., 2015b).

2.2 Neural Networks as Function Approximator for Policy and Value

We use neural networks (NN) to approximate the policy and the value function. Neural networks consist of multiple layers of linear mappings, with non-linear activations, i.e.,

$$\vec{h}_i = \sigma(\vec{W}\vec{h}_{i-1} + \vec{b}), \qquad i = 1, \ldots, m \tag{4}$$

where $h_i \in \mathbb{R}^{d_i}$ is the output of the i-th layer, $\vec{W} \in \mathbb{R}^{d_i \times d_{i-1}}$ is a weight matrix, $\vec{b} \in \mathbb{R}^{d_i}$ a bias vector, σ a non linear activation function, and d_i is the number of neurons in the layer (or the size of the layer). The input $\vec{h}_0$ to the first layer is the input to the neural network, in our case the observation of the environment. The choice of activation functions is wide, however most DRL applications use the *tanh* function or the (leaky) rectified linear unit (RELU).

In TRPO and similar approaches (Schulman et al., 2017), policy and value approximator usually use the same network structure up to the last layer as *feature network*. Note however, that in TRPO the parameters of the networks are not shared. To obtain a value function, the output of the feature network is mapped by the last layer to a scalar value. In contrast, the policy is usually represented as a parametric distribution, where the last layer of the policy network maps to the parameters of the distribution. Hence, for a Gaussian distribution we would get

$$\pi(a_t|s_t) = \mathcal{N}(a_t; \mu(s_t), \Sigma(s_t)), \tag{5}$$

where $\mu(s_t), \Sigma(s_t)$ is given by the output of the policy network.

2.3 Mean Embeddings of Distributions

A general issue with neural networks is that the structure needs to be fixed in advance before optimizing the parameters of the network. A variable dimensionality of the inputs during or after training is hard to implement into the structure of the network. In the case of learning policies for swarms this is a critical issue since the actual number of agents in the swarm is not important as long as there are enough agents to solve the task. Moreover, as a key argument of swarm robotics is the robustness against failures due to the high redundancy, a policy should be able to deal with changes in the number of swarm agents. Furthermore, with homogeneous agents, the allocation of the specific agents to the position in the swarm is arbitrary, switching the position of two agents does not change the state of the swarm. Thus, if the representation respects this invariance to permutation (i.e. allocation to positions) and to the size of the swarm, the search space for policy and value function is drastically reduced.

In prior work (Gebhardt et al., 2018), we have leveraged the embeddings of probability distributions into reproducing kernel Hilbert spaces (RKHS) (Smola et al., 2007) to construct a kernel function that enables this invariant representation of the swarm. An RKHS $\mathcal{H}$ is uniquely defined by a positive definite kernel function $k(\vec{x}, \vec{x}') := \langle \varphi(\vec{x}), \varphi(\vec{x}') \rangle_{\mathcal{H}}$, where the feature function $\varphi(\vec{x})$ is usually intrinsic to the kernel function and maps the vector $\vec{x}$ into a potentially infinite dimensional space. We can embed a marginal distribution $p(X)$ as the expected feature mapping $\mathbb{E}_X[\varphi(\vec{x})]$. In practice, we use a sample based estimator

$$\hat{\mu}_X = \frac{1}{N} \sum_{i=1}^{m} \varphi(\vec{x}_i) = \frac{1}{N} \sum_{i=1}^{m} k(\vec{x}_i, \cdot). \tag{6}$$

In (Gebhardt et al., 2018), we have used such embeddings to represent the swarm as a distribution, where each agent is a sample from the distribution. While we cannot represent the

embedding explicitly (due to the infinite dimensionality), we can construct a kernel function which compares two swarm configurations. Hüttenrauch et al. (2018) have introduced the deep mean embeddings, which are inspired by the kernel embeddings of distributions but use a neural network to explicitly compute the feature mappings. In this work we build on the deep mean embeddings, but introduce a more broad formulation: the deep M-embeddings.

3 Deep M-embeddings

As input to the deep M-embeddings (DME), we assume N observations $o_i \in \mathbb{R}^{d_o}$ from an environment, where each observation has the same nature (e.g., each observation is the position of an agent or each observation is the location of an obstacle). Note that the number of observations, N, does not need to be fixed or predefined, as we will see later on. The DMEs allow to compute a compact representation of a variable set of homogeneous inputs to a neural network.

We consider three different types of deep DMEs: deep mean embeddings, deep max embeddings, and deep soft-max embeddings. All variants of the DMEs have in common that they map each observation $\vec{o}_i$ through a feature network $\Phi : \mathbb{R}^{d_o} \to \mathbb{R}^{d_\phi}$, yielding a feature vector $\phi(\vec{o}_i)$ as output.

The **deep mean embedding** then combines the feature vectors $\phi(\vec{o}_i)$ by computing the mean, i.e.,

$$m_j = \frac{1}{N} \sum_{i=1}^{N} \phi_j(\vec{o}_i), \qquad j = 1, \ldots, d_\phi, \tag{7}$$

where j is the index of the output vector. This embedding is the direct translation of the mean embeddings discussed in Section 2.3 using the feature vector defined by the neural network.

Instead of averaging, the **deep max embedding** takes the element wise max of the feature vectors $\phi(\vec{o}_i)$, i.e.,

$$m_j = \max_i \phi_j(\vec{o}_i), \qquad j = 1, \ldots, d_\phi. \tag{8}$$

The intuition behind this embedding is that the feature network could learn to activate certain entries of the feature vector to express details of the inputs. Rather than averaging over these activations, we might want to combine the activations similar to an operation.

The **deep soft-max embedding** computes the element-wise soft-max of the feature vectors. Using a temperature vector $\vec{\beta}$, which is a variable of the network and can be learned alongside the other network variables, the soft-max embedding allows to scale between the characteristics of the deep mean embedding and the deep max embedding for each element of the feature vector. The output of the deep soft-max embedding is given by

$$m_j = \sum_{i=1}^{N} w_{ij} \phi_j(o_i), \qquad w_{ij} = \frac{\exp\left(\beta_j \phi_j(o_i)\right)}{\sum_i \exp\left(\beta_j \phi_j(o_i)\right)}, \qquad j = 1, \ldots, d_\phi. \tag{9}$$

The deep soft-max embedding is at least theoretically preferable over the mean- and the max-embedding as it can represent both characteristics. Moreover, these characteristics can be scaled per element of the feature vector and can be learned along the other network parameters.

The network structure of a deep soft-max embedding is depicted in Figure 1. The deep mean embedding and the deep max embedding have the same structure but use a different reduction for the output of the feature network.

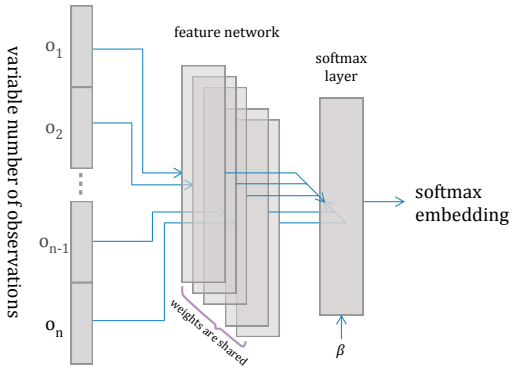

Fig. 1 Structure of the soft-max embedding network. The network takes a variable number of observations as input. Each observation is mapped by the same feature network (which can be an arbitrary network structure) to obtain feature activations for each observation. The soft-max layer combines the feature activations of all observations into a single representation. The temperature parameter $\vec{\beta}$ of the soft-max embedding is a variable of the network which can be optimized alongside the other network parameters.

4 Learning Swarm Policies

We consider the problem of object manipulation and object assembly with a swarm of *homogeneous agents*. The approach we present in this paper learns a policy that acts locally, i.e., the policy takes the *local observations* of the agent as input and returns an action for the agent, similar to Gupta et al. (2017). The problem settings in which each agent acts individually are often formulated as partially observable Markov decision process (POMDP) as the state and intentions of the other agents are unknown. Solutions to POMDPs usually require a stateful policy, where the state maintains a belief about the state of the other agents. Learning such policies requires advanced techniques with recurrent neural networks. To circumvent this problem, we assume in this paper that each agent get *full observations* of the state of the system.

These assumptions, i.e., homogeneous agents and local observations of the full state, are strong, however they provide some benefits compared to the approach of a single actor that computes joint actions for the entire swarm. First, the policies that are learned with our approach could be applied later to robots without the need for a central control unit. Second, the evaluation of the policy on the agents is parallelized by the swarm. From each step in simulation, we get multiple samples of state, action, reward and next state, depending on the number of agents in the swarm. In the next paragraphs, we present the agent model, the policy network, and the reward functions we have used for learning the swarm policies in detail.

4.1 Swarm Agents

We assume homogeneous, disc-shaped agents with single or double integrator dynamics. Our agents are inspired by the Kilobots (see Figure 2 for an example), however, the dynamics and the perceptual abilities of our agents differ substantially. While the Kilobots are actuated via two vibration motors that lead to a rotational or linear movement via the slip-stick principle (Rubenstein et al., 2014), we simulate dynamics that take directly linear and rotational

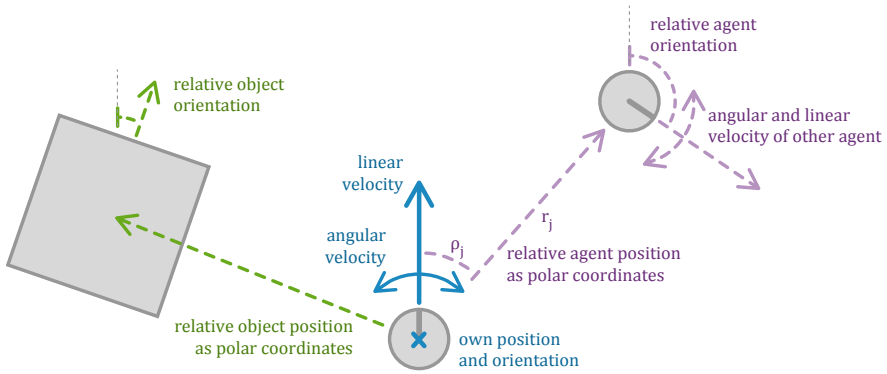

Fig. 3 Perception model of the swarm agents. The agent observes the other agent's relative positions in polar coordinates, the relative orientation (using a sin/cos transformation for the angle), and the angular and linear velocity (purple). The positions and orientations are also observed relative to the agent's state (green). Lastly, the agent observes its own position, orientation, and velocities (blue).

velocities for the single integrator or accelerations for the double integrator. With the single integrator dynamics, the state of the agent is the position as (x, y) coordinates and the orientation as angle α. The action is the two dimensional vector $a = (v_l, v_\alpha)$ with the linear and angular velocity of the agent. For the double integrator dynamics, the state of the agent is the position (x, y), the orientation α, and the velocity (v_l, v_α). The action is in this case the linear and angular acceleration (a_l, a_α).

Fig. 2 A Kilobot.

The agents in our setup observe the full state of the environment from a local perspective. The relative positions of the other agents and of the objects are observed as polar coordinates (r, ρ), where we transform the angle to $\sin(\rho), \cos(\rho)$. The orientations of other agents and objects are perceived relative to the own orientation of the agent. If we use agents with double integrator dynamics, we also observe the velocities of the other agents. To distinguish between different object types, we added the color of the objects to the observations of the agents. Finally, the agents have a proprioception of their own location, their own orientation as sin/cos transformation and in the case of double integrator dynamics also their own velocity. Figure 3 summarizes the local observations of the swarm agents. In difference to our model, the Kilobots are not able to perceive the full state of their environment, but only light intensities and an estimate of the distance to other agents in their local neighborhood.

4.2 Policy and Value Function Network

To learn policies and value function approximators in scenarios with a variable number of homogeneous observations, we propose a network structure that uses an M-embedding for each type of homogeneous observation. For example, in the scenario of object manipulation with robot swarms, the observations of the swarm agents form a set of homogeneous inputs and, similarly, the observations of the objects form another set of homogeneous inputs. We feed the remaining inputs (e.g., the proprioceptive observations of the swarm agent)

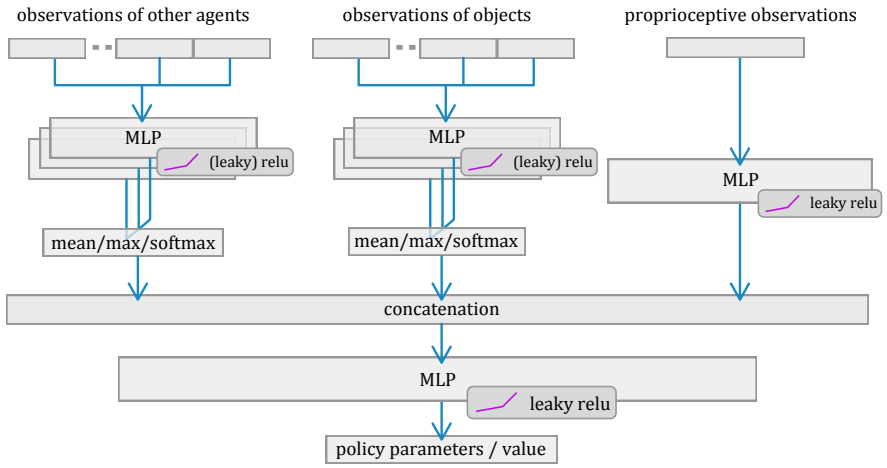

Fig. 4 Structure of the policy and value function network. The observations of the other agents and the observations of the objects are processed by an M-embedding. The proprioceptive observations are processed by an MLP. The output of the embeddings and the MLP are concatenated and mapped by another MLP to the parameters of the policy distribution or to a value of the observations.

through a set of fully connected layers. The fixed sized outputs of the M-embeddings are then concatenated together with the activations of the remaining inputs and fed through another set of fully connected layers. A schematic diagram of the network architecture is depicted in Figure 4.

The proposed network structure can then be used in any deep reinforcement learning method for approximating a value function and/or the policy with sets of homogeneous inputs. In our experimental evaluations, which we discuss in the next section, we have used an actor-critic variant of TRPO which estimates a value function from TD(λ) errors estimated using generalized advantage estimation (Schulman et al., 2015b).

5 Experimental Setup and Evaluation

In the following paragraphs, we want to present our experimental setup and discuss the results we have obtained from the evaluations of the proposed algorithm. With our experimental evaluations, we address the following questions:

1. How do the different deep M-embeddings perform in comparison to each other?
2. How does our proposed network structure perform against a simple multi-layer perceptron which has been used in state-of-the-art robot learning applications?
3. Can we solve challenging tasks of swarm robotics such as the assembly of multiple objects, or the segregation of different object types?
4. How does the β variable in the softmax-embedding change during learning?
5. Can we transfer the learned policies to different swarm sizes and a different number of objects?

Before addressing these questions in Section 5.3, we present the simulation environment, in which we have conducted our experiments, and give a short overview of the tasks and reward functions we have used for the evaluation.

5.1 The Kilobot Gym

The Kilobot Gym[1] is a simulation framework based on the OpenAI Gym (Brockman et al., 2016) which allows to evaluate reinforcement learning algorithms for swarm robotics. The framework offers two modes of operation: either the agents follow a simple, hard-coded logic, e.g., a phototactic behavior, where the swarm is controlled via a global input signal such as a light source; or the agents receive individual actions based on their local state and observation. While we have used the former mode in prior work (Gebhardt et al., 2018), we use the latter in this work to learn a policy that acts locally for the individual agent.

We simulate the physics using the 2D simulation framework Box2d (Catto, 2018) at a rate of 10 Hz. At each simulation step, the velocities for all agents are computed (either we take directly the velocity as action or we integrate the acceleration at each time step) and set to the respective bodies in the simulation environment. The simulation is stepped 20 times for each call to the environments step function which results in a time step of 2 seconds per action. Since the Kilobots are a very slow system, taking such a long time step is not an issue compared to highly dynamic systems.

5.2 Tasks and Reward Functions

We evaluate the proposed network structure on three tasks of object manipulation with robot swarms. The first task is simply to push the objects through the environment, the second task is to assemble a set of objects, and the third task is the segregation of two types of objects.

A critical part of reinforcement learning is the credit assignment of the reward to the decisive actions for the obtained reward. Techniques such as reward shaping (Ng et al., 1999) have been applied to alleviate this problem. In the multi-agent setting this problem becomes even more prevalent, since we assume global reward signals that might be induced by any of the agents actions. Thus, giving credit to the correct action selection of the individual agent becomes an even harder task. In our setting, we want to achieve a certain manipulation of the objects, hence the reward is only indirectly coupled with the action of the agent. While we use techniques such as GAE (Schulman et al., 2015b) to learn a baseline that removes a lot of variance from the estimation of the expected reward of a trajectory, we still noticed that the learning is very sensible to the selection of the reward function. For example, using an absolute reward, e.g., the distance of the object to a goal, or a sparse reward, e.g., giving a reward of 1 if the object is close enough to a goal pose, did prevent the learner from finding a good solution. Instead, we had to use relative rewards between the current state $\vec{s}_t$ and the next state $\vec{s}_{t+1}$.

*Push Objects.*The first task is to push the objects in the environment. To evaluate a state-action pair, we first compute the difference $d(o_{i,t}, o_{i,t+1})$ between the current and the next position of all objects and then take the sum

$$r_t = \sum_i d(o_{i,t}, o_{i,t+1})$$

as reward. Thus, the more an object is pushed, the higher is the reward.

[1] Code available at `https://github.com/gregorgebhardt/gym-kilobots`

Assemble Objects. In the second task, the swarm has to assemble all objects in the scene. First, we compute the point-wise distances $d(o_i, o_j)$ between all object positions at the current time step t and the next time step $t+1$, respectively. We then take the sum over the difference between the distances as reward

$$r_t = \sum_{i,j>i} d(o_{i,t}, o_{j,t}) - d(o_{i,t+1}, o_{j.t+1}).$$

Hence, if the objects approach each other this difference is positive and so is the reward.

Segregate Objects. In this last task, the goal is to segregate the objects in the scene into groups of the same kind. The reward is composed of two parts: one for assembling each group of objects with the same type, and one for separating the object groups. The first part is computed similar to the reward in the previous task, but for each group individually, i.e.,

$$r_{g,t} = \frac{1}{|g|} \sum_{i,j>i} \delta(i \in g)\delta(j \in g)(d(o_{i,t}, o_{j,t}) - d(o_{i,t+1}, o_{j,t+1})),$$

where $\delta(i \in g)$ is the indicator if object o_i belongs to group g and $|g|$ is the cardinality of group g. Similar to the previous task, this reward reflect if the objects of each group are approaching each other or not. The second part is computed using the differences of the point-wise distances of the mean positions of all groups, i.e.,

$$r_{\mu,t} = \frac{1}{n} \sum_{g,h>g} d(\mu_{g,t+1}, \mu_{h,t+1}) - d(\mu_{g,t}, \mu_{j,t}),$$

with groups g and h and the number of groups n. This reward reflects if the group center diverge. The total reward of this task is then computed as weighted sum $r_t = 1.5\frac{1}{n}\sum_g r_{g,t} + r_{\mu,t}$.

5.3 Experimental Evaluation

The swarm scenario requires different characteristics of the Deep M-Embeddings. In a first experiment, we want to evaluate how the different combinations of DMEs perform in our swarm scenario. The task was to simply move the objects and accordingly the reward function sums over the distances of the object positions between state $\vec{s}$ and $\vec{s}'$. The learning curves of the task for different combinations of DMEs are depicted in Figure 5. Most interestingly, using a mean embedding for the object observations prevents the learning of the policy. Choosing a softmax embedding for the objects together with a mean embedding or a max embedding for the swarm leads to worse performance compared to the remaining combinations (including taking a softmax-embedding for both observations) which all yield approximately similar results on this task.

Deep M-Embeddings outperform standard multi-layer perceptrons. In a second experiment, we want to compare the proposed algorithm to learning a simple multi-layer perceptron (MLP) which has been used for example to learn a policy for a humanoid in Schulman et al. (2015b). In this experiment, the task is to assemble the objects in the environment. The reward function computes the point-wise distance between the objects and returns the sum over the differences between the distances in state $\vec{s}$ and next state $\vec{s}'$. We have learned the

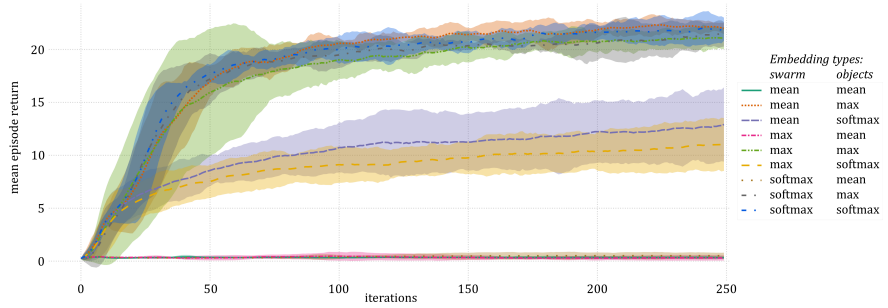

Fig. 5 Learning curves for the 'moving objects' task for all combinations of DMEs for the swarm observations and the object observations.

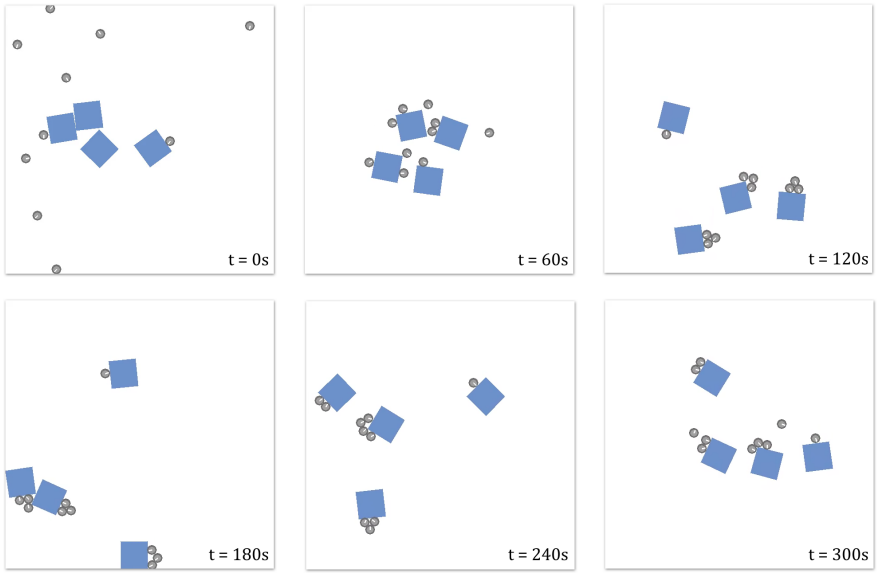

Fig. 6 Exemplary animation of the moving objects task. With the learned policy, the agents collect the objects and move them in circular trajectories across the environment. For this animation, we used a policy network with the mean-embedding for the swarm observations and a max-embedding for the object observations.

policies for this task with the combination of DMEs that have showed to yield good results in the previous experiment and with an MLP with 100, 50, and 25 neurons in three layers using tanh activations. Figure 8 depicts the learning curves for all policy types. In Figure 7, we show an exemplary animation of the task with a learned policy.

The Deep M-Embeddings can discern multiple object types. In the third experiment, we wanted to investigate if a learned policy was capable to distinguish between two object types and treat them differently. The task of this experiment is to segregate two types of objects and assemble each group individually. The agents observe the type of the object as a one-hot-encoding with the other object observations. The reward should be positive if the swarm moves objects of the same type closer together and separates them from objects of the other

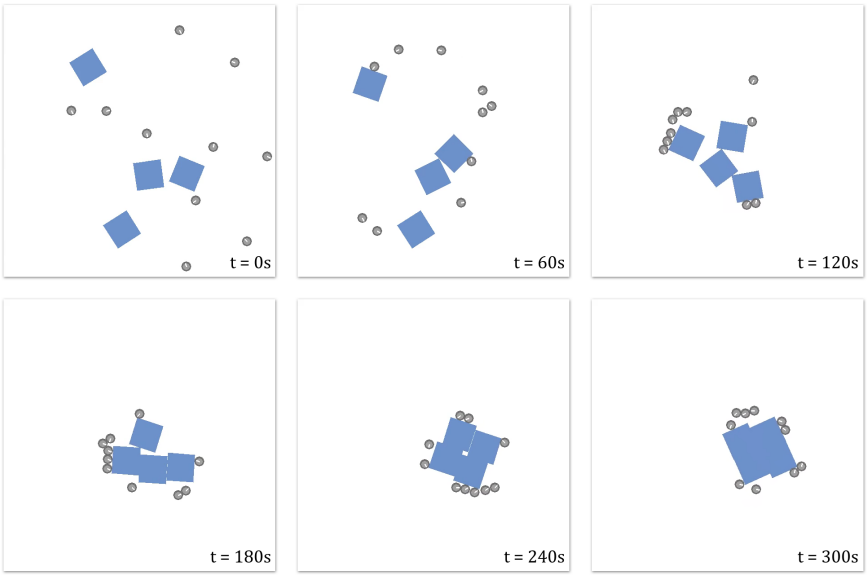

Fig. 7 Exemplary animation of the assembly task. The agents successfully assemble the objects in a rotary movement. For this animation, we used a policy network with the mean-embedding for the swarm observations and a max-embedding for the object observations.

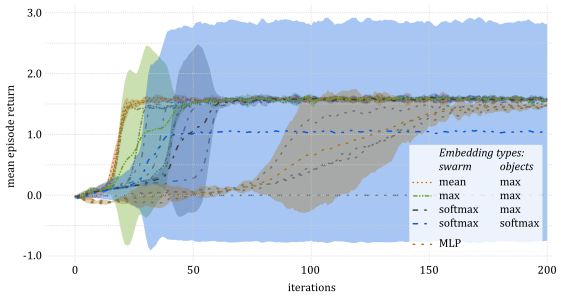

Fig. 8 Learning curves of the object assembly task for different combinations of the DMEs for swarm and object observations in comparison to a standard MLP as policy network.

type. To this end, the reward function computes for each object type the point-wise distances of the objects and the mean object position. From there, the reward is computed is then computed based on the differences in the point wise distances and the mean positions from the current to the next state. Learning curves for this experiment can be found in Figure 9. An animation of the task can be found in Figure 10.

We can see, that the policies are capable to successfully segregate the two object types. However, the agents always prefer to segregate one of the two object types and neglect the other. In general, such a task would require an hierarchical approach in which an upper hierarchy decides which object type should be segregated or a recurrent approach in which the policy is able to make long term decisions. We leave such an approach for future work.

The β-values of the soft-max embedding change during learning. In addition, we have inspected the β-values of the softmax-embedding in this experiment. Figure 11 depicts the β-values of the value function network and the policy network, where we have used a softmax

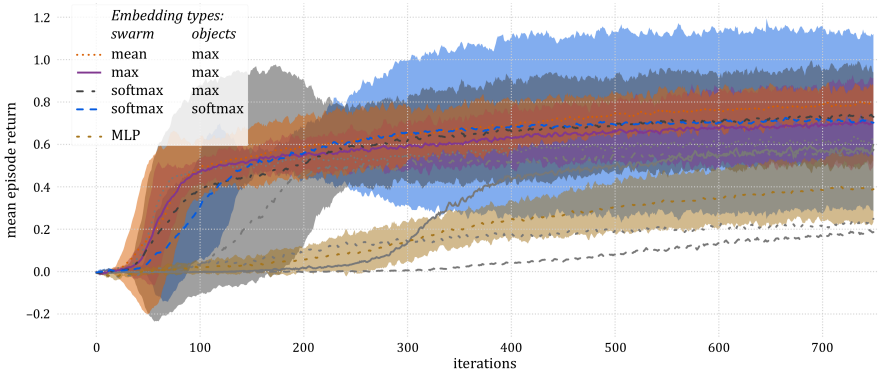

Fig. 9 Learning curves for the segregation task. The colored lines and shaded areas show the mean and two times the standard deviation of the best four out of five trials. The trials that were not included into the statistics are depicted in gray.

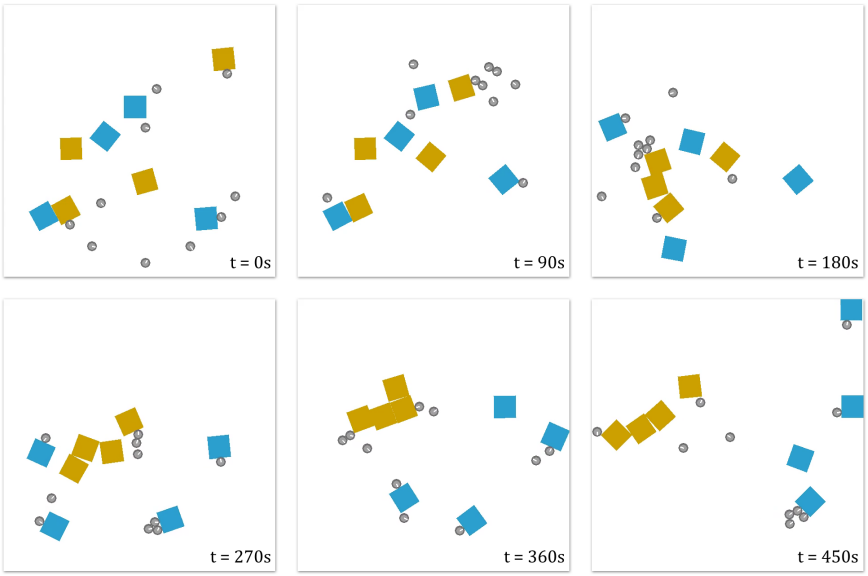

Fig. 10 Exemplary animation of the segregation task. The agents successfully separate the objects into two groups. For this animation, we used a policy network with the softmax-embedding for the swarm observations and a softmax-embedding for the object observations.

embedding for both, the swarm observations and the object observations. Interestingly, the β-values did change only slightly in the policy network, while they changed to much larger extend in the value function network. The changes in the policy network are too small to have an effect on the characteristics of the softmax-embedding. In the softmax-embedding for the swarm observations in the value function network, the changes of the β-values are roughly equally distributed into positive and negative changes. Thus, some of the features have developed a more max-like characteristic (positive changes), while others have developed a

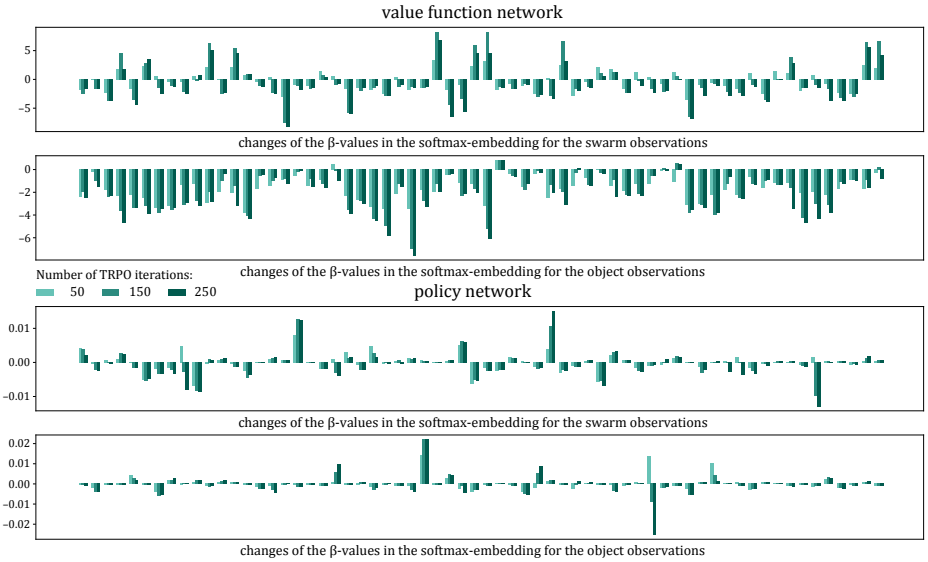

Fig. 11 Changes of the β-values in the softmax-embeddings of the policy network and the value function network after 50, 150, and 250 learning iterations. While the optimization changes the β-values in the value function network, the parameters in the policy network do not change.

more mean-like characteristic (negative changes). In contrast, the changes of the β-values in the softmax-embedding for the object observations in the value function network are nearly all negative. Hence, all these features were updated towards a more mean-like embedding. The latter finding is a bit contradicting to the results of the first experiment in which mean-embeddings for the object observations did lead to very poor performance. An explanation to this could be that for the value function network it is more important to have a distribution over the object features while for the policy network the exact locations of the objects are crucial.

Learned policies are transferable to other swarm sizes and numbers of objects. Finally, we have investigated the transferability of the learned policies to different swarm sizes and different numbers of objects in the environment. We evaluated this ability with the policies learned on the object assembly task and the object segregation task. In the object assembly task we have used a policy with mean embedding for the swarm observations and with max embedding for the object observations. In the object segregation task with used a softmax embedding for both. Figure 12 depicts the average episode return for different settings of the number of agents and objects in both tasks. We can clearly see that the policy generally can be transferred to scenarios with more and less agents in the swarm as well as more and less objects in the environment. While more agents seem to have a positive impact on the outcome, changes in the number of objects tend to have a negative effect on the results. This can be partly explained by the limited space in the environment. Note however, that the reward function and thus the return is not completely independent of the number of objects as we compute the mean over the point-wise object distances. The more objects we have in the environment, the smaller this distance will be already in the beginning and, thus, the smaller the return we can obtain. With too many agents in the environment (i.e., 25 agents), the return tends to

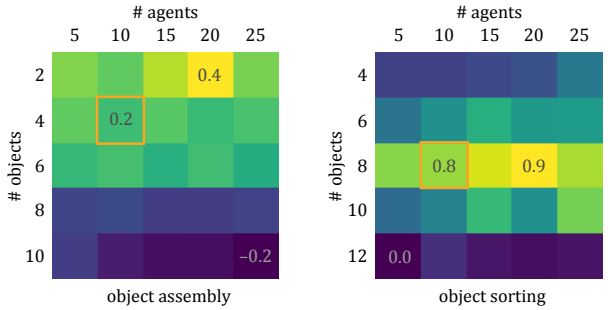

Fig. 12 Evaluation of the transferability to different swarm sizes and different numbers of objects in the environment. Depicted is the mean return over 10 roll outs. The orange squares denote the settings in which the evaluated policy has been learned. Numbers are given for min/max results and for the learning setting.

decrease because the agents start to obstruct the efforts of other agents by pushing objects from different sides. While this experiment should demonstrate that the learned policies can be transferred to different swarm sizes and different numbers of objects, using a variable number of observations during the learning of the policies could further improve the results. Videos of all experiments can be found at `https://tinyurl.com/dme-videos`.

6 Conclusions

In this paper, we present a method to learn policies for manipulating objects with a swarm of homogeneous agents. The policy for the agents is learned from a common reward signal using a centralized critic and distributed actors. Each agent uses the identical policy to compute its actions based on the local observations. To allow for variable number of observations (of other agents but also of objects in the environment) and to reduce the search space of the parameters, we introduce the deep M-embeddings. The deep M-embeddings are inspired by the kernel mean embeddings and provide a network structure that computes a fixed size feature representation from a variable number of inputs. We show in experimental evaluations, that the deep M-embeddings can be employed to learn complex policies for object manipulation with robot swarms in which they outperform classical multi-layer perceptrons in terms of the achieved return but also considering the sample complexity.

The different forms of the deep M-embeddings—mean embedding, max embedding and soft-max embedding—allow to learn embeddings of different characteristics. Our experiments justify, that these characteristics are necessary to learn the policies in the swarm settings. While mean and max embeddings can be used to learn a representation for the swarm state, the object state needs to be represented by a max or a soft-max embedding. While the β-values in the soft-max embedding are learned with the other parameters of the network, it seems that the learning is rather slow and not sufficient find the best characteristics as the use of soft-max embeddings for both, the swarm and the object observations, does not yield good results. In future work, these issues need to be investigated more thoroughly to understand why certain characteristics are necessary for certain types of observation and to leverage the adaptability of the soft-max embeddings better.

References

Al-Shedivat M, Bansal T, Burda Y, Sutskever I, Mordatch I, Abbeel P (2018) Continuous adaptation via meta-learning in nonstationary and competitive environments. In: International Conference on Learning Representations, URL https://openreview.net/forum?id=Sk2u1g-0-

Arulkumaran K, Deisenroth MP, Brundage M, Bharath AA (2017) A brief survey of deep reinforcement learning. IEEE Signal Processing Magazine 34(6):26–38, DOI 10.1109/MSP.2017.2743240, URL http://arxiv.org/abs/1708.05866

Bonabeau E, Dorigo DdRDFM, Dorigo M, Théraulaz G, Theraulaz G (1999) Swarm Intelligence: From Natural to Artificial Systems. OUP USA

Brockman G, Cheung V, Pettersson L, Schneider J, Schulman J, Tang J, Zaremba W (2016) Openai gym. arXiv:160601540 [cs] URL http://arxiv.org/abs/1606.01540

Catto E (2018) Box2d is a 2d physics engine for games. URL https://github.com/erincatto/Box2D

Finn C, Abbeel P, Levine S (2017) Model-agnostic meta-learning for fast adaptation of deep networks. arXiv:170303400 [cs] URL http://arxiv.org/abs/1703.03400

Foerster J, Farquhar G, Afouras T, Nardelli N, Whiteson S (2017) Counterfactual multi-agent policy gradients. arXiv:170508926 [cs] URL http://arxiv.org/abs/1705.08926

Gebhardt GH, Daun K, Schnaubelt M, Neumann G (2018) Learning robust policies for object manipulation with robot swarms. In: Proceedings of the 2018 IEEE International Conference on Robotics and Automation (ICRA), Brisbane

Grover A, Al-Shedivat M, Gupta JK, Burda Y, Edwards H (2018) Learning policy representations in multiagent systems. arXiv:180606464 [cs, stat] URL http://arxiv.org/abs/1806.06464

Gupta JK, Egorov M, Kochenderfer M (2017) Cooperative multi-agent control using deep reinforcement learning. In: Autonomous Agents and Multiagent Systems, Springer International Publishing, Lecture Notes in Computer Science, pp 66–83

Hasselt Hv, Guez A, Silver D (2016) Deep reinforcement learning with double q-learning. In: Thirtieth AAAI Conference on Artificial Intelligence, URL https://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12389

Hausknecht M, Stone P (2015) Deep recurrent q-learning for partially observable mdps. In: 2015 AAAI Fall Symposium Series, URL https://www.aaai.org/ocs/index.php/FSS/FSS15/paper/view/11673

Hochreiter S, Schmidhuber J (1997) Long short-term memory. Neural Computation 9(8):1735–1780, DOI 10.1162/neco.1997.9.8.1735, URL https://doi.org/10.1162/neco.1997.9.8.1735

Hüttenrauch M, Šošić A, Neumann G (2018) Deep reinforcement learning for swarm systems. arXiv:180706613 [cs, stat] URL http://arxiv.org/abs/1807.06613

Kakade S (2001) A natural policy gradient. In: Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic, MIT Press, Cambridge, MA, USA, NIPS'01, pp 1531–1538, URL http://dl.acm.org/citation.cfm?id=2980539.2980738

Kupcsik A, Deisenroth MP, Peters J, Ai Poh L, Vadakkepat P, Neumann G (2015) Model-based contextual policy search for data-efficient generalization of robot skills. Artificial Intelligence URL http://www.ias.informatik.tu-darmstadt.de/uploads/Publications/Kupcsik_AIJ_2015.pdf

Lauer M, Riedmiller M (2000) An algorithm for distributed reinforcement learning in cooperative multi-agent systems. In: In Proceedings of the Seventeenth International Conference on Machine Learning, Morgan Kaufmann, pp 535–542

Lillicrap TP, Hunt JJ, Pritzel A, Heess N, Erez T, Tassa Y, Silver D, Wierstra D (2015) Continuous control with deep reinforcement learning. arXiv:150902971 [cs, stat] URL http://arxiv.org/abs/1509.02971

Lowe R, WU Y, Tamar A, Harb J, Pieter Abbeel O, Mordatch I (2017) Multi-agent actor-critic for mixed cooperative-competitive environments. In: Advances in Neural Information Processing Systems 30, Curran Associates, Inc., pp 6379–6390, URL http://papers.nips.cc/paper/7217-multi-agent-actor-critic-for-mixed-cooperative-competitive-environments.pdf

Matignon L, Laurent GJ, Le Fort-Piat N (2007) Hysteretic q-learning : an algorithm for decentralized reinforcement learning in cooperative multi-agent teams. In: IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS'07., San Diego, CA., United States, vol sur CD ROM, pp 64–69, URL https://hal.archives-ouvertes.fr/hal-00187279

Mnih V, Kavukcuoglu K, Silver D, Graves A, Antonoglou I, Wierstra D, Riedmiller M (2013) Playing atari with deep reinforcement learning. arXiv:13125602 [cs] p 9, URL https://arxiv.org/abs/1312.5602

Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, Bellemare MG, Graves A, Riedmiller M, Fidjeland AK, Ostrovski G, Petersen S, Beattie C, Sadik A, Antonoglou I, King H, Kumaran D, Wierstra D, Legg S, Hassabis D (2015) Human-level control through deep reinforcement learning. Nature 518(7540):529–533, DOI 10.1038/nature14236, URL https://www.nature.com/articles/nature14236/

Ng AY, Harada D, Russell SJ (1999) Policy invariance under reward transformations: Theory and application to reward shaping. In: Proceedings of the Sixteenth International Conference on Machine Learning, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, ICML '99, pp 278–287, URL http://dl.acm.org/citation.cfm?id=645528.657613

Omidshafiei S, Pazis J, Amato C, How JP, Vian J (2017) Deep decentralized multi-task multi-agent reinforcement learning under partial observability. In: International Conference on Machine Learning, pp 2681–2690, URL http://proceedings.mlr.press/v70/omidshafiei17a.html

Palmer G, Tuyls K, Bloembergen D, Savani R (2018) Lenient multi-agent deep reinforcement learning. In: Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems, International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, AAMAS '18, pp 443–451, URL http://dl.acm.org/citation.cfm?id=3237383.3237451

Panait L, Sullivan K, Luke S (2006) Lenient learners in cooperative multiagent systems. In: Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems, ACM, New York, NY, USA, AAMAS '06, pp 801–803, DOI 10.1145/1160633.1160776, URL http://doi.acm.org/10.1145/1160633.1160776

Peng P, Wen Y, Yang Y, Yuan Q, Tang Z, Long H, Wang J (2017) Multiagent bidirectionally-coordinated nets: Emergence of human-level coordination in learning to play starcraft combat games. arXiv:170310069 [cs] URL http://arxiv.org/abs/1703.10069

Peters J, Vijayakumar S, Schaal S (2005) Natural actor-critic. In: Machine Learning: ECML 2005, Springer Berlin Heidelberg, Lecture Notes in Computer Science, pp 280–291

Pugh J, Martinoli A (2006) Multi-robot learning with particle swarm optimization. In: Proceedings of the Fifth International Joint Conference on Autonomous Agents and Mul-

tiagent Systems, ACM, New York, NY, USA, AAMAS '06, pp 441–448, DOI 10.1145/1160633.1160715, URL http://doi.acm.org/10.1145/1160633.1160715

Pugh J, Martinoli A (2007) Parallel learning in heterogeneous multi-robot swarms. In: 2007 IEEE Congress on Evolutionary Computation, pp 3839–3846, DOI 10.1109/CEC.2007.4424971

Rashid T, Samvelyan M, de Witt CS, Farquhar G, Foerster J, Whiteson S (2018) Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning. arXiv:180311485 [cs, stat] URL http://arxiv.org/abs/1803.11485

Rubenstein M, Ahler C, Hoff N, Cabrera A, Nagpal R (2014) Kilobot: A low cost robot with scalable operations designed for collective behaviors. Robotics and Autonomous Systems 62(7):966–975, DOI 10.1016/j.robot.2013.08.006

Rusu AA, Colmenarejo SG, Gulcehre C, Desjardins G, Kirkpatrick J, Pascanu R, Mnih V, Kavukcuoglu K, Hadsell R (2015) Policy distillation. arXiv:151106295 [cs] URL http://arxiv.org/abs/1511.06295

Schulman J, Levine S, Moritz P, Jordan MI, Abbeel P (2015a) Trust region policy optimization. In: International Conference on Machine Learning, pp 1889–1897, URL http://arxiv.org/abs/1502.05477

Schulman J, Moritz P, Levine S, Jordan M, Abbeel P (2015b) High-dimensional continuous control using generalized advantage estimation, URL http://arxiv.org/abs/1506.02438

Schulman J, Wolski F, Dhariwal P, Radford A, Klimov O (2017) Proximal policy optimization algorithms. arXiv:170706347 [cs] p 12, URL http://arxiv.org/abs/1707.06347

Silver D, Lever G, Heess N, Degris T, Wierstra D, Riedmiller M (2014) Deterministic policy gradient algorithms. In: Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32, JMLR.org, Beijing, China, ICML'14, pp I–387–I–395, URL http://dl.acm.org/citation.cfm?id=3044805.3044850

Smola AJ, Gretton A, Song L, Schölkopf B (2007) A hilbert space embedding for distributions. In: Proceedings of the 18th international conference on Algorithmic Learning Theory, Springer Berlin Heidelberg, Lecture Notes in Computer Science, vol 4754, pp 13–31, DOI 10.1007/978-3-540-75225-7, URL http://link.springer.com/chapter/10.1007/978-3-540-75225-7_5

Sunehag P, Lever G, Gruslys A, Czarnecki WM, Zambaldi V, Jaderberg M, Lanctot M, Sonnerat N, Leibo JZ, Tuyls K, Graepel T (2017) Value-decomposition networks for cooperative multi-agent learning. arXiv:170605296 [cs] URL http://arxiv.org/abs/1706.05296

Sutton RS, Barto AG (2018) Reinforcement learning: an introduction, second edition edn. Adaptive computation and machine learning series, The MIT Press, Cambridge, MA

Sutton RS, McAllester D, Singh S, Mansour Y (1999) Policy gradient methods for reinforcement learning with function approximation. In: Proceedings of the 12th International Conference on Neural Information Processing Systems, MIT Press, Cambridge, MA, USA, NIPS'99, pp 1057–1063, URL http://dl.acm.org/citation.cfm?id=3009657.3009806

Yang Y, Luo R, Li M, Zhou M, Zhang W, Wang J (2018) Mean field multi-agent reinforcement learning. arXiv:180205438 [cs] URL http://arxiv.org/abs/1802.05438

Zheng L, Yang J, Cai H, Zhang W, Wang J, Yu Y (2017) Magent: A many-agent reinforcement learning platform for artificial collective intelligence. arXiv:171200600 [cs] URL http://arxiv.org/abs/1712.00600