Variational Meta Reinforcement Learning for Social Robotics

Anand Ballou*, Xavier Alameda-Pineda, Chris Reinke

Inria, Univ. Grenoble Alpes, CNRS, Grenoble INP, LJK

655, Avenue de l'Europe, 38334, Montbonnot, France

Abstract

With the increasing presence of robots in our every-day environments, improving their social skills is of utmost importance. Nonetheless, social robotics still faces many challenges. One bottleneck is that robotic behaviors need to be often adapted as social norms depend strongly on the environment. For example, a robot should navigate more carefully around patients in a hospital compared to workers in an office. In this work, we investigate meta-reinforcement learning (meta-RL) as a potential solution. Here, robot behaviors are learned via reinforcement learning where a reward function needs to be chosen so that the robot learns an appropriate behavior for a given environment. We propose to use a variational meta-RL procedure that quickly adapts the robots' behavior to new reward functions. As a result, given a new environment different reward functions can be quickly evaluated and an appropriate one selected. The procedure learns a vectorized representation for reward functions and a meta-policy that can be conditioned on such a representation. Given observations from a new reward function, the procedure identifies its representation and conditions the meta-policy to it. While investigating the procedures' capabilities, we realized that it suffers from posterior collapse where only a subset of the dimensions in the representation encode useful information resulting in a reduced performance. Our second contribution, a radial basis function (RBF) layer, partially mitigates this negative effect. The RBF layer lifts the representation to a higher dimensional space, which is more easily exploitable for the meta-policy. We demonstrate the interest of the RBF layer and the usage of meta-RL for social robotics on four robotic simulation tasks.

Keywords: Social Robotics, Meta-Learning, Deep Reinforcement Learning, Radial-Basis-Functions

1. Introduction

Since the emergence of social robotics [1], substantial efforts have been invested in developing algorithms to allow the deployment of interactive robots in every-day environments such as malls, train stations or hospitals. Given the complex dynamics of social environments, the approach of manually designing behaviors, for example by using state machines, is often not feasible. As an alternative, reinforcement learning (RL) [2] has been introduced, allowing to autonomously train social behaviors for complex environments. However, classical RL introduces its own set of problems when applied to robotics [3]: (i) RL requires large amounts of training data, meaning hours and hours of observations that have to be collected with the robot, and (ii) how to define the goal of learning, i.e. the reward function, is far from being trivial and properly addressed. In RL the reward function defines which states a robot should try to reach or avoid. It is often a weighted sum over several reward components: $R(s) = \sum_i w_i r_i(s)$. For example, for social navigation [4], [5] besides other components, the reward function might include a positive reward component for reaching the robot's goal position or a negative reward component to avoid getting to close to people.

How to define these components and to set their weights so that the robot learns the intended behavior is not obvious. Having a too strong negative reward for being close to people might result in taking large unnecessary detours. Whereas giving a too strong weight on the goal position might result in disregarding the personal space of people and even colliding with them. Finding the correct weights and components that result in the intended behavior often requires an iterative process where several variations of the reward function have to be tested. Moreover, with classical RL the behavior that results for each reward function variation has to be learned from scratch needing many observations.

Both problems make the application of RL especially problematic for social robotics. Socially appropriate behaviors are strongly context-depended. They usually have to be adapted to different environments. For example, a social robot might be used to navigate in a care center for elderly or an office. In the care center the robot should avoid getting too close to people to avoid that they feel unsafe. Whereas, in an office environment this restriction

*Corresponding author

Email addresses: anand.ballou@inria.fr (Anand Ballou), xavier.alameda-pineda@inria.fr (Xavier Alameda-Pineda), chris.reinke@inria.fr (Chris Reinke)

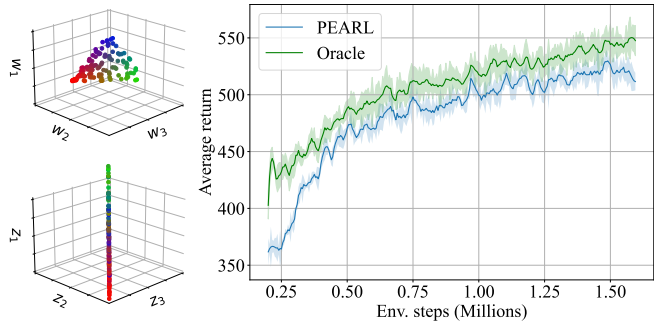


Figure 1: **Graphic visualization of the posterior collapse of PEARL.** (left-top) The ground truth task representation, where each task corresponds to a coloured dot. The three primary colors correspond to the strength of each of three reward component weights. (left-bottom) PEARL's learned task representation uses only one dimension (z_1) and exhibits posterior collapse in the two other dimensions (z_2 and z_3). (right) As a result, PEARL's learning performance compared to a task conditioned policy using the ground truth (Oracle) is reduced. The proposed RBF-PEARL aims to close this gap by mitigating the negative impact of posterior collapse during training.

might be less important and the robot could move closer around people to reach its goal position faster. Being able to adapt a robot behavior quickly to the specific needs of a certain new environment is essential for the practical application of social robotics.

We propose the use of meta [6] and transfer RL [7, 8] to overcome the issues of RL for social robotics. These paradigms aim to adapt the learning process to a certain problem domain to be more efficient on a new target task of this domain, or by reusing knowledge from solved tasks to improve the performance of learning a new target task. Of particular interest for us are methods that are able to adapt to a new target task using only few observations. In our case, tasks are represented by reward functions, i.e. different definitions of their components and component weights. Meta-RL allows to test efficiently several reward functions given a new environment, e.g. a care center or office space, to find the function that results in the most appropriate behavior for the new environment. One class of meta-RL methods able to adapt to new tasks using only few observations are task conditioned policies. These methods learn a behavior, i.e. a policy, that is conditioned on a latent vector representation z of a task. How to represent tasks and solve them is learned on a set of source tasks. Given a new target task, a small amount of observations are collected to compute its task representation and to condition the policy to it.

A promising research direction for task conditioned polices are variational architectures [9], such as PEARL [10]. Instead of learning a deterministic representation of tasks, PEARL learns to represent them via a distribution, exploiting the flexibility of probabilistic models and the representation power of deep neural networks. We investigated the usability of such variational meta-RL frameworks for social robotic tasks, and realized that the learned encoder suffers from posterior collapse resulting in a reduced learning performance (Fig. 1). We propose to use a radial basis function (RBF) layer [11] to transform the task representation before giving it to the downstream task conditioned policy, thus constructing an embedding that is more suitable to represent tasks. RBF networks are universal function approximators [12] whose parameters can be learned, and exhibit interesting results in classification tasks [13, 14, 15] as well as in value-learning for continuous action DRL [16].

In summary, our contribution is two-fold:

1. Successfully demonstrate the usage of meta-RL on three robotics tasks and four different settings by quickly adapting to various reward functions.
2. Improving the existing PEARL algorithm by introducing a RBF layer that transforms the task representation allowing a better training of the task dependent behavior.

In the following, we first discuss work related to our topic and methods, then introduce our methodology, to finally present the experimental protocol and associated findings.

2. Related work

2.1. Social Robotics

In the recent past, several studies investigating the use of reinforcement learning for social robotics [17, 18]. One crucial factor for the success of reinforcement learning is the design of reward functions [3], as they shape the optimal robot behavior. Badly designed reward functions can lead to catastrophic consequences, where the robot learns either undesired or even dangerous actions [19]. This is even more challenging when using reinforcement learning in real-world systems, as in social robotics.

To partially address this challenge, several approaches have been proposed for designing safe and efficient reward functions for social robotics. A prominent line of research in this direction is to obtain rewards directly from a human instructor, i.e. teaching the robot [20, 21, 22, 23, 24]. By definition, these approaches require either an explicit or implicit feedback from a human during the whole training process. While this might be suitable for simple tasks requiring few examples to properly learn the task, it is not well suited for learning problems requiring hundreds (or more) samples for learning. A different line of research consists on using task-driven rewards [25, 26, 27]. In this paradigm, the robot receives a reward after finishing the task, and the reward corresponds to how efficiently the agent performed the task. While this alleviates the constraint of constant human supervision, it implies that the robot receives very sparse rewards, thus effectively increasing the number of samples needed for learning, and yielding an overall more difficult learning problem. At the cross-roads of these two lines, hybrid approaches have been

proposed [28], where the authors define a reward which they use to train their robots and then fine tune the reward function according to the feedback of the users so as to learn better policies.

The use of RL in social robotics is therefore challenging because (i) it is unclear how to design appropriate reward functions, and (ii) increasing the sample efficiency in real-world applications is of utmost importance. In this paper we propose to explore the use of meta-RL for social robotics. Rather than learning one behavior corresponding to one reward function, we would like to learn several behaviors corresponding to different combinations or reward function components.

2.2. Deep Meta-Reinforcement-Learning

Deep meta-RL procedures have the goal of improving the learning speed on a new target task by using experience from a set of similar meta-training tasks. Several approaches exist. One prominent direction has the goal of "learning to learn". The idea is to model the learning process directly with a recurrent or recursive deep network that is trained to solve certain types of task quickly [29, 30, 31, 32]. Another prominent direction is to learn good initialization parameters for a deep network model that allows it to be faster trained on a target task [33, 34, 35, 36]. Both approaches improve the learning performance, but they still use gradient-based, deep network training for the learning procedure on a target task. As a consequence, they still require many observation and training iterations to learn a target task.

A third direction overcomes the need for gradient-based training. The procedures learn a general deep network model that solves similar tasks. Given a target task the model is conditioning to it with a description of the task by giving it as a vector input to the model. One such algorithm is PEARL [37, 10]. It utilizes a variational inference method to learn how to represent tasks and to identify a good task representation based on some task observations. PEARL showed that it needs up to 100× less observations to learn a target tasks compared to gradient-based methods. This motivates our choice of PEARL as our base approach for meta-RL in social robotics.

2.3. Posterior Collapse

As for any variational method, PEARL can suffer from posterior collapse, meaning that at least one of the latent bottleneck dimensions become not informative during training. This behavior can be easily identified by looking at the per-dimension Kullback-Leibler divergence between the posterior and prior distributions. Once a dimension collapses, it is not needed to ensure good reconstruction. Several factors can cause posterior collapse, e.g., a local optima [38], the objective function itself [39, 40, 41, 42], a too unconstrained variance [43] or the fact that the posterior approximation lags behind the true posterior model [44]. In the case of PEARL, having collapsed dimensions translates into a reduction of the task identification power of the method. To mitigate the negative impact of posterior collapse, we propose to lift up the representation power of the latent representation by using a radial basis function layer. In our experiments, this proves more beneficial than increasing the network capacity of PEARL.

3. Approach

3.1. Preliminaries

Reinforcement Learning. Tasks in RL are formalized as Markov decision processes (MDPs). An MDP is a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$ with state space $\mathcal{S}$ and action space $\mathcal{A}$. The agent transitions from a state $s_t \in \mathcal{S}$ by action $a_t \in \mathcal{A}$ to state s_{t+1} at time step t. The transition probability density function $\mathcal{P}(s_{t+1}|s_t, a_t)$ defines the environment dynamics giving the probabilities for transitions. For each transition the agent receives a reward defined by the reward function: $r_t = \mathcal{R}(s_t, a_t, s_{t+1}) \in \mathbb{R}$. The probability transition function and reward function are unknown to the agent. The goal of the agent is to maximize the expected future return for each time step t: $G_t = \mathrm{E}\left[\sum_{k=0}^{\infty} \gamma^k \mathcal{R}(s_{t+k}, a_{t+k}, s_{t+1+k})\right]$ where the discount factor $\gamma \in [0, 1)$ defines the importance of short-term rewards relative to the long-term ones. RL agents maximize the return by learning a policy $\pi(a|s) = \Pr(A_t = a|S_t = s)$ that defines the probability of the agent to take action $a \in \mathcal{A}$ in state $s \in \mathcal{S}$.

In the case of robotics tasks, the state is, for example, represented by the input from the robots visual (cameras) and audio (microphones) sensors. Actions are the motor commands send to its actuators. Taking an action will have an impact on the environment and will create some stochastic change in it. The goal of the task is given by the reward function, e.g. the robot is rewarded if it reaches a certain goal position.

Variational Meta-RL (PEARL). In this work, we would like an agent to adapt quickly to a new reward function using only a few observations. We formalize this problem as a meta-RL problem. Each reward function represents a task $\tau \in \mathcal{T}$ with $\tau = \{\mathcal{R}(s_t, a_t, s_{t+1})\}$. We assume a distribution $\rho(\tau)$ over the task. We differentiate two meta-learning phases: 1) meta-training and 2) meta-testing. During meta-training, a number of training tasks is sampled according to $\rho(\tau)$. Based on these training tasks, a task conditioned policy $\pi(a|s, \tau)$ is learned. During meta-testing, a different set of test tasks is sampled from $\rho(\tau)$ and the adaptation of the learned policy to these tasks is measured.

We chose PEARL [10] as our meta-RL algorithm. PEARL learns a task-conditioned policy $\pi(a|s, z_\tau)$ where $z_\tau \in Z = \mathbb{R}^d$ is a low-dimensional task representation. The task representation z_τ conditions the policy towards maximizing the return of the reward function for task τ. The representation $z_\tau = f(c^\tau)$ is computed based on observed transitions from a task, called the context $c^\tau = \{c_n^\tau\}$, where

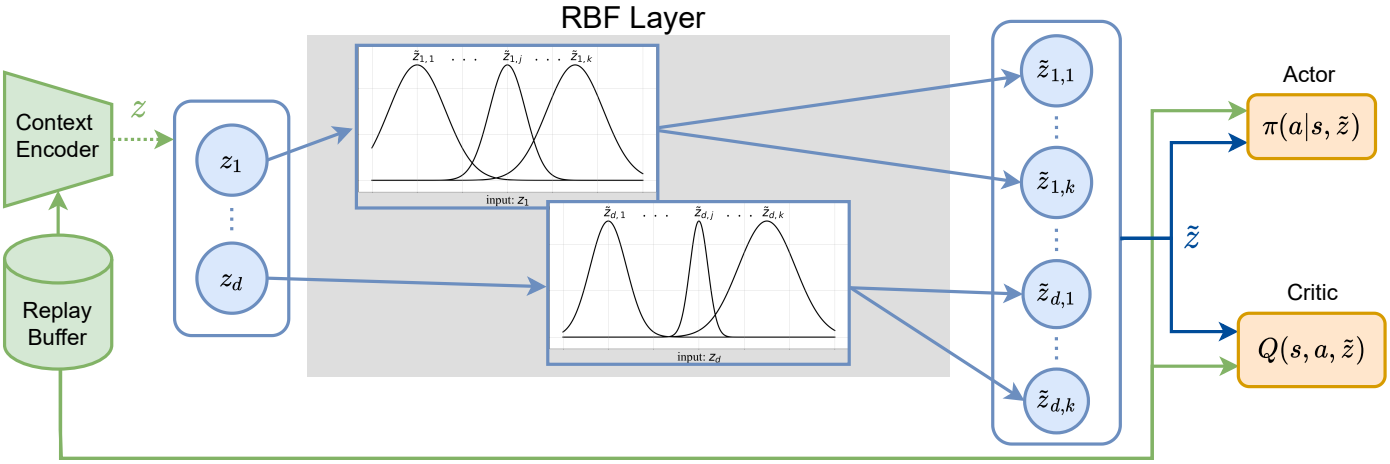


Figure 2: **PEARL-RBF Architecture.** The proposed meta-RL procedure learns to adapt to unseen test reward functions. The context encoder uses data from the replay buffer to infer the posterior over the latent context variable z. The latent context sampled from the posterior $z = (z_1, \dots z_d)$ is then fed to the RBF network, which uplifts every input dimension m to a different k-dimensional representation: $z_m \to (\tilde{z}_{m,1}, \dots, \tilde{z}_{m,k})$. The resulting task representation $\tilde{z}$ is used to condition the actor and critic network.

$c_n^\tau = (s_n, a_n, r_n, s'_n)$. PEARL uses a variational method, similar to variational autoencoders (VAEs) [9], to estimate the posterior distribution of the low-dimensionsal representation given the context $p(z|c)$. It approximates the posterior with an inference network $q_\phi(z|c)$ parametrized by ϕ. The network is trained on a log-likelihood objective resulting in the following variational lower bound:

$$\mathbb{E}_\tau \left[\mathbb{E}_{z \sim q_\phi(z|c^\tau)} \left[\mathcal{R}(\tau, z) + \beta D_{KL}(q_\phi(z|c^\tau)) \| p(z)) \right] \right] \quad (1)$$

where $\mathcal{R}(\tau, z)$ is the return for task τ using the policy conditioned on z and $p(z)$ is a standard Gaussian prior over z. The inference network is a product of independent Gaussian factors for each transition in c^τ:

$$q_\phi(z|c^\tau) \propto \prod_{n=1}^{N} \mathcal{N}\left(f_\phi^\mu(c_n^\tau), f_\phi^\sigma(c_n^\tau) \right) \quad (2)$$

where f_ϕ^μ and f_ϕ^σ are represented by a neural network.

During meta-training, the policy $\pi(a|s, z)$ is learned using the soft actor-critic (SAC) [45] algorithm. SAC is off-policy, consisting of an actor $\pi_{\theta_\pi}(a|s, z)$ and a critic $Q_{\theta_Q}(s, a, z)$ network. PEARL jointly trains the inference, actor and critic networks using the reparameterization trick similar to VAEs [9]. During a meta-training step, the training procedure has two phases: 1) data collection and 2) network parameters update. In the data collection step, a replay buffer $\mathcal{B}^\tau$ is filled with the transitions from K trajectories for each training task τ. Then, for each trajectory PEARL samples a task representation $z \sim q_\phi(z|c^\tau)$ to condition the policy where c^τ is sampled from the replay buffer $\mathcal{B}^\tau$. During the second phase, the procedure updates the network parameters for each training task τ. It first samples a batch of context c^τ from recently sampled transitions in the replay buffer $\mathcal{B}^\tau$. Then, the task representation $z \sim q_\phi(z|c^\tau)$ is sampled from the posterior distribution of z given the context c^τ. The critic and actor networks are updated using the task representation and independently sampled transitions from the whole replay buffer. The loss of the critic is given by:

$$\mathcal{L}_{critic} = \mathrm{E}_{\substack{(s,a,r,s') \sim \mathcal{B} \\ z \sim q_\phi(z|c)}} \left[Q_\theta(s, a, z) - (r + \dot{V}(s', \dot{z})) \right]^2 \quad (3)$$

where $\dot{V}$ is the value, i.e. the maximum Q-value, of a target network, and $\dot{z}$ indicates that gradients are not being computed through it. A target network is necessary here because directly implementing Q learning with neural networks proved to be unstable in many environments [46]. The actor loss is given by:

$$\mathcal{L}_{act} = \mathrm{E}_{\substack{s \sim \mathcal{B}, a \sim \pi_\theta \\ z \sim q_\phi(z|c)}} \left[log(\pi_\theta(a|s, \dot{z}) - (Q_\theta(s, a, \dot{z})) \right] \quad (4)$$

The loss of the inference network for the task representation is composed of the critic loss and the Kullback–Leibler divergence term from (1):

$$\mathcal{L}_\phi = \mathrm{E}_{\substack{(s,a,r,s') \sim \mathcal{B} \\ z \sim q_\phi(z|c)}} \left[\mathcal{L}_{critic} + \beta D_{KL}(q_\phi(z|c^\tau)) \| p(z)) \right]. \quad (5)$$

For meta-testing, a test task τ is first explored for a few hundred time steps. The policy $\pi_e(a|s, z_e)$ used for exploration is conditioned on a task representation sampled from the Gaussian prior $z_e \sim p(z)$. After the exploration phase, the context c^τ collected with π_e is used to compute the final task representation given by the sample mean of the posterior mean: $z^\tau = \frac{1}{N} \sum_{n=1}^{N} f_\phi^\mu(c_n^\tau)$. More details on the meta-testing phase are provided in the experimental section.

3.2. RBF for Variational Meta-RL

We noticed that in several scenarios PEARL suffers from posterior collapse (of the learned task representation

z). As a result, not all dimensions of z are used. The task information is compressed in few dimensions making it difficult for the downstream policy and critic network to learn from z. To compensate for this, we propose to transform the task representation $\tilde{z} = \varphi(z)$ to a representation that is easier to process for downstream networks. Inspired by [15, 16], we propose the usage of radial basis function (RBF) layers to transform the task representation.

We construct the RBF layer based on the idea of RBF networks [11] which are universal function approximators [12]. RBF networks consist of a layer of hidden neurons that have a Gaussian activation function. The output of the networks is a weighted sum over the Gaussian activations. In a similar manner, our proposed RBF layer consist of a layer of neurons. For each input dimension $z_j \in \mathbb{R}$ from the original task representation exist N RBF neurons: $\tilde{z}_{j,1}, \ldots, \tilde{z}_{j,N}$. Each neuron represents a radial basis function having a Gaussian shape:

$$\tilde{z}_{i,j} = \exp\left(-\delta_{i,j}||z_i - c_{i,j}||^2\right), \tag{6}$$

where $\delta_{i,j} \in \mathbb{R}$ is a scaling factor and $c_{i,j} \in \mathbb{R}$ is the center, i.e. the point of the highest activation. δ and c are the parameters of the RBF layer which can be either fixed or trained using gradient descent based on the loss function of downstream networks.

In summary, we propose to transform the task representation using a RBF layer: $\tilde{z} = \varphi(z)$. The resulting representation is then given to the task conditioned actor $\pi_\theta(a|s, \tilde{z})$ and critic $Q(s, a, \tilde{z})$ network.

4. Experimental Results

We propose to test our approach on three different environments inspired from social interaction situations. In the first environment, the agent, represented by a robotic head, needs to learn how to control its position following different criteria (e.g. maximizing the number of people in the field of view or facing the speaker). In the second environment, the agent must learn to navigate safely and in a socially-compliant manner through a crowd of people. The third one is a continuous control environment focused around robotic locomotion. The purpose of these three environments is twofold. First we want to show the effectiveness of our proposed methodology to generate different behaviors in human robot interaction tasks. Secondly, we assess whether or not the use of RBF layers is beneficial for variational meta-RL in terms of training and adaptation efficiency.

4.1. Evaluation protocol

Baseline. The most natural baseline to the proposed RBF-PEARL is standard PEARL. However, directly comparing with PEARL seems unfair, since adding RBF layers increases the number of parameters of the actor and critic networks. We therefore adjust the number of parameters of the actor and critic networks of the PEARL baseline so as to match the number of parameters of the RBF-PEARL. Beyond this adjustment, both methods use separated actor and critic networks, and both networks consist in a 3-layers MLPs with 300 neurons per layer. Both methods use a 3-layer MLP with 200 neurons per layer as encoder. In more details, if the latent dimension is d, and we use one RBF layer with k neurons, the first layer of the actor/critic networks would have $300d$ and $300kd$ parameters for PEARL and RBF-PEARL respectively. For the sake of a fair comparison, we add an extra layer at the beginning of PEARL's actor/critic network with kd output neurons, thus yielding the number of parameters of PEARL and RBF-PEARL actor and critic networks comparable. Unless otherwise stated, we use $k = 9$.
As a separate baseline, we used a modified version of the soft actor critic algorithm. We trained one agent per tasks, using only 200 observations. However we strongly increased the numbers of gradient step performed per observations in order to force the network to learn a behavior with only this 200 observations. We used this baseline to compare the performances classical RL algorithms (here SAC) trained on 200 observation can reach with respect to RBF-PEARL and PEARL with 200 adaption steps. We except the performances of these agents trained with soft actor critic algorithm to be lower than the performances of PEARL and RBF-PEARL.

Evaluation protocol. For meta-training, we sample 100 tasks for the three environments. Evaluation is performed on 20 meta-testing tasks that are different from those used at meta-training time. To evaluate on each meta-testing task τ, we collect 200 time steps with a task representation z_e sampled from the standard Gaussian prior. Then we aggregate this 200 time steps to compute a task representation z^τ sampled from the posterior estimation given by the encoder. To compute the final test-time performance, we record the performance of the policy associated to the task representation z^τ for one episode. We repeat the training process 5 times, and report the mean performance and the associated standard error.

4.2. Gaze control environment

Environment description. We inspire from recent research in RL for social robotics [18] to design this environment, where we aim to learn a gaze control strategy for a robotic head. In our setup, the robot observations are multimodal consisting of visual, auditory, and proprioception cues. To this end, we assume normalized scene coordinates of size 2×1. Regarding the visual observations, we assume a regular head camera that extracts pose cues from the 0.4×0.3 field of view (FoV). In more detail, we assume that a multi-person pose estimation method is available, and provides one heatmap for each of the $J = 18$ landmarks (e.g. nose, neck, left shoulder, right hip). The heatmap associated to each landmark indicates the probability of the presence of that landmark at every position. Since state-of-the art pose estimators provide these heatmaps in low resolution,

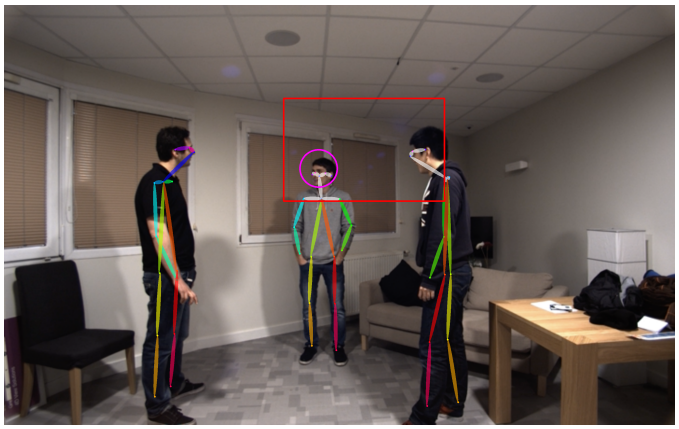


Figure 3: **Illustration of the gaze control environment.** The field of view is shown in red, the active speaker in purple, and the visible/non-visible landmarks are shown in white/color respectively.

the visual observations will consist on J 7×7 heatmaps. Regarding the auditory observations, we emulate the output of a sound source localization algorithm. Given the precision of current sound source localization methods, it seems reasonable to represent the auditory input as a 14×8 heatmap corresponding to the entire scene. Each of its cells corresponds to the probability of having an active speaker in this direction. Importantly, while the visual features correspond to the current field of view of the camera, the auditory features correspond to the entire scene, since audio localization is not limited by the camera's FoV. Lastly, the proprioception cues consist of the coordinates of the robots current field of view center. It is encoded using a $\mathbb{R}^2$ vector representing both the pan and tilt angles of the robotic head, defining to which part of the scene corresponds the visual input. An observation of the gaze control environment is shown in Figure 3 (left).

Regarding the action space of the agent A, we set it to $[-1, 1]^2$,corresponding to the pan and tilt angular velocities respectively. We choose to normalize the action space so as to stabilize the network learning. The maximum pan and tilt velocities correspond to shifting the camera field of view by 0.16 and 0.11 in normalized scene coordinates at every time step respectively.

Reward components. We define three reward components that will be combined to generate various tasks (i.e. reward functions). We will focus on the number of people in the field of view, the presence of a speaker in the field of view, and reducing spurious robot movements. These three components are very generic, and relevant for any social robot. The dimension of the latent space of both PEARL and RBF-PEARL is set to $d = 3$.

We want to reward the robot for having people in the field of view of the camera, since this means the robot would be looking at people. Naively reward the number of people within the field of view leads to an action policy that explores until finding a person, and then follows the person. A more natural behavior is to check on previously detected people. We therefore propose to use a visual reward component that depends on the last time the agent saw a person. More precisely:

$$R_{\text{vis}} = \sum_{p \in P_{\text{vis}}} 2 - \exp(-t_p) \tag{7}$$

where P_{vis} is the set of people whose face is in the visual field of view and $t_p \in [0, \infty]$ is the time since the person was last seen (before the current frame). When a person keeps on being in the field of view, the reward is close to 1. When a person not seen for long time reappears in the field-of-view, the reward is close to 2. In this way, the desired behavior is encouraged.

Regarding audio, we would like the robot to look at the speaking person(s), and therefore we define the audio component of the reward function as:

$$R_{\text{aud}} = \begin{cases} 0 & \text{Nobody speaks} \\ -0.5 & \text{Speakers are outside the FoV} \\ 2|P_{\text{aud}}| & \text{Speakers within the FoV} \end{cases} \tag{8}$$

where P_{aud} is the set of speakers in the field of view.

Finally, we would like to penalize large and fast movements since they are quite unnatural in social interactions. This is why we propose a movement (negative) component of the reward function, defined as:

$$R_{\text{mov}} = -K_{\text{mov}} \sqrt{a_{pan}^2 + a_{tilt}^2} \tag{9}$$

where a_{pan} and a_{tilt} are the two components of the action space, and $K_{\text{mov}} = 16$ is a constant to put R_{mov} in a similar numeric range as R_{vid} and R_{aud}.

Tasks. In order to construct different tasks (reward functions) we propose to use the defined components in two different ways. First, with simple combinations as in [18], and then with more complex ones (i.e. non-linear). The first family of reward functions is generated by sampling convex combinations of the three components defined above:

$$R^\tau = \omega^\tau_{\text{vis}} R_{\text{vis}} + \omega^\tau_{\text{aud}} R_{\text{aud}} + \omega^\tau_{\text{mov}} R_{\text{mov}} \ , \tag{10}$$

where $\omega^\tau_{\text{vis}}, \omega^\tau_{\text{aud}}, \omega^\tau_{\text{mov}} \in [0, 1]$ are random convex weights, meaning that $\omega^\tau_{\text{vis}} + \omega^\tau_{\text{aud}} + \omega^\tau_{\text{mov}} = 1$.

We also wanted to compare the algorithms in more complex environments. To that aim, we design our second family of reward function using random multi-layer-perceptron (MLP) networks. These input the value of the three reward components defined above. The MLPs have 1 to 3 layers with 4 to 6 neurons each, that are activated with a sigmoid with probability 0.75. The number of layers, the neurons per layer, the activation and the weights are sampled randomly. Formally, we write:

$$R^\tau = f^\tau(R_{vis}, R_{aud}, R_{mov}; W^\tau) \tag{11}$$

where f^τ represents the sampled MLP network with sampled connection weights W^τ.

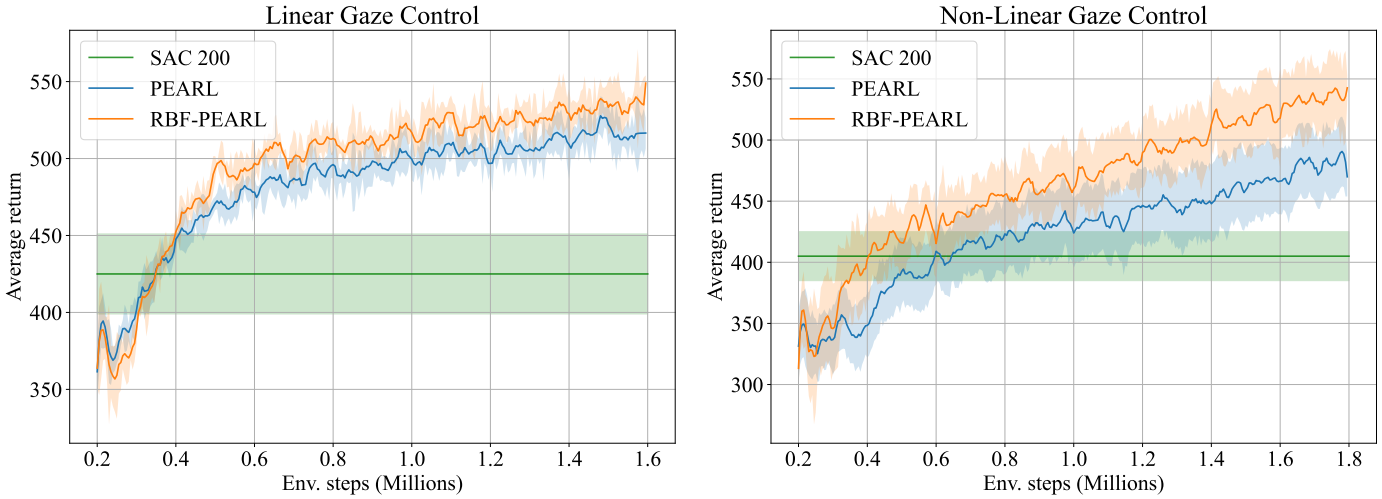


Figure 4: **Results on the gaze control environment.** (right) Test-task performance vs. samples collected during meta-training on the linear family of reward functions for the gaze control environment. RBF-PEARL outperforms PEARL. (left) Test-task performance vs. samples collected during meta-training on the non-linear family of reward functions on the gaze control environment. RBF-PEARL also outperforms PEARL with a larger margin.

Results. In Figure 4, we report the average return over the set of meta-testing tasks over the met-training iterations. More precisely we plot the average return mean and standard deviation over the five independent runs, for the gaze control environment with convex (top) and non-linear (bottom) combinations of reward components.

Generally speaking, both PEARL and RBF-PEARL are able to provide a better adaptation starting point with the training progress. In addition, we observe that RBF-PEARL has a steeper learning curve than PEARL on both types of reward functions. More precisely for convex combinations of reward components, RBF-PEARL performs comparably to PEARL during the first 400k steps. From this point on, RBF-PEARL systematically outperforms PEARL by a margin of 20-30. For the non-linear combinations of reward components, RBF-PEARL exhibits superior performance from 600k meta-training steps on, by a margin of roughly 50. Overall, on these two families of tasks RBF-PEARL is faster than PEARL and has better asymptotic performance, thus show-casing the benefit of using the RBF layer. As excepted, both PEARL and RBF-PEARL achieves better performances than the batch of agents trained using the soft actor critic algorithm with only 200 observations (SAC 200). In the Linear Gaze Control environment, the average performances reach by agents trained with the soft actor critic algorithm is inferior to the performances of PEARL and RBF-PEARL by a margin of 100. In the Non-Linear Gaze Control environment, the gap is lower between PEARL and SAC 200 as PEARL outperforms SAC 200 by a margin of 50. With RBF-PEARL, the gap of performances with SAC 200 is more significant as RBF-PEARL exhibits superior performance by a margin of 130.

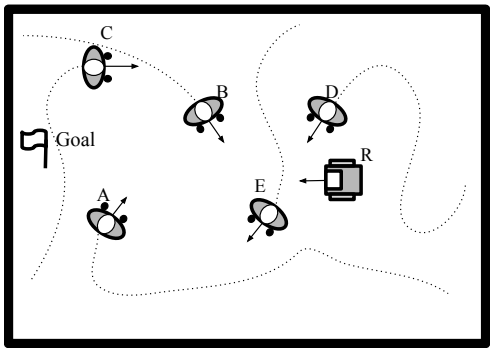


Figure 5: **Schematic representation of the social navigation environment**. The robot R must move towards its goal position while navigating around five human agents $(A, \ldots, E)$.

4.3. Social navigation environment

Environment description. For this task, we took inspiration from the literature on social group and crowd navigation [4], [5]. We propose to learn a navigation strategy for a mobile robot. Our simulation environment is an empty room of dimension 15×10 m. The room is populated with five human agents, whose position at the beginning of each episode is randomly initialized, see Figure 5. Likewise, we randomly sample a robot goal position at the beginning of an episode. The robot should reach the goal position before the end of the episode without disturbing the human agents. In our setting, each human agent is given a random goal position. During an episode the human agent will go towards the goal position and will be assigned a new one after reaching the original goal. To simulate the behavior of human agents, we model the human agents' motion using a social force model [47] to generate plausible trajectories. This framework also limits the amount of collision between human agents. In our setup, the robot has access to the coordinates of all people in the scene,

their velocities and their orientations. The robot also has access to its own velocity, coordinates and goal position. While the robot always begins the episode at the same position (coordinates $(14, 5)$), its goal position is randomly sampled following $x \sim \mathcal{U}(0.0, 2.0)$ and $y \sim \mathcal{U}(0.0, 10.0)$, for the x and y coordinates respectively. Lastly, the action space is a continuous two dimensional space set to $[-15, 15] \times [-2, 2]$, which corresponds respectively to the angular velocity in $rad.s^{-1}$ and linear velocity in $m.s^{-1}$. The maximum value of the linear and angular velocity are chosen such that the robot can go as fast as any human agent in the scene. In this environment, we use a smaller number of neurons $k = 5$ for our RBF-layer.

Reward components. We define a set of five reward components that will be combined to generate various tasks. The dimension of the latent space of both PEARL and RBF-PEARL is set to $d = 5$ for this environment.

First, the goal component R_g is designed to reward the agent for reaching the goal position:

$$R_g = 1 - \frac{d(r, g)}{D}, \tag{12}$$

where $d(r, g)$ is the distance between the robot and the goal and D is a normalizing factor to guarantee that the goal component stays within $[-1, 1]$.

Second, the collision component R_c is built to penalize collisions between the robot and human agents:

$$R_c = \begin{cases} 0 & d(r, h_i) > d_c \\ -1 & d(r, h_i) < d_c \end{cases} \tag{13}$$

where $d(r, h_i)$ is the distance between the robot and the human agent i and d_c is the collision threshold between the robot and the human agent.

Third, the social component R_s is designed to reward the robot in maintaining a safe distance from all human agents. The social component depends on the distance between the robot and each of the human agent. If the distance between the robot and one of them is below a certain threshold, the robot will be penalized. The closer the robot is to the person, the higher the penalty will be. If the robot is close to more than one human agent, only the closest person to the robot is taken into consideration (that is to say the human agent that will generate the lowest reward):

$$R_s = \min_i \left[\frac{d(r, h_i)}{d_s} - 1 \right] \tag{14}$$

where $d(r, h_i)$ is the distance between the robot and the human agent i and d_s can be understood as a threshold. Indeed, if the minimum distance between the robot and a human is below d_s, the reward becomes negative. Thus, d_s can be seen as the distance at which the robot enters the comfort zone of people.

Fourth, the approach component R_a is designed to reward the robot for positioning itself so as to avoid making other humans agents aware of it. This component is inspired from the literature in social robotics, see [48]. In this paper, the authors tried to quantify how aware people are of a robot when it approach them. They use the relative positions and orientations of the robot and the human agent to evaluate the awareness of the robot by the human. In our approach, we want to minimize the distraction created by the robot when it navigates. Therefore we are trying to minimize the awareness of the robot by the human agents in the scene. We compute the awareness of the robot by each human agent and use this as our approach component by rewarding low awareness of the robot. The approach component is defined for each human agent. It is a combination of visibility and direction. While the visibility assesses how much visible the robot is for the human agent, the direction assesses if the robot is going in a direction that would make the human agent more aware/afraid of it.

The visibility for human agent i is defined as :

$$R_{\text{visible},i} = \begin{cases} 1 - \frac{\theta_{i,r}}{\theta_{th}} & \theta_{i,r} < \theta_{th} \\ -\frac{\theta_{i,r} - \theta_{th}}{\pi - \theta_{th}} & \text{otherwise} \end{cases} \tag{15}$$

where θ_{th} is a threshold angle from which the robot is visible to the human agent and $\theta_{i,r}$ is the angle of the robot relative to the human agent i motion. If the robot is in front of the human agent, the value will be close to 1, and if the robot is in the back of the human the value will be close to -1.

The direction for human agent i is defined as :

$$R_{\text{direction},i} = \begin{cases} 1 - \frac{\theta_{r,i}}{\pi/2} & \theta_{i,r} < \theta_{th} \\ 1 & \text{otherwise} \end{cases} \tag{16}$$

It is designed to address how the human agent perceives the robot coming toward him. If the robot is coming closer to the human agent, he/she will become more aware of it and be distracted by it. On the other hand, even if the robot is visible to the human agent, if it move away from him/her, it will be less likely that the human agent will be distracted by the robot.

To compute the approach reward component, we compute the minimum over the visibility/direction product, over the human agents:

$$R_a = \min_i R_{\text{visible},i} \cdot R_{\text{direction},i} \tag{17}$$

Fifth, and last, the velocity component R_v is designed to penalize the robot for going too fast when it is in front of people. It is preferable to avoid having the robot being fast in front of people as they may be disturbed or even afraid by it. In our context in particular, we want to avoid having the people focusing on the robot. Thus we define a velocity component which penalizes the robot for fast-moving if it is visible to the others human agents in the scene. The velocity component for each human agent i is

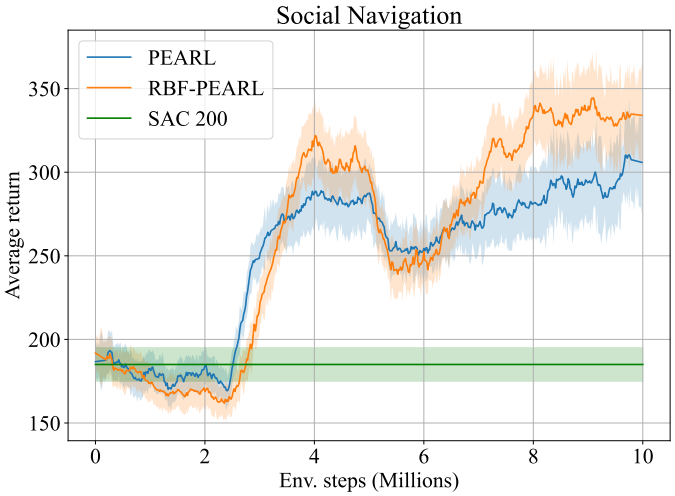


Figure 6: **Results on the social navigation environment.** Test-task performance vs. samples collected during meta- training on the family of reward functions corresponding to the social navigation environment. After 6 millions steps, RBF-PEARL outperforms PEARL.

therefore defined as:

$$R_{v,i} = \begin{cases} -e^{v}(1 - \frac{\theta_{i,r}}{\theta_{th}}) & \theta_{i,r} < \theta_{th} \\ 0 & \text{otherwise} \end{cases} \tag{18}$$

where θ_{th} is a threshold angle from which the robot is visible and $\theta_{i,r}$ is the angle of the robot relative to the human agent motion. To compute the reward component we take the same methodology as with the social and approach component, that is to say that we choose the minimum value obtained from all the human agents in the scene:

$$R_v = \min_i R_{v,i}. \tag{19}$$

Tasks. In order to evaluate the meta-RL algorithms in this setting, we propose to test it with convex combinations, as in the gaze control environment. This social navigation environment is more challenging because the higher number of reward components makes it more difficult for the learning process, and for the behavior generation as there is a wider variety of possible reward functions. Thus our family of reward functions is generated using convex combinations of the five components defined above:

$$R^{\tau} = \omega_g^{\tau} R_g + \omega_c^{\tau} R_c + \omega_s^{\tau} R_s + \omega_a^{\tau} R_a + \omega_v^{\tau} R_v \ , \tag{20}$$

where $\omega_g^{\tau}, \omega_c^{\tau}, \omega_s^{\tau}, \omega_a^{\tau}, \omega_v^{\tau} \in [0, 1]$ are the randomly sampled convex weights leading to task τ, meaning that $\omega_v^{\tau} + \omega_c^{\tau} + \omega_s^{\tau} + \omega_a^{\tau} + \omega_v^{\tau} = 1$.

Results. In Figure 6, we report the average return over the meta-testing tasks with the progress of the meta-training for PEARL and RBF-PEARL. As in the previous environment, we report mean and standard deviation over five runs. Similarly to the previous case, the performance of the two methods look similar during the first steps of the training. More importantly, the RBF layer seems to have a positive impact on the asymptotic performance. Indeed, after 6 millions steps, the RBF-PEARL algorithm performs better than PEARL. Similarly to the gaze control environment, both PEARL and RBF-PEARL performs better than SAC-200 by a large margin (170)

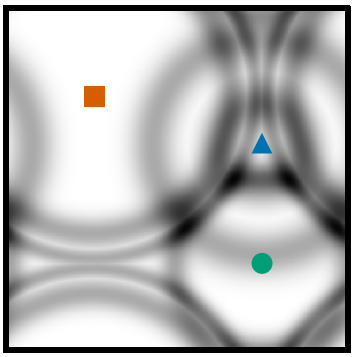


Figure 7: **Illustration for the racer environment.** The three markers are depicted in blue, green and orange. Dark regions correspond to high reward regions. In the task depicted in the figure, the read and blue markers have two Gaussians, while the green one has only one. The number of Gaussians, their mean and standard deviation are randomly sampled for each task.

4.4. Racer environment

Environment description. We evaluated the algorithms in an additional non-social task with complex, non-linear reward functions, called the racer environment [49]. The agent has to navigate in a continuous two dimensional scene for two hundred time steps (Fig. 7). Similar to a car, the agent has an orientation and momentum, so that it can only drive straight, or in a right or left curve. The agent reappears on the opposite side if it exits one side. At the beginning of an episode the agent is randomly placed in the environment. The agents state is a vector $s \in \mathbb{R}^{120}$ corresponding to the agents' position and orientation. The position is encoded using a 10×10 evenly distributed grid of two-dimensional Gaussian radial basis functions. Similarly, the orientation is also encoded using 20 Gaussian radial basis functions. The action space of the agent consists of a one dimensional continuous space set to $[-1, 1]$. The value of the action correspond to the force applied to the agent, which then modifies the agents orientation and position. For example, if the value of the action is close to -1, the agent will make a left curve.

Reward components. We define three reward components, each of them associated to one of the markers in Figure 7. More precisely, each reward component r_k is defined as the maximum over Gaussian-shaped functions over the distance to the k-th marker d_k:

$$r_k = \max \left\{ \exp\left(-\frac{(d_k - \mu_{k,j})^2}{\sigma_{k,j}} \right) \right\}_{j=1}^{n_k} \tag{21}$$

where n_k is the number of Gaussians for marker k, and $\mu_{k,j}$ and $\sigma_{k,j}$ are the mean and standard deviation of the j-th

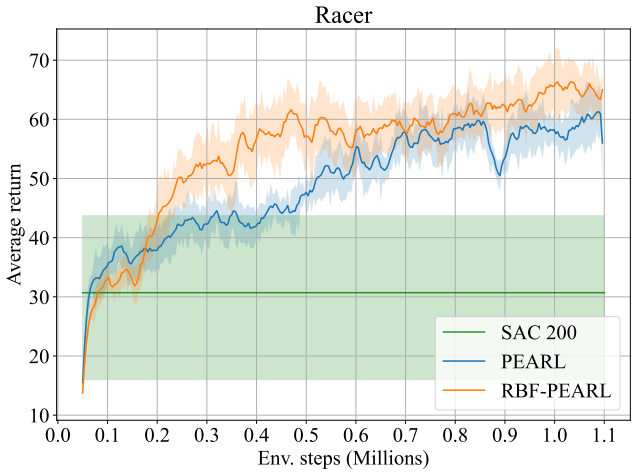


Figure 8: **Results on the racer environment.** Test-task performance during meta-training. Once again with complex reward functions, the interest of the RBF layer is confirmed.

Gaussian of marker k. For each task, the parameters of the reward components are randomly sampled, as explain in the following.

Tasks. The tasks differ in the parameters of each of the three reward components. The number of Gaussians is sampled uniformly: $n_k \sim \mathcal{U}\{1, 2\}$. The two parameters of each Gaussian component are sampled according to $\mu_{k,j} \sim \mathcal{U}(0.0, 0.7)$ and $\sigma_{k,j} \sim \mathcal{U}(0.001, 0.01)$. This sampling instantiates the three reward components for task τ, r_k^τ, and the final reward function writes:

$$R^\tau = \frac{1}{3} \sum_{k=1}^{3} r_k^\tau(d_k). \tag{22}$$

The dimension of the latent space of both PEARL and RBF-PEARL is set to $d = 3$.

Results. RBF-PEARL shows consistently a stronger performance than PEARL for the racer environment (Fig. 8). After $300,000$ steps, RBF-PEARL constantly outperforms PEARL. The asymptotic performances of RBF-PEARL is also higher than PEARL. The average return at the end of the training is $54 \pm (11)$ for PEARL and $65 \pm (2)$ for RBF-PEARL. Also, both PEARL and RBF-PEARL outperforms SAC 200 by a large margin, performing two times better with a performances of only 30 for SAC 200.

5. Discussion

We would like to discuss four question that we believe deserve some attention. First, whether variational meta-RL is well suited for social robotics. Second, what is the impact of the RBF layer in variational meta-RL. Third, how the trainablility of the RBF layer's parameters and the number of RBF neurons influence its performance. And finally, what is the mechanisms behind the improved performance of the RBF layer.

5.1. Variational Meta-RL for Social Robotics

We proposed the application of variational meta-RL to allow robots to quickly adapt to different social scenarios. We achieve this by enabling robots to adapt to new reward functions which define the requirements of social scenarios, such as different preferred social distances of humans. Indeed, our results from four simulation experiments show that robotic agents are able to quickly adapt with a variational meta-RL procedure (PEARL) to different scenarios, i.e. reward functions, requiring only 200 observations (Fig. 4, 6, and 8). In difference, a classical RL algorithm trained on 200 observation (SAC 200) reaches a significant lower performance. In the linear and non linear gaze control environments the performances of SAC 200 were respectively 23% lower and 25 % lower than RBF-PEARL. In the social navigation environment, the performances of SAC 200 were 47% lower. In the racer environment, the performances of SAC 200 were 53% lower than RBF-PEARL.

To further demonstrate the interest of variational meta-RL, we compared the performance of SAC trained on more than 200 steps to the final models obtained with PEARL and RBF-PEARL. As done in the previous experiments, the two meta-RL algorithms only have 200 environment steps to adapt to each test environment. Experiments are done for the linear-gaze, non-linear gaze and racer environments (Fig. 9). Please note, for these experiments the number of gradient descent iterations per collected environment observation is the same for the meta-RL algorithms and for SAC. In the previous experiments (Sec. 4), SAC was given more gradient descent iterations per observation to allow it to converge to a policy. In the first two environments, SAC requires $50,000$ environment steps to reach the performance of PEARL and RBF-PEARL, which only benefit of 200 environment steps. Surprisingly, even after training for $200,000$ steps, SAC does not outperform significantly both meta-RL algorithms. This is specially true for the third environment (racer), where the performance of SAC does not seem to be superior to the meta-RL algorithms, even after $100,000$ environment steps. It would appear that SAC does not manage to learn a moderately optimal action policy in the racer environment. After carefully looking at our results, we realize that the optimal hyperparameters of SAC highly depend on the learned task, and no set of parameters seems to be commonly optimal for all tasks.

Lastly, we evaluated if the learned meta-policy by RBF-PEARL is able to produce diverse and meaningful behavior when adapted to different reward functions. We plotted the trajectories of the robot agent in the social navigation environment for four different reward weight combinations (Fig. 10). The meta-policy is adapted to each combination based on observations from 200 time steps. The adapted behaviors are easily distinguishable from each other. For a reward function that depends only on reaching the goal (a), the robot has trouble reaching the goal position as it bumps two times into humans and does not manage

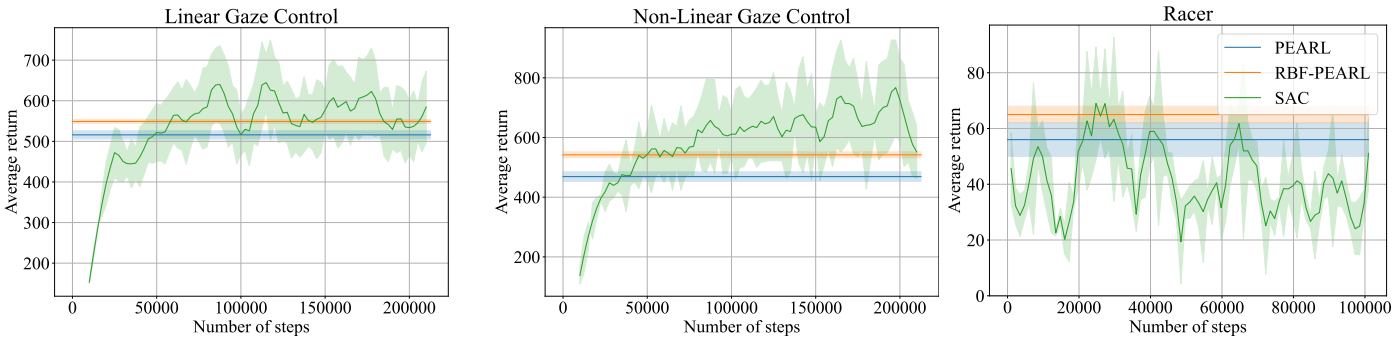
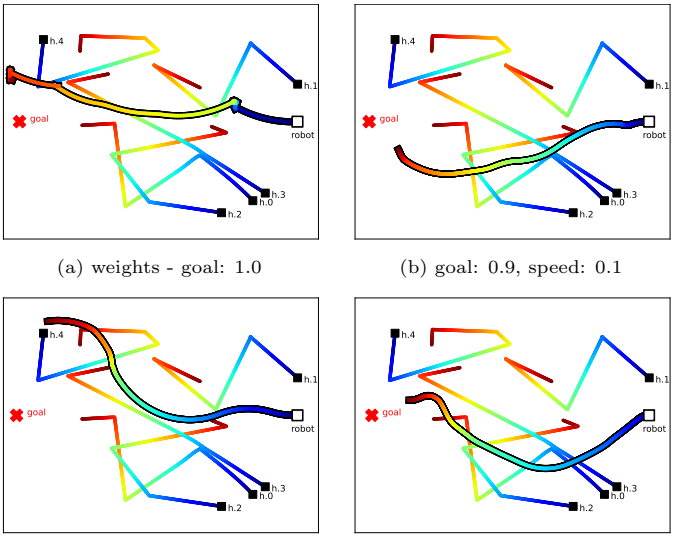


Figure 9: **Performances reached by PEARL and RBF-PEARL compared to a full training with SAC:** SAC needs several thousand environment steps before reaching the same performance as PEARL and RBF-PEARL that use only 200 environments steps after their meta-training phase. Results show the average test-task performance and standard deviation over 20 tasks and 5 seeds per environment steps. (left): Gaze control environment with linear reward functions. (middle): Gaze control environment with non-linear reward functions. (right): Racer environment. For the racer environment, SAC shows a low performance because for one set of hyperparameters it can't solve each of the 20 test tasks. PEARL and RBF-PEARL have not such a dependency on its hyperparameters.



(a) weights - goal: 1.0

(b) goal: 0.9, speed: 0.1

(c) goal: 0.75, approach: 0.25

(d) goal: 0.75, social: 0.25

Figure 10: **Trajectories in the social navigation environment for 4 different weight combinations of the reward function**. RBF-PEARL learns to generate different behaviors depending on the reward weight combination. The robot trajectory is represented by the line with the black contour. The color indicates the time step.

to modify its path consequently. However putting more weight on the speed component (b) helps the robot to reach the goal position, as the robot learns to have a better control of its angular and linear speed. By putting more weight on the social and approach components (c and d), the robot actively tries to avoid people around him even if it results on a failure to reach the goal position. In conclusion, variational meta-RL successfully allows to quickly adapt an agent to different reward functions which allows to find efficiently an appropriate behavior for different social scenarios.

5.2. Performance of PEARL vs. RBF-PEARL

We compared the performance of RBF-PEARL to two versions of PEARL. The first version, called PEARL, uses a neural network model with a similar number of parameters as RBF-PEARL. This is achieved by using a MLP layer with kd output neurons where k is the numbers of neurons of the RBF layer and d is the latent dimension, followed by a ReLU activation unit instead of a RBF layer before the actor and critic network to process the task representation z. All four experiments show that RBF-PEARL outperforms consistently PEARL on meta-test task performance (Fig. 4, 6, and 8). We further compared their asymptotic performances on the 20 meta-testing tasks in more detail (Table 1). In the two linear environments, RBF-PEARL increases the performances over PEARL by a margin of 6% in the linear gaze control environment and 8% in the social navigation environment. For the two non linear environments the improvement is larger. In the racer environment, the performance rises by 27% from the one obtained by PEARL. In the non linear gaze control environment, the difference is 13% compared to PEARL.

In addition, we also report results obtained with a version of PEARL without the additional MLP layer, called Vanilla PEARL. We find that the performances of PEARL and Vanilla PEARL are very similar in three of the four tested environments (Table 1). The differences on the average final asymptotic return between the two is less than 5% and within their confidence ranges. PEARL only improves significantly over Vanilla PEARL by 18% for the social navigation environment.

Algorithm	Lin. Gaze	Non-Lin. Gaze	Social Nav.	Racer
PEARL	516 ± 10	460 ± 16	305 ± 30	56 ± 6
Vanilla PEARL	514 ± 6	469 ± 16	253 ± 18	56 ± 5
RBF-PEARL	$\mathbf{549} \pm 5$	$\mathbf{542} \pm 10$	$\mathbf{344} \pm 31$	65 ± 3
RBF-PEARL-fp	543 ± 7	518 ± 15	322 ± 12	$\mathbf{72} \pm 6$

Table 1: **Final performances of different PEARL and RBF-PEARL versions:** RBF-PEARL outperforms PEARL and the learning of RBF parameters is in most environments beneficial over fixed parameters (RBF-PEARL-fp). Reported are the mean $\pm$ standard deviation over 5 seeds for each algorithms meta-testing performance at the end of meta-training.

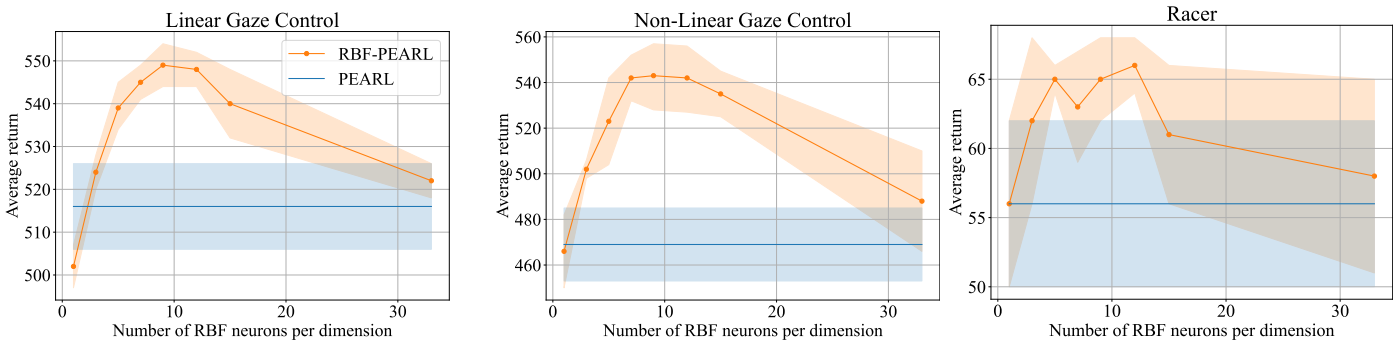


Figure 11: **Ablation study on the number of RBF neurons per input dimension for RBF-PEARL:** The optimal number of neurons is around 10 for the three evaluated environments. Reported are the mean and standard deviation over 5 seeds for each algorithms meta-testing performance after a certain amount of meta-training steps.

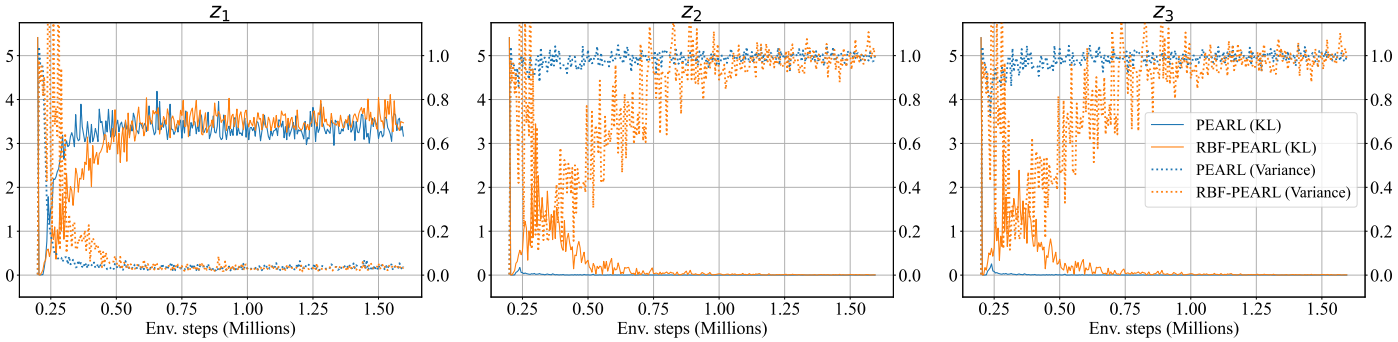


Figure 12: **Kullback-Leibler (KL) divergence and variance of task representation z for the linear gaze control environment:** Posterior collapse occurs in two dimensions (z_2, z_3) for both PEARL and RBF-PEARL (note that RBF-PEARL delays the collapse). We report the average KL divergence (solid lines) and variance (dotted lines) of each dimension of the task representation variable z during meta-training. The average is taken over five tasks chosen randomly among the 100 training tasks.

In summary, the results of PEARL and Vanilla PEARL compared to RBF-PEARL show that the RBF layer allows to improve the performance of PEARL significantly. This effect can not be explained with a difference in their model capacity, as PEARL and RBF-PEARL have a similar number of parameters. Instead, the computational properties of the RBF layer seem to be the important factors.

5.3. Impact of trainablity and number of RBF neurons

We evaluated the effect of training the RBF parameters, i.e. centers c and scaling factors δ (6), to an RBF-PEARL architecture with fixed parameters (RBF-PEARL-fp). The centers are fixed by evenly distributing them over an interval that was set to encompass the space of task representations z. The scaling factors are fixed based on a function of the distance between center points. They were chosen so that two neighboring RBF neurons have both an activation of 0.5 for a representation $z_k \in \mathbb{R}$ lying in the middle between both their centers. Overall we see that training the centers and scaling factors of the RBF layer have a beneficial impact on the asymptotic performances of the RBF-PEARL algorithm (Table 1). On the non-linear gaze control and social navigation environments, training the RBF parameters improves the final asymptotic performances by 2% to 6%. Only, in the racer environment having fixed parameters improved performance of 10%. In summary, the advantage of learning the parameters of the RBF layer is task-dependent, but seems to be for most tasks beneficial.

Lastly, we analyze the effect of the number of neurons per input dimension in the RBF layer (Fig. 11). On the three evaluated environments, we found that the number of neurons has a noticeable effect on the asymptotic performance. The optimal number is around 10 on the three evaluated environments: 9 for linear gaze control, 9 for non-linear gaze control, and 12 for racer. We believe that the performances drop for higher numbers of neurons may be due to overfitting on the training tasks.

5.4. How does the RBF layer improve performance?

The RBF layer improves the performance of the variational meta-RL procedure, but what are the mechanisms behind this improvement? In general, the layer can have three potential influences on the learning procedure. First, as its input, task representation z, is also learned, it could alter the learning objective of z resulting in a different representation. Second, it could alter the temporal learning dynamics of the task representation, leading also to different learning dynamics of the downstream policy and value networks. Third, its output $\tilde{z}$ could provide an improved representation for the policy and value networks. We investigated these factors on the linear gaze control

task (Sec. 4.2). We restricted our analysis to a RBF layer with 3 RBF neurons per input dimension. This low dimensional representation makes it easier to analyze and visualize the results compared to the optimal configuration with 9 RBF neurons per input dimension.

Influence on the learned input task representation z. The learned task representations z of PEARL without (Fig. 1, left-bottom) and with a RBF layer (Fig. 14) have only minor differences. The average and standard deviation over the 100 meta-training task representation means μ per dimension are for PEARL: z_1: 0.04 ± 3.68, z_2: 1.01 ± 0.006, z_3: 1.01 ± 0.003; and for RBF-PEARL: z_1: 0.06 ± 3.51, z_2: 1.02 ± 0.009, z_3: 1.03 ± 0.032.

Both representations have a posterior collapse in two (z_2, z_3) of the three dimensions. Only dimension z_1 is representing a meaningful distinction of tasks. The representation by PEARL without a RBF layer shows a minor larger spread of the task representations in dimension z_1 than RBF-PEARL (3.68 compared to 3.51). And RBF-PEARL has a minor larger spread in dimension z_3 (0.032 compared to 0.003), but both these differences seem negligible.

Both representations show a clear clustering of tasks where tasks with a high reward weight on the visual component (red colored) are on one side in z_1. Representation of the tasks with a high weight on the movement component are clustered on the opposite end (green colored). Tasks with a high weight on the audio component (blue/brown colored) are in the middle. Although the representations (with and without RBF layer) are inverted to each other, both have this general cluster topology which should therefore not result in a difference on the downstream networks that learn based on them.

We further evaluated if the learned representation z obtained with the RBF layer has an influence on the performance increase. We trained an actor and critic network with the pre-trained context encoder obtained from PEARL and RBF-PEARL. No significant differences neither on their learning curves nor on their final performances can be observed (Fig. 13). This indicates that differences in the performance of PEARL vs. RBF-PEARL do not result from their differences in the learned representation z.

In summary, the differences between the learned task representation z with and without the RBF layer are minor. They do not explain the increase in performance of RBF-PEARL.

Influence on the temporal dynamics of learning task representation z. We analyzed the temporal dynamics of learning z by looking how posterior collapse happens on the three dimensions of task representation z. The differences on how posterior collapse occurred between PEARL and RBF-PEARL could explain their performance differences. To measure it, we examine the KL divergence and the variance of the posterior distribution on each of the dimensions

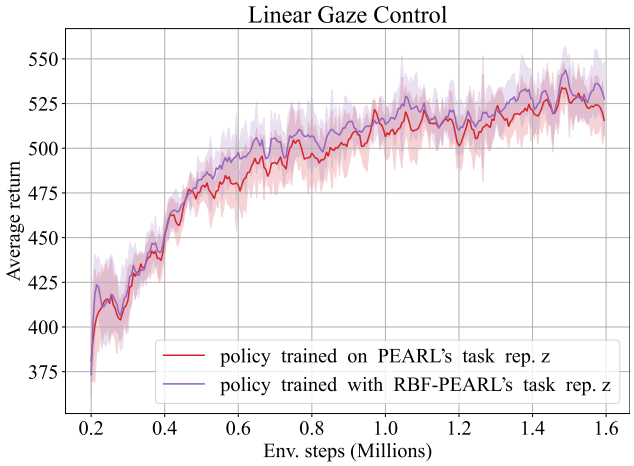
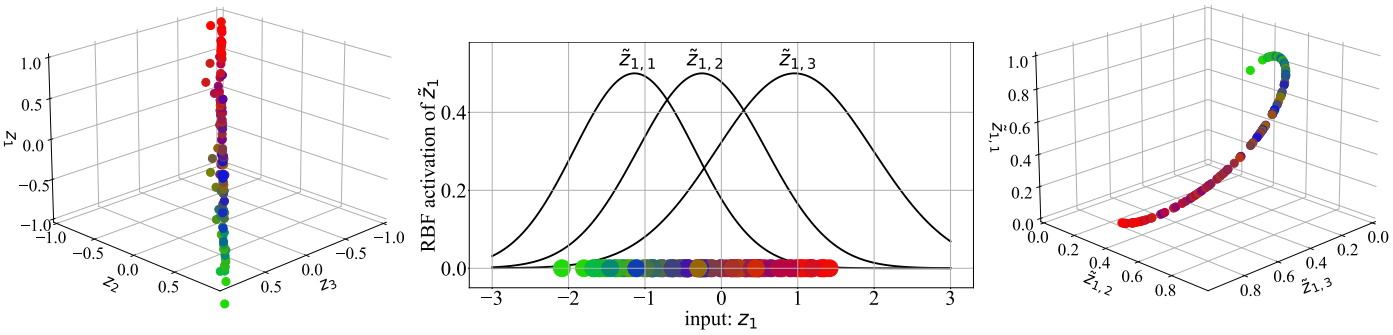


Figure 13: **Performance of policies trained on learned (and frozen) context-encoders from PEARL and RBF-PEARL:** Test-task performance during meta-training on the linear gaze-control environment. It shows no significant differences between the two encoders indicating that the performance difference between PEARL and RBF-PEARL is not due to their differences in the learned task representation z.

of the task representation during the meta-training stage (Fig. 12). Posterior collapse happens for both methods in two of the three dimensions (z_2, z_3). Nonetheless, RBF-PEARL delays the posterior collapse for 400k steps compared to PEARL. Similarly also for dimension z_1, RBF-PEARL requires longer to learn the final representation as shown by the longer time of the KL loss and variance to reach their asymptotic levels (Fig. 12, left). This delay could explain why RBF-PEARL performs slightly below PEARL for the first 400k steps (Fig. 4, left). Afterwards the KL loss and variance are similar between RBF-PEARL and PEARL.

In conclusion, the RBF layer has a temporal effect on the learning of task representation z by delaying it. This includes a delay of the posterior collapse. This might affect the final performance of the RL algorithm, for example, by inducing a higher exploration during learning. Nonetheless, the impact of this effect can not be clearly defined and we believe it to be of minor consequence for the learning performance.

Influence of the output representation $\tilde{z}$. The final influence that the RBF layer has on the performance of RBF-PEARL is by its output representation $\tilde{z}$ that is given as input to the downstream policy and value networks instead of z. We visualized this representation for the RBF neurons that encode the non-collapsing dimension z_1 (Fig. 14, right). It lifts the one-dimensional representation z_1 into a three-dimensional space $\tilde{z}_1$. Analyzing the shape of the Gaussians associated with $\tilde{z}_1$ (Fig. 14, middle), each Gaussian is centered around a specific cluster of task representations. The first Gaussian $\tilde{z}_{1,1}$ specializes in tasks with a high weight on the movement component (green colored). Representations of tasks with a high weight on the audio component (blue/brown colored) are centered around the



(a) Task Representation z (b) RBF Gaussians for z_1 (c) RBF Task Representation $\tilde{z}_1$

Figure 14: **RBF-PEARL's task representation with the Gaussian associated to dimension z_1 with no posterior collapse:** The RBF layer projects task representation $z_1 \in \mathbb{R}$ in a higher dimension $\tilde{z}_1 \in \mathbb{R}^3$, where each RBF Gaussian learns to represent a specific task type. Each colored dot corresponds to the representation of one of the 100 meta-training tasks. The color represents the pre-dominant reward weight component of the reward function, i.e. the task type. Red: largest weight is the visual weight. Blue/brown: high audio weights. Green: high movement weights. (left): Three dimensional task representation z learned by the task encoder. The tasks are only spread in dimension z_1. The other dimensions (z_2, z_3) have a posterior collapse. (middle): RBF Gaussians associated to the input dimension without posterior collapse (z_1). Each Gaussian specializes to represent a different task type. (right): The three dimensional representation obtained by the RBF layer for input dimension z_1.

activation region of the second Gaussian $\tilde{z}_{1,2}$. The third Gaussian $\tilde{z}_{1,3}$ specializes in tasks with a high weight on the visual component (red colored). We believe this effect is the main cause of the RBF layers performance increase. The objective of the downstream networks is to learn specific policies and value functions for the different tasks, i.e. tasks in which the visual, audio, or movement component is more important. Differentiating between these tasks is difficult from the one-dimensional representation z_1 learned by standard PEARL. To identify for example audio tasks which are clustered in the middle of the representation in z_1 (Fig. 1, left-bottom), the downstream networks have to learn a rule that defines this region using two borders: $y > z_1 > x$. In contrast, for RBF representation $\tilde{z}_1$ it is only necessary to identify if a certain Gaussian has a large activation. In the case of audio tasks, the second Gaussian should be mainly activated: $\tilde{z}_{1,2} > x$. This seems to reduce the complexity of the rules that the downstream networks have to learn to identify tasks making it easier to learn specific policies and values for them.

In summary, we believe the main effect that the RBF layer has to improve the performance is based on its changed task representation $\tilde{z}$. The representation seems to allow the downstream networks to identify certain tasks easier and to learn specific outputs for them. Nonetheless, this explanation is only an intuition and should be further explored in future research.

6. Conclusion

In this exploratory study, we investigate the use and limitations of variational meta-RL for social robotics. We showed that meta-RL successfully learns to adapt quickly (within 200 steps) to different reward function formulations which can help to identify the reward function that describes a wanted social behavior faster. Nonetheless, state-of-the-art methods exhibited a posterior collapse in our task, which is problematic in meta-RL since the encoder is supposed to accumulate information for better generalization, and collapsed encoding dimensions cannot do so. We started investigating how to mitigate posterior collapse in variational meta-RL by adding a RBF network after each encoded dimension. Based on the result in Tab. 1, and the analysis of the representation learned by RBF-PEARL, our algorithm improves the performances of meta-RL algorithm for reward design in social robotics. Our RBF layer improve asymptotic performances of the PEARL algorithm in several different environment, all inspired of social robotics tasks. The PEARL algorithm learns a sub-optimal representation of the task. While we do not solve this issue, the RBF-PEARL algorithm mitigates the effect of this sub-optimal representation, providing the actor and critic network with a different representation of the task using the dimensions of the task representation that do not suffer of posterior collapse. The results clearly demonstrate that the use of RBF network mitigates the effect of posterior collapse, and allows for steeper learning curves and higher asymptotic performance. We believe such studies open the door for better understanding of meta-RL for social robotics, a clearly underinvestigated domain. We hope that our findings will help fostering research in this direction.

Acknowledgements

This research was partially funded by the ANR MIAI institute (ANR-19-P3IA-0003), H2020 SPRING (#871245), and by the ANR ML3RI (ANR-19-CE33-0008-01).

References

[1] T. Fong, I. Nourbakhsh, K. Dautenhahn, A survey of socially interactive robots, Robotics and autonomous systems 42 (3-4) (2003) 143–166.

[2] R. S. Sutton, A. G. Barto, Reinforcement learning: An introduction, MIT press, 2018.

[3] N. Akalin, A. Loutfi, Reinforcement learning approaches in social robotics, Sensors 21 (4) (2021) 1292.

[4] Y. F. Chen, M. Liu, M. Everett, J. P. How, Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning, in: 2017 IEEE international conference on robotics and automation (ICRA), IEEE, 2017, pp. 285–292.

[5] Z. Zhou, P. Zhu, Z. Zeng, J. Xiao, H. Lu, Z. Zhou, Robot navigation in a crowd by integrating deep reinforcement learning and online planning, Applied Intelligence (2022) 1–17.

[6] Y. Li, Deep reinforcement learning, arXiv preprint arXiv:1810.06339.

[7] M. E. Taylor, P. Stone, Transfer learning for reinforcement learning domains: A survey., Journal of Machine Learning Research 10 (7).

[8] Z. Zhu, K. Lin, J. Zhou, Transfer learning in deep reinforcement learning: A survey, arXiv preprint arXiv:2009.07888.

[9] D. P. Kingma, M. Welling, Auto-encoding variational bayes, arXiv:1312.6114.

[10] K. Rakelly, A. Zhou, C. Finn, S. Levine, D. Quillen, Efficient off-policy meta-reinforcement learning via probabilistic context variables, in: International conference on machine learning, PMLR, 2019, pp. 5331–5340.

[11] D. S. Broomhead, D. Lowe, Radial basis functions, multivariable functional interpolation and adaptive networks, Tech. rep., Royal Signals and Radar Establishment Malvern (United Kingdom) (1988).

[12] J. Park, I. W. Sandberg, Universal approximation using radial-basis-function networks, Neural computation 3 (2) (1991) 246–257.

[13] P. H. Zadeh, R. Hosseini, S. Sra, Deep-rbf networks revisited: Robust classification with rejection, arXiv preprint arXiv:1812.03190.

[14] S. Pineda-Arango, D. Obando-Paniagua, A. Dedeoglu, P. Kurzendörfer, F. Schestag, R. Scholz, Improving sample eficiency with normalized rbf kernels, arXiv preprint arXiv:2007.15397.

[15] W. Chen, X. Han, G. Li, C. Chen, J. Xing, Y. Zhao, H. Li, Deep rbfnet: Point cloud feature learning using radial basis functions, arXiv preprint arXiv:1812.04302.

[16] K. Asadi, R. E. Parr, G. D. Konidaris, M. L. Littman, Deep rbf value functions for continuous control, arXiv preprint arXiv:2002.01883.

[17] M. Vázquez, A. Steinfeld, S. E. Hudson, Maintaining awareness of the focus of attention of a conversation: A robot-centric reinforcement learning approach, in: 2016 25th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN), IEEE, 2016, pp. 36–43.

[18] S. Lathuilière, B. Massé, P. Mesejo, R. Horaud, Neural network based reinforcement learning for audio–visual gaze control in human–robot interaction, Pattern Recognition Letters 118 (2019) 61–71.

[19] D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, D. Mané, Concrete problems in ai safety, arXiv preprint arXiv:1606.06565.

[20] R. Barraquand, J. L. Crowley, Learning polite behavior with situation models, in: 2008 3rd ACM/IEEE International Conference on Human-Robot Interaction (HRI), IEEE, 2008, pp. 209–216.

[21] H. B. Suay, S. Chernova, Effect of human guidance and state space size on interactive reinforcement learning, in: 2011 Ro-Man, IEEE, 2011, pp. 1–6.

[22] H. B. Suay, R. Toris, S. Chernova, A practical comparison of three robot learning from demonstration algorithm, International Journal of Social Robotics 4 (4) (2012) 319–330.

[23] C.-Y. Yang, M.-J. Lu, S.-H. Tseng, L.-C. Fu, A companion robot for daily care of elders based on homeostasis, in: 2017 56th Annual Conference of the Society of Instrument and Control Engineers of Japan (SICE), IEEE, 2017, pp. 1401–1406.

[24] A. L. Thomaz, G. Hoffman, C. Breazeal, Reinforcement learning with human teachers: Understanding how people want to teach robots, in: ROMAN 2006-The 15th IEEE International Symposium on Robot and Human Interactive Communication, IEEE, 2006, pp. 352–357.

[25] Y. Gao, W. Barendregt, M. Obaid, G. Castellano, When robot personalisation does not help: Insights from a robot-supported learning study, in: 2018 27th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN), IEEE, 2018, pp. 705–712.

[26] A. Tapus, C. Ţăpuş, M. J. Matarić, User—robot personality matching and assistive robot behavior adaptation for post-stroke rehabilitation therapy, Intelligent Service Robotics 1 (2) (2008) 169–183.

[27] J. Chan, G. Nejat, A learning-based control architecture for an assistive robot providing social engagement during cognitively stimulating activities, in: 2011 IEEE International Conference on Robotics and Automation, IEEE, 2011, pp. 3928–3933.

[28] P.-H. Ciou, Y.-T. Hsiao, Z.-Z. Wu, S.-H. Tseng, L.-C. Fu, Composite reinforcement learning for social robot navigation, in: 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE, 2018, pp. 2553–2558.

[29] Y. Duan, J. Schulman, X. Chen, P. L. Bartlett, I. Sutskever, P. Abbeel, Fast reinforcement learning via slow reinforcement learning, arXiv preprint arXiv:1611.02779.

[30] J. X. Wang, Z. Kurth-Nelson, D. Tirumala, H. Soyer, J. Z. Leibo, R. Munos, C. Blundell, D. Kumaran, M. Botvinick, Learning to reinforcement learn, arXiv preprint arXiv:1611.05763.

[31] N. Mishra, M. Rohaninejad, X. Chen, P. Abbeel, A simple neural attentive meta-learner, arXiv preprint arXiv:1707.03141.

[32] R. Fakoor, P. Chaudhari, S. Soatto, A. J. Smola, Meta-q-learning, arXiv preprint arXiv:1910.00125.

[33] C. Finn, P. Abbeel, S. Levine, Model-agnostic meta-learning for fast adaptation of deep networks, in: International Conference on Machine Learning, PMLR, 2017, pp. 1126–1135.

[34] B. C. Stadie, G. Yang, R. Houthooft, X. Chen, Y. Duan, Y. Wu, P. Abbeel, I. Sutskever, Some considerations on learning to explore via meta-reinforcement learning, arXiv preprint arXiv:1803.01118.

[35] J. Rothfuss, D. Lee, I. Clavera, T. Asfour, P. Abbeel, Promp: Proximal meta-policy search, arXiv preprint arXiv:1810.06784.

[36] T. Xu, Q. Liu, L. Zhao, J. Peng, Learning to explore via meta-policy gradient, in: International Conference on Machine Learning, PMLR, 2018, pp. 5463–5472.

[37] T. Z. Zhao, A. Nagabandi, K. Rakelly, C. Finn, S. Levine, Meld: Meta-reinforcement learning from images via latent state models, arXiv preprint arXiv:2010.13957.

[38] B. Dai, Z. Wang, D. Wipf, The usual suspects? reassessing blame for vae posterior collapse, in: International Conference on Machine Learning, PMLR, 2020, pp. 2313–2322.

[39] S. Zhao, J. Song, S. Ermon, Infovae: Information maximizing variational autoencoders, arXiv preprint arXiv:1706.02262.

[40] A. Razavi, A. v. d. Oord, B. Poole, O. Vinyals, Preventing posterior collapse with delta-vaes, arXiv preprint arXiv:1901.03416.

[41] C. K. Sønderby, T. Raiko, L. Maaløe, S. K. Sønderby, O. Winther, Ladder variational autoencoders, arXiv preprint arXiv:1602.02282.

[42] C.-W. Huang, S. Tan, A. Lacoste, A. Courville, Improving explorability in variational inference with annealed variational objectives, arXiv preprint arXiv:1809.01818.

[43] C. Reinke, M. Etcheverry, P.-Y. Oudeyer, Intrinsically motivated exploration for automated discovery of patterns in morphogenetic systems, in: International Conference on Learning Representations, 2020.

[44] J. He, D. Spokoyny, G. Neubig, T. Berg-Kirkpatrick, Lagging inference networks and posterior collapse in variational autoen-

coders, arXiv preprint arXiv:1901.05534.
[45] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, et al., Soft actor-critic algorithms and applications, arXiv preprint arXiv:1812.05905.
[46] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, M. Riedmiller, Playing atari with deep reinforcement learning, arXiv preprint arXiv:1312.5602.
[47] C. Pedica, H. Vilhjálmsson, Social perception and steering for online avatars, in: International Workshop on Intelligent Virtual Agents, Springer, 2008, pp. 104–116.
[48] S. Satake, T. Kanda, D. F. Glas, M. Imai, H. Ishiguro, N. Hagita, A robot that approaches pedestrians, IEEE Transactions on Robotics 29 (2) (2012) 508–524.
[49] C. Reinke, X. Alameda-Pineda, Xi-learning: Successor feature transfer learning for general reward functions, arXiv preprint arXiv:2110.15701.