Fast Adaptation to New Environments via Policy-Dynamics Value Functions

Roberta Raileanu [1] **Max Goldstein** [1] **Arthur Szlam** [2] **Rob Fergus** [1]

Abstract

Standard RL algorithms assume fixed environment dynamics and require a significant amount of interaction to adapt to new environments. We introduce Policy-Dynamics Value Functions (PD-VF), a novel approach for rapidly adapting to dynamics different from those previously seen in training. PD-VF explicitly estimates the cumulative reward in a space of policies and environments. An ensemble of conventional RL policies is used to gather experience on training environments, from which embeddings of both policies and environments can be learned. Then, a value function conditioned on both embeddings is trained. At test time, a few actions are sufficient to infer the environment embedding, enabling a policy to be selected by maximizing the learned value function (which requires no additional environment interaction). We show that our method can rapidly adapt to new dynamics on a set of MuJoCo domains. Code available at `policy-dynamics-value-functions`.

1. Introduction

Deep reinforcement learning (RL) has achieved impressive results on a wide range of complex tasks (Mnih et al., 2015; Silver et al., 2016; 2017; 2018; Jaderberg et al., 2019; Berner et al., 2019; Vinyals et al., 2019). However, recent studies have pointed out that RL agents trained and tested on the same environment tend to overfit to that environment's idiosyncracies and are unable to generalize to even small perturbations (Whiteson et al., 2011; Rajeswaran et al., 2017; Zhang et al., 2018c;a; Henderson et al., 2018; Cobbe et al., 2019; Raileanu & Rocktäschel, 2020; Song et al., 2020). It is often the case that besides the test environments being different from the train environments, they will also have costly interactions, scarce or unavailable feedback, and irreversible consequences. For example, a self-driving car might have to adjust its behavior depending on weather conditions, or a prosthetic control system might have to adapt to a new human. In these cases it is crucial for RL agents to find and execute appropriate policies as quickly as possible.

Our approach is inspired by Sutton et al. (2011) who introduced the notion of general value functions (GVFs), which can be used to gather knowledge about the world in the form of predictions. A GVF estimates the expected return of an arbitrary policy on a certain task (as defined by a reward function, a termination function and a terminal-reward function). Similarly, in this work, we aim to learn a value function conditioned on elements of a space of policies and tasks, but here, a "task" is specified by the transition function of the MDP instead of the reward function.

More specifically, we propose PD-VF, a novel framework for rapid adaptation to new environment dynamics. PD-VF consists of four phases: (i) a *reinforcement learning phase* in which individual policies are learned for each environment in our training set using standard RL algorithms, (ii) a *self-supervised phase* in which trajectories generated by these policies are used to learn embeddings for both policies and environments, (iii) a *supervised training phase* in which a neural network is used to learn the value function of a certain policy acting in some environment. The network takes as inputs the initial state of the environment, as well as the corresponding policy and environment embeddings (as learned in the previous phase) and is trained with supervision of the cumulative reward obtained during an episode, and finally (iv) an *evaluation phase* in which, given a new environment, its dynamics embedding is inferred using the first few steps of an episode. Then, a policy is selected by finding the policy embedding that maximizes the learned value function. The selected policy is used to act in the environment until the episode ends.

Our framework uses self-supervised interactions with the environment to learn an embedding space of both dynamics and policies. By learning a value function in the policy-dynamics space, PD-VF can discover useful patterns in the complex relation between a family of environment dynamics, various behaviors, and the expected return. The value function is designed to model non-optimal policies along

[1]Department of Computer Science, New York University, New York, USA [2]Facebook AI Research, New York, USA. Correspondence to: Roberta Raileanu <raileanu@cs.nyu.edu>.

Proceedings of the 37^{th} International Conference on Machine Learning, Online, PMLR 119, 2020. Copyright 2020 by the author(s).

with optimal policies in given environments so that it can understand how changes in dynamics relate to changes in the return of different policies. PD-VF uses the learned space of dynamics to rapidly embed a new environment in that space using only a few interactions. At test time, PD-VF can evaluate or rank policies (from a certain family) on unseen environments without the need of full rollouts (*i.e.* it does not require full trajectories or rewards to update the policy). We evaluate our method on a set of continuous control tasks (with varying dynamics) in MuJoCo (Todorov et al., 2012). The dynamics of each task instance are determined by physical parameters such as wind direction or limb length and can be sampled from a continuous or discrete distribution. Performance is evaluated on a single episode at test time to emphasize rapid adaptation. We show that PD-VF outperforms other meta-learning and transfer learning approaches on new environments with unseen dynamics.

2. Related Work

Our work draws inspiration from multiple research areas such as transfer learning (Taylor & Stone, 2009; Higgins et al., 2017), skill and task embedding (Devin et al., 2016; Zhang et al., 2018b; Hausman et al., 2018; Petangoda et al., 2019), and general value functions (Precup et al., 2001; Sutton et al., 2011; White et al., 2012).

Multi-Task and Transfer Learning. Taylor & Stone (2009) presents an overview of transfer learning methods in RL. A popular approach for transfer in RL is multi-task learning (Taylor & Stone, 2009; Teh et al., 2017), a paradigm in which an agent is trained on a family of related tasks. By simultaneously learning about different tasks, the agent can exploit their common structure, which can lead to faster learning and better generalization to unseen tasks from the same family (Taylor & Stone, 2009; Lazaric, 2012; Ammar et al., 2012; 2014; Parisotto et al., 2015; Borsa et al., 2016; Gupta et al., 2017; Andreas et al., 2017; Oh et al., 2017; Hessel et al., 2019). A large body of work has been inspired by the Horde architecture (Sutton et al., 2011), which consists of a number of RL agents with different policies and goals. Each agent is tasked with estimating the value function of a particular policy on a given task, thus collectively representing knowledge about the world. Building on these ideas, other methods leverage the shared dynamics of the tasks (Barreto et al., 2017; Zhang et al., 2017; Madjiheurem & Toni, 2019) or the similarity among value functions and the associated optimal policies (Schaul et al., 2015; Borsa et al., 2018; Hansen et al., 2019; Siriwardhana et al., 2019). These approaches assume the same underlying transition function for all tasks. In contrast, we focus on transferring knowledge across tasks with different dynamics.

Meta-Learning and Robust Transfer. A popular approach for fast adaptation to new environments is meta reinforcement learning (meta RL) (Cully et al., 2015; Finn et al., 2017; Wang et al., 2017; Duan et al., 2016; Xu et al., 2018; Houthooft et al., 2018; Sæmundsson et al., 2018; Nagabandi et al., 2018; Humplik et al., 2019; Rakelly et al., 2019). Meta RL methods have been designed to work well with dense reward and recent work has shown that they struggle to learn from a limited number of interactions and optimization steps at test time (Yang et al., 2019). In contrast, our framework is capable of rapid adaptation to new environment dynamics and does not require dense reward or a large number of interactions to find a good policy. Moreover, PD-VF does not update the model parameters at test time, which makes it less computationally expensive than meta RL. Another common approach for transfer across dynamics is model-based RL, which uses Gaussian processes (GPs) or Bayesian neural networks (BNNs) to estimate the transition function (Doshi-Velez & Konidaris, 2013; Killian et al., 2017). However, such methods require fictional rollouts to train a policy from scratch at test time, which makes them computationally expensive and limits their applicability for real-world tasks. Yao et al. (2018) uses a fully-trained BNN to further optimize latent variables during a single test episode, but requires an optimal policy for each training instance, which makes it harder to scale. Robust transfer methods either require a large number of interactions at test time (Rajeswaran et al., 2017) or assume that the distribution over hidden variables is known or controllable (Paul et al., 2018). An alternative approach was proposed by Pinto et al. (2017) who use an adversary to perturb the system, achieving robust transfer across physical parameters such as friction or mass.

Skill and Task Embeddings. A large body of work proposes the use of learned skill and task embeddings for transfer in RL (Da Silva et al., 2012; Sahni et al., 2017; Oh et al., 2017; Gupta et al., 2017; Hausman et al., 2018; He et al., 2018). For example, Hausman et al. (2018) use approximate variational inference to learn a latent space of skills. Similarly, Arnekvist et al. (2018) learn a stochastic embedding of optimal Q-functions for various skills and train a universal policy conditioned on this embedding. In both Hausman et al. (2018) and Arnekvist et al. (2018), adaptation to a new task is done in the latent space with no further updates to the policy network. Co-Reyes et al. (2018) learn a latent space of low-level skills that can be controlled by a higher-level policy, in the context of hierarchical reinforcement learning. This embedding is learned using a variational autoencoder (Kingma & Welling, 2013) to encode state trajectories and decode states and actions. Zintgraf et al. (2018) use a meta-learning approach to learn a deterministic task embedding. Wang et al. (2017) and Duan et al. (2017) learn embeddings of expert demonstrations to aid imitation learning using variational and deterministic methods, respectively. More recently, Perez et al. (2018) learn dynamic models with

auxiliary latent variables and use them for model-predictive control. Zhang et al. (2018b) use separate dynamics and reward modules to learn a task embedding. They show that conditioning a policy on this embedding helps transfer to changes in transition or reward function. While the above approaches might learn embeddings of skills or tasks, none of them leverage *both* the latent space of policies and that of the environments for estimating the expected return and using it to select an effective policy at test time.

More similar to our work is that of Yang et al. (2019), who also focus on fast adaptation to new environment dynamics and evaluate performance on a single episode at test time. Yang et al. (2019) train an inference model and a probe to estimate the underlying latent variables of the dynamics, which are then used as input to a universal control policy. While similar in scope, our approach is significantly different from that of Yang et al. (2019). Importantly, Yang et al. (2019) does not learn a latent space of policies and instead trains a universal policy on all the environments. Learning a value function in a space of policies and dynamics allows the function approximator to capture relations among dynamics, behaviors (both optimal as well as non-optimal), and rewards that a universal policy cannot learn. Moreover, the learned structure can aid transfer to new dynamics.

3. Policy-Dynamics Value Functions

In this work, we aim to design an approach that can quickly find a good policy in an environment with new and unknown dynamics, after being trained on a family of environments with related dynamics. The problem can be formalized as a family of Markov decision processes (MDPs) defined by $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma)$, where $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \gamma)$ are the corresponding state space, action space, reward function, and discount factor. Each instance of the family is a stationary MDP with transition function $\mathcal{T}_d(s'|s,a) \in \mathcal{T}$. Each $\mathcal{T}_d$ has a hidden parameter d that is sampled once from a distribution $\mathcal{D}$ and held constant for that instance (*i.e.* episode). $\mathcal{T}_d$ can be continuous or discrete in d. By design, the latent variable d that defines the MDP's dynamics cannot be observed from individual states.

We present Policy-Dynamics Value Functions (PD-VF), a novel framework for rapid adaptation across such MDPs with different dynamics. PD-VF is an extension of a value function that not only conditions on a state, but also on a policy and a transition function.

A conventional value function $V : \mathcal{S} \to \mathcal{R}$ is defined as the expected future return from state s of policy π:

$$V(s) = \mathbb{E}\left[G_t | S_t = s\right] = \mathbb{E}\left[\sum_{k=t+1}^{T} \gamma^k r_k | S_t = s\right].$$

Formally, we define a *policy-dynamics value function* or PD-VF as a function $W : \mathcal{S} \times \Pi \times \mathcal{T} \to \mathcal{R}$ with two auxiliary inputs representing the policy π and the dynamics d:

$$W(s, \pi, d) = \mathbb{E}\left[G_t | S_t = s, A_t \sim \pi, S_{t+1} \sim \mathcal{T}_d\right].$$

3.1. Problem setup

The dynamics distribution $\mathcal{D}$ is partitioned into two disjoint sets $\mathcal{D}_{train}$ and $\mathcal{D}_{test}$. These are used to generate a set of training and test environments, each having different transition functions, drawn from their respective distributions.

Our model is learned on the training environments in three stages: (i) a reinforcement learning phase, (ii) a self-supervised phase and (iii) a supervised phase. The resulting PD-VF model is evaluated on test environments, where it only experiences a single episode in each. This evaluation setting probes PD-VF's ability to very quickly adapt to previously unseen dynamics.

3.2. Reinforcement Learning Phase

The first phase of training uses standard model-free RL algorithms to acquire experience in the training environments. An ensemble of N policies are trained, each with a different random seed on one of the training environments. For each policy, we save a number of checkpoints at different stages throughout training. Then, we collect trajectories using each of these checkpoints in each of our training environments. This results in experience from a diverse set of policies (some good, some bad) across environments with different dynamics. Importantly, this dataset contains the behaviors of policies in environments they haven't been trained on. In the next section, we describe how the collected trajectories are used to learn policy and dynamics embeddings.

3.3. Self-Supervised Learning Phase

The goal of this phase is to learn an embedding space of the dynamics that captures variations in the transition function, as well as an embedding space of the policies that captures variations in the agent behavior. The space of dynamics is learned using an encoder E_d parameterised as a Transformer (Vaswani et al., 2017), and a decoder D_d parameterised as a feed-forward network. The encoder takes as input a *set* of transitions $\{(s_t, a_t, s_{t+1})\}$ from the first N_d steps in each episode and outputs a vector embedding for the dynamics z_d. The decoder takes as inputs the state s_t, action a_t and dynamics embedding z_d, and predicts the next state $\hat{s}_{t+1}$. The parameters θ_d and ϕ_d of the encoder and decoder are trained to minimize the ℓ_2 error of $\hat{s}_{t+1}$ and s_{t+1}. Formally,

$$z_d = E_d(\{(s_t, a_t, s_{t+1})\};\ \theta_d)$$

$$\hat{s}_{t+1} = D_d(s_t, a_t, z_d;\ \phi_d).$$

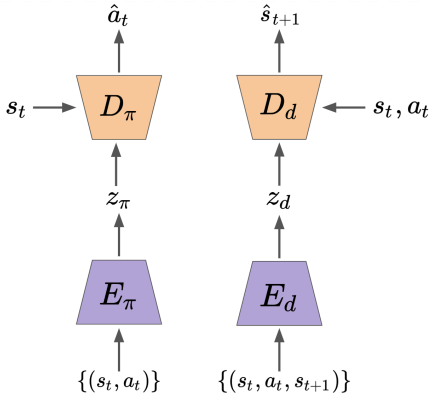


Figure 1. In the **self-supervised learning phase**, a pair of autoencoders is trained using transitions generated by a diverse set of policies in a set of environments with different dynamics. By exploiting the Markov property of the environment, distinct latent embeddings of the dynamics z_d and policy z_π are produced.

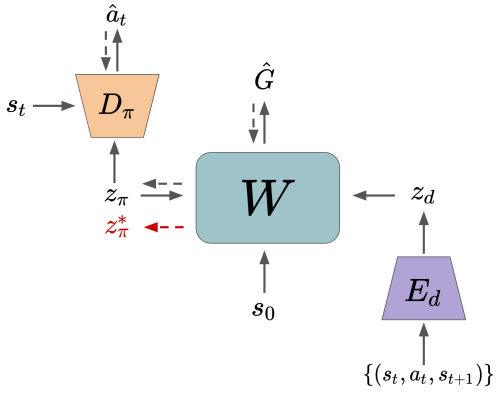


Figure 2. In the **supervised learning phase**, a parametric value function W is trained to predict the expected return G for an entire space of policies and dynamics. W takes as inputs the initial state s_0, policy embedding z_π, and dynamics embedding z_d (estimated from a small set of transitions). We train W in a supervised fashion, using Monte-Carlo estimates of the expected return G for policy π in environment with dynamics $\mathcal{T}_d$. At test time, z_π is optimized to maximize $\hat{G}$ (red dashed arrow), resulting in z_π^* which is then decoded to an actual policy via D_π.

This arrangement exploits the inductive bias that, conditioned on d, the environment is Markovian. By using no positional encoding in the Transformer, the input transitions lack any temporal ordering, thus preserving the Markov property. The decoder receives no historical information (since it is unnecessary in a Markovian setting), so it is forced to embed information about the dynamics into z_d to make good predictions. Because the input set contains the actions in each triple, the encoder has no incentive to encode policy information into z_d. This modeling choice encourages z_d to only contain information about the dynamics, rather than the policy used to generate the transitions.

Similarly, the space of policies is learned using an encoder E_π parameterised as a Transformer and a decoder D_π parameterised as a feed-forward network. The encoder takes as input a set (again using the Markov property as an inductive bias) of state-action pairs $\{(s_t, a_t)\}$ from a full episode and outputs a vector embedding for the policy z_d. The decoder takes as inputs the state s_t and the policy embedding z_π to predict the action taken by the policy $\hat{a}_t$. Since the policy encoder does not have direct access to full environment transitions, z_π is constrained to capture information about the policy without elements of the dynamics. The parameters θ_π and ϕ_π of the encoder and decoder are trained to minimize the ℓ_2 error of $\hat{a}_t$ and a_t. Formally,

$$z_\pi = E_\pi(\{(s_t, a_t)\};\ \theta_\pi)$$

$$\hat{a}_t = D_\pi(s_t, z_\pi;\ \phi_\pi).$$

Both the policy and the dynamics embeddings are normalized to have unit ℓ_2-norm.

See Figure 1 for an overview of the self-supervised learning phase.

3.4. Supervised Learning Phase

In this phase, the goal is to train an estimator W of the expected return $\hat{G}$ for a space of policies and dynamics. More specifically, W is a function approximator conditioned on the learned policy and dynamics embeddings, z_π and z_d.

A central idea of our PD-VF framework is that W provides a scoring function over the policy embedding space. It thus provides a mechanism to allow on-the-fly optimization of z_π with respect to the estimated return $\hat{G}$, without the need for any environment interaction, given an estimate (or embedding) of the environment's dynamics. This is key to PD-VF's ability to rapidly find an effective policy in a new environment, only requiring enough environment interaction to give a reliable estimate of the dynamics embedding z_d (just a few steps in practice). We choose W to have a quadratic form to permit easy optimization with respect to z_π:

$$\hat{G} = W(s_0, z_\pi, z_d) = z_\pi^T\, A(s_0, z_d; \psi)\, z_\pi.$$

The matrix $A(s_0, z_d; \psi)$ is a function of the initial environment state s_0 as well as the dynamics embedding z_d. Note that A only needs to model the initial state s_0 rather than an arbitrary state s since the optimization w.r.t z_π occurs only once, at the start of an episode. Since A must be Hermitian positive-definite, a feed-forward network with parameters ψ is first used to obtain a lower triangular matrix $L(s_0, z_d; \psi)$. Then A is constructed from LL^T.

Optimizing the policy embedding z_π: The optimization of the policy embedding z_π has a closed-form solution which is achieved by performing a singular value decomposition,

$A = USV^T$, and taking the top singular vector of this decomposition z_π^*. Unit ℓ_2 normalization is then applied to z_π^*. We refer to this vector z_π^* as the *optimal policy embedding* (OPE) of the PD-VF.

Learning ψ – Initial stage: We collect training data for the PD-VF in the following manner. First, we randomly select a policy and an environment from our training set (described in Section 3.2). Second, we generate full trajectories of that policy in the selected environment and cache the average return obtained across all episodes. This gives us a Monte-Carlo estimate for the expected return of the corresponding policy in that particular environment. Then, we use the first N_d steps of that trajectory to infer the dynamics embedding. Similarly, we use the full trajectory to infer the policy embedding (via E_π, not the above optimization procedure). After collecting this data into a buffer, we train the estimator W in a supervised fashion by predicting the expected return G given an initial state s_0, a policy embedding z_π and a dynamics embedding z_d.

Learning ψ – Data Aggregation for the Value Function: For the method to work well, it is important that the learned value function W makes accurate predictions for the entire policy space, and especially for the OPE z_π^* (which correspond to the policies selected to act in the environment). One way to ensure that these estimates are accurate is by adding the OPEs to the training data. After initial training of the PD-VF on the original dataset of policy and dynamics embeddings, we use an iterative algorithm that alternates between collecting a new dataset of OPEs and training the PD-VF on the aggregated data (including the original data as well as data added from all previous iterations). We use early stopping to select the best value function (*i.e.* the one with the lowest loss) to be used at test time.

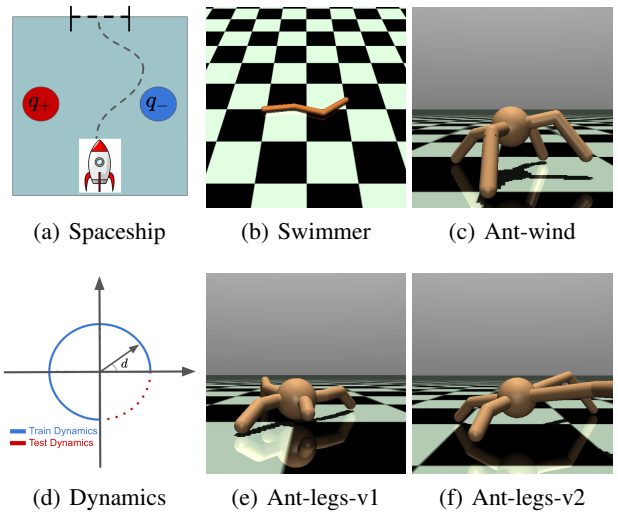
Learning ψ – Data Aggregation for the Policy Decoder: Similarly, the policy decoder may poorly estimate an agent's actions in states not seen during training. Thus, we iteratively train the policy decoder using a combination of the original set of states as well as new states generated by the policy embeddings that maximize the current value function. More specifically, we use the current OPEs (corresponding to the policies that PD-VF thinks are best) as inputs to the policy decoder to generate actions and interact with the environment. Then, we add the states visited by this policy to the data. The policy decoder is trained using the aggregated collection of states which includes both the states visited by the original collection of policies as well as the states visited by the current OPEs selected by the PD-VF.

See Figure 2 for an overview of the supervised learning phase.

3.5. Evaluation Phase

At test time, we want to find a policy that performs well on a single episode of an environment with unseen dynamics. This proceeds as follows: (i) the agent uses one of the pre-trained RL policies to act for N_d steps; (ii) the generated transitions are then used to infer the dynamics embedding z_d; (iii) once an estimate of the dynamics is obtained, the matrix $A(s_0, z_d;\ \psi)$ can be computed; (iv) we employ the closed-form optimization described above to compute the optimal policy embedding z_π^*; (v) the policy decoder, conditioned on the z_π^* embedding, is then used to take actions in the environment until the end of the episode. Note that only a small number of interactions with a new environment is needed in order to adapt, the policy selection being performed internally within the PD-VF model. Performance is evaluated on a single trajectory of each environment instance.

4. Experiments



(a) Spaceship (b) Swimmer (c) Ant-wind

(d) Dynamics (e) Ant-legs-v1 (f) Ant-legs-v2

Figure 3. (a) - (c) illustrate the continuous control domains used for testing adaptation to unseen environment dynamics. In Spaceship, Swimmer, and Ant-wind, the train and test distribution of the dynamics is continuous as illustrated in (d). (e) and (f) show two instances of the Ant-legs task in which limb lengths sampled from a discrete distribution determine the dynamics.

4.1. Experimental Setup

We evaluate PD-VF on four continuous control domains, and compare it with an upper bound, four baselines, and four ablations. For each domain, we create a number of environments with different dynamics. Then, we split the set of environments into training and test subsets, so that at test time, the agent has to find a policy that behaves well on unseen dynamics. For all our experiments, we show the

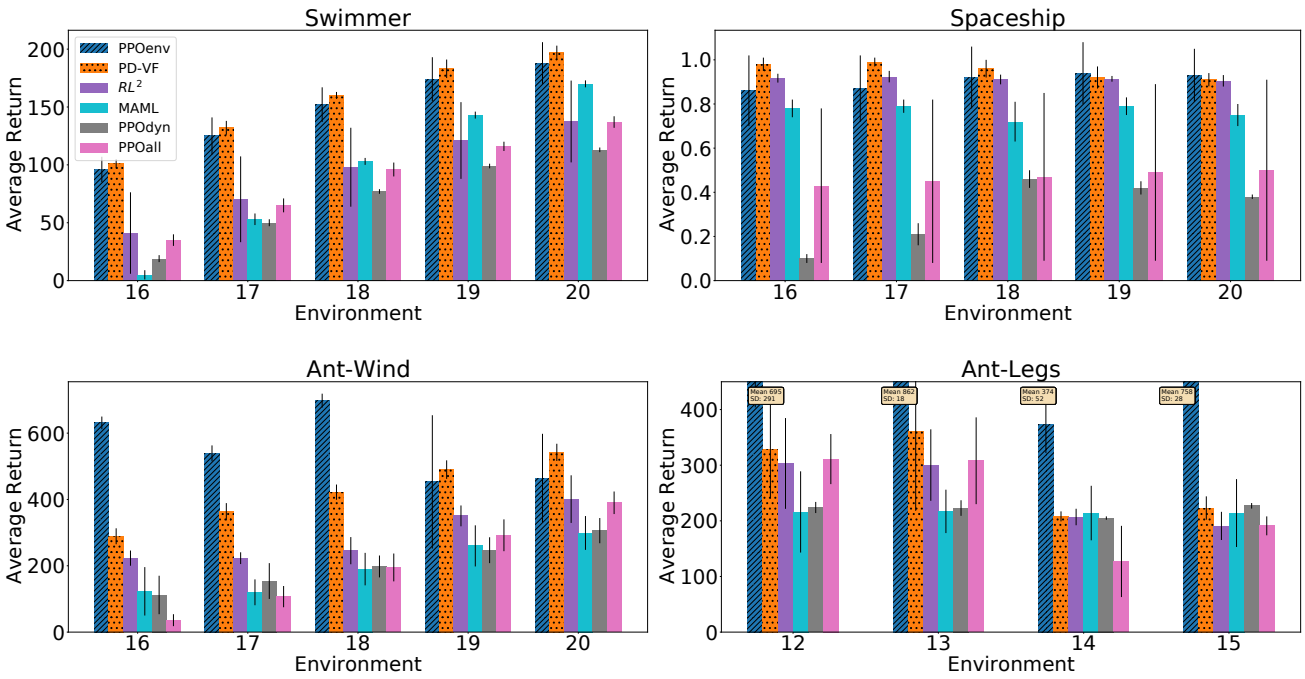


Figure 4. **Test Performance.** Average return on test environments with unseen dynamics in Swimmer (top-left), Spaceship (top-right), Ant-wind (bottom-left), and Ant-legs (bottom-right) obtained by PD-VF, the upper bound PPOenv, and baselines RL^2, MAML, PPOdyn, and PPOall. PD-VF outperforms these baselines on most test environments and, in some cases, it is comparable with PPOenv (which was trained directly on the test environments).

mean and standard deviation of the average return (over 100 episodes) across 5 different seeds of each model. The dynamics embeddings are inferred using at most $N_d = 4$ interactions with the environment.

4.2. Environments

Spaceship is a new continuous control domain designed by us. The task consists of moving a spaceship with a unit point charge from one end of a 2D room through a door at the other end. The action space consists of a fixed-magnitude force vector that is applied at each timestep. The room contains two fixed electric charges that deflect / attract the ship as it moves through the environment (see Figure 3(a)). The polarity and magnitude of these charges are parameterised by d and determine the environment dynamics. The distribution of dynamics $\mathcal{D}$ is chosen to be circular and centered (see Figure 3(d)). Samples d are drawn at intervals of $\pi/10$, each forming a different environment instance with charge configuration $(\cos(d), \sin(d))$. The 5 samples in the range $[\frac{3}{4}2\pi, \ldots, 2\pi]$ are held out as evaluation environments, the rest being used for training.

Swimmer is a family of environments with varying dynamics based on MuJoCo's Swimmer-v3 domain (Todorov et al., 2012). The goal is to control a three-link robot in a viscuous fluid to swim forward as fast as possible (Figure 3(b)). The dynamics are determined by a 2D current within the fluid, whose direction changes between environments (but has fixed magnitude). The current direction is determined by an angle d, which is sampled in the same manner as for Spaceship above, *i.e.* train on $3/4$ of all possible directions and hold out the other $1/4$ for evaluation.

Ant-wind is a family of environments based on MuJoCo's Ant-v3 domain in which the goal is to make a four-legged creature walk forward as fast as possible (Figure 3(c)). The environment dynamics are determined by the direction of a wind d, which is sampled from a continuous distribution in the same way as for Swimmer.

Ant-legs is a second task based on MuJoCo's Ant-v3 domain, in which the dynamics are sampled from a discrete distribution. The training environments are generated by fixing three ankle lengths (short, medium, and long) and generating all possible permutations for the four legs. The length of the ant leg is fixed to medium across all training environments. Symmetries in the training environments are removed by considering ants with the same number of short, medium, or long legs to be the same and choosing one ant from each equivalency class. There are four test environments with both the leg and ankle lengths being either short or long. Note that the test environments are significantly different from all the training ones, thus making Ant-legs

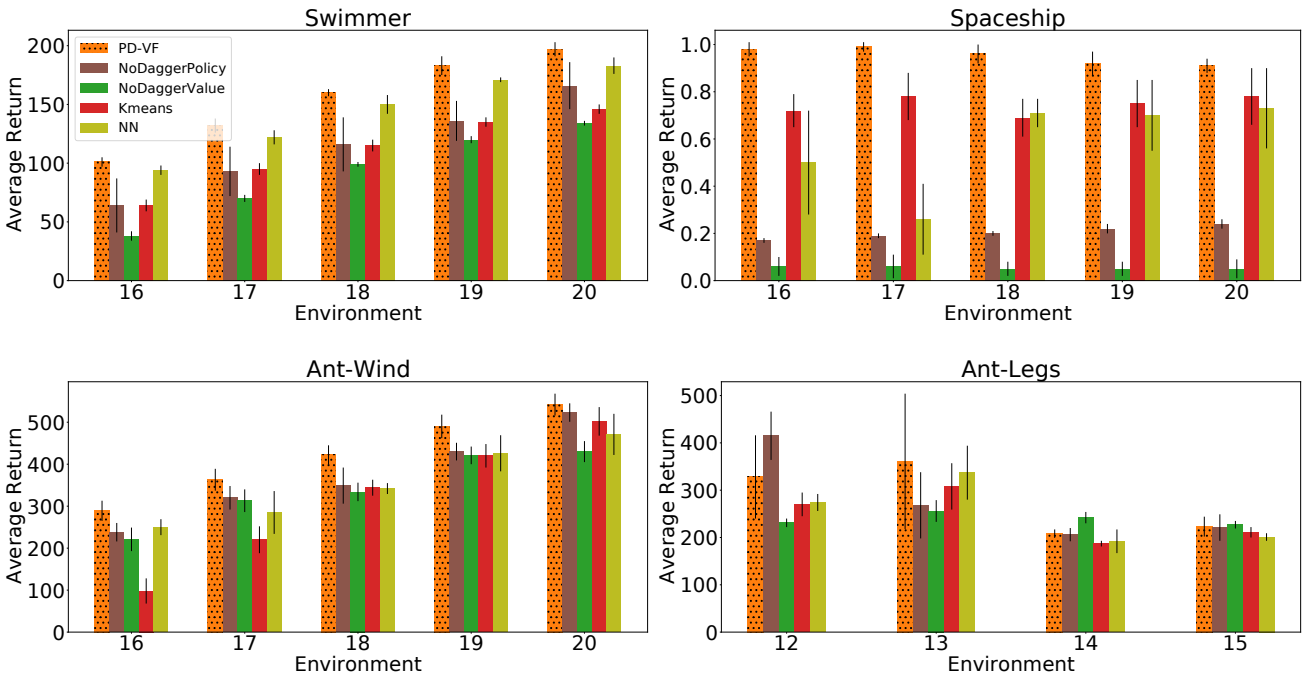


Figure 5. **Test Performance.** Average return in Swimmer (top-left), Spaceship (top-right), Ant-wind (bottom-left), and Ant-legs (bottom-right) obtained by PD-VF, NoDaggerPolicy, NoDaggerValue, Kmeans, and NN. PD-VF is better than these ablations overall.

a challenging setting for our method. Figures 3(e) and 3(f) show two instances of this environment.

4.3. Baselines

We use PPO (Schulman et al., 2017) as the base RL algorithm for all the baselines and for the reinforcement learning phase of training the PD-VF (Sec. 3.2). We use Adam (Kingma & Ba, 2014) for optimization. All models use the same network architecture for the policy and value functions. For a given environment, all methods use the same number of steps N_d (at the beginning of each episode) to infer the embedding of the environment dynamics. Then, they each use a single policy network to act in the environment until the end of the episode. We report the cumulative reward obtained by each method throughout an episodes (in which they first infer the environment dynamics which determines the policy used for acting until the end of the episode). We compare with the following baselines:

PPOenv trains a PPO policy for each environment in our set. This is used as an upper bound for the other models.

MAML is the meta-learning algorithm from Finn et al. (2017). MAML generally requires some amount of training on the test environments, so to make it more comparable to our method and the other baselines, we allow one gradient step using a trajectory of length N_d (i.e. the same length as the one used by PD-VF to infer the embedding of the environment dynamics). Thus, MAML has an advantage over PD-VF which does not make any parameter updates at test time.

RL^2 is the meta-learning algorithm from Wang et al. (2016) and Duan et al. (2016), which uses a recurrent policy that takes as input the previous action and reward.

PPOdyn trains (using PPO) a single policy network conditioned on the dynamics embedding. At test time, it first infers the dynamics embedding and then conditions the pretrained policy network on that vector. This is a close implementation of the approach in Yang et al. (2019)[1].

PPOall trains a single PPO policy on all the training environments and uses it on the test environments without any additional fine-tuning.

We also compare PD-VF with four ablations:

NN finds the environment that is closest (in Euclidean metric) to the test environment's embedding and uses the PPOenv policy trained on that environment to act. This ablation aims to tease out the effect of using both the learned space of policies and that of dynamics to adapt to new environments, from that of only using the learned dynamics space.

[1] An exact match was not feasible as code for Yang et al. (2019) was not available.

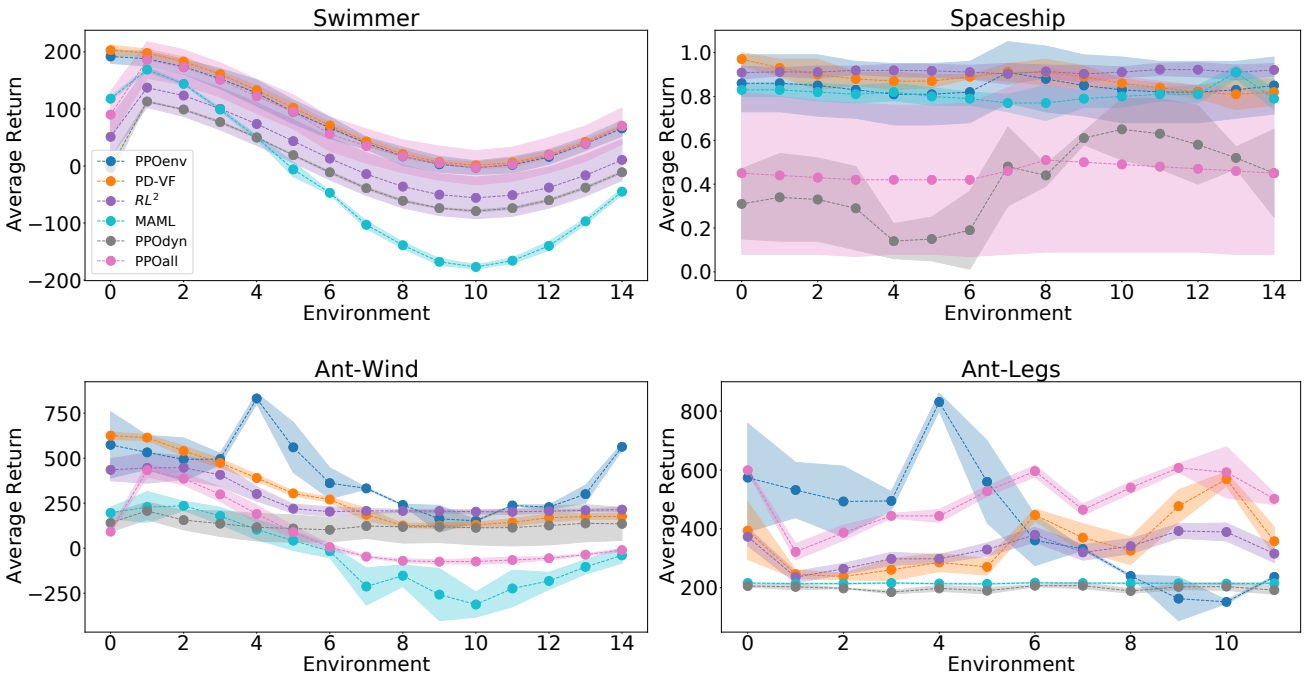


Figure 6. **Train Performance.** Average return on train environments in Swimmer (top-left), Spaceship (top-right), Ant-wind (bottom-left), and Ant-legs (bottom-right) obtained by PD-VF, the upper bound PPOenv, and baselines RL^2, MAML, PPOdyn, and PPOall. PD-VF outperforms the baselines and ablations on most test environments and, in some cases, it is comparable with PPOenv (which was trained directly on the test environments). While other methods also perform reasonably well on the training environments, they generalize poorly to new environments with unseen dynamics.

Kmeans clusters the environment embeddings (using trajectories collected in Section 3.2) into K clusters. Then, for each cluster, we train a new PPO policy on all the environments assigned to that cluster. At test time, we find the closest cluster for the given environment embedding and use the policy corresponding to that cluster to act in the environment.

NoAggValue trains a PD-VF without using dataset aggregation for the value function (see Section 3.4).

NoAggPolicy uses PD-VF without using dataset aggregation for the policy decoder (see Section 3.4).

5. Results

5.1. Adaptation to New Environment Dynamics

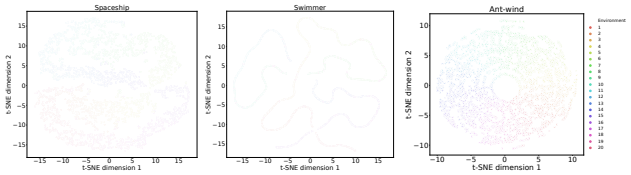
As seen in Figures 4 and 5, PD-VF outperforms all other methods on test environments with new dynamics. In some cases (particularly on Spaceship and Swimmer), our approach is comparable to the PPOenv upper bound which was directly trained on the respective test environment (in contrast, PD-VF has never interacted with that environment before). While the strength of PD-VF lies in quickly adapting to new dynamics, its performance on training environments is still comparable to that of the other baselines, as shown in Figure 6. This result is not surprising since current state-of-the-art RL algorithms such as PPO can generally learn good policies for the environments they are trained on, given enough interactions, updates, and the right hyperparamters. However, as predicted, standard model-free RL methods such as the baseline PPOall do not generalize well to environments with dynamics different from the ones experiences during training. Even meta-learning approaches like MAML or RL^2 struggle to adapt when they are allowed to use only a short trajectory for updating the policy at test time, as is the case here.

But most importantly, PD-VF also outperforms the approaches that use the dynamics embedding such as NN, Kmeans, and PPOdyn. This supports our claim that learning a value function for an entire space of policies (rather than for a single optimal policy as standard RL methods do) can be beneficial for adapting to unseen dynamics. By simultaneously estimating the return of a collection of policies in a family of environments with different but related dynamics, PD-VF can learn how variations in dynamics relate to differences in the performance of various policies. This allows the model to rank different policies and understand that sub-optimal behaviors in certain environments might

be optimal in others. Thus, at least in theory, PD-VF has the ability to find policies that are better than the ones seen during training. Our empirical results indicate that this might also hold true in practice. Overall, PD-VF proves to be more robust to changes in dynamics relative to the other methods, especially in completely new environments.

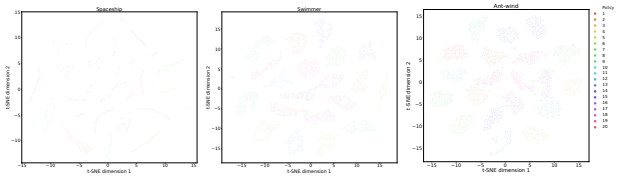
5.2. Analysis of Learned Embeddings

The performance of PD-VF relies on learning useful policy and dynamics embeddings that capture variations in agent behaviors and transition functions, respectively. In this section, we analyze the learned embeddings.



(a) (b) (c)

Figure 7. t-SNE plots of the learned **environment embeddings** z_d for Spaceship (a), Swimmer (b), and Ant-wind (c). The color corresponds to the *environment* that generated the transitions used to encode the corresponding dynamics embeddings. The plot contains embeddings of both train and test environments.



(a) (b) (c)

Figure 8. t-SNE plots of the learned **policy embeddings** z_π for Spaceship (a), Swimmer (b), and Ant-wind (c). The color corresponds to the *policy* that generated the transitions used to encode the corresponding policy embeddings. The plot contains embeddings of policies trained on both train and test environments.

Figure 7 shows a t-SNE plot (van der Maaten & Hinton, 2008) of the learned dynamics embeddings on the three continuous control domains used for evaluating our method. Environment i corresponds to dynamics defined by $d = i \times \pi/10$ (*i.e.* the direction of the wind in Swimmer's environment 1 is at $\pi/10$ degrees). Environments 1 - 15 are used for training, while 16 - 20 are used for evaluation. The latent space captures the continuous nature of the distribution used to generate the environment dynamics. For example, in Figure 7(c), one can see the wind direction corresponding to a particular environment, indicating that the learned embedding space uncovers the manifold structure of the true dynamics distribution. Even if, during training, the dynamics model never sees trajectories through the test environments, it is still able to embed them within the 1D manifold, thus preserving smoothness in the latent space.

Similarly, Figure 8 shows the corresponding t-SNE (van der Maaten & Hinton, 2008) of the learned policy embeddings for Spaceship, Swimmer, and Ant. The embeddings are clustered according to the policy that generated them.

6. Discussion and Future Work

In this work, we propose policy-dynamics value functions (PD-VF), a novel framework for fast adaptation to new environment dynamics. The key idea is to learn a value function conditioned on both a policy and a dynamics embedding which are learned in a self-supervised way. At test time, the environment embedding can be inferred from only a few interactions, which allows the selection of a policy that maximizes the learned value function. PD-VF has a number of desirable properties: it leverages the structure in both the policy and the dynamics space to estimate the expected return, it only needs a small number of steps to adapt to unseen dynamics, it does not update any parameters at test time, and it does not require dense reward or long rollouts to find an effective policy in a new environment. Empirical results on a set of continuous control domains show that PD-VF outperforms other methods on unseen dynamics, while being competitive on training environments.

PD-VF opens up many promising directions for future research. First of all, the formulation can be extended to estimate the value function not only for a family of policies and environment dynamics, but also for a family of reward functions. Another avenue for future research is to use a more general class of function approximators (such as neural networks) to parameterise the value estimator instead of a quadratic form. The PD-VF framework can, in principle, also be used to evaluate a family of policies and environments on other metrics of interest besides the expected return, such as, for example, reward variance, agent prosociality, deviation from expert behavior, and so on. Another interesting direction is to integrate additional constraints (or prior knowledge) to the optimization problem (*e.g.* maximize expected return while only using policies in a certain region of the policy space). As noted by Precup et al. (2001), Sutton et al. (2011), and White et al. (2012), learning about multiple policies in parallel via general value functions can be useful for lifelong learning. Similarly, PD-VF can be a useful tool for an agent to continually gather knowledge about various policies and dynamics in the world. Finally, PD-VF can also be applied to multi-agent settings for adapting to different opponents or teammates whose behaviors determine the environment dynamics.

Acknowledgements

Roberta and Max were supported by the DARPA L2M grant.

References

Ammar, H. B., Tuyls, K., Taylor, M. E., Driessens, K., and Weiss, G. Reinforcement learning transfer via sparse coding. In *Proceedings of the 11th international conference on autonomous agents and multiagent systems*, volume 1, pp. 383–390. International Foundation for Autonomous Agents and Multiagent Systems . . . , 2012.

Ammar, H. B., Eaton, E., Taylor, M. E., Mocanu, D. C., Driessens, K., Weiss, G., and Tuyls, K. An automated measure of mdp similarity for transfer in reinforcement learning. In *Workshops at the Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2014.

Andreas, J., Klein, D., and Levine, S. Modular multitask reinforcement learning with policy sketches. In *International Conference on Machine Learning*, pp. 166–175, 2017.

Arnekvist, I., Kragic, D., and Stork, J. A. Vpe: Variational policy embedding for transfer reinforcement learning. *2019 International Conference on Robotics and Automation (ICRA)*, pp. 36–42, 2018.

Barreto, A., Dabney, W., Munos, R., Hunt, J. J., Schaul, T., van Hasselt, H. P., and Silver, D. Successor features for transfer in reinforcement learning. In *Advances in neural information processing systems*, pp. 4055–4065, 2017.

Berner, C., Brockman, G., Chan, B., Cheung, V., Debiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., Józefowicz, R., Gray, S., Olsson, C., Pachocki, J. W., Petrov, M., de Oliveira Pinto, H. P., Raiman, J., Salimans, T., Schlatter, J., Schneider, J., Sidor, S., Sutskever, I., Tang, J., Wolski, F., and Zhang, S. Dota 2 with large scale deep reinforcement learning. *ArXiv*, abs/1912.06680, 2019.

Borsa, D., Graepel, T., and Shawe-Taylor, J. Learning shared representations in multi-task reinforcement learning. *arXiv preprint arXiv:1603.02041*, 2016.

Borsa, D., Barreto, A., Quan, J., Mankowitz, D., Munos, R., van Hasselt, H., Silver, D., and Schaul, T. Universal successor features approximators. *arXiv preprint arXiv:1812.07626*, 2018.

Co-Reyes, J. D., Liu, Y., Gupta, A., Eysenbach, B., Abbeel, P., and Levine, S. Self-consistent trajectory autoencoder: Hierarchical reinforcement learning with trajectory embeddings. In *ICML*, 2018.

Cobbe, K., Klimov, O., Hesse, C., Kim, T., and Schulman, J. Quantifying generalization in reinforcement learning. In *ICML*, 2019.

Cully, A., Clune, J., Tarapore, D., and Mouret, J.-B. Robots that can adapt like animals. *Nature*, 521:503–507, 2015.

Da Silva, B., Konidaris, G., and Barto, A. Learning parameterized skills. *arXiv preprint arXiv:1206.6398*, 2012.

Devin, C., Gupta, A., Darrell, T., Abbeel, P., and Levine, S. Learning modular neural network policies for multi-task and multi-robot transfer. *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2169–2176, 2016.

Doshi-Velez, F. and Konidaris, G. Hidden parameter markov decision processes: A semiparametric regression approach for discovering latent task parametrizations. *IJCAI : proceedings of the conference*, 2016:1432–1440, 2013.

Duan, Y., Schulman, J., Chen, X., Bartlett, P. L., Sutskever, I., and Abbeel, P. Rl^2: Fast reinforcement learning via slow reinforcement learning. *ArXiv*, abs/1611.02779, 2016.

Duan, Y., Andrychowicz, M., Stadie, B. C., Ho, J., Schneider, J., Sutskever, I., Abbeel, P., and Zaremba, W. One-shot imitation learning. In *NIPS*, 2017.

Finn, C., Abbeel, P., and Levine, S. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 1126–1135. JMLR. org, 2017.

Gupta, A., Devin, C., Liu, Y., Abbeel, P., and Levine, S. Learning invariant feature spaces to transfer skills with reinforcement learning. *arXiv preprint arXiv:1703.02949*, 2017.

Hansen, S., Dabney, W., Barreto, A., Van de Wiele, T., Warde-Farley, D., and Mnih, V. Fast task inference with variational intrinsic successor features. *arXiv preprint arXiv:1906.05030*, 2019.

Hausman, K., Springenberg, J. T., Wang, Z., Heess, N. M. O., and Riedmiller, M. A. Learning an embedding space for transferable robot skills. In *ICLR*, 2018.

He, Z., Julian, R., Heiden, E., Zhang, H., Schaal, S., Lim, J. J., Sukhatme, G., and Hausman, K. Zero-shot skill composition and simulation-to-real transfer by learning task representations. *arXiv preprint arXiv:1810.02422*, 2018.

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., and Meger, D. Deep reinforcement learning that matters. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Hessel, M., Soyer, H., Espeholt, L., Czarnecki, W., Schmitt, S., and van Hasselt, H. Multi-task deep reinforcement learning with popart. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 3796–3803, 2019.

Higgins, I., Pal, A., Rusu, A. A., Matthey, L., Burgess, C., Pritzel, A., Botvinick, M. M., Blundell, C., and Lerchner, A. Darla: Improving zero-shot transfer in reinforcement learning. In *ICML*, 2017.

Houthooft, R., Chen, R. Y., Isola, P., Stadie, B. C., Wolski, F., Ho, J., and Abbeel, P. Evolved policy gradients. *ArXiv*, abs/1802.04821, 2018.

Humplik, J., Galashov, A., Hasenclever, L., Ortega, P. A., Teh, Y. W., and Heess, N. Meta reinforcement learning as task inference. *arXiv preprint arXiv:1905.06424*, 2019.

Jaderberg, M., Czarnecki, W. M., Dunning, I., Marris, L., Lever, G., Castaneda, A. G., Beattie, C., Rabinowitz, N. C., Morcos, A. S., Ruderman, A., et al. Human-level performance in 3d multiplayer games with population-based reinforcement learning. *Science*, 364(6443):859–865, 2019.

Killian, T. W., Konidaris, G., and Doshi-Velez, F. Robust and efficient transfer learning with hidden parameter markov decision processes. *Advances in neural information processing systems*, 30:6250–6261, 2017.

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

Kingma, D. P. and Welling, M. Auto-encoding variational bayes. *CoRR*, abs/1312.6114, 2013.

Lazaric, A. Transfer in reinforcement learning: a framework and a survey. In *Reinforcement Learning*, pp. 143–173. Springer, 2012.

Madjiheurem, S. and Toni, L. State2vec: Off-policy successor features approximators. *arXiv preprint arXiv:1910.10277*, 2019.

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

Nagabandi, A., Clavera, I., Liu, S., Fearing, R. S., Abbeel, P., Levine, S., and Finn, C. Learning to adapt in dynamic, real-world environments through meta-reinforcement learning. *arXiv preprint arXiv:1803.11347*, 2018.

Oh, J., Singh, S., Lee, H., and Kohli, P. Zero-shot task generalization with multi-task deep reinforcement learning. *arXiv preprint arXiv:1706.05064*, 2017.

Parisotto, E., Ba, J. L., and Salakhutdinov, R. Actor-mimic: Deep multitask and transfer reinforcement learning. *arXiv preprint arXiv:1511.06342*, 2015.

Paul, S., Osborne, M. A., and Whiteson, S. Fingerprint policy optimisation for robust reinforcement learning. In *ICML*, 2018.

Perez, C. F., Such, F. P., and Karaletsos, T. Efficient transfer learning and online adaptation with latent variable models for continuous control. *ArXiv*, abs/1812.03399, 2018.

Petangoda, J. C., Pascual-Diaz, S., Adam, V., Vrancx, P., and Grau-Moya, J. Disentangled skill embeddings for reinforcement learning. *ArXiv*, abs/1906.09223, 2019.

Pinto, L., Davidson, J., Sukthankar, R., and Gupta, A. Robust adversarial reinforcement learning. In *ICML*, 2017.

Precup, D., Sutton, R. S., and Dasgupta, S. Off-policy temporal-difference learning with function approximation. In *ICML*, pp. 417–424, 2001.

Raileanu, R. and Rocktäschel, T. Ride: Rewarding impact-driven exploration for procedurally-generated environments. *ArXiv*, abs/2002.12292, 2020.

Rajeswaran, A., Lowrey, K., Todorov, E. V., and Kakade, S. M. Towards generalization and simplicity in continuous control. In *Advances in Neural Information Processing Systems*, pp. 6550–6561, 2017.

Rakelly, K., Zhou, A., Finn, C., Levine, S., and Quillen, D. Efficient off-policy meta-reinforcement learning via probabilistic context variables. In *International conference on machine learning*, pp. 5331–5340, 2019.

Sæmundsson, S., Hofmann, K., and Deisenroth, M. P. Meta reinforcement learning with latent variable gaussian processes. *arXiv preprint arXiv:1803.07551*, 2018.

Sahni, H., Kumar, S., Tejani, F., and Isbell, C. Learning to compose skills. *arXiv preprint arXiv:1711.11289*, 2017.

Schaul, T., Horgan, D., Gregor, K., and Silver, D. Universal value function approximators. In *International conference on machine learning*, pp. 1312–1320, 2015.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

Siriwardhana, S., Weerasakera, R., Matthies, D. J., and Nanayakkara, S. Vusfa: Variational universal successor features approximator to improve transfer drl for target driven visual navigation. *arXiv preprint arXiv:1908.06376*, 2019.

Song, X., Jiang, Y., Tu, S., Du, Y., and Neyshabur, B. Observational overfitting in reinforcement learning. *ArXiv*, abs/1912.02975, 2020.

Sutton, R. S., Modayil, J., Delp, M., Degris, T., Pilarski, P. M., White, A., and Precup, D. Horde: a scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In *AAMAS*, 2011.

Taylor, M. E. and Stone, P. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(Jul):1633–1685, 2009.

Teh, Y., Bapst, V., Czarnecki, W. M., Quan, J., Kirkpatrick, J., Hadsell, R., Heess, N., and Pascanu, R. Distral: Robust multitask reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 4496–4506, 2017.

Todorov, E., Erez, T., and Tassa, Y. Mujoco: A physics engine for model-based control. *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033, 2012.

van der Maaten, L. and Hinton, G. E. Visualizing data using t-sne. 2008.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008, 2017.

Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575 (7782):350–354, 2019.

Wang, J. X., Kurth-Nelson, Z., Soyer, H., Leibo, J. Z., Tirumala, D., Munos, R., Blundell, C., Kumaran, D., and Botvinick, M. M. Learning to reinforcement learn. *ArXiv*, abs/1611.05763, 2016.

Wang, Z., Merel, J., Reed, S. E., de Freitas, N., Wayne, G., and Heess, N. M. O. Robust imitation of diverse behaviors. In *NIPS*, 2017.

White, A., Modayil, J., and Sutton, R. S. Scaling lifelong off-policy learning. In *2012 IEEE International Conference on Development and Learning and Epigenetic Robotics (ICDL)*, pp. 1–6. IEEE, 2012.

Whiteson, S., Tanner, B., Taylor, M. E., and Stone, P. Protecting against evaluation overfitting in empirical reinforcement learning. *2011 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*, pp. 120–127, 2011.

Xu, Z., van Hasselt, H., and Silver, D. Meta-gradient reinforcement learning. In *NeurIPS*, 2018.

Yang, J., Petersen, B., Zha, H., and Faissol, D. Single episode policy transfer in reinforcement learning. *arXiv preprint arXiv:1910.07719*, 2019.

Yao, J., Killian, T. W., Konidaris, G., and Doshi-Velez, F. Direct policy transfer via hidden parameter markov decision processes. 2018.

Zhang, A., Ballas, N., and Pineau, J. A dissection of overfitting and generalization in continuous reinforcement learning. *arXiv preprint arXiv:1806.07937*, 2018a.

Zhang, A., Satija, H., and Pineau, J. Decoupling dynamics and reward for transfer learning. *ArXiv*, abs/1804.10689, 2018b.

Zhang, C., Vinyals, O., Munos, R., and Bengio, S. A study on overfitting in deep reinforcement learning. *arXiv preprint arXiv:1804.06893*, 2018c.

Zhang, J., Springenberg, J. T., Boedecker, J., and Burgard, W. Deep reinforcement learning with successor features for navigation across similar environments. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2371–2378. IEEE, 2017.

Zintgraf, L. M., Shiarlis, K., Kurin, V., Hofmann, K., and Whiteson, S. Fast context adaptation via meta-learning. In *ICML*, 2018.