# Fast Adaptation to New Environments via Policy-Dynamics Value Functions

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

[1]New York University, [2]Facebook AI Research

## Introduction

**Motivation:** Standard reinforcement learning (RL) algorithms assume fixed environment dynamics and require a many interactions to adapt to new scenarios.

**Problem:** How can RL agents rapidly **adapt** to changes in their environment's **dynamics**?

**Key Idea: Learn a value function that generalizes not only across states, but also across policies and dynamics.**

**Intuition:** This **policy-dynamics value function** can discover useful patterns among different dynamics, behaviors, and returns, which cannot be captured with a single policy and value function. This enables faster adaptation to new environments.

## Problem Setting

Consider a **family of MDPs** defined by $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma)$.

Each environment has a **different transition function** $\mathcal{T}_d(s'|s,a) \in \mathcal{T}$ determined by a hidden variable $d \sim D$.

**Train** on a family of MDPs with **different but related dynamics**: $d \sim D_{train}$.

**Test** on environments with **new dynamics** from the same family: $d \sim D_{test}$ with $D_{test} \cup D_{train} = D$ and $D_{test} \cap D_{train} = \emptyset$.

## Policy-Dynamics Value Function

A conventional value function $V : \mathcal{S} \rightarrow \mathcal{R}$ is defined as the expected return from state $s$ when acting with a fixed policy $\pi$:

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

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

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

$W$ explicitly estimates the **expected return for a family of policies and environments** with different dynamics.

**Note:** The PD-VF can **predict how each policy in our space will perform on each environment in our family of MDPs**. Given a new environment, we can search in this entire space of policies and select the one that works best in that particular setting rather than trying to learn a policy that performs well in a wide range of dynamics, which might be too difficult.

## Reinforcement Learning Phase

Train an ensemble of policies on each of the training environments using standard RL algorithms.

Use these policies to collect trajectories from all the training environments.

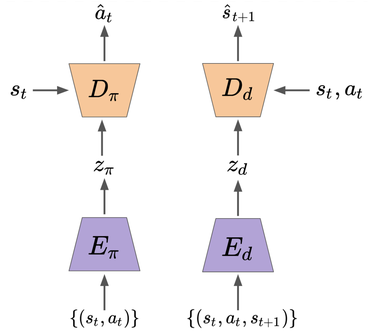
This results in **experience from a diverse set of policies (both optimal and suboptimal) across different environments**.

## Self-Supervised Learning Phase

The collected trajectories are used to **learn policy and dynamics embeddings**.

The two embeddings are learned using a pair of autoencoders with **Transformer encoders** that take as input sets of transitions (generated in the previous phase).

The use of Transformers without positional encoding exploits the **Markov property** of the interactions.




**Policy embedding**:

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

**Dynamics embedding**:

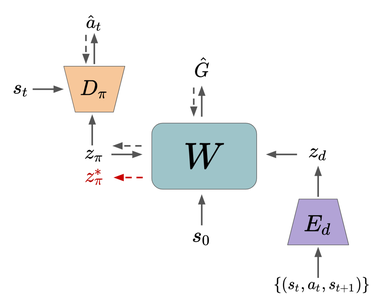
$$z_d = E_d(\{(s_t, a_t, s_{t+1})\};\ \theta_d)$$
$$\hat{s}_{t+1} = D_d(s_t, a_t, z_d;\ \phi_d)$$

## Supervised Learning Phase

The PD-VF $W$ is trained with supervision to predict the expected return $G$ for an initial state, a policy embedding, and a dynamics embedding.

**At test time, we find $z_\pi^\star$ that maximizes $G$, which is decoded to an actual policy via $D_\pi$.**

$W$ is parameterized as a quadratic in $z_\pi$ for fast optimization at test time.




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

## Evaluation Phase

**The dynamics embedding is inferred using only a few interactions (< 5) with a new environment.**

Then, a policy is selected by finding the policy embedding that maximizes the PD-VF.

The selected policy is used to act in the environment until the episode ends.

## Experiments

**PPOenv** is an upper bound that trains a PPO policy directly on the test environment.

**MAML** is the meta-learning algorithm from Finn et al. 2017.

**RL$^2$** is the meta-learning algorithm from Wang et al. 2016.

**PPOdyn** trains a single policy conditioned on the dynamics embedding.

**PPOall** trains a single PPO policy on all the training environments.

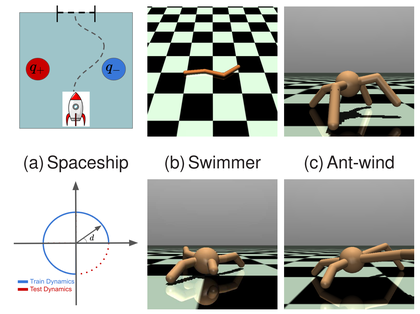



Figure 1: (a) - (c): continuous distribution of the dynamics given by (d). (e), (f): discrete distribution of the dynamics.

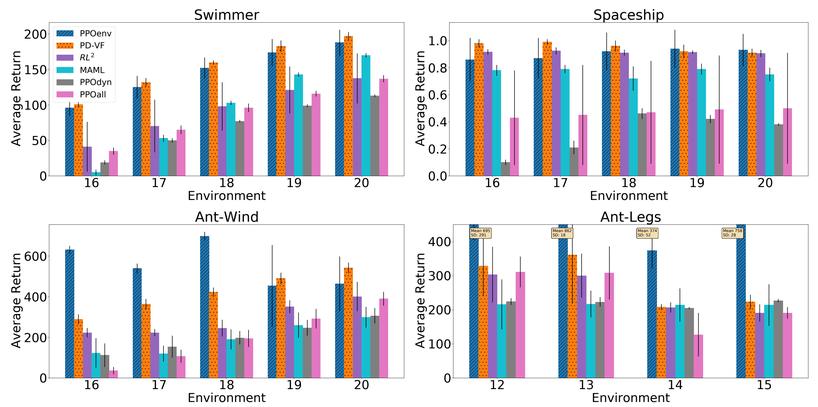



Figure 2: PD-VF outperforms competitive methods on test environments, demonstrating that it can quickly adapt to new dynamics.

## Discussion

**In contrast to meta-learning techniques, PD-VF does not require parameter updates, long rollouts, or dense rewards to adapt.**

We proposed a **general framework which could be extended** to task variation, multi-agent settings, continual learning, or inclusion of prior knowledge and other constraints during the optimization process.