Neural Dynamic Policies for End-to-End Sensorimotor Learning

Shikhar Bahl [1] Mustafa Mukadam [2] Abhinav Gupta [1] Deepak Pathak [1]

Abstract

The current dominant paradigm in sensorimotor control, whether imitation or reinforcement learning, is to train policies directly in raw action spaces. This forces the agent to make decisions at each point in training, and limits scalability to complex tasks. In contrast, classical robotics research has exploited dynamical systems as policy representations to learn behaviors via demonstrations. These techniques, however, lack the flexibility provided by deep learning and have remained under-explored in such settings. In this work, we begin to close this gap and embed dynamics structure into deep neural network-based policies by reparameterizing action spaces with differential equations. We propose Neural Dynamic Policies (NDPs) that make predictions in trajectory distribution space as opposed to raw control spaces. The embedded structure allows us to perform end-to-end policy learning in both reinforcement and imitation learning setups. We show that NDPs achieve state-of-the-art performance on many robotic control tasks in simulation, as well as on digit writing using demonstrations.

1 Introduction

Consider an embodied agent tasked with throwing a ball into a bin. Not only does the agent need to decide where and when to release the ball, but also reason about the whole trajectory that it should take such that the ball is imparted with the correct momentum to reach the bin. This form of reasoning is necessary to perform many such everyday tasks. Common methods in deep learning for robotics tackle this problem either via imitation or reinforcement. However, in most cases, the agent's policy is trained in raw action spaces like torque, joint angle, or end-effector position, which forces the agent to make decisions at each time step of the trajectory instead of making decisions in the trajectories space itself (see Figure 1 (left). But then how do we reason about trajectories as actions?

[1]Carnegie Mellon University [2]Facebook AI Research. Correspondence to: Shikhar Bahl <sbahl2@cs.cmu.edu>.

Proceedings of the 37th International Conference on Machine Learning, Vienna, Austria, PMLR 119, 2020. Copyright 2020 by the author(s).

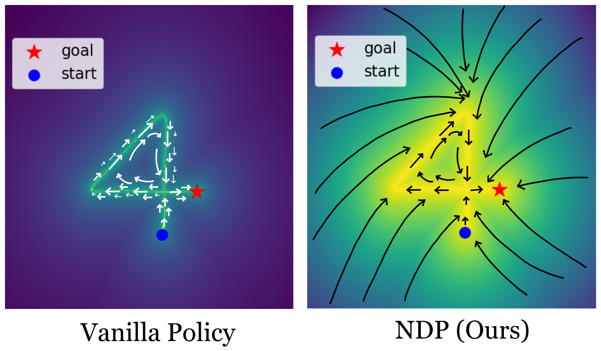


Figure 1: Vector field induced by NDPs. The goal is to draw a planar digit (4). The dynamical structure in NDP induces a smooth vector field in trajectory space. In contrast, a vanilla policy has to reason individually in different parts.

A good trajectory parameterization is one that is able to capture a large set of agent's behaviors or motions while being physically plausible. In fact, a similar question is also faced by physicists while modeling physical phenomena in nature. Several dynamical systems, in science ranging from motion of planets to that of pendulums, are described by differential equations of the form $\ddot{y} = m^{-1}f(y, \dot{y})$, where y is the generalized coordinate, $\dot{y}$ and $\ddot{y}$ are time derivatives, m is mass, and f is force.

Can a similar parameterization be used to describe the behavior of a robotic agent? Indeed, classical robotics has leveraged this connection to represent task specific robot behaviors for many years. In particular, dynamic movement primitives (DMPs) (Schaal, 2006; Ijspeert et al., 2013; 2002; 2003) have been one prominent approaches in this area. Despite their successes, they haven't been explored much beyond behavior cloning paradigms. This is partly because these methods tend to be sensitive to parameter tuning and are not as flexible or as generalizable as current end-to-end deep learning based approaches.

In this work, we propose to bridge this gap by embedding dynamics structure into deep neural network-based policies such that the agent can directly learn in

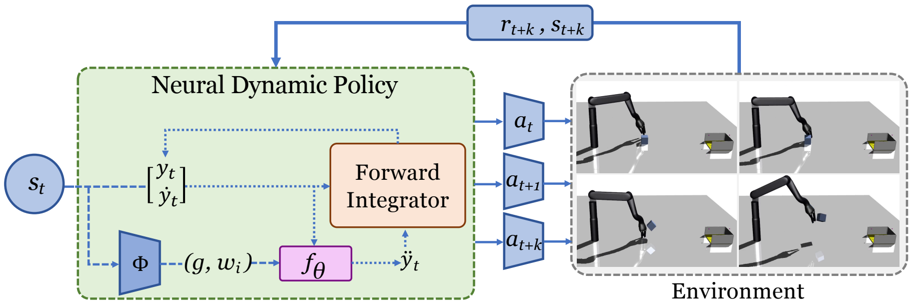


Figure 2: Given an observation from the environment, s_t, our Neural Dynamic Policy generates parameters w (weights of basis functions) and g (goal for the robot) for a forcing function f_θ. An open loop controller then uses this function to outputs a set of actions for the robot that are executed in the environment, collecting future states and rewards for training.

the space of physically plausible trajectory distributions (see Figure 1(right)). Our key insight is to reparameterize the action space in a deep policy network with nonlinear differential equations corresponding to a dynamical system and train it end-to-end using reinforcement learning (RL) or imitation learning.

We propose Neural Dynamic Policies (NDPs) to allow embedding desired dynamical structure as a layer in deep networks. The parameters of the dynamical system are then predicted as outputs of the preceding layers in the architecture conditioned on the input. **The 'deep' part of the policy then only needs to reason in the lower-dimensional space of building a dynamical system that then lets the overall policy easily reason in the space of trajectories.** In this paper, we employ the aforementioned DMPs as the structure for the dynamical system and show its differentiability, although they only serve as a design choice for our early work here and can easily be swapped for a different differentiable dynamical structure. We expand on alternatives later in the discussion section.

We evaluate NDPs for both imitation as well as reinforcement learning. We show that NDPs can utilize high-dimensional inputs like images via imitation learning to accomplish a digit writing task. We show how to train NDPs with RL across several continuous control tasks in simulation. NDPs exhibit state-of-the-art performance in several tasks when compared to multiple baselines.

2 Neural Dynamic Policies (NDPs)

2.1 Modeling Trajectories with Dynamical Systems

It is common in classical robotics to represent movement behaviors with dynamical systems. Specifically, consider the second order differential equation structure imposed by Dynamic Movement Primitives (Ijspeert et al., 2013; Schaal, 2006). Let the state of the robot be y, velocity $\dot{y}$ and acceleration $\ddot{y}$ (either in joint-angle or end-effector space). Given a desired goal state g, the behavior is represented as:

$$\ddot{y} = \alpha(\beta(g-y) - \dot{y}) + f(x), \quad \dot{x} = -a_x \tag{1}$$

where α, β are global parameters that allow critical damping. f is a non-linear forcing function which captures the shape of trajectory and operates over x which serves to replace time dependency across trajectories, making the system time invariant, and evolves via a first-order linear system. f is usually a design choice. We use a sum of weighted Gaussian radial basis functions (Ijspeert et al., 2013) shown below:

$$f(x,g) = \frac{\sum \psi_i w_i}{\sum \psi_i} x(g - y_0), \quad \psi_i = e^{(-h_i(x-c_i)^2)} \tag{2}$$

where i indexes over n, the number of basis functions. Coefficients $c_i = e^{\frac{-i\alpha_x}{n}}$, $h_i = \frac{n}{c_i}$ are horizontal shift and width of each basis function. This set of nonlinear differential equations induces a smooth trajectory distribution that acts as an attractor towards a goal position (see Figure 1). We now combine this dynamical structure with deep neural network based policies in an end-to-end differentiable manner.

2.2 NN Layer Parameterized by a Dynamical System

We embed a dynamical system described by the DMP equation (1) in a Neural Network. There are two key parameters that define the behavior of the dynamical system from Section 2.1: basis function weights $w = \{w_1, \ldots, w_i, \ldots, w_n\}$ and goal g. NDPs employ a neural network Φ which takes an unstructured input s (not to be confused with robot state y) and predicts the parameters w, g of the dynamical system. These predicted w, g are then used to solve the second order differential equation (1) to obtain system states $\{y, \dot{y}, \ddot{y}\}$. Depending on the difference between the robot's coordinate system for y and desired action a, we may need an inverse controller $\Omega(.)$ to convert y to a, i.e., $a = \Omega(y, \dot{y}, \ddot{y})$. For instance, if y is in joint angle space and a is torque control, we use robot's inverse dynamics controller as $\Omega(.)$.

As summarized in Figure 2, NDPs are defined as $\pi(a|s;\theta) \triangleq \Omega\big(\mathtt{DE}\big(\Phi(s;\theta)\big)\big)$ where $\mathtt{DE}(w,g) \rightarrow \{y, \dot{y}, \ddot{y}\}$ denotes solution of the differential equation 1. The forward pass of $\pi(a|s)$ involves solving the dynamical system and backpropagation requires the system to be differentiable. We now show how we differentiate through the dynamical system to train the parameters θ of NDPs.

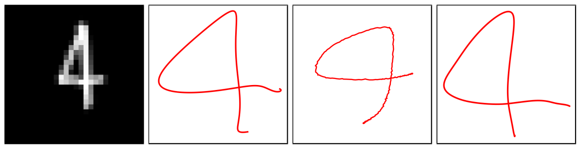


Figure 3: Imitation (supervised) learning results on test images of digit writing task. Given an input image (left), the output action is the end-effector position. We find that trajectories output by NDPs (ours) are dynamically smooth and more accurate than baselines

2.3 Differentiating through a Dynamical System

To train NDPs, estimated policy gradients must flow from a, through the parameters of the dynamical system w and g, to the network $\Phi(s;\theta)$. At any time t, given the previous state of robot y_{t-1} and velocity $\dot{y}_{t-1}$ the output of the DMP in Equation (1) is given by the acceleration

$$\ddot{y}_t = \alpha(\beta(g - y_{t-1}) - \dot{y}_{t-1} + f(x_t, g) \tag{3}$$

Through Euler integration, we can find the next velocity and position after a small time interval dt

$$\dot{y}_t = \dot{y}_{t-1} + \ddot{y}_{t-1}dt, \quad y_t = y_{t-1} + \dot{y}_{t-1}dt \tag{4}$$

We can use these relationships as well as Equations (2)-(4) to compute gradients of the trajectory from the DMP with respect to w and g. In practice, we can use all m robot states y to obtain actions, or sub-sample $k \in 1, m$ actions. This allows robot operation at much higher frequencies (.5-5KHz) than the environment (usually 100Hz).

2.4 NDPs for Imitation and RL

Training NDPs in imitation learning setup is rather straightforward. Given a sequence of input $\{s, s', \ldots\}$, NDP's $\pi(s;\theta)$ outputs a sequence of actions $a, a' \ldots$. In our experiments, s is the high dimensional image input. Let the demonstrated action sequence be τ_{target}, we just take an L2 loss between the predicted sequence and ground truth.

We now show how an NDP can be used as a policy, π in the RL setting. As discussed in Section 2.3, NDP samples k actions for the agent to execute in the environment given input observation s. One could use any underlying RL algorithm to optimize the policy. In this paper, we use Proximal Policy Optimization (PPO) (Schulman et al., 2017) and treat a independently when computing the policy gradient for each step of the NDP rollout and backprop via reinforce objective. There are two choices for value function critic $V^\pi(s)$: either predict a common value function for all the actions in k-step rollout or predict different critic values for each step in the rollout. We found that the latter works better in practice. We call this *multi-action critic architecture* and predict k different estimates of value using k-heads on top of critic network. Later, in the experiments we perform ablations over the choice of k. To further create a strong baseline comparison, as we discuss in Section 3.2, we also design and compare against a variant of PPO to predict multiple actions using our multi-action critic architecture.

Method	Train	Test (held-out)
CNN	10.42 ± 5.26	10.59 ± 4.63
CNN-DMP (Pahic et al., 2018)	9.44 ± 4.59	8.46 ± 8.45
NDP (ours)	**0.70 ± 0.36**	**0.74 ± 0.34**

Table 1: Imitation learning on digit writing task. We report mean loss across digit classes. Input is the image of digit and outputs are end-effector positions. NDP significantly outperforms baselines.

Inference with NDPs: In the case of inference, NDP uses policy π once every k environment steps, hence takes $\lceil \frac{H}{k} \rceil$ forward passes. In real world settings, due to large overhead, reducing inference time can help decrease overall time costs. Additionally, deployed systems are not as powerful as those used to train RL methods on simulators, so inference costs end up accumulating. Furthermore, as discussed in Section 2.3, the rollout length of NDP can be more densely sampled at test-time than at training allowing the robot to produce smooth and dynamically stable motions. Compared to about 100Hz frequency of the simulation, in practice our method can make decisions much faster at about .5-5KHz.

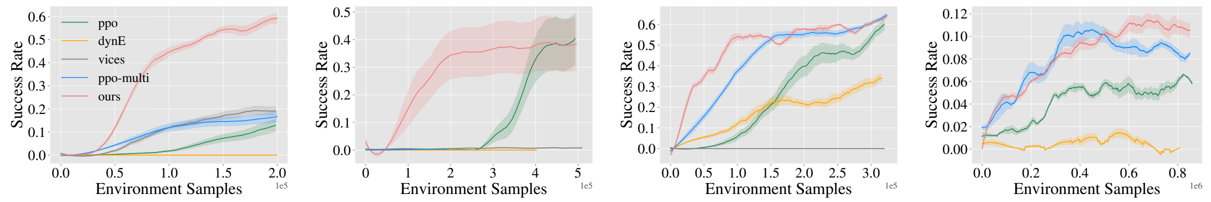
3 Experimental Evaluation

3.1 Imitation (Supervised) Learning

To evaluate NDPs in imitation learning setup, we perform the task of learning to write digits using a 2D end-effector. The goal is to train a planar robot to trace the digit. The output action is the robot's end-effector position, and supervision is obtained via ground truth trajectories. We compare NDPs to a regular behavior cloning policy parameterized by a CNN and the prior approach which maps image to DMP parameters (Pahic et al., 2018) (dubbed, CNN-DMP). CNN-DMP (Pahic et al., 2018) trains a single DMP for the whole trajectory and requires supervised demonstrations, in contrast to NDPs, which can generate multiple DMPs across time. We present qualitative results for this setup in Figure 3 as well as quantitative results in Table 1, which reports trajectory reconstruction loss. NDP outperforms both CNN and CNN-DMP (Pahic et al., 2018) drastically. NDP also produces much higher quality and smoother reconstructions as shown in Figure 3. This shows how our method can efficiently capture dynamic motions.

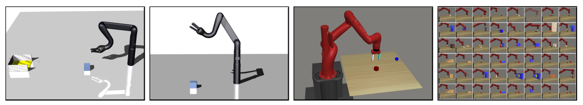
3.2 Reinforcement Learning

We evaluate our approach on dynamic environments include **Throwing** and **Picking** (Ghosh et al., 2017). We also evaluate on quasi-static tasks such **Pushing**, from the Meta-World (Yu et al., 2019) task suite, as well as a setup that requires



(a) Throwing (b) Picking (c) Pushing (d) MT50 (Joint 50 Tasks)

Figure 4: Evaluation of RL setup for continuous control tasks. Y axis is success rate and X axis denotes number of environment samples. Our method (pink) uses k = 5 steps per DMP. We compare against PPO (Schulman et al., 2017) (green), a multi-action version of PPO (blue), which also outputs 5 actions at a time, VICES (Martin-Martin et al., 2019) (gray) and DYN-E (Whitney et al., 2019) (yellow).



(a) Throwing (b) Picking (c) Pushing (d) 50 Tasks

Figure 5: Environment snapshot for different tasks used in experiments. (a,b) are adapted from (Ghosh et al., 2017) while (c-f) tasks are adapted from (Yu et al., 2019)

learning all 50 tasks (**MT50**) jointly (see Figure 5). In contrast to imitation learning where dynamic rollout length of NDP is high ($k = 300$), we set $k = 5$ in RL because the reward becomes too sparse if k is very large. Figure 4 shows the results from RL experiments on the tasks described above. We compare NDP PPO (Schulman et al., 2017). NDP is able to operate the robot world at a higher frequency, while observing the environments once every k steps while acting at every step. As described in Section 2.4, we compare to PPO-multi, which predicts multiple actions using our multi-action critic architecture. All methods are compared in terms of environment samples observed. We also compare to Variable Impedance Control in End-Effector Space (VICES) (Martin-Martin et al., 2019) and Dynamics-Aware Embeddings (DYN-E) (Whitney et al., 2019). VICES learns to output parameters of an Impedance (or PD) controller directly. DYN-E, using a forward prediction model, learns a lower-dimensional action embedding.

Experimental results show that our method, NDP, outperforms SOTA methods such as PPO. Our method sees gains in both efficiency and performance in most tasks. The final task of training jointly across 50 Meta-World tasks is too hard for all methods. Nevertheless, NDP attains a higher absolute performance but doesn't show efficiency gains. PPO-multi, performs well in some cases (Pushing) but is inconsistent in its performance, failing completely at times (Picking). Our method also outperforms VICES (Martin-Martin et al., 2019) and Dyn-E (Whitney et al., 2019). VICES is slightly successful throwing, but suffers in more complex settings due to a large action space dimensionality. DYN-E, on the other hand, performs well on tasks such as Pushing, which has simpler contacts, but fails to scale to complex environments. Through these experiments, we show the diversity and versatility of NDPs. It is able to reason in a space of physically meaningful trajectories, but it does not lose the advantages and flexibility other policy setups have.

4 Related Work

Various methods in the field of control and robotics have employed dynamical systems to create more structured learning. Most of these (Rana et al., 2020; Ravichandar et al., 2017) use dynamical systems to model demonstrations, but do not tackle generalization or go beyond imitation. Previous works have proposed and used DMPs (Schaal, 2006; Ijspeert et al., 2013; Kober and Peters, 2009) for robot control, but have mostly focused on learning from demonstration. Recently, DMPs have been used in the context of deep learning, for example by Pahic et al. (2018), however they are only used as single DMPs representing trajectories, for imitation. Work has been done in representing dynamical systems (Conkey and Hermans, 2019; Ude et al., 2010; Calinon et al., 2010; Cheng et al., 2020; Huang et al., 2019), however these tackle problems that require domain knowledge. Many works have made use of action parameterization, such incorporating DMPs as options into RL (Daniel et al., 2016; Parisi et al., 2015), using controller parameters as actions ((Martin-Martin et al., 2019)) or learning an action embedding (Whitney et al., 2019). However, these methods are either not flexible to multiple tasks or do not reason at a trajectory level, and thus do not scale.

5 Discussion

Our method attempts to bridge the gap between classical robotics and control and recent approaches in deep learning and deep RL. We propose a novel re-parameterization of action spaces via a Neural Dynamic Policies, a set of policies which impose the structure of a dynamical system on action spaces. The use of DMPs in this work was a design choice within our architecture which allows for any form of dynamical structure that is differentiable. One can setup a dynamical structure such that it explicitly models and learns various aspects of the dynamical system, such as the metric, potential, and damping. While this brings advantages in better representation it also brings challenges in learning. We leave these directions for future work to explore.

References

Sylvain Calinon, Irene Sardellitti, and Darwin G. Caldwell. Learning-based control strategy for safe human-robot interaction exploiting task and robot redundancies. *IROS*, 2010. 4

Ching-An Cheng, Mustafa Mukadam, Jan Issac, Stan Birchfield, Dieter Fox, Byron Boots, and Nathan Ratliff. Rmpflow: A computational graph for automatic motion policy generation. *Algorithmic Foundations of Robotics XIII*, 2020. 4

Adam Conkey and Tucker Hermans. Active learning of probabilistic movement primitives. *2019 IEEE-RAS 19th International Conference on Humanoid Robots (Humanoids)*, 2019. 4

Christian Daniel, Gerhard Neumann, Oliver Kroemer, and Jan Peters. Hierarchical relative entropy policy search. *Journal of Machine Learning Research*, 2016. 4

Dibya Ghosh, Avi Singh, Aravind Rajeswaran, Vikash Kumar, and Sergey Levine. Divide-and-conquer reinforcement learning. *arXiv preprint arXiv:1711.09874*, 2017. 3, 4

Yanlong Huang, Leonel Rozo, João Silvério, and Darwin G Caldwell. Kernelized movement primitives. *The International Journal of Robotics Research*, 2019. 4

Auke J Ijspeert, Jun Nakanishi, and Stefan Schaal. Learning attractor landscapes for learning motor primitives. In *NeurIPS*, 2003. 1

Auke Jan Ijspeert, Jun Nakanishi, and Stefan Schaal. Movement imitation with nonlinear dynamical systems in humanoid robots. In *ICRA*. IEEE, 2002. 1

Auke Jan Ijspeert, Jun Nakanishi, Heiko Hoffmann, Peter Pastor, and Stefan Schaal. Dynamical movement primitives: Learning attractor models for motor behaviors. *Neural Computation*, 2013. 1, 2, 4

Jens Kober and Jan Peters. Learning motor primitives for robotics. In *ICRA*, 2009. 4

Roberto Martin-Martin, Michelle A. Lee, Rachel Gardner, Silvio Savarese, Jeannette Bohg, and Animesh Garg. Variable impedance control in end-effector space: An action space for reinforcement learning in contact-rich tasks. *IROS*, 2019. 4

Rok Pahic, Andrej Gams, Aleš Ude, and Jun Morimoto. Deep encoder-decoder networks for mapping raw images to dynamic movement primitives. *ICRA*, 2018. 3, 4

Simone Parisi, Hany Abdulsamad, Alexandros Paraschos, Christian Daniel, and Jan Peters. Reinforcement learning vs human programming in tetherball robot games. In *IROS*, 2015. 4

Muhammad Asif Rana, Anqi Li, Dieter Fox, Byron Boots, Fabio Ramos, and Nathan Ratliff. Euclideanizing flows: Diffeomorphic reduction for learning stable dynamical systems. *arXiv preprint arXiv:2005.13143*, 2020. 4

Harish Ravichandar, Iman Salehi, and Ashwin Dani. Learning partially contracting dynamical systems from demonstrations. In *CoRL*, 2017. 4

Stefan Schaal. Dynamic movement primitives-a framework for motor control in humans and humanoid robotics. In *Adaptive motion of animals and machines*. Springer, 2006. 1, 2, 4

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv:1707.06347*, 2017. 3, 4

Aleš Ude, Andrej Gams, Tamim Asfour, and Jun Morimoto. Task-specific generalization of discrete and periodic dynamic movement primitives. *Transactions on Robotics*, 2010. 4

William Whitney, Rajat Agarwal, Kyunghyun Cho, and Abhinav Gupta. Dynamics-aware embeddings. *arXiv preprint arXiv:1908.09357*, 2019. 4

Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. *arXiv preprint arXiv:1910.10897*, 2019. 3, 4