# Deep Off-Policy Iterative Learning Control

**Swaminathan Gurumurthy*** SGURUMUR@ANDREW.CMU.EDU
*Carnegie Mellon University*

**J. Zico Kolter** ZKOLTER@CS.CMU.EDU
*Carnegie Mellon University*
*Bosch Center for AI*

**Zachary Manchester** ZMANCHES@ANDREW.CMU.EDU
*Carnegie Mellon University*

**Editors:** N. Matni, M. Morari, G. J. Pappas

## Abstract

Reinforcement learning has emerged as a powerful paradigm to learn control policies while making few assumptions about the environment. However, this lack of assumptions in popular RL algorithms also leads to sample inefficiency. Furthermore, we often have access to a simulator that can provide approximate gradients for the rewards and dynamics of the environment. Iterative learning control (ILC) approaches have been shown to be very efficient at learning policies by using approximate simulator gradients to speed up optimization. However, they lack the generality of reinforcement learning approaches. In this paper, we take inspiration from ILC and propose an update equation for the value-function gradients (computed using the dynamics Jacobians and reward gradient obtained from an approximate simulator) to speed up value-function and policy optimization. We add this update to an off-the-shelf off-policy reinforcement learning algorithm and demonstrate that using the value-gradient update leads to a significant improvement in sample efficiency (and sometimes better performance) both when learning from scratch in a new environment and while fine-tuning a pre-trained policy in a new environment. Moreover, we observe that policies pretrained in the simulator using the simulator jacobians obtain better zero-shot transfer performance and adapt much faster in a new environment.

**Keywords:** Reinforcement Learning; Iterative Learning Control; Differentiable Simulators

## 1. Introduction

Deep reinforcement learning has been applied successfully in domains ranging from games (Silver et al., 2017; Berner et al., 2019) to real-world robotics control tasks (Hwangbo et al., 2019; Xu et al., 2019; Chen et al., 2022; Xie et al., 2018). However, widespread adoption in real-world domains is hampered by sample inefficiency, as even simple tasks may need hundreds of thousands to millions of samples. One contributing factor to the observed sample inefficiency is the use of zeroth-order gradient estimates for the policy and value function updates, which frequently arise from the presupposition that the training environments are not differentiable.

However, we often have access to an approximate simulator of the environment that can provide approximate jacobians and gradients of the dynamics and reward function respectively (either analytically or through finite differences). In fact, another set of approaches to online adaptation, based on iterative learning control (ILC), have been shown to be very sample-efficient by utilizing these approximate jacobians to speed up policy optimization, albeit in very limited settings, involving

* Code available at https://github.com/RoboticExplorationLab/Deep-ILC

repetitive tasks and systems with smooth dynamics (Moore, 1993; Arimoto et al., 1984; Schoellig et al., 2012; Mueller et al., 2012).

In this work, we aim to build on the generality of reinforcement learning based approaches while utilizing approximate jacobians from the simulator to speed up policy learning. Specifically, we propose a simple update to the gradient of the value function by utilizing the simulator gradients, which we call the value-gradient update. The update boils down to simply minimizing the residual of the gradient of the Bellman error and can be added to any actor-critic reinforcement learning algorithm (even off-policy algorithms!). We show that the value gradient update is in fact equivalent to the multiplier (corresponding to the dynamics constraint) updates performed in optimal control approaches such as (i)LQR (Kalman et al., 1960; Mayne, 1973). We further show that, the update can be easily modified to accommodate approximate jacobians from a simulator by simply substituting the approximate jacobians evaluated along the rollouts (collected from the real environment) in the value gradient update, just like ILC.

This update has several advantages: 1) The ability to use approximate jacobians makes it useful even when we do not know the accurate model of the system; 2) By explicitly supervising the value function gradients, we obtain more reliable value function gradients thereby speeding up policy optimization; 3) It can be used to perform off-policy updates (especially important considering we want to use past experiences for better sample efficiency) on the value function and policy; and 4) It can handle non-smooth and discontinuous dynamics/losses by simply skipping or clipping the gradient updates on transitions with such discontinuities.

The contributions of this work are as follows:

- We introduce the value gradient update, a general update that can be used with any value function based RL algorithm given an (approximate) simulator. Unlike previous work, its compatibility with SOTA off-policy algorithms and its ability to accommodate approximate simulator jacobians makes it particularly appealing for applicability in real systems.
- We add the update to SAC, a popular off-policy RL algorithm, and demonstrate that adding the value gradient update leads to up to 2-3x sample efficiency (and sometimes better performance) across six tasks. In all the tasks, we assume access to a 'nominal' model of the system (often a simplified model with inaccurate parameters) and a more complicated, realistic model of the system as a proxy for the real world. Interestingly, we also observe that policies/value functions pretrained using the value-gradient update in the nominal model show better zero-shot performance and adapt faster in the real system compared to the RL baseline, indicating that using the simulator jacobians even for pre-training leads to more robust policies.

## 2. Related Work

**Using differentiable simulators for policy optimization** Trajectory Optimization (Bertsekas, 2012; Betts, 2001; Mayne, 1973) approaches have traditionally used differentiable models for optimizing over a sequence of states and actions. However, most learning based approaches have relied on reinforcement learning (Sutton and Barto, 2018; Kober et al., 2013; Bertsekas, 2019) (i.e not assuming differentiability of the simulator) for optimizing policies despite the physics being analytically differentiable. Multiple works recently (Metz et al., 2021; Suh et al., 2022) have pointed out issues with using first order estimates of the gradient for policy optimization. (Metz et al., 2021; Parmas et al., 2018) shows that simulator gradients can often be chaotic and differentiating through multiple simulation steps can result in vanishing and exploding gradients. (Suh et al., 2022) further points out that first order estimates can also suffer from higher bias or variance in certain cases.

(Levine and Koltun, 2013; Mora et al., 2021) address these issues by computing optimal trajectories using trajectory optimization and then performing imitation learning. (Xu et al., 2022) address the exploding/vanishing gradient issue by only differentiating through a short horizon and using a value function instead at the tail to amortize the value and gradient estimates. (Parag et al., 2022) learns the value function using sobolev descent on optimal trajectories on smooth environments. However, they assume access to accurate gradients from the simulator. Thus, neither of these methods are suitable for learning in the real world using an approximate simulator as (Xu et al., 2022) is an on-policy algorithm and requires multiple parallel runs for each update, and (Mora et al., 2021; Parag et al., 2022) require optimal trajectories which are not easy to obtain from the real system.

**Using approximate gradients from a simulator** We often have access to an approximate differentiable simulator even when trying to learn policies in the real world. One common approach to use this approximate model is to perform model predictive control where we replan the trajectories using the approximate model at each time step (Mayne, 2014). This is also common practice with iterative learning control (Agarwal et al., 2021; Schöllig and D'Andrea, 2009; Moore, 2012; Vemula et al., 2022) based approaches where the jacobians from the simulator are evaluated along the sampled trajectory and used to find the updated actions for some repetitive task. ILC has also been extended to update the model (Abbeel et al., 2006; Jackson et al., 2022) where the model is then used to perform policy optimization or trajectory optimization assuming the learnt model is accurate. (Heess et al., 2015) further extend it by using the real samples and the approximate gradients from the learned model to compute policy gradients. However, they use importance sampled estimates of these policy gradients to compute off-policy updates which is hard to scale and unstable due to the higher variance(Liu et al., 2020; Glynn, 1994).

In this work, we take inspiration from these approaches and propose an update to the value function, using the approximate jacobians from a differentiable simulator, that can be used with any value function based RL algorithm. The suitability of this update for off-policy RL methods makes it particularly appealing compared to most prior work in this area.

## 3. Preliminaries and Background

### 3.1. Notation

We address policy and value function learning in continuous action spaces using approximate simulators with differentiable dynamics and reward functions. We will consider Markov decision processes (MDPs) with infinite horizon, but most of the proposed modifications can easily be extended to finite horizon as well. The MDP state transitions are modelled as a function $f : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$ and the reward using a bounded reward function $r : \mathcal{S} \times \mathcal{A} \rightarrow [r_{min}, r_{max}]$. We also assume access to a simulator $g : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$ which approximately models the transition dynamics $f$. The simulator $g$ and the reward function $r$ are assumed differentiable in most regions of the state action space.

### 3.2. Q-learning based Actor Critic Methods

Q-learning based actor critic methods such as DDPG (Lillicrap et al., 2015) and SAC (Haarnoja et al., 2018) learn a Q-function by performing Bellman backups over the transitions $(s, a, s', r)$ sampled from the MDP as follows :

$$Q(s, a) := r(s, a) + \gamma Q(s', a'(s'))|_{s'=f(s,a)}, \tag{1}$$

where, $a'(s')$ can be obtained by using the current policy estimate $\pi_\theta(s')$ or by solving : $max_{a'} Q(s', a')$. For methods such as DDPG and SAC, we further learn a policy by optimizing

$$\max_\theta \mathcal{J}_\pi(\theta) = \max_\theta \mathbb{E}_{(s\sim\mathcal{D}, a\sim\pi_\theta(.|s))}[Q(s, a) + \alpha\mathcal{R}(\pi_\theta(a|s))], \tag{2}$$

Here, we actually care more about the gradient of the value function than the exact value function itself since the gradient of policy objective only consists of $\nabla_a Q(s,a)$ :

$$\nabla_\theta \mathcal{J}_\pi(\theta) = \mathbb{E}_{s\sim\mathcal{D},\epsilon\sim\mathcal{N}}[\nabla_a Q(s,a)\nabla_\theta a_\theta(s,\epsilon) + \alpha\nabla_\theta \mathcal{R}(\pi_\theta(a_\theta(s,\epsilon)|s)], \tag{3}$$

where $\mathcal{R}(\pi_\theta(a|s))$ here is a regularizer and $a_\theta(s,\epsilon)$ is the policy output.

### 3.3. Iterative Learning Control

For repetitive tasks involving taking the same sequence of actions repeatedly, iterative learning control (ILC) offers a very sample efficient approach to adapt policies using real world rollouts. In this paper, we focus on the recent optimization based reformulations of the ILC problem (Schöllig and D'Andrea, 2009; Agarwal et al., 2021; Vemula et al., 2022) where we evaluate the control inputs by performing rollouts in the true system while computing updates to the controls using the approximate model. In an LQR like setup, this update can be computed by linearizing the dynamics and quadraticizing the cost around the observed trajectory and then computing the LQR updates:

$$\begin{aligned}\min_{\Delta a,\Delta s} J(\Delta s,\Delta a) = &\sum_{t=0}^{N}(s_t+\Delta s_t)^T Q(s_t+\Delta s_t) + \sum_{t=0}^{N-1}(a_t+\Delta a_t)^T R(a_t+\Delta a_t)\\ &\text{s.t}\quad \Delta s_{t+1} = \hat{A}_t^g \Delta s_t + \hat{B}_t^g \Delta a_t\end{aligned} \tag{4}$$

where $s_{0:N}$ is the trajectory observed when the controls $a_{0:N-1}$ are rolled out in the true system $f$, and $\hat{A}_t^g$, $\hat{B}_t^g$ are obtained by linearizing the approximate model $g$ along the observed trajectory.'

## 4. Method

In this section, we introduce the value gradient update. We show that it complements the TD update and can be introduced directly into existing reinforcement learning algorithms. We then point out its relationship with the LQR dual update. We will also discuss how to adapt it to settings when we have access to only approximate jacobians from an imperfect simulator. Finally, we add the value gradient update to SAC as an example of an off-policy RL algorithm incorporating the update.

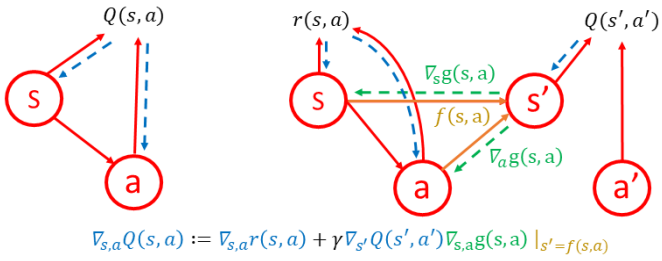


Figure 1: Diagram illustrating the value gradient update with an approximate simulator. The dotted lines indicate the gradient flow and the solid lines indicate the forward pass. The left diagram shows the value gradients directly computed from the Q function whereas the right diagram shows the value gradients computed using a single step rollout with the approximate jacobians $\nabla_{s,a} g(s,a)$ computed using the simulator. Note that the rollout is still computed in the true environment $f(s,a)$. The value gradient update uses the rolled out estimate to supervise the amortized gradients on the left.

### 4.1. Value gradient update

The value gradient update can be derived as a straightforward extension of the TD update presented above by differentiating Eq. 1 with respect to the inputs :

$$\nabla_{s,a} Q(s,a) := \nabla_{s,a} r(s,a) + \gamma \nabla_{s,a} Q(s', a'(s'))|_{s'=f(s,a)}, \tag{5}$$

where the R.H.S can be further expanded by applying chain rule,

$$\nabla_{s,a} Q(s,a) := \nabla_{s,a} r(s,a) + \gamma\Big(\nabla_{s'} Q(s', a'(s')) \; + \nabla_{a'} Q(s',a')\nabla_{s'} a'(s')\Big)\nabla_{s,a} f(s,a). \tag{6}$$

where, $a'(s')$ is the action taken at $s'$ (which could be expressed as $\pi_\theta(s')$, or $\text{argmax}_u\, Q(s', u)$, or simply as a constant $a'$). Figure 1 shows the updates and the corresponding computation graph diagramatically. Note that the R.H.S in Eq 6 can be obtained simply using matrix vector products and doesn't require explicit computation/storage of the jacobians. Thus, one could potentially compute the R.H.S using any auto-diff package by simply computing the target Q values (i.e., R.H.S in Eq. 1) and auto-diff it w.r.t the input (s,a) pair. This is useful especially for on-policy methods where each sample is used for a single or a few updates. Alternatively, for off-policy methods, where the samples are re-used multiple times for update computation, we can explicitly compute and store the jacobians in the buffer. The R.H.S can then be computed by simply plugging in the jacobians in Eq. 6 explicitly while computing the update.

As with most deep RL methods, instead of directly performing the updates in Eq. 5, we parameterize the Q function using a function approximator (say a neural network) and minimize the square of the residual in Eq. 5.

$$\min_{\phi} \mathbb{E}_{(s,a,s',r)\sim\mathcal{D}}\Big[\big(\nabla_{s,a}Q_\phi(s,a) - (\nabla_{s,a}r(s,a) + \gamma\nabla_{s,a}Q_{\bar{\phi}}(s', a'(s'))_\perp\big)^2\Big] \tag{7}$$

where, $\perp$ is the stop gradient operator denoted to mean that the terms within the corresponding parenthesis wouldn't affect the gradient of the loss. We call this the value gradient objective. In practice, we just minimize a weighted combination of the Bellman residual and the above loss together. We can also derive a similar update rule for methods with state value function, $V(s)$, by simply computing the gradient of the corresponding Bellman update and minimizing the gradient residual similar to Eq 7. In the case of actor critic methods, we can simultaneously update the policy using the current value function estimate. For example, with algorithms like DDPG, the policy is updated by taking a gradient step on the optimization problem : $\max_\theta Q(s, a_\theta(s))$.

### 4.2. Relationship with Optimal control

The optimal control formulation for the above problem can be expressed as follows:

$$\begin{aligned} &\max_{s_t,a_t} \sum_{t=0}^{N} r(s_t, a_t) \\ &\text{s.t} \quad s_{t+1} = f(s_t, a_t) \end{aligned} \tag{8}$$

Solving the above problem using iLQR yields the following update to the co-states (multiplier variables corresponding to each dynamics constraint), $\lambda_t^V$, at each iteration

$$\lambda_t^V = (\nabla_{s_t} f(s_t, a_t))^T \lambda_{t+1}^V + \nabla_{s_t} r(s_t, a_t). \tag{9}$$

Interestingly, the co-states are also the gradients of the value function, $\nabla_{s_t} V(s_t)$ and the update we get in Eq. 9 are indeed the value gradient updates we proposed in section 4.1 corresponding to $V$. Likewise, to obtain the value gradient update in Eq. 6 corresponding to the Q-function, we simply need to add the following constraint to the optimization problem above at each time step:

$$a_{t+1} = a'(f(s_t, a_t)) \tag{10}$$

The corresponding co-state/multiplier updates for the concatenated constraints can be written as :

$$\lambda_t^Q = \begin{bmatrix} \nabla_{s_t,a_t} f(s_t, a_t) \\ \nabla_{s'} a'(s') \nabla_{s_t,a_t} f(s_t, a_t)|_{s'=f(s_t,a_t)} \end{bmatrix}^T \lambda_{t+1}^Q + \nabla_{s_t,a_t} r(s_t, a_t) \tag{11}$$

which is the same update as Eq. 6 where $\lambda_t^Q$ corresponds to $\nabla_{s_t,a_t} Q(s_t, a_t)$. This suggests that the update we proposed above actually just solves an amortized version of the iLQR problem where the value function and its gradients are instead represented using a function approximator (say a neural network).

### 4.3. Gradient update with approximate jacobians

The update introduced above assumes access to exact jacobians from the simulator. However, in a lot of scenarios, although we can collect experiences from the true environment (say the real world), we only have access to jacobians evaluated using an approximate simulator. However, taking inspiration from the ILC setup, we observe that we can simply use the approximate jacobians obtained from the simulator and substitute it in Eq 6 to obtain:

$$\nabla_{s,a} Q(s,a) := \nabla_{s,a} r(s,a) + \gamma \Big( \nabla_{s'} Q(s', a'(s)) \; + \nabla_{a'} Q(s', a') \nabla_{s'} a'(s) \Big) \nabla_{s,a} g(s,a). \tag{12}$$

Note that the sample $(s, a, r, s')$ is still obtained from a rollout in the true environment. We simply evaluate the state and action jacobians $\nabla_{s,a} g(s,a)$ using the simulator $g$ at these samples and use it in Eq 12. These jacobians could be computed analytically if the simulator is differentiable or using finite differences when the simulator is non-differentiable.

In practice, as mentioned in section 4.1, we represent the value function and policy as neural nets and minimize a loss comprising a weighted combination of the residual of Eq. 12 and the Bellman residual.

### 4.4. Practical Implementations

As we discussed in section 4.1, the value gradient update can be used with most existing reinforcement learning approaches based on Bellman residual minimization. In this section, we describe one such algorithmic implementations we use in the paper, built on top of an off-policy reinforcement learning algorithm called soft actor-critic (SAC) (Haarnoja et al., 2018).

We simply modify the critic objective in SAC to include the value gradient objective:

$$\mathcal{L}_q(\phi) = \mathbb{E}_{(s,a,s',r)\sim\mathcal{D}} \Big[ \Big( Q_\phi(s,a) - \Big( r(s,a) + \gamma \big( Q_\phi(s', \pi_\theta(s')) - \alpha \log(\pi_\theta(a'|s')) \big) \Big)_\perp \Big)^2 \Big] \tag{13}$$

$$\mathcal{L}_s(\phi) = \mathbb{E}_{(s,a,s')\sim\mathcal{D}} \Big[ \Big( \nabla_s Q_\phi(s,a) - \Big( \nabla_s r(s,a) \; + \gamma \nabla_{s'} \big( Q_{\bar{\phi}}(s', a') - \alpha \log(\pi_\theta(a'|s')) \big) \nabla_s g(s,a) \Big)_\perp \Big)^2 \Big] \tag{14}$$

$$\mathcal{L}_a(\phi) = \mathbb{E}_{(s,a,s')\sim\mathcal{D}} \Big[ \Big( \nabla_a Q_\phi(s,a) - \Big( \nabla_a r(s,a) + \gamma \nabla_{s'} \big( Q_{\bar{\phi}}(s', a') - \alpha \log(\pi_\theta(a'|s')) \big) \nabla_a g(s,a) \Big)_\perp \Big)^2 \Big] \tag{15}$$

$$\min_\phi \mathcal{L}_q(\phi) + \beta_s \mathcal{L}_s(\phi) + \beta_a \mathcal{L}_a(\phi) \tag{16}$$

where $\mathcal{L}_q(\phi)$ is the original Bellman error, $\mathcal{L}_s(\phi)$ is the value gradient error for the state and $\mathcal{L}_a(\phi)$ is the value gradient error for the action, with $\beta_s$ and $\beta_a$ being the corresponding coefficients.

Other than the value gradient update, we keep the rest of the pipeline the same, i.e, we use the same actor objective,

$$\min \mathcal{L}_\pi(\theta) = \mathbb{E}_{s_t\sim D} \big[ \mathbb{E}_{a_t\sim\pi_\theta(.|s_t)} [\alpha \log(\pi_\theta(a_t|s_t)) - Q_\phi(s_t, a_t)] \big], \tag{17}$$

and perform alternating updates on the actor and the critic as in SAC.

**Handling non-smooth dynamics and contact** : For systems with non-smooth dynamics, the jacobians in Eq. 14 and 15 are ill-conditioned leading to very high variance in the updates as pointed out in (Suh et al., 2022). Thus, we instead express the losses in 14 and 15 using a hinge loss with a hinge at the moving average of the median of the loss. This reduces the sensitivity to high magnitude gradients thereby reducing the overall variance of the objective. This is one of the key advantages of our approach compared to analytic policy gradient or other on-policy alternatives where such solutions are not possible leading exploding and vanishing gradient issues.

## 5. Experiments

We present experimental evidence to show that adding the value gradient update leads to better sample efficiency both when training the policies from scratch in the true environment and while transferring policies trained in the approximate simulator to the true environment. For the experiments in this paper, we will refer to the approximate simulator as the *nominal* model and is modelled with a simplified model with perturbed parameters. It's also assumed differentiable unless otherwise specified. We refer to the true environment as the *real* model representing the full model of the system. We aim to learn policies for the real model with as few samples from it as possible.

### 5.1. Tasks

**Cartpole** The task here is a simple swing-up task on a cartpole system. In the real model, we include viscous damping on all joints, coulomb friction at contact points (i.e the floor and cart), and a deadband in the control. None of these were modelled in the nominal model. Further, the mass of the cart and pole in the real system are 2x and 3x the masses in the nominal model respectively.

**Acrobot** The task here is the swing-up task with the acrobot model from (Gillen et al., 2020). We use the same model for both the nominal and the real model except that the first link in the real model is $20\%$ longer and has $20\%$ higher mass.

**Quadrotor** We use the Quadrotor model from (Jackson et al., 2022). The task here is to get the quadrotor from a randomly initialized point to the origin. The real model additionally models the aerodynamic drag terms which are not modelled in the nominal model. The nominal model also has a parametric error of $10\%$ on the system parameters (e.g. mass, rotor arm length, etc.).

**Airplane** We use the high fidelity airplane model constructed from wind-tunnel data (Manchester et al., 2017). Similar to the quadrotor task, the task here is for the airplane to get to the origin from a distribution of randomly initialized initial positions. The real model uses the full model of the system whereas the nominal model uses a simple flat-plate wing model with linear lift and quadratic drag coefficient approximations.

**HalfCheetah** We use the halfcheetah model and reward functions from the Dflex simulator (Xu et al., 2022). We use the original simulator as the nominal model. For the real model, we reduce damping and friction by $30\%$ and increase the contact stiffness and contact damping coefficient 5x.

**Hopper** We use the hopper model and reward functions from the Dflex simulator (Xu et al., 2022). We use the original simulator as the nominal model. For the real model, we reduce damping and friction by $20\%$ and increase the contact stiffness and contact damping coefficient by 5 times.

### 5.2. Algorithm details

We use the SAC implementation from (Tandon) as the baseline RL algorithm with the policies and value functions represented as 2 hidden layer multi-layer perceptrons with relu and tanh nonlinearity respectively. We use the same architecture and hyperparameters for both the finetuning and training

from scratch experiments. For the gradient terms, we observed that the value gradient losses lead to more stable updates when expressed as a hinge loss instead of quadratic loss as pointed out in 4.4. Further, the value gradient loss coefficients $\beta_s$ and $\beta_a$ in Eq 4.4 are computed online to keep the ratio of gradient contributions of the different loss terms roughly constant.

$$\beta_s = \zeta_s \frac{\nabla_\phi \mathcal{L}_q}{\nabla_\phi \mathcal{L}_s}, \qquad \beta_a = \zeta_a \frac{\nabla_\phi \mathcal{L}_q}{\nabla_\phi \mathcal{L}_a}, \tag{18}$$

where $\zeta_s$ and $\zeta_a$ are constant hyperparameters. This heuristic has interesting connections with treating these coefficients as lagrange multipliers. We find that this also stabilizes training. These updates can however be expensive if performed at each iteration. Thus, we perform the updates every 20 iterations for most experiments except the HalfCheetah finetuning experiments (where we update them every 2 iterations).

**Hyperparameters** For most hyperparameters, we stick to the defaults used in (Tandon). We use the entropy coefficient $\alpha = 0.2$ for all experiments, critic hidden size of 512 for all experiments other than cartpole and an actor hidden size of 256 for all experiments. We perform a single gradient update on the policy/value function using samples drawn from the buffer for each step taken in the environment. We use a constant value for the value gradient loss coefficients $\zeta_s = 0.5$ and $\zeta_a = 0.5$ for all experiments except the halfcheetah finetuning experiments where we use $\zeta_s = 0$ and $\zeta_a = 0.5$.

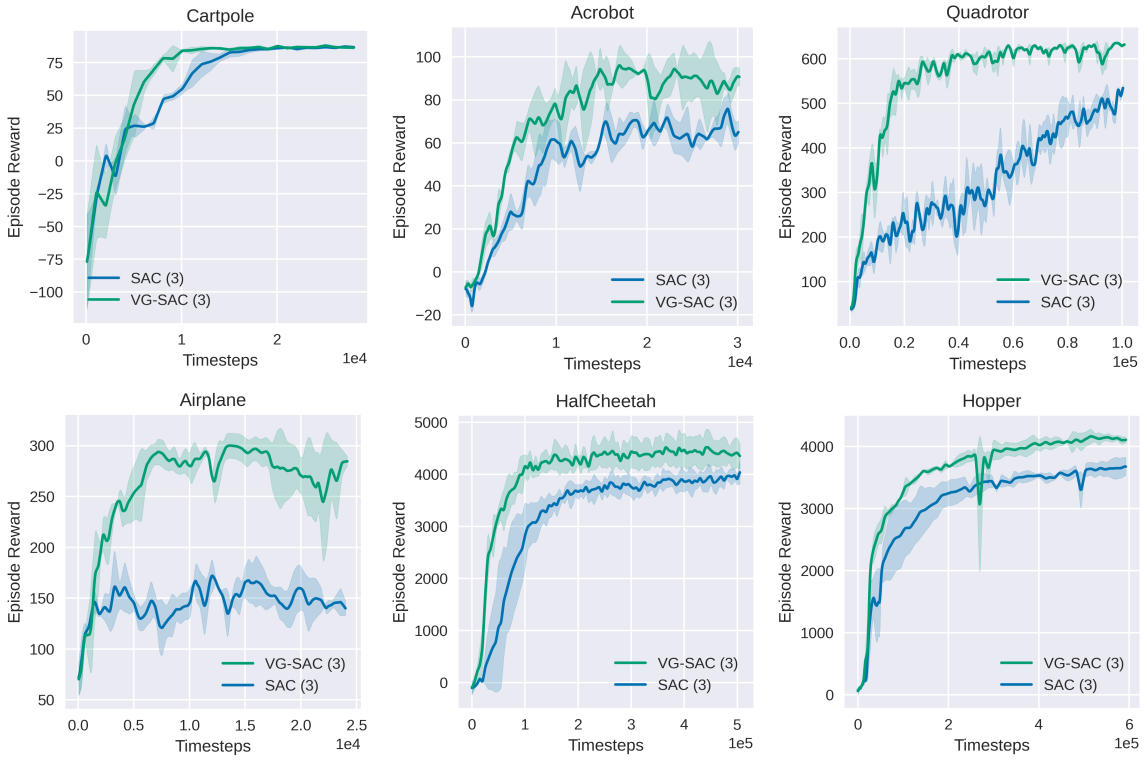


Figure 2: Comparison of methods using the value gradient objective (VG-SAC) v/s RL baselines (SAC) on training from scratch in the real model. SAC is a policy/value function trained from scratch in the real model using Soft actor-critic. VG-SAC is a policy/value function trained from scratch using the value gradient objective. All the plots have been evaluated using 3 random seeds.

### 5.3. Training from scratch

In the first set of experiments, we compare the sample efficiency gains obtained from using the value gradient update when training from scratch in the real-model for all the tasks in 5.1. Specifically, we use a baseline SAC implementation as the RL algorithm and use the approximate dynamics jacobian and reward gradients obtained from the nominal model to compute the value gradient losses in Eq 4.4. Figure 2 shows the training plots comparing the two methods. SAC (Blue) in all the plots represents the RL baseline and VG-SAC (Green) represents our modification which additionally uses the value gradient losses. We observe that across all the environments, using the value gradient objective leads to much faster convergence and in some environments (Airplane, Acrobot and Hopper) leads to superior performance.

**Finite Difference Jacobians** : We adapt VG-SAC to use finite difference jacobians (we call this model VG-SAC-FD) from non-differentiable approximate dynamics models. We compare it with SAC on 2 representative systems in Figure 3. Note that the DMCHalfCheetah model here (unlike experiments in Figure 2) is using the Mujoco model from DM control (Tunyasuvunakool et al., 2020) which is non-differentiable and needs to use finite difference jacobians. We observe that VG-SAC-FD obtains similar improvements in sample efficiency to the full jacobian version, despite using finite difference jacobians from the approximate simulator. This suggests that the value gradient update is effective even when the simulator is not differentiable.

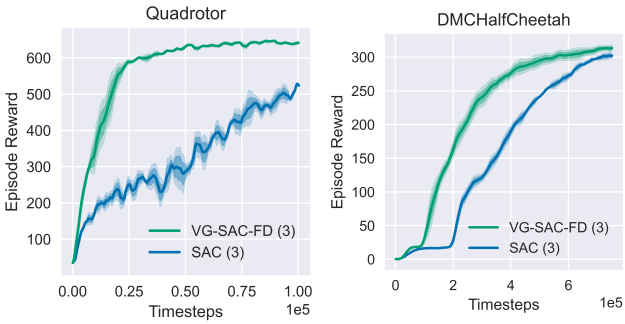

Figure 3: Value gradient update with finite difference jacobians

### 5.4. Finetuning

In this section, we experiment with pre-training the policy and value function using the nominal model and then fine-tuning with the real model. Figure 4 shows the training plots for fine-tuning the pre-trained policies/value functions in the real model. In all the environments, we experiment with pre-training using SAC with and without the value gradient updates. We call them RLpretrain and VGpretrain respectively. Note that, while pre-training using the value gradient objective, we use the accurate gradients since we are pre-training on the nominal model. We pre-train the RL variants for longer to obtain maximum possible performance for both pre-trained policies. The pre-trained policies on Cartpole, Quadrotor, Hopper and Halfcheetah obtain similar returns of approximately 86, 650, 5300 and 4300 respectively in the nominal models for both the RL and the VG variants. However, the pre-trained RL variants in Acrobot and Airplane saturate at lower maximum returns of 84 and 296 as opposed to 109 and 304 for the value gradient variant on the nominal models.

We observe that for both sets of pre-trained networks, using the value gradient objective for fine-tuning leads to significant speedups for fine-tuning on Cartpole, Quadrotor, Airplane and Hopper tasks. However, on Acrobot, all variants struggle to improve during fine-tuning (probably due to exploration related issues). On Half-cheetah, the both fine-tuning methods perform similarly on the VG pretrained networks but VG fine-tuning outperforms the RL fine-tuning on the RL pretrained network.

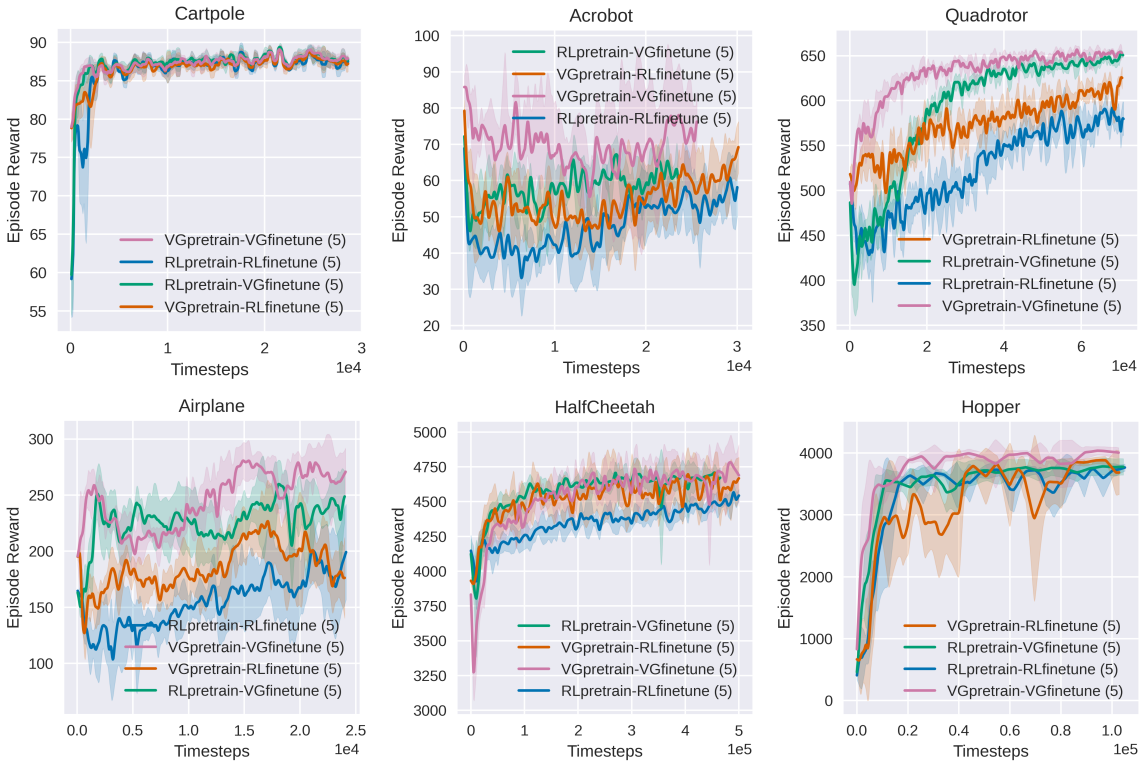


Figure 4: Comparison of methods using value gradient objective v/s the RL baselines (SAC) while transfering a pre-trained policy to a real model. RLpretrain indicates the networks were pretrained in the nominal model using SAC. VGpretrain indicates the networks were pretrained in the nominal model using the value gradient objective. Likewise, RLfinetune and VGfinetune indicate the networks were fine tuned on the real model using SAC and the value gradient objective respectively. All the plots have been evaluated using 5 random seeds.

**Choice of pre-training algorithm** Further, we observe that across all tasks, the networks pre-trained with the value gradient objective are easier to transfer, i.e, they often start with higher rewards and converge with fewer samples. Specifically, comparing RLpretrain-RLfinetune vs VGpretrain-RLfinetune in Figure 4, we observe that although both variants are fine-tuned using the same algorithm, VGpretrain-RLfinetune consistently converges faster and starts off from a higher return than RLpretrain-RLfinetune across most environments (except hopper). The same is true when comparing RLpretrain-VGfinetune and VGpretrain-VGfinetune for most environments (except HalfCheetah where they are comparable). This indicates that using the value gradient objective even for pre-training can significantly boost transfer performance.

## 6. Discussions and Conclusion

We introduce the value gradient update and demonstrate that it can be easily incorporated into any actor critic RL algorithm by using the dynamics jacobians and reward gradients obtained from an (approximate) simulator. We demonstrated the benefits of incorporating the value gradient update in a standard off-policy RL algorithm called soft actor-critic on various sim2sim tasks. We show that adding the value gradient update leads to much better sample efficiency both while training policies from scratch or fine-tuning a policy pre-trained in a perturbed environment. Moreover, we show that policies pre-trained using the value gradient update transfer faster in the new environments showing that it offers benefits for both pre-training and fine-tuning. In the future, we believe these methods need to be validated on hardware to demonstrate that the observed improvements transfer to a sim2real setting.

## Acknowledgments

Swaminathan Gurumurthy is supported by a grant from the Bosch Center for Artificial Intelligence. We further thank Taylor Howell, Simon Le Cleac'h, Chiyen Lee and Ben Boksner for helping us with some experimental setup and providing feedback at various points in the project. We would also like to thank Shaojie Bai for brainstorming ideas especially in the initial stages of the project.

## References

P. Abbeel, Morgan Quigley, and A. Ng. Using inaccurate models in reinforcement learning. *Proceedings of the 23rd international conference on Machine learning*, 2006.

Naman Agarwal, Elad Hazan, Anirudha Majumdar, and Karan Singh. A regret minimization approach to iterative learning control. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 100–109. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/agarwal21b.html.

Suguru Arimoto, Sadao Kawamura, and Fumio Miyazaki. Bettering operation of robots by learning. *J. Field Robotics*, 1:123–140, 1984.

Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

Dimitri Bertsekas. *Dynamic programming and optimal control*, volume 1. Athena scientific, 2012.

Dimitri Bertsekas. *Reinforcement learning and optimal control*. Athena Scientific, 2019.

John T Betts. *Practical Methods for Optimal Control Using Nonlinear Programming*. Society for Industrial and Applied Mathematics, 2001.

Tao Chen, Jie Xu, and Pulkit Agrawal. A system for general in-hand object re-orientation. In *Conference on Robot Learning*, pages 297–307. PMLR, 2022.

Sean Gillen, Marco Molnar, and Katie Byl. Combining deep reinforcement learning and local control for the acrobot swing-up and balance task. *2020 59th IEEE Conference on Decision and Control (CDC)*, pages 4129–4134, 2020.

Peter W Glynn. Importance sampling for markov chains: Asymptotics for the variance. *Stochastic Models*, 10(4):701–717, 1994.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

Nicolas Heess, Gregory Wayne, David Silver, Timothy Lillicrap, Tom Erez, and Yuval Tassa. Learning continuous control policies by stochastic value gradients. *Advances in neural information processing systems*, 28, 2015.

Jemin Hwangbo, Joonho Lee, Alexey Dosovitskiy, Dario Bellicoso, Vassilios Tsounis, Vladlen Koltun, and Marco Hutter. Learning agile and dynamic motor skills for legged robots. *Science Robotics*, 4(26):eaau5872, 2019.

Brian Edward Jackson, Jeong Hun Lee, Kevin Tracy, and Zachary Manchester. Data-efficient model learning for control with jacobian-regularized dynamic mode decomposition. In *6th Annual Conference on Robot Learning*, 2022. URL https://openreview.net/forum?id=ED0G14V3WeH.

Rudolf Emil Kalman et al. Contributions to the theory of optimal control. *Bol. soc. mat. mexicana*, 5(2):102–119, 1960.

Jens Kober, J Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

Sergey Levine and Vladlen Koltun. Guided policy search. In *International conference on machine learning*, pages 1–9. PMLR, 2013.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Yao Liu, Pierre-Luc Bacon, and Emma Brunskill. Understanding the curse of horizon in off-policy evaluation via conditional importance sampling. In *International Conference on Machine Learning*, pages 6184–6193. PMLR, 2020.

Zachary R Manchester, Jeffrey I Lipton, Robert J Wood, and Scott Kuindersma. A variable forward-sweep wing design for enhanced perching in micro aerial vehicles. In *55th AIAA Aerospace Sciences Meeting*, page 0011, 2017.

David Q. Mayne. Differential dynamic programming–a unified approach to the optimization of dynamic systems. *Control and dynamic systems*, 10:179–254, 1973.

David Q. Mayne. Model predictive control: Recent developments and future promise. *Automatica*, 50(12):2967–2986, 2014. ISSN 0005-1098. doi: https://doi.org/10.1016/j.automatica.2014.10.128. URL https://www.sciencedirect.com/science/article/pii/S0005109814005160.

Luke Metz, C Daniel Freeman, Samuel S Schoenholz, and Tal Kachman. Gradients are not all you need. *arXiv preprint arXiv:2111.05803*, 2021.

Kevin L Moore. Iterative learning control for deterministic systems. 1993.

Kevin L Moore. Iterative learning control for deterministic systems. 2012.

Miguel Angel Zamora Mora, Momchil Peychev, Sehoon Ha, Martin Vechev, and Stelian Coros. Pods: Policy optimization via differentiable simulation. In *International Conference on Machine Learning*, pages 7805–7817. PMLR, 2021.

Fabian L Mueller, Angela P Schoellig, and Raffaello D'Andrea. Iterative learning of feed-forward corrections for high-performance tracking. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3276–3281. IEEE, 2012.

Amit Parag, Sébastien Kleff, Léo Saci, Nicolas Mansard, and Olivier Stasse. Value learning from trajectory optimization and sobolev descent: A step toward reinforcement learning with superlinear convergence properties. In *International Conference on Robotics and Automation*, 2022.

Paavo Parmas, Carl Edward Rasmussen, Jan Peters, and Kenji Doya. Pipps: Flexible model-based policy search robust to the curse of chaos. In *International Conference on Machine Learning*, pages 4065–4074. PMLR, 2018.

Angela P Schoellig, Fabian L Mueller, and Raffaello D'andrea. Optimization-based iterative learning for precise quadrocopter trajectory tracking. *Autonomous Robots*, 33(1):103–127, 2012.

Angela Schöllig and Raffaello D'Andrea. Optimization-based iterative learning control for trajectory tracking. In *2009 European Control Conference (ECC)*, pages 1505–1510. IEEE, 2009.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

Hyung Ju Suh, Max Simchowitz, Kaiqing Zhang, and Russ Tedrake. Do differentiable simulators give better policy gradients? In *International Conference on Machine Learning*, pages 20668–20696. PMLR, 2022.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

Pranjal Tandon. Pytorch implementation of soft actor critic. URL https://github.com/pranz24/pytorch-soft-actor-critic.

Saran Tunyasuvunakool, Alistair Muldal, Yotam Doron, Siqi Liu, Steven Bohez, Josh Merel, Tom Erez, Timothy Lillicrap, Nicolas Heess, and Yuval Tassa. dm$_control$ : $Softwareandtasksforcontinuouscontrol$.*Soft* $100022, 2020$. $ISSN2665 - 9638$. $doi$ : . URL https://www.sciencedirect.com/science/article/pii/S2665963820300099.

Anirudh Vemula, Wen Sun, Maxim Likhachev, and J Andrew Bagnell. On the effectiveness of iterative learning control. In *Learning for Dynamics and Control Conference*, pages 47–58. PMLR, 2022.

Zhaoming Xie, Glen Berseth, Patrick Clary, Jonathan Hurst, and Michiel van de Panne. Feedback control for cassie with deep reinforcement learning. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1241–1246. IEEE, 2018.

Jie Xu, Tao Du, Michael Foshey, Beichen Li, Bo Zhu, Adriana Schulz, and Wojciech Matusik. Learning to fly: computational controller design for hybrid uavs with reinforcement learning. *ACM Transactions on Graphics (TOG)*, 38(4):1–12, 2019.

Jie Xu, Miles Macklin, Viktor Makoviychuk, Yashraj Narang, Animesh Garg, Fabio Ramos, and Wojciech Matusik. Accelerated policy learning with parallel differentiable simulation. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=ZSKRQMvttc.