A Pontryagin Perspective on Reinforcement Learning

Onno Eberhard[1,2] OEBERHARD@TUE.MPG.DE
Claire Vernade[2] CLAIRE.VERNADE@UNI-TUEBINGEN.DE
Michael Muehlebach[1] MICHAELM@TUE.MPG.DE
[1]*Max Planck Institute for Intelligent Systems, Tübingen, Germany*
[2]*University of Tübingen*

Editors: N. Ozay, L. Balzano, D. Panagou, A. Abate

Abstract

Reinforcement learning has traditionally focused on learning state-dependent policies to solve optimal control problems in a *closed-loop* fashion. In this work, we introduce the paradigm of *open-loop reinforcement learning* where a fixed action sequence is learned instead. We present three new algorithms: one robust model-based method and two sample-efficient model-free methods. Rather than basing our algorithms on Bellman's equation from dynamic programming, our work builds on *Pontryagin's principle* from the theory of open-loop optimal control. We provide convergence guarantees and evaluate all methods empirically on a pendulum swing-up task, as well as on two high-dimensional MuJoCo tasks, significantly outperforming existing baselines.

Keywords: Reinforcement learning, Open-loop control, Non-convex optimization

1. Introduction

Reinforcement learning (RL) refers to "the optimal control of incompletely-known Markov decision processes" (Sutton and Barto, 2018, p. 2). It has traditionally focused on applying dynamic programming algorithms, such as value iteration or policy iteration, to situations where the environment is unknown. These methods solve optimal control problems in a closed-loop fashion by learning feedback policies, which map states (x_t) to actions (u_t). In contrast, this work introduces the paradigm of *open-loop reinforcement learning* (OLRL), in which fixed action sequences $u_{0:T-1}$, over a horizon T, are learned instead. The closed-loop and open-loop control paradigms are illustrated in Fig. 1.

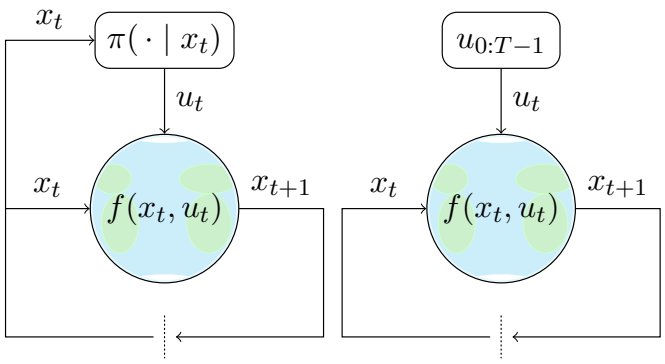

Figure 1: Comparison of closed-loop (left) and open-loop (right) control. In closed-loop RL, the goal is to learn a policy π. In open-loop RL, a fixed sequence of actions $u_{0:T-1}$ is learned instead, with u_t independent of the states $x_{0:t}$.

An open-loop controller receives no observations from its environment. This makes it impossible to react to unpredictable events, which is essential in many problems, particularly those with stochastic or unstable dynamics. For this reason, RL research has historically focused exclusively on closed-loop control. However, many environments are perfectly predictable. Consider the classic example of swinging up an inverted pendulum. If there are no disturbances, then this task can be solved flawlessly without feedback (as we demonstrate in Section 4.1). Where open-loop control is viable, it brings

considerable benefits. As there is no need for sensors, it is generally much cheaper than closed-loop control. It can also operate at much higher frequencies, since there is no bandwidth bottleneck due to sensor delays or computational processing of measurements. Importantly, the open-loop optimal control problem is much simpler, as it only involves optimizing an action sequence (finding one action per time step). In contrast, closed-loop optimal control involves optimizing a policy (finding one action for each state of the system), which can be considerably more expensive. In this way, open-loop control circumvents the curse of dimensionality without requiring function approximation.

For these reasons, open-loop control is widely used in practice (Diehl et al., 2006; van Zundert and Oomen, 2018; Sferrazza et al., 2020), and there exists a large body of literature on the theory of open-loop optimal control (Pontryagin et al., 1962). However, the setting of incompletely-known dynamics has received only little attention. In this work, we introduce a family of three new open-loop RL algorithms by adapting the existing theory to this setting. Whereas closed-loop RL is largely based on approximating the Bellman equation, the central equation of dynamic programming, we base our algorithms on approximations of *Pontryagin's principle*, the central equation of open-loop optimal control. We first introduce a model-based method whose convergence we prove to be robust to modeling errors. This is a novel and non-standard result which depends on a careful analysis of the algorithm. We then extend this procedure to settings with completely unknown dynamics and propose two fully online model-free methods. Finally, we empirically demonstrate the robustness and sample efficiency of our methods on an inverted pendulum swing-up task and on two complex MuJoCo tasks.

Related work. Our work is inspired by numerical optimal control theory (Betts, 2010; Geering, 2007), which deals with the numerical solution of trajectory optimization problems. Whereas existing methods assume that the dynamics are known, our algorithms only require an approximate model (model-based OLRL) or no model at all (model-free OLRL), and rely on a simulator to provide samples. An in-depth review of related work can be found in Appendix A. All appendices are contained in the full version of this paper (Eberhard et al., 2024).

2. Background

We consider a reinforcement learning setup with continuous state and action spaces $\mathcal{X} \subset \mathbb{R}^D$ and $\mathcal{U} \subset \mathbb{R}^K$. Each episode lasts T steps, starts in the fixed initial state x_0, and follows the deterministic dynamics $f : \mathcal{X} \times \mathcal{U} \to \mathcal{X}$, such that $x_{t+1} = f(x_t, u_t)$ for all times $t \in [T-1]_0$.[1] After every transition, a deterministic reward $r(x_t, u_t) \in \mathbb{R}$ is received, and at the end of an episode, an additional terminal reward $r_T(x_T) \in \mathbb{R}$ is computed. The value of state x_t at time t is the sum of future rewards

$$v_t(x_t; u_{t:T-1}) \doteq \sum_{\tau=t}^{T-1} r(x_\tau, u_\tau) + r_T(x_T) = r(x_t, u_t) + v_{t+1}\{f(x_t, u_t); u_{t+1:T-1}\},$$

where we defined v_T as the terminal reward function r_T. Our goal is to find a sequence of actions $u_{0:T-1} \in \mathcal{U}^T$ maximizing the total sum of rewards $J(u_{0:T-1}) \doteq v_0(x_0; u_{0:T-1})$. We will tackle this trajectory optimization problem using gradient ascent. Although our goal is to learn an open-loop controller (an action sequence), we assume that the state is fully observed during the training process.

Pontryagin's principle. The gradient of the objective function J with respect to the action u_t is

$$\nabla_{u_t} J(u_{0:T-1}) = \nabla_u r(x_t, u_t) + \nabla_u f(x_t, u_t) \underbrace{\nabla_x v_{t+1}(x_{t+1}; u_{t+1:T-1})}_{\lambda_{t+1} \in \mathbb{R}^D}, \tag{1}$$

1. For $n \in \mathbb{N}$, we write $[n] \doteq \{1, 2, \dots, n\}$ and $[n]_0 \doteq \{0, 1, \dots, n\}$. Unless explicitly mentioned, all time-dependent equations hold for all $t \in [T-1]_0$.

where the terms of J related to the earlier time steps $\tau \in [t-1]_0$ vanish, as they do not depend on u_t. We denote Jacobians as $(\nabla_y f)_{i,j} \doteq \frac{\partial f_j}{\partial y_i}$. The *costates* $\lambda_{1:T}$ are defined as the gradients of the value function along the given trajectory. They can be computed through a backward recursion:

$$\lambda_T \doteq \nabla v_T(x_T) = \nabla r_T(x_T) \tag{2}$$

$$\lambda_t \doteq \nabla_x v_t(x_t; u_{t:T-1}) = \nabla_x r(x_t, u_t) + \nabla_x f(x_t, u_t)\lambda_{t+1}. \tag{3}$$

The gradient (1) of the objective function can thus be obtained by means of one forward pass through the dynamics f (a rollout), yielding the states $x_{0:T}$, and one backward pass through (2) and (3), yielding the costates $\lambda_{1:T}$. The stationarity condition arising from setting (1) to zero, where the costates are computed from (2) and (3), is known as *Pontryagin's principle*. (Pontryagin's principle in fact goes much further than this, as it generalizes to infinite-dimensional and constrained settings.) We re-derive (1) to (3) using the method of Lagrange multipliers in Appendix C.

3. Method

If the dynamics are known, then the trajectory can be optimized by performing gradient ascent with the gradients computed according to Pontryagin's equations (1) to (3). In this work, we adapt this idea to the domain of reinforcement learning, where the dynamics are unknown. In RL, we are able to interact with the environment, so the forward pass through the dynamics f is not an issue. However, the gradient computation according to Pontryagin's principle requires the Jacobians $\nabla_x f_t \doteq \nabla_x f(x_t, u_t)$ and $\nabla_u f_t \doteq \nabla_u f(x_t, u_t)$ of the unknown dynamics. In our methods, which follow the structure of Algorithm 1, we therefore replace these Jacobians by estimates $A_t \simeq \nabla_x f_t$ and $B_t \simeq \nabla_u f_t$. Before discussing concrete methods for open-loop RL, whose main concern is the construction of appropriate estimates A_t and B_t, we first show that replacing $\nabla_x f_t$ and $\nabla_u f_t$ in this way is sensible. In particular, we show that, under certain assumptions on the accuracy of A_t and B_t, Algorithm 1 converges to an unbiased local optimum of the true objective J. In the following sections we then discuss model-based and model-free open-loop RL methods.

3.1. Convergence of Algorithm 1

Our convergence result relies on the following three assumptions.

Assumption 1 *All rewards are encoded in the terminal reward r_T. In other words, $r(x, u) = 0$ for all $x \in \mathcal{X}$ and $u \in \mathcal{U}$.*

This assumption is without loss of generality, since we can augment the state x_t by a single real variable ρ_t that captures the sum of the running rewards (i.e., $\rho_0 = 0$ and $\rho_{t+1} = \rho_t + r(x_t, u_t)$). An equivalent setup that satisfies Assumption 1 is then obtained by defining a new terminal reward function $r'_T(x_T, \rho_T) \doteq r_T(x_T) + \rho_T$ and setting the running rewards r' to zero.

Algorithm 1: Open-loop reinforcement learning

Input: Optimization steps $N \in \mathbb{N}$, step size $\eta > 0$

Initialize $u_{0:T-1}$ (initial action sequence)
for $k = 1, 2, \ldots, N$ **do**
  $x_{0:T} \leftarrow \text{rollout}(u_{0:T-1})$ `// Forw. pass`
  `// Backward pass`
  $\tilde{\lambda}_T \leftarrow \nabla r_T(x_T)$
  for $t = T-1, T-2, \ldots, 0$ **do**
    `// Jacobian estimation`
    $A_t, B_t \simeq \nabla_x f(x_t, u_t), \nabla_u f(x_t, u_t)$
    `// Pontryagin update`
    $\tilde{\lambda}_t \leftarrow \nabla_x r(x_t, u_t) + A_t \tilde{\lambda}_{t+1}$
    $g_t \leftarrow \nabla_u r(x_t, u_t) + B_t \tilde{\lambda}_{t+1}$
    $u_t \leftarrow u_t + \eta g_t$ `// Grad. ascent`

Assumption 2 *There exist constants $\gamma, \zeta > 0$ with $\gamma + \zeta + \gamma\zeta < 1$ such that for any trajectory $(u_{0:T-1}, x_{0:T})$ encountered by Algorithm 1, the following properties hold for all $t \in [T-1]_0$:*

(a) The error of A_{t+s} is bounded, for all $s \in [T-t]$, in the following way:

$$\|A_{t+s} - \nabla_x f_{t+s}\| \leq \frac{\gamma}{3^s} \frac{\underline{\sigma}(\nabla_u f_t)}{\bar{\sigma}(\nabla_u f_t)} \left\{ \prod_{i=1}^{s-1} \frac{\underline{\sigma}(\nabla_x f_{t+i})}{\bar{\sigma}(\nabla_x f_{t+i})} \right\} \underline{\sigma}(\nabla_x f_{t+s}).$$

(b) The error of B_t is bounded in the following way: $\|B_t - \nabla_u f_t\| \leq \zeta \underline{\sigma}(\nabla_u f_t)$.

Here, $\underline{\sigma}(A)$ and $\bar{\sigma}(A)$ denote the minimum and maximum singular value of A, and $\|A\| \doteq \bar{\sigma}(A)$.

This assumption restricts the errors of the estimates A_t and B_t that are used in place of the true Jacobians $\nabla_x f_t$ and $\nabla_u f_t$ in Algorithm 1. Although the use of the true system for collecting rollouts prevents a buildup of error in the forward pass, any error in the approximate costate $\tilde{\lambda}_t$ can still be amplified by the Jacobian estimates of earlier time steps, A_τ for $\tau \in [t-1]$, during the backward pass. Thus, to ensure convergence to a stationary point of the objective function J, the errors of these estimates need to be small. This is particularly important for t close to T, as these errors will be amplified over more time steps. Assumption 2 provides a quantitative characterization of this intuition.

Assumption 3 *There exists a constant $L > 0$ such that, for all action sequences $u^A_{0:T-1}, u^B_{0:T-1} \in \mathcal{U}^T$ and all times $t \in [T-1]_0$, $\|\nabla_{u_t} J(u^A_{0:T-1}) - \nabla_{u_t} J(u^B_{0:T-1})\| \leq L\|u^A_t - u^B_t\|$.*

This final assumption states that the objective function J is L-smooth with respect to the action u_t at each time step $t \in [T-1]_0$, which is a standard assumption in nonconvex optimization. This implies that the dynamics f are smooth as well. We are now ready to state the convergence result.

Theorem 4 *Suppose Assumptions 1 to 3 hold with γ, ζ, and L. Let $\mu \doteq 1 - \gamma - \zeta - \gamma\zeta$ and $\nu \doteq 1 + \gamma + \zeta + \gamma\zeta$. If the step size η is chosen small enough such that $\alpha \doteq \mu - \frac{1}{2}\eta L \nu^2$ is positive, then the iterates $(u^{(k)}_{0:T-1})_{k=0}^{N-1}$ of Algorithm 1 satisfy, for all $N \in \mathbb{N}$ and $t \in [T-1]_0$,*

$$\frac{1}{N} \sum_{k=0}^{N-1} \|\nabla_{u_t} J(u^{(k)}_{0:T-1})\|^2 \leq \frac{J^\star - J(u^{(0)}_{0:T-1})}{\alpha \eta N},$$

where $J^\star \doteq \sup_{u \in \mathcal{U}^T} J(u)$ is the optimal value of the initial state.

Proof See Appendix E. The proof depends on a novel intricate analysis of the backpropagation procedure in the case of an accurate forward pass and an inaccurate backward pass. This technique may also be applicable to other (non-control) domains, as described in Appendix A. ■

3.2. Model-based open-loop RL

The most direct way to approximate the Jacobians $\nabla_x f_t$ and $\nabla_u f_t$ is by using a (learned or manually designed) differentiable model $\tilde{f} : \mathcal{X} \times \mathcal{U} \to \mathcal{X}$ of the dynamics f and setting $A_t = \nabla_x \tilde{f}(x_t, u_t)$ and $B_t = \nabla_u \tilde{f}(x_t, u_t)$ in Line 6 of Algorithm 1. Theorem 4 guarantees that this *model-based* open-loop RL method (see Algorithm B.1) is robust to a certain amount of modeling error. In contrast to this, consider the more naive method of using the model to directly obtain a gradient by differentiating

$$J(u_{0:T-1}) \simeq r(x_0, u_0) + r\{\underbrace{\tilde{f}(x_0, u_0)}_{\tilde{x}_1}, u_1\} + \cdots + r_T\{\underbrace{\tilde{f}(\tilde{f}(\cdots \tilde{f}(\tilde{f}(x_0, u_0), u_1)\cdots), u_{T-1})}_{\tilde{x}_T}\}$$

with respect to the actions $u_{0:T-1}$ using the backpropagation algorithm. In Appendix D, we show that this *planning* approach is exactly equivalent to an approximation of Algorithm 1 where, in addition to setting $A_t = \nabla_x \tilde{f}(x_t, u_t)$ and $B_t = \nabla_u \tilde{f}(x_t, u_t)$, the forward pass of Line 3 is replaced by the imagined forward pass $\tilde{x}_{0:T}$ through the model $\tilde{f}$. In Section 4, we empirically demonstrate that this planning method, whose convergence is not guaranteed by Theorem 4, is much less robust to modeling errors than the open-loop RL approach. Note that neither method is related to *model-predictive control* (MPC), which relies on measurements to re-plan at every step. MPC is a closed-loop method that solves a fundamentally different problem from the one we address in this work.

3.3. Model-free on-trajectory open-loop RL

Access to a reasonably accurate model may not always be feasible, and as Algorithm 1 only requires the Jacobians of the dynamics along the current trajectory, a global model is also not necessary. In the following two sections, we propose two methods that directly estimate the Jacobians $\nabla_x f_t$ and $\nabla_u f_t$ from rollouts in the environment. We call these methods *model-free*, as the estimated Jacobians are only valid along the current trajectory, and thus cannot be used for planning.

Our goal is to estimate the Jacobians $\nabla_x f(\bar{x}_t, \bar{u}_t)$ and $\nabla_u f(\bar{x}_t, \bar{u}_t)$ that lie along the trajectory induced by the action sequence $\bar{u}_{0:T-1}$. These Jacobians measure how the next state ($\bar{x}_{t+1}$) changes if the current state or action ($\bar{x}_t$, $\bar{u}_t$) are slightly perturbed. More formally, the dynamics f may be linearized about the reference trajectory $(\bar{u}_{0:T-1}, \bar{x}_{0:T})$ as

$$\underbrace{f(x_t, u_t) - f(\bar{x}_t, \bar{u}_t)}_{\Delta x_{t+1}} \simeq \nabla_x f(\bar{x}_t, \bar{u}_t)^\top \underbrace{(x_t - \bar{x}_t)}_{\Delta x_t} + \nabla_u f(\bar{x}_t, \bar{u}_t)^\top \underbrace{(u_t - \bar{u}_t)}_{\Delta u_t},$$

which is a valid approximation if the perturbations $\Delta x_{0:T}$ and $\Delta u_{0:T-1}$ are small. By collecting a dataset of $M \in \mathbb{N}$ rollouts with slightly perturbed actions, we can thus estimate the Jacobians by solving the (analytically tractable) least-squares problem

$$\underset{[A_t^\top \; B_t^\top] \in \mathbb{R}^{D \times (D+K)}}{\arg\min} \sum_{i=1}^{M} \| A_t^\top \Delta x_t^{(i)} + B_t^\top \Delta u_t^{(i)} - \Delta x_{t+1}^{(i)} \|^2. \tag{4}$$

This technique is illustrated in Fig. 2a (dashed purple line). Using these estimates in Algorithm 1 yields a model-free method we call *on-trajectory*, as the gradient estimate relies only on data generated based on the current trajectory (see Algorithm B.2 for details). We see a connection to on-policy methods in closed-loop reinforcement learning, where the policy gradient estimate (or the Q-update) similarly depends only on data generated under the current policy. Like on-policy methods, on-trajectory methods will benefit greatly from the possibility of parallel environments, which could reduce the effective complexity of the forward pass stage from $M+1$ rollouts to that of a single rollout.

Exploiting the Markovian structure. Consider a direct linearization of the objective function J about the current trajectory. Writing the action sequence as a vector $\bar{\boldsymbol{u}} \doteq \mathrm{vec}(\bar{u}_{0:T-1}) \in \mathbb{R}^{TK}$, this linearization is given, for $\boldsymbol{u} \in \mathbb{R}^{TK}$ close to $\bar{\boldsymbol{u}}$, by

$$J(\boldsymbol{u}) \simeq J(\bar{\boldsymbol{u}}) + \nabla J(\bar{\boldsymbol{u}})^\top (\boldsymbol{u} - \bar{\boldsymbol{u}}).$$

We can thus estimate the gradient of the objective function by solving the least squares problem

$$\nabla J(\bar{\boldsymbol{u}}) \simeq \underset{\boldsymbol{g} \in \mathbb{R}^{TK}}{\arg\min} \sum_{i=1}^{M} \{ J(\boldsymbol{u}_i) - J(\bar{\boldsymbol{u}}) - \boldsymbol{g}^\top (\boldsymbol{u}_i - \bar{\boldsymbol{u}}) \}^2,$$

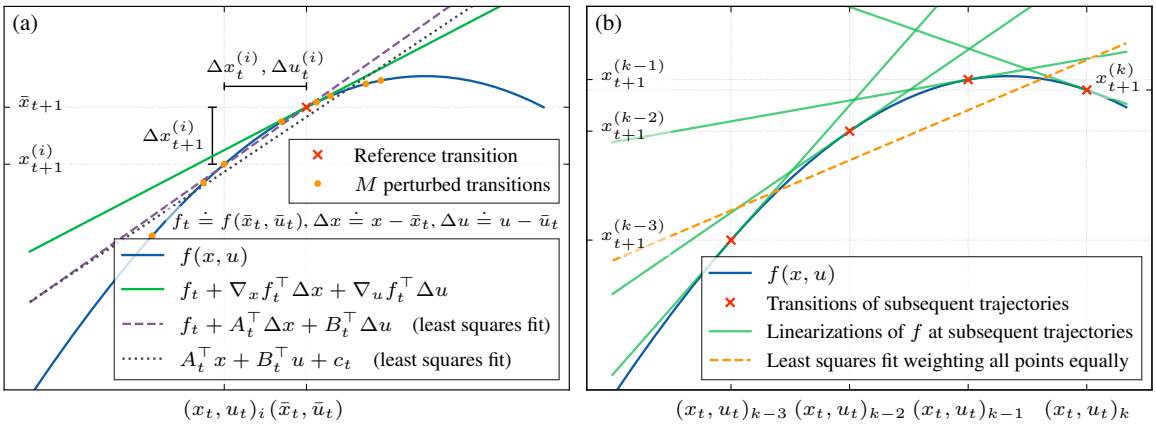

Figure 2: (a) The Jacobians of f (slope of the green linearization) at the reference point $(\bar{x}_t, \bar{u}_t)$ can be estimated from the transitions $\{(x_t^{(i)}, u_t^{(i)}, x_{t+1}^{(i)})\}_{i=1}^M$ of M perturbed rollouts. (b) The Jacobians of subsequent trajectories (indexed by k) remain close. To estimate the Jacobian at iteration k, the most recent iterate $(k-1)$ is more relevant than older iterates.

where $\{\boldsymbol{u}_i\}$ are $M \in \mathbb{N}$ slightly perturbed action sequences. Due to the dimensionality of $\bar{\boldsymbol{u}}$, this method requires $\mathcal{O}(TK)$ rollouts to estimate the gradient. In contrast to this, our approach leverages the Markovian structure of the problem, including the fact that we observe the states $x_{0:T}$ in each rollout. As the Jacobians are estimated jointly at all time steps, we can expect to get a useful gradient estimate from only $\mathcal{O}(D^2 + DK)$ rollouts, which significantly reduces the sample complexity if T is large. This gain in efficiency is demonstrated empirically in Section 4.

3.4. Model-free off-trajectory open-loop RL

The on-trajectory algorithm is sample-efficient in the sense that it leverages the problem structure, but a key inefficiency remains: the rollout data sampled at each iteration is discarded after the action sequence is updated. In this section, we propose an *off-trajectory* method that implicitly uses the data from previous trajectories to construct the Jacobian estimates. Our approach is based on the following observation. If the dynamics f are smooth and the step size η is small, then the updated trajectory $(u_{0:T-1}^{(k)}, x_{0:T}^{(k)})$ will remain close to the previous iterate $(u_{0:T-1}^{(k-1)}, x_{0:T}^{(k-1)})$. Furthermore, the Jacobians along the updated trajectory will be similar to the previous Jacobians, as illustrated in Fig. 2b. Thus, we propose to estimate the Jacobians along the current trajectory from *a single rollout only* by bootstrapping our estimates using the Jacobian estimates from the previous iteration.

Consider again the problem of estimating the Jacobians from multiple perturbed rollouts, illustrated in Fig. 2a. Instead of relying on a reference trajectory and (4), we can estimate the Jacobians by fitting a linear regression model to the dataset of M perturbed transitions. Solving

$$\underset{[A_t^\top \; B_t^\top \; c_t] \in \mathbb{R}^{D \times (D+K+1)}}{\arg\min} \sum_{i=1}^{M} \| A_t^\top x_t^{(i)} + B_t^\top u_t^{(i)} + c_t - x_{t+1}^{(i)} \|^2 \tag{5}$$

yields an approximate linearization $f(x_t, u_t) \simeq A_t^\top x_t + B_t^\top u_t + c_t = F_t z_t$, with $F_t \doteq [A_t^\top \; B_t^\top \; c_t]$ and $z_t \doteq (x_t, u_t, 1) \in \mathbb{R}^{D+K+1}$. This approximation is also shown in Fig. 2a (dotted gray line).[2]

2. If we replace (4) by (5) in Algorithm B.2, we get a slightly different on-trajectory method with similar performance.

At iteration k, given the estimate $F_t^{(k-1)}$ and a new point $z_t^{(k)} = (x_t^{(k)}, u_t^{(k)}, 1)$ with corresponding target $x_{t+1}^{(k)}$, computing the new estimate $F_t^{(k)}$ is a problem of online linear regression. We solve this regression problem using an augmented version of the *recursive least squares* (RLS) algorithm (e.g., Ljung, 1999, Sec. 11.2). By introducing a prior precision matrix $Q_t^{(0)} \doteq q_0 I$ for each time t, where $q_0 > 0$, we compute the update at iteration $k \in \mathbb{N}$ (see Algorithm B.3) as

$$\begin{aligned} Q_t^{(k)} &= \alpha Q_t^{(k-1)} + (1-\alpha) q_0 I + z_t^{(k)} \{z_t^{(k)}\}^\top \\ F_t^{(k)} &= F_t^{(k-1)} + \{Q_t^{(k)}\}^{-1} z_t^{(k)} \{x_{t+1}^{(k)} - F_t^{(k-1)} z_t^{(k)}\}^\top . \end{aligned} \tag{6}$$

Forgetting and stability. The standard RLS update of the precision matrix corresponds to (6) with $\alpha = 1$. In the limit as $q_0 \to 0$, the RLS algorithm is equivalent to the batch processing of (5), which treats all points equally. However, as illustrated in Fig. 2b, points from recent trajectories should be given more weight, as transitions that happened many iterations ago will give little information about the Jacobians along the current trajectory. We can incorporate a *forgetting factor* $\alpha \in (0, 1)$ into the precision update with the effect that past data points are exponentially downweighted:

$$Q_t^{(k)} = \alpha Q_t^{(k-1)} + z_t^{(k)} \{z_t^{(k)}\}^\top \quad \rightsquigarrow \quad Q_t^{(k)} = \alpha^k q_0 I + \sum_{i=1}^{k} \alpha^{k-i} z_t^{(i)} \{z_t^{(i)}\}^\top . \tag{7}$$

This forgetting factor introduces a new problem: instability. If subsequent trajectories lie close to each other, then the sum of outer products may become singular (e.g., if all $z_t^{(i)}$ are identical, then the sum has rank 1). As the prior $q_0 I$ is downweighted, at some point inverting Q may become numerically unstable. Our modification in (6) adds $(1-\alpha) q_0 I$ in each update, which has the effect of removing the α^k coefficient in front of $q_0 I$ in (7). If the optimization procedure converges, then eventually subsequent trajectories will indeed lie close together. Although (6) prevents issues with instability, the quality of the Jacobian estimates will still degrade, as this estimation inherently requires perturbations (see Section 3.3). In Algorithm B.3, we thus slightly perturb the actions used in each rollout to get more diverse data.

4. Experiments

4.1. Inverted pendulum swing-up

We empirically evaluate our algorithms on the inverted pendulum swing-up task shown in Fig. 3. As a performance criterion we define $J_{\max} \doteq \max_{k \in [N]_0} J(u_{0:T-1}^{(k)})$ as the return achieved by the best action sequence over a complete learning process of N optimization steps. The task is considered *solved* if $J_{\max}$ exceeds a certain threshold. A detailed description of the task is given in Appendix F. We repeat our experiments with 100 random seeds and show 95% bootstrap confidence intervals in all plots.

Robustness: model-based open-loop RL. In Theorem 4, we proved that our model-based open-loop RL method (Algorithm B.1) can accommodate some model error and still converge to a local maximum of the true objective. To test the robustness of our algorithm against model misspecification, we use a pendulum system with inaccurate parameters as the model $\tilde{f}$. Concretely, if m_i is the i^{th} parameter of the true system (cf. Appendix F), we sample the corresponding model parameter $\tilde{m}_i$ from a log-normal distribution centered at m_i, such that $\tilde{m}_i = \xi m_i$, with $\ln \xi \sim \mathcal{N}(0, s^2)$. The severity of the model error is then controlled by the scale parameter s. In Fig. 4, we compare the performance of our method with the planning procedure described in Section 3.2, in which the forward pass is performed through the model $\tilde{f}$ instead of the real system f. Whereas the planning

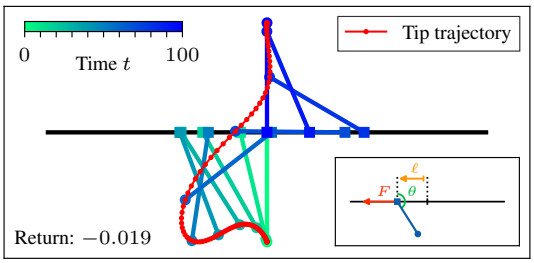

Figure 3: The inverted pendulum swing-up task. The goal is to control the force F such that the tip of the pendulum swings up above the base. The shown solution was found by the on-trajectory method of Section 3.3.

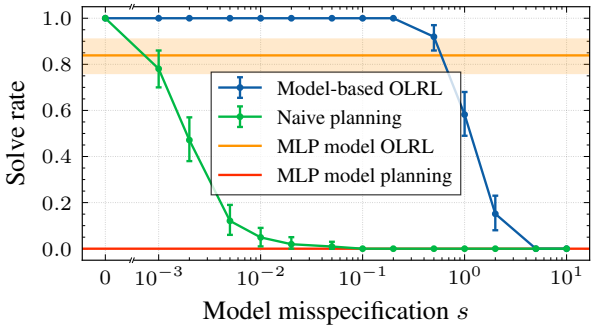

Figure 4: The model-based open-loop RL algorithm can solve the pendulum problem reliably even with a considerable model error.

method only solves the pendulum reliably with the true system as the model ($s = 0$), the open-loop RL method can accommodate a considerable model misspecification.

In a second experiment, we represent the model $\tilde{f}$ by a small multi-layer perceptron (MLP). The model is learned from 1000 rollouts, with the action sequences sampled from a pink noise distribution, as suggested by Eberhard et al. (2023). Figure 4 compares the performance achieved with this model by our algorithm and by the planning method. As the MLP model represents a considerable misspecification of the true dynamics, only the open-loop RL method manages to solve the pendulum task.

Structure: on-trajectory open-loop RL. Our model-free on-trajectory method (Algorithm B.2) uses rollouts to directly estimate the Jacobians needed to update the action sequence. It is clear from (4) that more rollouts (i.e., larger M) will give more accurate Jacobian estimates, and therefore increase the quality of the gradient approximation. In Fig. 5, we analyze the sample efficiency of the this algorithm by comparing the performance achieved at different values of M, where the number N of optimization steps remains fixed. We compare our method to the finite-difference approach described at the end of Section 3.3 and to the gradient-free cross-entropy method (CEM; Rubinstein, 1999). Both these methods also update the action sequence on the basis of M perturbed rollouts in the environment. As in our method, the M action sequences are perturbed using Gaussian white noise with noise scale σ. We describe both baselines in detail in Appendix H. The *oracle* performance shown in Fig. 5 corresponds to Algorithm 1 with the true gradient, i.e., $A_t = \nabla_x f_t$ and $B_t = \nabla_u f_t$.

Figure 5 shows that the performance of both the finite-difference method and CEM heavily depends on the choice of the noise scale σ, whereas our method performs identically for all three values of σ. Even for tuned values of σ, the finite-difference method and CEM still need approximately twice as many rollouts per iteration as the open-loop RL method to reliably swing up the pendulum. At 10 rollouts per iteration, our method matches the oracle's performance, while both baselines are below the *solved* threshold. This empirically confirms our theoretical claims at the end of Section 3.3, where we argue that exploiting the Markovian structure of the problem leads to increased sample efficiency.

Efficiency: off-trajectory open-loop RL. Finally, we turn to the method proposed in Section 3.4 (Algorithm B.3), which promises increased sample efficiency by estimating the Jacobians in an off-trajectory fashion. The performance of this algorithm is shown in Fig. 6, where the learning curves of all our methods as well as the two baselines and the oracle are plotted. For the on-trajectory methods compared in Fig. 5, we chose for each the minimum number of rollouts M such that, under the best choice for σ, the method would reliably solve the swing-up task. The hyperparameters for

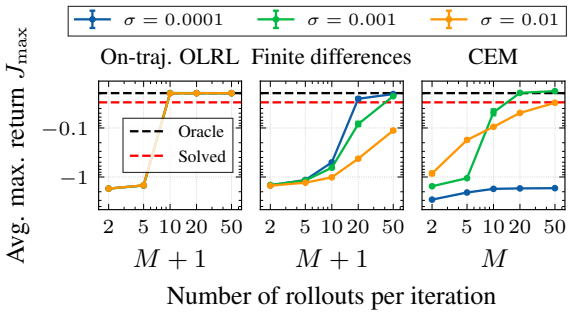

Figure 5: The on-trajectory open-loop RL method is more sample-efficient than the finite-difference and cross-entropy methods. It is also much less sensitive to the noise scale σ.

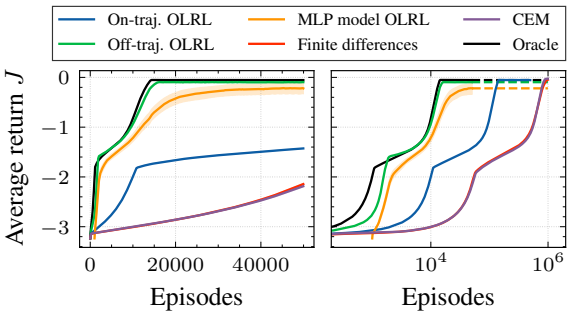

Figure 6: Learning curves on the pendulum task. On the right, we show a longer time period in log scale. The off-trajectory open-loop RL method converges almost as fast as the oracle method.

all methods are summarized in Appendix I. It can be seen that the off-trajectory method, which only requires one rollout per iteration, converges much faster than the on-trajectory open-loop RL method.

4.2. MuJoCo

While the inverted pendulum is illustrative for analyzing our algorithms empirically, it is a relatively simple task with smooth, low-dimensional, and deterministic dynamics. In this section, we test our method in two considerably more challenging environments: the `Ant-v4` and `HalfCheetah-v4` tasks provided by the OpenAI Gym library (Brockman et al., 2016; Towers et al., 2023), implemented in MuJoCo (Todorov et al., 2012). These environments are high-dimensional, they exhibit non-smooth contact dynamics, and the initial state is randomly sampled at the beginning of each episode.

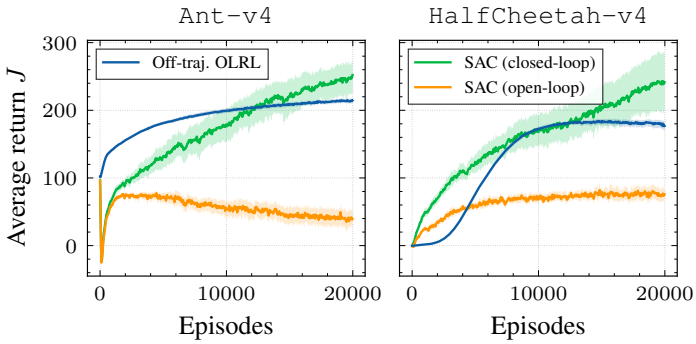

Figure 7: Learning curves of our off-trajectory open-loop RL method and soft actor-critic (SAC) for two MuJoCo tasks. All experiments were repeated with 20 random seeds, and we show 95%-bootstrap confidence intervals for the average return. The horizon is fixed to $T = 100$.

We tackle these two tasks with our model-free off-trajectory method (Algorithm B.3). The results are shown in Fig. 7, where we compare to the closed-loop RL baseline soft actor-critic (SAC; Haarnoja et al., 2018). It can be seen that the open-loop RL method performs comparably to SAC, even though SAC learns a closed-loop policy that is capable of adapting its behavior to the initial condition.[3] In the figure, we also analyze the open-loop performance achieved by SAC. Whereas the closed-loop performance is the return obtained in a rollout where the actions are taken according to the mean of the Gaussian policy, the open-loop return is achieved by blindly executing exactly the same actions in a new episode. The discrepancy in performance is thus completely due to the stochasticity in the initial state. In Appendix G, we show that our method also works with a longer horizon T.

3. In this comparison, our method is further disadvantaged by the piecewise constant "health" terms in the reward function of `Ant-v4`. Our method, exclusively relying on the gradient of the reward function, ignores these.

The results demonstrate that the open-loop RL algorithm is robust to a certain level of stochasticity in the initial state of stable dynamical systems. Additionally, while our convergence analysis depends on the assumption of smooth dynamics, these experiments empirically demonstrate that the algorithms are also able to tackle non-smooth contact dynamics. Finally, we see that the high dimensionality of the MuJoCo systems is handled without complications. While soft actor-critic is an elegant and powerful algorithm, the combination with deep function approximation can make efficient learning more difficult. Our methods are considerably simpler and, because they are based on Pontryagin's principle rather than dynamic programming, they evade the curse of dimensionality by design, and thus do not require any function approximation.

5. Discussion

This paper makes an important first step towards understanding how principles from open-loop optimal control can be combined with ideas from reinforcement learning while preserving convergence guarantees. We propose three algorithms that address this *open-loop RL* problem, from robust trajectory optimization with an approximate model to sample-efficient learning under fully unknown dynamics. This work focuses on reinforcement learning in continuous state and action spaces, a class of problems known to be challenging (Recht, 2019). Although this setting allows us to leverage continuous optimization techniques, we expect that most ideas will transfer to the discrete setting, and we would be interested to see further research on this topic.

It is interesting to note that there are many apparent parallels between our open-loop RL algorithms and their closed-loop counterparts. The distinction between model-based and model-free methods is similar to that in closed-loop RL. Likewise, the on-trajectory and off-trajectory methods we present show a tradeoff between sample efficiency and stability that is reminiscent of the tradeoffs between on-policy and off-policy methods in closed-loop RL. The question of exploration, which is central to reinforcement learning, also arises in our case. We do not address this complex problem thoroughly here but instead rely on additive Gaussian noise to sample diverse trajectories.

Limitations of open-loop RL. Another important open question is how open-loop methods fit into the reinforcement learning landscape. An inherent limitation of these methods is that an open-loop controller can, by definition, not react to unexpected changes in the system's state, be it due to random disturbances or an adversary. An open-loop controller cannot balance an inverted pendulum in its unstable position[4], track a reference trajectory in noisy conditions, or play Go, where reactions to the opponent's moves are constantly required. In these situations, open-loop RL is not viable or only effective over a very short horizon T. However, if the disturbances are small, and the system is not sensitive to small changes in state or action (roughly speaking, if the system is stable and non-chaotic), then a reaction is not necessary, and open-loop RL works even for long horizons T (as we highlight in our MuJoCo experiments, cf. Appendix G). Open-loop control can be viewed as a special case of closed-loop control, and therefore it is clear that closed-loop control is much more powerful. Our algorithms provide a first solution to the open-loop RL problem and are not intended to replace any of the existing closed-loop RL algorithms. In control engineering, it is common to combine feedback and feedforward techniques. In many situations, it can be shown that such a combination will significantly outperform a solution based on feedback alone (e.g., Åström and Murray, 2021, Sec. 12.4). We believe that ultimately a combination of open-loop and closed-loop techniques will also be fruitful in reinforcement learning and think that this is an important direction for future research.

4. Except with a clever trick called *vibrational control* (Meerkov, 1980).

Acknowledgments

We thank the International Max Planck Research School for Intelligent Systems (IMPRS-IS) for their support. C. Vernade is funded by the German Research Foundation (DFG) under both the project 468806714 of the Emmy Noether Programme and under Germany's Excellence Strategy – EXC number 2064/1 – Project number 390727645. M. Muehlebach is funded by the German Research Foundation (DFG) under the project 456587626 of the Emmy Noether Programme.

References

Karl J. Åström and Richard M. Murray. *Feedback Systems: An Introduction for Scientists and Engineers*. Princeton University Press, second edition, 2021. 10

John T. Betts. *Practical Methods for Optimal Control and Estimation Using Nonlinear Programming*. SIAM, 2010. 2

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym. *arXiv:1606.01540*, 2016. URL http://arxiv.org/abs/1606.01540. 9

Moritz Diehl, Hans Georg Bock, Holger Diedam, and Pierre-Brice Wieber. Fast direct multiple shooting algorithms for optimal robot control. In *Fast Motions in Biomechanics and Robotics: Optimization and Feedback Control*, pages 65–93. Springer, 2006. URL https://doi.org/10.1007/978-3-540-36119-0_4. 2

Onno Eberhard, Jakob Hollenstein, Cristina Pinneri, and Georg Martius. Pink noise is all you need: Colored noise exploration in deep reinforcement learning. In *Proceedings of the Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=hQ9V5QN27eS. 8

Onno Eberhard, Claire Vernade, and Michael Muehlebach. A Pontryagin perspective on reinforcement learning. *arXiv:2405.18100*, 2024. URL http://arxiv.org/abs/2405.18100. 2

Hans P. Geering. *Optimal Control with Engineering Applications*. Springer, 2007. 2

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1856–1865. PMLR, 2018. URL http://proceedings.mlr.press/v80/haarnoja18b.html. 9

Lennart Ljung. *System Identification: Theory for the User*. Prentice Hall, 1999. 7

Semyon M. Meerkov. Principle of vibrational control: Theory and applications. *IEEE Transactions on Automatic Control*, 25(4):755–762, 1980. URL https://doi.org/10.1109/TAC.1980.1102426. 10

Lev S. Pontryagin, Vladimir G. Boltayanskii, Revaz V. Gamkrelidze, and Evgenii F. Mishchenko. *Mathematical Theory of Optimal Processes*. Wiley, 1962. 2

Benjamin Recht. A tour of reinforcement learning: The view from continuous control. *Annual Review of Control, Robotics, and Autonomous Systems*, 2:253–279, 2019. URL https://doi.org/10.1146/annurev-control-053018-023825. 10

Reuven Rubinstein. The cross-entropy method for combinatorial and continuous optimization. *Methodology and Computing in Applied Probability*, 1:127–190, 1999. URL https://doi.org/10.1023/A:1010091220143. 8

Carmelo Sferrazza, Michael Muehlebach, and Raffaello D'Andrea. Learning-based parametrized model predictive control for trajectory tracking. *Optimal Control Applications and Methods*, 41 (6):2225–2249, 2020. URL https://doi.org/10.1002/oca.2656. 2

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT press, second edition, 2018. 1

Emanuel Todorov, Tom Erez, and Yuval Tassa. MuJoCo: A physics engine for model-based control. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, 2012. URL https://doi.org/10.1109/IROS.2012.6386109. 9

Mark Towers, Jordan K. Terry, Ariel Kwiatkowski, John U. Balis, Gianluca de Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Arjun KG, Markus Krimmel, et al. Gymnasium, 2023. URL https://github.com/Farama-Foundation/Gymnasium. 9

Jurgen van Zundert and Tom Oomen. On inversion-based approaches for feedforward and ILC. *Mechatronics*, 50:282–291, 2018. URL https://doi.org/10.1016/j.mechatronics.2017.09.010. 2