2017 IEEE 56th Annual Conference on Decision and Control (CDC)
December 12-15, 2017, Melbourne, Australia

Improving state-action space exploration in reinforcement learning using geometric properties

Ion Matei, Raj Minhas, Johan de Kleer and Anurag Ganguli

Abstract— **Learning a model or learning a policy that optimizes some objective function relies on data-sets that describe the behavior of the system. When such sets are unavailable or insufficient, additional data may be generated through new experiments (if feasible) or through simulations (if an accurate model is available). In this paper we describe a third alternative that is based on the availability of a qualitative model of the physical system. In particular, we show how the number of experiments used in reinforcement learning can be reduced by leveraging geometric properties of the system. The geometric properties are independent of any particular instantiation of the qualitative model. As an illustrative example, we apply our approach to a cart-pole system.**

I. Introduction

Learning models of physical systems or learning policies for optimizing some objective function relies on potentially large data-sets that describe the behavior of the system. When such data-sets are not available, we look for alternatives to supplement the training data set. Such alternatives may include data originating from simulations, when a model of the system is available, or from experiments on the real system. Both alternatives bring their own challenges. For example, to build a physics-based model from first principles, we need information about the physical processes that govern the behavior of the system, and the set of parameters that control these physical processes. Unfortunately, it is usually the case that such parameters are not easily available [8]. Often the components of a physical system originate from different manufacturers who are typically not keen to share technical proprietary information about their products. Performing experiments in the field may also not be feasible, since the operator of the system may not be willing to disturb the normal operation of deployed systems.

We propose a third alternative that neither involves model simulations, nor performing experiments. Our approach is based on using available data as a seed for generating a new source of virtual data that describes the behavior of the system. In particular, we apply a transformation to a time series describing a trajectory of the system, and obtain a different time series that describes another *feasible* trajectory. This is called *symmetry*, and it represents a function that maps a trajectory to one or many feasible trajectories [3]. The number of additional trajectories depends on the symmetry type. In the case of discrete symmetries, one additional trajectory can be generated. In the case of parameterized symmetries, theoretically, we can generate infinitely many. In this paper we limit ourselves to systems whose behavior can be described by ordinary differential equations (ODEs) or difference equations (DEs).

Finding symmetries for ODEs is a rather challenging task. In the case of ODEs that model physical systems, we can look for a number possible symmetries such as rotation, translation or scaling. Checking whether a map is a symmetry requires some knowledge about the system. In the context of this paper, we assume we have an understanding of the physical processes behind the behavior of the system. With this understanding, we can produce a system of parameterized ODEs that generates a mathematical model. However the parameters of the ODEs are assumed to be *unknown*. We refer to this mathematical model as a *qualitative* model. The dynamical behavior of qualitative models has been extensively studied in [6], [7], [17]. In this paper, we are interested in geometrical properties of such models that will enable learning. For example, the motion of a cart-pole can be described by two ODEs. The unknown parameters include the cart/pole masses, length of the pendulum, moment of inertia, and so on. By leveraging the geometric properties of the physical system, we show how to modify the reinforcement algorithm so that we can extend the coverage of the state-action space during learning. We use the popular cart-pole system as a demonstrative example.

Paper structure: In Section II we briefly describe the Markov decision problem (MDP) and recall the Q-learning algorithm [16]. Section III introduces the notion of a symmetry map for a dynamical system and shows how the Q-learning algorithm can be enhanced by including new data generated through the symmetry maps. Special cases are discussed in deterministic and stochastic setups. In Section IV we compare the original SARSA algorithm to its symmetry-based modified version in the context of learning a stabilizing policy for a cart-pole system.

II. Reinforcement learning

Reinforcement learning refers to finding a map between situations and actions in order to maximize a reward function. The learning is performed through discovery of states-actions that improve the reward function. We focus on a particular type of reinforcement algorithms based on direct methods [15], in which the goal is to approximate the optimal action-value function directly by iterating the value function along trajectory samples. In the model free cases, learning

Ion Matei, Johan de Kleer and Anurag Ganguli are with the System Sciences Laboratory at Palo Alto Research Center (PARC) (emails: ion.matei@parc.com,dekleer@parc.com,anurag.ganguli@parc.com); Raj Minhas is with the Interactions and Analytics Laboratory at PARC (email: raj.minhas@parc.com).

the optimal policy depends on the richness of the data-set describing the response of the system as a result of actions applied to it. This data set is referred to as *experience*. In the following, we describe how system symmetries can be used to cope with the lack of sufficient experience for learning. The main idea is that at the same time with the standard value iterations along the sample-trajectories in the experience data set, we also iterate the value function over trajectories generated through the system symmetries. This approach achieves higher coverage of the state-action space and improves convergence rates.

Let the tuple $(\mathcal{X}, \mathcal{A}, P, r, \gamma)$ be a Markov decision process (MDP), where $\mathcal{X} \subset \mathbb{R}^n$ is a compact state-space, $\mathcal{A} \subset \mathbb{R}^m$ is the finite action space, P is the transition probability for the underlying Markov process $P(X_{k+1} \in S_{k+1} | X_{0:k} = x_{0:k}, U_{0:k} = u_{0:k}) = P(X_{k+1} \in S_{k+1} | X_k = x_k, U_k = u_k) = P(S_{k+1} | x_k, u_k)$, with $S_k \subset \mathcal{X}$, and where $r : \mathcal{X} \times \mathcal{A} \times \mathcal{X} \to \mathbb{R}$ is a deterministic reward function, and $0 < \gamma \leq 1$ is the positive discount factor. Solving the MDP translates to finding a policy that selects control actions $\{U_k\}_{k \geq 0}$ maximizing the expected total discounted reward $V(\{U_k\}, x) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k R(X_k, U_k) | X_0 = x\right]$, where $R(x,u)$ is a random function representing the reward received for using control input u in state x. In particular $\mathbb{E}[R(x,u)] = \int_{\mathcal{X}} r(x,u,y) dP(y|x,u)$, where $r(x,u,y)$ is the deterministic reward function that sets the reward for making the transition from x to y as a result of applying control input u.

The optimal action sequence is obtained by computing the *optimal value function* $V^*(x)$, where $V^*(x) = \max_{\{U_t\}} \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k R(X_k, U_k) | X_0 = x\right]$. The optimal value function satisfies the Bellman optimality equation $V^*(x) = \max_{u \in \mathcal{U}} \mathbb{E}\left[R(X_k, U_k) + \gamma V^*(X_{k+1}) | X_k = x, U_k = u\right] = \max_{u \in \mathcal{U}} \int_{\mathcal{X}} \left[r(x,u,y) + \gamma V^*(y)\right] dP(y|x,u)$, where X_{k+1} is the state the system reaches as a result of applying the input u, in state x, and with conditional probability distribution $P(X_{k+1} \in S | X_k = x, U_k = u)$. The optimal Q-function is given by $Q^*(x,u) = \mathbb{E}\left[R(X_k, U_k) + \gamma V^*(X_{k+1}) | X_k = x, U_k = u\right] = \int_{y \in \mathcal{X}} \left[r(x,u,y) + \gamma V^*(y)\right] dP(y|x,u)$, representing the expected discounted cost when the current state and action are x and u, respectively. The optimal control sequence $\{U_k\}$ is derived from solving a sequence of optimization problems of the form

$$U_k^* = \arg\max_{u \in \mathcal{U}} Q^*(X_k, u), \ \forall k, \tag{1}$$

which defines an optimal policy $\pi^* : \mathcal{X} \to \mathcal{A}$. This policy defines the probability distribution of the actions for each current state, that is, $P(U_k = u | X_k = x) = \pi_k(x)$. The policy is called stationary when it does not explicitly depend on time, and deterministic when actions are assigned with probability one. In this paper, we focus on deterministic and stationary policies. Therefore, once the function Q^* is known for all pairs (x,u), the optimal policy can be exactly determined. If the transition probability P is known, a fixed-point iteration can be used to determine the function Q^*. In practice, however this is not always the case, especially when the system model is not completely available, which is our working assumption. The Q-learning algorithm [16] was invented to overcome this hurdle, In this algorithm, given an (infinite) sample trajectory $\{x_k\}$, and corresponding control and reward sequences $\{u_k\}$ and $\{r_k\}$, the Q-function is learnt by successive updates of the form

$$Q(x_k, u_k) \leftarrow Q(x_k, u_k) + \alpha_k \Delta, \tag{2}$$

where $\{\alpha_k\}$ is the step-size sequence and Δ is the temporal difference

$$\Delta = r(x_k, u_k, x_{k+1}) + \gamma \max_{b \in \mathcal{A}} Q(x_{k+1}, b) - Q(x_k, u_k). \tag{3}$$

It is well understood [14] that if each pair (x,u) is updated infinitely often and $\sum_k \alpha_k(x,u) = \infty$, $\sum_k \alpha_k^2(x,u) < \infty$, iteration (2) converges with probability one to the optimal Q-function. In the case where the state-action space is too large (which is the case when the state space is continuous), generating experiences to update each state-action pair (x,u) is not feasible. A common solution to this challenge is to use an approximation for the Q-function, which can be represented as a table, linear function, decision tree, or more complex functions represented by parameterized neural networks [9], [10]. The basic Q-learning algorithm is shown in Algorithm 1. A common choice for selecting the policy π is the ϵ-greedy policy that encourages exploration of the state space at the beginning of the algorithm. Conditions for the convergence of Algorithm 1 are introduced in [14].

Algorithm 1: Q-learning algorithm

Initialization:
  $Q(\cdot,\cdot)$ arbitrarly
for *each episode* **do**
  Initialize x
  for *each step k of the episode* **do**
    Choose u for state x using a policy π derived from Q
    Apply control input u and observe the next state x^+, $r(x,u,x^+)$
    Update the Q-function values: $Q(x,u) \leftarrow Q(x,u) + \alpha_k \left[r(x,u,x^+) + \gamma \max_u Q(x^+,u) - Q(x,u)\right]$
    (Update the approximation of the Q-function)
Compute the optimal policy according to (1)

Obtaining an accurate approximation for the Q-function depends on the size of the training data, which is dictated by the available experience (explored sample trajectories). To extend the learning beyond the experienced actions, approaches based on explicit coding by using CMAC algorithm [13], or based on using additional models [12] have been proposed. Other ideas propose spreading the cost predictions towards neighborhoods of the visited states [11]. Similar to the spatial spreading idea, we propose spreading of action value updates towards state-action pairs different from the ones visited at certain instances. Different from [11], we do not necessarily choose neighborhoods of the visited states, but use a procedure that selects new state-action pairs that result from geometric properties of the system. Before describing in more detail what we mean by geometric properties, we give the intuition of our approach. Suppose that $\{x_k\}$ is a

sample trajectory of the underlying Markov process obtained with some policy $\pi(x_k)$. The corresponding sample control process and rewards are denoted as $\{u_k\}$ and $\{r_k\}$, respectively. Then using the Q-learning algorithm we can iteratively learn the function Q for pairs (x_k, u_k). Assume that we have an invertible map $\Gamma : \mathcal{X} \times \mathcal{A} \to \mathcal{X} \times \mathcal{A}$ that generates new state-action pairs $\Gamma(x,u) = (\hat{x}, \hat{u})$. Moreover, Γ is constructed in such a way that if we apply the transformation Γ on all pairs (x_k, u_k) and obtain $\Gamma(x_k, u_k) = (\hat{x}_k, \hat{u}_k))$ the sequence $\{\hat{x}_k\}$ is also a feasible trajectory of the systems resulting from a control sequence $\{\hat{u}_k\}$. An immediate consequence of this is that we can simultaneously update the Q-value function using both (x_k, u_k) and $(\hat{x}_k, \hat{u}_k)$, by adding a new update iteration similar to (3). In addition, if the reward function satisfies certain assumptions introduced later in the paper, we can show that in fact $Q(\hat{x}_k, \hat{u}_k)$ can be directly computed from $Q(x_k, u_k)$, without having to actually add a new iteration. To be more concrete, let us consider the cart-pole example, and assume that starting from an initial position x_0 and angle θ_0 we perform actions $\{u_0, u_1\}$ aimed at stabilizing the cart-pole at the origin $x = 0$, $\theta = 0$ (Figure 1). We would expect that if we start from $(-x_0, -\theta_0)$, by mirroring the control inputs from the previous case, that is, applying the sequence $\{-u_0, -u_1\}$, we will again move the cart-pole towards the origin (Figure 2). More importantly, this will happen regardless of the parameters of the system (e.g., the cart, pendulum masses, length of the pendulum, and so on). In the following section we discuss the properties a system must satisfy so that the approach we described intuitively can be formally presented.

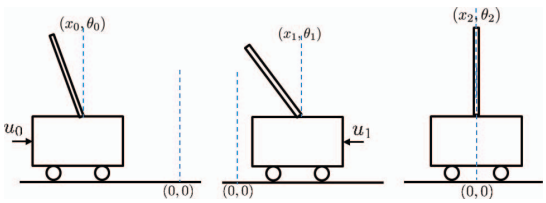

Fig. 1: Cart-pole is brought to the origin by applying $\{u_0, u_1\}$ starting from (x_0, θ_0)

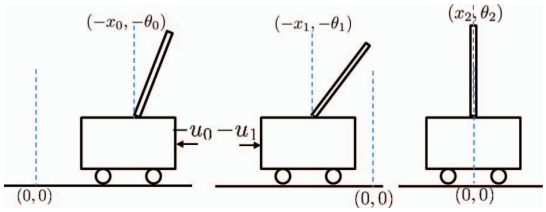

Fig. 2: Cart-pole is brought to the origin by applying $\{-u_0, -u_1\}$ starting from $(-x_0, -\theta_0)$

III. Symmetry based cost spreading

In this section we first introduce the symmetry notion for difference equations. Next, we show how symmetry maps can be used to augment the experience with virtual state transitions that can be further used to evaluate the Q-function values at additional state-action pairs.

A. Symmetries

Let us assume that behavior of a (physical) system is described by the difference equation

$$x_{k+1} = f(x_k, u_k; \theta) \tag{4}$$

where $x_k \in \mathbb{R}^n$ is the state vector, $u_k \in \mathbb{R}^m$ is the vector of inputs (actions) and θ is a set of parameters. We assume that we know the (physical) laws that govern the behavior of the system, but the vector of parameters θ is *unknown*. In the cart-pole example, this means that we can write discrete-time equations that govern the motion of the cart and pendulum as a result of applying a force to the cart, but parameters such as cart and pendulum masses, or the length of the pendulum are unknown. As an immediate extension to the symmetry notion for ordinary differential equations [3], we introduce the symmetry notion for difference equations: maps that transform sample trajectories into other sample trajectories. The solution of (4) can be represented as $x_k = f(f(\dots f(f(x_0, u_0), u_1) \dots, u_{k-2}), u_{k-1})$.

A map $\Gamma : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n \times \mathbb{R}^m$ is a symmetry if it is a diffeomorphism[1] and $(\hat{x}, \hat{u}) = \Gamma(x,u)$ is also a solution of (4), namely $\hat{x}_k = f(f(\dots f(f(\hat{x}_0, \hat{u}_0), \hat{u}_1) \dots, \hat{u}_{k-2}), \hat{u}_{k-1})$.

Example 3.1: Consider a linear system $x_{k+1} = Ax_k + Bu_k$, with solution

$$x_k = A^k x_0 + \sum_{i=0}^{k-1} A^{k-i} B u_i. \tag{5}$$

We look for a symmetry map for which we impose a particular structure, namely $\Gamma(x,u) = (Mx, Nu) = (\hat{x}, \hat{u})$, with M and N invertible matrices of appropriate dimensions. Using the symmetry condition we have that $\hat{x}_k = MA^k M^{-1} \hat{x}_0 + \sum_{i=0}^{k-1} MA^{k-i} BN^{-1} \hat{u}_i$ which can be checked that it has the same form as (5) provided that $AM = MA$ and $BN = MB$.■

The symmetry map Γ can be represented in vector form as $\Gamma(x,u) = [\Gamma_x(x,u), \Gamma_u(x,u)]$, where $\hat{x} = \Gamma_x(x,u)$ and $\hat{u} = \Gamma_u(x,u)$. In the case where the input is chosen based on a stationary, state dependent policy $u = \pi(x)$, we have that $\hat{x} = \Gamma_x(x, \pi(x)) = \bar{\Gamma}_x(x)$, and $\hat{u} = \Gamma_u(x, \pi(x)) = \bar{\Gamma}_u(x) = (\bar{\Gamma}_u \circ \bar{\Gamma}_x^{-1})(\hat{x})$. The inverse transformations are $x = \bar{\Gamma}_x^{-1}(\hat{x})$ and $u = (\pi \circ \bar{\Gamma}_x^{-1})(\hat{x})$. In addition, the symmetry condition can now be expressed as $\bar{\Gamma}_x \circ f = f \circ \bar{\Gamma}_x$.

B. Modified Q-learning algorithm

In this section we show how we can use symmetry maps to improve state-action space exploration, by updating state-action pairs that are not part of the experience episodes. We first consider the deterministic case, that is, $dP(x_{k+1}|x_k, u_k) = \delta(f(x_k, u_k))dx_{k+1}$, where $\delta(x)$ is the Dirac delta function. Next we consider a particular class of stochastic difference equations.

[1] A diffeomorphism is a smooth invertible mapping whose inverse is also smooth.

1) Deterministic case: For a deterministic behavior dictated by a difference equation as in (4), the Bellman optimality equation for the Q-function becomes $Q^*(x,u) = r(x,u,y) + \gamma \max_b Q^*(y,b)$, where $y = f(x,u)$. Using the symmetry maps, we can generate new trajectories starting from some given experience, trajectories that can be also used to update the Q-function values. The modified version of the Q-learning algorithm is shown in Algorithm 2, where we introduced a new iteration for the trajectory generated by the symmetry maps. Note that we applied the Q-function iteration at some pair (x,u) after we computed the control input u^+ corresponding to the next state x^+ resulted from applying u. The reason is because we need the pair (x^+,u^+) to compute the alternative trajectory $(\hat{x}^+,\hat{u}^+) = \Gamma(x^+,u^+)$. The modification of the Q-learning can be replicated to other learning algorithms. One such example is the SARSA algorithm [14], that differs from the Q-learning algorithm through the policy choice.

Algorithm 2: Modified Q-learning algorithm

Initialization:
  $Q(\cdot,\cdot)$ arbitrarly
for *each episode* **do**
  Initialize x
  Choose u for state x using a policy π derived from Q
  for *each step k of the episode* **do**
    Apply control input u and observe x^+, $r(x,u,x^+)$
    Choose u^+ for state x^+ using a policy π derived from Q
    Update the Q-function values: $Q(x,u) \leftarrow Q(x,u) + \alpha_k\left[r(x,u,x^+) + \gamma \max_a Q(x^+,a) - Q(x,u)\right]$
    Compute $(\hat{x},\hat{u}) = \Gamma(x,u)$ and $(\hat{x}^+,\hat{u}^+) = \Gamma(x^+,u^+)$, where Γ is a symmetry map
    Update the Q-function values: $Q(\hat{x},\hat{u}) \leftarrow Q(\hat{x},\hat{u}) + \alpha_k\left[r(\hat{x},\hat{u},\hat{x}^+) + \gamma \max_a Q(\hat{x}^+,a) - Q(\hat{x},\hat{u})\right]$
    (Update the approximation of the Q-function)
Compute the optimal policy according to (1)

Under certain conditions described in what follows, the Q-function evaluations at points generated from symmetry maps can be directly computed from the Q-function values computed at points from the experience. The following results shows a relation between optimal values of V and Q computed at experience points and their counterparts when applying a symmetry map to them.

Proposition 3.1: Let Γ be a symmetry of (4), and let $(\hat{x},\hat{u})$ be a state-action pair resulting from applying the symmetry map to a state-action pair (x,u), that is, $(\hat{x},\hat{u}) = \Gamma(x,u)$. Assume also that there exists a scalar η so that the reward function satisfies the constraint $r(\hat{x},\hat{u},\hat{y}) = \eta r(x,u,y)$, for all pairs (x,u), where $y = f(x,u)$ and $\hat{y} = f(\hat{x},\hat{u})$. Then the optimal-value and optimal Q-functions satisfy $V^*(\hat{x}) = \eta V^*(x)$ and $Q^*(\hat{x},\hat{u}) = \eta Q^*(x,u)$. In addition, if $u^* = \pi^*(x)$ is the optimal policy, it holds that $\hat{u}^* = (\bar{\Gamma}_u^* \circ \bar{\Gamma}_x^{*-1})(\hat{x})$, where $\bar{\Gamma}_x^*(x) = \Gamma_x(x,\pi^*(x))$ and $\bar{\Gamma}_u^*(x) = \Gamma_u(x,\pi^*(x))$.■

Corollary 3.1: Let $\{(x_k,u_k,r_k)\}_{k\geq 0}$ be a state-action-reward trajectory of (4) and all assumptions of Proposition 3.1 hold. Assume also that in addition to the standard iteration of the Q-learning algorithm, we perform the update $Q_{k+1}(\hat{x}_k,\hat{u}_k) = \eta Q_{k+1}(x_k,u_k)$, where $(\hat{x}_k,\hat{u}_k) = \Gamma(x_k,u_k)$. Then the modified Q-learning algorithm converges with probability one to the optimal Q-function. ■

Provided the assumption of Proposition 3.1 hold, Algorithm 2 can be simplified as follows.

Algorithm 3: Simplification of the modified Q-learning algorithm

Initialization:
  $Q(\cdot,\cdot)$ arbitrarly
for *each episode* **do**
  Initialize x
  for *each step k of the episode* **do**
    Choose u for state x using a policy π derived from Q
    Apply control input u and observe x^+, $r(x,u,x^+)$
    Update the Q-function values: $Q(x,u) \leftarrow Q(x,u) + \alpha_k\left[r(x,u,x^+) + \gamma \max_a Q(x^+,a) - Q(x,u)\right]$
    Compute $(\hat{x},\hat{u}) = \Gamma(x,u)$
    Update the Q-function values and the state-action pair: $Q(\hat{x},\hat{u}) = \eta Q(x,u)$, $x \leftarrow x^+$, $u \leftarrow u^+$
    (Update the approximation of the Q-function)
Compute the optimal policy according to (1)

Remark 3.1: The use of the symmetry-based trajectories in the Q-learning algorithm has two main advantages: (1) expansion of the state-action space exploration which is very important when we use function approximations to represent the Q-function; (2) potential increase in the speed of convergence, since Q-function updates at pair (x_k,u_k) may have already happened when updating the Q-function along the symmetry-based trajectories.

2) Stochastic case: If we consider the case where the randomness originates from the initial conditions, and the dynamics of the system is given by $X_{k+1} = f(X_k,U_k;\theta)$, then the analysis would follow the same direction as in the previous section. This follows from the fact that, as in the deterministic case, $dP(x_{k+1}|x_k,u_k) = \delta(f(x_k,u_k))dx_{k+1}$, that is, we can exactly characterize the next state given the current state and the current control input values. A more interesting case is when we consider a noise that affects the evolution of the dynamical system. In particular, we consider the map $f : \mathcal{X}\times\mathcal{A}\times\mathcal{W} \to \mathcal{X}$, and we assume that the behavior of the system is described by a stochastic difference equation

$$X_{k+1} = f(X_k,U_k,W_k;\theta), \tag{6}$$

where in addition to the states and inputs, an i.i.d. noise W_k perturbs the state evolution. Extensions of the symmetry notion in the context of stochastic differential equations (SDE) have been studied in [1], [2], [4], [5]. We can extend the definition of a stochastic symmetry to the stochastic difference equations. Let $\{X_k,U_k,W_k\}$ be a solution of (6). The map $\Gamma : \mathcal{X}\times\mathcal{A}\times\mathcal{X} \to \mathcal{X}\times\mathcal{A}\times\mathcal{X}$ is a *strong symmetry* of (6) if $(\hat{X}_k,\hat{U}_k,W_k) = \Gamma(X_k,U_k,W_k) = (\Phi(X_k,U_k),W_k)$ is a solution of (6), where $\Phi : \mathcal{X}\times\mathcal{A} \to \mathcal{X}\times\mathcal{A}$ is a diffeomorphism. The *strong* attribute associated to the symmetry comes from the assumption that the symmetry leaves the statistical properties of the noise unchanged [1]. The following result introduces conditions for connecting the optimal Q-function computed

along a trajectory of (6), to the optimal Q-function computed along a new trajectory obtained from applying the symmetry map to the initial trajectory.

Proposition 3.2: Let $\{(X_k, U_k)\}_{k\geq 0}$ be a trajectory of (6), where U_k is chosen based on some policy $U_k = \pi(X_k)$, and where $X_0 = x_0$. Let $\{(\hat{X}_k, \hat{U}_k)\}_{k\geq 0}$ be an alternative trajectory obtained by applying a strong stochastic symmetry map. In addition, assume that there exists a scalar η such that $R(\hat{X}_k, \hat{U}_k) = \eta R(X_k, U_k)$ with probability one, and that $dP(X_{k+1}|X_k, U_k) = dP(\hat{X}_{k+1}|\hat{X}_k, \hat{U}_k)$. Then for any x_0 we have that $V^*(\hat{x}_0) = \eta V^*(x_0)$ and $Q^*(\hat{x}_0, \hat{u}_0) = \eta Q^*(x_0, u_0)$, with $u_0 = \pi(x_0)$.■

As a concrete example, consider the linear system $X_{k+1} = AX_k + BU_k + W_k$, where $W_k \sim \mathcal{N}(0, \Sigma_w)$. As in the deterministic case, let M and N be two matrices such that $AM = MA$ and $MB = BN$. In addition, assume that $M^T \Sigma_w M = \Sigma_w$. It follows that $\Gamma(X, U, W) = (\Phi(X, U), W) = (MX, NU, W)$ is a strong stochastic symmetry. Indeed, by applying the transformation $\Phi(\cdot, \cdot)$ we get that $\hat{X}_{k+1} = A\hat{X}_k + B\hat{U}_k + \hat{W}_k$, where $\hat{W}_k = MW_k$. But since, W_k, and $\hat{W}_k$ are both zero mean noise with identical covariance matrices, we can safely replace $\hat{W}_k$ by W_k. Let $R(X_k, U_k) = X_k^T \mathcal{Q} X_k + U_k^T \mathcal{R} U_k$. Then if in addition we have $M^T \mathcal{Q} M = \eta \mathcal{Q}$ and $N^T \mathcal{Q} N = \eta \mathcal{R}$, than all the assumptions of Proposition 3.2 are satisfied, and therefore $V^*(\hat{x}_0 = \eta V^*(x_0)$ for some initial state x_0.

IV. Illustrative examples

In this section we apply the modified version of the Q-learning algorithm for a cart-pole system under a full state observation assumption. The cart-pole is described by a set of nonlinear equations:

$$(M+m)\ddot{x} + b\dot{x} - ml\ddot{\theta}\cos\theta + ml\dot{\theta}^2 \sin\theta = F \tag{7}$$

$$(I+ml^2)\ddot{\theta} - mgl\sin\theta - ml\ddot{x}\cos\theta = 0, \tag{8}$$

where $(x, \dot{x}, \theta, \dot{\theta})$ is the state vector, u is the force applied to the cart, and the parameters of the system are described in Table I (the parameter values listed there are not available to the learning algorithm). By denoting $\mathbf{x} = (x, \dot{x}, \theta, \dot{\theta})$, (7)-(8) can be compactly represented as $\dot{\mathbf{x}} = f(\mathbf{x}, F)$.

We use a first order approximation to obtain discrete-time versions of the system dynamics, that is, $\mathbf{x}_{k+1} = \mathbf{x}_k + hf(\mathbf{x}_k, F_k)$, which for small enough h, can provide a good approximation of the continuous-time dynamics. More importantly, if $\Gamma(\mathbf{x}, F) = (\hat{\mathbf{x}}, \hat{F})$ is a symmetry map for the continuous-time dynamics, then the same is true for the discrete-time approximation, that is, $\hat{\mathbf{x}}_{k+1} = \hat{\mathbf{x}}_k + hf(\hat{\mathbf{x}}_k, \hat{F}_k)$, where $(\hat{\mathbf{x}}_k, \hat{F}_k) = \Gamma(\mathbf{x}_k, F_k)$. For the purpose of learning, we discretize the state and action/input space. Let s_k, a_k denote the spatially discretized state vector and the action, respectively, where $s_k \in \mathcal{S}$, where $\mathcal{S} = \mathcal{S}_1 \times \mathcal{S}_2 \times \mathcal{S}_3 \times \mathcal{S}_4$, with $\mathcal{S}_1$ = {-1,-0.5,-0.25,0,0.25,0.5,1}, $\mathcal{S}_2$ = {-1,1}, $\mathcal{S}_3$ = {-0.2094 -0.1571,-0.1047, -0.0524,0,0.0524,0.1047,0.1571,0.2094}, $\mathcal{S}_4$ ={-1,1}. We can regard s_k as a quantized version of $\mathbf{x}_k$ whose dynamics are given by projections of $\mathbf{x}_k$ on $\mathcal{S}$.

The control policy is determined using the SARSA [14] learning algorithm with a table based representation of the Q function. We recall that in SARSA the Q-value iteration is given by $Q_{k+1}(s_k, a_k) = Q_k(s_k, a_k) + \alpha\left[r_{k+1} + \gamma Q_k(s_{k+1}, a_{k+1}) - Q_t(s_k, a_k)\right]$, where s_{k+1} is the projection of $\mathbf{x}_{k+1}$ on $\mathcal{S}$, with $\mathbf{x}_{k+1} = f(\mathbf{x}_k, a_k)$. The policy used in the iteration is an ϵ-greedy policy described in what follows. Let ε be a positive small scalar, and ν be a positive scalar uniformly drawn from the interval $[0, 1]$. The control policy at iteration k, as a function of s_{k+1} is defined as

$$a_{k+1} = \begin{cases} \min_a Q_k(s_{k+1}, a) & \nu > \varepsilon^k \\ \text{uniformly chosen from } \mathcal{A} & \nu \leq \varepsilon^k \end{cases}$$

The cost function used in SARSA is $c(x, \dot{x}, \theta, \dot{\theta}) = c_1 - c_2\theta^2 - c_3|x| - c_4|\dot{\theta}|$, where c_1, c_2, c_3, c_4 are positive constants. Note that the cost function is symmetric. It can be easily checked that the map $\Gamma(\mathbf{x}, u) = (-\mathbf{x}, -u)$ is a discrete symmetry which implies that $Q(s, a) = Q(-s, -a)$, due to the symmetry of the reward function. Hence the modified SARSA algorithm adds an additional update at each iteration, namely $Q_{k+1}(-s_k, -a_k) = Q_{k+1}(s_k, a_k)$.

We consider three discretized action spaces, as shown in Table II. For each action space, we executed 100 trials of the SARSA algorithm and its modified version, where each trail included 300 episodes, each episode having 1000 steps. An episode is declared successful if $x_k \in [-4, 4]$ and $\theta_k \in [-\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}]$ over the entire 1000 steps, otherwise it is declared a failure. As convergence metric, we use the average number of failures over the 100 trails. The simulations results are shown in Figures 3-4. Each figure has two graphs: the first graph shows the average number of failures corresponding to the three action space choices. The second graph shows the average reward corresponding to the three action space choices. We note a significant difference in the number of failures both in mean and standard deviation. The simulation results for the SARSA algorithm and its modified version are shown in Tables III-IV. We observe that in all cases, the modified SARSA performed better, in both convergence speed and reward function. In addition to the average values, we also plot the variances of the number of failures and rewards. Due to the random nature of the control policy, we get different values for the number of failures and rewards over the 100 trials.

Symbol	Explanation	Value
M	mass of the cart	$1.0Kg$
m	mass of the pendulum	$0.1Kg$
b	coefficient of friction for cart	$0.1N.s/m$
l	length to pendulum center of mass	$0.3m$
I	mass moment of inertia of the pendulum	$0.006kg.m^2$
g	gravitational acceleration	$9.81m/s^2$

TABLE I: Parameters of the cart-pole system

action space 1	action space 2	action space 3
$\{-1, 1\}$	$\{-1, 0, 1\}$	$\{-1, -0.5, 0.5, 1\}$

TABLE II: Action spaces

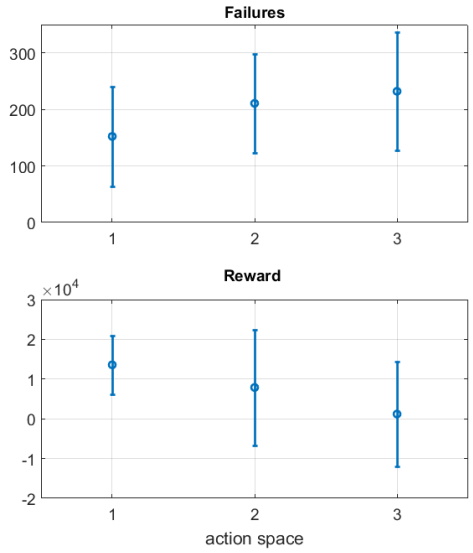

Fig. 3: SARSA algorithm: $c_1 = 10$, c_2 =1e3, $c_3 = 5$, $c_4 = 10$

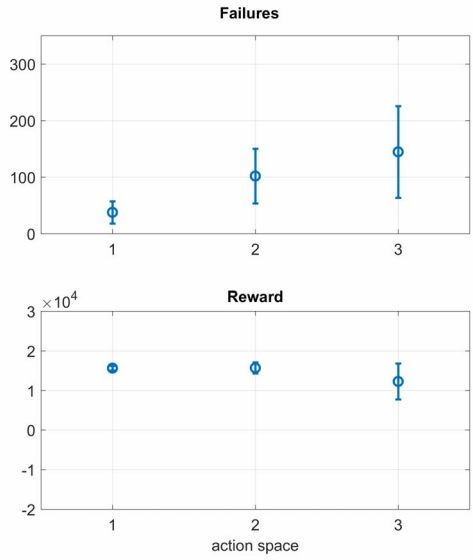

Fig. 4: Modified SARSA algorithm: $c_1 = 10$, c_2 =1e3, $c_3 = 5$, $c_4 = 10$

	act space 1	act space 2	act space 3
Avg # of failures	152.3(±88.17)	210.7(±87.28)	232(±104.1)
Avg reward	1.359e4(±7432)	7889(±1.453e4)	1202(±1.316e4)

TABLE III: SARSA Algorithm

	act space 1	act space 2	act space 3
Avg # failures	37 (±19.5)	101.8(±48.52)	144.6(±80.93)
Avg reward	1.562e4(±552.3)	1.567e4(±1399)	1.226e4(±4526)

TABLE IV: Modified SARSA Algorithm

V. Conclusions

We discussed an approach for enriching training data sets based on geometric properties of dynamical systems. Checking for such properties often requires only qualitative knowledge of the system behavior, which was our working assumption. In particular, we showed how new system trajectories can be generated by applying symmetry maps on solutions of ODEs describing the behavior of a physical system, and how the augmented data set can be used for learning. Specifically, the Q-learning algorithm was modified to accommodate symmetry-based trajectories. As future work, we plan to further explore how regularities of dynamical systems can be used to improve learning, especially in cases of small training data sets.

References

[1] Francesco C. De Vecchi, Paola Morando, and Stefania Ugolini. Symmetries of stochastic differential equations: A geometric approach. *Journal of Mathematical Physics*, 57(6), 2016.

[2] Giuseppe Gaeta. Symmetry of stochastic equations. *Proceedings of Institute of Mathematics of NAS*, 50(1):98–109, 2004.

[3] P.E. Hydon. *Symmetry Methods for Differential Equations: A Beginner's Guide*. Cambridge Texts in Applied Mathematics. Cambridge University Press, 2000.

[4] Roman Kozlov. On symmetries of stochastic differential equations. *Communications in Nonlinear Science and Numerical Simulation*, 17(12):4947 – 4951, 2012.

[5] Roman Kozlov. On symmetries of the fokker–planck equation. *Journal of Engineering Mathematics*, 82(1):39–57, 2013.

[6] Benjamin Kuipers. Qualitative simulation: Then and now, 1993.

[7] Benjamin Kuipers. Reasoning with qualitative models. *Artificial Intelligence*, 59:125–132, 1993.

[8] I. Matei, A. Ganguli, T. Honda, and J. de Kleer. The case for a hybrid approach to diagnosis: A railway switch. In *Proceedings of the 26th International Workshop on Principles of Diagnosis (DX-2015)*, pages 225–232, Aug 2015.

[9] Francisco S. Melo, Sean P. Meyn, and M. Isabel Ribeiro. An analysis of reinforcement learning with function approximation. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 664–671, 2008.

[10] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015.

[11] Carlos H. C. Ribeiro and Csaba Szepesvri. Q-learning combined with spreading: Convergence and results. In *Proceedings of ISRF-IEE International Conference: Intelligent and Cognitive Systems, Neural Networks Symposium*, pages 32–36, 1996.

[12] Richard S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the SevenLh International Conference on Machine Learning*, pages 216–224, 1990.

[13] Richard S. Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in Neural Information Processing Systems 8*, pages 1038–1044. MIT Press, 1996.

[14] R.S. Sutton and A.G. Barto. *Reinforcement learning: An introduction*, volume 116. Cambridge Univ Press, 1998.

[15] Csaba Szepesvári. Algorithms for reinforcement learningn. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 4(1):1–103, January 2010.

[16] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3):279–292, 1992.

[17] Brian C. Williams and Johan de Kleer. Qualitative reasoning about physical systems: A return to roots. *Artificial Intelligence*, 51(1):1 – 9, 1991.