Annual Review of Control, Robotics, and Autonomous Systems

A Tour of Reinforcement Learning: The View from Continuous Control

Benjamin Recht

Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, California 94720, USA; email: brecht@berkeley.edu

Annu. Rev. Control Robot. Auton. Syst. 2019. 2:253–79

First published as a Review in Advance on December 14, 2018

The *Annual Review of Control, Robotics, and Autonomous Systems* is online at control.annualreviews.org

https://doi.org/10.1146/annurev-control-053018-023825

Keywords

reinforcement learning, control theory, machine learning, optimization

Abstract

This article surveys reinforcement learning from the perspective of optimization and control, with a focus on continuous control applications. It reviews the general formulation, terminology, and typical experimental implementations of reinforcement learning as well as competing solution paradigms. In order to compare the relative merits of various techniques, it presents a case study of the linear quadratic regulator (LQR) with unknown dynamics, perhaps the simplest and best-studied problem in optimal control. It also describes how merging techniques from learning theory and control can provide nonasymptotic characterizations of LQR performance and shows that these characterizations tend to match experimental behavior. In turn, when revisiting more complex applications, many of the observed phenomena in LQR persist. In particular, theory and experiment demonstrate the role and importance of models and the cost of generality in reinforcement learning algorithms. The article concludes with a discussion of some of the challenges in designing learning systems that safely and reliably interact with complex and uncertain environments and how tools from reinforcement learning and control might be combined to approach these challenges.

1. INTRODUCTION

Reinforcement learning (RL) is the subfield of machine learning that studies how to use past data to enhance the future manipulation of a dynamical system. A control engineer might be puzzled by such a definition and interject that this is precisely the scope of control theory. That the RL and control communities remain practically disjoint has led to the codevelopment of vastly different approaches to the same problems. However, it should be impossible for a control engineer not to be impressed by the recent successes of the RL community, such as solving Go (1).

Indeed, given this dramatic recent progress in RL, a tremendous opportunity lies in deploying its data-driven systems in more demanding interactive tasks, including self-driving vehicles, distributed sensor networks, and agile robotic systems. For RL to expand into such technologies, however, the methods must be both safe and reliable—the failure of such systems can have severe societal and economic consequences, including the loss of human life. How can we guarantee that our new data-driven automated systems are robust? These types of reliability concerns are at the core of control engineering, and RL practitioners might be able to make their methods robust by applying appropriate control tools for engineering systems to match prescribed safety guarantees.

This survey aims to provide a language for the control and reinforcement learning communities to begin communicating, highlighting what each can learn from the other. Control is the theory of designing complex actions from well-specified models, while reinforcement learning often makes intricate, model-free predictions from data alone. Yet both RL and control aim to design systems that use richly structured perception, perform planning and control that adequately adapt to environmental changes, and exploit safeguards when surprised by a new scenario. Understanding how to properly analyze, predict, and certify such systems requires insights from current machine learning practice and from the applied mathematics of optimization, statistics, and control theory. With a focus on problems in continuous control, I will try to disentangle the similarities and differences of methods of the complementary perspectives and present a set of challenging problems whose solution will require significant input from both sets of practitioners.

I focus first on casting RL problems in an optimization framework, establishing the sorts of methodological tools brought to bear in contemporary RL. I then lay out the main solution techniques of RL, including the dichotomy between the model-free and model-based methodologies. Next, I try to put RL and control techniques on the same footing through a case study of the linear quadratic regulator (LQR) with unknown dynamics. This baseline will illuminate the various trade-offs associated with techniques from RL and control. In particular, we will see that the so-called model-free methods popular in deep RL are considerably less effective in both theory and practice than simple model-based schemes when applied to LQR. Perhaps surprisingly, I also show cases where these observations continue to hold on more challenging nonlinear applications. I then argue that model-free and model-based perspectives can be unified, combining their relative merits. This leads to a concluding discussion of some of the challenges at the interface of control and learning that must be solved before we can build robust, safe learning systems that interact with an uncertain physical environment, which will surely require tools from both the machine learning and control communities.

2. WHAT IS REINFORCEMENT LEARNING?

Reinforcement learning is the study of how to use past data to enhance the future manipulation of a dynamical system. How does this differ from ordinary machine learning? The main view of this survey is of RL as optimal control when the dynamics are unknown. Our goal will be to find a sequence of inputs that drives a dynamical system to maximize some objective, beginning with minimal knowledge of how the system responds to inputs.

In the classic optimal control problem, we begin with a dynamical system governed by the difference equation $x_{t+1} = f_t(x_t, u_t, e_t)$, where x_t is the state of the system, u_t is the control action, and e_t is a random disturbance; f_t is the rule that maps the current state, control action, and disturbance at time t to a new state. Assume that at every time, we receive some reward $R(x_t, u_t)$ for our current x_t and u_t. The goal is to maximize this reward. In terms of mathematical optimization, we aim to solve the problem

$$\begin{aligned} &\text{maximize } \mathbb{E}_{e_t}\left[\sum_{t=0}^{N} R_t(x_t, u_t)\right] \\ &\text{subject to } x_{t+1} = f_t(x_t, u_t, e_t) \\ &\qquad (x_0 \text{ given}). \end{aligned} \tag{1}$$

That is, we aim to maximize the expected reward over N time steps with respect to the control sequence u_t, subject to the dynamics specified by the state-transition rule f_t. The expected value is over the disturbance and assumes that u_t is to be chosen having seen only the states x_0 through x_t and previous inputs u_0 through u_{t-1}. R_t is the reward gained at each time step and is determined by the state and control action. Note that x_t is not really a decision variable in the optimization problem; it is determined entirely by the previous state, control action, and disturbance. I will refer to a trajectory, τ_t, as a sequence of states and control actions generated by a dynamical system:

$$\tau_t = (u_1, \ldots, u_{t-1}, x_0, \ldots, x_t)\,. \tag{2}$$

Since the dynamics are stochastic, the optimal control problem typically allows a controller to observe the state before deciding upon the next action (2). This allows a controller to continually mitigate uncertainty through feedback. Hence, rather than optimizing over deterministic sequences of controls u_t, we instead optimize over policies. A control policy (or simply "a policy") is a function, π, that takes a trajectory from a dynamical system and outputs a new control action. Note that π has access only to previous states and control actions.

To slightly lower the notational burden, I will work with the time-invariant version of Problem 1, assuming that the dynamical update rule is constant over time and that the rewards for state–action pairs are also constant:

$$\begin{aligned} &\text{maximize } \mathbb{E}_{e_t}\left[\sum_{t=0}^{N} R(x_t, u_t)\right] \\ &\text{subject to } x_{t+1} = f(x_t, u_t, e_t),\ u_t = \pi_t(\tau_t) \\ &\qquad (x_0 \text{ given}). \end{aligned} \tag{3}$$

The policies π_t are the decision variables of the problem.

Let us now directly bring machine learning into the picture. What happens when we do not know the state-transition rule f? There are a variety of commonly occurring scenarios when we might lack such knowledge. We may have unknown relationships between control forces and torques in a mechanical system. Or we could have a considerably more complicated system, such as a massive data center with complex heat-transfer interactions between the servers and the cooling systems. Can we still solve Problem 3 well without a precise model of the dynamics? Some lines of work even assume that we do not know the reward function R, but for the purpose of this survey, it makes no difference whether R is known or unknown. The important point is that we cannot solve this optimization problem using standard optimization methods unless we know the dynamics. We must learn something about the dynamical system and subsequently choose the best policy based on our knowledge.

2.1. The Episodic Oracle Model

The main paradigm in contemporary RL is to play the following game. We decide on a policy π and horizon length L. We then pass this policy either to a simulation engine or to a real physical system, which returns a trajectory τ_L and a sequence of rewards $\{R(x_t, u_t)\}$. We want to find a policy that maximizes the reward with the fewest total number of samples computed by the oracle, and we are allowed to do whatever we would like with the previously observed trajectories and reward information when computing a new policy. If we were to run m queries with horizon length L, we would pay a total cost of mL. However, we are free to vary our horizon length for each experiment. This is our oracle model and is called episodic RL (see, for example, chapter 3 of Reference 3, chapter 2 of Reference 4, or Reference 5). We want the expected reward to be high for our derived policy, but we also need the number of oracle queries to be small.

This oracle model is considerably more complicated than those typically considered in oracle models for optimization (6). Each episode returns a complex feedback signal of states and rewards. What is the best way to tie this information together in order to improve performance? What is the best way to query and probe a system to achieve high-quality control with as few interventions as possible? Here, "best" is also not clearly defined. Do we decide an algorithm is best if it crosses some reward threshold in the fewest number of samples? Or is it best if it achieves the highest reward given a fixed budget of samples? Or maybe there is a middle ground? This oracle provides a rich and complex model for interacting with a system and brings with it considerably more complexity than exists in standard stochastic optimization settings. What is the most efficient way to use all of the collected data in order to improve future performance?

2.2. Connections to Supervised Learning

The predominant paradigm of machine learning is supervised learning or prediction. In prediction, the goal is to predict the variable y from a vector of features x such that, on new data, you are predicting y from x with high accuracy. This form of machine learning includes classification and regression as special cases. Most of the time, when experts use the term machine learning colloquially, they are referring to this sort of prediction. From this perspective, niche topics like semisupervised learning (7) and matrix completion (8) are prediction tasks as well.

By contrast, there are two special variables in RL: u and r. The goal now is to analyze the features x and then subsequently choose a policy that emits u so that r is large.[1] There are an endless number of problems where this formulation is applied (3, 9, 10), from online decision-making in games (1, 11–13) to engagement maximization on Internet platforms (14, 15). A key distinguishing aspect of RL is the control action u. Unlike in prediction, the practitioner can vary u, which has implications both for learning (e.g., designing experiments to learn about a given system) and for control (e.g., choosing inputs to maximize reward).

RL is clearly more challenging than supervised learning, but, at the same time, it can be considerably more valuable. RL provides a useful framework to conceptualize interaction in machine learning and promises to help mitigate changing distributions, gaming, adversarial behavior, and unexpected amplification. There is a precarious trade-off that must be carefully considered: RL demands interventions with the promise that these actions will directly lead to valuable returns, but the resulting complicated feedback loops are hard to study in theory, and failures can have catastrophic consequences.

[1]To achieve notational consistency, I am adopting the control-centric notation of denoting state–action pairs as (x, u) rather than (s, a), as is commonly used in RL.

3. STRATEGIES FOR SOLVING REINFORCEMENT LEARNING PROBLEMS

Let us now turn to a taxonomy of the varied algorithmic frameworks for RL, focused on solving Problem 3 when the state-transition function is unknown. Model-based RL fits a model to previously observed data and then uses this model in some fashion to approximate the solution to Problem 3. Model-free RL eschews the need for a system model, directly seeking a map from observations to actions.

The role of models in RL remains hotly debated. Model-free methods, as discussed below, aim to solve optimal control problems only by probing the system and improving strategies based on past rewards and states. Many researchers argue for algorithms that can innately learn to control without access to the complex details required to simulate a dynamical system. They argue that it is often easier to find a policy for a task than it is to fit a general-purpose model of the system dynamics (see, for example, the discussion in chapter 3 of Reference 16). Model-free methods are primarily divided into two approaches: policy search and approximate dynamic programming. Policy search directly searches for policies by using data from previous episodes in order to improve the reward. Approximate dynamic programming uses Bellman's principle of optimality to approximate Problem 3 using previously observed data.

Throughout, my aim will be to highlight the main conceptual ideas of different approaches and to avoid embroiling myself in a thorough discussion of the myriad of technical details required to make all of the statements crisply precise. What is important is that all of the approaches surveyed reduce to some sort of function fitting from noisy observations of the dynamical system, though performance can be drastically different depending on how you fit this function. In model-based RL, we fit a model of the state transitions to best match observed trajectories. In approximate dynamic programming, we estimate a function that best characterizes the cost to go for experimentally observed states. And in direct policy search, we attempt to find a policy that directly maximizes the optimal control problem using only input–output data. The main questions are which of these approaches makes the best use of samples and how quickly the derived policies converge to optimality.

3.1. Model-Based Reinforcement Learning

One of the simplest and perhaps most obvious strategies to solve Problem 3 is to estimate a predictive model for the dynamical process and then use it in a dynamic programming solution to the prescribed control problem. The estimated model is called the nominal model, and I will refer to control design that assumes the estimated model is true as nominal control. Nominal control serves as a useful baseline algorithm.

Estimation of dynamical systems is called system identification in the control community (17). System identification differs from conventional estimation because one needs to carefully choose the right inputs to excite various degrees of freedom and because dynamical outputs are correlated over time with the parameters we hope to estimate, the inputs we feed to the system, and the stochastic disturbances. Once data are collected, however, conventional machine learning tools can be used to find the system that best agrees with the data and can be applied to analyze the number of samples required to yield accurate models (18, 19).

Suppose we want to build a predictor of x_{t+1} from the trajectory history. A simple, classic strategy is to inject a random probing sequence u_t for control and then measure how the state responds. Up to stochastic noise, we should have that

$$x_{t+1} \approx \varphi(x_t, u_t)\,, \qquad 4.$$

where φ is some model aiming to approximate the true dynamics; φ might arise from a first-principles physical model or might be a nonparametric approximation by a neural network. The state-transition function can then be fit using supervised learning. For instance, a model can be fit by solving the least squares problem

$$\text{minimize}_{\varphi} \sum_{t=0}^{N-1} ||x_{t+1} - \varphi(x_t, u_t)||^2 .$$

Let $\hat{\varphi}$ denote the function fit to the collected data to model the dynamics. Let ω_t denote a random variable that we will use as a model for the noise process. With such a point estimate for the model, we might solve the optimal control problem

$$\begin{aligned} &\text{maximize } \mathbb{E}_{\omega_t}\left[\sum_{t=0}^{N} R(x_t, u_t)\right] \\ &\text{subject to } x_{t+1} = \hat{\varphi}(x_t, u_t) + \omega_t, \;\; u_t = \pi_t(\tau_t) . \end{aligned}$$

In this case, we are solving the wrong problem to get our control policies π_t. Not only is the model incorrect, but this formulation requires some plausible model of the noise process. But if $\hat{\varphi}$ and f are close, this approach might work well in practice.

3.2. Approximate Dynamic Programming

Approximate dynamic programming approaches the RL problem by directly approximating the optimal control cost and then solving the resulting approximation with techniques from dynamic programming. The dynamic programming solution to Problem 3 is based on the principle of optimality: If you have found an optimal control policy for a time horizon of length N, $\pi_1, \ldots, \pi_N$, and you want to know the optimal strategy starting at state x at time t, then you just have to take the optimal policy starting at time t, $\pi_t, \ldots, \pi_N$. Dynamic programming then lets us recursively find a control policy by starting at the final time and recursively solving for policies at earlier times.

Define the Q-function for Problem 3 to be the mapping

$$\mathcal{Q}(x, u) = \max \left\{ \mathbb{E}_{e_t}\left[\sum_{t=0}^{N} R(x_t, u_t)\right] \; : \; x_{t+1} = f(x_t, u_t, e_t), \; (x_0, u_0) = (x, u) \right\} . \qquad 5.$$

The Q-function determines the value of the optimal control problem that is attained when the first action is set to be u and the initial condition is x. Note that it then trivially follows that the optimal value of Problem 3 is $\max_u \mathcal{Q}(x_0, u)$, and the optimal policy is $\pi(x_0) = \arg\max_u \mathcal{Q}(x_0, u)$. If we had access to the Q-function, we would have everything we need to know to take the first step in the optimal control problem. We can use dynamic programming to compute this Q-function and the Q-function associated with every subsequent action. That is, we define the terminal Q-function to be

$$\mathcal{Q}_N(x, u) = R(x, u)$$

and then define recursively

$$\mathcal{Q}_k(x, u) = R(x, u) + \mathbb{E}_e \left\{ \max_{u'} \mathcal{Q}_{k+1}[f(x, u, e), u'] \right\} . \qquad 6.$$

This is the dynamic programing algorithm in a nutshell: We can recursively define the Q-functions by passing backward in time, and then compute the optimal controls from any starting x_0 by applying the policy that maximizes the right-hand side of Equation 6 at each time step. Equation 6 is known as Bellman's equation. Note that for all time, the optimal policy is $u_k = \arg\max_u \mathcal{Q}_k(x_k, u)$ and depends only on the current state.

Approximate dynamic programming methods typically try to compute these action-value functions from data. They do so by assuming that the Q-function is stationary [i.e., $\mathcal{Q}_k(x, u) = \mathcal{Q}(x, u)$ for all k and some function $\mathcal{Q}$]. Such stationarity indeed arises assuming the time horizon is infinite. Consider the limit:

$$\begin{aligned} &\text{maximize } \lim_{N\to\infty} \mathbb{E}_{e_t}\left[\frac{1}{N}\sum_{t=0}^{N} R(x_t, u_t)\right] \\ &\text{subject to } x_{t+1} = f(x_t, u_t, e_t),\ u_t = \pi_t(\tau_t) \\ &\qquad (x_0 \text{ given}). \end{aligned} \tag{7.}$$

And we define the Q-function $\mathcal{Q}(x_0, u_0)$ to be the average reward accrued running from state x_0 with initial action u_0. Unfortunately, Equation 7 is not directly amenable to dynamic programming without introducing further technicalities. For mathematical convenience and to connect to common practice in RL, it is useful to instead consider the discounted reward problem

$$\begin{aligned} &\text{maximize } (1-\gamma)\mathbb{E}_{e_t}\left[\sum_{t=0}^{\infty} \gamma^t R(x_t, u_t)\right] \\ &\text{subject to } x_{t+1} = f(x_t, u_t, e_t),\ u_t = \pi_t(\tau_t) \\ &\qquad (x_0 \text{ given}), \end{aligned} \tag{8.}$$

where γ is a scalar in (0, 1) called the discount factor. For γ close to 1, the discounted reward is approximately equal to the average reward (16). The discounted cost has particularly clean optimality conditions that make it amenable to estimation. If we define $\mathcal{Q}_\gamma(x, u)$ to be the Q-function obtained from solving Problem 8 with initial condition x, then we have a discounted version of dynamic programming, now with the same Q-functions on the left- and right-hand sides:

$$\mathcal{Q}_\gamma(x, u) = R(x, u) + \gamma \mathbb{E}_e \left\{ \max_{u'} \mathcal{Q}_\gamma [f(x, u, e), u'] \right\}.$$

The optimal policy is now for all times to let

$$u_t = \arg\max_u \mathcal{Q}_\gamma(x_t, u). \tag{9.}$$

This is a remarkably simple formula, which is part of what makes Q-learning methods so attractive.

We can try to solve for the Q-function using stochastic approximation. If we draw a sample trajectory using the policy given by Equation 9, then we should have (approximately and in expectation)

$$\mathcal{Q}_\gamma(x_k, u_k) \approx R(x_k, u_k) + \gamma \max_{u'} \mathcal{Q}_\gamma(x_{k+1}, u').$$

Thus, beginning with some initial guess $\mathcal{Q}_\gamma^{(\text{old})}$ for the Q-function, we can update

$$Q_\gamma^{(\text{new})}(x_k, u_k) = (1-\eta) Q_\gamma^{(\text{old})}(x_k, u_k) + \eta \left[R(x_k, u_k) + \gamma \max_{u'} Q_\gamma^{(\text{old})}(x_{k+1}, u') \right], \tag{10.}$$

where η is a step size or learning rate. Equation 10 forms the basis of Q-learning algorithms (20, 21).

Surveying approximate dynamic programming using only Q-functions is somewhat unorthodox. Most introductions to approximate dynamic programming instead focus on value functions, where

$$V(x) = \max_u Q(x, u) .$$

Methods for estimating value functions are also widely used in RL and developed through the perspective of estimation and stochastic approximation. In particular, temporal difference algorithms are derived from the value-function-centric perspective (22–26).

Note that in all cases here, though we have switched away from models, there is no free lunch. We are still estimating functions, and we need to assume that the functions have some reasonable structure, or we cannot learn them. Choosing a parameterization of the Q-function is a modeling assumption. The term model free, when casually claimed in RL research, almost always means that there is no model of the state-transition function. However, this does not mean that modeling is not heavily built into the assumptions of model-free RL algorithms. Moreover, for continuous control problems, these methods appear to make an inefficient use of samples. Suppose the internal state of the system is of dimension d. When modeling the state-transition function, Equation 4 provides d equations per time step. By contrast, we are using only one equation per time step in approximate dynamic programming. Such inefficiency is certainly seen in practice below. Also troubling is the fact that we had to introduce the discount factor in order to get a simple Bellman equation. One can avoid discount factors, but this requires considerably more sophisticated analysis. Large discount factors do in practice lead to brittle methods, and the discount becomes a hyperparameter that must be tuned to stabilize performance. We illustrate below when and how these issues arise in practice in control.

3.3. Direct Policy Search

The most ambitious form of control without models attempts to directly learn a policy function from episodic experiences without ever building a model or appealing to the Bellman equation. From the oracle perspective, these policy-driven methods turn the problem of RL into derivative-free optimization.

3.3.1. A generic algorithm for sampling to optimize. Let us begin with a review of a general paradigm for leveraging random sampling to solve optimization problems. Consider the general unconstrained optimization problem

$$\text{maximize}_{z \in \mathbb{R}^d} \; R(z) . \qquad 11.$$

Any optimization problem like this is equivalent to an optimization over probability distributions on z:

$$\text{maximize}_{p(z)} \; \mathbb{E}_p[R(z)] .$$

If $z_\star$ is the optimal solution, then we get the same value if we put a δ-function around $z_\star$. Moreover, if p is a probability distribution, it is clear that the expected value of the reward function can never be larger than the maximal reward achievable by a fixed z. So we can either optimize over z or optimize over distributions over z.

Since optimizing over the space of all probability densities is intractable, we must restrict the class of densities over which we optimize. For example, we can consider a family parameterized by a parameter vector ϑ: $p(u;\,\vartheta)$ and attempt to optimize

$$\text{maximize}_{\vartheta}\ \mathbb{E}_{p(z;\,\vartheta)}[R(z)]. \tag{12.}$$

If this family of distributions contains all of the δ-functions, then the optimal value will coincide with the nonrandom optimization problem. But if the family does not contain δ-functions, the resulting optimization problem only provides a lower bound on the optimal value no matter how good of a probability distribution we find.

That said, this reparameterization provides a powerful and general algorithmic framework for optimization. In particular, we can compute the derivative of $J(\vartheta) := \mathbb{E}_{p(z;\,\vartheta)}[R(z)]$ using the following calculation (called the log-likelihood trick):

$$\begin{aligned}
\nabla_{\vartheta} J(\vartheta) &= \int R(z)\nabla_{\vartheta}\, p(z;\,\vartheta)\mathrm{d}z \\
&= \int R(z)\left[\frac{\nabla_{\vartheta}\, p(z;\,\vartheta)}{p(z;\,\vartheta)}\right] p(z;\,\vartheta)\mathrm{d}z \\
&= \int \left[R(z)\nabla_{\vartheta} \log p(z;\,\vartheta)\right] p(z;\,\vartheta)\mathrm{d}z \\
&= \mathbb{E}_{p(z;\,\vartheta)}\left[R(z)\nabla_{\vartheta} \log p(z;\,\vartheta)\right].
\end{aligned}$$

This derivation reveals that the gradient of J with respect to ϑ is the expected value of the function

$$G(z, \vartheta) = R(z)\nabla_{\vartheta} \log p(z;\,\vartheta). \tag{13.}$$

Hence, if we sample z from the distribution defined by $p(z;\,\vartheta)$, we can compute $G(z, \vartheta)$ and will have an unbiased estimate of the gradient of J. We can follow this direction and will be running stochastic gradient descent on J, defining Algorithm 1.

Algorithm 1 (REINFORCE).

```
**Hyperparameters:** step sizes $\alpha_j > 0$.
**Initialize:** $\vartheta_0$ and $k = 0$.
**while** ending condition not satisfied **do**
    Sample $z_k \sim p(z;\, \vartheta_k)$.
    Set $\vartheta_{k+1} = \vartheta_k + \alpha_k R(z_k)\nabla_{\vartheta} \log p(z_k;\, \vartheta_k)$.
    $k \leftarrow k + 1$.
**end while**
```

Algorithm 1 is typically called REINFORCE (27), and its main appeal is that it is trivial to implement. If you can efficiently sample from $p(z;\,\vartheta)$, you can run this algorithm on essentially any problem. But such generality comes with a significant cost. The algorithm operates on stochastic gradients of the sampling distribution, but the function we cared about optimizing (R) is accessed only through function evaluations. Direct search methods that use the log-likelihood trick are necessarily derivative-free optimization methods and, in turn, are necessarily less effective than methods that compute actual gradients, especially when the function evaluations are noisy (28). Another significant concern is that the choice of distribution can lead to very high variance in the stochastic gradients. Such high variance in turn implies that many samples need to be drawn to find a stationary point.

That said, the ease of implementation should not be readily discounted. Direct search methods are trivial to implement, and oftentimes reasonable results can be achieved with considerably less effort than custom solvers tailored to the structure of the optimization problem. There are two primary ways that this sort of stochastic search arises in RL: policy gradient and pure random search.

3.3.2. Policy gradient. As seen from Bellman's equation, the optimal policy for Problem 3 is always deterministic. Nonetheless, the main idea behind policy gradient is to use probabilistic policies. Probabilistic policies are optimal for other optimization-based control problems, such as control of partially observed Markov decision processes (29, 30) or in zero-sum games. Hence, exploring their value for the RL problems studied in this survey does not appear too outlandish at first glance.

We fix our attention on parametric, randomized policies such that u_t is sampled from a distribution $p(u|\tau_t;\vartheta)$ that is a function only of the currently observed trajectory and a parameter vector ϑ. A probabilistic policy induces a probability distribution over trajectories:

$$p(\tau;\vartheta) = \prod_{t=0}^{L-1} p(x_{t+1}|x_t, u_t)p(u_t|\tau_t;\vartheta). \qquad 14.$$

Moreover, we can overload the notation and define the reward of a trajectory to be

$$R(\tau) = \sum_{t=0}^{N} R_t(x_t, u_t).$$

Then our optimization problem for RL tidily takes the form of Equation 12. Policy gradient thus proceeds by sampling a trajectory using the probabilistic policy with parameters ϑ_k and then updating using REINFORCE.

Using the log-likelihood trick and Equation 14, it is straightforward to verify that the gradient of J with respect to ϑ is not an explicit function of the underlying dynamics. However, at this point this should not be surprising: By shifting to distributions over policies, we push the burden of optimization onto the sampling procedure.

3.3.3. Pure random search. An older and more widely applied method to solve Problem 11 is to directly perturb the current decision variable z by random noise and then update the model based on the received reward at this perturbed value. That is, we apply Algorithm 1 with sampling distribution $p(z;\vartheta) = p_0(z-\vartheta)$ for some distribution p_0. The simplest examples for p_0 would be the uniform distribution on a sphere or a normal distribution. Perhaps less surprisingly here, REINFORCE can again be run without any knowledge of the underlying dynamics. Note that in this case, the REINFORCE algorithm has a simple interpretation in terms of gradient approximation. Indeed, REINFORCE is equivalent to approximate gradient ascent of R

$$\vartheta_{t+1} = \vartheta_t + \alpha g_\sigma(\vartheta_k)$$

with the gradient approximation

$$g_\sigma(\vartheta) = \frac{R(\vartheta+\sigma\epsilon) - R(\vartheta-\sigma\epsilon)}{2\sigma}\epsilon\,.$$

This update says to compute a finite-difference approximation to the gradient along the direction ϵ and move along the gradient. One can reduce the variance of such a finite-difference estimate by sampling along multiple random directions and averaging:

$$g_\sigma^{(m)}(\vartheta) = \frac{1}{m} \sum_{i=1}^{m} \frac{R(\vartheta + \sigma\epsilon_i) - R(\vartheta - \sigma\epsilon_i)}{2\sigma} \epsilon_i \, .$$

This is akin to approximating the gradient in the random subspace spanned by the ϵ_i.

This particular algorithm and its generalizations go by many different names. Probably the earliest proposal for this method was made by Rastrigin (31). In an unexpected historical surprise, Rastrigin initially developed this method to solve RL problems! His main motivating example was an inverted pendulum. A rigorous analysis using contemporary techniques was provided by Nesterov & Spokoiny (32). Random search was also discovered by the evolutionary algorithms community, where it is called the (μ, λ)-evolution strategy (33, 34). Random search has also been studied in the context of stochastic approximation (35) and bandits (36, 37). Algorithms that are invented independently by four different communities probably have something good going for them.

The random search method is considerably simpler than the policy gradient algorithm, but it uses much less structure from the problem as well. Since RL problems tend to be nonconvex, it is not clear which of these approaches is better unless we focus on specific instances. In light of this, in the next section we turn to a set of instances where we may be able to glean more insights about the relative merits of all of the approaches to RL covered in this section.

3.4. Deep Reinforcement Learning

Note that in this section I have spent no time discussing deep reinforcement learning. That is because there is nothing conceptually different other than the use of neural networks for function approximation. That is, if you want to take any of the described methods and make them deep, you simply need to add a neural network. In model-based RL, φ is parameterized as a neural network; in approximate dynamic programming, the Q-functions or value functions are assumed to be well approximated by neural networks; and in policy search, the policies are set to be neural networks. The algorithmic concepts themselves do not change. However, convergence analysis certainly will change, and algorithms like Q-learning might not even converge. The classic text *Neuro-Dynamic Programming* by Bertsekas & Tsitsiklis (9) discusses the adaptations needed to admit function approximation. By eliminating the complicating variable of function approximation, we can get better insights into the relative merits of these methods, especially when focusing on a simple set of instances of optimal control, namely, the linear quadratic regulator.

4. SIMPLIFYING THEME: THE LINEAR QUADRATIC REGULATOR

With this varied list of approaches to reinforcement learning, it is difficult from afar to judge which method fares better on which problems. It is likely best to start simple and small and find the simplest nontrivial problem that can assist in distinguishing the various approaches to control. Though simple models are not the end of the story in analysis, it tends to be the case that if a complicated method fails to perform on a simple problem, then there is a flaw in the method.

I would argue that in control, the simplest nontrivial class of instances of optimal control is those with convex quadratic rewards and linear dynamics—that is, the problem of the linear quadratic regulator:

$$\begin{aligned} &\text{minimize } \mathbb{E}_{e_t}\left(\frac{1}{2}\sum_{t=0}^{N} x_t^T Q x_t + u_t^T R u_t + \frac{1}{2}x_{N+1}^T S x_{N+1}\right), \\ &\text{subject to } x_{t+1} = Ax_t + Bu_t + e_t,\ u_t = \pi_t(\tau_t) \\ &\qquad\qquad (x_0 \text{ given}). \end{aligned} \qquad 15.$$

Here, Q, R, and S are positive semidefinite matrices. Note that we have switched to minimization from maximization, as is conventional in optimal control. The state transitions are governed by a linear update rule with A and B appropriately sized matrices.

A few words are in order to defend this baseline as instructive for general problems in continuous control and RL. Though linear dynamics are somewhat restrictive, many systems are linear over the range we would like them to operate. Indeed, enormous engineering effort goes into designing systems so that their responses are as close to linear as possible. From an optimization perspective, linear dynamics are the only class where we are guaranteed that our constraint set is convex, which is another appealing feature for analysis.

What about cost functions? Whereas dynamics are typically handed to the engineer, cost functions are completely at their discretion. Designing and refining cost functions are part of optimal control design, and different characteristics can be extracted by iteratively refining cost functions to meet specifications. This is no different in machine learning, where, for example, combinatorial losses in classification are replaced with smooth losses like logistic or squared loss. Designing cost functions is a major challenge and tends to be an art form in engineering. But since we are designing our cost functions, we should focus our attention on costs that are easier to solve. Quadratic cost is particularly attractive not only because it is convex, but also because of how it interacts with noise. The cost of the stochastic problem is equal to that of the noiseless problem plus a constant that is independent of the choice of u_t. The noise will degrade the achievable cost, but it will not affect how control actions are chosen.

Note that when the parameters of the dynamical system are known, the standard LQR problem admits an elegant dynamic programming solution (38). The control action is a linear function of the state

$$u_t = -K_t x_t$$

for some matrix K_t that can be computed via a simple linear algebraic recursion with only knowledge of A, B, Q, and R.

In the limit as the time horizon tends to infinity, the optimal control policy is static, linear state feedback:

$$u_t = -Kx_t,$$

where K is a fixed matrix defined by

$$K = (R + B^T MB)^{-1} B^T MA$$

and M is a solution to the discrete algebraic Riccati equation

$$M = Q + A^T MA - (A^T MB)(R + B^T MB)^{-1}(B^T MA)\,. \qquad 16.$$

That is, for LQR on an infinite time horizon, $\pi_t(x_t) = -Kx_t$. Here, M is the unique solution of the Riccati equation where all of the eigenvalues of $A - BK$ have magnitude less than 1. Finding this specific solution is relatively easy using standard linear algebraic techniques (38).

There are a variety of ways to derive these formulae. In particular, one can use dynamic programming as in Section 3.2. In this case, one can check that the Q-function on a finite time horizon satisfies a recursion

$$\mathcal{Q}_k(x, u) = x^T Qx + u^T Ru + (Ax + Bu)^T M_{k+1}(Ax + Bu) + c_k$$

for some positive definite matrix M_{k+1}. The limits of these matrices are the solution of Equation 16.

Though LQR cannot capture every interesting optimal control problem, it has many of the salient features of the generic optimal control problem. Dynamic programming recursion lets us compute the control actions efficiently, and for long time horizons, a static policy is nearly optimal.

Now the main question to consider in the context of RL: What happens when we do not know A and B? What is the right way to interact with the dynamical system in order to quickly and efficiently get it under control? Let us now dive into the different styles of RL and connect them to ideas in control, using LQR as a guiding baseline.

4.1. The Sample Complexity of Model-Based Reinforcement Learning for the Linear Quadratic Regulator

For LQR, maximum likelihood estimation of a nominal model is a least squares problem:

$$\text{minimize}_{A,B} \sum_{t=0}^{N-1} ||x_{t+1} - Ax_t - Bu_t||^2 .$$

How well do these model estimates work for the LQR problem? Suppose we treat the estimates as true and use them to compute a state feedback control from a Riccati equation. While we might expect this to work well in practice, how can we verify the performance? As a simple case, suppose that the true dynamics are slightly unstable, so that A has at least one eigenvalue of magnitude larger than 1. It is fully possible that the least squares estimate of one of the diagonal entries will be less than 1, and, consequently, the optimal control strategy using the estimate will fail to account for the poorly estimated unstable mode. How can we include the knowledge that our model is just an estimate and not accurate with a small sample count? One possible solution is to use tools from robust control to mitigate this uncertainty.

My collaborators and I have been considering an approach to merge robust control and high-dimensional statistics dubbed coarse-ID control. The general framework consists of the following three steps:

1. Use supervised learning to learn a coarse model of the dynamical system to be controlled. I will refer to the system estimate as the nominal system.
2. Using either prior knowledge or statistical tools like the bootstrap, build probabilistic guarantees about the distance between the nominal system and the true, unknown dynamics.
3. Solve a robust optimization problem that optimizes control of the nominal system while penalizing signals with respect to the estimated uncertainty, ensuring stable, robust execution.

As long as the true system behavior lies in the estimated uncertainty set, we are guaranteed to find a performant controller. The key here is that we are using machine learning to identify not only the plant to be controlled, but also the uncertainty. Indeed, the main advances in the past two decades of estimation theory consist of providing reasonable estimates of such uncertainty sets with guaranteed bounds on their errors as a function of the number of observed samples. Taking these new tools and merging them with old and new ideas from robust control allows us to bound the end-to-end performance of a controller in terms of the number of observations.

The coarse-ID procedure is well illustrated through the case study of LQR (39). We can guarantee the accuracy of the least squares estimates for A and B using novel probabilistic analysis (40). With the estimate of model error in hand, one can pose a robust variant of the standard LQR optimal control problem that computes a robustly stabilizing controller seeking to minimize the worst-case performance of the system given the (high-probability) norm bounds on our modeling errors.

To design a good control policy, we here turn to state-of-the-art tools from robust control. We leverage the recently developed system-level synthesis framework (41, 42) to solve this robust optimization problem. System-level synthesis lifts the system description into a higher-dimensional space that enables efficient search for controllers. The proposed approach provides nonasymptotic bounds that guarantee finite performance on the infinite time horizon and quantitatively bound the gap between the computed solution and the true optimal controller.

Suppose in LQR that we have a state dimension d and a control dimension p. Denote the minimum cost achievable by the optimal controller as J_*. Our analysis guarantees that, after a observing a trajectory of length T, we can design a controller that will have infinite-time-horizon cost $\hat{J}$ with

$$\frac{\hat{J} - J_\star}{J_\star} = \tilde{O}\left(\sqrt{\frac{d+p}{T}}\right) .$$

Here, the notation $\tilde{O}(\cdot)$ suppresses logarithmic factors and instance-dependent constants. In particular, we can guarantee that we stabilize the system after seeing only a finite amount of data.

Where coarse-ID control differs from nominal control is that it explicitly accounts for the uncertainty in the least squares point estimate. By appending this uncertainty to the original LQR optimization problem, we can circumvent the need to study perturbations of Riccati equations. Moreover, since the approach is optimization based, it can be readily applied to other optimal control problems beyond the LQR baseline.

4.2. The Sample Complexity of Model-Free Reinforcement Learning for the Linear Quadratic Regulator

Since we know that the Q-function for LQR is quadratic, we can try to estimate it by dynamic programming. Such a method was probably first proposed by Bradtke et al. (43). More recently, Tu & Recht (44) showed that the least squares temporal differencing algorithm, first developed by Bradtke & Barto (24), could estimate the value function of LQR to low error with $\tilde{O}\left(\sqrt{d^2/T}\right)$ samples. This estimator can then be combined with a method to improve the estimated policy over time.

Note that the bound on the efficiency of the estimator here is worse than the error obtained for estimating the model of the dynamical system. While comparing worst-case upper bounds is certainly not valid, it is suggestive that, as mentioned above, temporal differencing methods use only one defining equation per time step, whereas model estimation uses d equations per time step. So while the conventional wisdom suggests that estimating Q-functions for specific tasks should be simpler than estimating models, the current methods appear to be less efficient with aggregated data than system identification methods are.

With regard to direct search methods, we can already see variance issues enter the picture even for small LQR instances. Consider the most trivial example of LQR:

$$R(u) = ||u||^2.$$

Let $p(u; \vartheta)$ be a multivariate Gaussian with mean ϑ and variance $\sigma^2 I$. Then

$$\mathbb{E}_{p(u;\vartheta)}[R(u)] = \|\vartheta\|^2 + \sigma^2 d.$$

Obviously, the best thing to do would be to set $\vartheta = 0$. Note that the expected reward is off by $\sigma^2 d$ at this point, but at least this would be finding a good guess for u. Also, as a function of ϑ, the cost is strongly convex, and the most important thing to know is the expected norm of the gradient, as this will control the number of iterations. Now, after sampling u from a Gaussian with mean ϑ_0 and variance $\sigma^2 I$ and using Equation 13, the first gradient will be

$$g = -\frac{||\omega - \vartheta_0||^2 \omega}{\sigma^2},$$

where ω is a normally distributed random vector with mean zero and covariance $\sigma^2 I$. The expected norm of this stochastic gradient is on the order of

$$O\left(\sigma d^{1.5} + \sigma^{-1} d^{0.5} \|\vartheta_0\|\right),$$

which indicates a significant scaling with dimension.

Several works have analyzed the complexity of this method (28, 36, 37), and the upper and lower bounds strongly depend on the dimension of the search space. The upper bounds also typically depend on the largest magnitude reward B. If the function values are noisy, even for convex functions, the convergence rate is $O[(d^2 B^2/T)^{-1/3}]$, and this assumes you get the algorithm parameters exactly right. For strongly convex functions, this can be reduced to $O[(d^2 B^2/T)^{-1/2}]$ function evaluations, but this result is also rather fragile to the choice of parameters. Finally, note that just adding a constant offset to the reward dramatically slows down the algorithm. If you start with a reward function whose values are in [0,1] and subtract one million from each reward, this will increase the running time of the algorithm by a factor of one million, even though the ordering of the rewards among parameter values remains the same.

5. NUMERICAL COMPARISONS

The preceding analyses of the RL paradigms when applied to LQR are striking. A model-based approach combining supervised learning and robust control achieves nearly optimal performance given its sampling budget. Approximate dynamic programming appears to fare worse in terms of worst-case performance. And direct policy search seems to be of too high variance to work in practice. In this section, we implement these various methods and test them on some simple LQR instances to see how these theoretical predictions reflect practice.

5.1. A Double Integrator

As a simple test case, consider the classic problem of a discrete-time double integrator with the dynamical model

$$x_{t+1} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} x_t + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u_t. \qquad 17.$$

Such a system could model, say, the position (first state) and velocity (second state) of a unit mass object under force u.

As an instance of LQR, we can try to steer this system to reach point 0 from initial condition $x_0 = [-1, 0]$ without expending much force:

$$Q = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, \qquad R = r_0 \tag{18}$$

for some scalar r_0. Note that even in this simple instance there is an element of design: Changing the value of r_0 changes the character of the control law, allowing one to weigh the importance of expending control energy against the time required to reach the destination.

To compare the different approaches, I ran experiments on this instance with a small amount of noise (e_t zero mean with covariance $10^{-4}I$) and training episode length $L = 10$. The goal was to design a controller that works on an arbitrarily long time horizon using the fewest number of simulations of length L.

With one simulation (10 samples), using a white noise input with unit variance, the nominal estimate is correct to three digits of precision. And, not surprisingly, this returns a nearly optimal control policy. Right out of the box, this nominal control strategy works well on this simple example. Note that using a least squares estimator makes the nominal controller's life hard here because all prior information about sparsity on the state-transition matrices is discarded. In a more realistic situation, the only parameter that would need to be estimated would be the (2,1) entry in B, which governs how much force is put out by the actuator and how much mass the system has.

Now, let us compare this approach with approximate dynamic programming and policy search methods. For policy search, let us restrict to policies that use a static, linear gain, as would be optimal on an infinite time horizon. Note that a static linear policy works almost as well as a time-varying policy for this simple LQR problem with two state dimensions. Moreover, there are only two decision variables for this simple problem. For policy gradient, I used the Adam algorithm to shape the iterates (45). I also subtracted the mean reward of previous iterates, a popular baseline subtraction heuristic to reduce variance [Dayan (46) attributes this heuristic to Sutton (47) and Williams (48)]. I was unable to get policy gradient to converge without these additional algorithmic ornamentations. I also compared against a simple approximate dynamic programming method called least squares policy iteration (LSPI), proposed by Lagoudakis & Parr (49). I ran each of these methods using 10 different random seeds. **Figure 1** plots the median performance of the various methods with error bars encompassing the maximum and minimum over all trials. Both nominal control and LSPI are able to find high-quality controllers with only 10 observations. Direct policy methods, on the other hand, require many times as many samples. Policy gradient, in particular, requires thousands of times as many samples as simple nominal control.

5.2. Unstable Laplacian Dynamics

As an illustrative example of the power of LQR as a baseline, let us now move to a considerably harder instance of LQR and show how it highlights issues of robustness and safety. Consider an idealized instance of data center cooling, a popular application of RL (50).

Define the model to have three heat sources coupled to their own cooling devices. Each component of the state x is the internal temperature of one heat source, and the sources heat up under a constant load. They also shed heat to their neighbors. This can be approximately modeled by a

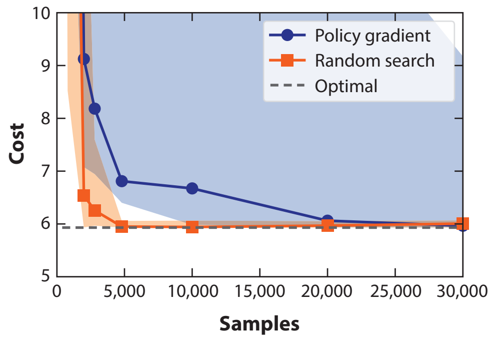

Figure 1

Cost for the double-integrator model for various reinforcement learning algorithms. The solid plots denote the median performance, and the shaded regions capture the maximum and minimum performance. Nominal control and least squares policy iteration are indistinguishable from the optimal controller in this experiment and hence are omitted.

linear dynamical system with state-transition matrices

$$A = \begin{bmatrix} 1.01 & 0.01 & 0 \\ 0.01 & 1.01 & 0.01 \\ 0 & 0.01 & 1.01 \end{bmatrix}, \qquad B = I\,.$$

Note that the open loop system here is unstable: With any nonzero initial condition, the state vector will blow up because the limit of A^k is infinite. Moreover, if a method estimates one of the diagonal entries of A to be less than 1, we might guess that this mode is actually stable and put less effort into cooling that source. So it is imperative to obtain a high-quality estimate of the system's true behavior for near-optimal control. Or, rather, we must be able to ascertain whether our current policy is safe, or the consequences can be disastrous.

Let us try to solve the LQR problem with the settings $Q = I$ and $R = 1{,}000I$. This models a high relative cost for power consumption and hence may encourage small control inputs on modes that are estimated as stable. What happens for our RL methods in this instance?

Figure 2 compares nominal control to two versions of the robust LQR problem described in Section 4.1. To solve the robust LQR problem, we end up solving a small semidefinite programming problem as described by Dean et al. (39). These semidefinite programs are solved on my laptop in well under a second. Note that estimating the error from data only yields slightly worse LQR performance than exactly knowing the true model error.

Note also that the nominal controller does tend to frequently find controllers that fail to stabilize the true system. A necessary and sufficient condition for stabilization is for the matrix $A + BK$ to have all of its eigenvalues be less than 1. We can plot how frequently the various search methods find stabilizing control policies when looking at a finite horizon in **Figure 2*b***. The robust optimization really helps here to provide controllers that are guaranteed to find a stabilizing solution. On the other hand, in industrial practice, nominal control does seem to work quite well. A great open problem is to find reasonable assumptions under which the nominal controller is stabilizing.

Figure 3 additionally compares the performance to model-free methods in this instance. Here we again see that they are indeed far off from their model-based counterparts. The x axis has

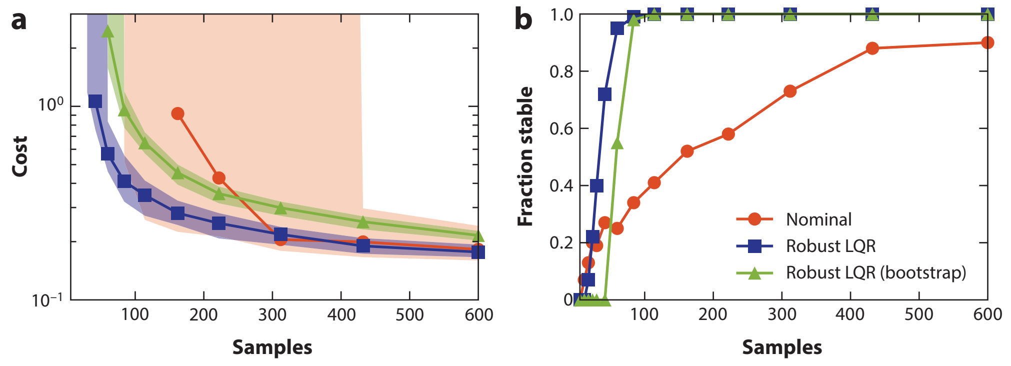

Figure 2

(*a*) Cost for the Laplacian model for varied models. The blue curve shows the performance of a robust LQR when it is provided with the true distance between the estimate and model. The green curve shows the performance when the uncertainty is learned from data using a bootstrap simulation (51, 52). (*b*) The fraction of the time that the synthesized control strategy returns a stabilizing controller. Abbreviation: LQR, linear quadratic regulator.

increased by a factor of 10, and yet even the approximate dynamic programming approach does not find a decent solution. Surprisingly, LSPI, which worked very well on the double integrator, now performs worse than random search. This is likely because the LSPI subroutine requires a stabilizing controller for all iterations and also requires careful tuning of the discount factor. Not only are model-free methods sample hungry, but they fail to be safe. And safety is much more critical than sample complexity.

6. BEYOND THE LINEAR QUADRATIC REGULATOR

Studying simple baselines such as LQR often provides insights into how to approach more challenging problems. In this section, we explore some directions inspired by our analysis of LQR.

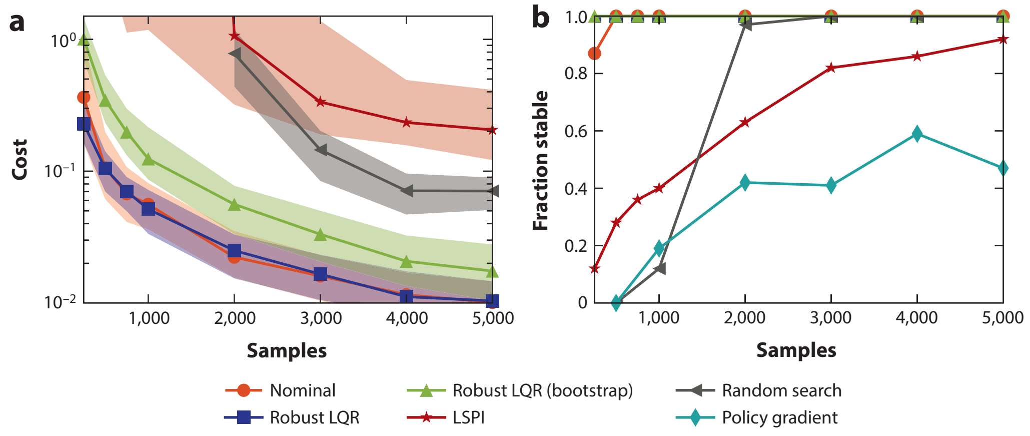

Figure 3

(*a*) Cost for the Laplacian model for varied models over 5,000 iterations. (*b*) The fraction of the time that the synthesized control strategy returns a stabilizing controller. Abbreviations: LQR, linear quadratic regulator; LSPI, least squares policy iteration.

6.1. Derivative-Free Methods for Optimal Control

Random search works well on simple linear problems and appears to be better than more complex methods like policy gradient. Does simple random search work less well on more difficult problems?

The answer, it turns out, is yes. The deep RL community has recently been using a suite of benchmarks to compare methods, maintained by OpenAI (**https://gym.openai.com/envs/#mujoco**) and based on the MuJoCo simulator (53). Here, the optimal control problem is to get the simulation of a legged robot to walk as far and quickly as possible in one direction. Some of the tasks are very simple, but some are quite difficult, like the complicated humanoid models with 22 degrees of freedom. The dynamics of legged robots are well specified by Lagrange's equations (54), but planning locomotion from these models is challenging because it is not clear how to best design the objective function and because the model is piecewise linear. The model changes whenever part of the robot comes into contact with a solid object, and hence a normal force is introduced that was not previously acting upon the robot. Hence, getting robots to work without having to deal with complicated nonconvex nonlinear models seems like a solid and interesting challenge for the RL paradigm. Moreover, seminal work by Tedrake et al. (55) demonstrated that direct policy search could rapidly find feedback control policies for certain constrained legged robot designs.

Levine & Koltun (56) were among the first to use MuJoCo as a test bed for learning-based control, and they were able to achieve walking in complex simulators without special-purpose techniques. Since then, these techniques have become standard continuous control benchmarks for RL (see, for example, 57–61). Recently, Salimans and his collaborators at OpenAI (62) showed that random search worked quite well on these benchmarks. In particular, they fit neural network controllers using random search with a few algorithmic enhancements. Random search had indeed enjoyed significant success in some corners of the robotics community, and others had noted that in their applications, random search outperformed policy gradient (63). In another piece of great work, Rajeswaran et al. (64) showed that natural policy gradient could learn linear policies that could complete these benchmarks. That is, they showed that static linear state feedback, like the kind we use in LQR, was also sufficient to control these complex robotic simulators. This of course left an open question: Can simple random search find linear controllers for these MuJoCo tasks?

Mania et al. (65) tested this out, coding up a rather simple version of random search with a couple of small algorithmic enhancements. Many RL papers were using statistics of the states and whitening the states before passing them into the neural network mapping from state to action. This study found that when random search performed the same whitening with linear controls, this algorithm was able to get state-of-the-art results on all of the MuJoCo benchmark tasks.

There are a few of important takeaways from this study. On the one hand, the results suggest that these MuJoCo demos are easy, or at least considerably easier than they were believed to be. Benchmarking is difficult, and having only a few simulation benchmarks encourages overfitting to these benchmarks. Indeed, it does seem like these benchmarks are more about taking advantage of simulation approximations in MuJoCo than they are about learning reasonable policies for walking. In terms of benchmarking, this is what makes LQR so attractive: LQR with unknown dynamics is a reasonable task to master, as it is easy to specify new instances, and it is relatively easy to understand the limits of achievable performance.

Second, note that since our random search method is fast, we can evaluate its performance on many random seeds. All model-free methods exhibit alarmingly high variance on these benchmarks. For instance, on the humanoid task, the model is slow to train almost a quarter of the time even when supplied with what we thought were good parameters (see **Figure 4*b***). And, for those random seeds, we found that the method returned rather peculiar gaits. Henderson et al. (66) and Islam et al. (67) observed this phenomenon with deep RL methods, but our results on linear

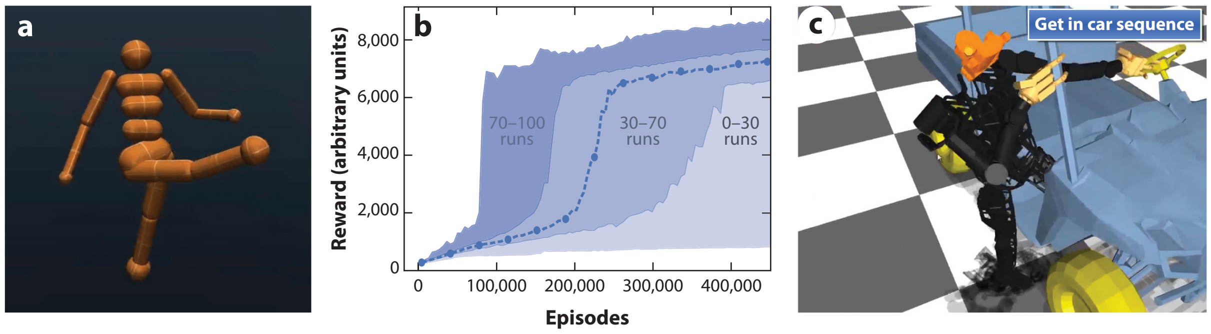

Figure 4

(*a*) Sample frame of the MuJoCo humanoid. (*b*) Variance of learning performance on 100 runs of random search on the humanoid model. Note that though high rewards are often achieved, it is more common to observe poor control performance from a random initialization. (*c*) Using model predictive control and a poor model, complex actions with humanoid simulations can be executed, such as climbing into a vehicle. Panel *b* adapted from Reference 65; panel *c* adapted with permission from a supplementary video from Reference 68.

controllers suggest that such high variability will be a symptom of all model-free methods. Though direct policy search methods are easy to code up, their reliability on any reasonable control task remains in question.

6.2. Receding Horizon Control

Approximate dynamic programming is closely related to canonical receding horizon control (RHC), also known as model predictive control. In RHC, an agent makes a plan based on a simulation from the present until a short time into the future. The agent then executes one step of this plan, and based on what it observes after taking this action, it returns to short-time simulation to plan the next action. This feedback loop allows the agent to link the actual impact of its choice of action with what was simulated, and hence it can correct for model mismatch, noise realizations, and other unexpected errors.

To relate RHC to approximate dynamic programming, note that the discounted problem

$$\begin{aligned} &\text{maximize } (1-\gamma)\mathbb{E}_{e_t}\left[\sum_{t=0}^{N-1}\gamma^t R(x_t,u_t)+\gamma^N Q_\gamma(x_{N+1},u_{T+1})\right]\\ &\text{subject to } x_{t+1}=f(x_t,u_t,e_t)\\ &\qquad\qquad (x_0 \text{ given})\end{aligned}$$

is equivalent to Problem 8. Here, we have just unrolled the cost beyond one step. Though this is trivial, it is again incredibly powerful: The longer we make the time horizon, the less we need to worry about the Q-function being accurate. Of course, now we need to worry about the accuracy of the state-transition map, f. But, especially in problems with continuous variables, it is not at all obvious which accuracy is more important in terms of finding algorithms with fast learning rates and short computation times. There is a trade-off between learning models and learning value functions, and this is a trade-off that needs to be better understood.

Though RHC methods appear fragile to model mismatch, the repeated feedback inside RHC can correct for many modeling errors. As an example, it is worth revisiting the robotic locomotion tasks inside the MuJoCo framework. These tasks were actually designed to test the power of a nonlinear RHC algorithm developed by Tassa et al. (69). The receding horizon controller works to keep the robot upright even when the model is poorly specified. Moreover, the RHC approach to humanoid control solved for the controller in 7× real time in 2012. In 2013, the same research group published a cruder version of their controller that they used during the DARPA Robotics Challenge (68). All these behaviors are generated by RHC in real time. Though the resulting walking is not of the same quality as what can be obtained from computationally intensive long-horizon trajectory optimization, it does look considerably better than the sort of gaits typically obtained by popular RL methods.

Is there a middle ground between expensive offline trajectory optimization and real-time RHC? I think the answer is yes, in the same way that there is a middle ground between learning dynamical models and learning Q-functions. The performance of an RHC system can be improved by better modeling of the Q-function that defines the terminal cost: The better a model you make of the Q-function, the shorter a time horizon you need for simulation, and the closer you get to real-time operation. Of course, if you had a perfect model of the Q-function, you could just solve the Bellman equation, and you would then have the optimal control policy. But by having an approximation to the Q-function, you can still extract high performance in real time.

So what if we learn to iteratively improve the Q-function while running RHC? This idea has been explored in a project by Rosolia & Borrelli (70). In their approach, the terminal cost is learned by a method akin to nearest neighbors. The terminal cost of a state is the value obtained the last time that state was tried. If that state has not been visited, the cost is infinite. This formulation constrains the terminal condition to be in a state observed before. It enables the control system to explore new ways to decrease cost as long as it maintains the ability to reach a state that has already been demonstrated to be safe. This nearest-neighbors approach works surprisingly well in practice: In radio-controlled car demonstrations, the learned controller works better than a human operator after only a few laps around a fixed track.

Another reason to like this blended RHC approach to learning to control is that one can hard code in constraints on controls and states and easily incorporate models of disturbance directly into the optimization problem. Some of the most challenging problems in control are how to execute safely while continuing to learn more about a system's capability, and an RHC approach provides a direct route toward balancing safety and performance. Indeed, an interesting direction of future work would be merging the robust learning of coarse-ID control with RHC.

7. CHALLENGES AT THE CONTROL–LEARNING INTERFACE

We have set out to bridge the gap between the learning-centric views of RL and the model-centric views of control. Perhaps surprisingly, we found that for continuous control problems, machine learning seems best suited for model fitting rather than for direct control. Moreover, perhaps less surprisingly, we could seamlessly merge learned models and control action by accounting for the uncertainty in our model fits. We also showed that value functions and models could be learned in chorus and could provide impressive results on real embodied agents. These distinctions and connections are merely the beginning of what the machine learning and control communities can learn from each other. Let me close by discussing three particularly exciting and important research challenges that may be best solved with input from both perspectives.

7.1. Merging Perception and Control

One of the grand aspirations of RL is end-to-end control, mapping directly from sensors (e.g., pixels) to actions. Computer vision has made major advances by adopting an all-convolutional-network end-to-end approach in the research area, and many studies, including industrial research at NVIDIA (71), suggest that the same can be done for complex control tasks.

In general, this problem gets into very old intractability issues of nonlinear output feedback in control (72) and partially observed Markov decision processes in RL (73). Nonetheless, some early results in RL have shown promise in training optimal controllers directly from pixels (12, 58). Of course, these results have even worse sample complexity than the same methods trained from states, but they are making progress.

In my opinion, the most promising approaches in this space follow the ideas of guided policy search, which bootstraps standard state feedback to provide training data for a map from sensors directly to optimal action (56, 74). That is, a mapping from sensor to action can be learned iteratively by first finding the optimal action and then finding a map to that control setting. A coupling along these lines, where reliance on a precise state estimator is reduced over time, could potentially provide a reasonably efficient method for learning to control from sensors.

However, these problems remain daunting. Moving from fully observed scenarios to partially observed scenarios makes the control problem exponentially more difficult. How to use diverse sensor measurements in a safe and reliable manner remains an active and increasingly important research challenge (75–77).

7.2. Rethinking Adaptive Control

This survey has focused on episodic RL and has steered clear of a much harder problem: adaptive control. In the adaptive setting, we want to learn the policy online. We only get one trajectory, and the goal is, after a few steps, to have a model whose reward from here to eternity will be large. This is very different, and much harder, than what people are doing in RL. In episodic RL, you get endless access to a simulator. In adaptive control, you get one go.

Even for LQR, the best approach to adaptive control is not settled. Pioneering work in the 1980s used stochastic gradient-like techniques to find adaptive controls, but the guarantees for these schemes are all asymptotic (78). More recently, there has been a groundswell of activity in trying to understand this problem from the perspective of online learning. Beginning with work by Abbasi-Yadkori & Szepesvári (79), a variety of approaches have been devised to provide efficient schemes that yield near-optimal control cost. Abbasi-Yadkori & Szepesvári's approach achieves an optimal reward, building on techniques that give optimal algorithms for the multiarmed bandit (80, 81). But their method requires solving a hard nonconvex optimization problem as a subroutine. Follow-up work has proposed methods using Thompson sampling (82–84), approximate dynamic programming (85), and even coarse-ID control (86), though no method has been found that is efficiently implementable and achieves optimal cost. Again, this emphasizes that even the simple LQR problem is not at all simple. New techniques must be developed to fully understand this baseline problem, but it is clear that insights from both machine learning and control will be necessary to develop efficient adaptive control that can cope with an ever-evolving world.

7.3. Humans in the Loop

One final important problem, which might be the most daunting of all, is how machines should learn when humans are in the loop. What can humans who are interacting with the robots do, and how can we model human actions? Though economists may have a different opinion, humans are challenging to model.

One popular approach to modeling human–robot interaction is game theoretic. Humans can be modeled as solving their own optimal control problem, and then the human's actions enter as a disturbance in the optimal control problem (87). In this way, modeling humans is similar to modeling uncertain dynamic environments. But thinking of the humans as optimizers means that their behavior is constrained. If we know the cost, then we get a complex game-theoretic version of RHC (88, 89). But, as is usually the case, humans are bad at specifying their objectives, and hence what they are optimizing must be learned. This becomes a problem of inverse optimal control (90) or inverse RL (91), where we must estimate the reward functions of the human and understand the loss accrued for crudely modeling these rewards.

7.4. Toward Actionable Intelligence

As expressed above, I think that all of the daunting problems in machine learning are now RL problems. Whether they be autonomous transportation systems or seemingly mundane social network engagement systems, actively interacting with reality has high stakes. Indeed, as soon as a machine learning system is unleashed in feedback with humans, that system is an RL system. The broad engineering community must take responsibility for the now ubiquitous machine learning systems and understand what happens when we set them loose on the world.

Solving these problems will require advances in both machine learning and control. Perhaps this intersection needs a new name, so that researchers can stop arguing about territory. I personally am fond of the term actionable intelligence, as it sums up not only robotics but smarter, safer analytics. But regardless of what we call it, the point is that there is a large community spanning multiple disciplines that is invested in making progress on these problems. Hopefully this tour has set the stage for a lot of great research at the intersection of machine learning and control, and I am excited to see what progress the communities can make working together.

DISCLOSURE STATEMENT

The author received a 2018 Amazon AWS Machine Learning Research Award, was a consultant at Google from 2015 to 2017, and is a member of the Berkeley DeepDrive Consortium (sponsors listed at **https://deepdrive.berkeley.edu**) and Berkeley Artificial Intelligence Research (sponsors listed at **https://bair.berkeley.edu/sponsor.html**).

ACKNOWLEDGMENTS

Countless individuals have helped to shape the content of this review. First, this work was generously supported in part by two forward-looking programs at the US Department of Defense: the Mathematical Data Science program at the Office of Naval Research and the Foundations and Limits of Learning program at DARPA. Second, this article was distilled from a series on my blog at **http://argmin.net**. I greatly appreciated the lively debates on Twitter, and I hope that even those who disagree with my perspectives find their input incorporated into this survey.

I thank Chris Wiggins for sharing his taxonomy on machine learning, Roy Frostig for shaping Section 3.3, Pavel Pravdin for consulting on how to get policy gradient methods up and running, and Max Raginsky for perspectives on adaptive control and translations of Russian. I also thank Moritz Hardt, Eric Jonas, and Ali Rahimi for helping to shape the language, rhetoric, and focus of the blog series and this survey, and Nevena Lazic and Gergely Neu for many helpful suggestions for improving the article's readability and accuracy. Additionally, I thank my other colleagues in machine learning and control for many helpful conversations and pointers about this material: Murat Arcak, Karl Astrom, Francesco Borrelli, John Doyle, Andy Packard, Anders Rantzer, Lorenzo Rosasco, Shankar Sastry, Yoram Singer, Csaba Szepesvari, Claire Tomlin, and Stephen Wright. I also thank my colleagues in robotics—Anca Dragan, Leslie Kaebling, Sergey Levine, Pierre-Yves Oudeyer, Olivier Sigaud, Russ Tedrake, and Emo Todorov—for sharing their perspectives on the RL and optimization technologies that work for them and the challenges they face in their research.

I thank everyone who took CS281B with me in the spring of 2017, where I first tried to make sense of the problems in learning to control. And most importantly, a big thanks to everyone in my research group, who have been wrestling with these ideas with me for the past several years and who have done much of the research that shaped my views on this space, particularly Ross Boczar, Nick Boyd, Sarah Dean, Animesh Garg, Aurelia Guy, Qingqing Huang, Kevin Jamieson, Sanjay Krishnan, Laurent Lessard, Horia Mania, Nik Matni, Becca Roelofs, Ugo Rosolia, Ludwig Schmidt, Max Simchowitz, Stephen Tu, and Ashia Wilson.

Finally, special thanks to Camon Coffee in Berlin for letting me haunt their shop while writing.

LITERATURE CITED

1. Silver D, Huang A, Maddison CJ, Guez A, Sifre L, et al. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* 529:484–89
2. Bertsekas DP. 2017. *Dynamic Programming and Optimal Control*, Vol. 1. Nashua, NH: Athena Sci. 4th ed.
3. Sutton RS, Barto AG. 1998. *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press
4. Puterman ML. 1994. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Hoboken, NJ: Wiley-Interscience
5. Dann C, Brunskill E. 2015. Sample complexity of episodic fixed-horizon reinforcement learning. In *Advances in Neural Information Processing Systems 28*, ed. C Cortes, ND Lawrence, DD Lee, M Sugiyama, R Garnett, pp. 2818–26. Red Hook, NY: Curran
6. Nemirovski A, Yudin D. 1983. *Problem Complexity and Method Efficiency in Optimization*. New York: Wiley
7. Zhu X. 2005. *Semi-supervised learning literature survey*. Tech. Rep. 1530, Dep. Comput. Sci., Univ. Wisc., Madison
8. Hazan E, Kale S, Shalev-Shwartz S. 2012. Near-optimal algorithms for online matrix prediction. In *Proceedings of the 25th Annual Conference on Learning Theory*, ed. S Mannor, N Srebro, RC Williamson, pp. 38.1–13. Proc. Mach. Learn. Res. 23. N.p.: PMLR
9. Bertsekas DP, Tsitsiklis JN. 1996. *Neuro-Dynamic Programming*. Belmont, MA: Athena Sci.
10. Kaelbling LP, Littman ML, Moore AW. 1996. Reinforcement learning: a survey. *J. Artif. Intell. Res.* 4: 237–85
11. Bowling M, Burch N, Johanson M, Tammelin O. 2015. Heads-up limit hold'em poker is solved. *Science* 347:145–49
12. Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518:529–33
13. Tesauro G. 1995. TD-Gammon: a self-teaching backgammon program. In *Applications of Neural Networks*, ed. AF Murray, pp. 267–85. Boston, MA: Springer
14. Bottou L, Peters J, Quiñonero-Candela J, Charles DX, Chickering DM, et al. 2013. Counterfactual reasoning and learning systems: the example of computational advertising. *J. Mach. Learn. Res.* 14:3207–60
15. Strehl A, Langford J, Li L, Kakade SM. 2010. Learning from logged implicit exploration data. In *Advances in Neural Information Processing Systems 23*, ed. JD Lafferty, CKI Williams, J Shawe-Taylor, RS Zemel, A Culotta, pp. 2217–25. Red Hook, NY: Curran
16. Bertsekas DP. 2012. *Dynamic Programming and Optimal Control*, Vol. 2. Nashua, NH: Athena Sci. 4th ed.
17. Ljung L. 1998. *System Identification: Theory for the User*. Upper Saddle River, NJ: Prentice Hall. 2nd ed.
18. Campi MC, Weyer E. 2002. Finite sample properties of system identification methods. *IEEE Trans. Autom. Control* 47:1329–34
19. Vidyasagar M, Karandikar RL. 2008. A learning theory approach to system identification and stochastic adaptive control. *J. Process Control* 18:421–30
20. Tsitsiklis JN. 1994. Asynchronous stochastic approximation and Q-learning. *Mach. Learn.* 16:185–202
21. Watkins CJ, Dayan P. 1992. Q-learning. *Mach. Learn.* 8:279–92

22. Sutton RS. 1988. Learning to predict by the method of temporal differences. *Mach. Learn.* 3:9–44
23. Dayan P. 1992. The convergence of TD(λ) for general λ. *Mach. Learn.* 8:341–62
24. Bradtke SJ, Barto AG. 1996. Linear least-squares algorithms for temporal difference learning. *Mach. Learn.* 22:33–57
25. Bertsekas DP, Ioffe S. 1996. *Temporal differences-based policy iteration and applications in neuro-dynamic programming*. Tech. Rep. LIDS-P-2349, Lab. Inf. Decis. Syst., Mass. Inst. Technol., Cambridge, MA
26. Yu H, Bertsekas DP. 2009. Convergence results for some temporal difference methods based on least squares. *IEEE Trans. Autom. Control* 54:1515–31
27. Williams RJ. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.* 8:229–56
28. Jamieson KG, Nowak RD, Recht B. 2012. Query complexity of derivative-free optimization. In *Advances in Neural Information Processing Systems 25*, ed. F Pereira, CJC Burges, L Bottou, KQ Weinberger, pp. 2672–80. Red Hook, NY: Curran
29. Åström KJ. 1965. Optimal control of Markov processes with incomplete state information. *J. Math. Anal. Appl.* 10:174–205
30. Kaelbling LP, Littman ML, Cassandra AR. 1998. Planning and acting in partially observable stochastic domains. *Artif. Intell.* 101:99–134
31. Rastrigin LA. 1963. About convergence of random search method in extremal control of multi-parameter systems. *Avtomat. Telemekh.* 24:1467–73
32. Nesterov Y, Spokoiny V. 2017. Random gradient-free minimization of convex functions. *Found. Comput. Math.* 17:527–66
33. Beyer HG, Schwefel HP. 2002. Evolution strategies: a comprehensive introduction. *Nat. Comput.* 1:3–52
34. Schwefel HP. 1975. *Evolutionsstrategie und numerische optimierung*. PhD Thesis, Tech. Univ. Berlin
35. Spall JC. 1992. Multivariate stochastic approximation using a simultaneous perturbation gradient approximation. *IEEE Trans. Autom. Control* 37:332–41
36. Flaxman AD, Kalai AT, McMahan HB. 2005. Online convex optimization in the bandit setting: gradient descent without a gradient. In *Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 385–94. Philadelphia: Soc. Ind. Appl. Math.
37. Agarwal A, Dekel O, Xiao L. 2010. Optimal algorithms for online convex optimization with multi-point bandit feedback. In *COLT 2010: 23rd Conference on Learning Theory*, ed. AT Kalai, M Mohri, pp. 28–40. Madison, WI: Omnipress
38. Zhou K, Doyle JC, Glover K. 1995. *Robust and Optimal Control*. Upper Saddle River, NJ: Prentice Hall
39. Dean S, Mania H, Matni N, Recht B, Tu S. 2017. On the sample complexity of the linear quadratic regulator. arXiv:1710.01688 [math.OC]
40. Simchowitz M, Mania H, Tu S, Jordan MI, Recht B. 2018. Learning without mixing. In *Proceedings of the 31st Conference on Learning Theory*, ed. S Bubeck, V Perchet, P Rigollet, pp. 439–73. Proc. Mach. Learn. Res. 75. N.p.: PMLR
41. Matni N, Wang YS, Anderson J. 2017. Scalable system level synthesis for virtually localizable systems. In *2017 IEEE 56th Annual Conference on Decision and Control*, pp. 3473–80. New York: IEEE
42. Wang YS, Matni N, Doyle JC. 2016. A system level approach to controller synthesis. arXiv:1610.04815 [cs.SY]
43. Bradtke SJ, Ydstie BE, Barto AG. 1994. Adaptive linear quadratic control using policy iteration. In *Proceedings of the 1994 American Control Conference*, pp. 3475–79. New York: IEEE
44. Tu SL, Recht B. 2018. Least-squares temporal difference learning for the linear quadratic regulator. In *Proceedings of the 35th International Conference on Machine Learning*, ed. J Dy, A Krause, pp. 5005–14. Proc. Mach. Learn. Res. 80. N.p.: PMLR
45. Kingma D, Ba J. 2014. Adam: a method for stochastic optimization. arXiv:1412.6980 [cs.LG]
46. Dayan P. 1991. Reinforcement comparison. In *Connectionist Models: Proceedings of the 1990 Summer School*, ed. DS Touretzky, JL Elman, TJ Sejnowski, GE Hinton, pp. 45–51. San Mateo, CA: Morgan Kaufmann
47. Sutton RS. 1984. *Temporal credit assignment in reinforcement learning*. PhD Thesis, Univ. Mass., Amherst
48. Williams RJ. 1988. *Toward a theory of reinforcement-learning connectionist systems*. Tech. Rep. NU-CCS-88-3, Coll. Comput. Sci., Northeast. Univ., Boston

49. Lagoudakis MG, Parr R. 2003. Least-squares policy iteration. *J. Mach. Learn. Res.* 4:1107–49
50. Gao J. 2014. *Machine learning applications for data center optimization*. White Pap., Google, Mountain View, CA
51. Efron B. 1979. Bootstrap methods: another look at the jackknife. *Ann. Stat.* 7:1–26
52. Shao J, Tu D. 1995. *The Jackknife and Bootstrap*. New York: Springer-Verlag
53. Todorov E, Erez T, Tassa Y. 2012. MuJoCo: a physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–33. New York: IEEE
54. Murray RM. 2017. *A Mathematical Introduction to Robotic Manipulation*. Boca Raton, FL: CRC
55. Tedrake R, Zhang TW, Seung HS. 2004. Stochastic policy gradient reinforcement learning on a simple 3D biped. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2849–54. New York: IEEE
56. Levine S, Koltun V. 2013. Guided policy search. In *Proceedings of the 30th International Conference on Machine Learning*, ed. S Dasgupta, D McAllester, pp. 1–9. Proc. Mach. Learn. Res. 28(3). N.p.: PMLR
57. Silver D, Lever G, Heess N, Degris T, Wierstra D, Riedmiller M. 2014. Deterministic policy gradient algorithms. In *Proceedings of the 31st International Conference on Machine Learning*, ed. EP Xing, T Jebara, pp. 387–95. Proc. Mach. Learn. Res. 32(1). N.p.: PMLR
58. Lillicrap TP, Hunt JJ, Pritzel A, Heess N, Erez T, et al. 2015. Continuous control with deep reinforcement learning. arXiv:1509.02971 [cs.LG]
59. Schulman J, Levine S, Abbeel P, Jordan M, Moritz P. 2015. Trust region policy optimization. In *Proceedings of the 32nd International Conference on Machine Learning*, ed. F Bach, D Blei, pp. 1889–97. Proc. Mach. Learn. Res. 37. N.p.: PMLR
60. Schulman J, Moritz P, Levine S, Jordan M, Abbeel P. 2015. High-dimensional continuous control using generalized advantage estimation. arXiv:1506.02438 [cs.LG]
61. Wu Y, Mansimov E, Liao S, Grosse R, Ba J. 2017. Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation. arXiv:1708.05144 [cs.LG]
62. Salimans T, Ho J, Chen X, Sutskever I. 2017. Evolution strategies as a scalable alternative to reinforcement learning. arXiv:1703.03864 [stat.ML]
63. Stulp F, Sigaud O. 2013. Robot skill learning: from reinforcement learning to evolution strategies. *Paladyn* 4:49–61
64. Rajeswaran A, Lowrey K, Todorov E, Kakade S. 2017. Towards generalization and simplicity in continuous control. In *Advances in Neural Information Processing Systems 30*, ed. I Guyon, UV Luxburg, S Bengio, H Wallach, R Fergus, et al., pp. 6550–61. Red Hook, NY: Curran
65. Mania H, Guy A, Recht B. 2018. Simple random search provides a competitive approach to reinforcement learning. arXiv:1803.07055 [cs.LG]
66. Henderson P, Islam R, Bachman P, Pineau J, Precup D, Meger D. 2017. Deep reinforcement learning that matters. arXiv:1709.06560 [cs.LG]
67. Islam R, Henderson P, Gomrokchi M, Precup D. 2017. Reproducibility of benchmarked deep reinforcement learning tasks for continuous control. arXiv:1708.04133 [cs.LG]
68. Erez T, Lowrey K, Tassa Y, Kumar V, Kolev S, Todorov E. 2013. An integrated system for real-time model predictive control of humanoid robots. In *2013 13th IEEE-RAS International Conference on Humanoid Robots*, pp. 292–99. New York: IEEE
69. Tassa Y, Erez T, Todorov E. 2012. Synthesis and stabilization of complex behaviors through online trajectory optimization. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4906–13. New York: IEEE
70. Rosolia U, Borrelli F. 2018. Learning model predictive control for iterative tasks. A data-driven control framework. *IEEE Trans. Autom. Control* 63:1883–96
71. Bojarski M, Del Testa D, Dworakowski D, Firner B, Flepp B, et al. 2016. End to end learning for self-driving cars. arXiv:1604.07316 [cs.CV]
72. Blondel VD, Tsitsiklis JN. 2000. A survey of computational complexity results in systems and control. *Automatica* 36:1249–74
73. Papadimitriou CH, Tsitsiklis JN. 1987. The complexity of Markov decision processes. *Math. Oper. Res.* 12:441–50

74. Levine S, Finn C, Darrell T, Abbeel P. 2016. End-to-end training of deep visuomotor policies. *J. Mach. Learn. Res.* 17:1–40
75. Akametalu AK, Fisac JF, Gillula JH, Kaynama S, Zeilinger MN, Tomlin CJ. 2014. Reachability-based safe learning with Gaussian processes. In *53rd IEEE Conference on Decision and Control*, pp. 1424–31. New York: IEEE
76. Aswani A, Gonzalez H, Sastry SS, Tomlin CJ. 2013. Provably safe and robust learning-based model predictive control. *Automatica* 49:1216–26
77. Berkenkamp F, Turchetta M, Schoellig A, Krause A. 2017. Safe model-based reinforcement learning with stability guarantees. In *Advances in Neural Information Processing Systems 30*, ed. I Guyon, UV Luxburg, S Bengio, H Wallach, R Fergus, et al., pp. 908–18. Red Hook, NY: Curran
78. Goodwin GC, Ramadge PJ, Caines PE. 1981. Discrete time stochastic adaptive control. *SIAM J. Control Optim.* 19:829–53
79. Abbasi-Yadkori Y, Szepesvári C. 2011. Regret bounds for the adaptive control of linear quadratic systems. In *Proceedings of the 24th Annual Conference on Learning Theory*, ed. SM Kakade, U von Luxburg, pp. 1–26. Proc. Mach. Learn. Res. 19. N.p.: PMLR
80. Auer P, Cesa-Bianchi N, Fischer P. 2002. Finite-time analysis of the multiarmed bandit problem. *Mach. Learn.* 47:235–56
81. Lai TL, Robbins H. 1985. Asymptotically efficient adaptive allocation rules. *Adv. Appl. Math.* 6:4–22
82. Abbasi-Yadkori Y, Szepesvári C. 2015. Bayesian optimal control of smoothly parameterized systems: the lazy posterior sampling algorithm. In *Proceedings of the Thirty-First Conference on Uncertainty in Artificial Intelligence*, ed. M Meila, T Heskes, pp. 2–11. Arlington, VA: AUAI
83. Abeille M, Lazaric A. 2017. Thompson sampling for linear-quadratic control problems. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, ed. A Singh, J Zhu, pp. 176–84. Proc. Mach. Learn. Res. 54. N.p.: PMLR
84. Ouyang Y, Gagrani M, Jain R. 2017. Learning-based control of unknown linear systems with Thompson sampling. arXiv:1709.04047 [cs.SY]
85. Abbasi-Yadkori Y, Lazic N, Szepesvári C. 2018. The return of ϵ-greedy: sublinear regret for model-free linear quadratic control. arXiv:1804.06021 [cs.LG]
86. Dean S, Mania H, Matni N, Recht B, Tu S. 2018. Regret bounds for robust adaptive control of the linear quadratic regulator. arXiv:1805.09388 [cs.LG]
87. Sadigh D, Sastry S, Seshia SA, Dragan AD. 2016. Planning for autonomous cars that leverage effects on human actions. In *Robotics: Science and Systems XII*, ed. D Hsu, N Amato, S Berman, S Jacobs, chap. 29. N.p.: Robot. Sci. Syst. Found.
88. Bialas WF. 1989. Cooperative *n*-person Stackelberg games. In *Proceedings of the 28th IEEE Conference on Decision and Control*, Vol. 3, pp. 2439–44. New York: IEEE
89. Li N, Oyler DW, Zhang M, Yildiz Y, Kolmanovsky I, Girard AR. 2017. Game theoretic modeling of driver and vehicle interactions for verification and validation of autonomous vehicle control systems. *IEEE Trans. Control Syst. Technol.* 26:1782–97
90. Kalman RE. 1964. When is a linear control system optimal? *J. Basic Eng.* 86:51–60
91. Ng AY, Russell SJ. 2000. Algorithms for inverse reinforcement learning. In *Proceedings of the Seventeenth International Conference on Machine Learning*, ed. P Langley, pp. 663–70. San Francisco: Morgan Kaufmann