ECE 598: Interplay between Control and Machine Learning **Fall 2020**

Lecture 7

Introduction to Reinforcement Learning

Lecturer: Bin Hu, Date:09/24/2020

In this lecture, we will give a big picture for reinforcement learning. Recall that a MDP is defined by a tuple $\langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$ where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, P is the transition kernel, R is the reward, and γ is the discount factor. The goal is to choose the action sequence $\{a_t\}$ to maximize the total accumulated rewards $V(s_0) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k R(s_k, a_k) \big| s_0\right]$.

7.1 Policy/Controller

The action a_t is typically determined using a feedback function of s_t. This is a natural consequence of dynamic programming and the concept of feedback is very important for managing uncertainty. The feedback law mapping s_t to a_t is called "policy" (in the reinforcement learning literature) or "controller" (in some controls literature). A policy can be deterministic (i.e. $s_t = \pi(a_t)$ where π is a fixed nonlinear function) or stochastic (i.e. it maps each state to a probability distribution over the action space $\mathcal{A}$). Then the goal can be equivalently formulated as finding an optimal policy that maximizes the total accumulated rewards, i.e.

$$\pi^* = \arg\max_{\pi} \ \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k R(s_k, a_k) \big| a_k \sim \pi(\cdot|s_k), s_0\right].$$

Policy in the finite state/action space setting. If both $\mathcal{S}$ and $\mathcal{A}$ are finite, then the policy parameterization is straightforward. Any deterministic policy can be actually represented by a vector. For simplicity, consider $\mathcal{S} = \{1, 2, \ldots, n\}$ and $\mathcal{A} = \{1, 2, \ldots, L\}$. Then a deterministic policy π can be specified as as vector

$$\begin{bmatrix} \pi(1) \\ \pi(2) \\ \vdots \\ \pi(n) \end{bmatrix}$$

where $\pi(i) \in \mathcal{A}$. Clearly the above vector is in $\mathbb{R}^n$. If we further consider a stochastic policy, then each $\pi(i)$ is a probability distribution over $\mathcal{A}$ and hence is a vector in $\mathbb{R}^L$. Therefore, we have $\pi \in \mathbb{R}^{nL}$ for any stochastic policy.

Policy parameterization in control applications. For control problems, we need to "parameterize" the policy. In other words, the policy is a function mapping from states to actions, and we need to specify this function by introducing some parameters. In the simplest case, we can set $u_t = -Kx_t$ where K is a static matrix. Then the action is just a linear function of the state, and this is called linear state feedback control. It is also possible to use nonlinear functions. For example, we can parameterize the controller as a two-layer neural network, i.e. $u_t = W^2\sigma(W^1 x_t)$ where W^1 is the weight in the first layer, W^2 is the weight in the second layer, and σ is some nonlinear activation function. We can also increase the number of the layers and use deep neural networks to parameterize the policy. The deep neural network parameterization is more popular in the machine learning community.

7.2 Policy Evaluation

To find the best policy, at least we need some tools to assess the performance of a given policy. This task is called policy evaluation. The goal here is to calculate the value function for any given policy.

$$V^\pi(s_0) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k R(s_k, a_k) \big| a_k \sim \pi(\cdot|s_k), s_0\right].$$

Now we give a high-level review of policy evaluation methods:

1. When the model is known, one can solve the Bellman equation to obtain the value function of a given policy.

2. When the model is unknown, one can apply temporal difference learning methods (e.g. TD, GTD, TDC, LSTD) to fit (or learn) a value function from data.

3. There are two types of value functions: state value function and Q-function. TD learning can be applied to learn both. LSTD-Q is an off-policy method which is very efficient in learning Q-function.

$$Q^\pi(s, a) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k R(s_k, a_k) \big| s_0 = s, a_0 = a, a_k \sim \pi(\cdot|s_k)\, \forall k \geq 1\right].$$

4. On-policy vs. off-policy: On-policy methods rely on data generated by the policy one wants to evaluate. Off-policy methods use the data generated by the behavior policy to learn the value function of the target policy. Hence off-policy methods are safer in the sense that one can choose a safe behavior policy beforehand.

7.3 Optimal Control Design

Now we talk a little bit about how to find the optimal policy. When the model is known, one can solve the optimal Bellman equation to obtain the optimal policy. We are mostly interested in the case where the model is unknown. Here is a high-level review of popular reinforcement learning methods.

1. Value-based method: Q-learning which uses data to estimate the optimal Q-function. Usually special tricks such as replay buffer and frozen targets are required to make it work on control problems.
2. Approximate policy iteration: Estimate Q-function for a fixed policy and then use the Q-function to plan greedily to obtain a new policy. Then iterate between the two steps.
3. Policy-gradient methods: Try to use data to estimate the gradient $\nabla_\pi V$ and then improve the policy using the gradient information. Examples include policy gradient, natural policy gradient, trust region policy optimization (TRPO), proximal policy optimization (PPO), and actor-critic.
4. Model-based methods: estimate the model first and then solve the optimal Bellman equation or do model predictive control. Sometimes people will also look at regret bounds.

For control applications, another important research topic is how to enforce other constraints (stability, safety, robustness) on the policy. Many people are working on this right now. We will also talk about constrained policy optimization (CPO) and interior policy optimization (IPO).

7.4 Policy Optimization beyond the MDP Setup

Finally, let's have a brief discussion on the relationship between reinforcement learning and control. Many control problems can be formulated as constrained optimization. Specifically, consider a general nonlinear system

$$x_{t+1} = f(x_t, u_t, w_t)$$

where x_t is the state, u_t is the control action, and w_t is the process noise. Suppose the goal is to design a controller $u_t = K(x_t)$. Then many control design problems can be formulated as

$$\underset{K \in \mathcal{K}}{\text{minimize}}\, J(K)$$

where K is the policy parameter, J is a control performance measure, and $\mathcal{K}$ is a feasible set enforcing constraints on the policy. As discussed before, K is typically set a static matrix or a neural network. Now we discuss more on J and $\mathcal{K}$.

The control performance measure J. We have to come up a quantity which can somehow measures the performance of a given controller. In controls literature, the LQR cost or some norms (e.g. $\mathcal{H}_2$ or $\mathcal{H}_\infty$ norm) are typically used to measure the controller performance. In machine learning, the cost is typically handcrafted in a case-by-case manner. Sometimes one may need to use inverse reinforcement learning to learn a reward function from experts. Sometimes one has to use domain-expertise to come up a reasonable reward function. How to design J for complicated control tasks is more of an art than a science. When J has an additive structure, i.e. $J = \sum_{t=0}^{\infty} \gamma^t R(x_t, u_t)$, the above control problem just becomes a MDP and reinforcement learning methods can be directly applied. However, there also exist some control measure which are not in additive forms. For example, the $\mathcal{H}_\infty$ norm is an important performance metric in robust control but it is not a sum of rewards at different time steps. In principle, one can still calculate $\nabla J(K)$ and use the gradient information to find the optimal K. However, how to estimate $\nabla J(K)$ from data becomes an issue since most gradient estimation methods in reinforcement learning rely on the additive structure of the cost function.

Constrained set $\mathcal{K}$. The constraints will confine the policy search to a feasible set $\mathcal{K}$. Constraints on K are posed to account for either the stability, robustness, safety, or structural concerns on the system. A common example is the stability constraint, i.e. we need the system $x_{t+1} = f(x_t, K(x_t), w_t)$ to be a stable system. Sometimes the stability constraint is hidden in the optimization process since J becomes ∞ when K is not stabilizing. In model predictive control, safety constraints are typically posed to ensure x_t or u_t does not enter certain bad sets. In robust control, robustness constraints are typically posed to ensure stability subject to perturbation. In the reinforcement learning field, risk-related constraints have been considered.