12
Iterative Learning Control

(This lecture is related to the paper **Using Inaccurate Models in Reinforcement Learning** [1]. Reading the paper first is helpful in understanding the material).

In the previous lectures, we have been looking into using policy gradient methods to find a good policy. The main advantage of policy gradient methods is that they require us to know *very little* about the problem we are solving – we don't need to know the transition model, and we don't need to know the reward function either. All we need to do is collect a number of roll-out trajectories and estimate the policy gradient based on that. As a result of that, however, policy gradient methods have their natural limitation that they generally require *a large number* of trajectories to work reasonably well and they sometimes suffer from high variance in their gradient estimates.

In this lecture, we take a different path by assuming that we know *something* about the particular problem we are solving. In particular, we assume that we have a possibly inaccurate but hopefully helpful model of the system and we know the reward function. We will see how we can approach the reinforcement learning problem differently with this additional knowledge.

12.1 Model-based Reinforcement Learning

One straightforward way to solve this problem is to find an optimal policy with respect to the (possibly inaccurate) model that we already have. This idea lays the foundation of model-based reinforcement learning and optimal control. In fact, we have seen an example of model-based reinforcement learning techniques earlier in this class – LQR. It solves for the optimal policy for a linear model and a quadratic reward function – although any practical systems are hardly truly linear.

Given a (possibly time-varying) deterministic model $\hat{f}_t : \mathbb{S} \times \mathbb{A} \to \mathbb{S}$ and a reward function $r : \mathbb{S} \times \mathbb{A} \to \mathbb{R}$, there is a slightly more general way to find a good policy: solve for the policy gradient through back-propagation[1].

Assume that we parameterize our policy π_θ with parameter θ. Then, we can represent the model-based reinforcement learning as a block diagram:

Note that we are using $\hat{s}_t$, $\hat{a}_t$ and $\hat{J}$ here because they are *not* the actual state, action and total reward we would get from running the actual system, rather, they are just what we get from simulating through our approximated model $\hat{f}_t$. Note, however, that s_0 is not approximated because we assume that

[1] Or the adjoint method if you are an optimal controls person.

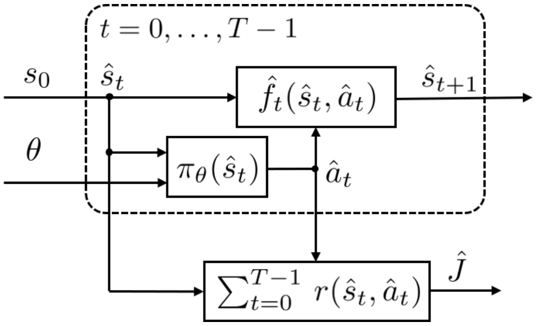


Figure 12.1.1: The block diagram representation of a model-based RL problem using an approximated model.

we start from a fixed state.

Recall from the earlier lecture that we can optimize the policy through the "forward-propagation, back-propagation, gradient ascent" scheme. We initialize our parameter at some arbitrary $\theta^{(0)}$. At i-th iteration, we do:

1. **Forward-propagation:** Forward simulate $\pi_{\theta^{(i)}}$ using the *approximated model* $\hat{f}_t$ and observe the simulated trajectory $\{s_0, a_0^{(i)}\}, \{\hat{s}_1^{(i)}, \hat{a}_1^{(i)}\}, \cdots \{\hat{s}_T^{(i)}, \hat{a}_T^{(i)}\}$ along with the approximated total reward $\hat{J}$.

2. **Back-propagation:** Compute the approximated policy gradient $\nabla_\theta \hat{J}(\theta)$ along the trajectory $\{s_0, a_0^{(i)}\}, \{\hat{s}_1^{(i)}, \hat{a}_1^{(i)}\}, \cdots \{\hat{s}_T^{(i)}, \hat{a}_T^{(i)}\}$ using back-propagation.

3. **Gradient-Ascent:** Update the parameter $\theta^{(i+1)} = \theta^{(i)} + \alpha \nabla_\theta \hat{J}(\theta)$.

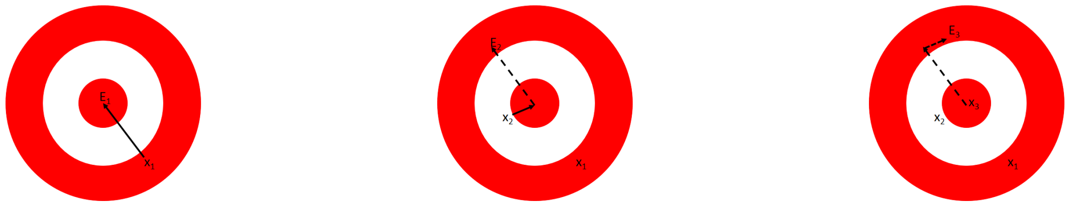
Note that the approximated policy gradient $\nabla_\theta \hat{J}(\theta)$ we used here is fundamentally different from the estimated policy gradient $\widetilde{\nabla}_\theta J(\theta)$ we used for the policy gradient methods. Here we use $\nabla_\theta \hat{J}(\theta)$, which is the *exact* gradient of the *approximated* total reward function, while $\widetilde{\nabla}_\theta J(\theta)$ is the *estimated* gradient for the *exact* total reward function.

12.2 *Iterative Learning Control*

A typical problem in model-based reinforcement learning is that no matter how well you try to model the system dynamics, there are always unmodeled errors that can easily throw your controller off course. A concrete example is given in [1], where the authors want to control an RC car to follow some trajectory. It is shown that the carpet threading is enough to cause their linearized system to drift away from the planned trajectory.

In contrast, it is against our human intuition that a super sophisticated model is required to perform many tasks such as steering a car. A young adult who only has a crude idea of how a car steers can learn to make good turns after a few trials: if the turn is too wide, steer more next time; if the turn is too tight, steer less next time. This idea is also illustrate in the target example in Fig 12.2.1.

Thus the key idea of Iterative Learning Control is (as the authors state): "...to use a real world trial to *evaluate* a policy but then use the simulator (or model) to estimate the *derivative* of the evaluation with respect to the policy parameters." *In other words, we can use the actual system to do forward propagation and then use our crude model for back propagation.*



(a) Initial shot is off (b) Aim at E_2 instead of E_1 (c) Aim at E_3 and hit the bull's eye

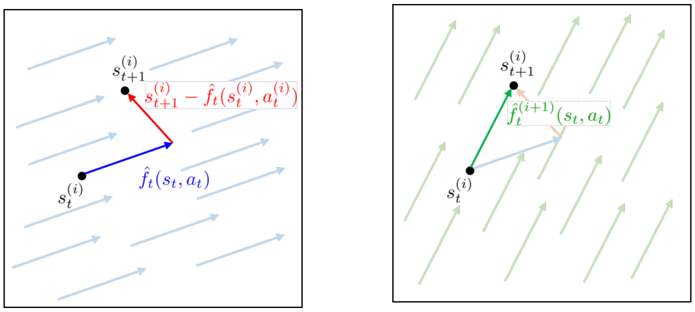
Figure 12.2.1: The target example: (a) Initially we aim at the bull's eye(E_1) but due to wind or miscalibrated sight we end up at x_1. (b) Instead of aiming at E_1 which will end up at x_1, we aim at E_2 which hopefully will end up at E_1. (c) Continue updating the offset until we hit the bull's eye.

The Algorithm

Consider an approximated MDP problem (approximated in a sense that the model isn't very accurate but still informative) $(\mathbb{S}, \mathbb{A}, \hat{f}_t, s_0, r)$, where $\mathbb{S}$ is the set of all possible states, A is the set of all actions, $\hat{f}_t : \mathbb{S} \times \mathbb{A} \rightarrow \mathbb{S}$ is the (possibly time-varying) deterministic approximated transition model, s_0 is the initial state and $r : \mathbb{S} \rightarrow \mathbb{R}$ is the reward function[2]. Assume both the system and the policy are deterministic and the policy is parameterized by θ. We initialize our parameter at $\theta^{(0)}$, the solution to the model-based reinforcement learning problem we saw in the previous part. Then the i-th iteration of the policy gradient proceeds as follows:

1. Execute the current policy $\pi_{\theta^{(i)}}$ on the *real system* and observe the actual trajectory $\{s_0, a_0^{(i)}\}, \{s_1^{(i)}, a_1^{(i)}\}, \cdots \{s_T^{(i)}, a_T^{(i)}\}$.
2. Augment the model by adding a (time-dependent) bias term to the original model at every time step t: $\hat{f}_t^{(i+1)}(s,a) = \hat{f}_t(s,a) + \left(s_{t+1}^{(i)} - \hat{f}_t(s_t^{(i)}, a_t^{(i)})\right)$.
3. Compute policy gradient $\nabla_\theta J(\theta)$ using back-propagation with the updated model and then update the parameter $\theta^{(i+1)} = \theta^{(i)} + \alpha \nabla_\theta J(\theta)$.

In each iteration i, adding the time-dependent bias terms corrects the old model so that if we re-run it with $\pi_{\theta^{(i)}}$ and $\hat{f}^{(i+1)}(s,a)$ we would get the exact same state-action sequence $\{s_0, a_0^{(i)}\}, \{s_1^{(i)}, a_1^{(i)}\}, \cdots \{s_T^{(i)}, a_T^{(i)}\}$.


$s_{t+1}^{(i)}$
$s_{t+1}^{(i)} - \hat{f}_t(s_t^{(i)}, a_t^{(i)})$
$s_t^{(i)}$
$\hat{f}_t(s_t, a_t)$
$s_{t+1}^{(i)}$
$\hat{f}_t^{(i+1)}(s_t, a_t)$
$s_t^{(i)}$

(a) The original model $\hat{f}_t$. (b) The augmented model $\hat{f}_t^{(t+1)}$.

Figure 12.2.2: At each iteration, we augment the model by adding a (time-dependent) bias term to the original model so that we would get the same trajectory.

Therefore when updating the parameters θ in step 3, the correct trajectory is used for computing the policy gradient. In most nonlinear control systems,

[2] Note that here that we assume that we know the *true* reward function. Note also that the reward is only defined on states in this paper but one can also define it as a function of both states and actions.

this means using the actual trajectory for the linearization points though the derivatives are computed using the old model (bias terms do not affect derivatives) at these correct trajectory points.

12.3 The Theory

Once again we assume the system is deterministic and assume our policy is parameterized by θ. Define the following function $s_t = h_t(s_0, \theta)$:

$$h_1(s_0, \theta) = s_1 = f_0(s_0, \pi_\theta(s_0)) \tag{12.3.1}$$

$$h_t(s_0, \theta) = f_{t-1}(s_{t-1}, \pi_\theta(s_{t-1})) \tag{12.3.2}$$

$$= f_{t-1}(h_{t-1}(s_0, \theta), \pi_\theta(h_{t-1}(s_0, \theta))) \tag{12.3.3}$$

In other words, $h_t(s_0, \theta)$ is the *real world* state at time t if we start at s_0 and follow the policy π_θ. Similarly we can define $\hat{s_t} = \hat{h}_t(s_0, \theta)$ which is the state at time t using the approximated model and following π_θ.

Let $s_0, s_1, \cdots s_T$ be the *real world* state sequence obtained when executing the policy π_θ. Then the true policy gradient is given by:

$$\nabla_\theta J(\theta) = \sum_{t=0}^{T} \nabla_{s_t} r(s_t) \frac{dh_t}{d\theta}\Big|_{s_0, s_1, \cdots s_{T-1}} \tag{12.3.4}$$

Note here that the derivatives $\frac{dh_t}{d\theta}$ are total derivatives since h_t is dependent on θ through all previous time steps $t' = 0, \ldots, t-1$. The chain rule (back-propagation) is applied to every term in $\frac{dh_t}{d\theta}$ by the definition of Eq. 12.3.3.

Similarly we can define the approximated policy gradient as follows:

$$\nabla_\theta \hat{J}(\theta) = \sum_{t=0}^{T} \nabla_{\hat{s}_t} r(\hat{s}_t) \frac{d\hat{h}_t}{d\theta}\Big|_{s_0, \hat{s}_1, \cdots \hat{s}_{T-1}} \tag{12.3.5}$$

Two sources of error make Eq. 12.3.5 differ from the the true policy gradient in Eq. 12.3.4:

1. The *derivative* in $\frac{d\hat{h}_t}{d\theta}$ is based on an inaccurate model.
2. The *derivatives* in both $\nabla_{\hat{s}_t} r(\hat{s}_t)$ and $\frac{d\hat{h}_t}{d\theta}$ are evaluated along the wrong trajectory.

What ILC does is that although we cannot deal with the first source of error, we can at least run the system to get the actual trajectory instead of using the wrong one predicted by our approximate model. The resulting gradient is thus:

$$\nabla_\theta \hat{J}(\theta) = \sum_{t=0}^{T} \nabla_{s_t} r(s_t) \frac{d\hat{h}_t}{d\theta}\Big|_{s_0, s_1, \cdots s_{T-1}} \tag{12.3.6}$$

Brief Proof of Convergence and Optimality

It has been proved that if the model isn't too bad and the problem is well-behaved enough[3] then following the gradient will converge to a neighborhood of a local optimum. More formally:

$$\left\| \frac{df_t}{ds} - \frac{d\hat{f}_t}{ds} \right\|_2 \le \epsilon \text{ and } \left\| \frac{df_t}{da} - \frac{d\hat{f}_t}{da} \right\|_2 \le \epsilon \Rightarrow \left\| \nabla_\theta \hat{J} - \nabla_\theta J \right\|_2 \le K\epsilon, \tag{12.3.7}$$

[3] Certain boundedness and smoothness conditions hold for the true MDP.

where K is a constant related to the properties of the problem, such as the dimensionality of the problem, upper bound for reward, horizon of the problem, etc.

Further, if an exact line search is done for each gradient ascent step (every gradient ascent step updates the parameter to the best parameter along the gradient direction), the algorithm converges to a region of local optimality,

$$\|\nabla_\theta J\|_2 \leq \sqrt{2}K\epsilon. \tag{12.3.8}$$

The above theorem guarantees that ILC converges to a region of local optimality. On the other hand, in practice when the policy is close the true optimal policy, it tends to oscillate without actually converging to the optimum.

12.4 Related Reading

[1] Abbeel, P., Quigley, M. and Ng, A.Y., *Using inaccurate models in reinforcement learning.*, ICML 2006.

,