Adaptive Control and Reinforcement Learning (16-899C, S09) Lecture #21 (4/07/09)

Conservative Policy Iteration (CPI)

Lecturer: Bagnell *Scribe: Eric Whitman*

1 Policy Search by Dynamic Programming

- fixed time horizon - rarely what we want - so not often useful
- performance bounds (tightest known)
- does not require the bellman equation - can be used to make it faster - overly general

Quote about Optimality: "An optimal policy hast he property that whatever the initial decision may be, the remaining decisions constitute an optimal policy with regard to ... [the resulting state]."

Implication: You can work backwards from the last time step without considering anything earlier, and without losing optimality.

1.1 Bounds

Assume reward $\in [0, 1]$

Define:

$V_{\pi_0 \ldots \pi_{T-1}} = \frac{1}{T} E[r(s_0) + r(s_1) \ldots + r(s_{T-1}) | \pi_0 \ldots \pi_{T-1}] \in [0, 1]$

Bellman: $V_{\pi_t \ldots \pi_{T-1}}(s) = \frac{1}{T} r(s) + E_{s'}[V_{\pi_t \ldots \pi_{T-1}}(s')]$

where s' is the result of π_t

$Q_a, \pi_{t+1}, \pi t + 2, ..., \pi T - 1(s)$ fix policy at t to choose a, then follow the policy from then on.

$\mu_t(s) = p(s_t)$ probability in state, need one distribution for each time step - used to focus the approximation effort

$\mu_t(s)$ is "similar" to the $p(s_t | \pi^*)$

μ_t comes from:

- trajectory
- teacher
- other policy, π^{adequate}
- μ close to flat

1.2 Procedure

for each $t \in (T-1, T-2, ..., 1, 0)$

1) $d_t = \text{generatesamples}(\pi_{t+1}, ..., \pi_{T-1}, \mu)$

2) $\pi_L = \text{Learn}(d_t)$

1.2.1 Details on step 1 - generating samples

Generate samples of the form (policy sequence, μ)

many (k) times:

sample s from μ

for each action:

$\tilde{Q}_{s,a} = \text{rollout}(a, \vec{\pi})$

return $\{s_i, \tilde{Q}_{s_i,a1}, \tilde{Q}_{s_i,a2}, \tilde{Q}_{s_i,a3}, ...\}_{i=1:k}$

1.2.2 Details on Rollout

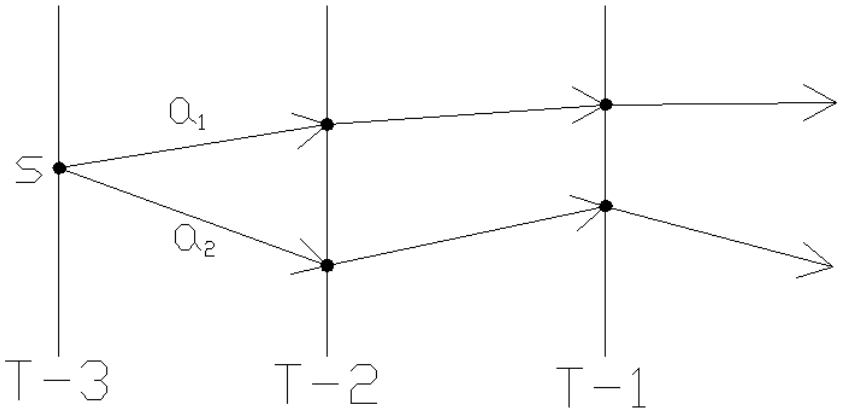

Figure 1: Figure 1: Rollout

Starting from state s at time t, perform all available actions. Run the existing policies from each or the resulting states. The cumulative rewards starting from state s and performing action i is represented by $\tilde{Q}_{s,a_i}$.

Stochastic Policies There is never a reason for a stochastic policy in finite horizon. This is not necessarily true in the infinite horizon case or in adversarial situations. For non-adversarial finite time horizon problems, time varying policies are better than stochastic policies.

Argument1: For the last time step, the expected value of the stochastic policy is the the average of the values of what the policy might do. The deterministic policy picks the best one, which must be

at least as good as the average. We can inductively make the same argument about the previous time step all the way back to the beginning.

Argument2: (Probablistic method) We can look at a stochastic policy as taking the state and another input ($\pi(s, \text{rand})$). A deterministic policy just picks the best rand. This argument is similar to the mean value theorum.

1.2.3 Details of step 2 - Learning

Learn(d)

find $\pi_t^* = \arg\max_{\pi' \in \Pi} E_d[V_{\pi', \pi_{t+1}, ..., \pi_{T-1}}(s)]$

Brute Force
for π' in Π
compute $\Sigma_i \tilde{Q}^i_{s_i, \pi'(s_i)}$
pick biggest

Learn Classifier(d)

Felix: $d' = \{(s_i, \arg\max_a \tilde{Q}_{s_i,a})\}$
return $\pi = \text{classifier}(d')$

doesn't understand importance of particular samples

breaks ties in bad ways, which can be important

Learn Weighted Classifier(d)

$d' = (s_i, \tilde{Q}_{s_i,a^*} - \tilde{Q}_{s_i,a_1}, \tilde{Q}_{s_i,a^*} - \tilde{Q}_{s_i,a_2}, ...)_{i=1:k}$
return $\pi = \text{classifier}(d')$

1.2.4 Regression

Regression is harder than classification (worse performance for the same number of examples).

Learn Regression(d)

$d' = (s_i, a_1, \tilde{Q}_{s_i,a_1})$
$(s_i, a_2, \tilde{Q}_{s_i,a_2})$
$\text{reg} = \text{regression}(d')$

$\pi = \arg\max_a \text{reg}(s, a)$

1.3 Bounds

$V_{\pi \text{ learn}}(s_0) \geq V_{\pi_{ref}}(s_0) - \epsilon \Sigma_t || \frac{\mu_{\text{ref},t}}{\mu_t} ||_\infty$

ϵ is the error of the classifier, $\epsilon = p(\text{wrong})$

π_{ref} is usually π^*.

implication: err on the side of broadness

Can't get bounds for other methods.

2 Compare to Value Iteration

start at last time step
get some reward
learn function $V_{T-1}(s)$ based on samples
samples: (s,reward)

effort is O(T). caches values at each timestep - linear work

goes wrong with error, need to do max over actions $Q_{s,a}$

Q will make errors (function approximator). These will accumulate as previous steps head towards areas that they thought were good but are really bad

2.1 Policy Search by Dynamic Programming on the other hand

Only cache policy - much safer.

However rollout takes more time than looking up values.

Effort is O(T^2). A factor of T worse than Value Iteration.

There are non-parametric local minima - local minima in terms of states.

It assumes μ, so it does no global exploration, just local exploration.

"If willing to give up exploration, we can do function approximation."