Topics in machine learning: Naresh Manwani

Siddharth Bhat

Monsoon 2019

Contents

Chapter 1

Policy iteration

$$\pi_{k+1}(s) = \arg\max_{a \in A(s)} r(s,a) + \gamma \sum_{s} P(s'|s,a) v_{\pi_k}(s')$$

Theorem 1 *The policy iteration algorithm generates a sequecnce of policies with non-decreasing state values. That is,* $V^{\pi_{k+1}} \geqslant V^{\pi_k}$, $V^{\pi} \in \mathbb{R}^n$, *is the vector of state values for state* π

Proof 1 F^{π_k} *is the bellman expectation operator (?)*
Since V^{π_k} *is a fixed point of* F^{π_k},

$$\begin{aligned}
V^{\pi_k} &= F^{\pi_k}(V^{\pi_k}) \leqslant F(V^{\pi_k}) && \textit{(upper bounded by max value)} \\
F(V^{\pi_k}) &= F^{\pi_{k+1}}(V^{\pi_k}) && \textit{(By defn of policy improvement step)} \\
V^{\pi_k} &\leqslant F^{\pi_{k+1}}(V^{\pi_k}) && \textit{(eqn 1)} \\
F^{\pi_{k+1}}(V^{\pi_k}) &\leqslant (F^{\pi_{k+1}})^2(V^{\pi_k}) && \textit{(Monotonicity of } F^{\pi_{k+1}}\textit{)} \\
\forall t \geqslant 1,\ F^{\pi_{k+1}}(V^{\pi_k}) &\leqslant (F^{\pi_{k+1}})^t(V^{\pi_k}) && \textit{(Monotonicity of } F^{\pi_{k+1}}\textit{)} \\
F^{\pi_{k+1}}(V^{\pi_k}) &\leqslant (F^{\pi_{k+1}})^t(V^{\pi_k}) \leqslant V^{\pi_{k+1}} && \textit{(Contraction mapping, } V^{\pi_{k+1}} \textit{ is fixed point)} \\
V^{\pi_k} &= F^{\pi_{k+1}}(V^{\pi_k}) \leqslant V^{\pi_{k+1}}
\end{aligned}$$

For a set of actions $\mathcal{A}$ and a set of states $\mathcal{S}$, the total number of policies is $|\mathcal{A}^{\mathcal{S}}|$. The number of computations per iteration is $O(|\mathcal{S}|^3)$. So the loose upper bound is be $O(|\mathcal{S}|^3 \times |\mathcal{A}^{\mathcal{S}}|)$.

1.1 Value iteration algorithm

```
let v n s = max [r s a + gamma * sum [(p s' s a) * v (n-1) s' | s' <- ss] | a <- as]
let vs = [v i | i <- [0..]]
-- | L infinity
let norm v v' = max [(v s - v' s) | s <- ss]
let out = head $
  dropWhile (\v v' -> norm (v' - v) < eps * (1 - gamma) / (2 * gamma)) $
  zip vs (tail vs)
let policy s = argmax as $ \a ->
  r s a + gamma * sum [ (p s' s a) * out s' | s' <- ss]
```

Theorem 2 *For the series V_n and the policy π_ϵ computed by the value iteration algorithm, then:*

$$\forall \epsilon > 0,\ \exists n_0 \in \mathbb{N}, \forall n \geqslant n_0, \quad \|V_{n+1} - V_n\|_\infty \leqslant \frac{\epsilon(1-\gamma)}{2\gamma}$$

Proof 2 *We need to show that the sequence $\{V_n\}_{n=0}^{\infty}$ is a Cauchy sequence. This has ben proven before by the use of contraction mapping. Thus, for a given $\epsilon' \geqslant 0, \exists n_0 \in \mathbb{N}, \forall n \geqslant n_0, \|V_{n+1} - V_n\|_\infty \leqslant \epsilon'$ by cauchy sequence. So, pick $\epsilon' = \frac{\epsilon(1-\gamma)}{2\gamma}$, and the proof immediately follows.*

Theorem 3 *If $\|V_{n+1} - V_n\|_\infty \leqslant \frac{\epsilon(1-\gamma)}{2\gamma}$, then $\|V_{n+1} - V^\star\|_\infty < \epsilon/2$*

Proof 3

$$\|V_{n+1} - V^\star\| = \|V_{n+1} - FV_{n+1} + FV_{n+1} - V^\star\| \leqslant \|V_{n+1} - FV^\star\| + \|FV_n - V_n\| \qquad \textit{(triangle inequality)}$$
$$\leqslant \|V_{n+1} - FV^\star\| + \gamma\|V_{n+1} - V^\star\|$$
$$\leqslant \gamma\|V_{n+1} - V_n\| + \gamma\|V_{n+1} - V^\star\|$$
$$(1-\gamma)\|V_{n+1} - V^\star\| \leqslant \gamma\|V_n - V_{n+1}\| \qquad \textit{(how?)}$$
$$\Longrightarrow \ldots$$

It appears that V^{π_ϵ} is just V_{n+2}??

Theorem 4 *The policy π_ϵ is ϵ-optimal: $\|V^\star - V^{\pi_\epsilon}\| \leqslant \epsilon$*

Chapter 2

Monte carlo methods for MDP

For dynamic programming, we needed to know the transition probability distribution $P(s, a, s')$, nor the reward function $r(s, a)$.

In the monte carlo methods, we assume that we do not know the transition probability distribution. We rely only on simulations.

This samples over *episodes* for a fixed policy: sequences of states, actions, and rewards.

2.1 Naive

$$\mathsf{Episode}_i(E_i) \equiv S_0^i \to A_0^i \to R_1^i \to S_1^i \to A_1^i \to R_2^i \cdots \to S_{T_i}$$

Episode E_i terminates at T_i.

- Let us define G_i to be the reward of E_i. $G_i \equiv \sum_{k=0}^{T_i} \gamma^k R_{k+1}^i$
- Estimate the value of π starting from s as $\hat{v}_\pi(s) = \frac{1}{m} \sum_{i=1}^{m} G_i$.
- Show by chernoff bounds that this is an OK estimate. We can use Chernoff as $\{G_i\}$ are independent, since the episodes $\{E_i\}$ are independent.

- *First-visit MC*: Average returns for the first time s is visited in an episode
- *Every-visit MC*: Average returns for every time s is visited in an episode.

Both of these asymptotally converge to the correct v_π.

2.1.1 First visit monte carlo policy evaluation

Run π from a fixed state s_0 for m times. This gives us m episodes. The ith episode is E_i, which terminates at step T_i.

$G(s, E_i)$ is defined as the return of π in run E_i, starting from the time instant of the first appearance of s in E_i till the final state. If state s occurs at time t_s, then $G(s, E_i) \equiv \sum_{j=t_s}^{T_i} \gamma^{(j-t_s)} R_{j+1}^i$. We start from $j+1$ since we want all rewards *after* our state. Note that the reward R_j is the reward granted *before* transitioning to the state S_j.

The value of a state s under the policy π is defined as:

$$\hat{v}_\pi \equiv \frac{1}{m} \sum_{i=1}^{m} G(s, E_i)$$

2.1.2 Every visit monte carlo

Every time we visit a state, we are able to find the return starting from that state.

```
-- | discount factor
gamma = 0.9

-- | an episode is a list of states,
-- with a possible (action,reward) pair
-- generating the next state
type Episode = [(S, Maybe (A, R))]

-- | Note the use of ParallelListComprehension!
reward :: Episode -> R
reward es = sum $ [gamma^i * r | i <- [1..] | (_, Just(_, r)) <- es]

-- | Return all possible tails, in order of longest
-- to shortest subsequence.
-- > tails[1, 2, 3] = [[1, 2, 3], [2, 3], [3], []]
tails :: Episode -> [Episode]
tails [] = []
tails xs = xs:tails (tail xs)

-- | Find the longest subsequence with start state s0
-- and calculate its reward. This assumes that
-- tails returns the longer subsequences first
firstVisit :: State -> Episode -> R
firstVisit s0 episodes =
    case dropWhile (\e -> fst <$> headMaybe e /= Just s0) (tails episodes) of
        (e:_) -> reward e
        _ -> 0

-- | Find all subsequences with start state s0, and calculate
-- their rewards
everyVisit :: State -> Episode -> R
everyVisit s0 episodes = sum $ do
    e <- tails episodes
    return $ if fst <$> headMaybe e /= Just s0 then 0 else reward e
```

For each run E_i and state s_0, let $G(s_0, E_i, j)$ be the return of π in the run E_i for the jth occurence of s_0 in E_i. Let $N_i(s)$ be the number of times state s has occured in episode E_i.

$$\hat{v}_\pi(s) = \frac{1}{\sum_{i=1}^{m} N_i(s)} \sum_{i=1}^{m} \sum_{j=1}^{N_i(s)} G(s, E_i, j)$$

Note that $G(s, E_i, j)$ for a fixed state s and E_i is a dependent variable for different j. That is, $G(s, E_i, 1)$ is dependent on $G(s, E_i, 0)$.

We also want $q_{pi}(s, a)$: The expected return starting from state s, taking an action a, and then following te policy π. (evaluation uses v, updates involve q).

2.2 Soft policy

Since our policy is deterministic, we cannot explore all state-action pairs. Therefore, we make our policy softly non-deterministic, by allowing transitions to all states with proabability ϵ. This allows us to explore all state-action pairs.

- On-policy: Use same policy to generate the episode and update the policy.
- Off-policy: Use some policy to generate the episode, and update a different policy.

$$\pi : S \times A \to \mathbb{R}$$
$$\pi(s|a) = \begin{cases} 1 - \epsilon + \frac{\epsilon}{|A|} & \text{if } a = a^* \\ \frac{\epsilon}{|A|} & \text{otherwise} \end{cases}$$

2.3 On policy first-visit MC control, for ϵ-soft policy

```
Q :: State -> Action -> Prob
Q s a = random

Returns :: State -> Action -> Real
Returns s a = 0


pi :: State -> Action -> Prob
pi = an arbitrary epsilon soft policy

- generate a new episode using pi
- for each (s, a) in the episode,
   G <- return following the first occurence of (s, a)
   append G to Returns(s, a)
   Q(s, a) <- average(Returns(s, a))

- for each s in the episode: A* <- arg max a Q(s, a)
- pi <- a new epsilon soft policy based on A*
```

Because the optimal policy for a finite MDP is deterministic, we choose to keep the policy slightly away from deterministic: This way, we get to explore the space, while still being optimal.

We will prove that this actually does improve the policy. Let the new policy be π_{k+1}, and the current policy be π_k.

$$\begin{aligned} q_\pi(s, \pi_{k+1}(a)) &= \sum_{a \in A(s)} \pi_{k+1}(a|s) q_{\pi_k}(s, a) \\ &= (1-\epsilon) \max_{q_{\pi_k}}(s, a) + \sum_{a \in A(s)} \frac{\epsilon}{|A(s)|} q_{\pi_k}(s, a) \\ &= (1-\epsilon) q_{\pi_k}(s, a^*) + \sum_{a \in A(s)} \frac{\epsilon}{|A(s)|} q_{\pi_k}(s, a) \end{aligned}$$

Note that $\pi'(a) = \frac{\pi_k(a|s) - \frac{\epsilon}{|A(s)|}}{1-\epsilon} \geqslant 0$. Also note that $\sum_a \pi'(a) = 1$. Now, since $avg \leqslant \max$, $\sum_a \pi'(a|s) q_{\pi_k}(s, a) \leqslant q_{\pi_k}(s, a^*)$.

$$\begin{aligned} &= (1-\epsilon) q_{\pi_k}(s, a^*) + \sum_{a \in A(s)} \frac{\epsilon}{|A(s)|} q_{\pi_k}(s, a) \\ &\geqslant (1-\epsilon) \sum_a \pi'(a|s) q_{\pi_k}(s, a) + \sum_{a \in A(s)} \frac{\epsilon}{|A(s)|} q_{\pi_k}(s, a) \\ &= (1-\epsilon) \left[\frac{\sum_a \left[\pi_k(a|s) - \frac{\epsilon}{|A(s)|}\right]}{1-\epsilon} \right] q_{\pi_k}(s, a) + \sum_{a \in A(s)} \frac{\epsilon}{|A(s)|} q_{\pi_k}(s, a) \\ &= \sum_a \pi_k(a|s) q_{\pi_k}(s, a) \qquad \left(\text{Cancellation of } \sum_{a \in A(s)} \frac{\epsilon}{|A(s)|} q_{\pi_k}(s, a) \right) \\ &= v_{\pi_k}(s) \end{aligned}$$

2.4 Off policy

We can use another policy μ to generate data, and we estimate q_π based on μ. Here, π is called the target policy, and μ is called the behaviour policy.

We need μ to *cover* $\pi : \pi(a|s) > 0 \implies \mu(a|s) > 0$. That is, every action taken by π must have non-zero proability to be taken by μ. That is, we can only choose to take actions that were taken by μ, since we can only really learn from the things that μ has done.

2.5 Importance sampling

We want to find $\mathbb{E}_q[x]$, for a distribution q that is hard to sample. We can only sample a distribution p that is easy to sample. What we do is to define a new random variable $\hat{x} = x\frac{q(x)}{p(x)}$. We know how to draw samples from $p(x)$ efficiently, so we create the $\hat{x}$ such that $\hat{x} \sim p \implies x \sim q$.

$$\mathbb{E}_p[\hat{x}] = \mathbb{E}_p\left[x\frac{q(x)}{p(x)}\right] = \sum_x p(x) \cdot x\frac{q(x)}{p(x)} = \sum_x xq(x) = \mathbb{E}_q[x]$$

2.6 Off-policy monte-carlo control

We take an episode that was sampled according to policy μ, and we use importance sampling to perturb it into a policy π.

For a given policy π, we define the probability of a subsequence of an episode occuring under policy π as $P_\pi(t,T) \equiv \prod_{k=t}^{T-1} \pi(A_k|S_k)p(S_{k+1}|A_k,S_k)$. Now, to importance sample, we will get:

$$\rho(t,T) \equiv \frac{\prod_{k=t}^{T-1} \pi(A_k|S_k)p(S_{k+1}|A_k,S_k)}{\prod_{k=t}^{T-1} \mu(A_k|S_k)p(S_{k+1}|A_k,S_k)} = \prod_{k=t}^{T-1} \frac{\pi(A_k|S_k)}{\mu(A_k|S_k)}$$

$$\mathbb{E}_{A_k \sim \mu(A_k|S_k)}\left[f(A_k)\frac{\pi(A_k|S_k)}{\mu(A_k|S_k)}\right] = E_{A_k \sim \pi(A_k|S_k)}[f(A_k)]$$

We want to use $f(A_k) = G_t$ That is, we want to importance sample the *reward* of a trajectory under μ and perturb it to reward under π.