University of Illinois Fall 2018

ECE 544NA: Pattern Recognition
Lecture 1: August 28

Lecturer: Alexander Schwing Scribe: YOUR FULL NAME

1 Recap

This section is review of MDP and Q-learning.

1.1 MDP

Markov Decision Process (MDP)

1. A set of states $s \in S$
2. A set of actions $a \in A_s$
3. A transition probability $P(s|s,a)$
4. A reward function $R(s,a,s)$ (sometimes just R(s) or R(s))
5. A start and maybe a terminal state

1.1.1 MDP Goal

We want to perform actions according to a policy π^* so as to maximize the expected future reward. At a given state, we want to know which action to take to maximize the future rewards.

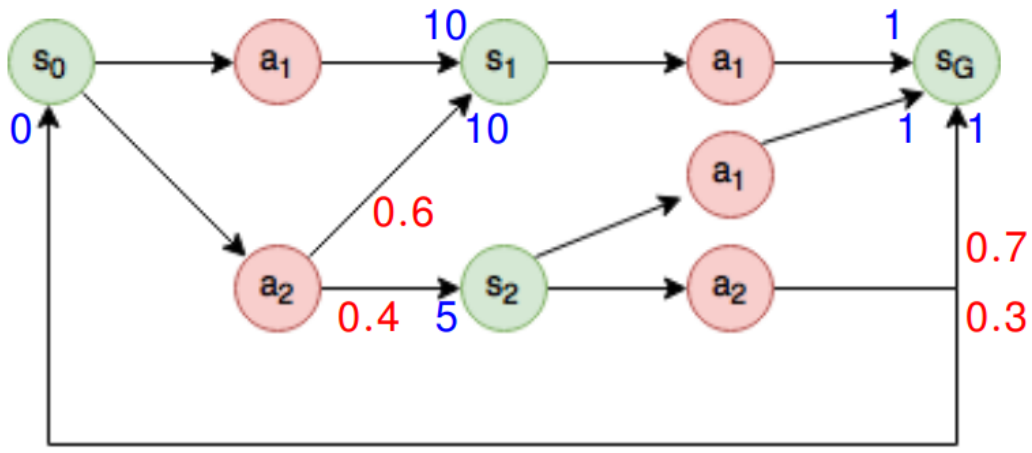
$$\pi(s) : S \rightarrow A_s$$

1.1.2 Methods to maximize expected rewards

1. Exhaustive search
2. Policy iteration
3. Value iteration

I. Exhaustive search:
Number of policies: $\prod_{s \in S} |A_s|$
Compute expected future rewards: $V^{\pi}(s_0)$
Choose optimized policy π^* to get largest future rewards $V^{\pi^*}(s_0)$

II. Policy iteration:
Example:



For policy $\pi(s0) = a1$**,** $\pi(s1) = a1$**:**
Policy graph:

$V^{\pi}(s_1) = 1$, $V^{\pi}(s_0) = 11$

For policy $\pi(s0) = a1$**,** $\pi(s1) = a1$**:**
Policy graph:

$V^{\pi}(s_1) = 1$, $V^{\pi}(s_2) = 1$, $V^{\pi}(s_0) = 0.6 \times (10 + 1) + 0.4 \times (5 + 1) = 9$

For policy $\pi(s0) = a2$**,** $\pi(s2) = a2$**:**
Policy graph:

$V^\pi(s_1) = 1$, $V^\pi(s_2) = 0.7 \times 1 + 0.3 \times V^\pi(s_0)$, $V^\pi(s_0) = 0.4 \times (5 + V^\pi(s_2)) + 0.6 \times (10 + V^\pi(s_1))$
By solving the linear system, we could get $V^\pi(s_0) = 10$, $V^\pi(s_2) = 3.7$
Policy Evaluation:

$$V^\pi(s) = \sum_{s' \in S} P(s'|s, \pi(s))[R(s, \pi(s), s') + V^\pi(s')]$$

Procedure:

1. Initialize policy π
2. Repeat until policy π does not change:
   Solve system of equations (e.g., iteratively)

   $$V^\pi_{i+1}(s) \leftarrow \sum_{s' \in S} P(s'|s, \pi(s))[R(s, \pi(s), s') + V^\pi_i(s')]$$

   Extract new policy π by using:

   $$\pi(s) = argmax_{a \in A_s} \sum_{s' \in S} P(s'|s, \pi(s))[R(s, \pi(s), s') + V^\pi(s')]$$

III. Value iteration

1. Changes search space (search over values, not over policies)
2. Compute the resulting policy at the end

Bellman optimality principle:

$$V^*(s) = \max_{a \in A_s} \sum_{s' \in S} P(s'|s, a)[R(s, a, s') + V^\pi_i(s')]$$

Iterative principle:

$$V_{i+1}(s) = \max_{a \in A_s} \sum_{s' \in S} P(s'|s, a)[R(s, a, s') + V_i(s')]$$

1.2 Q-learning

1.2.1 Bellman optimality principle

$$Q^*(s, a) = \sum_{s' \in S} P(s'|s, a)[R(s, a, s') + \max_{a' \in A_{s'}} Q^*(s', a')]$$

Idea: Run a simulator to collect experience tuples/samples (s,a,r,s').
Approximate transition probability using samples.

1.2.2 Algorithm sketch for Q-learning

1. Obtain a sample transition (s, a, r, s)
2. Obtained sample suggests:

   $$Q(s, a) \approx y(s, a, r, s') = r + \max_{a' \in A_{s'} Q(s', a')}$$

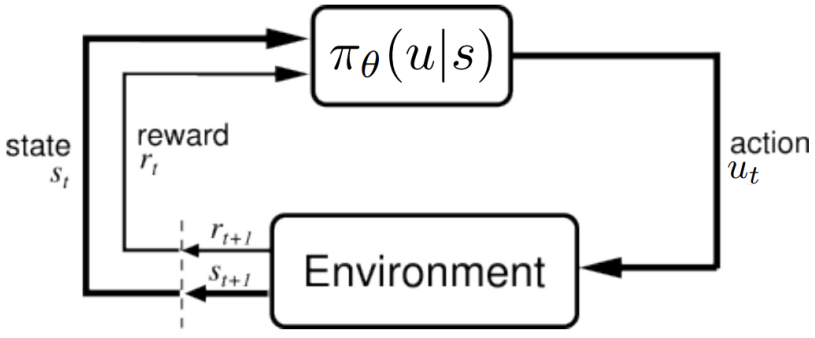
3. To account for missing transition probability we keep running average

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha y(s,a,r,s')$$

Difference between MDP and Q-learning: Evaluate fixed policy π, MDP uses policy evaluation while Q-learning uses value learning.

2 Policy Gradient

The general framework of reinforcement learning encompasses a broad variety of problems ranging from various forms of function optimization at one extreme to learning control at the other. While research in these individual areas tends to emphasize different sets of issues in isolation, it is likely that effective reinforcement learning techniques for autonomous agents operating in realistic environments will have to address all of these issues jointly. Thus while it remains a useful research strategy to focus on limited forms of reinforcement learning problems simply to keep the problems tractable, it is important to keep in mind that eventual solutions to the most challenging problems will probably require integration of a broad range of applicable techniques. [2]
In Q-learning, For a given state(S), we optimize future expected reward Q. In policy gradient, we directly optimize parametric policy $\pi_\theta(a|s)$.
For large-scale problems or problems where the system dynamics are unknown, the performance gradient will not be computable in closed form1. Thus the challenging aspect of the policy-gradient approach is to find an algorithm for estimating the gradient via simulation. Naively, the gradient can be calculated numerically by adjusting each parameter in turn and estimating the effect on performance via simulation (the so-called crude Monte-Carlo technique), but that will be prohibitively inefficient for most problems. Somewhat surprisingly, under mild regularity conditions, it turns out that the full gradient can be estimated from a single simulation of the system. The technique is called the score function or likelihood ratio method and appears to have been first proposed in the sixties (Aleksandrov, Sysoyev, & Shemeneva, 1968; Rubinstein, 1969) for computing performance gradients in i.i.d. (independently and identically distributed) processes. [1]
The following figure shows the relationship between action and the environmental setups. We have rollouts(known as policies), and the agent play the policies and get the future reward. We add the future reward and set of actions into the environment, this is a step of optimization.



Why we choose to use policy optimization?

1. π may be simpler than Q or V;
   E.g., robotic grasp;

2. V doesnt prescribe actions;
   would need dynamics model + Bellman back-up needed;

3. need to be able to efficiently solve $\arg\max_u Q_\theta(s, u)$;
   Challenge for continuous / high-dimensional action spaces* ;

4. Q requires efficient maximization: issue in continuous/high-dimensional action spaces;

5. Function Q gives you reward for all future steps, but not include current state R.

What is the advantage of using replay-memory?
When training the model, allowing the model to learn from earlier memories. This can help with convergence to speed up learning and learning from previous training step can break undesirable temporal correlations. At each step, a random batch of experiences are sampled from the buffer, storing all the previous experiences, to update models parameters. As a result, is that experience replay increases both data usage and computation efficiency.

2.1 Variant: Likelihood ratio policy gradient

1. Rollout, state-action sequence: = (s0, a0, s1, a1, . . .)
2. Expected reward: $R(\tau) = \sum_t R(s_t, a_t)$

$$U(\theta) = E[\sum_t R(s_t, a_t); \pi_\theta] = \sum_\tau P(\tau; \theta) R(\tau)$$

Goal:

$$\max_\theta U(\theta) = \max_\theta \sum_\tau P(\tau; \theta) R(\tau)$$

$$\begin{aligned}
\partial_\theta U(\theta) &= \partial_\theta \sum_\tau P(\tau; \theta) R(\tau) \\
&= \sum_\tau \partial_\theta P(\tau; \theta) R(\tau) \\
&= \sum_\tau P(\tau; \theta) \frac{\partial_\theta P(\tau; \theta)}{P(\tau; \theta)} R(\tau) \\
&= \sum_\tau \partial_\theta \log P(\tau; \theta) R(\tau) \\
\partial_\theta U(\theta) \approx g &= \frac{1}{m} \sum_{i=1}^{m} \partial_\theta \log P(\tau^i; \theta) R(\tau^i)
\end{aligned} \tag{1}$$

Note:
$P(\tau; \theta)$ is impossible to solve since it is emperical it won't have θ inside, which is the parameter we want to optimize. So we turn to use $\log P$.
The final $U(\theta)$ we derive is also valid even if R is discontinuous.
Note similarity to maximum likelihood.
Increase probability of paths τ with **positive** R.
Decrease probability of paths τ with **negative** R.

To solve equation(1), we need to take derivative of $\log P$.

$$
\begin{aligned}
\partial_\theta \log P(\tau;\theta) &= \partial_\theta \log[\prod_t P(s_{t+1|s_t,a_t})\pi_\theta(a_t|s_t)] \\
&= \partial_\theta[\sum_t \log P(s_{t+1|s_t,a_t}) + \sum_t \log \pi_\theta(a_t|s_t)] \\
&= \partial_\theta \sum_t \log \pi_\theta(a_t|s_t) \\
&= \sum_t \partial_\theta \log \pi_\theta(a_t|s_t) \\
\partial_\theta \log(\tau;\theta) &= \sum_t \partial_\theta \log \pi_\theta(a_t|s_t)
\end{aligned} \tag{2}
$$

Consequently, equation(1) becomes :

$$
\partial_\theta U(\theta) = \frac{1}{m}\sum_{i=1}^{m}(\sum_t \partial_\theta \log \pi_\theta(a_t^i|s_t^i))R(\tau^i)
$$

Intuitively, we know that this equation would give reward to those policy with positive reward while giving less weight to negative reward. But what if we do not have negative reward? Introduce baseline argument.

2.1.1 Baseline:

Issue when $R(\tau^i) > 0$ can be fixed.

$$
\partial_\theta U(\theta) = \frac{1}{m}\sum_{i=1}^{m}(\sum_t \partial_\theta \log \pi_\theta(a_t^i|s_t^i))(R(\tau^i) - b)
$$

Why we could add a baseline into the equation without having any impact on result?

$$
\begin{aligned}
\mathbb{E}(\partial_\theta \log P(\tau;\theta)) &= \sum_\tau P(\tau;\theta)\partial_\theta \log P(\tau;\theta) b \\
&= \sum_\tau \partial_\theta P(\tau;\theta) b \\
&= \partial_\theta(\sum_\tau P(\tau;\theta)) b \\
&= 0
\end{aligned} \tag{3}
$$

In Conclusion, this adding baseline b will not influence the result. A good choice of b is:

$$
b = \mathbb{E}[R(\tau)] = \frac{1}{m}\sum_{i=1}^{m} R(\tau^i)
$$

2.1.2 Temporal Structure

$$
\partial_\theta U(\theta) = \frac{1}{m}\sum_{i=1}^{m}(\sum_t \partial_\theta \log \pi_\theta(a_t^i|s_t^i))(\sum_t R(s_t^i, a_t^i) - b)
$$

Future actions do not depend on past rewards: lower variance via

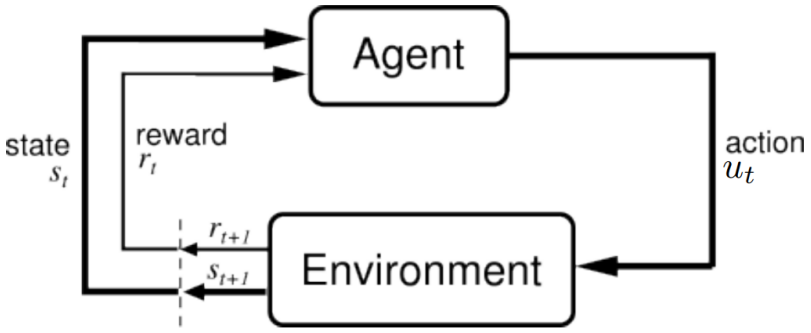
$$\partial_\theta U(\theta) = \frac{1}{m}\sum_{i=1}^{m}(\sum_t \partial_\theta \log \pi_\theta(a_t^i|s_t^i))(\sum_{\hat{t}\geq t} R(s_t^i, a_t^i) - b(s_{\hat{t}}^i))$$

Good choices for b is:

$$b(s_t) = \mathbb{E}[r_t + t_{t+1} + ...]$$

2.2 Algorithm

Reinforcement learning aka vanilla policy gradient.



1. Initial θ, b
2. For iteration = 1, 2, ...
   (a) Collect a set of trajectories $\tau^{(i)}$ by executing policy π_θ
   (b) Compute reward and bias
   $$R_t^{(i)} = \sum_{\hat{t}\geq t} \gamma^{\hat{t}-t} r_{\hat{t}}$$
   (c) Re-fit the baseline b
   (d) Update the policy using the policy gradient estimate $\hat{g}$

2.2.1 Implementation

$$\partial_\theta V_\theta(s0) \approx \sum_{t=1}^{T}\sum_{gt}[\pi_\theta(g_t|s_t)\partial_\theta \log \pi_\theta(g_t|s_t) \times (Q_\theta(s_t, g_t) - B_\phi(s_t))^2]$$

$$L_\phi = \sum_t E_{s_t} E_{g_t}(Q_\theta(s_t, g_t) - B_\phi(s_t))^2$$

2.3 Other Methods

Those methods below are derivative free.

Algorithm 1 PG training algorithm

Input: $D = (x^n, y^n) : n = 1 : N$;
Train $\pi_\theta(g1 : T|x)$ using MLE on D;
Train B_ϕ using MC estimates of Q_θ on a small subset of D;
for each epoch **do**
  for example (x^n, y^n) **do**
    Generate sequence $g1 : T$ $\pi_\theta(.|x^n)$
    for t = 1 : T **do**
    Compute $Q(g_{1:t-1}, g_t)$ for g_t with K
    Monte Carlo rollouts;
    Compute estimated baseline $B_\phi(g_{1:t-1})$
    end for
    Compute $G_\theta = \partial_\theta V_\theta$
    Compute $G_\phi = \partial_\phi L_\phi$
    SGD update of θ using G_θ
    SGD update of ϕ using G_ϕ
  end for
end for

2.3.1 cross-entropy Method

$$\max_\theta U(\theta) = \max_\theta \mathbb{E}[\sum_t R(s_t)|\pi_\theta]$$

CEM

for iter = 1, 2, ... **do**
  for population member e = 1, 2, ... **do**
    sample θ^e $P_{\mu^i}(\theta)$
    execute rollouts under $\pi_\theta(e)$
    store $(\theta^{(e)}, U(e))$
  end for
  $\mu^{i+1} = \arg\max_\mu \sum_e \log P_\mu(\theta^{(e)})$
  where e index over top p
end for

References

[1] J. Baxter and P. L. Bartlett. Infinite-horizon policy-gradient estimation. *Journal of Artificial Intelligence Research*, 15:319–350, 2001.

[2] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.