10-301/10-601 Introduction to Machine Learning

Machine Learning Department
School of Computer Science
Carnegie Mellon University

Reinforcement Learning: MDPs + Value Iteration

Matt Gormley
Lecture 22
Apr. 5, 2023

Reminders

- Homework 7: Hidden Markov Models
  - Out: Fri, Mar. 31
  - Due: Mon, Apr. 10 at 11:59pm

MARKOV DECISION PROCESSES

RL: Components

From the Environment (i.e. the MDP)

- State space, $\mathcal{S}$
- Action space, $\mathcal{A}$
- Reward function, $R(s,a)$, $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$
- Transition probabilities, $p(s' \mid s, a)$
  - Deterministic transitions:

$$p(s' \mid s, a) = \begin{cases} 1 \text{ if } \delta(s,a) = s' \\ 0 \text{ otherwise} \end{cases}$$

where $\delta(s,a)$ is a transition function

Markov Assumption

$$p(s_{t+1} \mid s_t, a_t, \ldots, s_1, a_1) = p(s_{t+1} \mid s_t, a_t)$$

From the Model

- Policy, $\pi : \mathcal{S} \to \mathcal{A}$
- Value function, $V^{\pi} : \mathcal{S} \to \mathbb{R}$
  - Measures the expected total payoff of starting in some state s and *executing* policy π

Markov Decision Process (MDP)

- For **supervised learning** the **PAC learning framework** provided assumptions about where our data came from:

$$\mathbf{x} \sim p^*(\cdot) \text{ and } y = c^*(\cdot)$$

- For **reinforcement learning** we assume our data comes from a **Markov decision process** (MDP)

Markov Decision Processes (MDP)

In RL, the source of our data is an MDP:

1. Start in some initial state $s_0 \in \mathcal{S}$
2. For time step t:
   1. Agent observes state $s_t \in \mathcal{S}$
   2. Agent takes action $a_t \in \mathcal{A}$ where $a_t = \pi(s_t)$
   3. Agent receives reward $r_t \in \mathbb{R}$ where $r_t = R(s_t, a_t)$
   4. Agent transitions to state $s_{t+1} \in \mathcal{S}$ where $s_{t+1} \sim p(s' \mid s_t, a_t)$
3. Total reward is $\sum_{t=0}^{\infty} \gamma^t r_t$
   – The value γ is the "discount factor", a hyperparameter $0 < \gamma < 1$

- Makes the same Markov assumption we used for HMMs! The next state only depends on the current state and action.
- *Def.*: we **execute** a policy π by taking action $a = \pi(s)$ when in state s

RL: Objective Function

- Goal: Find a policy $\pi : \mathcal{S} \to \mathcal{A}$ for choosing "good" actions that maximize:

$$\mathbb{E}[\text{total reward}] = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right]$$

- The above is called the
  "finite horizon expected future discounted reward"

- Can we define other notions of optimality?

EXPLORATION VS. EXPLOITATION

MDP Example: Multi-armed bandit

Single state: $|\mathcal{S}| = 1$

Three actions: $\mathcal{A} = \{1, 2, 3\}$

Deterministic transitions

Rewards are stochastic

MDP Example: Multi-armed bandit

Single state: $|\mathcal{S}| = 1$

Three actions: $\mathcal{A} = \{1, 2, 3\}$

Deterministic transitions

Rewards are stochastic

Bandit 1	Bandit 2	Bandit 3
???	???	???
???	???	???
???	???	???
???	???	???
???	???	???
???	???	???
???	???	???
???	???	???
???	???	???
???	???	???
???	???	???
???	???	???

FIXED POINT ITERATION

Fixed Point Iteration for Optimization

- Fixed point iteration is a general tool for solving systems of equations
- It can also be applied to optimization.

$$J(\boldsymbol{\theta})$$

$$\frac{dJ(\boldsymbol{\theta})}{d\theta_i} = 0 = f(\boldsymbol{\theta})$$

$$0 = f(\boldsymbol{\theta}) \Rightarrow \theta_i = g(\boldsymbol{\theta})$$

$$\theta_i^{(t+1)} = g(\boldsymbol{\theta}^{(t)})$$

1. Given objective function:
2. Compute derivative, set to zero (call this function f).
3. Rearrange the equation s.t. one of parameters appears on the LHS.
4. Initialize the parameters.
5. For i in $\{1,\ldots,K\}$, update each parameter and increment t:
6. Repeat #5 until convergence

Fixed Point Iteration for Optimization

- Fixed point iteration is a general tool for solving systems of equations
- It can also be applied to optimization.

$$J(x) = \frac{x^3}{3} + \frac{3}{2}x^2 + 2x$$

$$\frac{dJ(x)}{dx} = f(x) = x^2 - 3x + 2 = 0$$

$$\Rightarrow x = \frac{x^2 + 2}{3} = g(x)$$

$$x \leftarrow \frac{x^2 + 2}{3}$$

1. Given objective function:
2. Compute derivative, set to zero (call this function f).
3. Rearrange the equation s.t. one of parameters appears on the LHS.
4. Initialize the parameters.
5. For i in $\{1,\ldots,K\}$, update each parameter and increment t:
6. Repeat #5 until convergence

Fixed Point Iteration for Optimization

We can implement our example in a few lines of python.

$$J(x) = \frac{x^3}{3} + \frac{3}{2}x^2 + 2x$$

$$\frac{dJ(x)}{dx} = f(x) = x^2 - 3x + 2 = 0$$

$$\Rightarrow x = \frac{x^2 + 2}{3} = g(x)$$

$$x \leftarrow \frac{x^2 + 2}{3}$$

```python
def f1(x):
    '''f(x) = x^2 - 3x + 2'''
    return x**2 - 3.*x + 2.

def g1(x):
    '''g(x) = \frac{x^2 + 2}{3}'''
    return (x**2 + 2.) / 3.

def fpi(g, x0, n, f):
    '''Optimizes the 1D function g by fixed point iteration
    starting at x0 and stopping after n iterations. Also
    includes an auxiliary function f to test at each value.'''
    x = x0
    for i in range(n):
        print("i=%2d x=%.4f f(x)=%.4f" % (i, x, f(x)))
        x = g(x)
    i += 1
    print("i=%2d x=%.4f f(x)=%.4f" % (i, x, f(x)))
    return x

if __name__ == "__main__":
    x = fpi(g1, 0, 20, f1)
```

Fixed Point Iteration for Optimization

$$J(x) = \frac{x^3}{3} + \frac{3}{2}x^2 + 2x$$

$$\frac{dJ(x)}{dx} = f(x) = x^2 - 3x + 2 = 0$$

$$\Rightarrow x = \frac{x^2 + 2}{3} = g(x)$$

$$x \leftarrow \frac{x^2 + 2}{3}$$

```
$ python fixed-point-iteration.py
i= 0 x=0.0000 f(x)=2.0000
i= 1 x=0.6667 f(x)=0.4444
i= 2 x=0.8148 f(x)=0.2195
i= 3 x=0.8880 f(x)=0.1246
i= 4 x=0.9295 f(x)=0.0755
i= 5 x=0.9547 f(x)=0.0474
i= 6 x=0.9705 f(x)=0.0304
i= 7 x=0.9806 f(x)=0.0198
i= 8 x=0.9872 f(x)=0.0130
i= 9 x=0.9915 f(x)=0.0086
i=10 x=0.9944 f(x)=0.0057
i=11 x=0.9963 f(x)=0.0038
i=12 x=0.9975 f(x)=0.0025
i=13 x=0.9983 f(x)=0.0017
i=14 x=0.9989 f(x)=0.0011
i=15 x=0.9993 f(x)=0.0007
i=16 x=0.9995 f(x)=0.0005
i=17 x=0.9997 f(x)=0.0003
i=18 x=0.9998 f(x)=0.0002
i=19 x=0.9999 f(x)=0.0001
i=20 x=0.9999 f(x)=0.0001
```

VALUE ITERATION

Definitions for Value Iteration

Whiteboard

- Optimal policy
- State trajectory
- Value function
- Bellman equations
- Optimal value function
- Computing the optimal policy
- Ex: Path Planning

Example: Path Planning

RL: Optimal Value Function & Policy

- Optimal value function:

$$V^*(s) = \max_{a \in \mathcal{A}} R(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s,a) V^*(s')$$

  – System of $|\mathcal{S}|$ equations and $|\mathcal{S}|$ variables

- Optimal policy:

$$\pi^*(s) = \operatorname*{argmax}_{a \in \mathcal{A}} \underbrace{R(s,a)}_{\text{Immediate reward}} + \underbrace{\gamma \sum_{s' \in \mathcal{S}} p(s' \mid s,a) V^*(s')}_{\text{(Discounted) Future reward}}$$

RL Terminology

Question: Match each term (on the left) to the corresponding statement or definition (on the right)

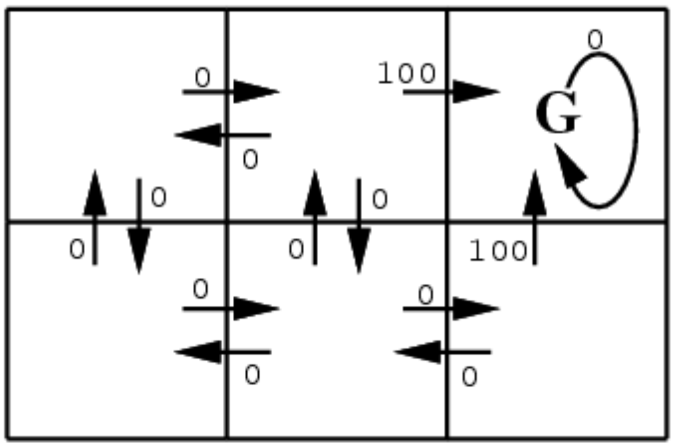
Terms:

A. a reward function
B. a transition probability
C. a policy
D. state/action/reward triples
E. a value function
F. transition function
G. an optimal policy
H. Matt's favorite statement

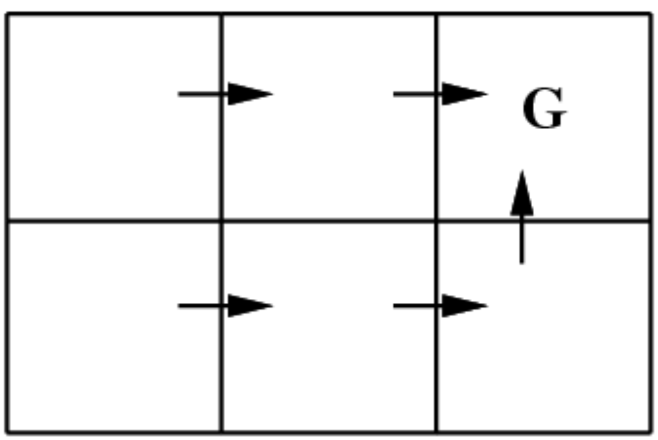
Statements:

1. gives the expected future discounted reward of a state
2. maps from states to actions
3. quantifies immediate success of agent
4. is a deterministic map from state/action pairs to states
5. quantifies the likelihood of landing a new state, given a state/action pair
6. is the desired output of an RL algorithm
7. can be influenced by trading off between exploitation/exploration

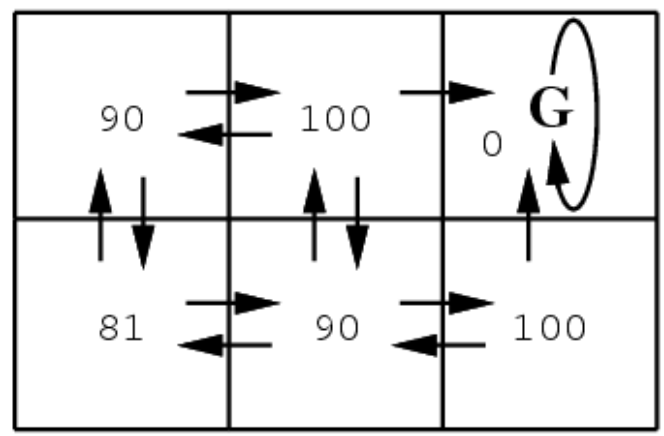
Example: Robot Localization



$r(s, a)$ (immediate reward) values



One optimal policy



$V^*(s)$ values

Figure from Tom Mitchell

Value Iteration

Whiteboard

– Value Iteration Algorithm

Value Iteration

Algorithm 1 Value Iteration

procedure VALUEITERATION($R(s,a)$ reward function, $p(\cdot|s,a)$ transition probabilities)
    Initialize value function $V(s) = 0$ or randomly
    while not converged **do**
        for $s \in \mathcal{S}$ **do**
            $V(s) = \max_a R(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V(s')$
    Let $\pi(s) = \operatorname{argmax}_a R(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V(s'), \;\forall s$
    return π

Variant 1: without Q(s,a) table

Value Iteration

Algorithm 1 Value Iteration

procedure VALUEITERATION($R(s,a)$ reward function, $p(\cdot|s,a)$ transition probabilities)
    Initialize value function $V(s) = 0$ or randomly
    while not converged **do**
        for $s \in \mathcal{S}$ **do**
            for $a \in \mathcal{A}$ **do**
                $Q(s,a) = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a)V(s')$
            $V(s) = \max_a Q(s,a)$
    Let $\pi(s) = \operatorname{argmax}_a Q(s,a), \ \forall s$
    return π

Variant 2: with Q(s,a) table

Synchronous vs. Asynchronous Value Iteration

Algorithm 1 Asynchronous Value Iteration

procedure AsynchronousValueIteration($R(s,a), p(\cdot|s,a)$)
  Initialize value function $V(s) = 0$ or randomly
  while not converged **do**
    for $s \in \mathcal{S}$ **do**
      $V(s) = \max_a R(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a)V(s')$
  Let $\pi(s) = \text{argmax}_a R(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a)V(s'), \ \forall s$
  return π

asynchronous updates: compute and update V(s) for each state one at a time

Algorithm 1 Synchronous Value Iteration

procedure SynchronousValueIteration($R(s,a), p(\cdot|s,a)$)
  Initialize value function $V(s)^{(0)} = 0$ or randomly
  $t = 0$
  while not converged **do**
    for $s \in \mathcal{S}$ **do**
      $V(s)^{(t+1)} = \max_a R(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a)V(s')^{(t)}$
    $t = t + 1$
  Let $\pi(s) = \text{argmax}_a R(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a)V(s'), \ \forall s$
  return π

synchronous updates: compute all the fresh values of V(s) from all the stale values of V(s), then update V(s) with fresh values

Value Iteration Convergence

very abridged

Theorem 1 (Bertsekas (1989))
V converges to V^*, if each state is visited infinitely often

Theorem 2 (Williams & Baird (1993))
if $max_s|V^{t+1}(s) - V^t(s)| < \epsilon$
then $max_s|V^{t+1}(s) - V^*(s)| < \frac{2\epsilon\gamma}{1-\gamma}, \forall s$

Theorem 3 (Bertsekas (1987))
greedy policy will be optimal in a finite number of steps (even if not converged to optimal value function!)

Holds for both asynchronous and sychronous updates

Provides reasonable stopping criterion for value iteration

Often greedy policy converges well before the value function