10-301/10-601 Introduction to Machine Learning

Machine Learning Department
School of Computer Science
Carnegie Mellon University

Reinforcement Learning:
Value Iteration & Policy Iteration

Matt Gormley & Henry Chai

Lecture 21

Mar. 31, 2025

Reminders

- **Homework 7: Deep Learning**
  - Out: Wed Mar-26
  - Due: Wed Apr-09 at 11:59pm
- **Homework 8: Deep RL**
  - Out: Wed Apr-09
  - Due: Wed Apr-16 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

Exploration vs. Exploitation Tradeoff

- In RL, there is a **tension** between two strategies an agent can follow when interacting with its environment:
  - **Exploration**: the agent takes actions to visit (state, action) pairs it has not seen before, with the hope of uncovering previously unseen high reward states
  - **Exploitation**: the agent takes actions to visit (state, action) pairs it knows to have high reward, with the goal of maximizing reward given its current (possibly limited) knowledge of the environment
- Balancing these two is critical to success in RL!
  - If the agent **only explores**, it performs no better than a random policy
  - If the agent **only exploits**, it will likely never discover an optimal policy
- One approach for trading off between these:
  the ϵ-greedy policy

RL: Objective Function

- Goal: Find a policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$ 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
  "infinite horizon expected future discounted reward"

finite horizon

$$\sum_{t=0}^{h} \gamma^t r_t$$

penalty avoidance

$$\sum_{t=0}^{\infty} \gamma^t \max(0, r_t)$$

no discounting

$$\sum_{t=0}^{h} r_t$$

regret from leaving the known good path

$$\sum_{t=0}^{\infty} \left(\gamma^t r_t + f(s_t)\right)$$

Reinforcement Learning: Objective Function

Objective Function

- Find a policy $\pi^* = \underset{\pi}{\operatorname{argmax}}\ V^\pi(s)\ \forall\, s \in \mathcal{S}$
- Assume stochastic transitions and deterministic rewards
- $V^\pi(s) = \mathbb{E}[$*discounted* total reward of starting in state s and executing policy π forever$]$

$$= \mathbb{E}_{p(s' \mid s,a)}[R(s_0 = s, \overbrace{\pi(s_0)}^{a_0})$$

$$+\ \gamma R(s_1, \underbrace{\pi(s_1)}_{a_1}) + \gamma^2 R(s_2, \underbrace{\pi(s_2)}_{a_2}) + \cdots]$$

$$= \sum_{t=0}^{\infty} \gamma^t \mathbb{E}_{p(s' \mid s,a)}[R(s_t, \pi(s_t))]$$

where $0 < \gamma < 1$ is some discount factor for future rewards

Deriving the Bellman Equations

(recursive definition of $V^\pi(s)$)

$$V^\pi(s) = R(s, \pi(s)) + \gamma \sum_{s' \in S} p(s' \mid s, a) \Big[R(s', \pi(s')) + \gamma \sum_{s'' \in S} p(s'' \mid s', a') \big[R(s'', \pi(s'')) + \gamma \sum_{s''' \in S} p(s''' \mid s'', a'') [R(s''', \pi(s''')) + \cdots] \big] \Big]$$

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

RL: Optimal Value Function & Policy

- Bellman Equations:

$$V^{\pi}(s) = R(s, \pi(s)) + \gamma \sum_{s' \in S} p(s' \mid s, \pi(s)) V^{\pi}(s')$$

- Optimal policy:
  - Given V^*, $R(s,a)$, $p(s' \mid s,a), \gamma$ we can compute this!

$$\pi^*(s) = \underset{a \in A}{\operatorname{argmax}} \underbrace{R(s,a)}_{\text{immediate reward}} + \underbrace{\gamma \sum_{s' \in S} p(s' \mid s,a) V^*(s')}_{\text{expected future discounted reward}}$$

- Optimal value function:

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

  - System of $|S|$ equations and $|S|$ variables (each variable is some $V^*(s)$ for some state s)
  - Can be written without π^* $\quad V^{\pi^*}(s) = R(s, \pi^*(s)) + \gamma \sum_{s' \in S} p(s' \mid s, \pi^*(s)) V^{\pi^*}(s')$

S	V^*	π^*
s_1	9	N
s_2	-2	E
s_3	7	E
$\vdots$		
s_{99}	1	W
s_{100}	0	S

FIXED POINT ITERATION

Fixed Point Iteration

$$f_1(x_1, \ldots, x_n) = 0$$
$$\vdots$$
$$f_n(x_1, \ldots, x_n) = 0$$

$$x_1 = g_1(x_1, \ldots, x_n)$$
$$\vdots$$
$$x_n = g_n(x_1, \ldots, x_n)$$

$$x_1^{(t+1)} = g_1(x_1^{(t)}, \ldots, x_n^{(t)})$$
$$\vdots$$
$$x_n^{(t+1)} = g_n(x_1^{(t)}, \ldots, x_n^{(t)})$$

- Fixed point iteration is a general tool for solving systems of equations
- Under the right conditions, it will converge

1. Assume we have n equations and n variables, written f(**x**) = 0 where **x** is a vector
2. Rearrange the equations s.t. each variable x_i has one equation where it is isolated on the LHS
3. Initialize the ~~parameters~~. variables
4. For i in {1,…,n}, update each parameter and increment *t*:
5. Repeat ~~#3~~ until convergence
   #4

Fixed Point Iteration

$$\cos(y) - x = 0$$
$$\sin(x) - y = 0$$

$$x = \cos(y)$$
$$y = \sin(x)$$

$$x^{(t+1)} = \cos(y^{(t)})$$
$$y^{(t+1)} = \sin(x^{(t)})$$

- Fixed point iteration is a general tool for solving systems of equations
- Under the right conditions, it will converge

1. Assume we have n equations and n variables, written f(**x**) = 0 where **x** is a vector
2. Rearrange the equations s.t. each variable x_i has one equation where it is isolated on the LHS
3. Initialize the parameters.
4. For i in {1,…,n}, update each parameter and increment *t*:
5. Repeat #5 until convergence

Fixed Point Iteration

$$\cos(y) - x = 0$$
$$\sin(x) - y = 0$$

$$x = \cos(y)$$
$$y = \sin(x)$$

$$x^{(t+1)} = \cos(y^{(t)})$$
$$y^{(t+1)} = \sin(x^{(t)})$$

We can implement our example in a few lines of code

```
from math import *

def f(x, y):
    eq1 = cos(y) - x
    eq2 = sin(x) - y
    return (eq1, eq2)

def g(x, y):
    x = cos(y)
    y = sin(x)
    return (x, y)

def fpi(x0, y0, n):
    '''Solves the system of equations 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
    y = y0
    for i in range(n):
        ox, oy = f(x,y)
        print("i=%2d x=%.4f y=%.4f f(x,y)=(%.4f, %.4f)" % (i, x, y, ox, oy))
        x,y = g(x,y)
    i += 1
    print("i=%2d x=%.4f y=%.4f f(x,y)=(%.4f, %.4f)" % (i, x, y, ox, oy))
    return x,y

if __name__ == "__main__":
    x,y = fpi(-1, -1, 20)
```

Fixed Point Iteration

```
$ python fixed-point-iteration.py
i= 0 x=-1.0000 y=-1.000 f(x,y)=(1.5403, 0.1585)
i= 1 x=0.5403 y=0.5144 f(x,y)=(0.3303, 0.0000)
i= 2 x=0.8706 y=0.7647 f(x,y)=(-0.1490, 0.0000)
i= 3 x=0.7216 y=0.6606 f(x,y)=(0.0681, 0.0000)
i= 4 x=0.7896 y=0.7101 f(x,y)=(-0.0313, 0.0000)
i= 5 x=0.7583 y=0.6877 f(x,y)=(0.0144, 0.0000)
i= 6 x=0.7727 y=0.6981 f(x,y)=(-0.0066, 0.0000)
i= 7 x=0.7661 y=0.6933 f(x,y)=(0.0031, 0.0000)
i= 8 x=0.7691 y=0.6955 f(x,y)=(-0.0014, 0.0000)
i= 9 x=0.7677 y=0.6945 f(x,y)=(0.0006, 0.0000)
i=10 x=0.7684 y=0.6950 f(x,y)=(-0.0003, 0.0000)
i=11 x=0.7681 y=0.6948 f(x,y)=(0.0001, 0.0000)
i=12 x=0.7682 y=0.6949 f(x,y)=(-0.0001, 0.0000)
i=13 x=0.7681 y=0.6948 f(x,y)=(0.0000, 0.0000)
i=14 x=0.7682 y=0.6948 f(x,y)=(-0.0000, 0.0000)
i=15 x=0.7682 y=0.6948 f(x,y)=(0.0000, 0.0000)
i=16 x=0.7682 y=0.6948 f(x,y)=(-0.0000, 0.0000)
i=17 x=0.7682 y=0.6948 f(x,y)=(0.0000, 0.0000)
i=18 x=0.7682 y=0.6948 f(x,y)=(-0.0000, 0.0000)
i=19 x=0.7682 y=0.6948 f(x,y)=(0.0000, 0.0000)
i=20 x=0.7682 y=0.6948 f(x,y)=(0.0000, 0.0000)
```

We can implement our example in a few lines of code

```python
from math import *

def f(x, y):
    eq1 = cos(y) - x
    eq2 = sin(x) - y
    return (eq1, eq2)

def g(x, y):
    x = cos(y)
    y = sin(x)
    return (x, y)

def fpi(x0, y0, n):
    '''Solves the system of equations 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
    y = y0
    for i in range(n):
        ox, oy = f(x,y)
        print("i=%2d x=%.4f y=%.4f f(x,y)=(%.4f, %.4f)" % (i, x, y, ox, oy))
        x,y = g(x,y)
    i += 1
    print("i=%2d x=%.4f y=%.4f f(x,y)=(%.4f, %.4f)" % (i, x, y, ox, oy))
    return x,y

if __name__ == "__main__":
    x,y = fpi(-1, -1, 20)
```

VALUE ITERATION

Roll Q1

RL Terminology

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

For full credit, select one statement for each term (i.e. one selection per row)

Terms:

A. a reward function 3
B. a transition probability 5
C. a policy 2
D. state/action/reward triples 7
E. a value function 1
F. transition function 4
G. an optimal policy 6

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

RL: Optimal Value Function & Policy

- Bellman Equations:

$$V^\pi(s) = \quad R(s,\pi(s)) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s, \pi(s)) V^\pi(s')$$

- Optimal policy:
  - Given V^*, $R(s,a)$, $p(s' \mid s,a), \gamma$ we can compute this!

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

- Optimal value function:

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

  - System of $|\mathcal{S}|$ equations and $|\mathcal{S}|$ variables (each variable is some $V^*(s)$ for some state s)
  - Can be written without π^*

S	V^*
S_1	$V(S_1)$
S_2	$V(S_2)$
S_3	
S_{100}	

Example: Path Planning

Value Iteration

Algorithm:

① Initialize $V(s)=0 \ \forall s$ (randomly)

② While not converged:

for $s \in S$:

for stochastic transitions: $V(s) = \max_{a \in A} R(s,a) + \gamma \sum_{s' \in S} p(s'|s,a)\, V(s')$

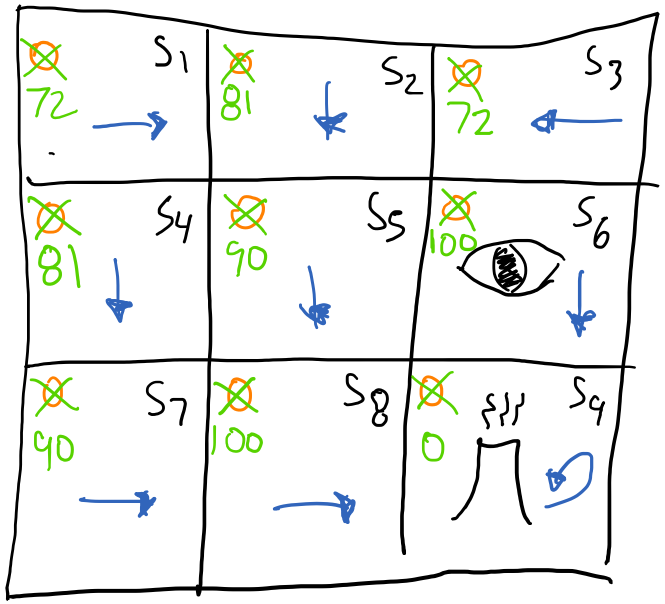
for deterministic transitions: $V(s) = \max_{a \in A} R(s,a) + \gamma\, V(\overbrace{\delta(s,a)}^{s'})$

③ Return

$$\pi^{greedy}(s) = \arg\max_{a \in A} R(s,a) + \gamma \sum_{s' \in S} p(s'|s,a)\, V(s')$$

$$= \arg\max_{a \in A} R(s,a) + \gamma\, V(\delta(s,a))$$

Example:



is terminal

$A = \{\uparrow, \downarrow, \leftarrow, \rightarrow\}$

$R(s,a) = -100$ if entering

$R(s,a) = +100$ if entering

$R(s,a) = 0$

transitions are deterministic

$\gamma = 0.9$

$V(s) = 0$ for $t=1$

$V(s)$ for $t=2$

Value Iteration

Algorithm 1 Value Iteration (deterministic transitions)

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

Variant 1: without Q(s,a) table