



10-301/10-601 Introduction to Machine Learning

Machine Learning Department
School of Computer Science
Carnegie Mellon University

Reinforcement Learning: Value Iteration

Matt Gormley & Geoff Gordon

Lecture 21

Nov. 10, 2025

Reminders

- **Homework 7: Deep Learning & LLMs**
 - **Out: Thu, Nov 6**
 - **Due: Sun, Nov 16 11:59pm**
- **Homework 8: Deep Reinforcement Learning**
 - **Out: Sun, Nov 16**
 - **Due: Mon, Nov 24 11:59pm**

Reinforcement Learning

- Three key challenges
 1. The algorithm has to gather its own training data
 2. The outcome of taking some action is often stochastic or unknown until after the fact
 3. Decisions can have a delayed effect on future outcomes (exploration-exploitation tradeoff)

MARKOV DECISION PROCESSES

RL: Components

From the Environment (i.e. the MDP)

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

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

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

Markov Assumption

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

From the Model

- Policy, $\pi : \mathcal{S} \rightarrow \mathcal{A}$
- Value function, $V^\pi : \mathcal{S} \rightarrow \mathbb{R}$
 - Measures the expected total payoff of starting in some state s and executing policy π
 - Def.: we **execute** a policy π by taking action $a = \pi(s)$ when in state s

RL: Components

(Stochastic Rewards)

From the Environment (i.e. the MDP)

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

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

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

Markov Assumption

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

From the Model

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

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 $s_{t+1} \sim p(s' | s_t, a_t)$
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 a Markov assumption: the next state only depends on the current state and action.

Markov Decision Processes (MDP)

(Stochastic Rewards)

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 $r_t = R(s_t, a_t, s_{t+1})$
 4. Agent transitions to state $s_{t+1} \in \mathcal{S}$ where $s_{t+1} \sim p(s' | s_t, a_t)$
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

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”
- Can we define other notions of optimality?

Reinforcement Learning: Objective Function

Objective Function

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

$$= \mathbb{E}_{p(s' | s, a)} [R(s_0 = s, \pi(s_0)) + \gamma R(s_1, \pi(s_1)) + \gamma^2 R(s_2, \pi(s_2)) + \dots]$$

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

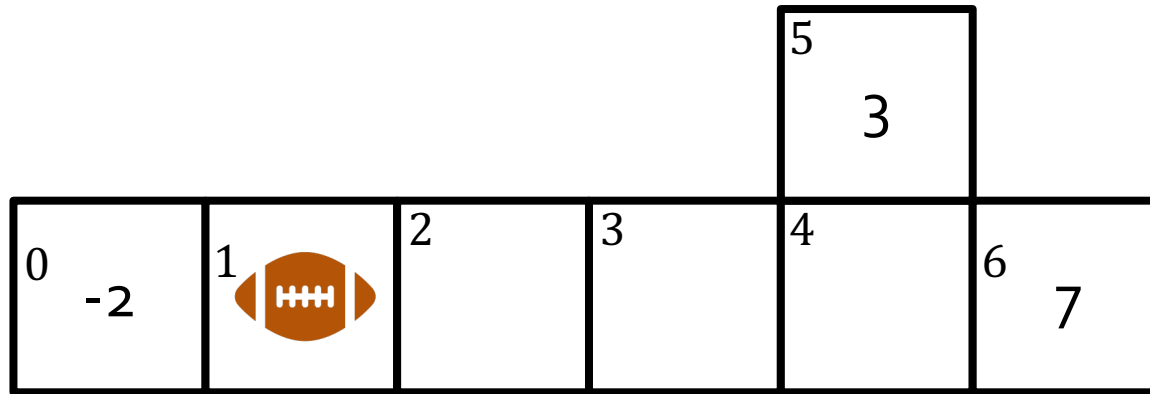
where $0 < \gamma < 1$ is some discount factor for future rewards

Deriving the Bellman Equations

RL: Optimal Value Function & Policy

- Bellman Equations:
- Optimal policy:
 - Given V^* , $R(s, a)$, $p(s' | s, a)$, γ we can compute this!
- Optimal value function:
 - System of $|\mathcal{S}|$ equations and $|\mathcal{S}|$ variables (each variable is some $V^*(s)$ for some state s)
 - Can be written without π^*

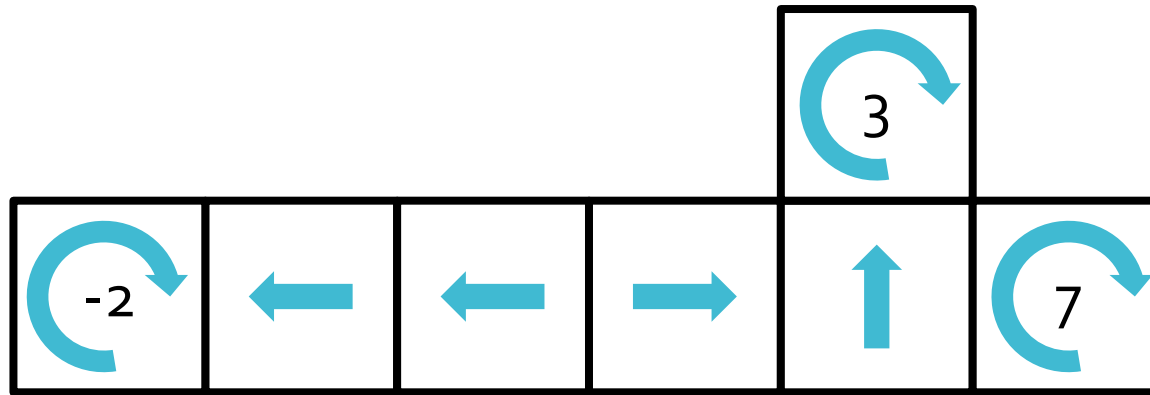
RL: Value Function Example



$$R(s, a) = \begin{cases} -2 & \text{if entering state 0 (safety)} \\ 3 & \text{if entering state 5 (field goal)} \\ 7 & \text{if entering state 6 (touch down)} \\ 0 & \text{otherwise} \end{cases}$$

$$\gamma = 0.9$$

RL: Value Function Example

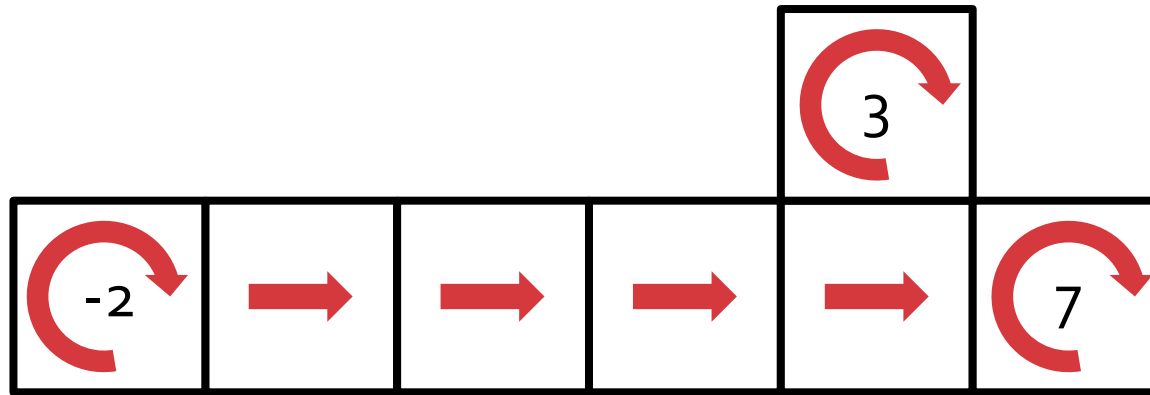


$$R(s, a) = \begin{cases} -2 & \text{if entering state 0 (safety)} \\ 3 & \text{if entering state 5 (field goal)} \\ 7 & \text{if entering state 6 (touch down)} \\ 0 & \text{otherwise} \end{cases}$$

$$\gamma = 0.9$$

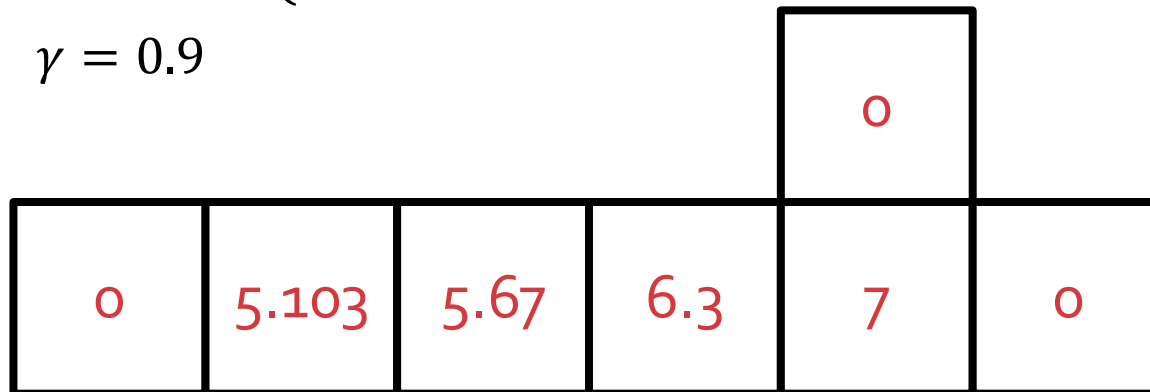


RL: Value Function Example



$$R(s, a) = \begin{cases} -2 & \text{if entering state 0 (safety)} \\ 3 & \text{if entering state 5 (field goal)} \\ 7 & \text{if entering state 6 (touch down)} \\ 0 & \text{otherwise} \end{cases}$$

$$\gamma = 0.9$$



STOCHASTIC REWARDS AND VALUE ITERATION

Optimal Value Function

(Stochastic Rewards)

For the optimal policy function π^* we can compute its **value function** as:

$$\begin{aligned} V^{\pi^*}(s) &= V^*(s) \\ &= \mathbb{E}[R(s_0, \pi^*(s_0), s_1) + \gamma R(s_1, \pi^*(s_1), s_2) \\ &\quad + \gamma^2 R(s_2, \pi^*(s_2), s_3) \cdots \mid s_0 = s, \pi^*]. \end{aligned}$$

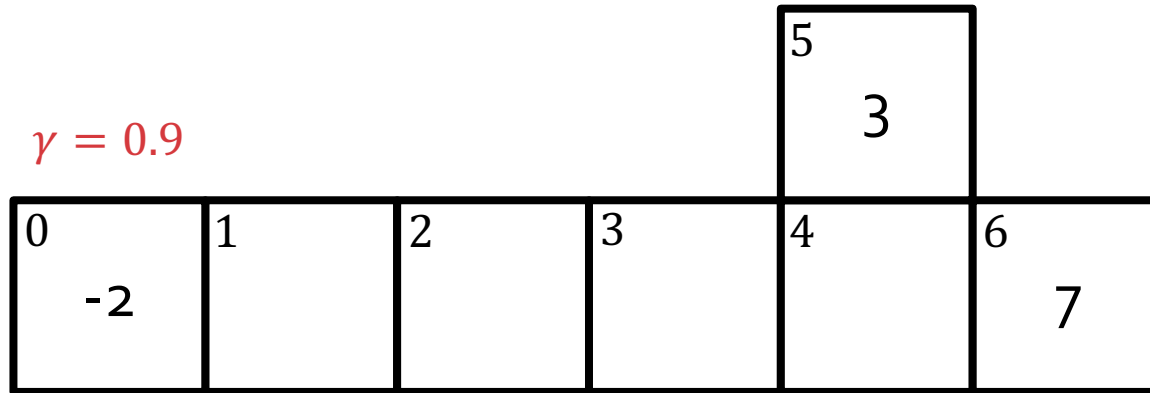
This **optimal value function** can be represented recursively as:

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

If $R(s, a, s') = R(s, a)$ (deterministic reward), then we have the form:

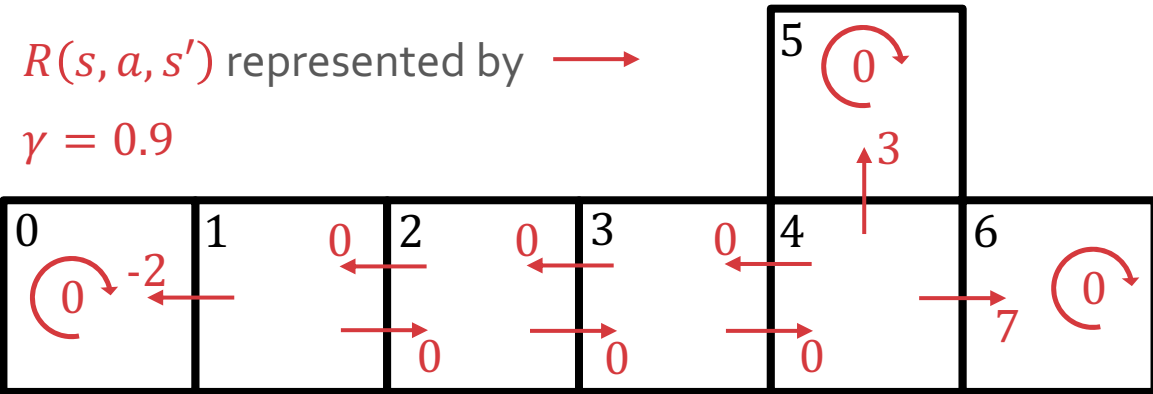
$$V^*(s) = \max_{a \in \mathcal{A}} \left\{ R(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s, a) V^*(s') \right\}.$$

Example: Stochastic Transitions and Rewards



$$R(s, a, s') = \begin{cases} -2 & \text{if entering state 0 (safety)} \\ 3 & \text{if entering state 5 (field goal)} \\ 7 & \text{if entering state 6 (touch down)} \\ 0 & \text{otherwise} \end{cases}$$

Example: Stochastic Transitions and Rewards

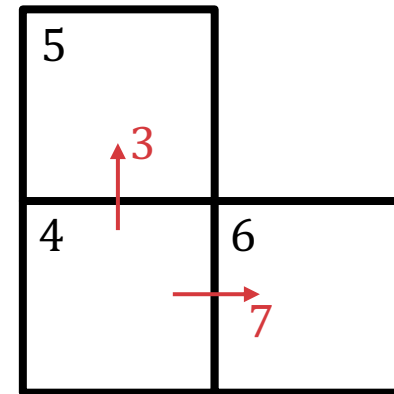


Suppose

- $p(s_6 | s_4, \rightarrow) = 0.8$
- $p(s_5 | s_4, \rightarrow) = 0.2$
- $p(s_6 | s_4, \uparrow) = 0.1$
- $p(s_5 | s_4, \uparrow) = 0.9$

$\gamma = 0.9$

What is $V^*(s_4)$?



FIXED POINT ITERATION

Fixed Point Iteration

$$f_1(x_1, \dots, x_n) = 0$$

⋮

$$f_n(x_1, \dots, x_n) = 0$$

$$x_1 = g_1(x_1, \dots, x_n)$$

⋮

$$x_n = g_n(x_1, \dots, x_n)$$

$$x_1^{(t+1)} = g_1(x_1^{(t)}, \dots, x_n^{(t)})$$

⋮

$$x_n^{(t+1)} = g_n(x_1^{(t)}, \dots, x_n^{(t)})$$

- 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 $f(\mathbf{x}) = 0$ where \mathbf{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 values of the variables.
4. For i in $\{1, \dots, n\}$, update each parameter and increment t :
5. Repeat #4 until convergence

Fixed Point Iteration

$$\begin{aligned}\cos(y) - x &= 0 \\ \sin(x) - y &= 0\end{aligned}$$

$$\begin{aligned}x &= \cos(y) \\ y &= \sin(x)\end{aligned}$$

$$\begin{aligned}x^{(t+1)} &= \cos(y^{(t)}) \\ y^{(t+1)} &= \sin(x^{(t)})\end{aligned}$$

- 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 $f(\mathbf{x}) = 0$ where \mathbf{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 values of the variables.
 4. For i in $\{1, \dots, n\}$, update each parameter and increment t :
 5. Repeat #4 until convergence

Fixed Point Iteration

We can implement our example in a few lines of code

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

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

$$x = \cos(y)$$

$$y = \sin(x)$$

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

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

```
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

```
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

RL Terminology

Poll 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
- B. a transition probability
- C. a policy
- D. state/action/reward triples
- E. a value function
- F. transition function
- G. an optimal policy

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:

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

- Optimal policy:

- Given V^* , $R(s, a)$, $p(s' | s, a)$, γ we can compute this!

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

- Optimal value function:

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

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

Example: Path Planning

Value Iteration

Algorithm:

Example:

Value Iteration

Algorithm 1 Value Iteration (deterministic transitions)

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

Variant 1: without $Q(s,a)$ table

Value Iteration

Algorithm 1 Value Iteration (stochastic transitions)

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

Variant 1: without $Q(s, a)$ table

Value Iteration

(Stochastic Rewards)

Algorithm 1 Value Iteration (stochastic transitions, stochastic rewards)

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

This is (more or less) fixed point iteration applied to the recursive definition of the optimal value function.

Value Iteration Convergence

very abridged

Theorem 1 (Bertsekas (1989))

V converges to V^ , if each state is visited infinitely often*

Holds for both asynchronous and synchronous updates

Theorem 2 (Williams & Baird (1993))

if $\max_s |V^{t+1}(s) - V^t(s)| < \epsilon$

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

Provides reasonable stopping criterion for value iteration

Theorem 3 (Bertsekas (1987))

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

Often greedy policy converges well before the value function

Synchronous vs. Asynchronous Value Iteration

Algorithm 1 Asynchronous Value Iteration

```
1: procedure ASYNCHRONOUSVALUEITERATION( $R(s, a), p(\cdot|s, a)$ )
2:   Initialize value function  $V(s) = 0$  or randomly
3:   while not converged do
4:     for  $s \in \mathcal{S}$  do
5:        $V(s) = \max_a R(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s, a)V(s')$ 
6:   Let  $\pi(s) = \operatorname{argmax}_a R(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s, a)V(s'), \forall s$ 
7:   return  $\pi$ 
```

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

Algorithm 1 Synchronous Value Iteration

```
1: procedure SYNCHRONOUSVALUEITERATION( $R(s, a), p(\cdot|s, a)$ )
2:   Initialize value function  $V(s)^{(0)} = 0$  or randomly
3:    $t = 0$ 
4:   while not converged do
5:     for  $s \in \mathcal{S}$  do
6:        $V(s)^{(t+1)} = \max_a R(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s, a)V(s')^{(t)}$ 
7:      $t = t + 1$ 
8:   Let  $\pi(s) = \operatorname{argmax}_a R(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s, a)V(s'), \forall s$ 
9:   return  $\pi$ 
```

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

Learning Objectives

Reinforcement Learning: Value and Policy Iteration

You should be able to...

1. Compare the reinforcement learning paradigm to other learning paradigms
2. Cast a real-world problem as a Markov Decision Process
3. Depict the exploration vs. exploitation tradeoff via MDP examples
4. Explain how to solve a system of equations using fixed point iteration
5. Define the Bellman Equations
6. Show how to compute the optimal policy in terms of the optimal value function
7. Explain the relationship between a value function mapping states to expected rewards and a value function mapping state-action pairs to expected rewards
8. Implement value iteration
9. Implement policy iteration
10. Contrast the computational complexity and empirical convergence of value iteration vs. policy iteration
11. Identify the conditions under which the value iteration algorithm will converge to the true value function
12. Describe properties of the policy iteration algorithm