L15: Introduction to Reinforcement Learning

Shan Wang

Lingnan College, Sun Yat-sen University

2020 Data Mining and Machine Learning LN3119

https://wangshan731.github.io/DM-ML/

Last lecture

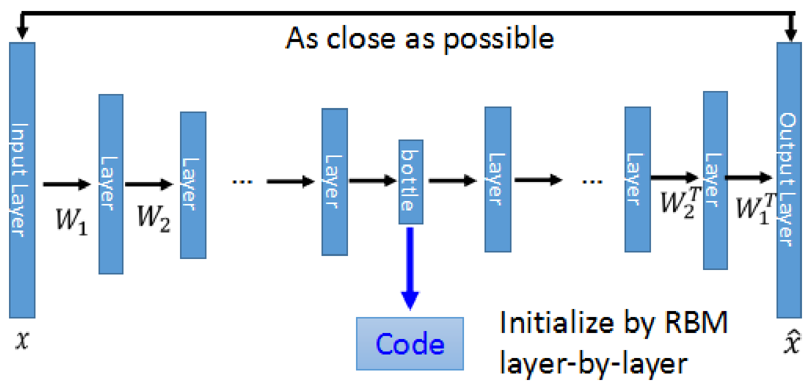
- Dimension Reduction
  - PCA (squared matrix)
  - SVD (general matrix)
  - Auto Encoder

Eigenvector of Matrix A

$$\mathbf{A}\mathbf{x} = \lambda \mathbf{x}$$

Eigenvalue of Matrix A

$$A_{m\times n} = U_{m\times m} S_{m\times n} V_{n\times n}^{T}$$



- EM Method
  - E-step: construct a (good) lower-bound of log-likelihood
  - M-step: optimize that lower-bound

What we have learned so far

- Supervised Learning
  - To perform the desired output given the data and labels
  - e.g., to build a loss function to minimize
- Unsupervised Learning
  - To analyze and make use of the underlying data patterns/structures
  - e.g., to build a log-likelihood function to maximize

Supervised Learning

- Given the training dataset of (data, label) pairs,
$$D = \{(\boldsymbol{x}_i, y_i)\}_{i=1,2,\ldots,N}$$
let the machine learn a function from data to label
$$y_i \backsimeq f_\theta(\boldsymbol{x}_i)$$
- Learning is referred to as updating the parameter θ

- Learning objective: make the prediction close to the ground truth
  - $f_\theta(\boldsymbol{x}_i)$ is as close to y_i as possible

Unsupervised Learning

- Given the training dataset

$$D = \{(\boldsymbol{x}_i)\}_{i=1,2,\ldots,N}$$

let the machine learn the data underlying patterns

- Sometimes build latent variables

$$z \rightarrow \boldsymbol{x}$$

- Estimate the probabilistic density function (p.d.f.)

$$p(\boldsymbol{x}|\theta) = \sum_{z} q(z)p(\boldsymbol{x}|z,\theta)$$

- Maximize the likelihood of training data

$$\prod_{i=1}^{N} p(\boldsymbol{x}_i|\theta)$$

Two Kinds of Machine Learning

- Prediction
  - Predict the desired output given the data (supervised learning)
  - Generate data instances (unsupervised learning)
  - We mainly covered this category in previous lectures
- Decision Making
  - Take actions based on a particular state in a dynamic environment (reinforcement learning)
    - to transit to new states
    - to receive immediate reward
    - to maximize the accumulative reward over time
  - Learning from interaction

Machine Learning Categories

- Supervised Learning
  - To perform the desired output given the data and labels $p(y|\boldsymbol{x})$
- Unsupervised Learning
  - To analyze and make use of the underlying data patterns/structures $p(\boldsymbol{x})$
- Reinforcement Learning
  - To learn a policy of taking actions in a dynamic environment and acquire rewards $\pi(a|\boldsymbol{x})$

Course Outline

- Supervised learning
  - Linear regression
  - Logistic regression
  - SVM and kernel
  - Tree models
- Unsupervised learning
  - Clustering
  - PCA
  - EM
- Deep learning
  - Neural networks
  - Convolutional NN
  - Recurrent NN
- Reinforcement learning
  - MDP
  - ADP
  - Deep Q-Network

This lecture

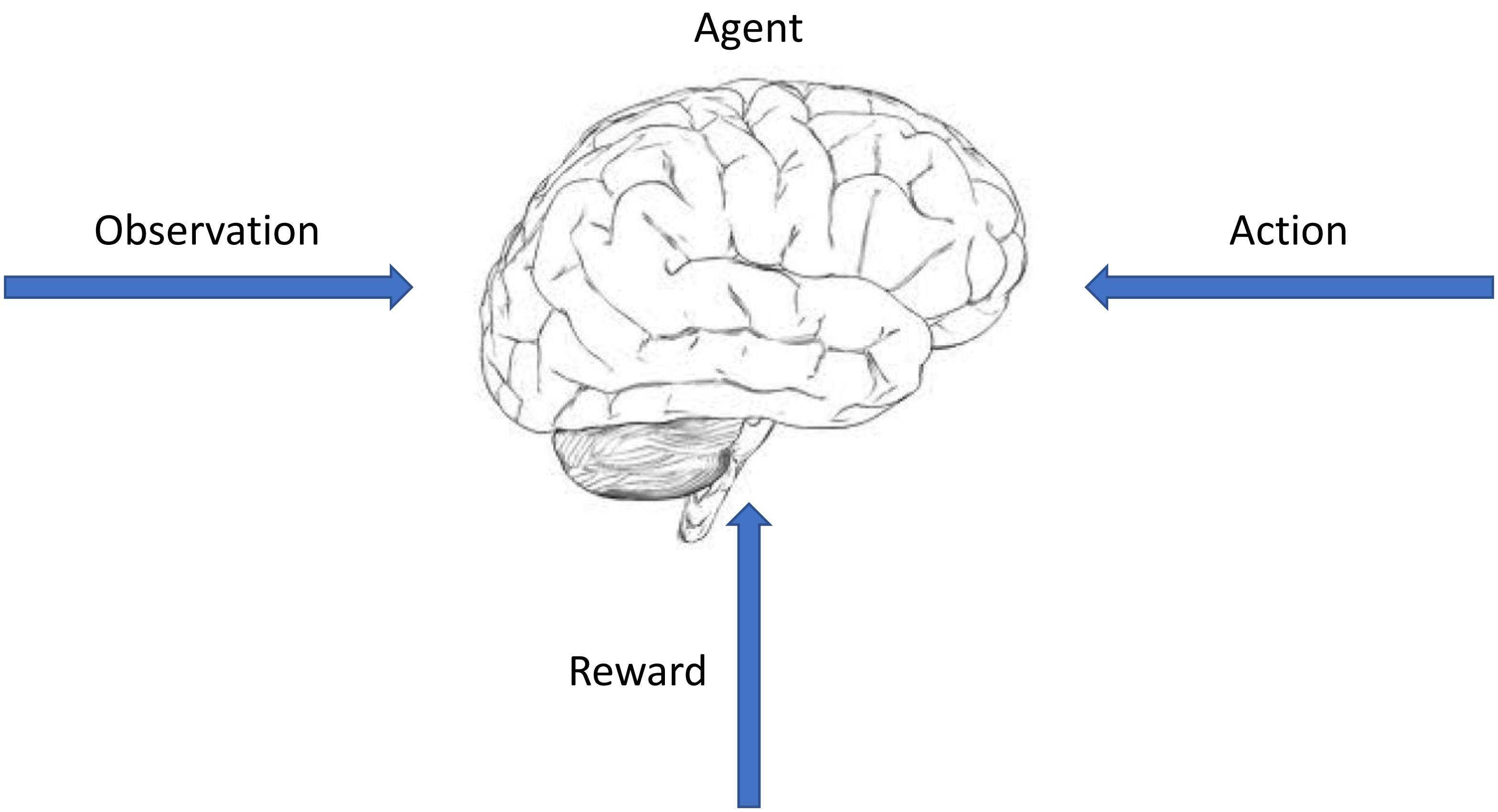
- Introduction to Reinforcement Learning
- Model-based Reinforcement Learning
  - MDP
  - Value iteration
  - Policy iteration
- Model-free Reinforcement Learning

Reference: CS 420, Weinan Zhang (SJTU)

Introduction to RL

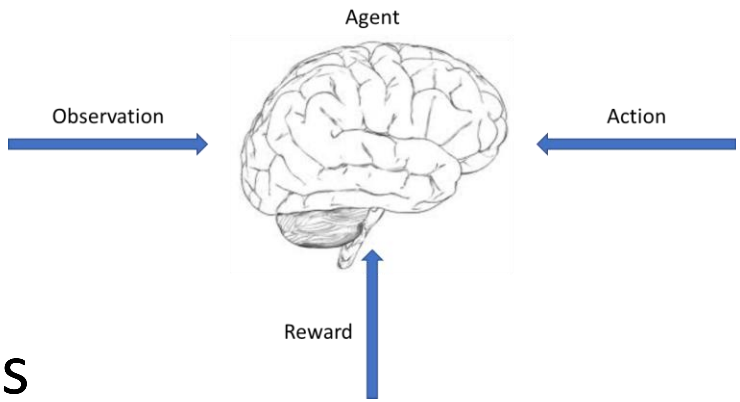
Reinforcement Learning

- Learning from interaction
  - Given the current situation, what to do next in order to maximize utility?



Reinforcement Learning Definition

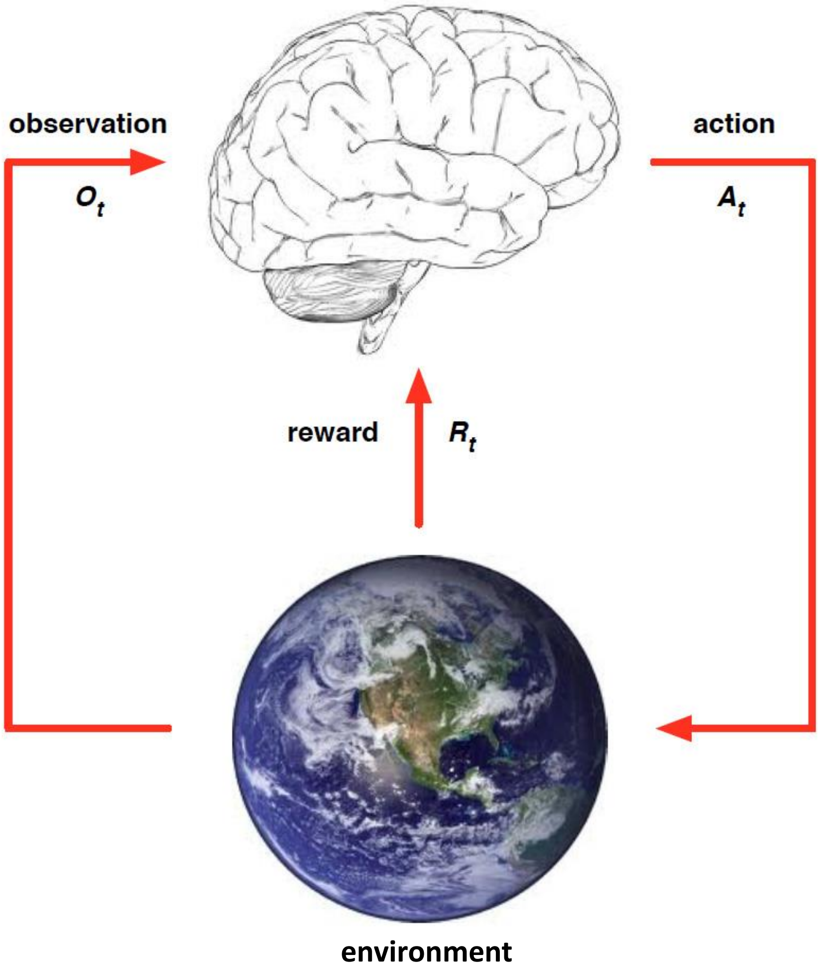
- A computational approach by learning from interaction to achieve a goal



- Three aspects
  - Sensation: sense the state of the environment to some extent
  - Action: able to take actions that affect the state and achieve the goal
  - Goal: maximize the cumulative reward over time

Process of RL

- At step t, the agent
  - Receives observation O_t
  - Receives scalar reward R_t
  - Executes action A_t
- The environment
  - Receives action A_t
  - Emits observation O_{t+1}
  - Emits scalar reward R_{t+1}
- t increments at environment step



Elements of RL Systems

- History is the sequence of observations, action, rewards
$$H_t = \{O_1, R_1, A_1, O_2, R_2, A_2, \dots, O_t, R_t\}$$
  - i.e. all observable variables up to time t
  - E.g., all the records of the Go game
- What happens next depends on the history:
  - The agent selects actions
  - The environment selects observations/rewards
- State is the information used to determine what happens next (actions, observations, rewards)
- Formally, state is a function of the history
$$S_t = f(H_t)$$

Elements of RL Systems

- Policy is the learning agent's way of behaving at a given time
  - It is a map from state to action
  - Deterministic policy
$$a = \pi(s)$$
  - Stochastic policy
$$\pi(s|a) = P(A_t = a|S_t = s)$$

Elements of RL Systems

- Reward
  - A scalar defining the goal in an RL problem
  - For immediate sense of what is good

- Value function
  - State value (Value of a state) is a scalar specifying what is good in the long run
  - Value function is a prediction of the cumulative future reward
    - Used to evaluate the goodness/badness of states (given the current policy)

$$v_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots | S_t = s]$$

Elements of RL Systems

- Reward
  - A scalar defining the goal in an RL problem
  - For immediate sense of what is good

- Value function
  - State value (value of a state) is a scalar specifying what is good in the long run, i.e., the cumulative reward

    $$v_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots | S_t = s]$$

  - Action value (value of a action) is a scalar specifying what is a good action at a specific state in the long run

$$Q_\pi(s, a) = \mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots | S_t = s, A_t = a]$$

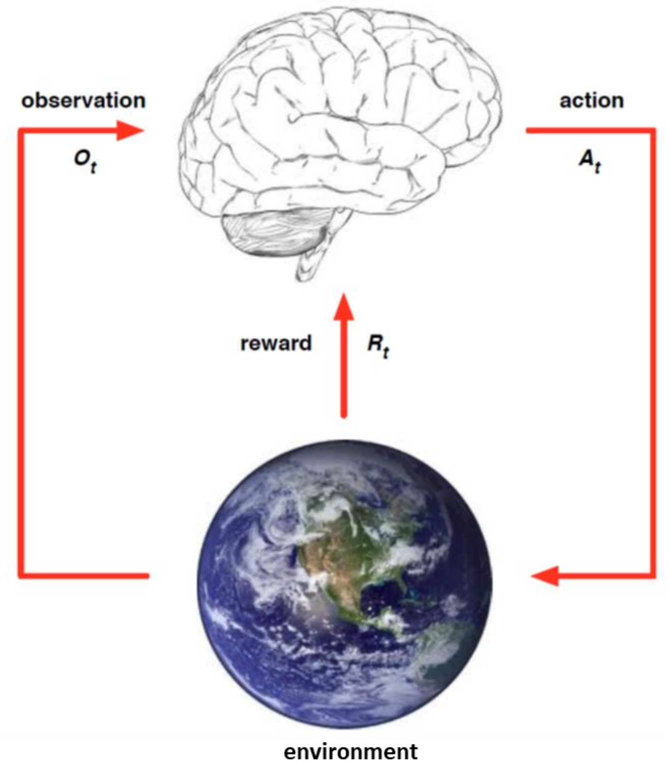
Elements of RL Systems

- A Model of the environment that mimics the behavior of the environment
  - Predict the next state

$$\mathcal{P}_{sa}(s') = \mathbb{P}[S_{t+1} = s' | S_t = s, A_t = a]$$

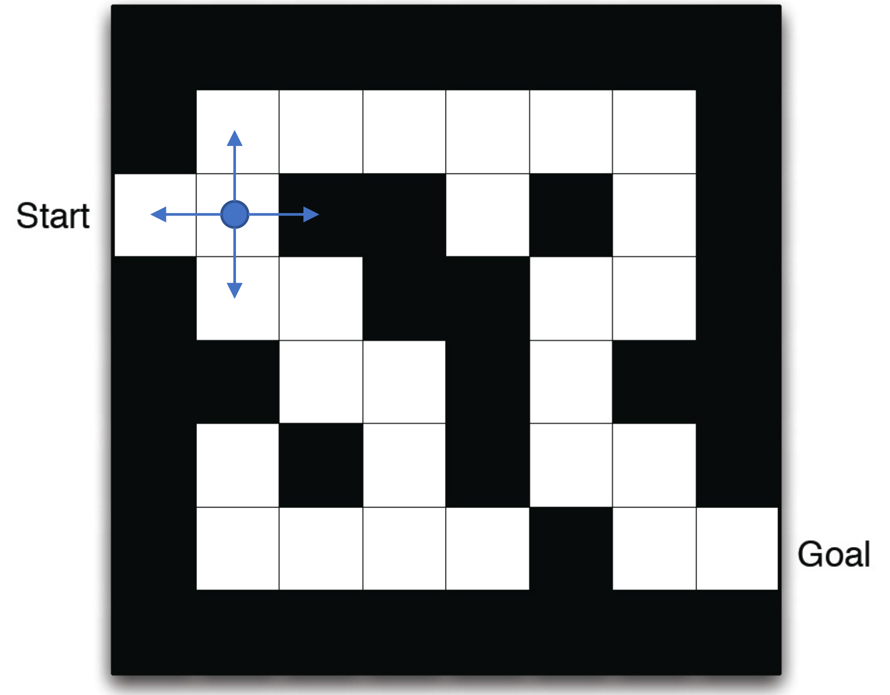
  - Predict the next (immediate) reward

$$\mathcal{R}_s(a) = \mathbb{E}[R_{t+1} | S_t = s, A_t = a]$$



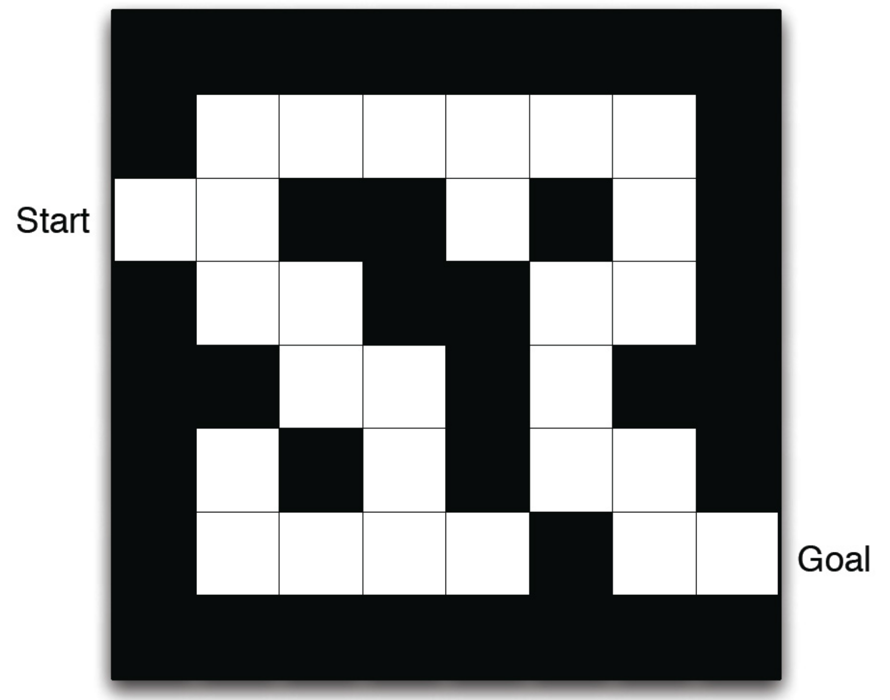
Maze Example

- State: agent's location
- Action: N,E,S,W
- State transition: move to the next grid according to the action
  - No move if the action is to the wall



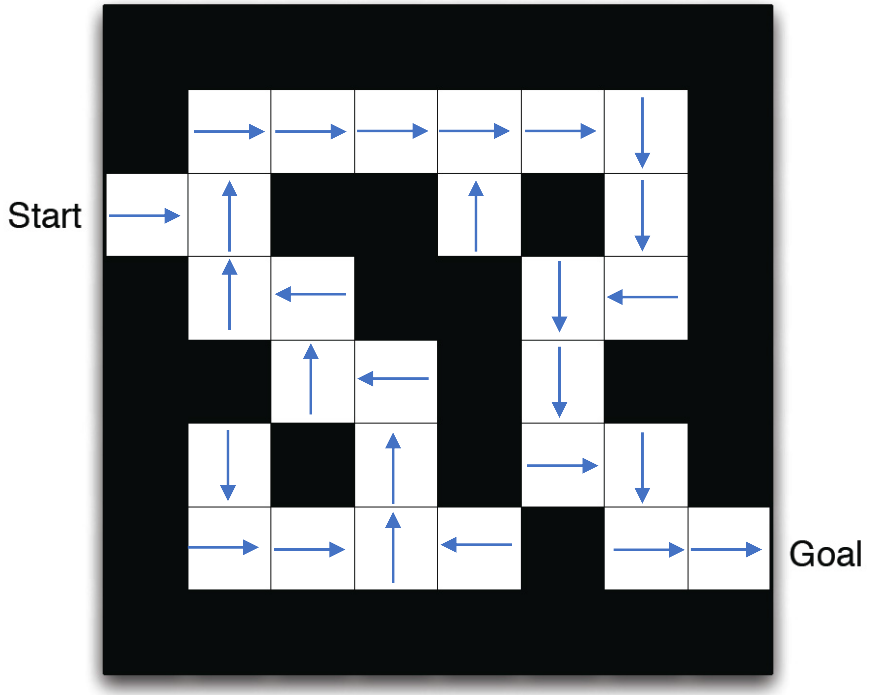
Maze Example

- State: agent's location
- Action: N,E,S,W
- State transition: move to the next grid according to the action
- Reward: -1 per time step



Maze Example

- State: agent's location
- Action: N,E,S,W
- State transition: move to the next grid according to the action
- Reward: -1 per time step

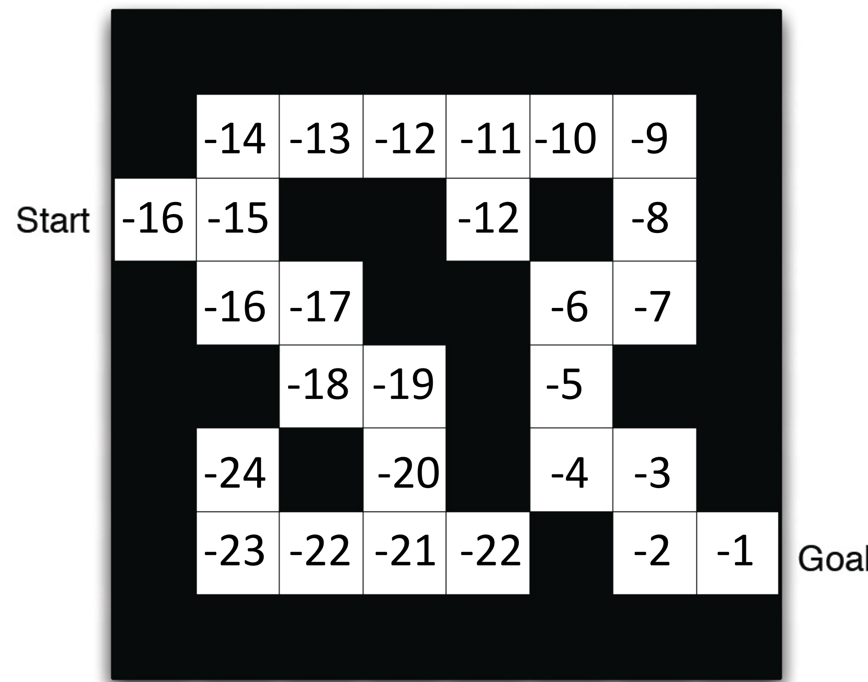


Given a policy as shown above

- Arrows represent policy $\pi(s)$ for each state s

Maze Example

- State: agent's location
- Action: N,E,S,W
- State transition: move to the next grid according to the action
- Reward: -1 per time step



Numbers represent value $v\pi(s)$ of each state s

Another example

- http://www.4399.com/flash/105474_1.htm

Model-based RL

Markov Decision Process

Markov Decision Process

- Markov decision processes (MDPs) provide a mathematical framework for modeling decision making in situations where outcomes are partly random and partly under the control of a decision maker.

- MDPs formally describe an environment for RL
  - where the environment is FULLY observable
  - i.e. the current state completely characterizes the process (Markov property)

Markov Property

- "The future is independent of the past given the present"
- Definition
  - A state s_t is Markov if and only if

$$P[s_{t+1}|s_t] = P[s_{t+1}|s_t, s_{t-1}, \dots, s_1]$$

- Properties
  - The state captures all relevant information from the history
  - Once the state is known, the history may be thrown away
  - i.e. the state is sufficient statistic of the future

Markov Decision Process

- A Markov decision process is a tuple $(S, A, \{P_{sa}\}, \gamma, R)$
  - S is the set of states
    - E.g., location in a maze, or current screen in an Atari game
  - A is the set of actions
    - E.g., move N, E, S, W, or the direction of the joystick and the buttons
  - P_{sa} are the state transition probabilities
    - For each state $s \in S$ and action $a \in A$, P_{sa} is a distribution over the next state in S
  - $\gamma \in [0,1]$ is the discount factor for the future reward
  - $R: S \times A \rightarrow \mathbb{R}$ is the reward function
    - Sometimes the reward is only assigned to state, i.e., irrelative to the action

Markov Decision Process

- The dynamics of an MDP proceeds as
  - Start in a state s_0
  - The agent chooses some action $a_0 \in A$
  - The agent gets the reward $R(s_0, a_0)$
  - MDP randomly transits to some successor state $s_1 \sim P_{s_0 a_0}$
  - This proceeds iteratively

$$s_0 \xrightarrow[R(s_0,a_0)]{a_0} s_1 \xrightarrow[R(s_1,a_1)]{a_1} s_2 \xrightarrow[R(s_2,a_2)]{a_2} s_3 \cdots$$

  - Until a terminal state S_T or proceeds with no end
  - The total payoff of the agent is

$$R(s_0, a_0) + \gamma R(s_1, a_1) + \gamma^2 R(s_2, a_2) + \cdots$$

Reward on State Only

- For a large part of cases, reward is only assigned to the state
  - E.g., in maze game, the reward is on the location
  - In game of Go, the reward is only based on the final territory
- The reward function $R(s): S \rightarrow \mathbb{R}$
- MDPs proceed

$$s_0 \xrightarrow[R(s_0)]{a_0} s_1 \xrightarrow[R(s_1)]{a_1} s_2 \xrightarrow[R(s_2)]{a_2} s_3 \cdots$$

- cumulative reward (total payoff)

$$R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots$$

MDP Goal and Policy

- The goal is to choose actions over time to maximize the expected cumulative reward
$$\mathbb{E}[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots]$$
- $\gamma \in [0,1]$ is the discount factor for the future reward, which makes the agent prefer immediate reward to future reward
  - In finance case, today's \$1 is more valuable than \$1 in tomorrow
- Given a particular policy $\pi(s): S \to A$
  - i.e. take the action $a = \pi(s)$ at state s
- Define the value function for π
$$V^\pi(s) = \mathbb{E}[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots | s_0 = s, \pi]$$
    - i.e. expected cumulative reward given the start state s and taking actions according to π

Bellman Equation for Value Function

- Define the value function for π

$$V^{\pi}(s) = \mathbb{E}[R(s_0) + \boxed{\gamma R(s_1) + \gamma^2 R(s_2) + \cdots} | s_0 = s, \pi]$$

$$\boxed{\gamma V^{\pi}(s_1)}$$

Bellman Equation:

$$V^{\pi}(s) = \underbrace{R(s)}_{\text{Immediate Reward}} + \underbrace{\gamma}_{\text{Time decay}} \sum_{s\prime \in S} \underbrace{P_{s\pi(s)}(s')}_{\text{State transition}} \underbrace{V^{\pi}(s')}_{\text{Value of the next state}}$$

Optimal Value Function

- The optimal value function for each state s is best possible sum of discounted rewards that can be attained by any policy
$$V^*(s) = \max_{\pi} V^{\pi}(s)$$
- The Bellman's equation for optimal value function
$$V^*(s) = R(s) + \gamma \max_{a \in A} \sum_{s' \in S} P_{sa}(s') V^*(s')$$
- The optimal policy
$$\pi^*(s) = \underset{a \in A}{\operatorname{argmax}} \sum_{s' \in S} P_{sa}(s') V^*(s')$$
- For every state s and every policy π
$$V^*(s) = V^{\pi^*}(s) \geq V^{\pi}(s)$$

Value Iteration & Policy Iteration

- Note that the value function and policy are correlated

$$V^{\pi}(s) = R(s) + \gamma \max_{a \in A} \sum_{s' \in S} P_{sa}(s') V^{\pi}(s')$$

$$\pi(s) = \underset{a \in A}{\operatorname{argmax}} \sum_{s' \in S} P_{sa}(s') V^{\pi}(s')$$

- It is feasible to perform iterative update towards the optimal value function and optimal policy
  - Value iteration
  - Policy iteration

Value Iteration

- For an MDP with finite state and action spaces

$$|S| < \infty, |A| < \infty$$

- Value iteration is performed as
  1. For each state s, initialize $V(s) = 0$
  2. Repeat until convergence {

     For each state, update

$$V(s) = R(s) + \gamma \max_{a \in A} \sum_{s' \in S} P_{sa}(s') V(s')$$

  }

- Note that there is no explicit policy in above calculation

Value Iteration Example: Shortest Path

g			

Problem

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

V_1

0	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1

V_2

0	-1	-2	-2
-1	-2	-2	-2
-2	-2	-2	-2
-2	-2	-2	-2

V_3

0	-1	-2	-3
-1	-2	-3	-3
-2	-3	-3	-3
-3	-3	-3	-3

V_4

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-4
-3	-4	-4	-4

V_5

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-5
-3	-4	-5	-5

V_6

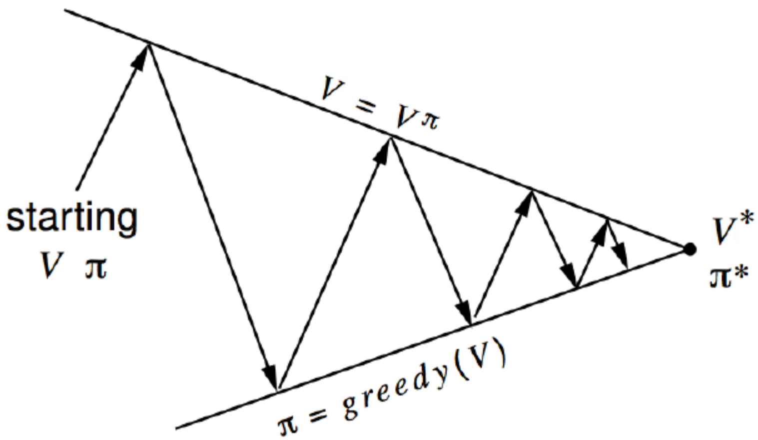
0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-5
-3	-4	-5	-6

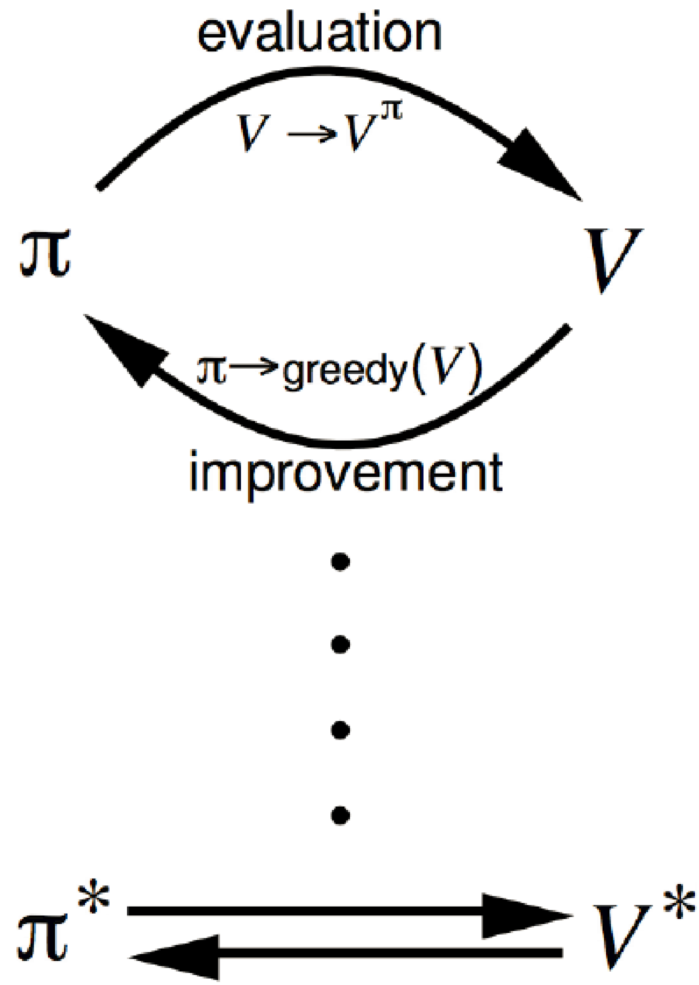
V_7

Policy Iteration

- For an MDP with finite state and action spaces
$$|S| < \infty, |A| < \infty$$
- Policy iteration is performed as
  1. Initialize π randomly
  2. Repeat until convergence {
     a) Let $V := V^{\pi}$
     b) For each state, update
$$\pi(s) = \underset{a \in A}{\operatorname{argmax}} \sum_{s' \in S} P_{sa}(s') V^{\pi}(s')$$
     }
- The step of value function update could be time-consuming

Policy Iteration





- Policy evaluation
  - Estimate V^{π}
  - Iterative policy evaluation
- Policy improvement
  - Generate $\pi' \geq \pi$
  - Greedy policy improvement

Value Iteration vs. Policy Iteration

- Value iteration

1. For each state s, initialize $V(s) = 0$
2. Repeat until convergence {

   For each state, update

$$V(s) = R(s) + \gamma \max_{a \in A} \sum_{s' \in S} P_{sa}(s') V(s')$$

}

- Policy iteration

1. Initialize π randomly
2. Repeat until convergence {
   a) Let $V := V^{\pi}$
   b) For each state, update

$$\pi(s) = \operatorname*{argmax}_{a \in A} \sum_{s' \in S} P_{sa}(s') V^{\pi}(s')$$

}

Remarks:

1. Value iteration is a greedy update strategy
2. In policy iteration, the value function update by bellman equation is costly
3. For small-space MDPs, policy iteration is often very fast and converges quickly
4. For large-space MDPs, value iteration is more practical (efficient)
5. If there is no state-transition loop, it is better to use value iteration

Learning an MDP Model

- So far we have been focused on
  - Calculating the optimal value function
  - Learning the optimal policy

  given a known MDP model
  - i.e. the state transition $P_{sa}(s')$ and reward function $R(s)$ are explicitly given
- In realistic problems, often the state transition and reward function are not explicitly given
- We only have some observations by experience
  - e.g., play games, inventory management with unknown demand, online advertisement…

Learning an MDP Model

- Learn an MDP model from "experience"
  - Learning state transition probabilities $P_{sa}(s')$

$$P_{sa(s')} = \frac{\#\text{times we took action } a \text{ in state } s \text{ and got to state } s'}{\#\text{times we took action } a \text{ in state } s}$$

  - Learning reward $R(s)$, i.e. the expected immediate reward

$$R(s) = \operatorname{avg}(R(s)^{(i)})$$

Learning Model and Optimizing Policy

Algorithm

1. Initialize π randomly

2. Repeat until convergence {

    a) Execute π in the MDP for some number of trials

    b) Using the accumulated experience in the MDP, update our estimates for P_{sa} and R

    c) Apply value iteration with the estimated P_{sa} and R to get the new estimated value function V

    d) Update π to be the greedy policy w.r.t. V

}

- Another branch of solution is to directly learning value & policy from experience without building an MDP
- i.e. **Model-free Reinforcement Learning**

Model-free RL

Model-free Prediction

Model-free Reinforcement Learning

- In realistic problems, often the state transition and reward function are not explicitly given
- Model-free RL is to directly learn value & policy from experience without building an MDP
- Key steps: (1) estimate value function; (2) optimize policy

Value Function Estimation

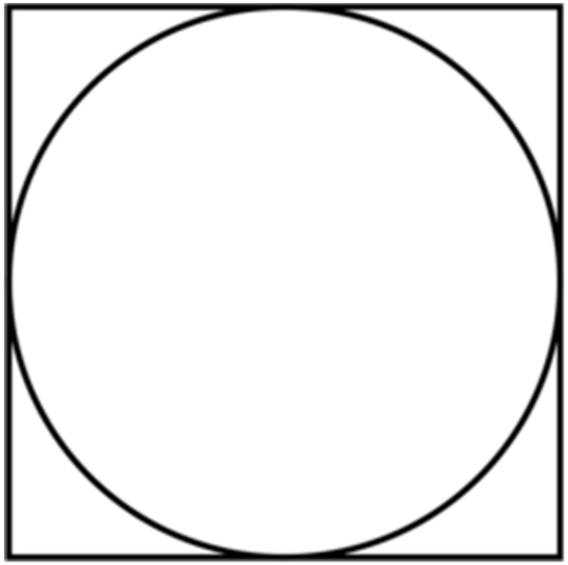
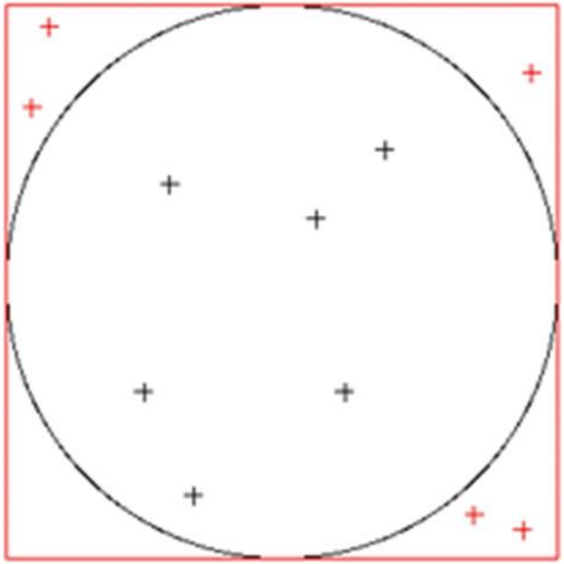
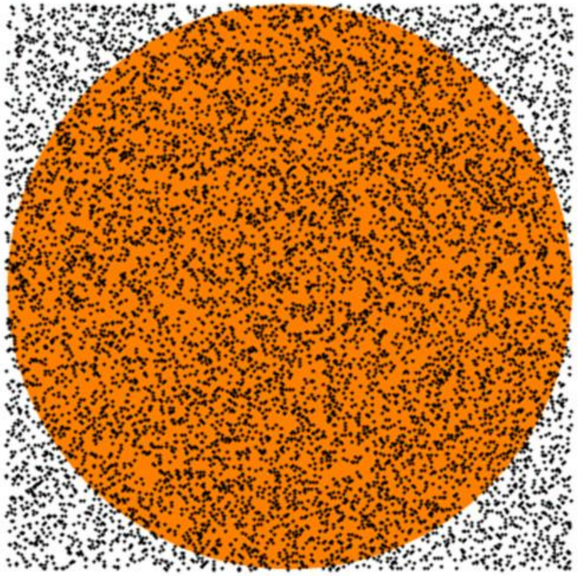
- In model-based RL (MDP), the value function is calculated by dynamic programming

$$V^{\pi}(s) = \mathbb{E}[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots | s_0 = s, \pi]$$
$$= R(s) + \gamma \sum_{s' \in S} P_{s\pi(s)}(s') V^{\pi}(s')$$

- Now in model-free RL
  - We cannot directly know P_{sa} and R
  - But we have a list of experiences to estimate the values

Monte-Carlo Methods

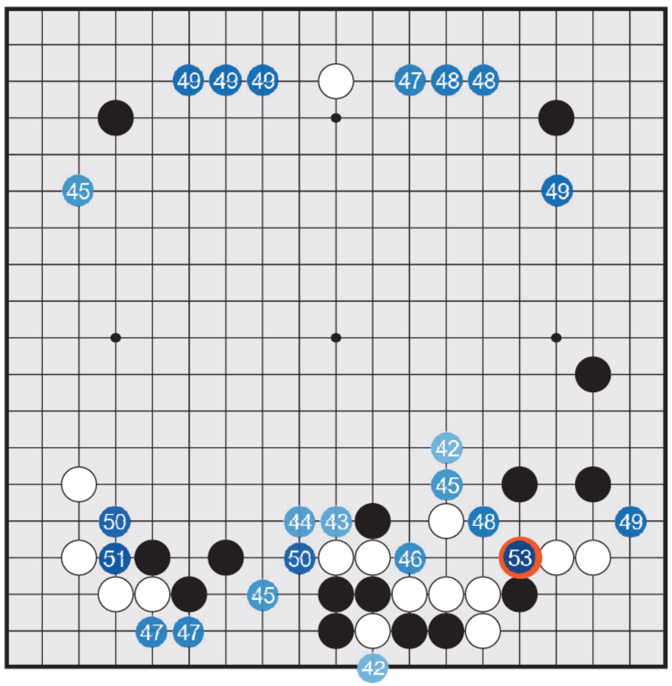
- Monte-Carlo methods are a broad class of computational algorithms that rely on repeated random sampling to obtain numerical results.
- Example, to calculate the circle's surface

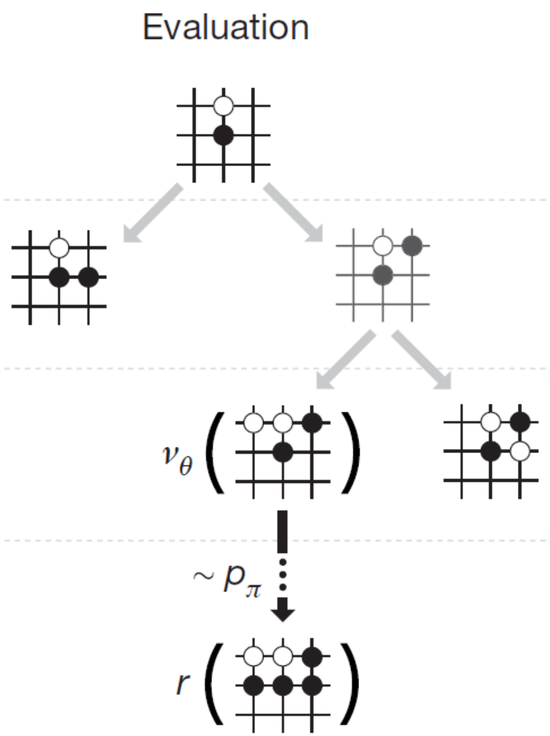




$$\text{Circle Surface} = \text{Square Surface} \times \frac{\#\text{points in circle}}{\#\text{points in total}}$$

Monte-Carlo Methods

- Go: to estimate the winning rate given the current state





$$\text{Win Rate}\,(s) = \frac{\#\text{win simulation cases started from } s}{\#\text{simulation cases started from } s \text{ in total}}$$

Monte-Carlo Value Estimation

- Goal: learn V^{π} from experience under policy π
- Recall that the return is the total discounted reward

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \gamma^{T-1} R_T$$

- Recall that the value function is the expected return

$$\begin{aligned} V^{\pi}(s) &= \mathbb{E}[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots | s_0 = s, \pi] \\ &= \mathbb{E}[G_t | s_t = s, \pi] \\ &\simeq \frac{1}{N} \sum_{i=1}^{N} G_t^{(i)} \end{aligned}$$

  - Sample N episodes from state s using policy π
  - Calculate the average of cumulative reward
  - $G_t^{(i)}$ is the G_t of ith sample

- Monte-Carlo policy evaluation uses empirical mean return instead of expected return

Monte-Carlo Value Estimation

- Implementation
  - Sample episodes under policy π

$$s_0^{(i)} \xrightarrow[R_1^{(i)}]{a_0^{(i)}} s_1^{(i)} \xrightarrow[R_2^{(i)}]{a_1^{(i)}} s_2^{(i)} \xrightarrow[R_3^{(i)}]{a_2^{(i)}} s_3^{(i)} \cdots s_T^{(i)} \sim \pi$$

- Every time-step t that state s is visited in an episode
  - Increment counter $N(s) = N(s) + 1$
  - Increment total return $S(s) = S(s) + G_t$
  - Value is estimated by mean return $V(s) = S(s)/N(s)$
  - By law of large numbers

$$V(s) \longrightarrow V^{\pi}(s) \text{ as } N(s) \longrightarrow \infty$$

Incremental Monte-Carlo Updates

- Update $V(s)$ incrementally after each episode
- For each state S_t with cumulative return G_t

$$N(s_t) = N(s_t) + 1$$

$$V(s_t) = V(s_t) + \frac{1}{N(s_t)}(G_t - V(s_t))$$

- For non-stationary problems (i.e. the environment could be varying over time), it can be useful to track a running mean, i.e. forget old episodes

$$V(s_t) = V(s_t) + \alpha(G_t - V(s_t))$$

Monte-Carlo Value Estimation

Idea: $V(s) = \frac{1}{N}\sum_{i=1}^{N} G_t^{(i)}$

Implementation: $V(s_t) = V(s_t) + \alpha(G_t - V(s_t))$

- MC methods learn directly from episodes of experience
- MC is model-free: no knowledge of MDP transitions / rewards
- MC learns from **complete** episodes
- MC uses the simplest possible idea: value = mean return
- Caveat: can only apply MC to episodic MDPs
  - All episodes must terminate

Lecture 15 Wrap-up

- ✓ Reinforcement Learning
- ✓ Model-based Reinforcement Learning
- ✓ Model-free Reinforcement Learning

Next Lecture

- Supervised learning
  - Linear regression
  - Logistic regression
  - SVM and kernel
  - Tree models
- Unsupervised learning
  - Clustering
  - PCA (Dimension Reduction)
  - EM
- Deep learning
  - Neural networks
  - Convolutional NN
  - Recurrent NN
- Reinforcement learning
  - MDP
  - ADP
  - Deep Q-Network

Questions?

Shan Wang (王杉)

https://wangshan731.github.io/