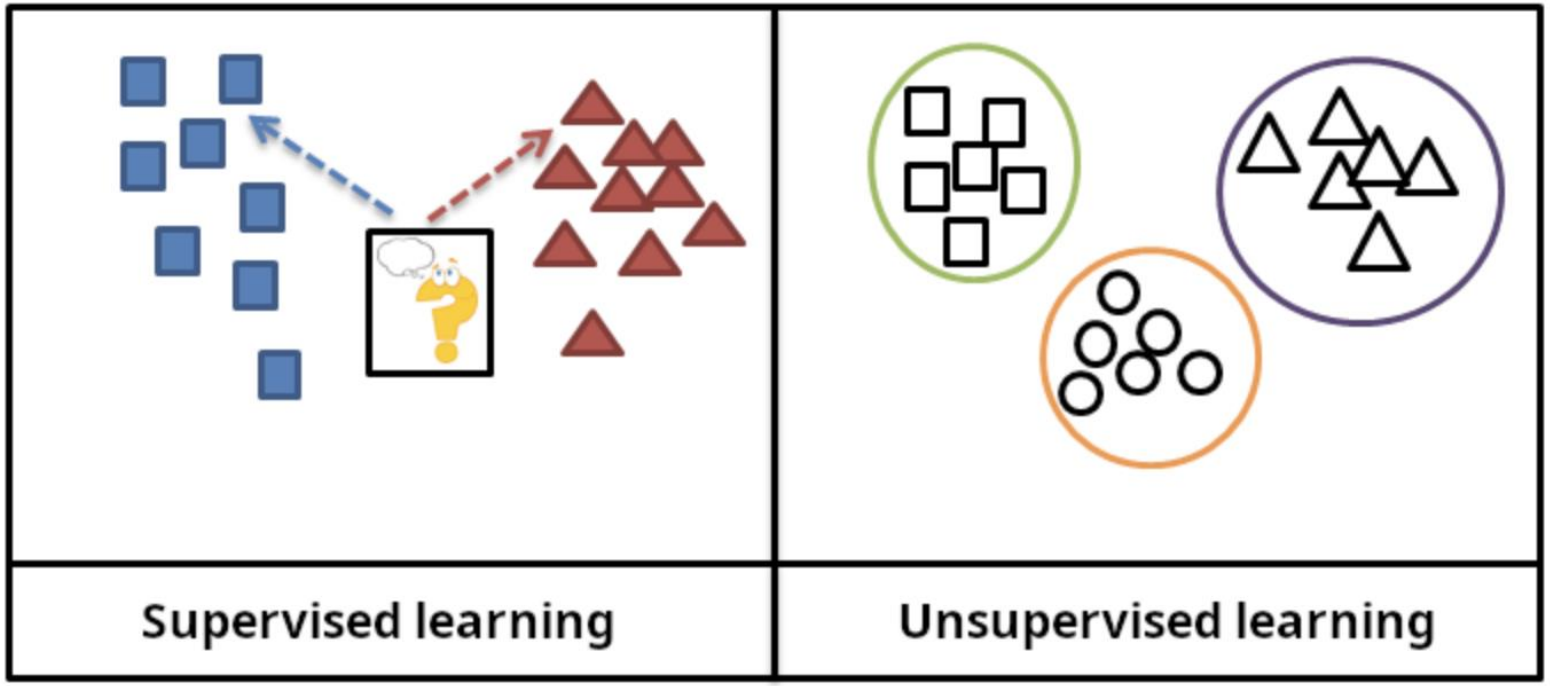
Supervised and Un-supervised learning

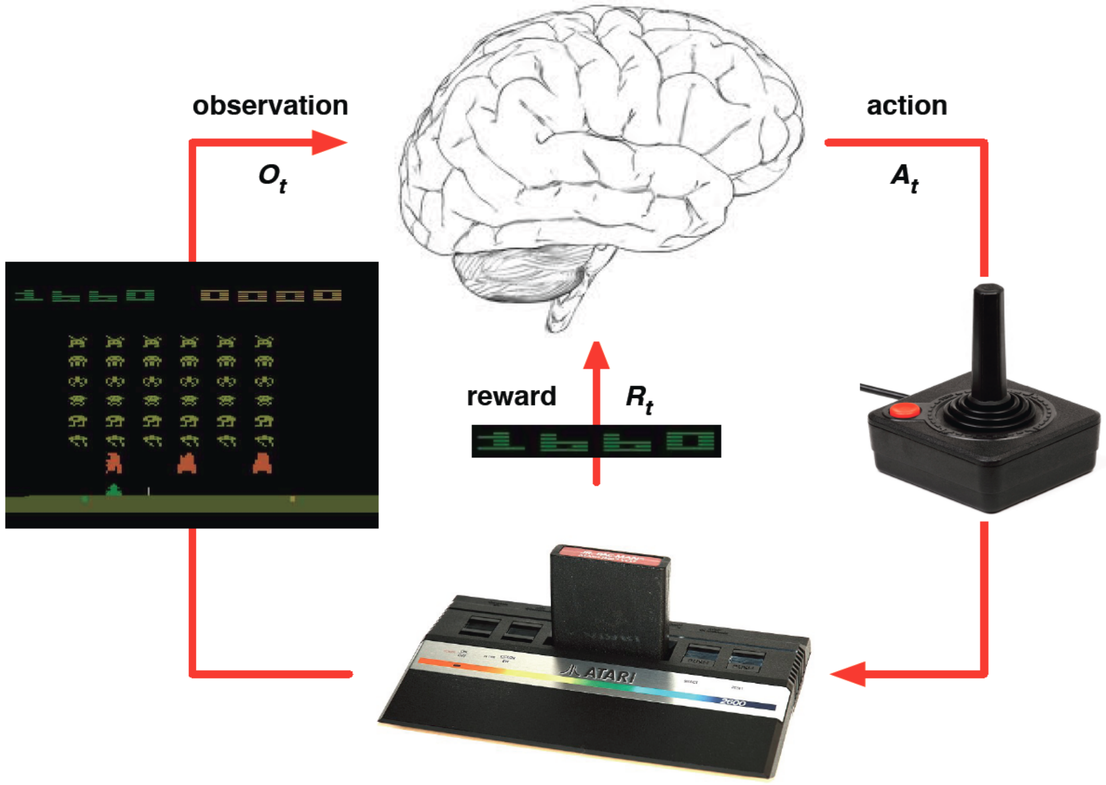


- Learning from data
- Using the supervision signals

- Seeking the pattern in dataset
- Without supervision signals

https://en.wikipedia.org/wiki/Supervised_learning

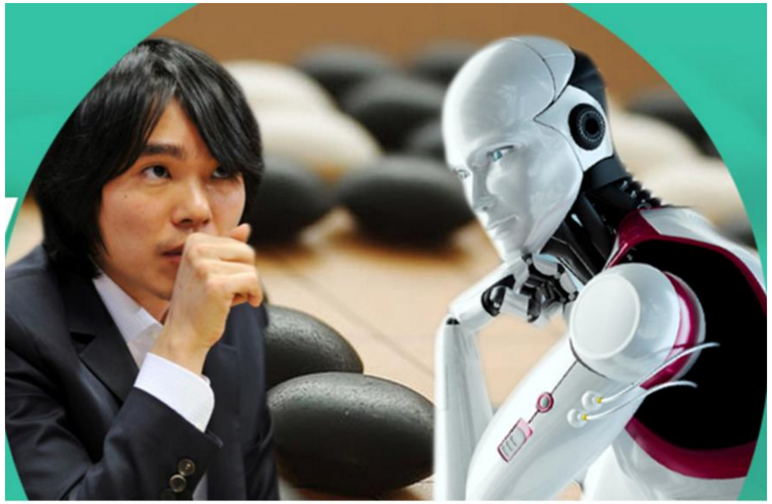
Reinforcement Learning
Learning to take actions: AI plays Atari Games

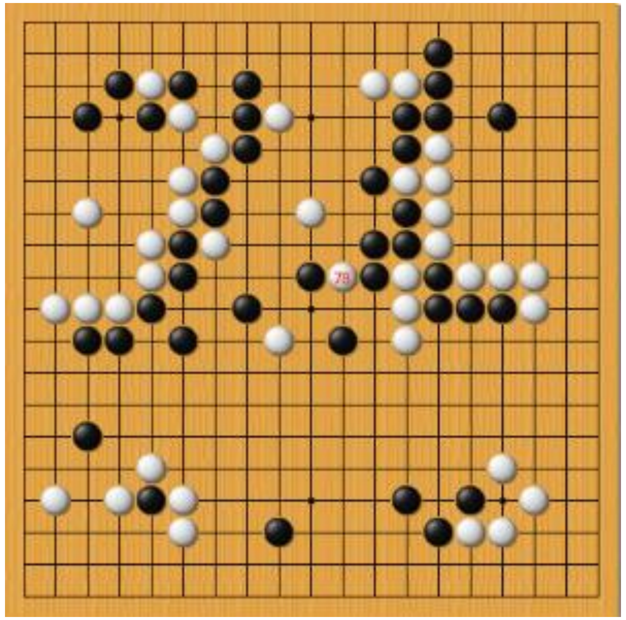


- Rules of the game are unknown
- Learn from interactive game-play
- Pick actions on joystick, see pixels and scores

Mnih V, Kavukcuoglu K, Silver D, et al. Playing atari with deep reinforcement learning[J]. arXiv preprint arXiv:1312.5602, 2013.

AlphaGo vs. the world's 'Go' champion





Rating List 2016

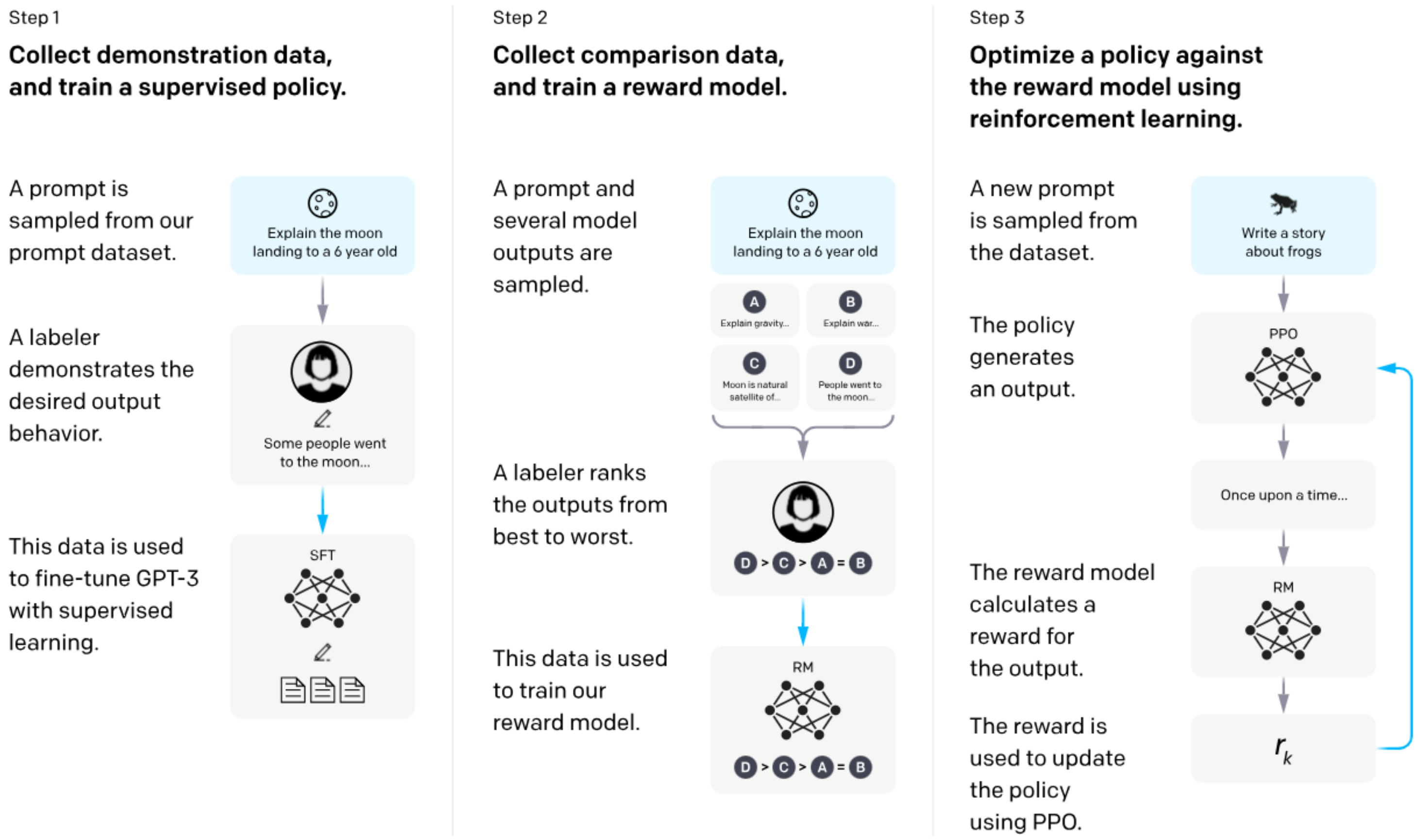
...s, check the History page. There is also a History of top ladies.

Rank	Name	♂ ♀	Flag	Elo
1	Ke Jie	♂		3621
2	Park Jungwhan	♂		3569
3	Iyama Yuta	♂		3546
4	Google AlphaGo			3533
5	Lee Sedol	♂		3521
6	Shi Yue	♂		3509
7	Park Yeonghun	♂		3509
8	Kim Jiseok	♂		3504
9	Mi Yuting	♂		3501
10	Zhou Ruiyang	♂		3498
11	Kang Dongyun	♂		3498
12	Tang Weixing	♂		3479
13	Lian Xiao	♂		3475
14	Chen Yaoye	♂		3472
15	Gu Zihao	♂		3468
16	Gu Li	♂		3455
17	Huang Yunsong	♂		3452
18	Jiang Weijie	♂		3448

http://www.goratings.org/

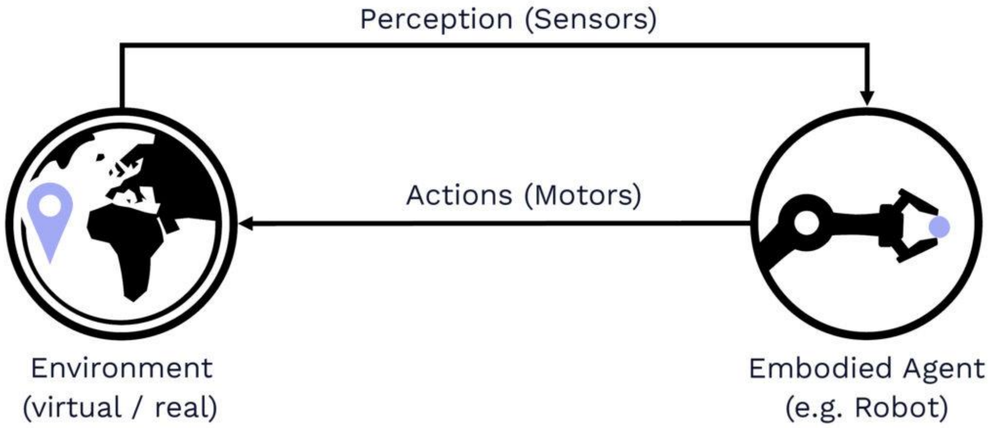
Coulom, Rémi. "Whole-history rating: A bayesian rating system for players of time-varying strength." Computers and games. Springer Berlin Heidelberg, 2008. 113-124.

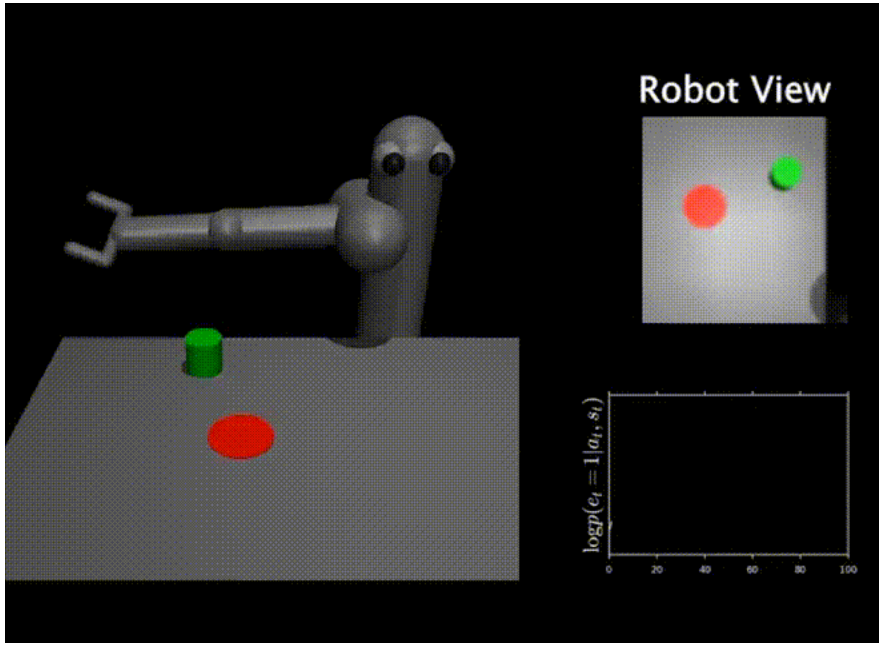
ChatGPT



OpenAI. "Training language models to follow instructions with human feedback.".

Robotics





- Perception of the environment
- Taking actions by control systems
- Learn from interactive

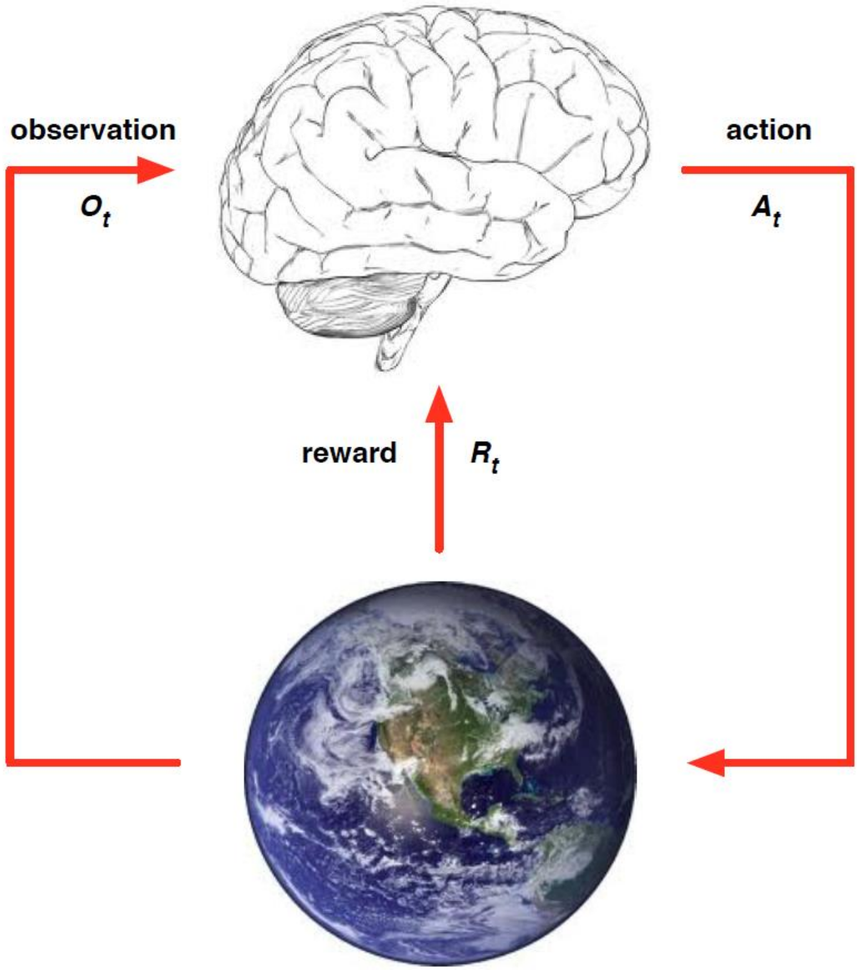
https://github.com/avisingh599/reward-learning-rl
https://sites.google.com/view/inverse-event

Content

- Reinforcement Learning (Definition and Value-based methods)
  - The model-based methods
    - **Markov Decision Process**
    - Planning by Dynamic Programming
  - The model-free methods
    - Model-free Prediction
      - Monte-Carlo and Temporal Difference
    - Model-free Control
      - On-policy SARSA and off-policy Q-learning
- Deep Reinforcement Learning

Reinforcement Learning

learning to take actions over an environment so as to maximize some numerical value which represents a long-term objective (goal).



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

Markov Decision Process

- Markov decision processes (MDPs) provide a mathematical framework for modeling decision making.

- MDPs formally describe an environment for RL
  - where the environment is FULLY observable

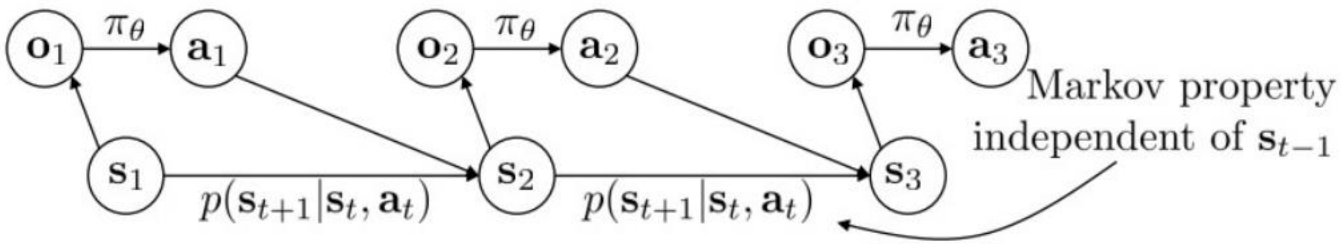
Markov Property

"The future is independent of the past given the present"

$$H_t = O_1, R_1, A_1, O_2, R_2, A_2, \ldots, O_{t-1}, R_{t-1}, A_{t-1}, O_t, R_t$$

- Definition
  - A state S_t is Markov if and only if $\mathbb{P}[S_{t+1}|S_t] = \mathbb{P}[S_{t+1}|S_1, \ldots, S_t]$

- Properties
  - The state captures all relevant information from the history
  - Why Markov? - Allow us to throw away history once state is known!



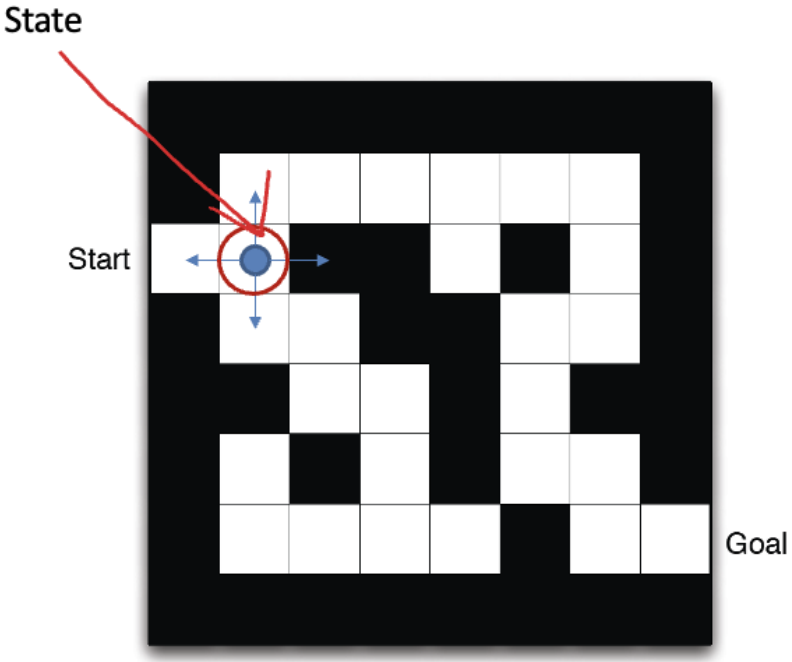
Markov Decision Process

- 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
- $R: S \times A \mapsto \mathbb{R}$ is the reward function
  - Sometimes the reward is only assigned to state

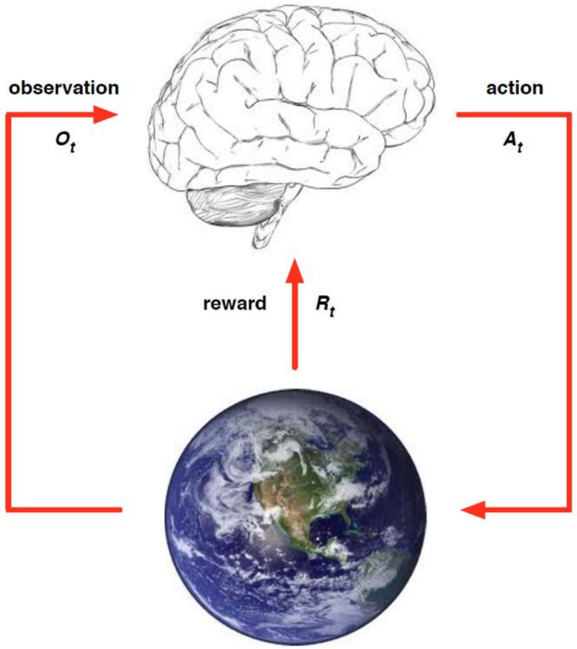
Markov Decision Process

The dynamics of an MDP proceeds as

- Start in a state s_0
  - E.g., location in a maze, or current screen in an Atari game
- The agent chooses some action $a_0 \in A$
  - E.g., move N, E, S, W, or the direction of the joystick and the buttons



- State: agent's location
- Action: N,E,S,W

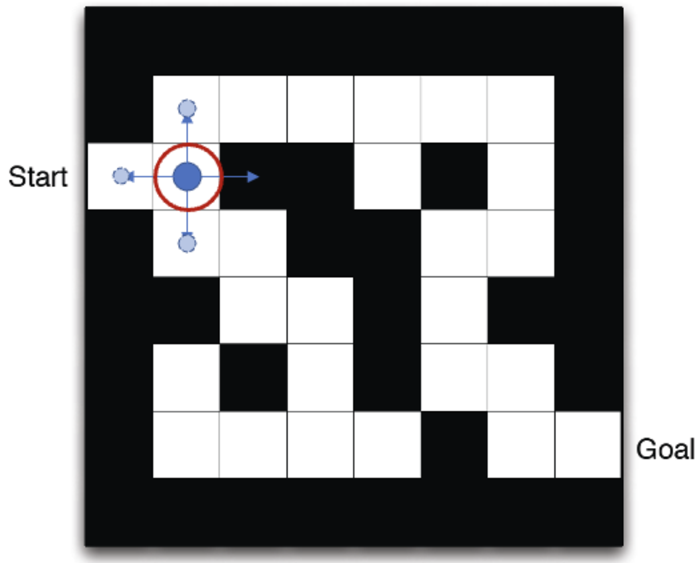


$$(s_2, a_2) + \cdots$$

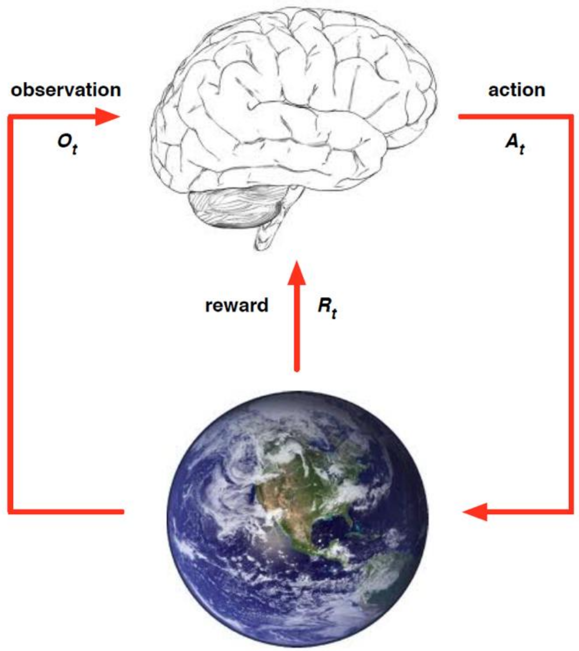
Markov Decision Process

The dynamics of an MDP proceeds as

- MDP randomly transits to some successor state $s_1 \sim P_{s_0 a_0}$



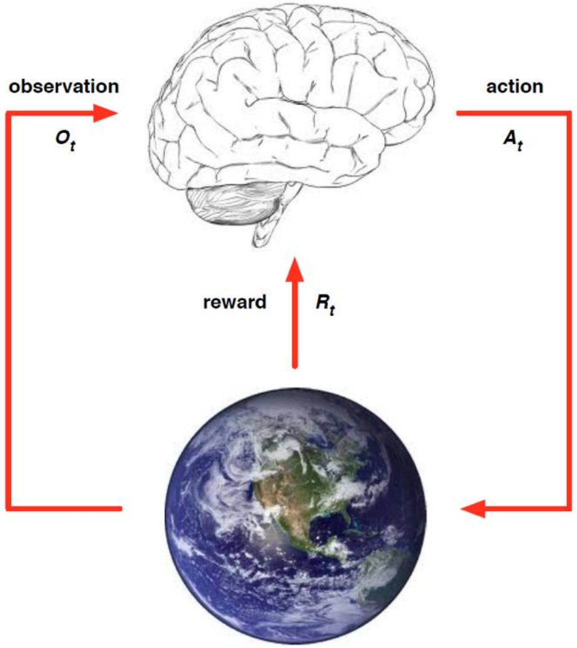
- State: agent's location
- Action: N,E,S,W
- State transition: move to the next grid according to the action
  - No move if the action is to the wall



Markov Decision Process

The dynamics of an MDP proceeds as

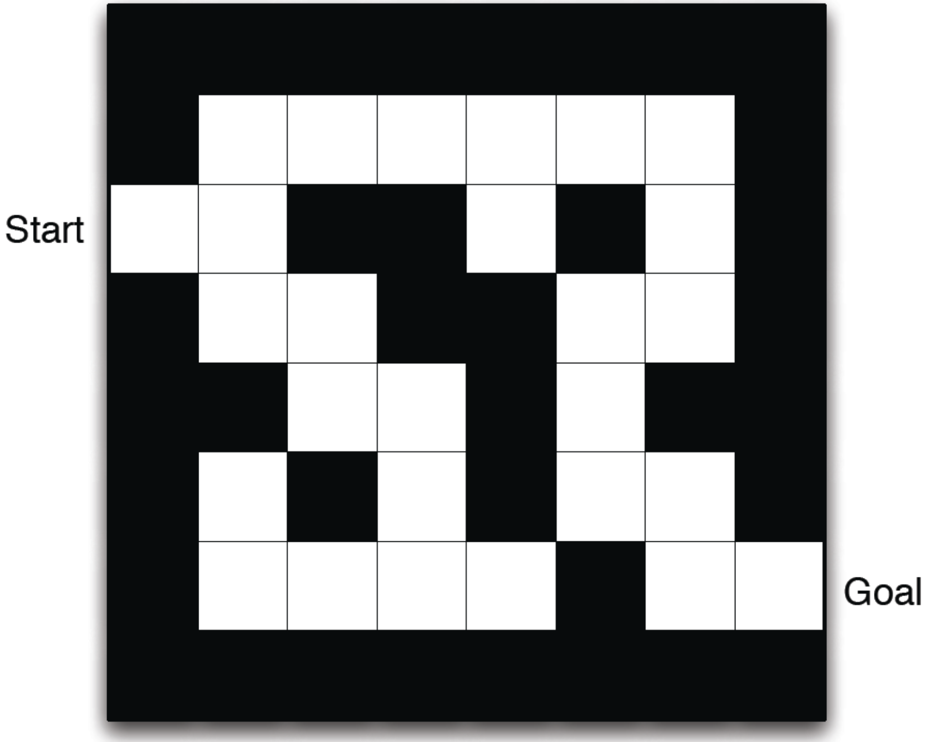
- Start in a state s_0
  - E.g., location in a maze, or current screen in an Atari game
- The agent chooses some action $a_0 \in A$
  - E.g., move N, E, S, W, or the direction of the joystick and the buttons
- The agent gets the reward $R(s_0,a_0)$
- MDP randomly transits to some successor state $s_1 \sim P_{s_0 a_0}$
- 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 $R(s_0, a_0) + \gamma R(s_1, a_1) + \gamma^2 R(s_2, a_2) + \cdots$

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

MDP Goal and How to Get Policy

- The goal is to choose actions over time to maximize the expected cumulated reward

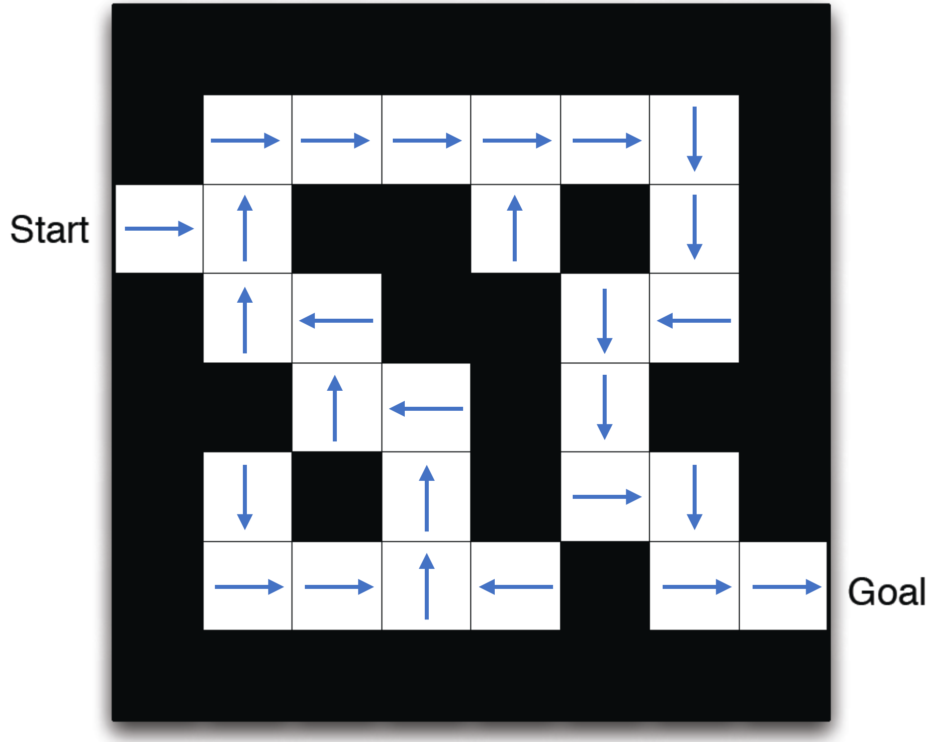
$$\mathbb{E}[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots]$$

- $\gamma \in [0,1]$ is the discount factor for the future reward, which makes the agent prefer immediate reward to future reward (care less about the rewards of states that are further in future)
  - In finance case, today's \$1 is more valuable than \$1 in tomorrow
- Given a particular policy $\pi(s) : S \mapsto A$
  - i.e. take the actior $a = \pi(s)$ 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 cumulated reward given the start state and taking actions according to π

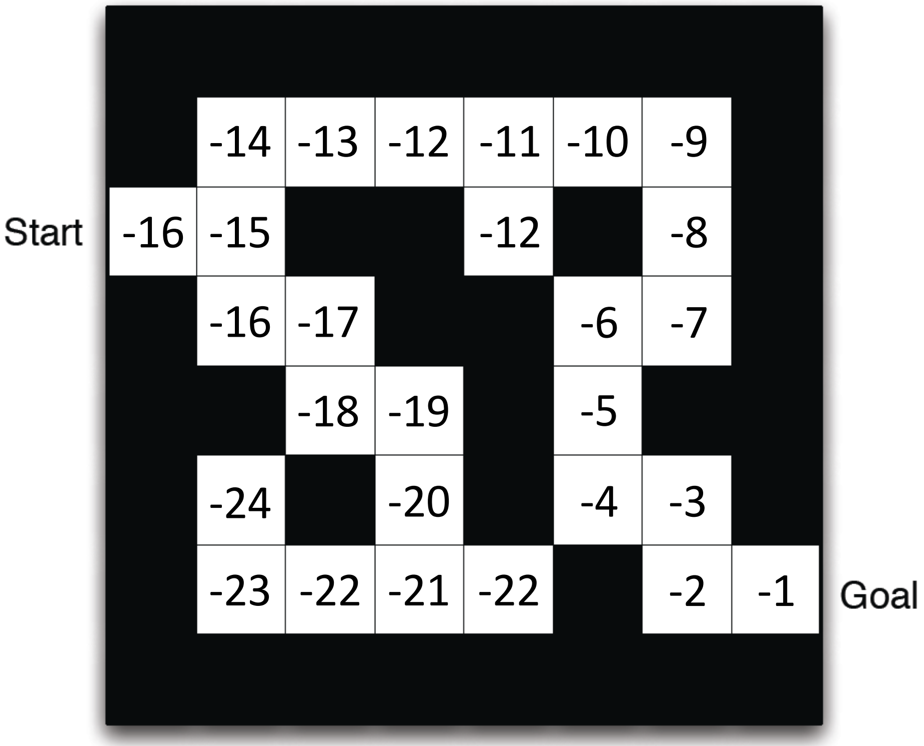
Maze Example



- 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

expected cumulated reward

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

Why its related to policy π?

Content

- Reinforcement Learning (Definition and Value-based methods)
  - The model-based methods
    - Markov Decision Process
    - **Planning by Dynamic Programming: Value Iteration and Policy Iteration**
  - The model-free methods
    - Model-free Prediction
      - Monte-Carlo and Temporal Difference
    - Model-free Control
      - On-policy SARSA and off-policy Q-learning
- Deep Reinforcement Learning

Value Function

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

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

Bellman Equation

Immediate Reward

Time decay

State transition

Rely on the policy

Value of the next state

Bertsekas, D. P., Bertsekas, D. P., Bertsekas, D. P., & Bertsekas, D. P. (1995). *Dynamic programming and optimal control* (Vol. 1, No. 2). Belmont, MA: Athena scientific.

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 optimality equation for optimal value function

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

- The optimal policy

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

- For every state *s* and every policy

$$V^*(s) = V^{\pi^*}(s) \geq V^{\pi}(s)$$

Dynamic Programming

- Note that the value function and policy are correlated

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

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

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

Value Iteration
– Bellman Optimality Function

- For an MDP with finite state and action spaces

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

- Value iteration is performed as

bellmen optimal function

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

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

   For each state, update V by bellmen optimal function

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

}

- Note that there is no explicit policy in above calculation

Value Iteration Example: Shortest Path

Terminal state / goal

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

Problem

g			

V_1

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

Initialize: All states are 0

V_2

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

V_3

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

V_4

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

V_5

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

V_6

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

V_7

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

Value Iteration Example: Shortest Path

Terminal state / goal

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

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

R(s) = -1 (penalty of move) max V(s') = 0

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

Value Iteration Example: Shortest Path

Terminal state / goal

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

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

R(s) = -1 max V(s') = -1

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 { $V := V^{\pi}$
     a) Evaluate value function by bellman expectation equation
     b) For each state, update

$$\pi(s) = \arg\max_{a \in A} \sum_{s' \in S} P_{sa}(s') V(s')$$

}

- The step of value function update could be time-consuming

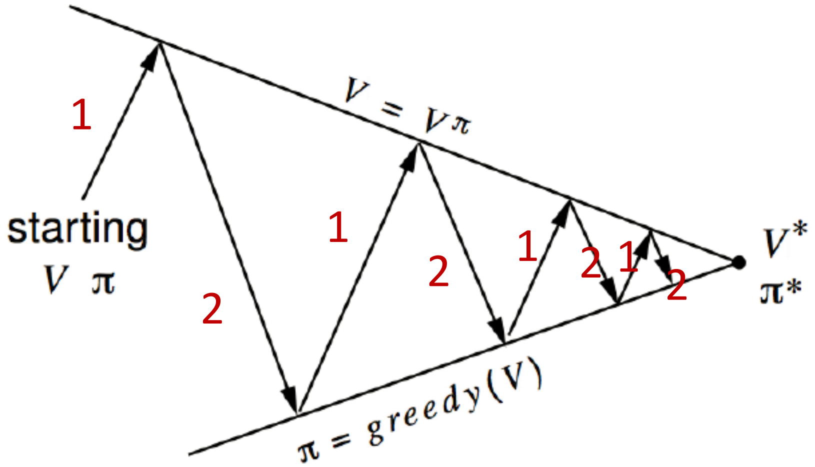
1. Evaluate V by current policy

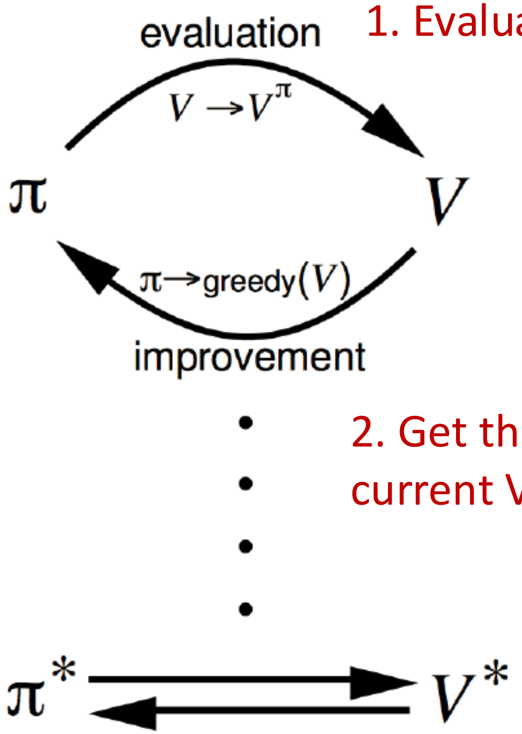
$$\boxed{\begin{aligned} V^{\pi}(s) &= R(s) + \gamma \sum_{s' \in S} P_{s\pi(s)}(s') V^{\pi}(s') \\ \pi(s) &= \arg\max_{a \in A} \sum_{s' \in S} P_{sa}(s') V^{\pi}(s') \end{aligned}}$$

bellmen expectation function: two steps

2. Get the best policy based on current V

Policy Iteration



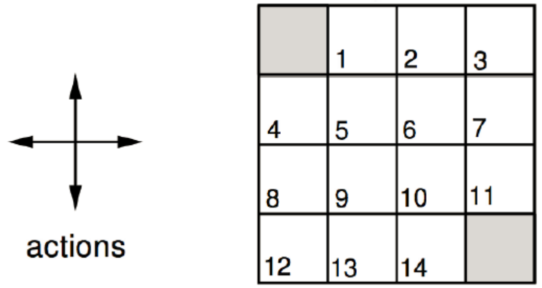


1. Evaluate V by current policy

2. Get the best policy based on current V

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

Evaluating a Random Policy in the Small Gridworld

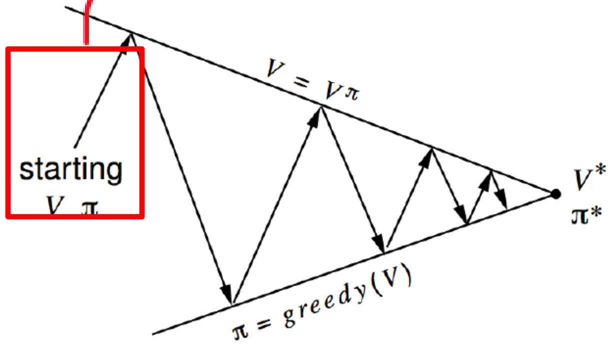


r = -1
on all transitions

- Undiscounted episodic MDP (γ=1)
- Nonterminal states 1,…,14
- Two terminal states (shaded squares)
- Actions leading out of the grid leave state unchanged
- Reward is -1 until the terminal state is reached
- Agent follows a uniform random policy

$$\pi(n|\cdot) = \pi(e|\cdot) = \pi(s|\cdot) = \pi(w|\cdot) = 0.25$$

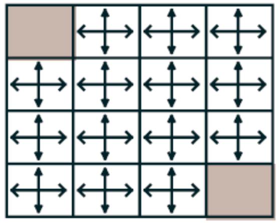
Evaluating a Random Policy in the Small Gridworld



V_k for the random policy	Greedy policy w.r.t. V_k

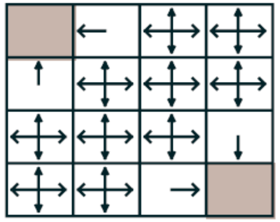
K=0

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0



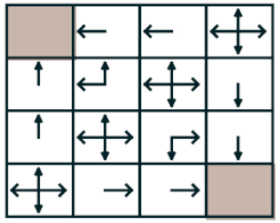
K=1

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0



K=2

0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0



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

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

Random policy

1. Initialize π randomly
2. Repeat until convergence { $V := V^\pi$
   a) Evaluate value function by bellman expectation equation
   b) For each state, update

$$\pi(s) = \arg\max_{a \in A} \sum_{s' \in S} P_{sa}(s') V(s')$$

}

Evaluating a Random Policy in the Small Gridworld

V_k for the random policy

K=3

0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

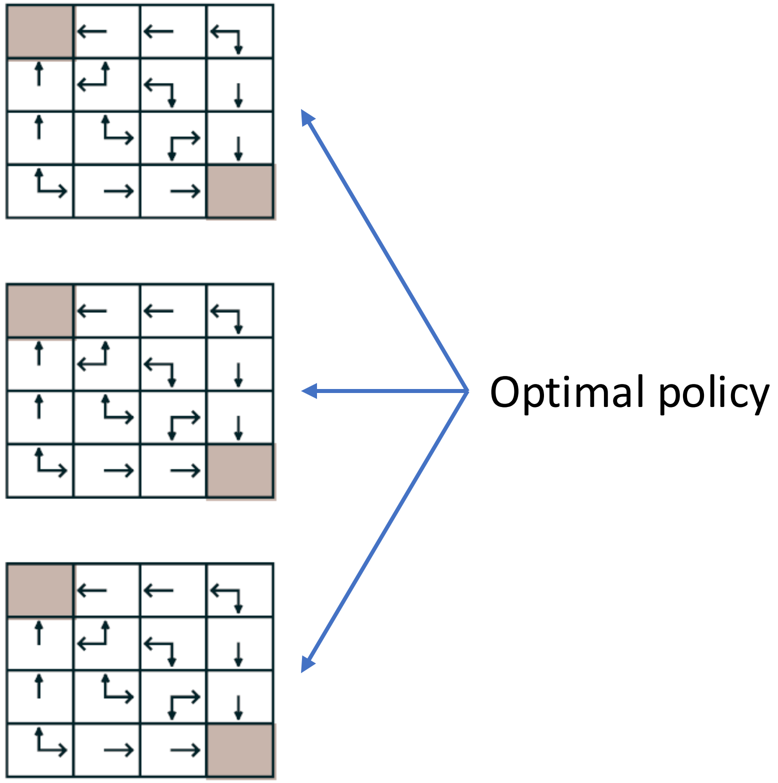
K=10

0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0

$K=\infty$

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

Greedy policy w.r.t. V_k



Value Iteration vs. Policy Iteration

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)

Content

- Reinforcement Learning (Definition and Value-based methods)
  - The model-based methods
    - **Markov Decision Process**
    - Planning by Dynamic Programming
  - The model-free methods
    - Model-free Prediction
      - Monte-Carlo and Temporal Difference
    - Model-free Control
      - On-policy SARSA and off-policy Q-learning
- Deep Reinforcement Learning

Value Function

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

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

Bellman Equation

Immediate Reward

Time decay

State transition

Rely on the policy

Value of the next state

Bertsekas, D. P., Bertsekas, D. P., Bertsekas, D. P., & Bertsekas, D. P. (1995). *Dynamic programming and optimal control* (Vol. 1, No. 2). Belmont, MA: Athena scientific.

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
  - For example, we have only observed some episodes

Episode 1: $s_0^{(1)} \xrightarrow[R(s_0)^{(1)}]{a_0^{(1)}} s_1^{(1)} \xrightarrow[R(s_1)^{(1)}]{a_1^{(1)}} s_2^{(1)} \xrightarrow[R(s_2)^{(1)}]{a_2^{(1)}} s_3^{(1)} \cdots s_T^{(1)}$

Episode 2: $s_0^{(2)} \xrightarrow[R(s_0)^{(2)}]{a_0^{(2)}} s_1^{(2)} \xrightarrow[R(s_1)^{(2)}]{a_1^{(2)}} s_2^{(2)} \xrightarrow[R(s_2)^{(2)}]{a_2^{(2)}} s_3^{(2)} \cdots s_T^{(2)}$

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

$$R(s) = \text{average}\left\{R(s)^{(i)}\right\}$$

- 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

     }

Model-free Reinforcement Learning

- In realistic problems, often the state transition and reward function are not explicitly given
  - For example, we have only observed some episodes

Episode 1: $s_0^{(1)} \xrightarrow[R(s_0)^{(1)}]{a_0^{(1)}} s_1^{(1)} \xrightarrow[R(s_1)^{(1)}]{a_1^{(1)}} s_2^{(1)} \xrightarrow[R(s_2)^{(1)}]{a_2^{(1)}} s_3^{(1)} \cdots s_T^{(1)}$

Episode 2: $s_0^{(2)} \xrightarrow[R(s_0)^{(2)}]{a_0^{(2)}} s_1^{(2)} \xrightarrow[R(s_1)^{(2)}]{a_1^{(2)}} s_2^{(2)} \xrightarrow[R(s_2)^{(2)}]{a_2^{(2)}} s_3^{(2)} \cdots s_T^{(2)}$

- Model-free RL is to directly learning value & policy from experience without building a model
- 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

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

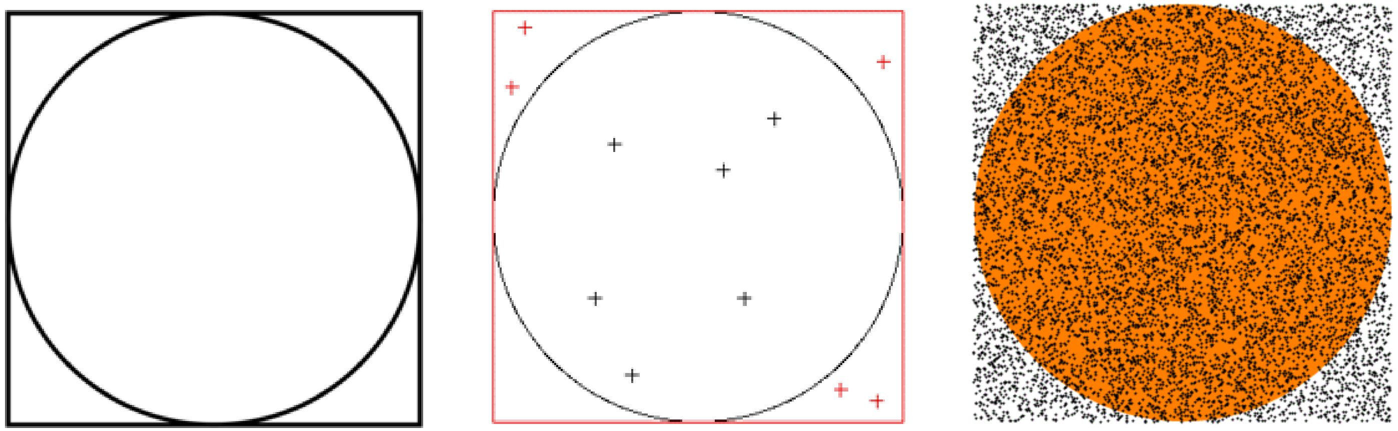
- Now in model-free RL
  - We cannot directly know P_{sa} and R
  - But we have a list of experiences to estimate the values

Episode 1: $s_0^{(1)} \xrightarrow[R(s_0)^{(1)}]{a_0^{(1)}} s_1^{(1)} \xrightarrow[R(s_1)^{(1)}]{a_1^{(1)}} s_2^{(1)} \xrightarrow[R(s_2)^{(1)}]{a_2^{(1)}} s_3^{(1)} \cdots s_T^{(1)}$

Episode 2: $s_0^{(2)} \xrightarrow[R(s_0)^{(2)}]{a_0^{(2)}} s_1^{(2)} \xrightarrow[R(s_1)^{(2)}]{a_1^{(2)}} s_2^{(2)} \xrightarrow[R(s_2)^{(2)}]{a_2^{(2)}} s_3^{(2)} \cdots s_T^{(2)}$

Monte-Carlo Methods

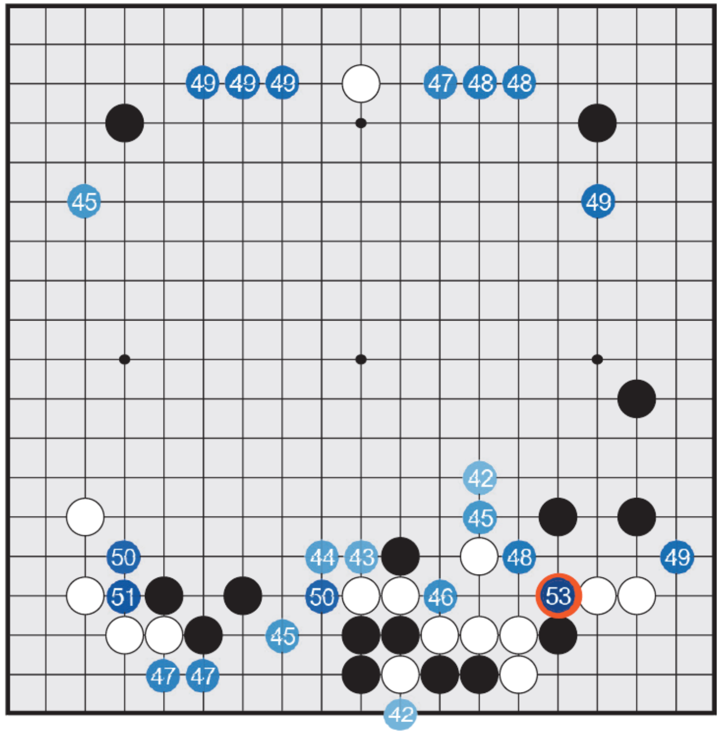
- Monte-Carlo methods are a broad class of computational algorithms that rely on repeated random sampling to obtain numerical results.
- For example, to calculate the circle's surface

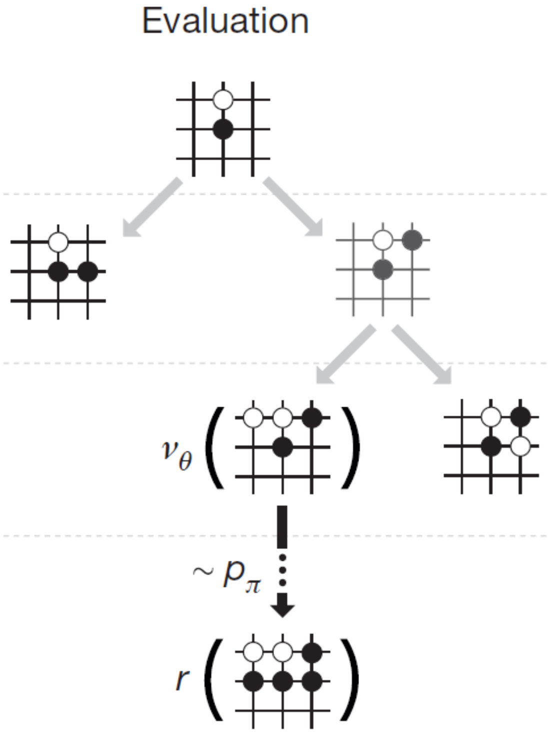


$$\text{Circle Surface} = \text{Square Surface} \times \frac{\#\text{points in circle}}{\#\text{points in total}}$$

Monte-Carlo Methods

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





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

Monte-Carlo Value Estimation

- Goal: learn V^π from episodes of experience under policy π

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

- Recall that the return is the total discounted reward

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

- 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 cumulated reward

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

Monte-Carlo Value Estimation

- Implementation
  - – Sample episodes 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 $N(s) \leftarrow N(s) + 1$
  - Increment total return $S(s) \leftarrow S(s) + G_t$
  - Value is estimated by mean return $V(s) = S(s)/N(s)$
  - By law of large numbers $V(s) \rightarrow V^{\pi}(s)$ as $N(s) \rightarrow \infty$

Monte-Carlo Value Estimation

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

Implementation: $$V(S_t) \leftarrow V(S_t) + \alpha(G_t - V(S_t))$$

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

Bellman Equation for Value Function

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

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

Bellman Equation

Immediate Reward

Time decay

State transition

Value of the next state

Bertsekas, D. P., Bertsekas, D. P., Bertsekas, D. P., & Bertsekas, D. P. (1995). *Dynamic programming and optimal control* (Vol. 1, No. 2). Belmont, MA: Athena scientific.

Temporal-Difference Learning

- TD methods learn directly from episodes of experience
- TD is model-free: no knowledge of MDP transitions / rewards
- TD learns from incomplete episodes, by bootstrapping

Sutton, Richard S. "Learning to predict by the methods of temporal differences." *Machine learning* 3.1 (1988): 9-44.

Monte Carlo vs. Temporal Difference

- The same goal: learn V^{π} from episodes of experience under policy π
- Incremental every-visit Monte-Carlo
  - Update value $V(S_t)$ toward actual return G_t

$$V(S_t) \leftarrow V(S_t) + \alpha(G_t - V(S_t))$$

- Simplest temporal-difference learning algorithm: TD(1)
  - Update value $V(S_t)$ toward estimated return $R_{t+1} + \gamma V(S_{t+1})$

$$V(S_t) \leftarrow V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$$

TD updates a guess towards a guess

guess

  - TD target: $R_{t+1} + \gamma V(S_{t+1})$
  - TD error: $\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$

Advantages and Disadvantages of MC vs. TD

- TD can learn before knowing the final outcome
  - TD can learn online after **every step**
  - MC must **wait until end of episode** before return is known

- TD can learn without the final outcome
  - TD can learn from incomplete sequences
  - MC can only learn from complete sequences
  - TD works in continuing (non-terminating) environments
  - MC only works for episodic (terminating) environments

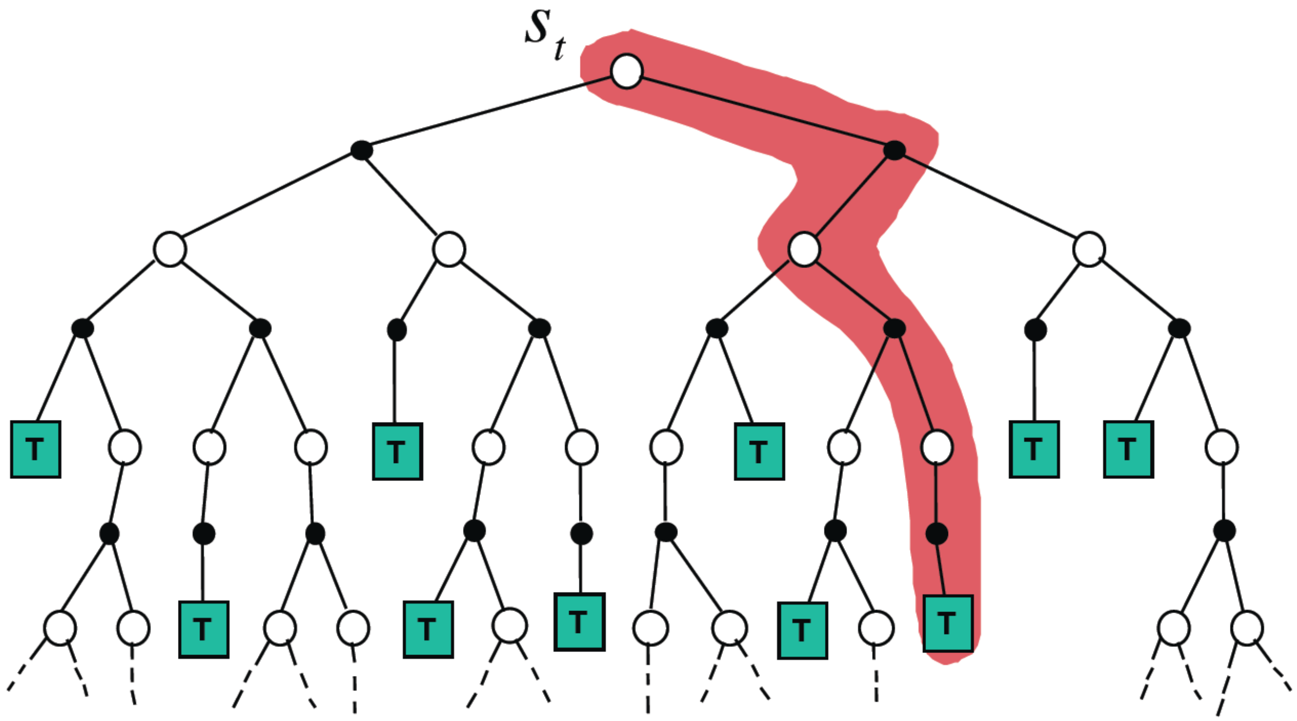
Advantages and Disadvantages of MC vs. TD (2)

- MC has high variance, zero bias
  - Good convergence properties
  - (even with function approximation)
  - Not very sensitive to initial value
  - Very simple to understand and use

- TD has low variance, some bias
  - Usually more efficient than MC
  - TD converges to $V^{\pi}(S_t)$
    - (but not always with function approximation)
  - More sensitive to initial value then MC

Monte-Carlo Backup

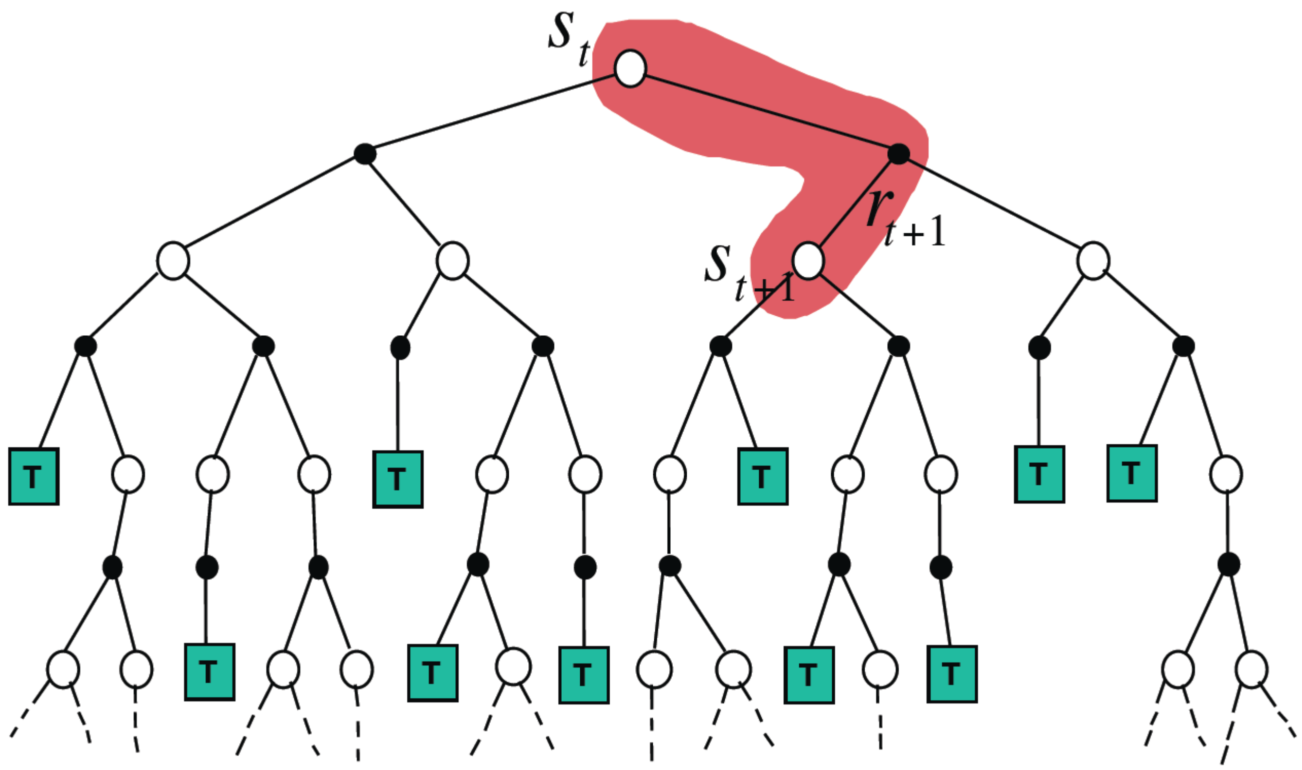
$$V(S_t) \leftarrow V(S_t) + \alpha(G_t - V(S_t))$$



Backup: Updating the value of a state using values of future states

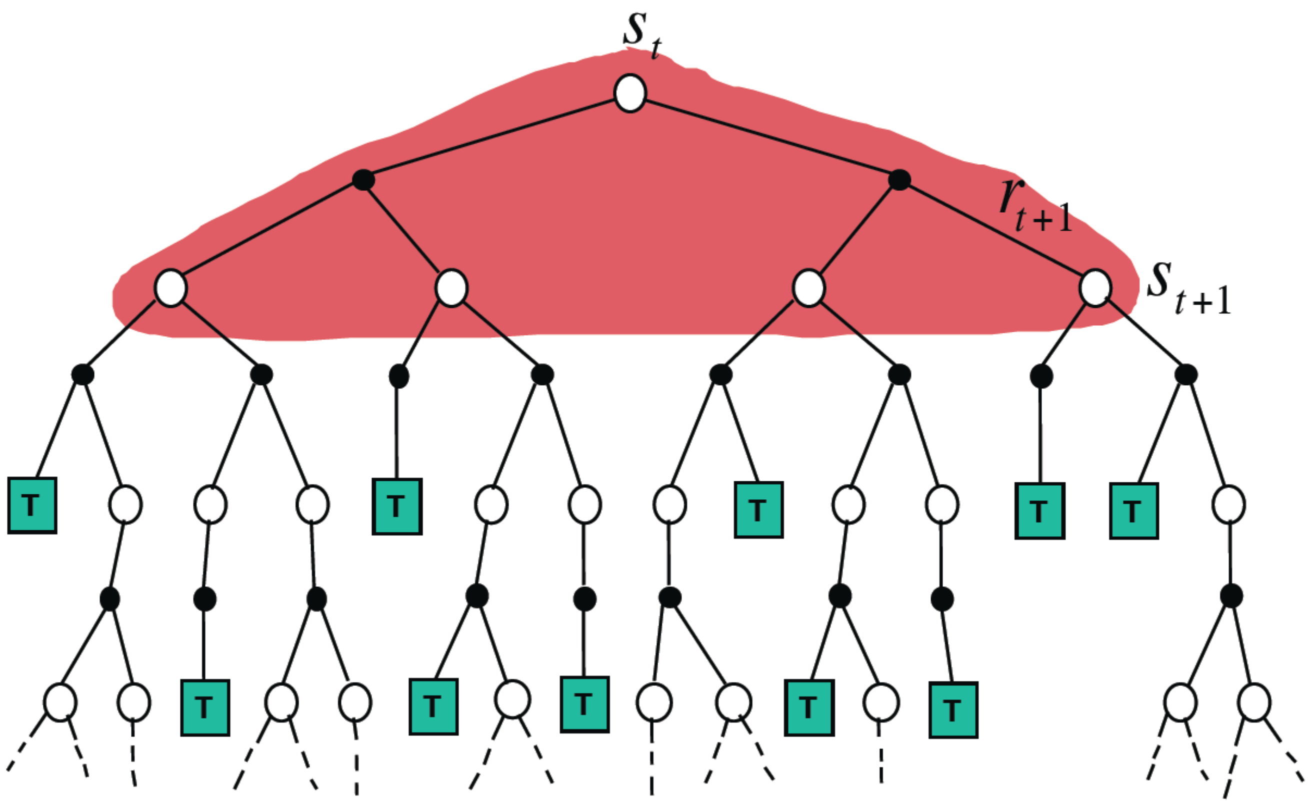
Temporal-Difference Backup

$$V(S_t) \leftarrow V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$$



Dynamic Programming Backup

$$V(S_t) \leftarrow \mathbb{E}[R_{t+1} + \gamma V(S_{t+1})]$$



Backup: Updating the value of a state using values of future states

Content

- Reinforcement Learning
  - The model-based methods
    - Markov Decision Process
    - Planning by Dynamic Programming
  - The model-free methods
    - Model-free Prediction
      - Monte-Carlo and Temporal Difference
    - **Model-free Control (Evaluation and Improvement)**
      - **On-policy SARSA and off-policy Q-learning**

State Value and Action Value

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

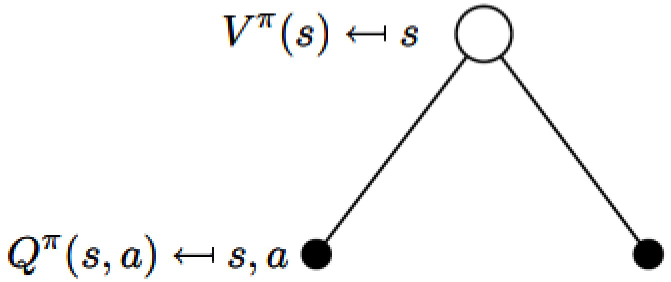
- State value
  - The state-value function $V^\pi(s)$ of an MDP is the expected return starting from state s and then following policy π

$$V^\pi(s) = \mathbb{E}_\pi[G_t | S_t = s]$$

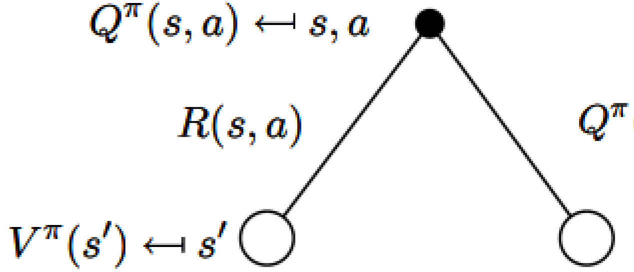
- State-Action value
  - The action-value function $Q^\pi(s,a)$ of an MDP is the expected return starting from state s, taking action a, and then following policy π

$$Q^\pi(s,a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a]$$

State Value and Action Value



$$V^\pi(s) = \sum_{a \in A} \pi(a|s) Q^\pi(s,a)$$



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

Bellman Equation

- Bellman expectation equation

$$Q_\pi(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \sum_{a' \in \mathcal{A}} \pi(a' \mid s') \, Q_\pi(s', a')$$

- Bellman optimality equation

$$Q_*(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \max_{a'} Q_*(s', a')$$

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

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

- The optimal policy

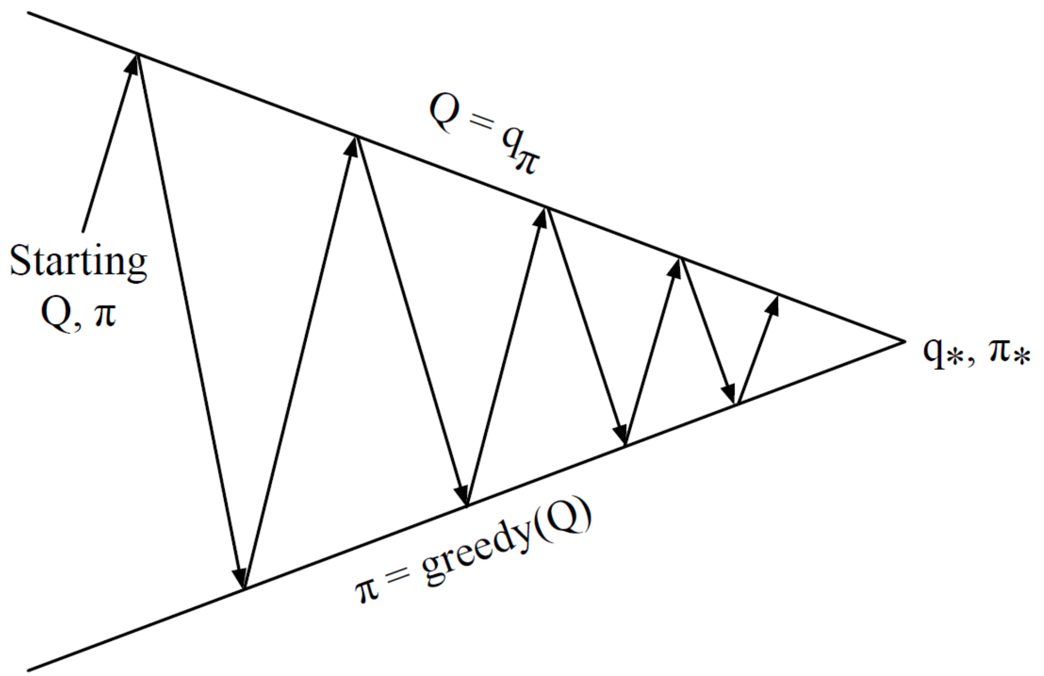
$$\pi^*(s) = \arg\max_{a \in A} \sum_{s' \in S} P_{sa}(s') V^*(s')$$

- For every state *s* and every policy

$$V^*(s) = V^{\pi^*}(s) \geq V^\pi(s)$$

Generalized Policy Iteration with Action-Value Function

- One maintains both an approximate policy and an approximate value function.
- Value function updated for the current policy, while the policy is improved with respect to the current value function
- however assume policy evaluation operates on an infinite number of episodes



- Policy evaluation: Monte-Carlo/TD policy evaluation, $Q = Q^\pi$
- Policy improvement: argmax/ε-greedy policy improvement

$$\pi(a|s) = \begin{cases} \epsilon/m + 1 - \epsilon & \text{if } a^* = \arg\max_{a \in A} Q(s,a) \\ \epsilon/m & \text{otherwise} \end{cases}$$

On- and Off-Policy Learning

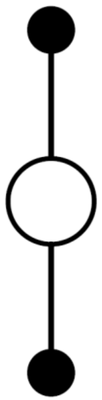
- Two categories of model-free RL

- On-policy learning
  - "Learn on the job"
  - Learn about policy π from experience sampled from π

- Off-policy learning
  - "Look over someone's shoulder"
  - Learn about policy π from experience sampled from another policy μ

SARSA (On-Policy TD Control)

- For each State-Action-Reward-State-Action by the current policy



At state s, take action a

Observe reward r

Transit to the next state s'

At state s', take action a'

- Bellman expectation equation

$$Q_\pi(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \sum_{a' \in \mathcal{A}} \pi(a' \mid s') \, Q_\pi(s', a')$$

- Simplest temporal-difference learning algorithm: TD(1)
  - Update value $V(S_t)$ toward estimated return $R_{t+1} + \gamma V(S_{t+1})$

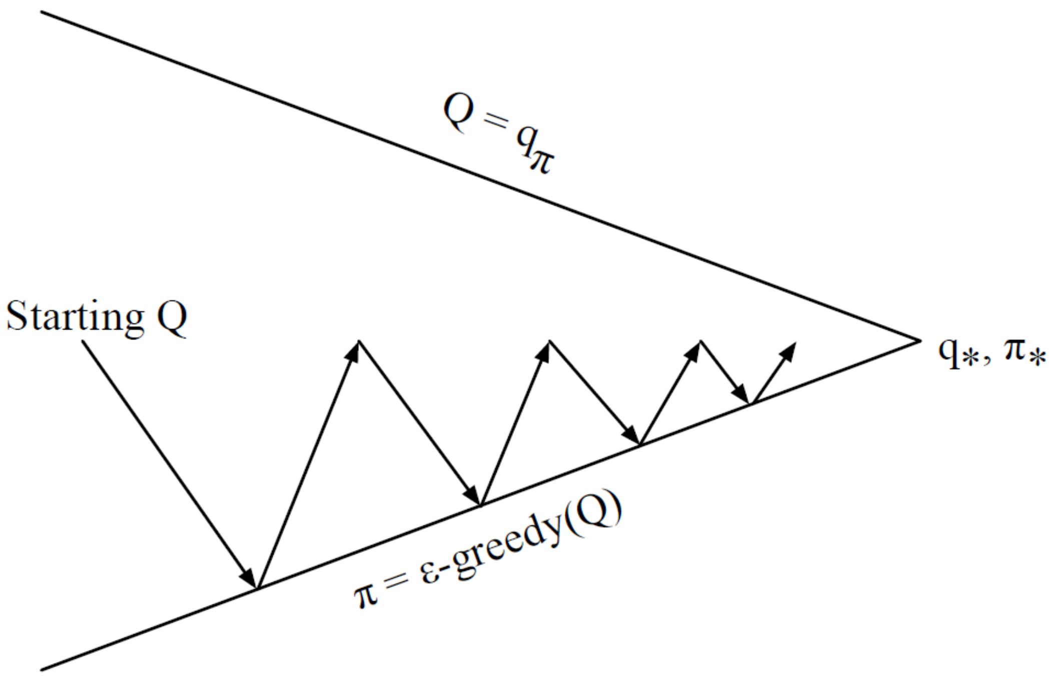
$$V(S_t) \leftarrow V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$$

- Updating action-value functions with Sarsa

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

- This is on-policy, as it learns about the value function of π from experience sampled from π

On-Policy Control with SARSA



Every time-step and update for a single state:

- Policy evaluation: Sarsa $Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma Q(s',a') - Q(s,a))$
- Policy improvement: ε-greedy policy improvement

SARSA Algorithm

Sarsa: An on-policy TD control algorithm

Initialize $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
    Initialize S
    Choose A from S using policy derived from Q (e.g., ϵ-greedy)
    Repeat (for each step of episode):
        Take action A, observe R, S'
        Choose A' from S' using policy derived from Q (e.g., ϵ-greedy)
        $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') - Q(S, A)\big]$
        $S \leftarrow S'$; $A \leftarrow A'$;
    until S is terminal

- NOTE: on-policy TD control sample actions by the current policy, i.e., the two ‘A’s in SARSA are both chosen by the current policy

Q-Learning

- Q-learning is trying to estimate optimal state-action value function, based on the optimal Bellman equation
    - Bellman optimality equation

$$Q_*(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \max_{a'} Q_*(s',a')$$

- Q-learning update rule is:

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha(r_{t+1} + \gamma \max_{a'} Q(s_{t+1},a') - Q(s_t,a_t))$$

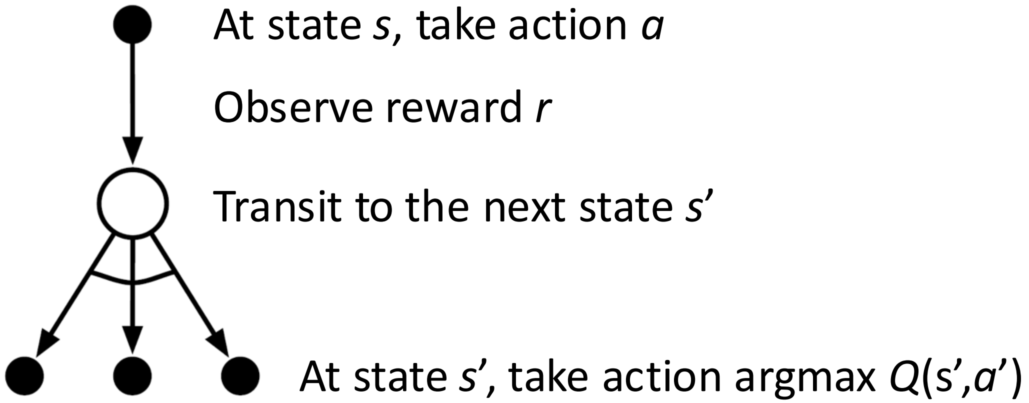
- Q-learning is an off-policy control, the behavior policy and target policy are different

Q-Learning

- Q-learning is trying to estimate optimal state-action value function, based on the Bellman optimality equation
    - Bellman optimality equation

$$Q_*(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \max_{a'} Q_*(s',a')$$

- Q-learning update rule is:



$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha(r_{t+1} + \gamma \max_{a'} Q(s_{t+1},a') - Q(s_t,a_t))$$

Off-Policy Control with Q-Learning

- The target policy π is greedy w.r.t. $Q(s,a)$

$$\pi(s_{t+1}) = \arg\max_{a'} Q(s_{t+1}, a')$$

- The behavior policy μ is e.g. ε-greedy policy w.r.t. $Q(s,a)$
- Q-learning is an off-policy control
  - Learning from SARS generated by another policy μ
  - The first action a and the corresponding reward r are from μ
  - The next action a' is picked by the target policy
- Why no importance sampling?
  - Bellman optimality function instead of expectation function
    - Bellman optimality equation

$$Q_*(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \max_{a'} Q_*(s', a')$$

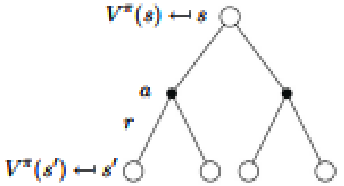
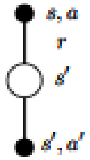
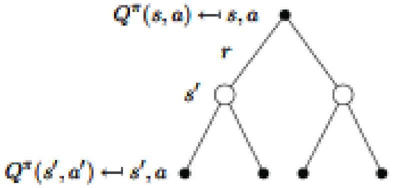
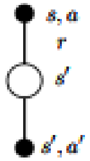
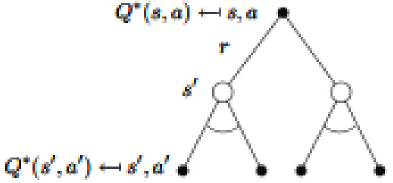
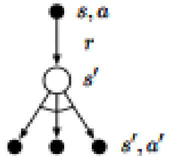
Off-Policy Control with Q-Learning

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(\mathit{terminal}, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$
  until S is terminal

Relationship Between DP and TD

	Full Backup (DP)	Sample Backup (TD)
Bellman Expectation Equation for $V^{\pi}(s)$	 Iterative Policy Evaluation	 TD Learning
Bellman Expectation Equation for $Q^{\pi}(s,a)$	 Q-Policy Iteration	 SARSA
Bellman Optimality Equation for $Q^{*}(s,a)$	 Q-Value Iteration	 Q-Learning

Relationship Between DP and TD

Full Backup (DP)	Sample Backup (TD)
Iterative Policy Evaluation $V(s) \leftarrow \mathbb{E}[r + \gamma V(s')\|s]$	TD Learning $V(s) \xleftarrow{\alpha} r + \gamma V(s')$
Q-Policy Iteration $Q(s,a) \leftarrow \mathbb{E}[r + \gamma Q(s',a')\|s,a]$	SARSA $Q(s,a) \xleftarrow{\alpha} r + \gamma Q(s',a')$
Q-Value Iteration $Q(s,a) \leftarrow \mathbb{E}\left[r + \gamma \max_{a'} Q(s',a')\|s,a\right]$	Q-Learning $Q(s,a) \xleftarrow{\alpha} r + \gamma \max_{a'} Q(s',a')$

where

$$x \xleftarrow{\alpha} y \equiv x \leftarrow x + \alpha(y - x)$$

Content

- Reinforcement Learning
  - The model-based methods
    - Markov Decision Process
    - Planning by Dynamic Programming
  - The model-free methods
    - Model-free Prediction
      - Monte-Carlo and Temporal Difference
    - Model-free Control
      - SARSA and Q-learning
- **Deep Reinforcement Learning**

Deep Q-learning

- However, in many cases, it is not practical as
  - *Action x State* space is very large
  - lacking generalisation
- *Q* function is commonly approximated by a function, either linear or non-linear

$$Q(s,a;\theta) \approx Q^*(s,a) \quad \text{where } \theta \text{ is model parameter}$$

- It can be trained by minimising a sequence of loss function

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha(r_{t+1} + \gamma \max_{a'} Q(s_{t+1},a') - Q(s_t,a_t))$$

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s')\sim \mathrm{U}(D)}\left[\left(\underbrace{r+\gamma \max_{a'} Q(s',a';\theta_i^-)}_{\text{target}} - Q(s,a;\theta_i)\right)^2\right]$$

where $U(D)$ is sample distribution from a pool of stored samples and θ_i^- only updates every C steps

Mnih V, Kavukcuoglu K, Silver D, et al. Human-level control through deep reinforcement learning[J]. Nature, 2015, 518(7540): 529-533.

Deep Q-learning

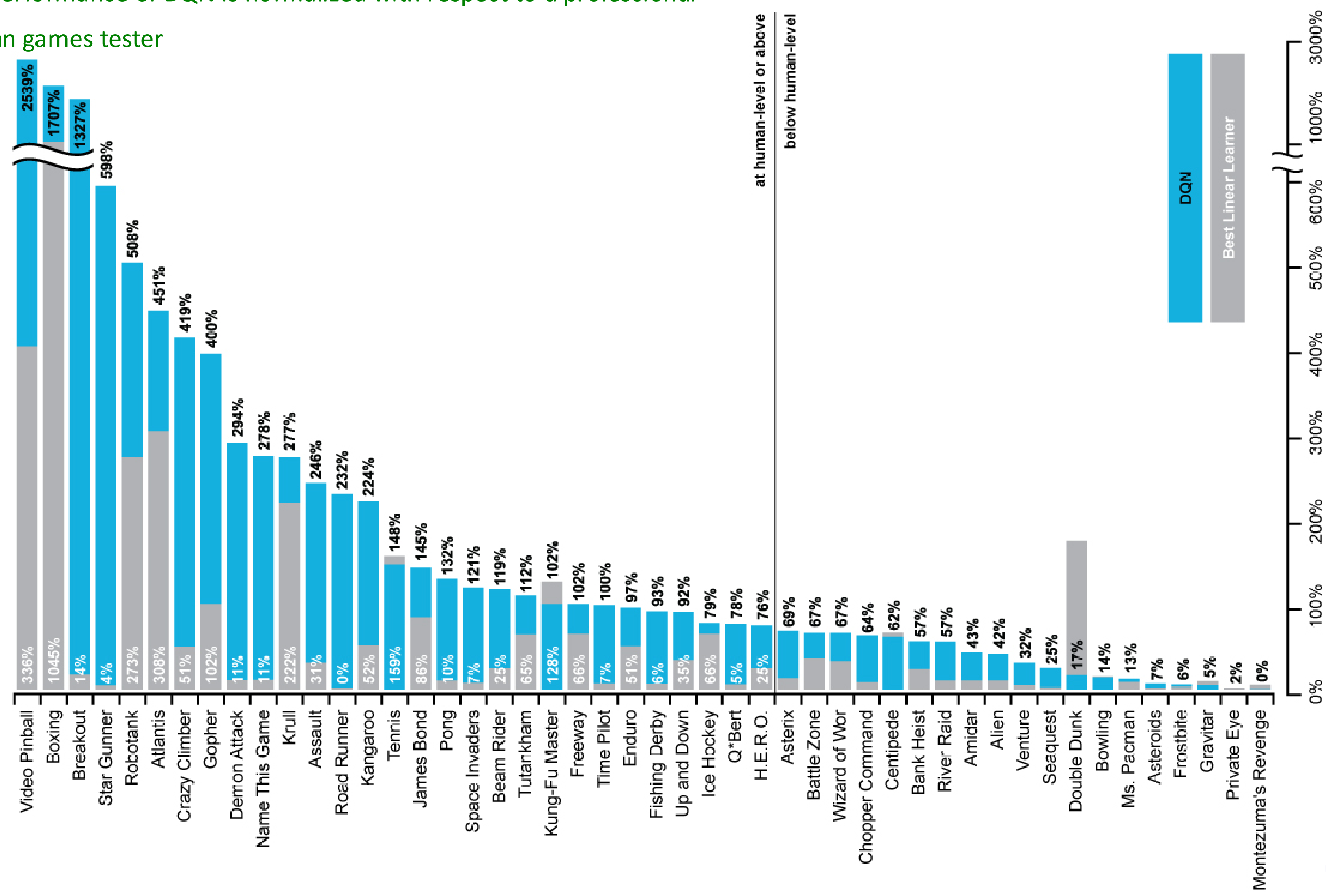
Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

Mnih V, Kavukcuoglu K, Silver D, et al. Human-level control through deep reinforcement learning[J]. Nature, 2015, 518(7540): 529-533.
Mnih V, Kavukcuoglu K, Silver D, et al. Playing atari with deep reinforcement learning[J]. arXiv preprint arXiv:1312.5602, 2013.

DQN results on atari games

The performance of DQN is normalized with respect to a professional human games tester



Mnih V, Kavukcuoglu K, Silver D, et al. Human-level control through deep reinforcement learning[J]. Nature, 2015, 518(7540): 529-533.