Single-agent Reinforcement Learning (Value-based approaches)

Prof. Jun Wang

Computer Science, UCL

Recap

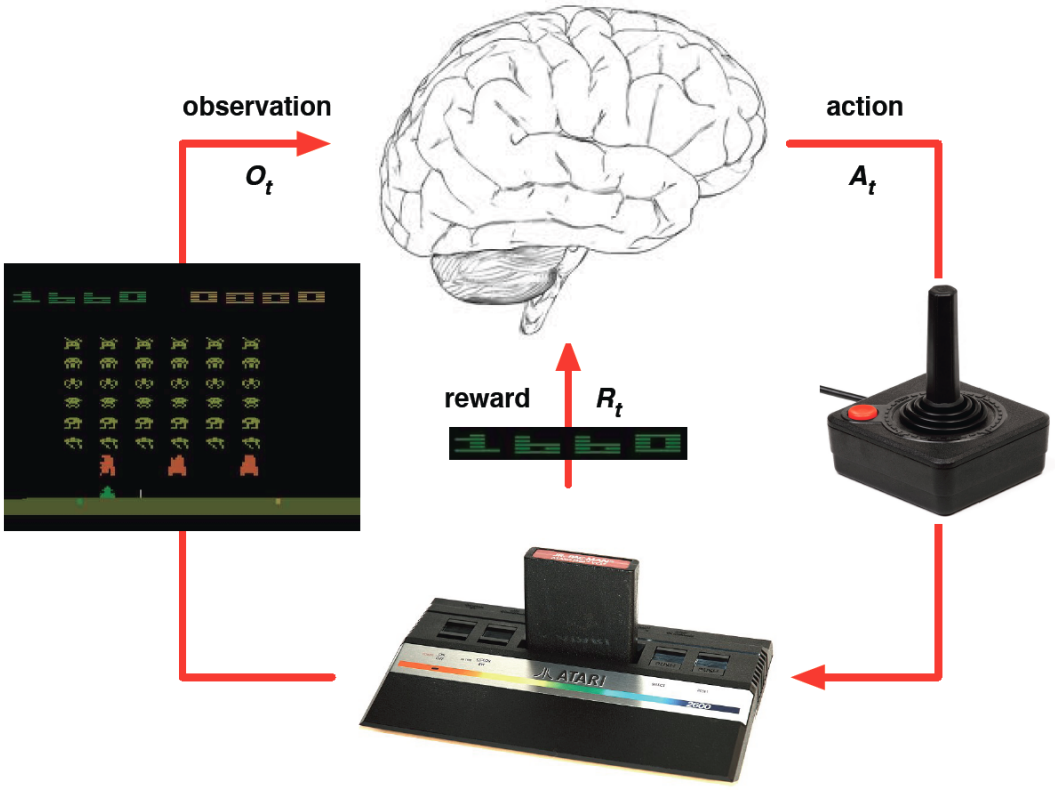
- Lecture 1: Multiagent AI and basic game theory
- Lecture 2: Potential games, and extensive form and repeated games
- Lecture 3: Solving (“Learning”) Nash Equilibria
- Lecture 4: Bayesian Games, auction theory and mechanism design
- Lecture 5: Learning and deep neural networks
- Lecture 6: Single-agent Learning (1)
- Lecture 7: Single-agent Learning (2)
- Lecture 8: Multi-agent Learning (1)
- Lecture 9: Multi-agent Learning (2)
- Lecture 10: Multi-agent Learning (3)

Subject to change

learning in single agent

- A typical AI concerns the learning performed by an individual agent
- In that setting, the goal is to design an agent that learns to function successfully in an environment that is unknown and potentially also changes as the agent is learning
  - Learning to recommend in collaborative filtering
  - Learning to predict click-through rate by logistic regression
  - Learning to take actions: **reinforcement learning**

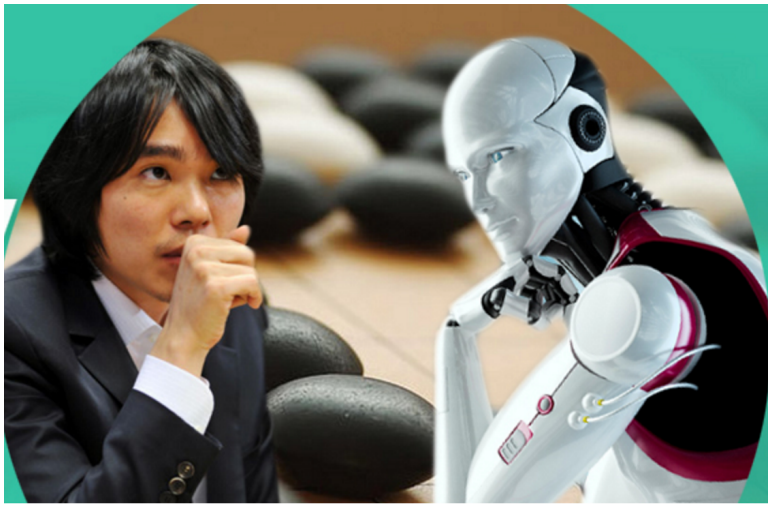
Learning to take actions: AI plays Atari Games

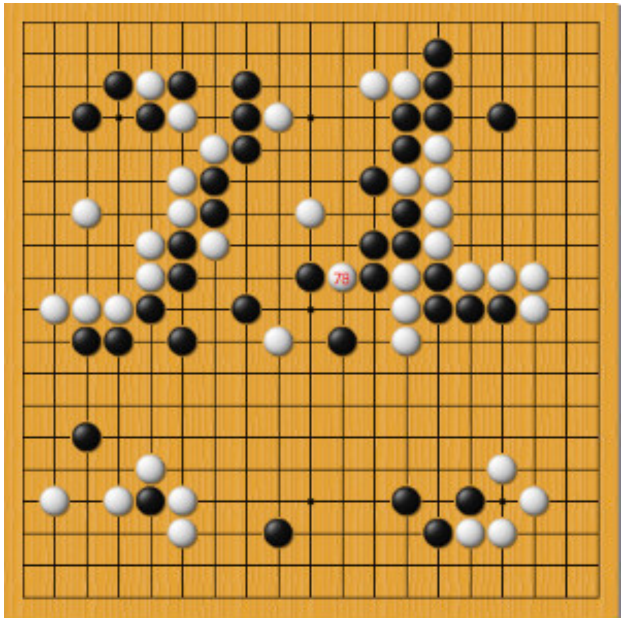


- Rules of the game are unknown
- Learn from interactive game-play
- Pick actions on joystick, see pixels and scores

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/

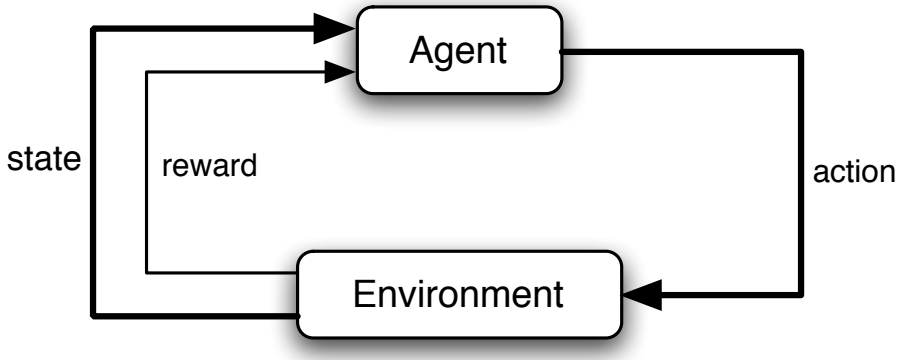
Coulom, Rémi. "Whole-history rating: A bayesian rating system for players of time-varying strength." Computers and games. Springer Berlin Heidelberg, 2008. 113-124.

Content

- Reinforcement Learning
  - The model-based methods
    - Markov Decision Process
    - Planning by Dynamic Programming
    - Convergence Analysis
  - The model-free methods
    - Model-free Prediction
      - Monte-Carlo and Temporal Difference
    - Model-free Control
      - On-policy SARSA and off-policy Q-learning
      - Convergence Analysis
- Deep Reinforcement Learning Examples
  - Atari games
  - Alpha Go

Reinforcement Learning

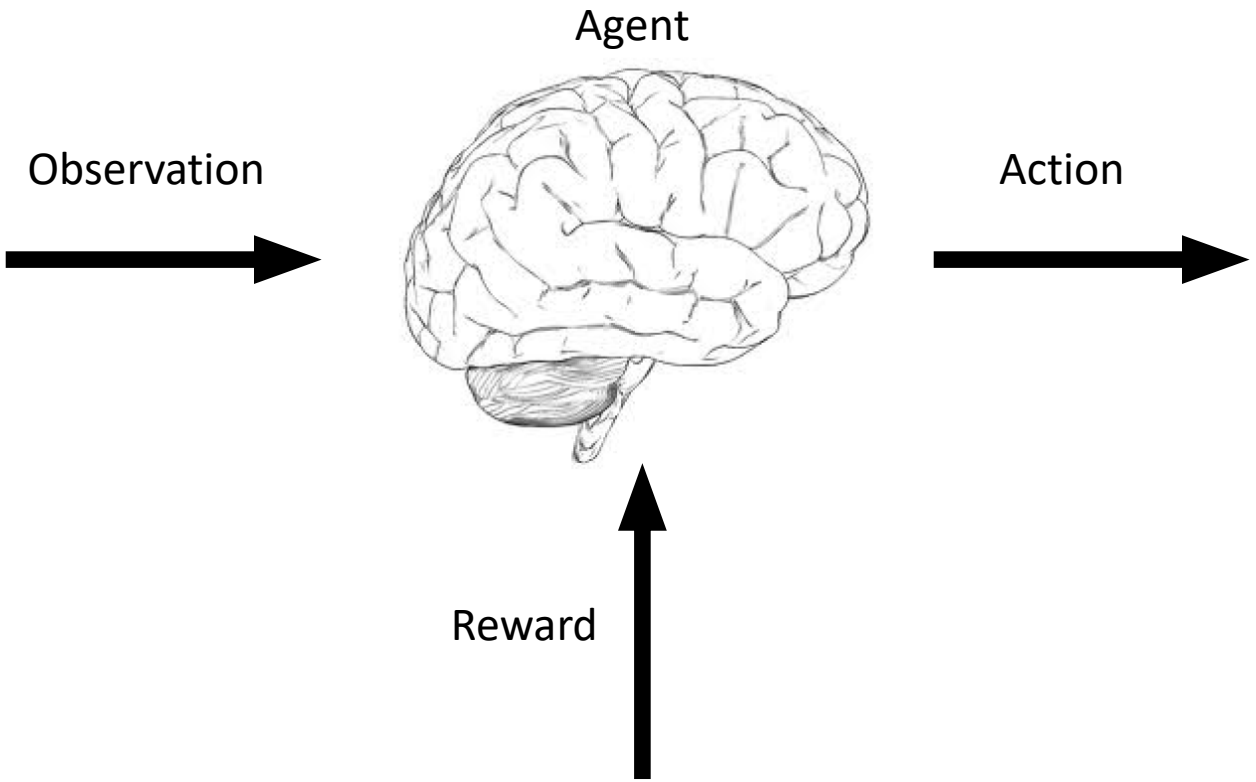
- Reinforcement learning (RL)
  - learning to take actions over an environment so as to maximize some numerical value which represents a long-term objective.
- The single agent reinforcement learning framework
  - A agent receives an environment's state and a reward associated with the last state transition.
  - It then calculates an action which is sent back to the system.
  - In response, the system makes a transition to a new state and the cycle is repeated.
  - The problem is to learn a way of controlling the environment so as to maximize the total reward.



Neto, Gonçalo. "From single-agent to multi-agent reinforcement learning: Foundational concepts and methods." *Learning theory course* (2005).

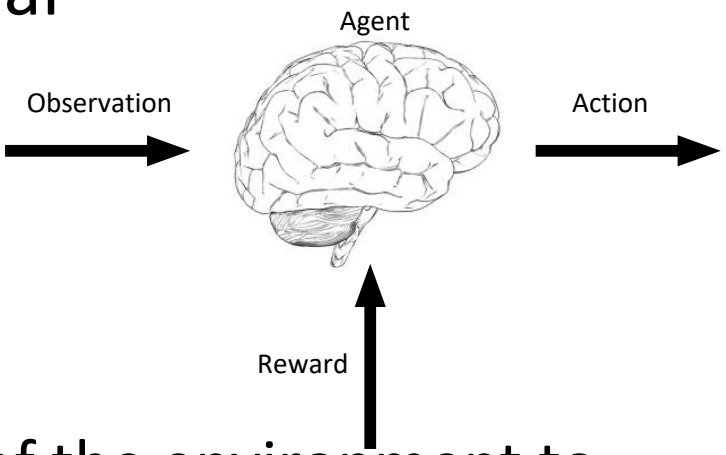
Reinforcement Learning

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



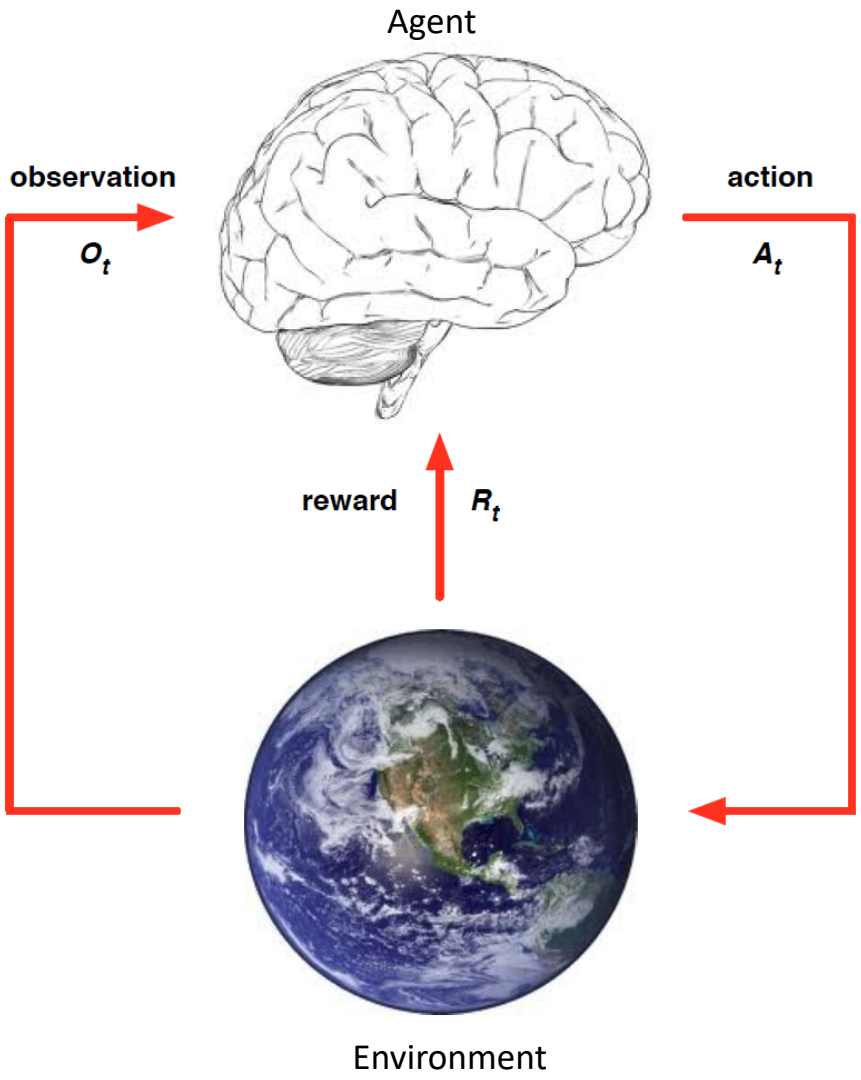
Reinforcement Learning Definition

- 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 cumulated reward

Reinforcement Learning



- 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

Elements of RL Systems

- History is the sequence of observations, action, rewards

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

  - i.e. all observable variables up to time *t*
  - E.g., the sensorimotor stream of a robot or embodied agent

- 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

$$\pi(a|s) = P(A_t = a|S_t = s)$$

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

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

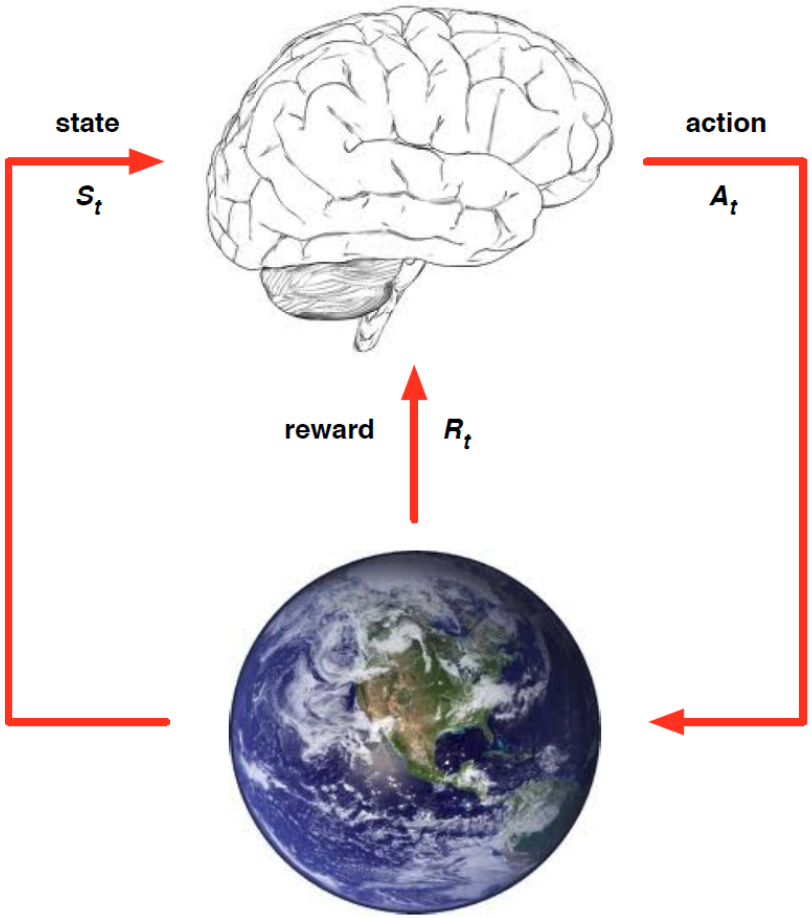
Elements of RL Systems

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

$$\mathcal{P}^a_{ss'} = \mathbb{P}[S_{t+1} = s' | S_t = s, A_t = a]$$

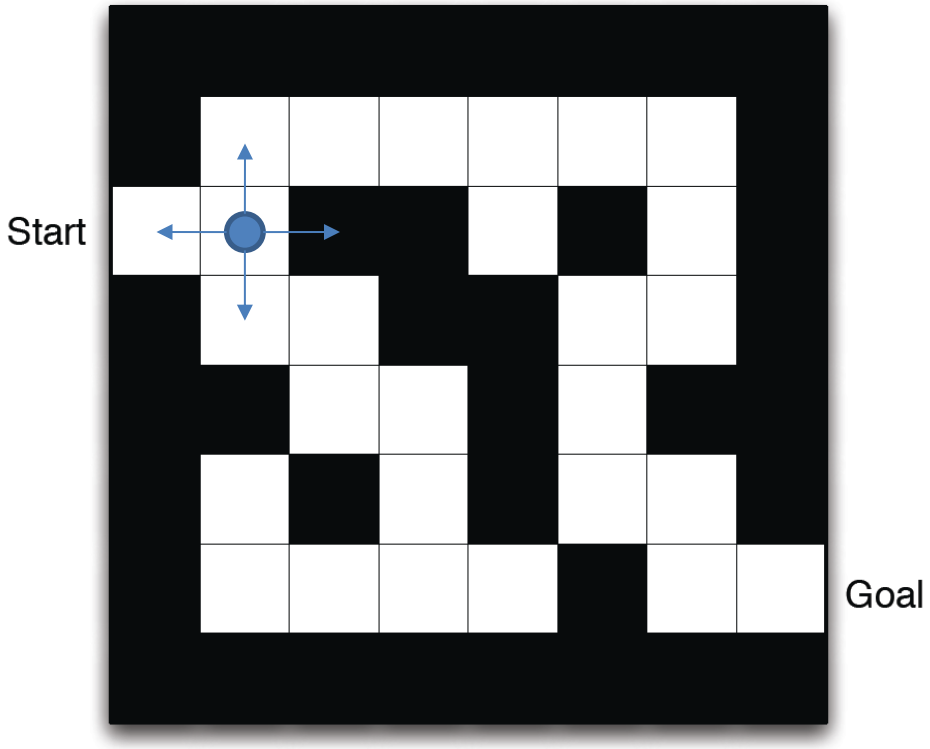
  - Predicts the next (immediate) reward

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



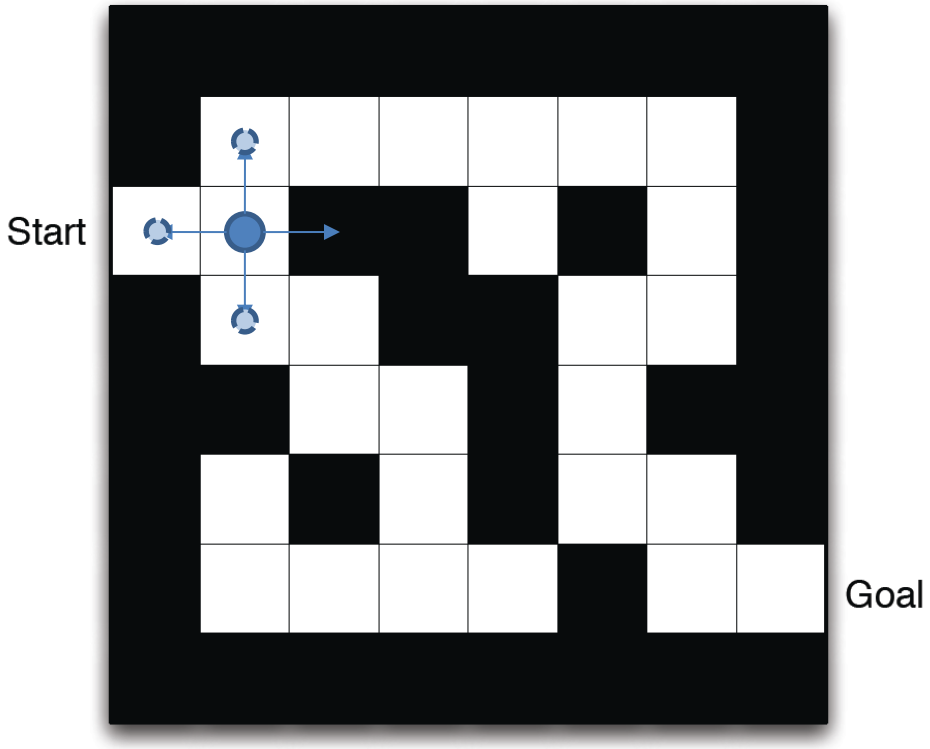
The agent may know the model of the environment or may not!

Maze Example



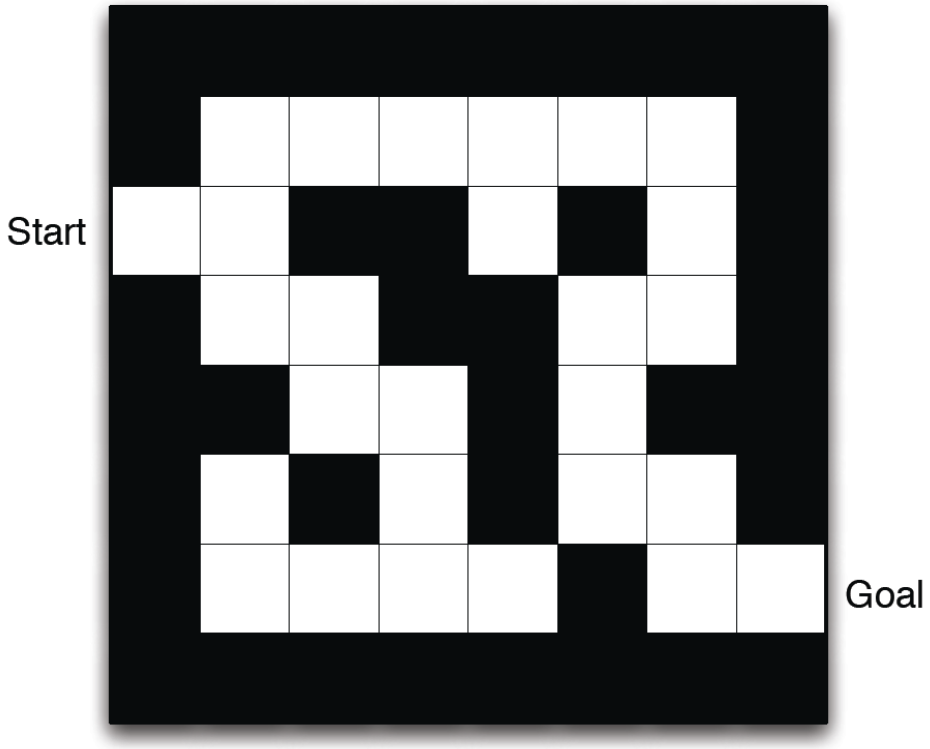
- State: agent's location
- Action: N,E,S,W

Maze Example



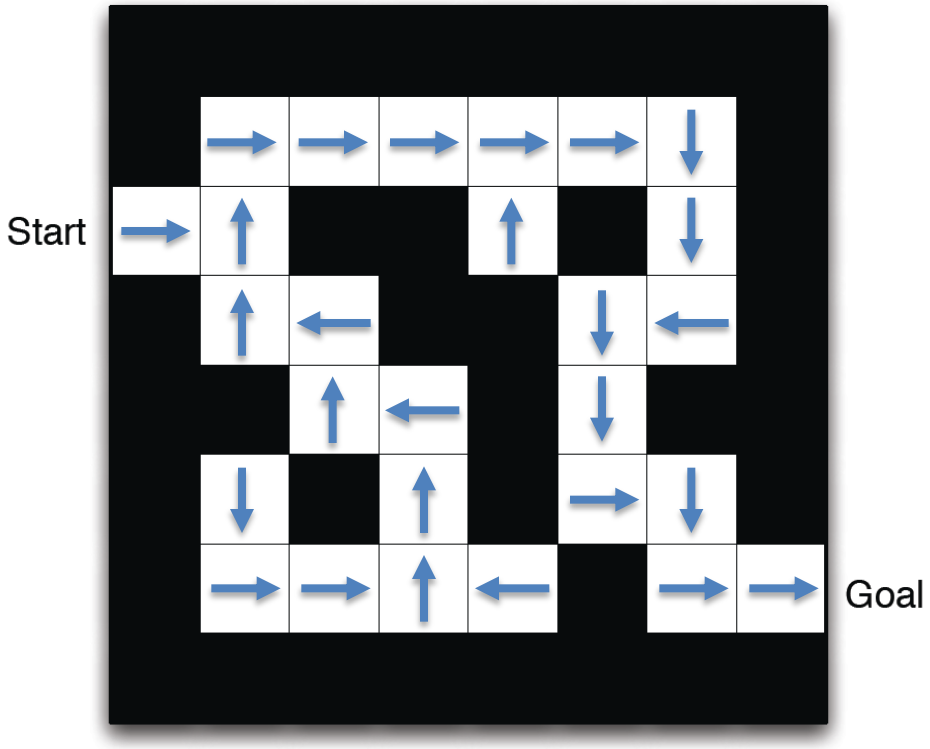
- State: agent's location
- Action: N,E,S,W
- State transition: move to the next grid according to the action
  - No move if the action is to the wall

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

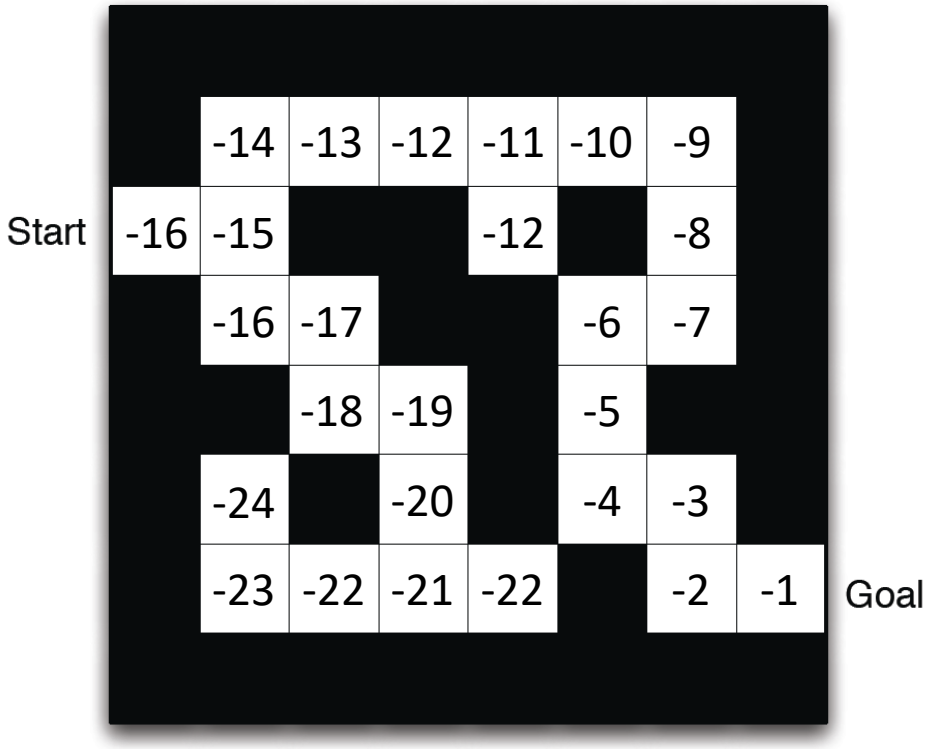
Maze Example



- 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

Content

- Reinforcement Learning
  - **The model-based methods**
    - **Markov Decision Process**
    - **Planning by Dynamic Programming**
    - **Convergence Analysis**
  - The model-free methods
    - Model-free Prediction
      - Monte-Carlo and Temporal Difference
    - Model-free Control
      - On-policy SARSA and off-policy Q-learning
      - Convergence Analysis
- Deep Reinforcement Learning Examples
  - Atari games
  - Alpha Go

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

$$\mathbb{P}[S_{t+1}|S_t] = \mathbb{P}[S_{t+1}|S_1, \ldots, S_t]$$

- Properties
  - The state captures all relevant information from the history
  - 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
- $R : S \times A \mapsto \mathbb{R}$ 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
- 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

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

MDP Goal and 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
  - 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 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 cumulated reward given the start state and taking actions according to π

Bellman Equation for Value Function

- Define the value function for π

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

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

Bellman Equation

a necessary condition for optimality

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.

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 π

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

Value Iteration & Policy Iteration

- Note that the value function and policy are correlated

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

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

- It is feasible to perform iterative update towards the optimal value function and optimal policy
  - 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 {

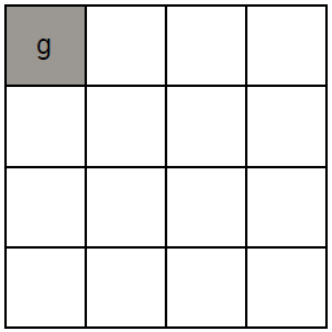
   For each state, update

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

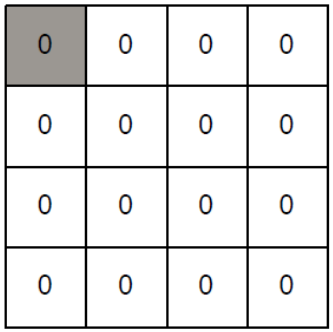
}

- Note that there is no explicit policy in above calculation

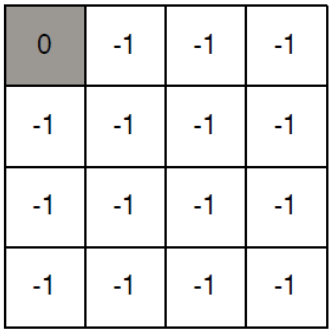
Value Iteration Example: Shortest Path



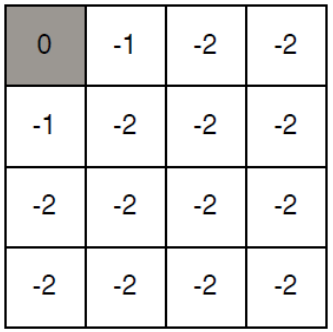
Problem



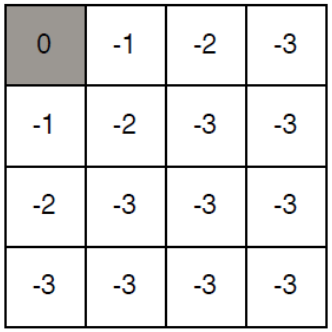
V_1



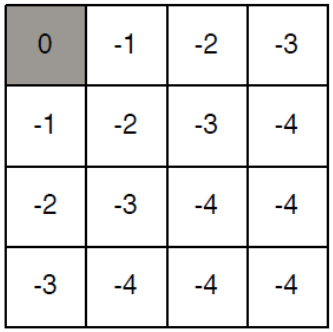
V_2



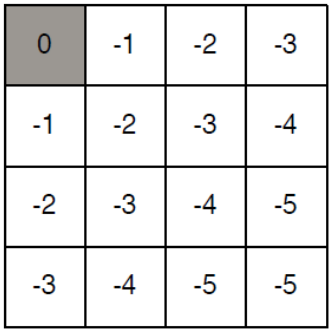
V_3



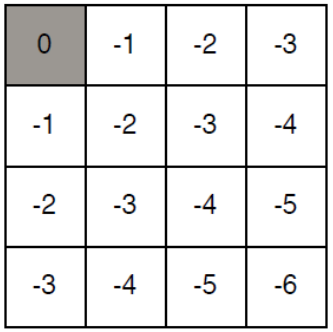
V_4



V_5



V_6



V_7

Policy Iteration

- For an MDP with finite state and action spaces

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

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

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

- Policy iteration is performed as

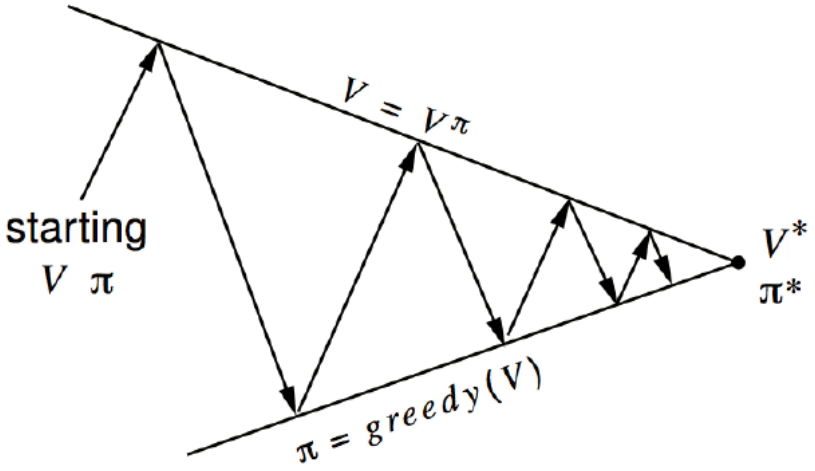
  1. Initialize π randomly
  2. Repeat until convergence {
     a) Let $V := V^{\pi}$
     b) For each state, update

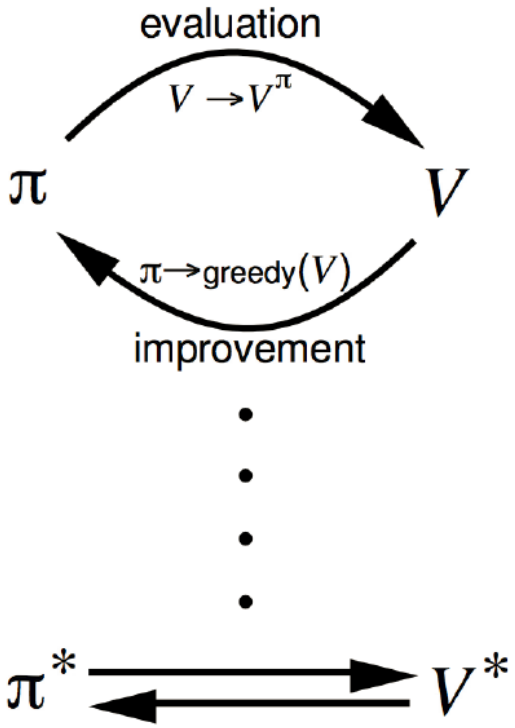
$$\pi(s) = \arg\max_{a \in A} \sum_{s' \in S} P_{sa}(s') V(s')$$

     }

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

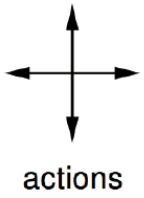
Policy Iteration

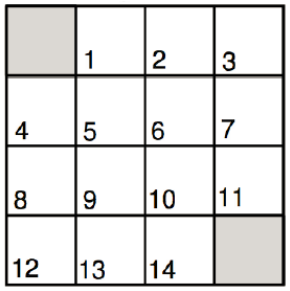




- Policy evaluation
  - Estimate V^{π}
  - Iterative policy evaluation
- Policy improvement
  - Generate $\pi' \geq \pi$
  - Greedy policy improvement

Evaluating a Random Policy in the Small Gridworld



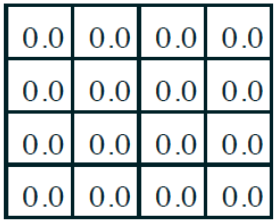
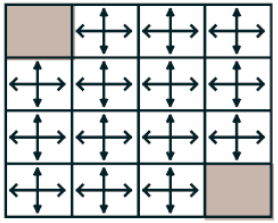
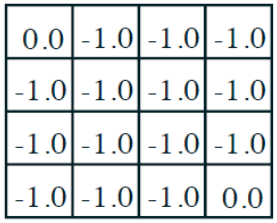
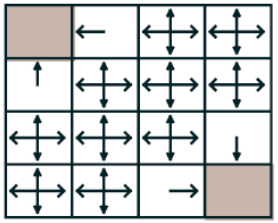
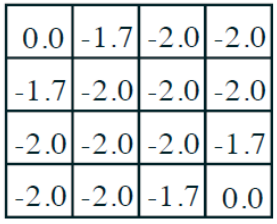
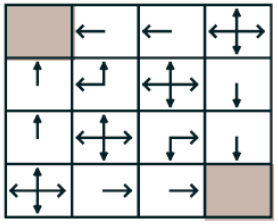


r = -1
on all transitions

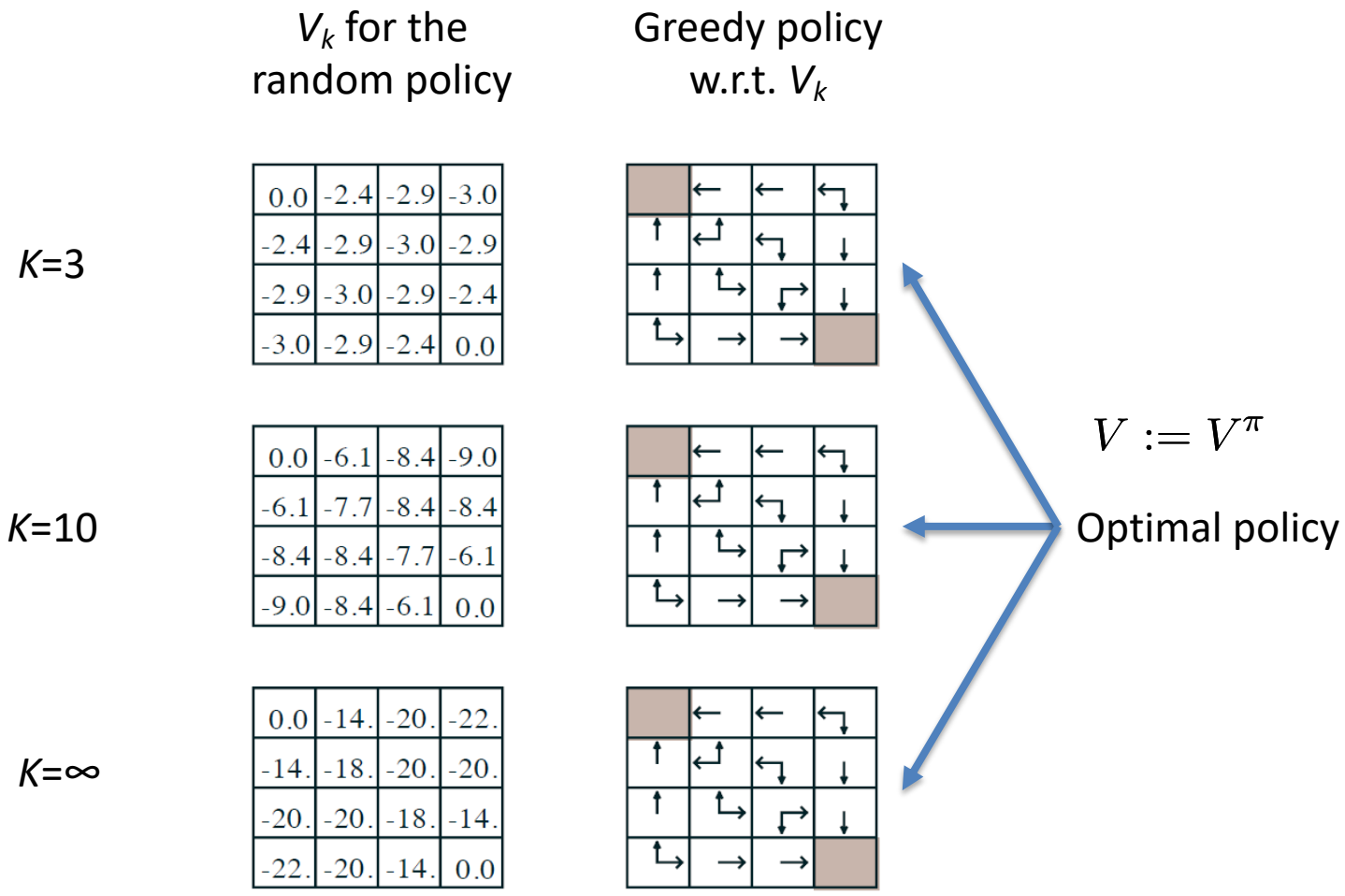
- Undiscounted episodic MDP (γ=1)
- Nonterminal states 1,…,14
- Two terminal states (shaded squares)
- Actions leading out of the grid leave state unchanged
- Reward is -1 until the terminal state is reached
- Agent follows a uniform random policy

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

Evaluating a Random Policy in the Small Gridworld

	V_k for the random policy	Greedy policy w.r.t. V_k	
K=0			Random policy
K=1			
K=2			

Evaluating a Random Policy in the Small Gridworld



Value Iteration vs. Policy Iteration

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

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

Value iteration

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

   For each state, update

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

   }

Policy iteration

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

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

   }

Remarks:

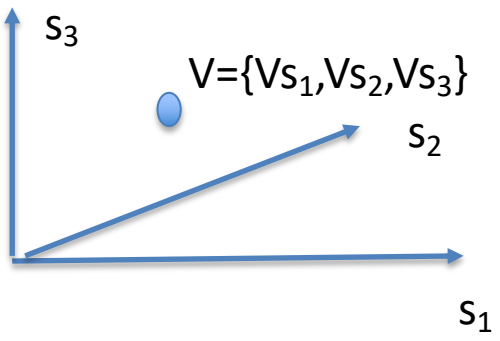
1. Value iteration is a greedy update strategy
2. In policy iteration, the value function update by Bellman equation is costly
3. For small-space MDPs, policy iteration is often very fast and converges quickly
4. For large-space MDPs, value iteration is more practical (efficient)

Convergence Analysis

- How do we know that value iteration converges to v^*?
- Or that iterative policy evaluation converges to v_π? And therefore that policy iteration converges to $v*$?
- Is the solution unique?
- How fast do these algorithms converge?
- These questions are resolved by **contraction mapping theorem.**

Value Function Space

- Consider the vector space V over value functions
- There are |S| dimensions
  - Each point in this space fully specifies a value function v(s)
  - What does **Bellman backup** do to the points in this space?
    - We will show that it brings value functions closer
    - And therefore the backups must converge on a unique solution

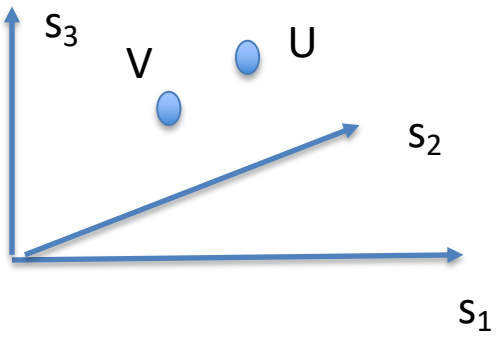


Value Function ∞-Norm

- We will measure distance between state-value functions u and v by

  the ∞-norm (infinity norm or max norm)
  i.e. the largest difference between state values,

$$||u - v||_\infty = \max_{s \in \mathcal{S}} |u(s) - v(s)|$$



Banach's Contraction Mapping Theorem

- For any metric space V that is complete (i.e. closed) under an operator T(v) , where T is a γ-contraction,
  - T converges to a unique fixed point: v = T(v)
  - At a linear convergence rate of γ

γ-contraction means: $d(T(v) - T(u)) \leq \gamma d(v, u)$ where d is the distance in the metric space and γ is the parameter (this makes sure that the operator brings value functions closer)

Bellman Operator on Q-function

- Let us define Bellman expectation operator on Q-function

$$
\begin{aligned}
Q^{\pi}(s,a) &= \mathbb{E}\left[\sum_{t=0}^{\infty}\gamma^{t}R_{t}|S_{0}=x,A_{0}=a\right] \\
&= \mathbb{E}\left[R\left(S_{0},A_{0}\right)+\gamma\sum_{t=0}^{\infty}\gamma^{t}R_{t+1}|S_{0}=s,A_{0}=a\right] \\
&= \mathbb{E}\left[R\left(X_{0},A_{0}\right)+\gamma Q^{\pi}\left(S_{1},\pi\left(S_{1}\right)\right)|S_{0}=s,A_{0}=a\right] \\
&= \underbrace{r(x,a)+\gamma\sum_{\mathcal{S}'}\mathcal{P}\left(s'|s,a\right)Q^{\pi}\left(s',\pi\left(s'\right)\right)}_{\triangleq(T^{\pi}Q^{\pi})(s,a)}
\end{aligned}
$$

- Bellman optimality operator on Q-function

$$\left(T^{*}Q\right)(s,a)\triangleq r(s,a)+\gamma\sum_{\mathcal{S}'}\mathcal{P}\left(s'|s,a\right)\max_{a'\in\mathcal{A}}Q\left(s',a'\right)$$

Bellman Operator on Q-function

- Value-based approaches try to find a fixed point $\hat{Q}$ such that $\hat{Q} \approx \mathsf{T}* \hat{Q}$
- And the greedy policy from Qˆ is thus close to the optimal policy π^*

$$Q^{\pi(s;\hat{Q})} \approx Q^{\pi^*} = Q^*$$

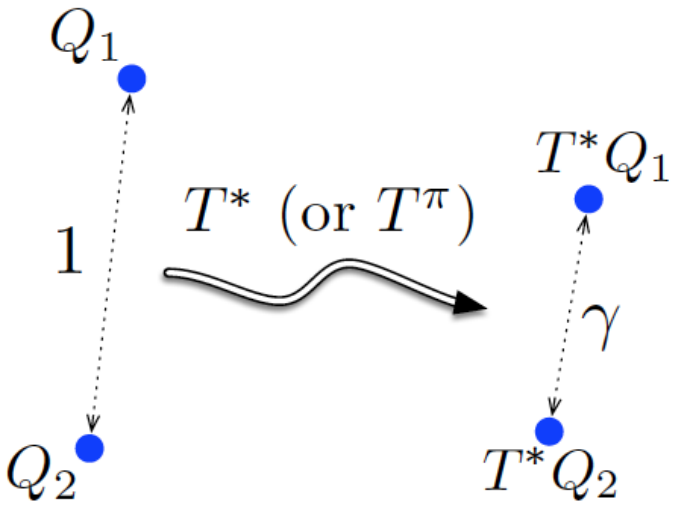
where

$$\pi(s;\hat{Q}) \leftarrow \underset{a \in \mathcal{A}}{\text{argmax}} \hat{Q}(s,a)$$

Contraction Property of Bellman Operator on Q-function

$$\| T^* Q_1 - T^* Q_2 \|_\infty \leq \gamma \| Q_1 - Q_2 \|_\infty$$

$$Q_{k+1} \leftarrow T^* Q_k$$

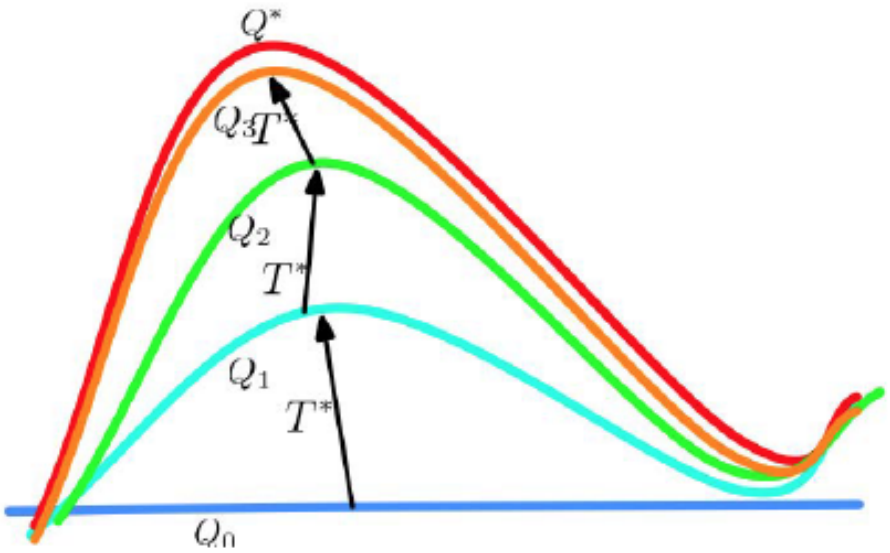


Contraction Property of Bellman Operator on Q-function

$$\|T^*Q_1 - T^*Q_2\|_\infty \leq \gamma \|Q_1 - Q_2\|_\infty$$

$$Q_{k+1} \leftarrow T^*Q_k$$

Let's say Q_2 is Q*



Proof of the Contraction Property

- We can see that

$$
\begin{aligned}
&\left|\left(T^{*} Q_{1}\right)(s, a)-\left(T^{*} Q_{2}\right)(s, a)\right| = \\
&\qquad \gamma\left|\sum_{\mathcal{S}^{\prime}} \mathcal{P}\left(s^{\prime} | s, a\right)\left[\max_{a^{\prime} \in \mathcal{A}} Q_{1}\left(s^{\prime}, a^{\prime}\right)-\max_{a^{\prime} \in \mathcal{A}} Q_{2}\left(s^{\prime}, a^{\prime}\right)\right]\right| \\
&\qquad \leq \gamma \sum_{\mathcal{S}^{\prime}} \mathcal{P}\left(s^{\prime} | s, a\right) \max_{a^{\prime} \in \mathcal{A}}\left|Q_{1}\left(s^{\prime}, a^{\prime}\right)-Q_{2}\left(s^{\prime}, a^{\prime}\right)\right| \\
&\qquad \leq \gamma \max_{\left(s^{\prime}, a^{\prime}\right) \in \mathcal{S}^{\prime} \times \mathcal{A}}\left|Q_{1}\left(s^{\prime}, a^{\prime}\right)-Q_{2}\left(s^{\prime}, a^{\prime}\right)\right| \underbrace{\sum_{\mathcal{S}^{\prime}} \mathcal{P}\left(s^{\prime} | s, a\right)}_{=1}
\end{aligned}
$$

- therefore we get

$$
\sup_{(s, a) \in \mathcal{S} \times \mathcal{A}}\left|\left(T^{*} Q_{1}\right)(s, a)-\left(T^{*} Q_{2}\right)(s, a)\right| \leq \gamma \sup_{(s, a) \in \mathcal{S} \times \mathcal{A}}\left|Q_{1}(s, a)-Q_{2}(s, a)\right|
$$

$$
\left\|T^{*} Q_{1}-T^{*} Q_{2}\right\|_{\infty} \leq \gamma\left\|Q_{1}-Q_{2}\right\|_{\infty}
$$

Proof of the Contraction Property

- Banach's fixed-point theorem can be used to show the existence and uniqueness of the optimal value function.
- Also, since we have $Q^* = T^* Q^*$

- Thus, for value iteration where $Q_{k+1} \leftarrow T^* Q_k$, we have

$$\| T^* Q_0 - Q^* \|_\infty = \| T^* Q_0 - T^* Q^* \|_\infty \leq \gamma \| Q_0 - Q^* \|_\infty$$

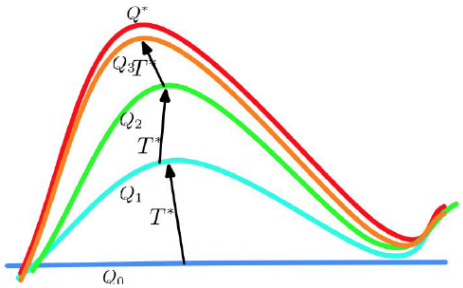
- Finally as k-> ∞, the RHS converges to zero, showing that

$$\| Q_k - Q^* \|_\infty = \left\| (T^*)^{(k)} Q_0 - T^* Q^* \right\|_\infty \leq \gamma^k \| Q_0 - Q^* \|_\infty$$

- This means $Q_k \rightarrow Q^*$

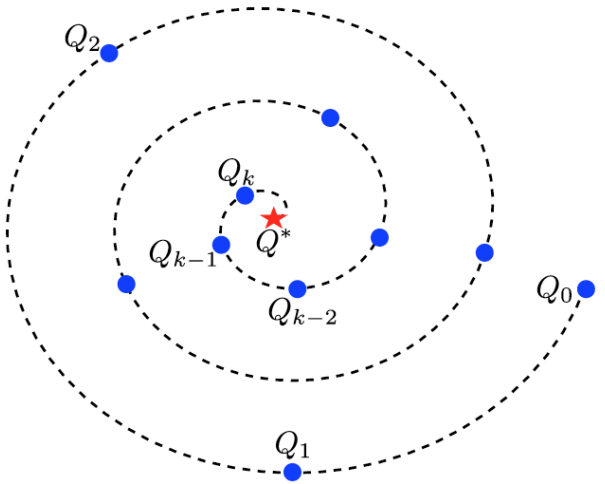
$$\| Q_k - Q^* \|_\infty \rightarrow 0.$$

Proof of the Contraction Property



$$Q_{k+1} \leftarrow T^* Q_k$$

Contraction Property



$$\|Q_k - Q^*\|_\infty \leq \gamma^k \|Q_0 - Q^*\|_\infty$$

Bellman Expectation Backup is a Contraction

- Define the Bellman expectation backup operator T^π
- This operator is a γ-contraction, i.e. it makes value functions closer by at least γ,

$$\begin{aligned}
\|T^\pi(u) - T^\pi(v)\|_\infty &= \|(\mathcal{R}^\pi + \gamma\mathcal{P}^\pi u) - (\mathcal{R}^\pi + \gamma\mathcal{P}^\pi v)\|_\infty \\
&= \|\gamma\mathcal{P}^\pi(u - v)\|_\infty \\
&\leq \|\gamma\mathcal{P}^\pi\| \, u - v \, \|_\infty\|_\infty \\
&\leq \gamma\|u - v\|_\infty
\end{aligned}$$

Bellman Optimality Backup is a Contraction

- Define the Bellman optimality backup operator T^*,

$$T^*(v) = \max_{a \in \mathcal{A}} \mathcal{R}^a + \gamma \mathcal{P}^a v$$

- This operator is a γ-contraction, i.e. it makes value functions closer by at least γ (similar to previous proof)

$$||T^*(u) - T^*(v)||_\infty \leq \gamma ||u - v||_\infty$$

Convergence of Value Iteration

- Thus, by contraction mapping theorem
  - The Bellman optimality operator $T*$ has a unique fixed point
  - $v*$ is a fixed point of $T*$ (by Bellman optimality equation)
  - Value iteration converges on $v*$
- However
  - Exact representation of the value function for large state space is infeasible, i.e. Q(s, a) for all (s, a) $\in S \times A$.
  - The exact summation in the Bellman operator requires to know the environmental model, i.e.

$T*(v) = \max_{a \in A} R^a + \gamma \mathbf{P}^a v$.

Content

- Reinforcement Learning
  - The model-based methods
    - Markov Decision Process
    - Planning by Dynamic Programming
    - Convergence Analysis
  - **The model-free methods**
    - **Model-free Prediction**
      - **Monte-Carlo and Temporal Difference**
    - **Model-free Control**
      - **On-policy SARSA and off-policy Q-learning**
      - **Convergence Analysis**
- Deep Reinforcement Learning Examples
  - Atari games
  - Alpha Go

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

$\vdots$

- Model-free RL is to directly learning 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

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

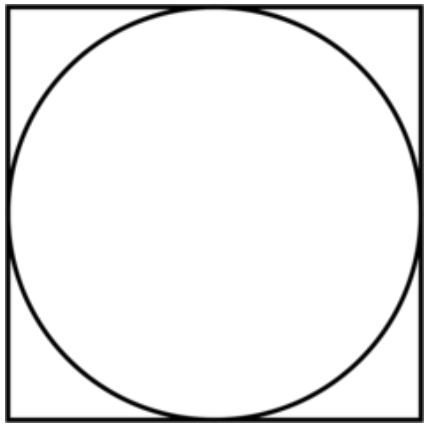
- Now in model-free RL
  - We cannot directly know P_{sa} and R
  - But we have a list of experiences to estimate the values

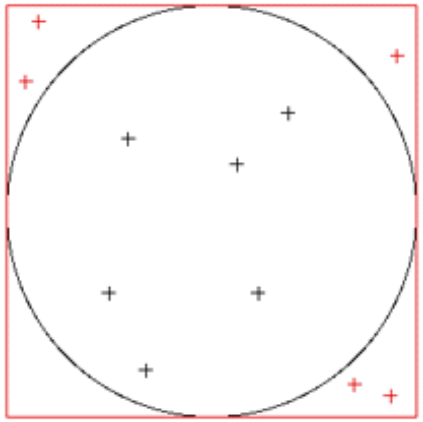
Episode 1: $s_0^{(1)} \xrightarrow[R(s_0)^{(1)}]{a_0^{(1)}} s_1^{(1)} \xrightarrow[R(s_1)^{(1)}]{a_1^{(1)}} s_2^{(1)} \xrightarrow[R(s_2)^{(1)}]{a_2^{(1)}} s_3^{(1)} \cdots s_T^{(1)}$

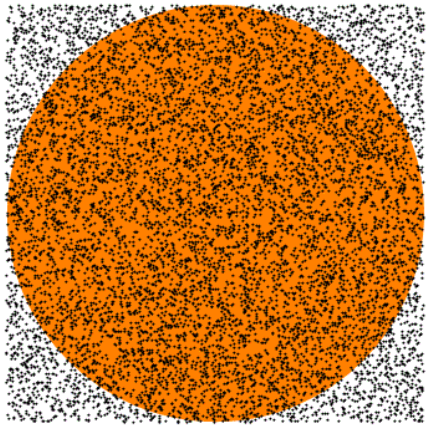
Episode 2: $s_0^{(2)} \xrightarrow[R(s_0)^{(2)}]{a_0^{(2)}} s_1^{(2)} \xrightarrow[R(s_1)^{(2)}]{a_1^{(2)}} s_2^{(2)} \xrightarrow[R(s_2)^{(2)}]{a_2^{(2)}} s_3^{(2)} \cdots s_T^{(2)}$

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



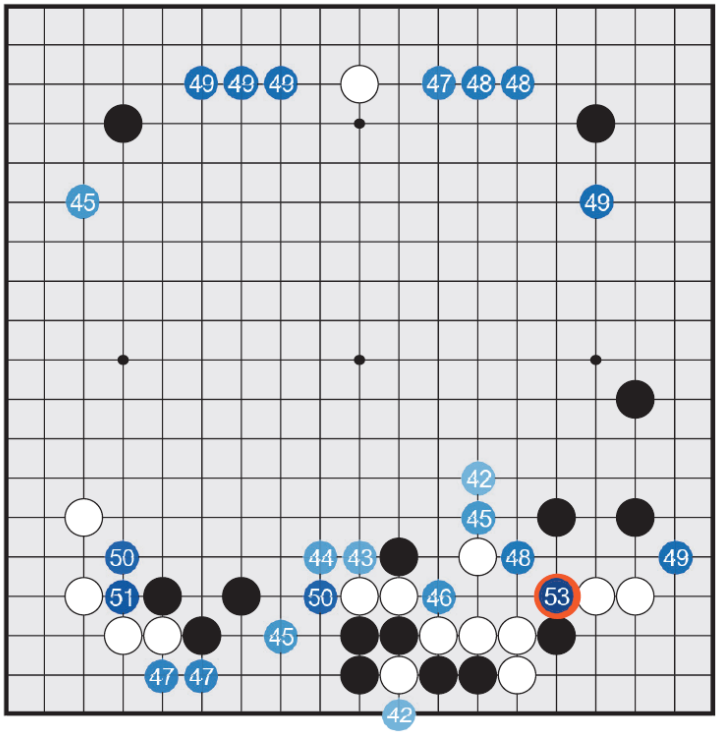


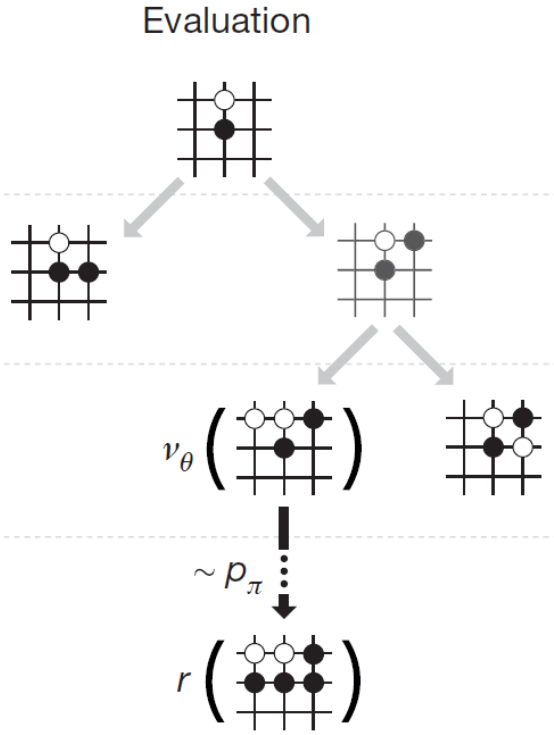


$$\text{Circle Surface} = \text{Square Surface} \times \frac{\#\text{points in circle}}{\#\text{points in total}}$$

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

Incremental Monte-Carlo Updates

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

$$N(S_t) \leftarrow N(S_t) + 1$$
$$V(S_t) \leftarrow V(S_t) + \frac{1}{N(S_t)}(G_t - V(S_t))$$

- 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

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

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

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
- TD updates a guess towards a guess

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots = R_{t+1} + \gamma V(S_{t+1})$$

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

Observation

Guess of future

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

Driving Home Example

Each day as you drive home from work, you try to predict how long it will take to get home

State	Elapsed Time (Minutes)	Predicted Time to Go	Predicted Total Time
Leaving office	0	30	30
Reach car, raining	5	35	40
Exit highway	20	15	35
Behind truck	30	10	40
Home street	40	3	43
Arrive home	43	0	43

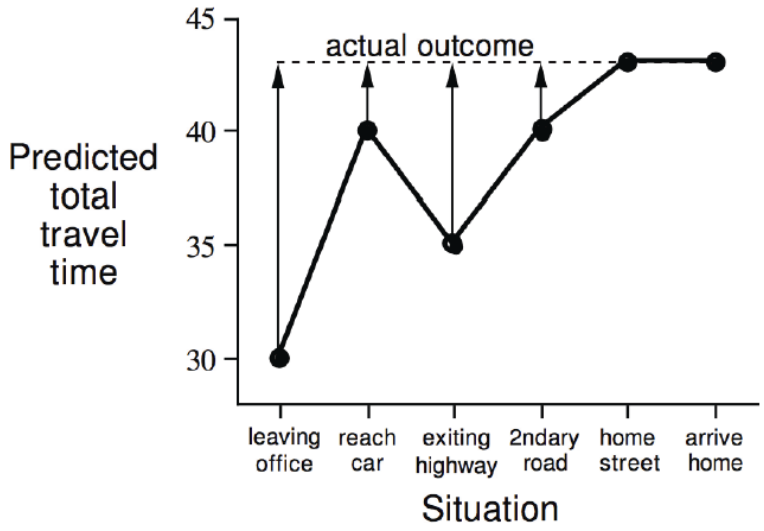
Sutton, Richard S., and Andrew G. Barto. *Reinforcement learning: An introduction*. Vol. 1. No. 1. Cambridge: MIT press, 1998.

Driving Home Example: MC vs. TD

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

$V(\text{exiting highway}) \leftarrow 15 + ((43\text{-}20)\text{-}15)$

Changes recommended by Monte Carlo methods (α=1)



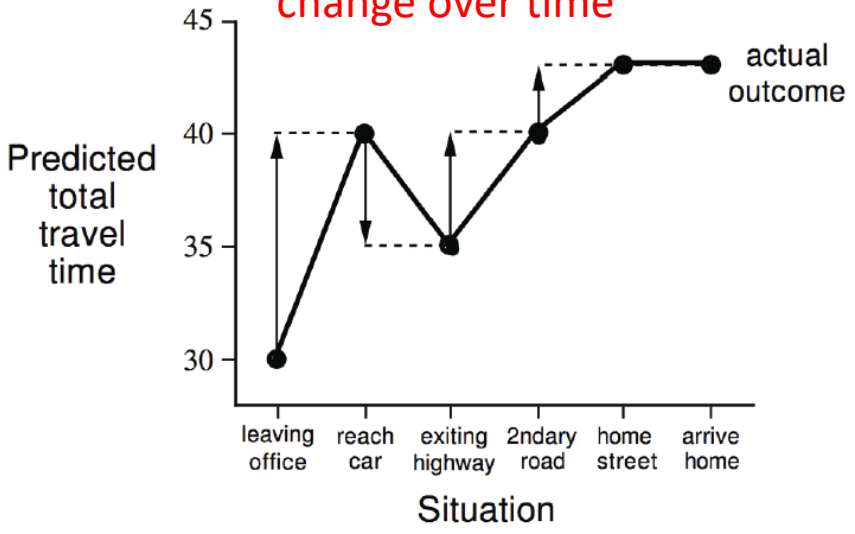
G_t is the award when reached home, so the change can only be made offline after have reached home

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

$V(\text{exiting highway}) \leftarrow 15 + (30\text{+}10\text{-}35)$

Changes recommended by TD methods (α=1)

Each error is proportional to the change over time



By contrast, in TD, each estimate would be updated toward the estimate that immediately follows it.

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

Bias/Variance Trade-Off

- Return $G_t = R_{t+1} + \gamma R_{t+2} + \ldots \gamma^{T-1} R_T$ is unbiased estimate of $V^\pi(S_t)$
- True TD target $R_{t+1} + \gamma V(S_{t+1})$ is unbiased estimate of $V^\pi(S_t)$
- TD target $R_{t+1} + \gamma \underbrace{V(S_{t+1})}_{\text{current estimate}}$ is biased estimate of $V^\pi(S_t)$
- TD target is of much lower variance than the return
  - Return depends on many random actions, transitions and rewards
  - TD target depends on one random action, transition and reward

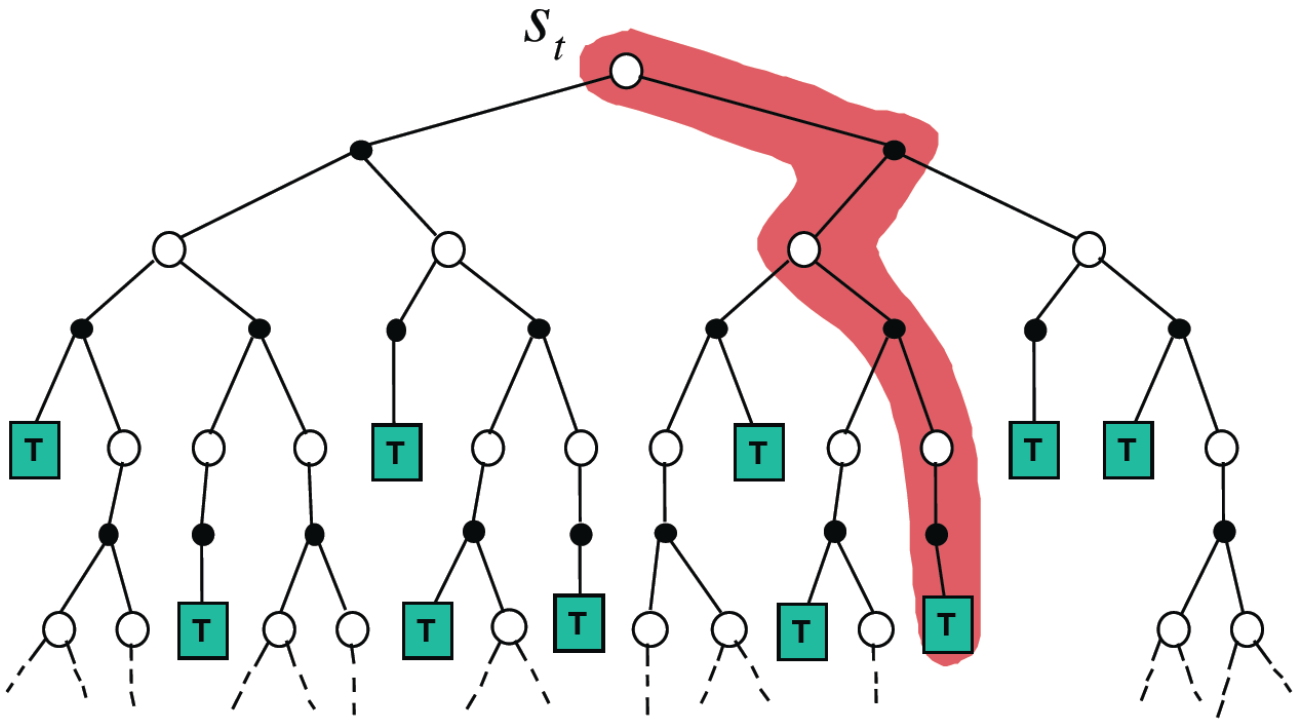
Advantages and Disadvantages of MC vs. TD (2)

- 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

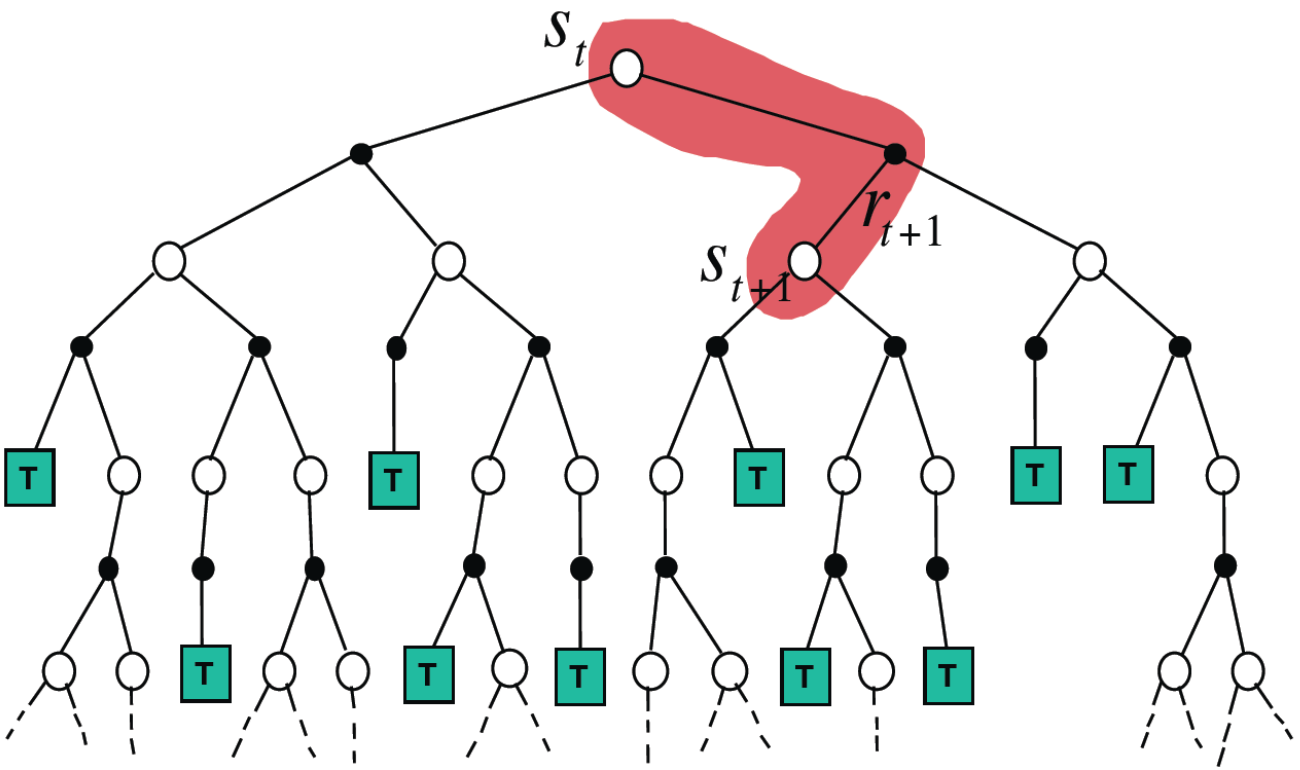
$$V(S_t) \leftarrow V(S_t) + \alpha(G_t - V(S_t))$$



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

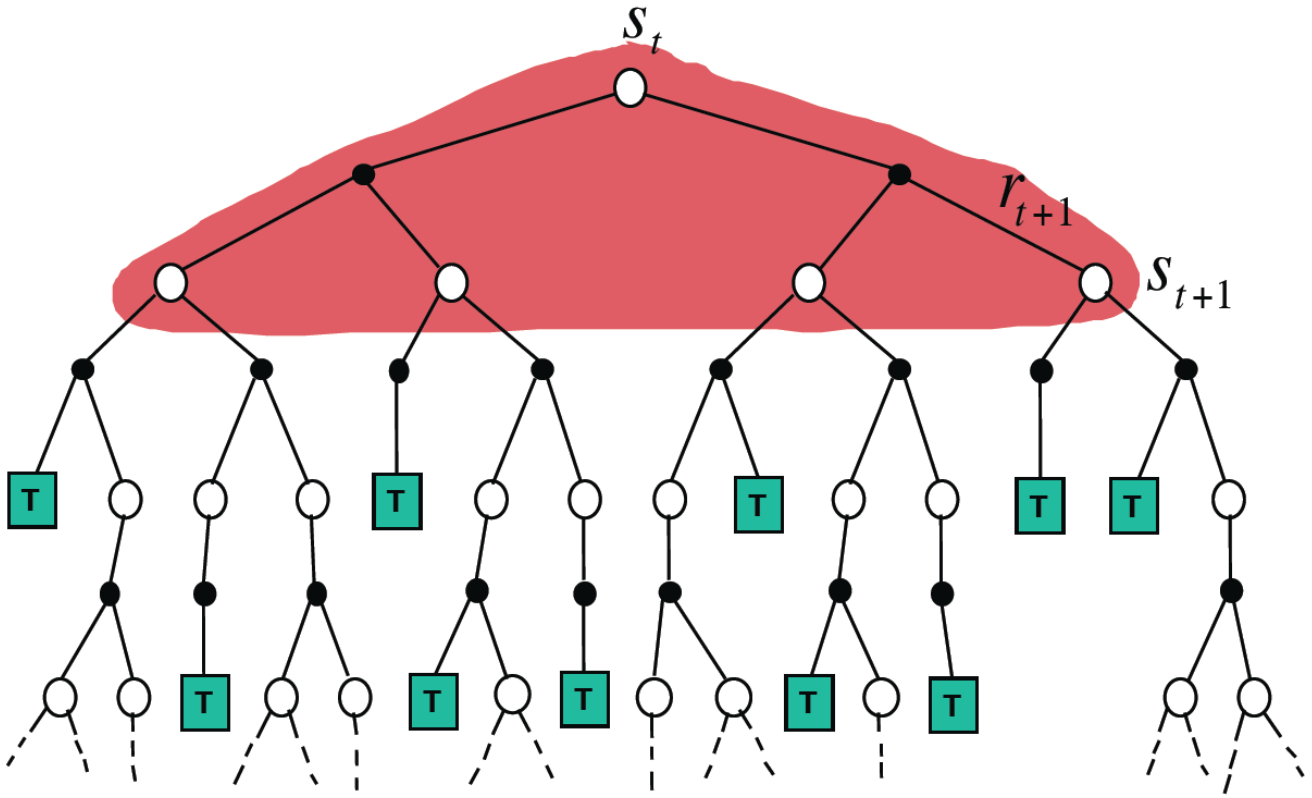
Temporal-Difference Backup

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



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**
      - **On-policy SARSA and off-policy Q-learning**
      - **Convergence Analysis**
- Deep Reinforcement Learning Examples
  - Atari games
  - Alpha Go

Uses of Model-Free Control

- Some example problems that can be modeled as MDPs
  - Elevator
  - Parallel parking
  - Ship steering
  - Bioreactor
  - Helicopter
  - Aeroplane logistics
  - Robocup soccer
  - Atari & StarCraft
  - Portfolio management
  - Protein folding
  - Robot walking
  - Game of Go
- For most of real-world problems, either:
  - MDP model is unknown, but experience can be sampled
  - MDP model is known, but is too big to use, except by samples
- Model-free control can solve these problems

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

State Value and Action Value

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

- 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

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

- 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

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

Bellman Expectation Equation

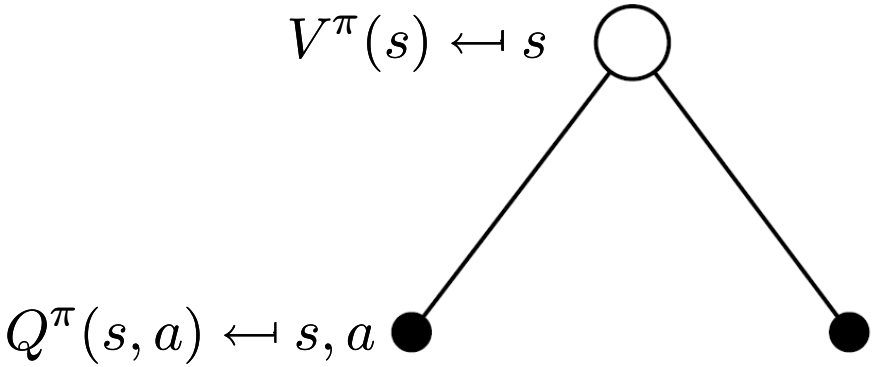
- The state-value function $V^{\pi}(s)$ can be decomposed into immediate reward plus discounted value of successor state

$$V^{\pi}(s) = \mathbb{E}_{\pi}[R_{t+1} + \gamma V^{\pi}(S_{t+1}) | S_t = s]$$

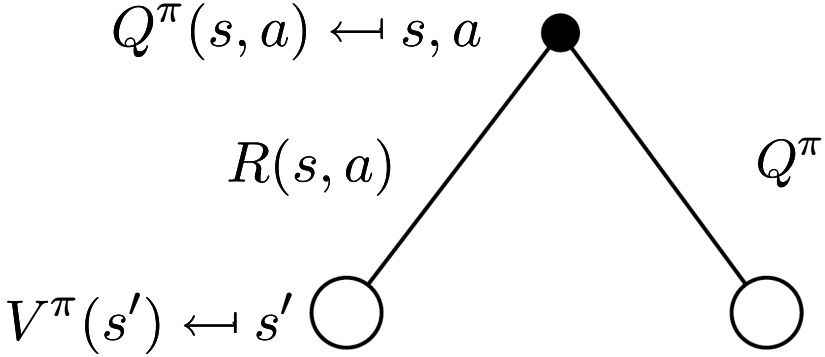
- The action-value function $Q^{\pi}(s,a)$ can similarly be decomposed

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}[R_{t+1} + \gamma Q^{\pi}(S_{t+1}, A_{t+1}) | S_t = s, A_t = a]$$

State Value and Action Value



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



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

Model-Free Policy Iteration

- Given state-value function $V(s)$ and action-value function $Q(s,a)$, model-free policy iteration shall use action-value function

- Greedy policy improvement over $V(s)$ requires model of MDP

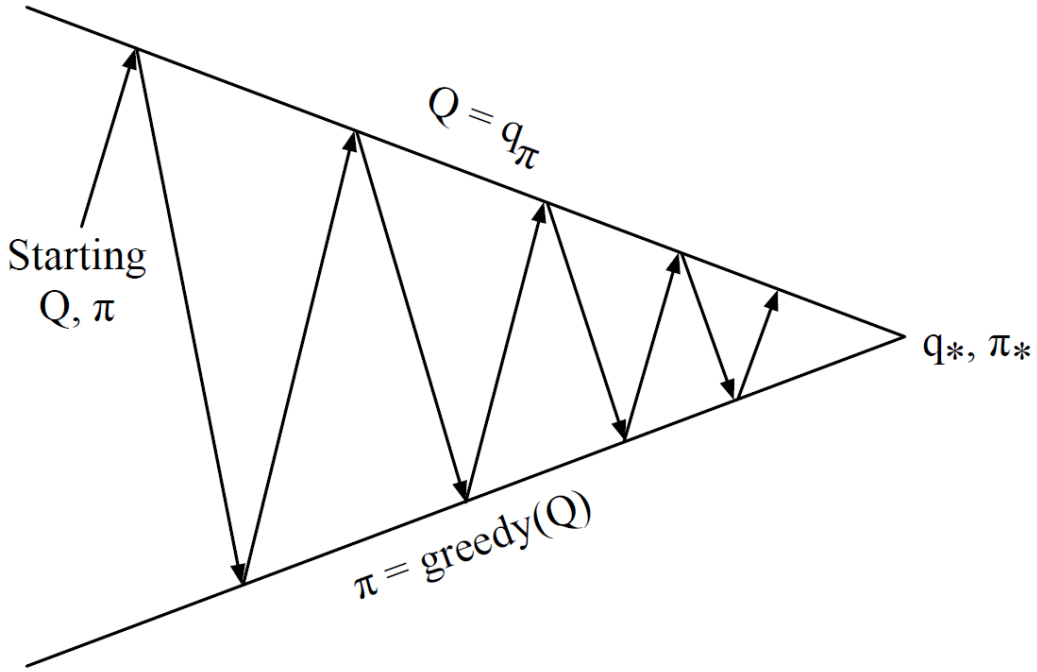
$$\pi^{\text{new}}(s) = \arg\max_{a \in A} \left\{ R(s,a) + \gamma \sum_{s' \in S} P_{sa}(s') V^{\pi}(s') \right\}$$

We don't know the transition probability

- Greedy policy improvement over $Q(s,a)$ is model-free

$$\pi^{\text{new}}(s) = \arg\max_{a \in A} Q(s,a)$$

Generalized Policy Iteration with Action-Value Function



- Policy evaluation: Monte-Carlo policy evaluation, $Q = Q^\pi$
- Policy improvement: Greedy policy improvement?

Example of Greedy Action Selection

- Greedy policy improvement over *Q*(*s*,*a*) is model-free

$$\pi^{\text{new}}(s) = \arg\max_{a \in A} Q(s,a)$$

- Given the right example
  - What if the first action is to choose the left door and observe reward=0?
  - The policy would be suboptimal if there is no exploration

Left:
20% Reward = 0
80% Reward = 5

Right:
50% Reward = 1
50% Reward = 3

"Behind one door is tenure – behind the other is flipping burgers at McDonald's."

ε-Greedy Policy Exploration

- Simplest idea for ensuring continual exploration
- All *m* actions are tried with non-zero probability
  - With probability 1-ε, choose the greedy action
  - With probability ε, choose an action at random

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

ε-Greedy Policy Improvement

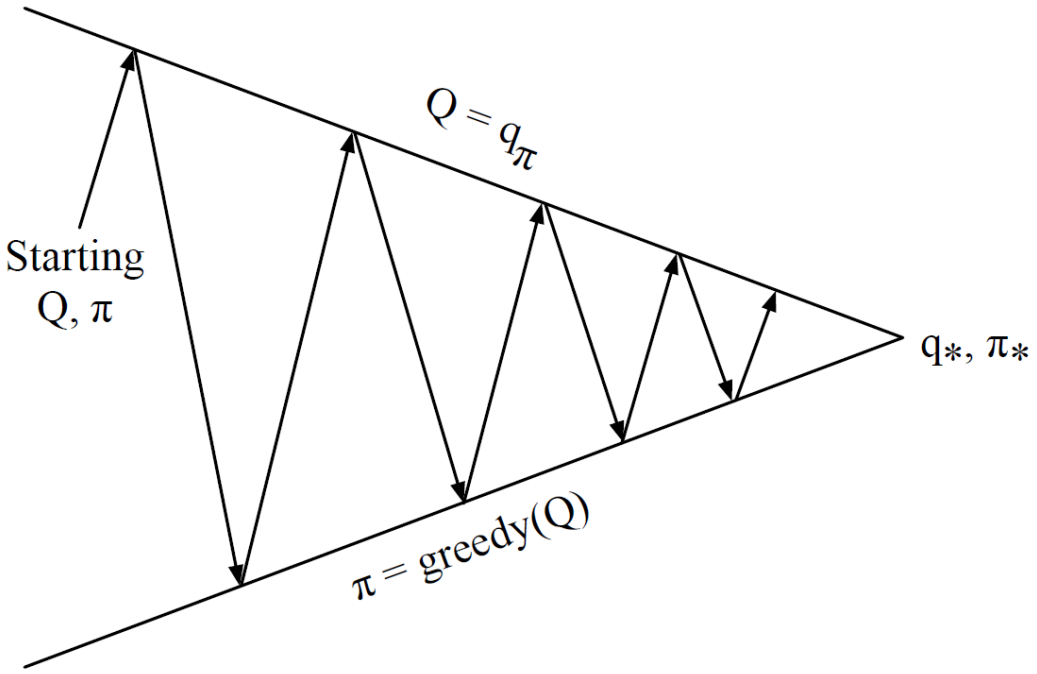
- Theorem
  - For any policy π, the ε-greedy policy π' w.r.t. Q^π is an improvement, i.e. $V^{\pi'}(s) \geq V^\pi(s)$

$$
\begin{aligned}
Q^\pi(s, \pi'(s)) &= \sum_{a \in A} \pi'(a|s) Q^\pi(s, a) \\
&= \frac{\epsilon}{m} \sum_{a \in A} Q^\pi(s, a) + (1 - \epsilon) \max_{a \in A} Q^\pi(s, a) \\
&\geq \frac{\epsilon}{m} \sum_{a \in A} Q^\pi(s, a) + (1 - \epsilon) \sum_{a \in A} \frac{\pi(a|s) - \epsilon/m}{1 - \epsilon} Q^\pi(s, a) \\
&= \sum_{a \in A} \pi(a|s) Q^\pi(s, a) = V^\pi(s)
\end{aligned}
$$

m actions

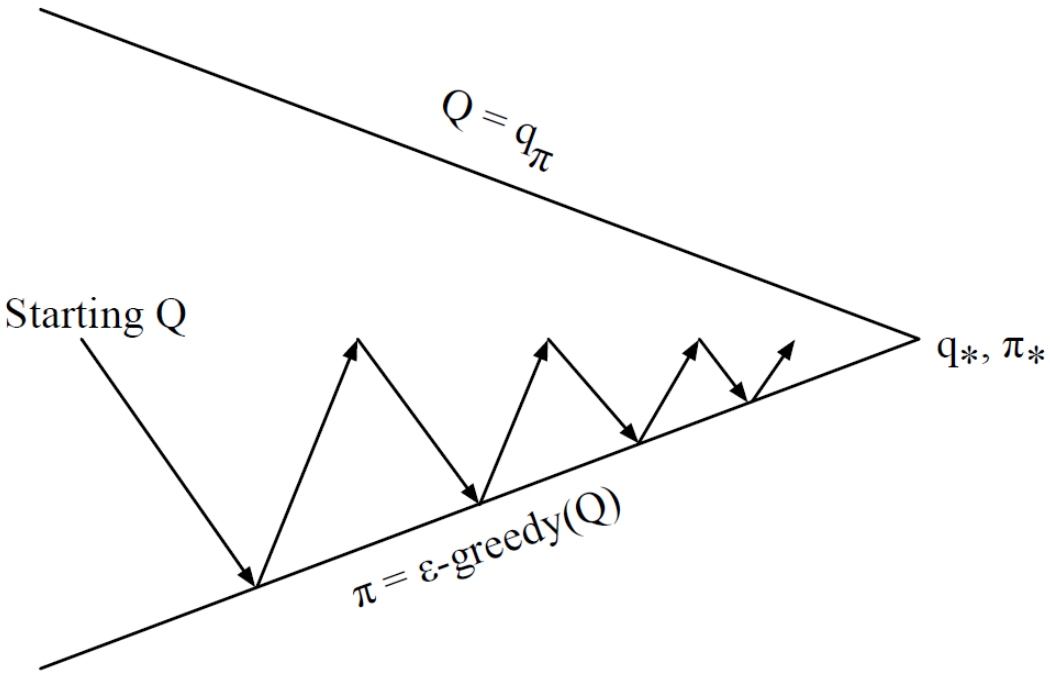
Generalized Policy Iteration with Action-Value Function

- 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 policy evaluation, $Q = Q^\pi$
- Policy improvement: ε-greedy policy improvement

Monte-Carlo Control



Every episode:

- Policy evaluation: Monte-Carlo policy evaluation, $Q \approx Q^\pi$
- Policy improvement: ε-greedy policy improvement

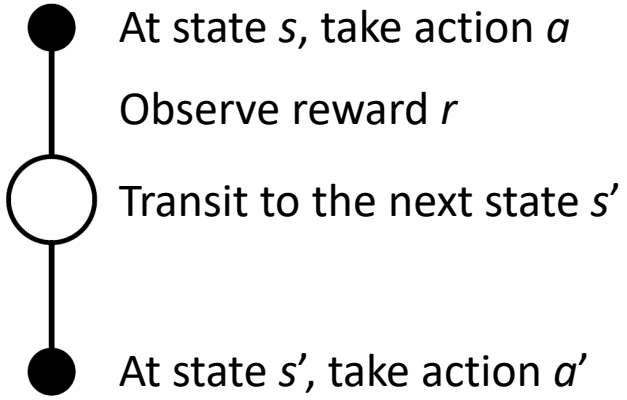
MC Control vs. TD Control

- Temporal-difference (TD) learning has several advantages over Monte-Carlo (MC)
  - Lower variance
  - Online
  - Incomplete sequences

- Natural idea: use TD instead of MC in our control loop
  - Apply TD to update action value $Q(s,a)$
  - Use ε-greedy policy improvement
  - Update the action value function every time-step

SARSA (On-Policy TD Control)

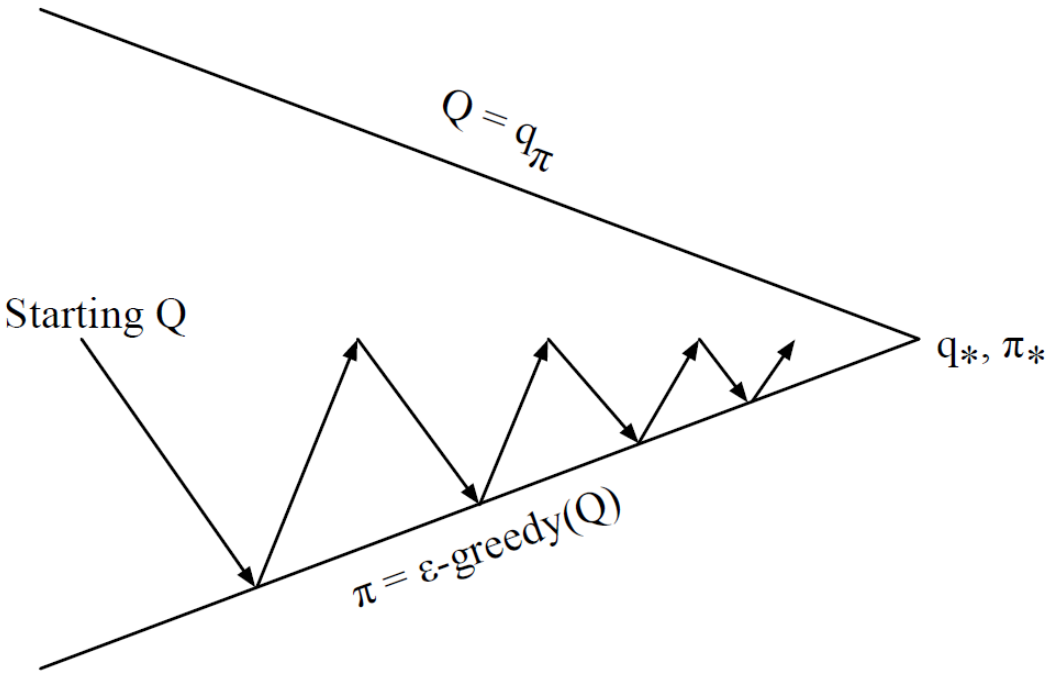
- For each State-Action-Reward-State-Action by the current policy



- Updating action-value functions with Sarsa

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

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

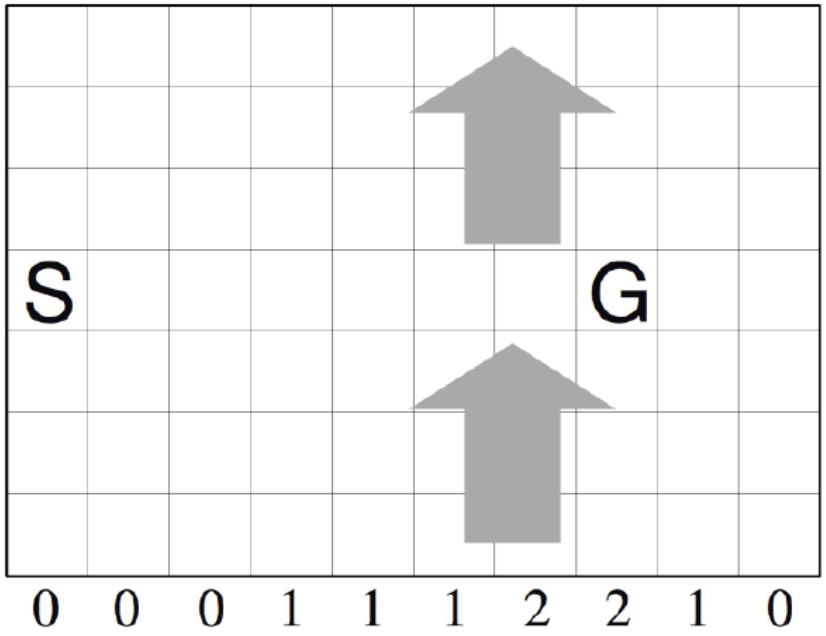
Sarsa: An on-policy TD control algorithm

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

SARSA Example: Windy Gridworld

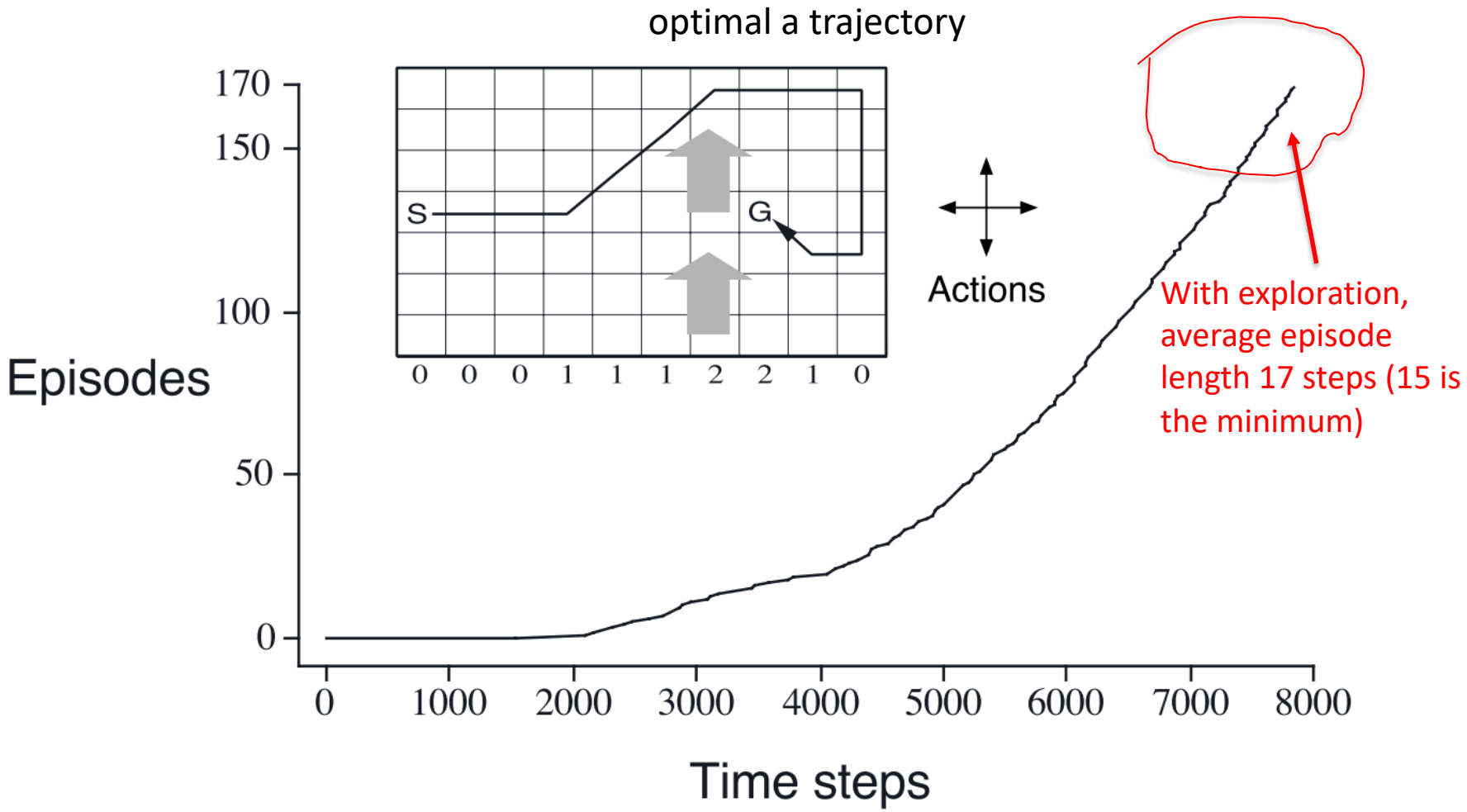
Standard gridworld with start and goal states, but with a crosswind upward through the middle of the grid (may shift up with unknown environment, see the numbers below)



- Reward = -1 per time-step until reaching goal
- Undiscounted

Sutton, Richard S., and Andrew G. Barto. *Reinforcement learning: An introduction*. Vol. 1. No. 1. Cambridge: MIT press, 1998.

SARSA Example: Windy Gridworld



Note: as the training proceeds, the Sarsa policy achieves the goal more and more quickly

Off-Policy Learning

- Evaluate target policy $\pi(a|s)$ to compute $V^{\pi}(s)$ or $Q^{\pi}(s,a)$
- While following behavior policy (used to simulate trajectories) $\mu(a|s)$

$$\{s_1, a_1, r_2, s_2, a_2, \ldots, s_T\} \sim \mu$$

- Why off-policy learning is important?
  - Learn from observing humans or other agents
  - Re-use experience generated from old policies
  - Learn about optimal policy while following exploratory policy
  - Learn about multiple policies while following one policy

Importance Sampling

- Estimate the expectation of a different distribution

$$\begin{aligned}\mathbb{E}_{x\sim p}[f(x)] &= \int_x p(x)f(x)dx \\ &= \int_x q(x)\frac{p(x)}{q(x)}f(x)dx \\ &= \mathbb{E}_{x\sim q}\left[\frac{p(x)}{q(x)}f(x)\right]\end{aligned}$$

- Re-weight each instance by $\beta(x) = \frac{p(x)}{q(x)}$

Importance Sampling for Off-Policy Monte-Carlo

- Use returns generated from μ to evaluate π
- Weight return G_t according to importance ratio between policies
- Multiply importance ratio along with episode

$$\{s_1, a_1, r_2, s_2, a_2, \ldots, s_T\} \sim \mu$$

$$G_t^{\pi/\mu} = \frac{\pi(a_t|s_t)}{\mu(a_t|s_t)} \frac{\pi(a_{t+1}|s_{t+1})}{\mu(a_{t+1}|s_{t+1})} \cdots \frac{\pi(a_T|s_T)}{\mu(a_T|s_T)} G_t$$

- Update value towards corrected return

$$V(s_t) \leftarrow V(s_t) + \alpha(G_t^{\pi/\mu} - V(s_t))$$

  - Cannot use if μ is zero when π is non-zero
  - Importance sample can dramatically increase variance

Importance Sampling for Off-Policy TD

- Use TD targets generated from μ to evaluate π
- Weight TD target $r+\gamma V(s')$ by importance sampling
- Only need a single importance sampling correction

$$V(s_t) \leftarrow V(s_t) + \alpha \left(\frac{\pi(a_t|s_t)}{\mu(a_t|s_t)} (r_{t+1} + \gamma V(s_{t+1})) - V(s_t) \right)$$

importance sampling correction

TD target

- Much lower variance than Monte-Carlo importance sampling
- Policies only need to be similar over a single step

Q-Learning

- For off-policy learning of action-value $Q(s,a)$
- No importance sampling is required (why?)
- The next action is chosen using behavior policy $a_{t+1} \sim \mu(\cdot|s_t)$
- But we consider alternative successor action $a \sim \pi(\cdot|s_t)$
- And update $Q(s_t,a_t)$ towards value of alternative action

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

action from π not μ

Off-Policy Control with Q-Learning

- Allow both behavior and target policies to improve
- The target policy π is greedy w.r.t. $Q(s,a)$

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

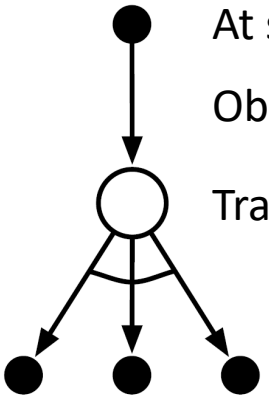
- The behavior policy μ is e.g. ε-greedy policy w.r.t. $Q(s,a)$
- The Q-learning target then simplifies

$$\begin{aligned} r_{t+1} + \gamma Q(s_{t+1}, a') &= r_{t+1} + \gamma Q(s_{t+1}, \arg\max_{a'} Q(s_{t+1}, a')) \\ &= r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') \end{aligned}$$

- Q-learning update

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

Q-Learning Control Algorithm



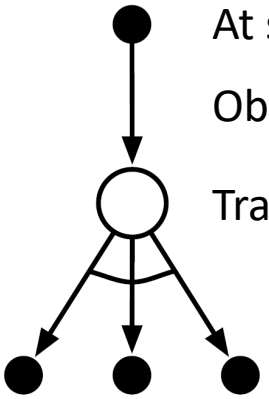
$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t))$$

- Theorem: Q-learning control converges to the optimal action-value function

$$Q(s, a) \rightarrow Q^*(s, a)$$

Watkins, Christopher JCH, and Peter Dayan. "Q-learning." *Machine learning* 8.3-4 (1992): 279-292.

Q-Learning Control Algorithm

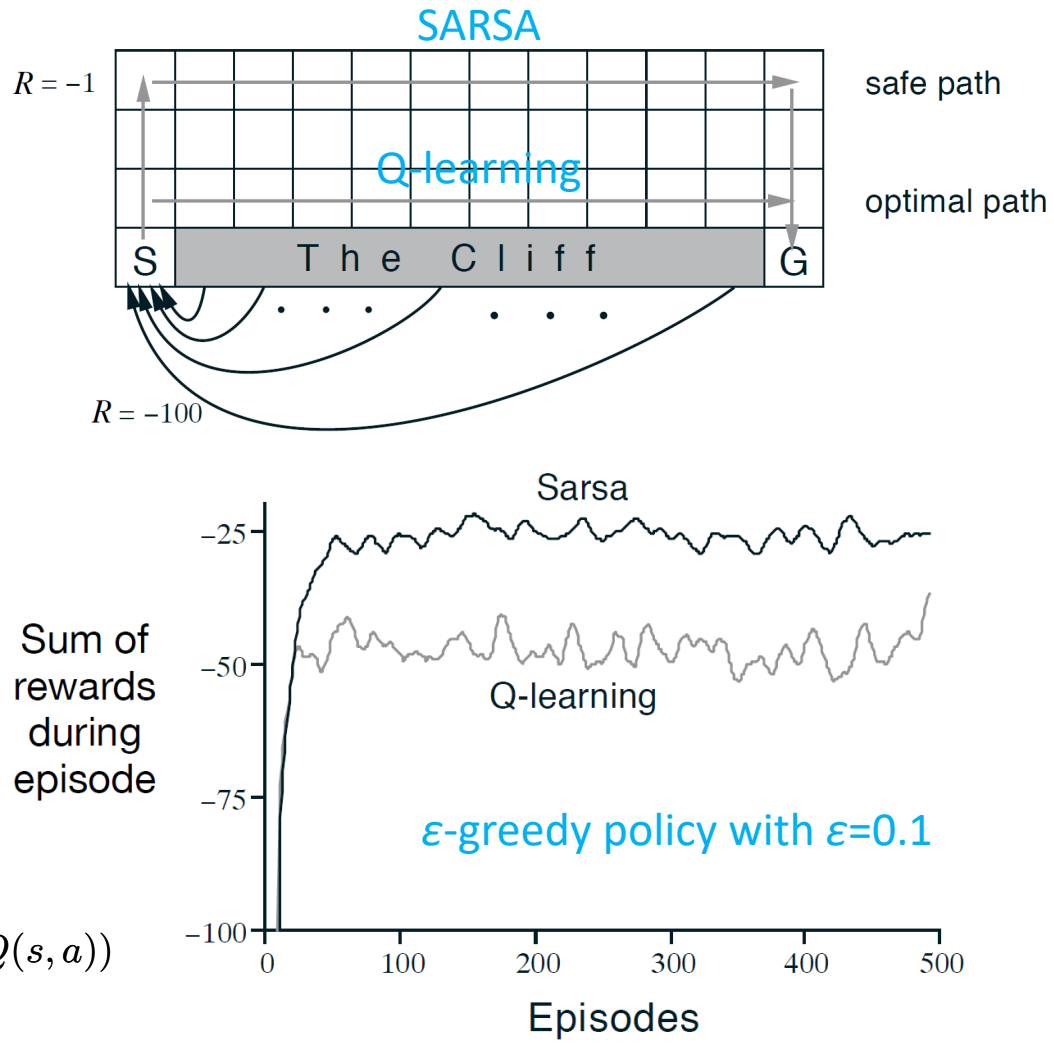


$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t))$$

- Why Q-learning is an off-policy control method?
  - 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 $\pi(s_{t+1}) = \arg\max_{a'} Q(s_{t+1}, a')$
- Why no importance sampling?
  - Action value function not state value function

SARSA vs. Q-Learning Experiments

- Cliff-walking
  - Undiscounted reward
  - Episodic task
  - Reward = -1 on all transitions
  - Stepping into cliff area incurs -100 reward and sent the agent back to the start
- Why the results are like this?



SARSA:

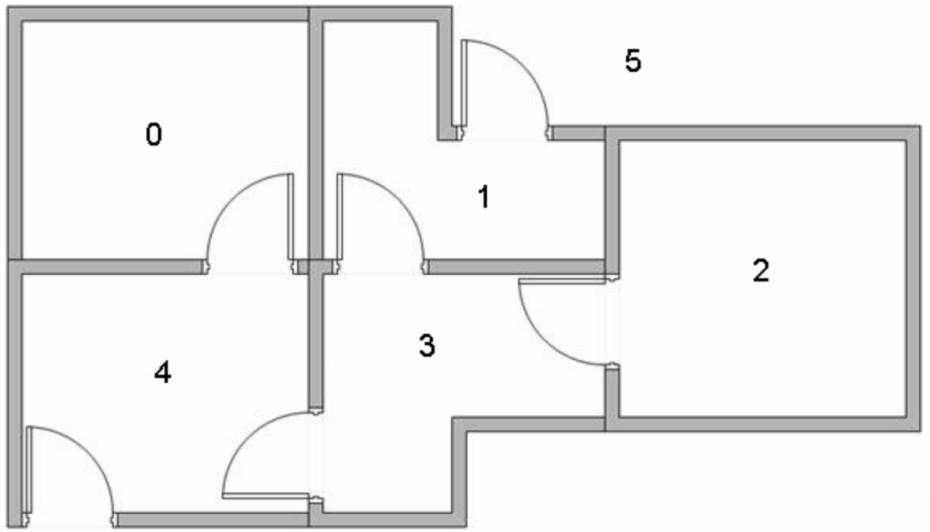
$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma Q(s',a') - Q(s,a))$$

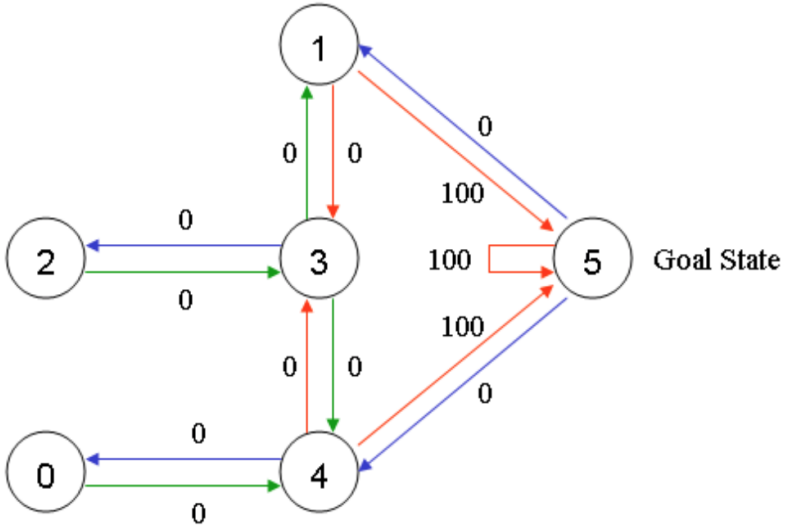
Q-learning:

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

Step-by-step example of Q-learning

- Agents are expected to find a route outside the room. Once going to the outside, i.e., No.5, reward 100.
- Each room, including outside, is a state, and the agent's move from one room to another will be an action.





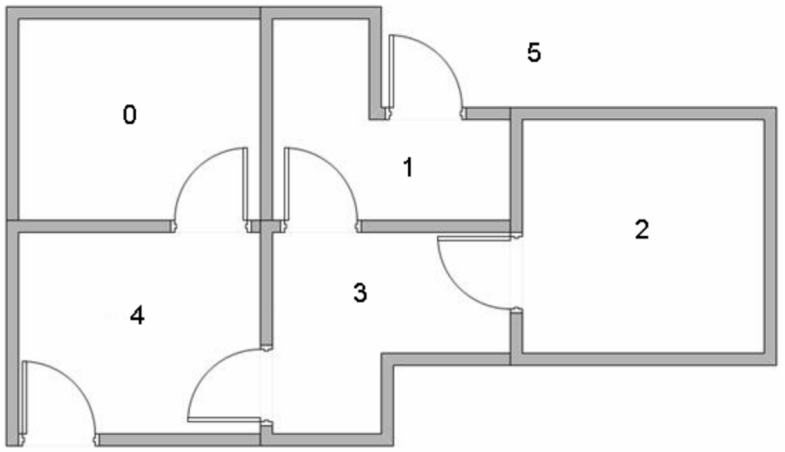
MDP

http://mnemstudio.org/path-finding-q-learning-tutorial.htm

Step-by-step example of Q-learning

- The reward table:

$$R = \begin{array}{c} \\ \text{State} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} & & & \text{Action} & & \end{array} \\ \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} -1 & -1 & -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & 0 & -1 & 100 \\ -1 & -1 & -1 & 0 & -1 & -1 \\ -1 & 0 & 0 & -1 & 0 & -1 \\ 0 & -1 & -1 & 0 & -1 & 100 \\ -1 & 0 & -1 & -1 & 0 & 100 \end{bmatrix} \end{array}$$



- We use -1 in the table to represent no connection between nodes.

Step-by-step example of Q-learning

- Agent can learn through experience.
- The agent can pass from one room to another but has no knowledge of the environment (model), and doesn't know which sequence of doors lead to the outside.

$$Q(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \left(\overbrace{\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}}}^{\text{learned value}} - \underbrace{Q(s_t, a_t)}_{\text{old value}} \right)$$

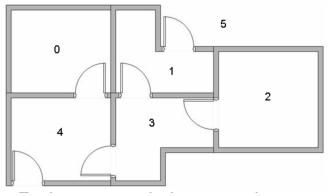
- We use -1 in the table to represent no connection between nodes.

$$Q_0 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Step-by-step example of Q-learning

$R =$

State \ Action	0	1	2	3	4	5
0	−1	−1	−1	−1	0	−1
1	−1	−1	−1	0	−1	100
2	−1	−1	−1	0	−1	−1
3	−1	0	0	−1	0	−1
4	0	−1	−1	0	−1	100
5	−1	0	−1	−1	0	100



- Set γ = 0.8, the initial state as Room 1.
  - Look at the second row (state 1) of matrix R.
    - There are two possible actions for the current state 1: go to state 3, or go to state 5.
    - By random selection, we select to go to 5 as our action.
  - Now let's imagine what would happen if our agent were in state 5.
    - Look at the sixth row of the reward matrix R (i.e. state 5). It has 3 possible actions: go to state 1, 4 or 5.

Q(state, action) = R(state, action) + Gamma ∗ Max[Q(nextstate, allactions)]

Q(1,5) = R(1,5) + 0.8 ∗ Max[Q(5,1),Q(5,4),Q(5,5)] = 100 + 0.8 ∗ 0 = 100

Step-by-step example of Q-learning

$$R = \begin{array}{c} \\ \text{State} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} & & & \text{Action} & & \\ 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} -1 & -1 & -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & 0 & -1 & 100 \\ -1 & -1 & -1 & 0 & -1 & -1 \\ -1 & 0 & 0 & -1 & 0 & -1 \\ 0 & -1 & -1 & 0 & -1 & 100 \\ -1 & 0 & -1 & -1 & 0 & 100 \end{bmatrix} \end{array}$$

- The next state, 5, now becomes the current state. Because 5 is the goal state, we've finished one episode.
- Our agent's brain now contains an updated matrix Q as:

$$Q_1 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Q(state, action) = R(state, action) + Gamma $*$ Max[Q(nextstate, allactions)]

Q(1,5) = R(1,5) + 0.8 $*$ Max[Q(5,1),Q(5,4),Q(5,5)] = 100 + 0.8 $*$ 0 = 100

Step-by-step example of Q-learning

$$R = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} \text{Action} & & & & & \\ 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} -1 & -1 & -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & 0 & -1 & 100 \\ -1 & -1 & -1 & 0 & -1 & -1 \\ -1 & 0 & 0 & -1 & 0 & -1 \\ 0 & -1 & -1 & 0 & -1 & 100 \\ -1 & 0 & -1 & -1 & 0 & 100 \end{bmatrix} \end{array} \begin{array}{l} \\ \text{State} \end{array}$$

- For the next episode, we start with a randomly choser state, say state 3, as our initial state.
  - Look at the fourth row of matrix R; it has 3 possible actions: go to state 1, 2 or 4.
  - By random selection, we select to go to state 1 as our action.
  - Now we imagine that we are in state 1. Look at the second row of reward matrix R (i.e. state 1). It has 2 possible actions: go to state 3 or state 5.
  - Then, we compute the Q value:

$$Q(3,1) = R(3,1)+0.8*\text{Max}[Q(1,3),Q(1,5)] = 0+0.8*\text{Max}(0,100) = 80$$

- The next state, 1, now becomes the current state. We repeat the inner loop of the Q learning algorithm because state 1 is not the goal state.

$$Q_2 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 80 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Step-by-step example of Q-learning

$$R = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} \text{Action} & & & & & \\ 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} -1 & -1 & -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & 0 & -1 & 100 \\ -1 & -1 & -1 & 0 & -1 & -1 \\ -1 & 0 & 0 & -1 & 0 & -1 \\ 0 & -1 & -1 & 0 & -1 & 100 \\ -1 & 0 & -1 & -1 & 0 & 100 \end{bmatrix} \end{array} \begin{array}{l} \\ \text{State} \end{array}$$

- So, starting the new loop with the current state 1,
  - there are two possible actions: go to state 3, or go to state 5. By lucky draw, our action selected is 5.
  - Now, imaging we're in state 5, there are three possible actions: go to state 1, 4 or 5.
  - We compute the Q value using the maximum value of these possible actions.

$$Q(1, 5) = R(1, 5)+0.8*\text{Max}[Q(5, 1), Q(5, 4), Q(5, 5)] = 100+0.8*0 = 100$$

- The updated entries of matrix Q, Q(5, 1), Q(5, 4), Q(5, 5), are all zero.
- The result of this computation for Q(1, 5) is 100 because of the instant reward from R(1, 5). This result does not change the Q matrix.

Step-by-step example of Q-learning

$R =$

State \ Action	0	1	2	3	4	5
0	−1	−1	−1	−1	0	−1
1	−1	−1	−1	0	−1	100
2	−1	−1	−1	0	−1	−1
3	−1	0	0	−1	0	−1
4	0	−1	−1	0	−1	100
5	−1	0	−1	−1	0	100

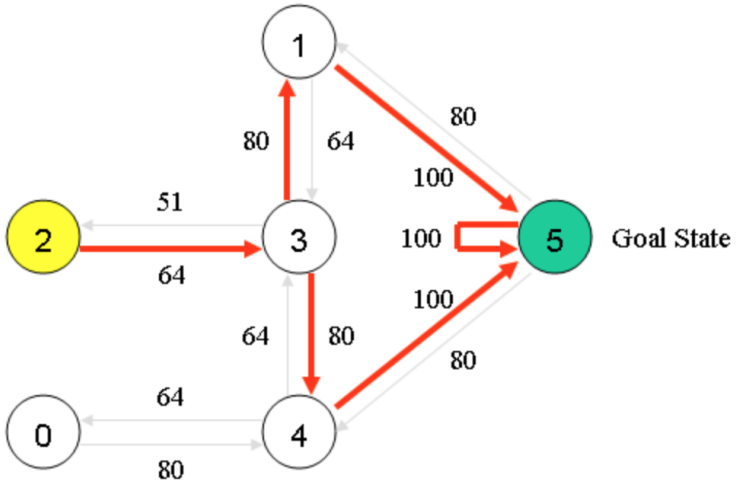
- Because 5 is the goal state, we finish this episode.
- Our agent has the updated matrix Q as:

$$Q_2 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 80 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

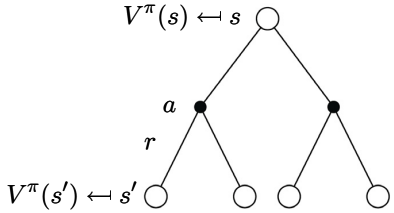
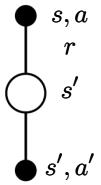
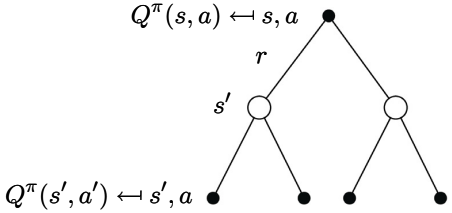
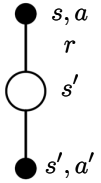
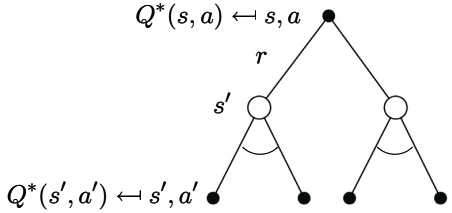
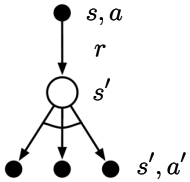
Step-by-step example of Q-learning

- If our agent learns more through further episodes, it will finally reach convergence values in matrix Q.
- Once the matrix Q gets close enough to a state of convergence, we know our agent has learned the most optimal paths to the goal state.
- Tracing the best sequences of states is as simple as following the links with the highest values at each state.

$$Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 400 & 0 \\ 0 & 0 & 0 & 320 & 0 & 500 \\ 0 & 0 & 0 & 320 & 0 & 0 \\ 0 & 400 & 256 & 0 & 400 & 0 \\ 320 & 0 & 0 & 320 & 0 & 500 \\ 0 & 400 & 0 & 0 & 400 & 500 \end{bmatrix} \end{array}$$



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 $V(s) \leftarrow \mathbb{E}[r + \gamma V(s') \mid s]$	TD Learning $V(s) \overset{\alpha}{\leftarrow} r + \gamma V(s')$
Q-Policy Iteration $Q(s,a) \leftarrow \mathbb{E}[r + \gamma Q(s',a') \mid s,a]$	SARSA $Q(s,a) \overset{\alpha}{\leftarrow} r + \gamma Q(s',a')$
Q-Value Iteration $Q(s,a) \leftarrow \mathbb{E}\left[r + \gamma \max_{a'} Q(s',a') \mid s,a\right]$	Q-Learning $Q(s,a) \overset{\alpha}{\leftarrow} r + \gamma \max_{a'} Q(s',a')$

where

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

Convergence Proof of Q-learning

- Recall that the optimal Q-function (SxA → R) is a fixed point of a contraction operator T:

$$\left(T^{*}Q\right)(s,a) \triangleq r(s,a) + \gamma \sum_{S'} \mathcal{P}\left(s'|s,a\right) \max_{a' \in \mathcal{A}} Q\left(s',a'\right)$$

- Let $\{s_t\}$ be a sequence of states by following policy π, $\{a_t\}$ the sequence of corresponding actions and $\{r_t\}$ the sequence of rewards.
- Then, given any initial estimate Q_0, Q-learning uses the following update rule:

$$Q_{t+1}(s_t,\ a_t) = Q_t(s_t,\ a_t) + \alpha_t(s_t,\ a_t)[r_t + \gamma \max_{b \in A} Q_t(s_{t+1},\ b) - Q_t(s_t,\ a_t)]$$

Convergence Proof of Q-learning

$$Q_{t+1}(s_t,\ a_t) = Q_t(s_t,\ a_t) + \alpha_t(s_t,\ a_t)[r_t + \gamma \max_{b \in A} Q_t(s_{t+1},\ b) - Q_t(s_t,\ a_t)]$$

Theorem:

Given a finite MDP, the Q-learning algorithm given by the above update rule converges w.p.1 to the optimal Q- function as long as

$$\sum_t \alpha_t(s,\ a) = \infty, \qquad \sum_t \alpha_t^2(s,\ a) < \infty$$

For all (s, a)∈S×A.

- At the (t + 1)th update, only the component (s_t, a_t) is updated. The theorem suggests that all state-action pairs be visited infinitely often.

Convergence Proof of Q-learning

Theorem (Stochastic Approximation):

- The random process $\{\Delta_t\}$ taking values in R^n and defined as

$$\triangle_{t+1}(x) = (1 - \alpha_t(x))\triangle_t(x) + \alpha_t(x)F_t(x)$$

converges to zero w.p.1 under the following assumptions:

$\mathcal{F}_t$: All the past information at t

- $0 \leq \alpha_t \leq 1, \sum_t \alpha_t(x) = \infty$ *and* $\sum_t \alpha_t^2(x) < \infty$
- $\|\mathbb{E}[F_t(x)|\mathcal{F}_t]\|_W \leq \gamma \|\Delta_t\|_W$, *with* $\gamma < 1$
- $\text{var}[F_t(x)|\mathcal{F}_t] \leq C\left(1 + \|\Delta_t\|_W^2\right)$, *for* $C > 0$

- Tommi Jaakkola, Michael I. Jordan, and Satinder P. Singh. On the convergence of stochastic iterative dynamic programming algorithms. Neural Computation, 66):1185-1201, 1994.

Convergence Proof of Q-learning

- Stochastic approximation theorem helps prove the converge of Q-learning
  - Start by rewriting the Q-learning update rule as

$$Q_{t+1}(s_t,\ a_t) = (1 - \alpha_t(s_t,\ a_t))Q_t(s_t,\ a_t)$$

$$+\alpha_t(s_t,\ a_t)[r_t + \gamma \max_{b \in A} Q_t(x_{t+1},\ b)]$$

  - Subtracting from both sides the quantity $Q^*(s_t, a_t)$ and letting $\triangle_t(s,\ a) = Q_t(s,\ a) - Q^*(s,\ a)$

  - We thus have:

$$\triangle_t(s_t,\ a_t) = (1 - \alpha_t(s_t,\ a_t))\triangle_t(s_t,\ a_t))$$

$$+\alpha_t(s,\ a)[r_t + \gamma \max_{b \in A} Q_t(s_{t+1},\ b) - Q^*(s_t, a_t)].$$

Convergence Proof of Q-learning

- If we write

$$F_t(s,a) = r(s,a,S(s,a)) + \gamma \max_{b \in A} Q_t(y,b) - Q^*(s,a),$$

  - where S(s,a) is a random sample state obtained from the Markov chain from the transition model, we have

$$\begin{aligned}\mathbb{E}\left[F_t(s,a)|\mathcal{F}_t\right] &= \sum_{y \in \mathcal{S}} P_a(s,y)\left[r(s,a,y) + \gamma \max_{b \in \mathcal{A}} Q_t(y,b) - Q^*(s,a)\right] \\ &= \left(TQ_t\right)(s,a) - Q^*(s,a)\end{aligned}$$

  - Using the fact that $Q^* = TQ^*$,

$$\mathbb{E}[F_t(s,a)|\mathcal{F}_t] = (TQ_t)(s,a) - (TQ^*)(s,a)$$

Convergence Proof of Q-learning

- Recall that we know that this operator is a contraction in the max-norm, i.e.

$$\|Q_k - Q^*\|_\infty = \left\|(T^*)^{(k)} Q_0 - T^* Q^*\right\|_\infty \leq \gamma^k \|Q_0 - Q^*\|_\infty$$

- So

$$\|\mathbb{E}[F_t(s,a)|\mathcal{F}_t]\|_\infty \leq \gamma \|Q_t - Q^*\|_\infty = \gamma \|\Delta_t\|_\infty$$

Convergence Proof of Q-learning

- for condition 3:

$$\mathrm{E}[F_t(s,a)|\mathcal{F}_t] = (TQ_t)(s,a) - (TQ^*)(s,a)$$

$$\begin{aligned}
&\mathrm{var}\left[F_t(s)|\mathcal{F}_t\right] \\
&= \mathbb{E}\left[\left(r(s,a,S(s,a)) + \gamma \max_{b\in A} Q_t(y,b) - Q^*(s,a) - \left(TQ_t\right)(s,a) + Q^*(s,a)\right)^2\right] \\
&= \mathbb{E}\left[\left(r(s,a,S(s,a)) + \gamma \max_{b\in A} Q_t(y,b) - \left(TQ_t\right)(s,a)\right)^2\right] \quad \text{Remove Q*} \\
&= \mathrm{var}\left[r(s,a,S(s,a)) + \gamma \max_{b\in A} Q_t(y,b)|\mathcal{F}_t\right] \quad \text{By definition of variance}
\end{aligned}$$

which depends on Q_t at most linearly and r is also bounded (assumption). Thus the variance condition holds:

$$\mathrm{var}\left[F_t(s)|\mathcal{F}_t\right] \leq C\left(1 + \|\Delta_t\|_W^2\right)$$

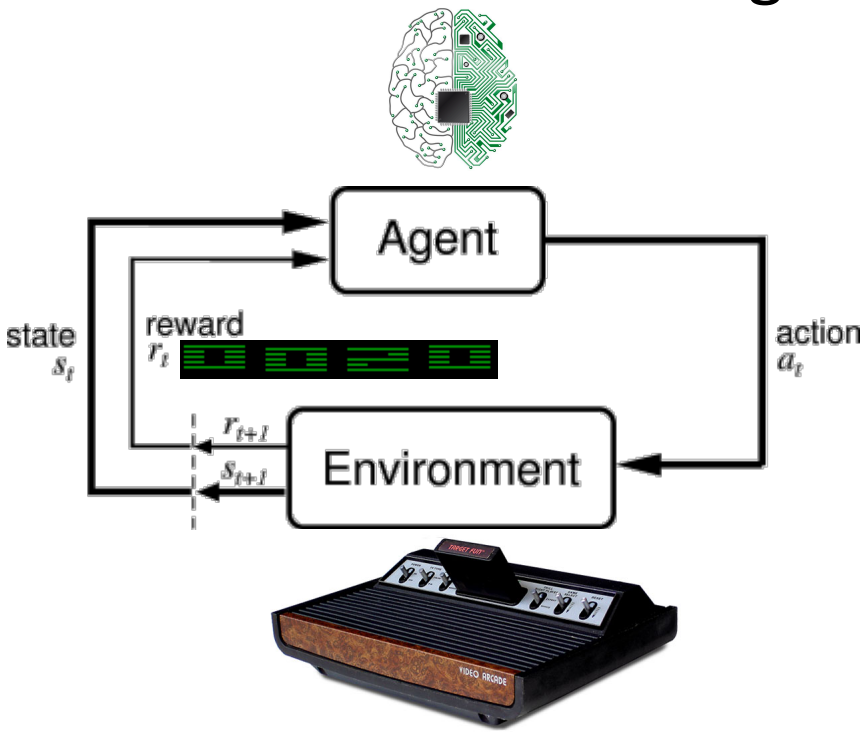
Melo, Francisco S. "Convergence of Q-learning: A simple proof." *Institute Of Systems and Robotics, Tech. Rep* (2001): 1-4.

Content

- Reinforcement Learning
  - The model-based methods
    - Markov Decision Process
    - Planning by Dynamic Programming
    - Convergence Analysis
  - The model-free methods
    - Model-free Prediction
      - Monte-Carlo and Temporal Difference
    - Model-free Control
      - On-policy SARSA and off-policy Q-learning
      - Convergence Analysis
- **Deep Reinforcement Learning Examples**
  - **Atari games**
  - Alpha Go

Reinforcement learning

- Computerised agent: Learning what to do
  - How to map situations (states) to actions so as to maximise a numerical reward signal



Sutton, Richard S., and Andrew G. Barto. Reinforcement learning: An introduction. MIT press, 1998.

Reinforcement learning

- {state *s*, action *a*, reward *r*} are made in stages

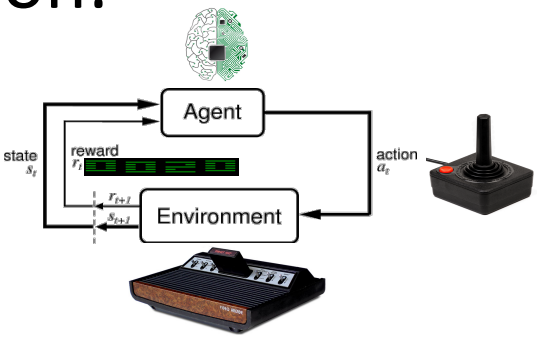
$$s_0, a_0, r_1, s_1, a_1, r_2, s_2, \ldots, s_{n-1}, a_{n-1}, r_n, s_n$$

The outcome of each decision is not fully predictable but can be observed before the next decision is made

- The objective is
  - to maximize a numerical measure of total reward

$$R = r_1 + r_2 + r_3 + \ldots + r_n$$

- Decisions cannot be viewed in isolation:
  - need to balance desire for immediate reward with possibility of high future reward



Sutton, Richard S., and Andrew G. Barto. Reinforcement learning: An introduction. MIT press, 1998.

Q-learning

- For game, the future *discounted* reward at time *t*

$R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+1} \ldots,$ where γ is discount factor (aka prob. of terminating)

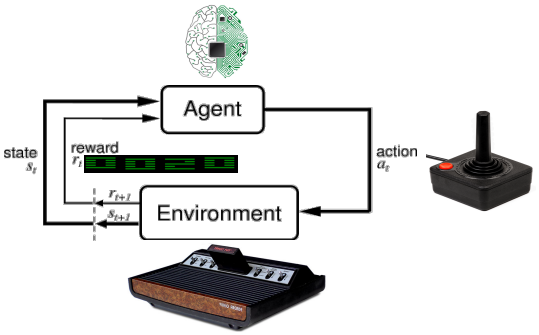
- We define the optimal action-value function

$Q^*(s,a) = \max_\pi [r_t + \gamma r_{t+1} + \gamma^2 r_{t+1} + \ldots \mid s_t = s, a_t = a, \pi],$ where $\pi = p(a \mid s)$ is a policy

- *Bellman* equation

$Q^*(s,a) = E_{s' \sim \varepsilon}[r + \gamma \max_{a'} Q^*(s,a) \mid s_t = s, a_t = a],$

where ε is sample distribution from an emulator



- So iteratively update Q* as above

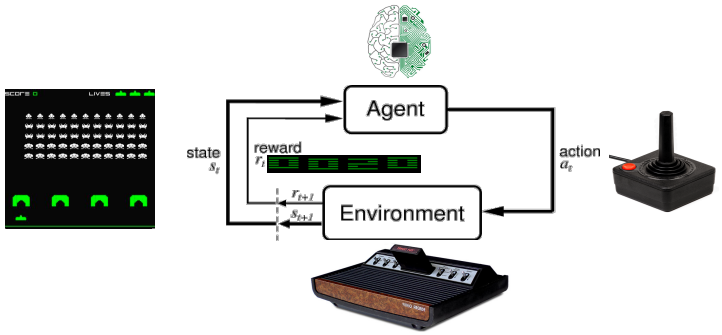
and $a_t^* = \max_a Q^*(s_t, a_t)$

Sutton, Richard S., and Andrew G. Barto. Reinforcement learning: An introduction. MIT press, 1998.

A simplified Q-learning algorithm

```
initialize Q[num_states,num_actions] arbitrarily
observe initial state s
repeat
      select and carry out an action a
      observe reward r and new state s'
      Q[s,a] = Q[s,a] + α(r + γ max_a' Q[s',a'] - Q[s,a])
      s = s'
until terminated
```

Maintain a Q table
and keep updating it



http://www.nervanasys.com/demystifying-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

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

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

target

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

- 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

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

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

target

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

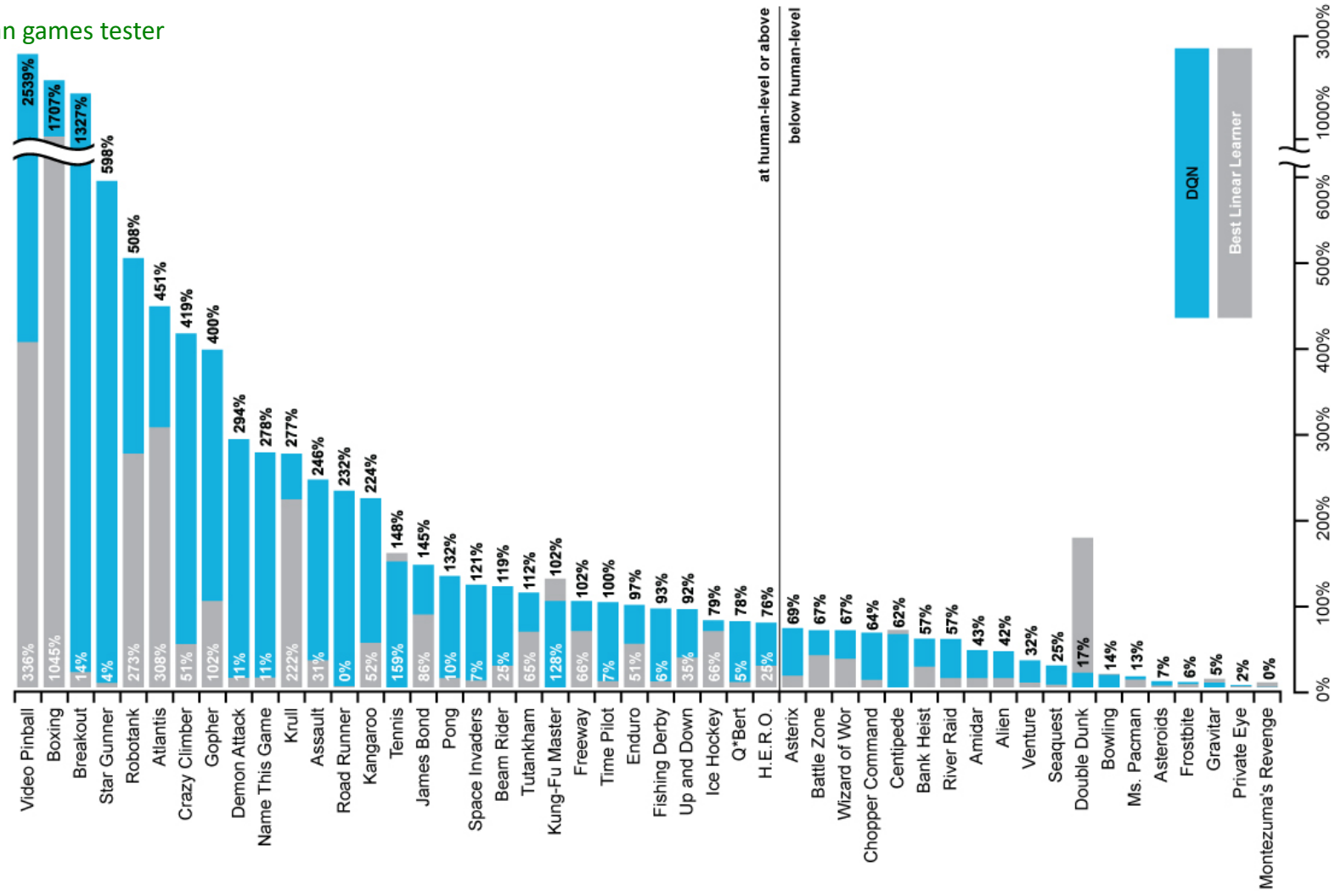
Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise 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.

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
      - On-policy SARSA and off-policy Q-learning
- Deep Reinforcement Learning Examples
  - Atari games
  - **Alpha Go**

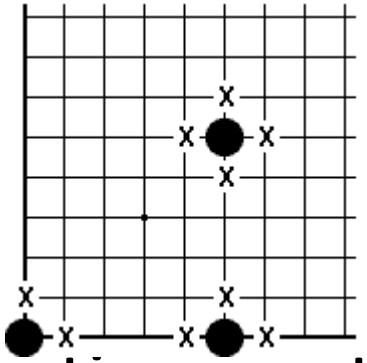
The Go Game

- Go (Weiqi) is an ancient game originated from China, with a history over 3000 years

The board: 19 x 19 lines, making 361 intersections (`points').
The move: starting from Black, stones are placed on points in alternation until the end
The goal: control the max. territory

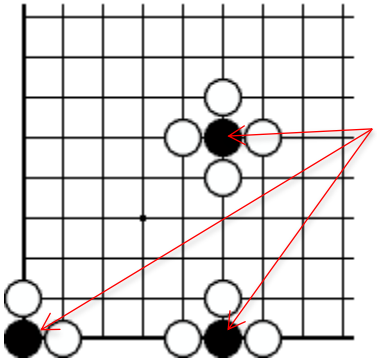
The Go Game: liberties

- Liberties: the unoccupied intersections (or points) that are horizontally or vertically adjacent to the stone



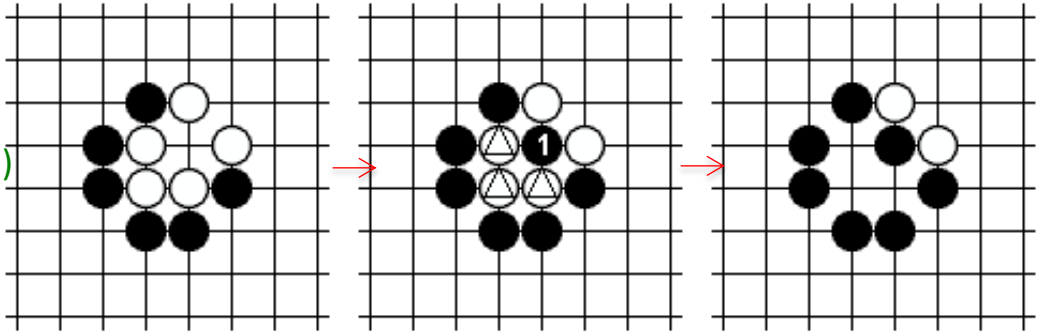
Liberties are marked as X

- Stones without liberties must be removed



Need to be removed (thus captured)

Black captures 3 stones from White



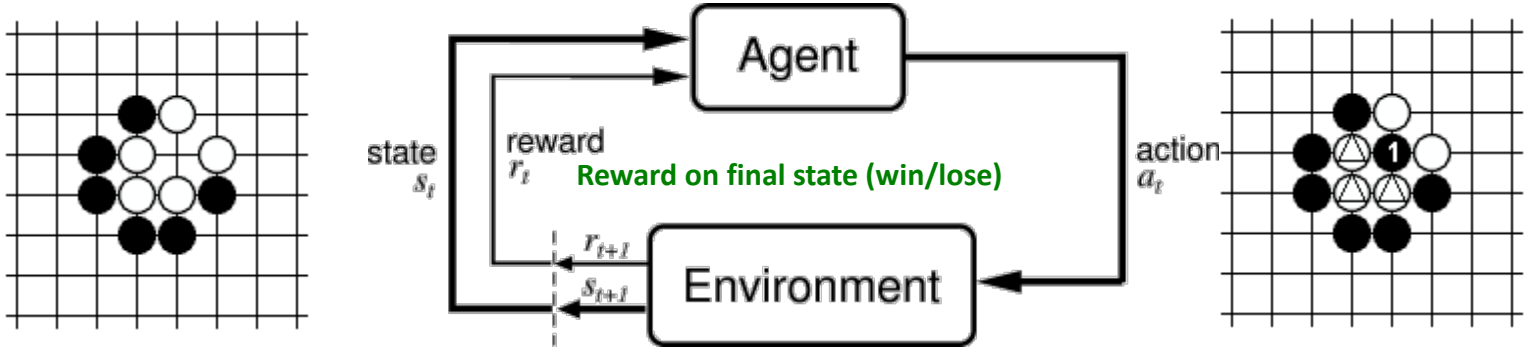
One of the great challenges of AI

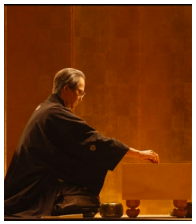
- It is far more complex than chess
  - Chess has 35^{80} possible moves
  - Go has 250^{150} possible moves
- Computers match or surpass top humans in *chess*, *backgammon*, *poker*, even *Jeopardy*
- But not Go game as 2016

https://jedionston.files.wordpress.com/2015/02/go-game-storm-trooper.jpg

AlphaGo

- Yet another example of deep reinforcement learning



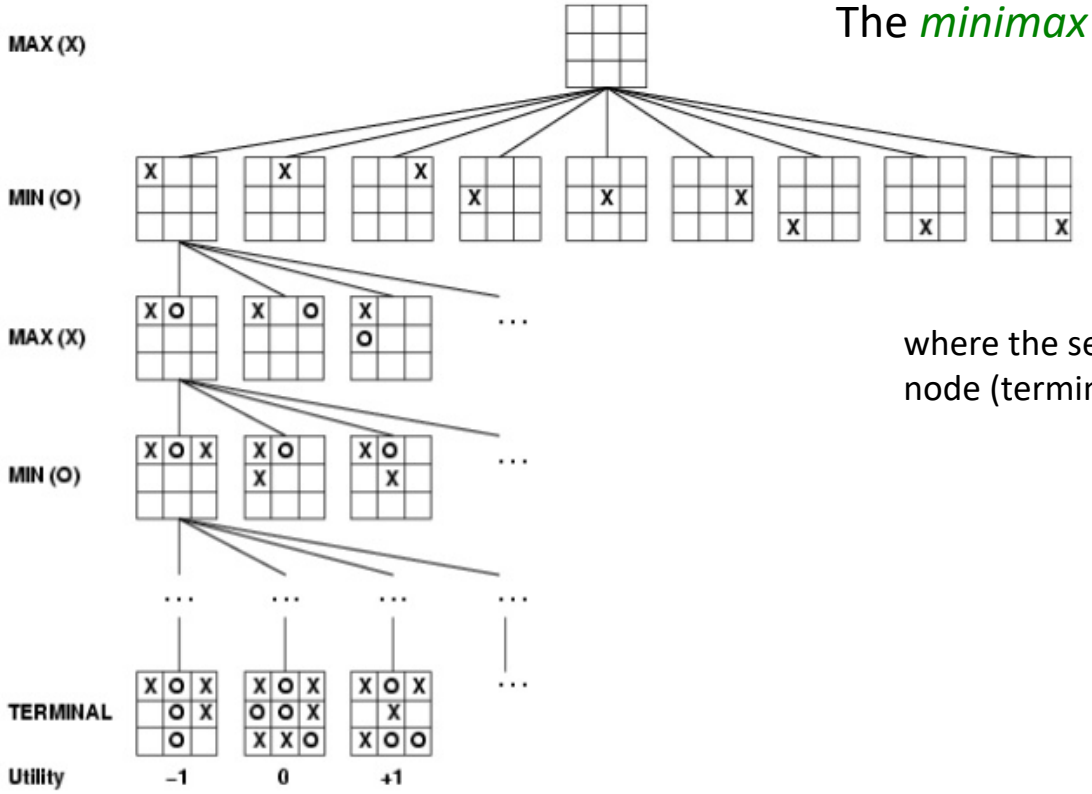


- State space $\mathcal{S}$
- Action space $\mathcal{A}$
- Known transition model: $p(s, a, s')$
- Reward on final states: win or lose

Silver, David, et al. "Mastering the game of Go with deep neural networks and tree search." Nature 529.7587 (2016): 484-489.

Minimax search

- Tic-tac-toe zero-sum two-player game



The *minimax* algorithm is recursively as

$$a_i^* = \text{argmax}_{a_i \in N_t} (\min_{a_{-i} \in N_{t+1}} u(a_{-i}, a_i))$$

a_i : action from you at step i

a_{-i} : action from opponent at step i

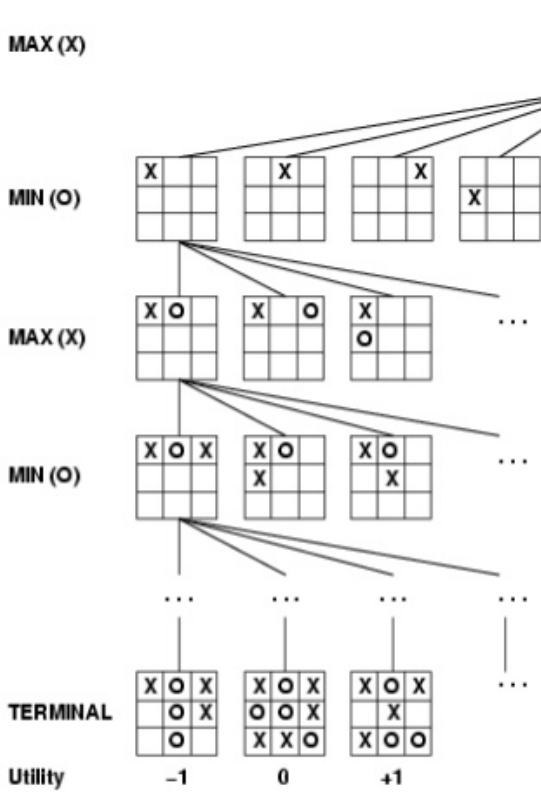
where the set N_{t+1} are the successors of node N_t. If N_t is a leaf node (terminal node), it is value of the node (for you)

Neumann, L. J., & Morgenstern, O. (1947). Theory of games and economic behavior (Vol. 60). Princeton: Princeton university press.

Russell, Stuart, Peter Norvig, and Artificial Intelligence. "A modern approach." Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs 25 (1995): 27.

Minimax search

- Tic-tac-toe zero-sum two-player game



The *minimax* algorithm is recursively as

$$a_i^* = \text{argmax}_{a_i \in N_t} (\min_{a_{-i} \in N_{t+1}} u(a_{-i}, a_i))$$

a_i : action from you at step i

a_{-i} : action from opponent at step i

where the set N_{t+1} are the successors of node N_t. If N_t is a leaf node (terminal node), it is value of the node (for you)

For Tic-tac-toe, num. terminal nodes 9! = 362,880, but for chess **10^{40}** and go game **10^{170}**!

Two solutions by applying prior knowledge:

a) look only at a certain number of moves ahead (piles) and apply heuristic evaluation function *u*, e.g. IBM deep blue

b) reduce N_i the set of possible move (each round), i.e., *p(action|state)*

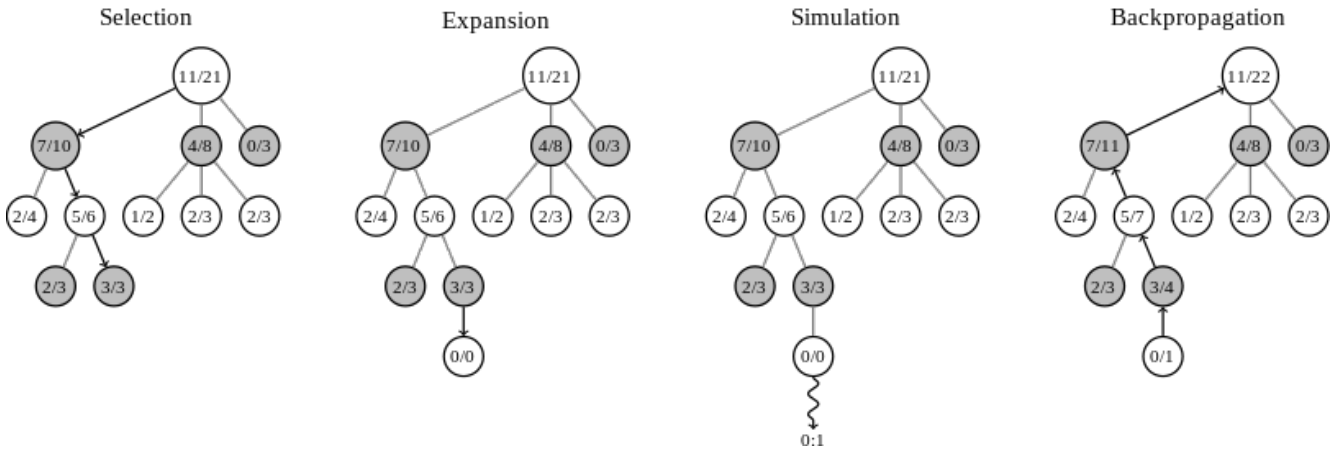
c) Random sampling *by* Monte-Carlo simulation

Neumann, L. J., & Morgenstern, O. (1947). Theory of games and economic behavior (Vol. 60). Princeton: Princeton university press.

Russell, Stuart, Peter Norvig, and Artificial Intelligence. "A modern approach." Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs 25 (1995): 27.

Monte-Carlo tree search (MCTS)

Each tree node stores the number of won w(a) /played playouts n(a)



- Start with the current state in the tree (e.g., node 11/21) as the root and repeatedly running the four steps until timeout
- Then the move (child for the root node) with the most simulations made is typically selected (**robust criteria**) rather than the move with the highest average win rate (**max criteria**)

- Coined 2006, MCTR in real-time expands search tree by simulation and is further extended using the bandit solution for exploration

$$a_i^* = \text{argmax}_{a_i}(w(a) + \sqrt{\frac{2\ln(n)}{n(a)}}),$$

$w(a)$: num win for a

n: num. visits; n(a): num. visits for a

- In 2008, *MoGo* achieved dan (master) level in 9×9 go

Browne, Cameron B., et al. "A survey of monte carlo tree search methods." Computational Intelligence and AI in Games, IEEE Transactions on 4.1 (2012): 1-43.

Kocsis, Levente, and Csaba Szepesvári. "Bandit based monte-carlo planning." Machine Learning: ECML 2006. Springer Berlin Heidelberg, 2006. 282-293.

Coulom, Rémi. "Efficient selectivity and backup operators in Monte-Carlo tree search." Computers and games. Springer Berlin Heidelberg, 2006. 72-83.

Final move section

- the "**best child**" of the root: the move finally played after the simulations
- different empirical ways to define which child is the best
  - **Max child** is the child that has the highest value.
  - **Robust child** is the child with the highest visit count.
  - **Robust-max child** is the child with both the highest visit count and the highest value
    - If there is no robust-max child, more simulations are played until a robust-max child occurs

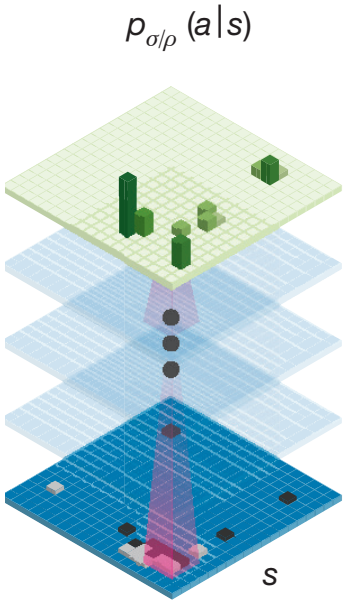
Schadd, Frederik Christiaan. "Monte-Carlo search techniques in the modern board game Thurn and Taxis." *Maastricht University: Maastricht, The Netherlands* (2009).

Selection Method	Max Child		Robust Child	
As Player	1	2	1	2
won	36.6	46.6	43.3	50.0
draw	5.6	6.6	0.0	0.0
loss	56.6	46.6	56.6	50.0

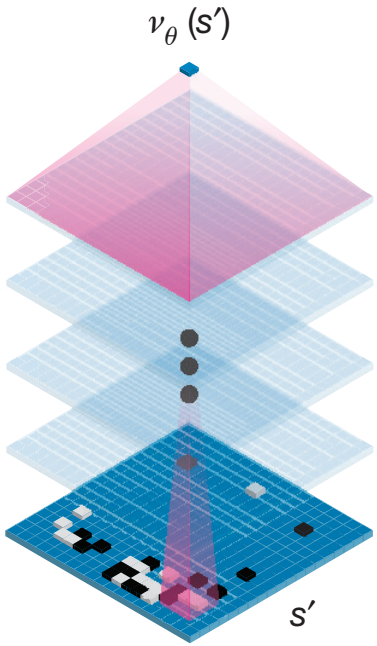
AlphaGo

- Employ two deep convolutional neural networks

Policy network



Value network



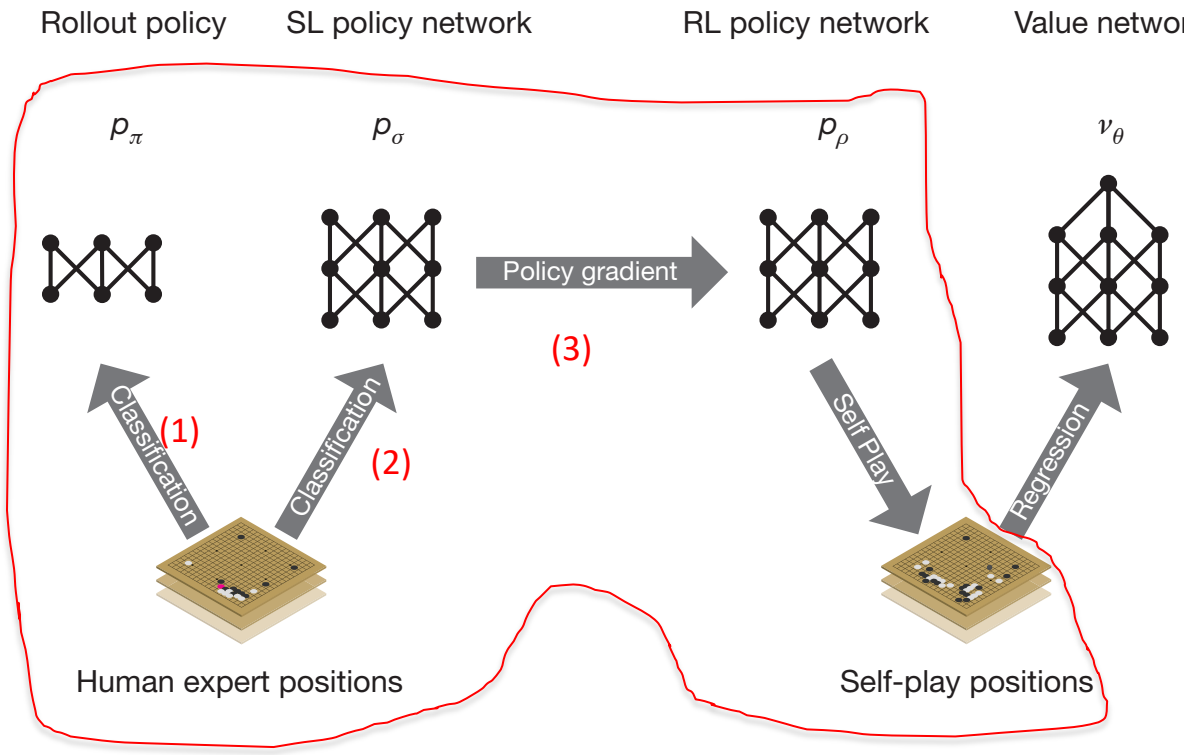
Policy network predicts p(action = a| state = s) of human moves

Value network predicts the expected terminal reward of a given state s

Silver, David, et al. "Mastering the game of Go with deep neural networks and tree search." Nature 529.7587 (2016): 484-489.

AlphaGo

- Training for policy network predicting human moves



Supervised trained (SL) from 30m positions from the KGS Go Server

(1) $p_\pi(a \mid s)$: fast prediction with linear softmax

(2) $p_\delta(a \mid s)$: more accurate prediction with 12-layer CNN. superversied learned from

(3) $p_\delta(a \mid s)$: further tuned by policy gradient from the terminal reward of self-play.

Gradient update

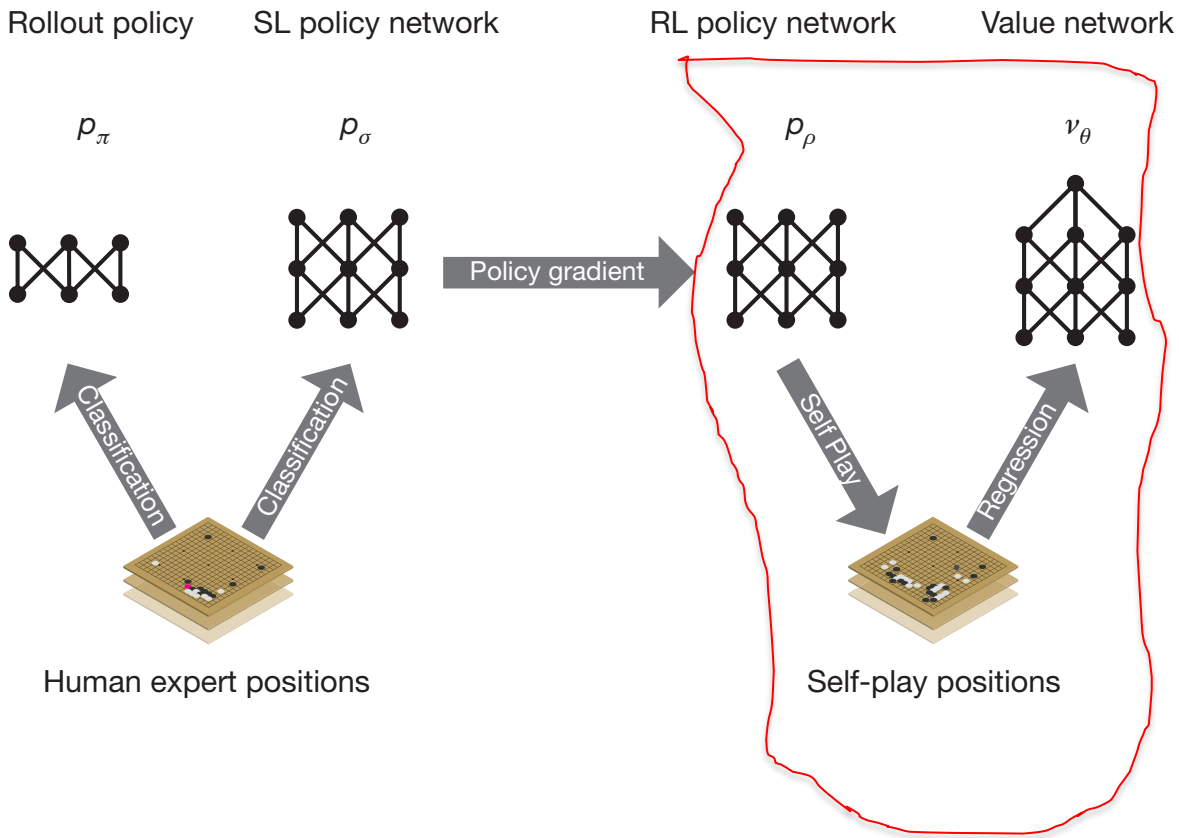
$$\Delta\rho \propto \frac{\partial \log p(a_t \mid s_t)}{\partial \rho} z_t$$

z_t : terminal reward

Silver, David, et al. "Mastering the game of Go with deep neural networks and tree search." Nature 529.7587 (2016): 484-489.

AlphaGo

- Training for value network predicting terminal reward



Selfplay p_p randomly to train value network

$v^\rho(s)$: position evaluation

$v^\rho(s) = \mathrm{E}[z \mid s_t = s, a_t ... T \sim \rho]$

z_t : terminal reward

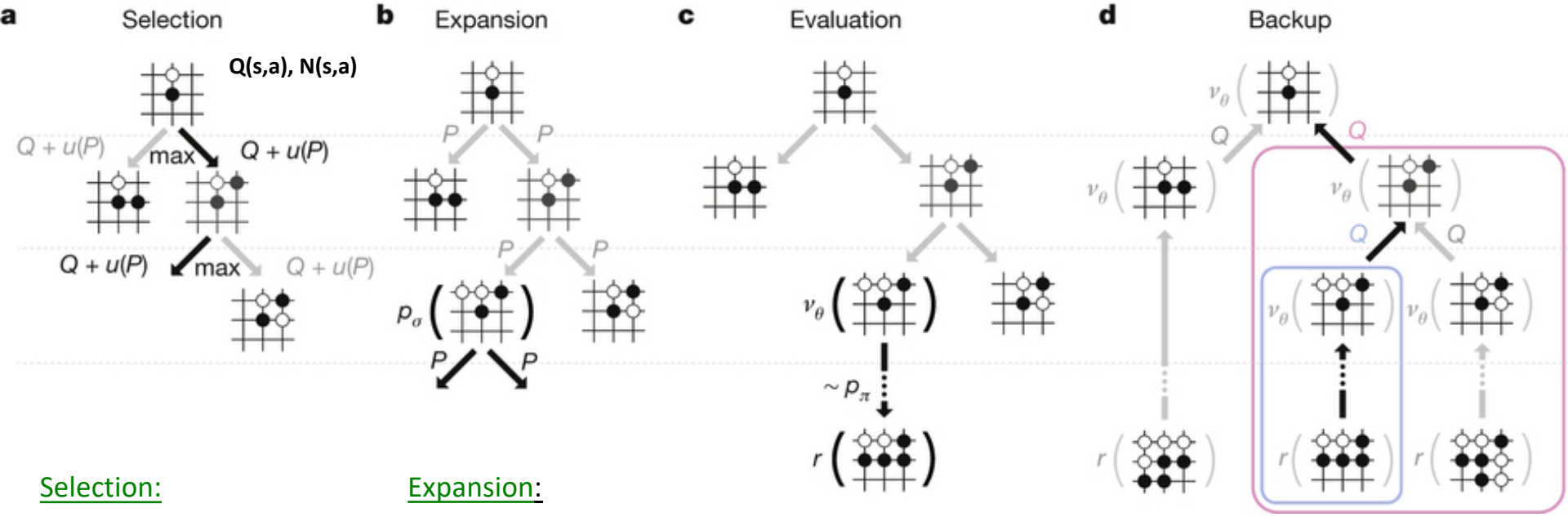
Gradient update:

$$\Delta\theta \propto \frac{\partial \log v_\theta(s)}{\partial \theta}(z - v(s))$$

Silver, David, et al. "Mastering the game of Go with deep neural networks and tree search." Nature 529.7587 (2016): 484-489.

AlphaGo: Monte Carlo tree search

- Combing the two networks (p(a|s) and v(s)) together



Selection:

$$a_t = \underset{a}{\operatorname{argmax}}(Q(s_t,a) + u(s_t,a))$$

where $u(s,a) \propto \frac{P(s,a)}{1+N(s,a)}$

Expansion:

Initiation of a new node with $P(s,a) = p_\sigma(a|s)$

Evaluation:

Random rollout played until terminal step T. leaf node update as $V(s_L) = (1-\lambda)v_\theta(s_L) + \lambda z_L$

Backup (Q and N):

$$Q(s,a) = \frac{1}{N(s,a)} \sum_{i=1}^{n} 1(s,a,i) V(s_L^i)$$

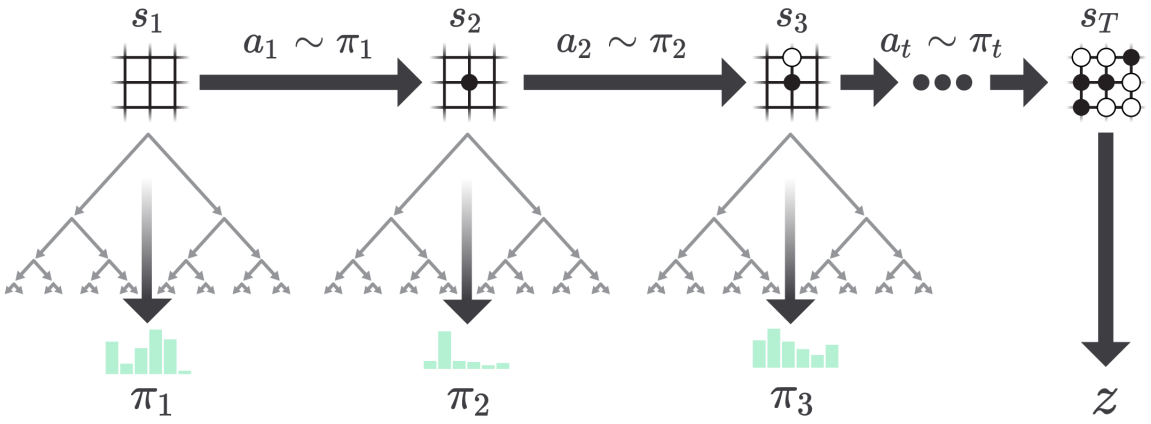
$$N(s,a) = \sum_{i=1}^{n} 1(s,a,i)$$

Once search is done, the most visited move is chosen (including AlphaZero stocastically with a temperature parameter and MuZero)

Silver, David, et al. "Mastering the game of Go with deep neural networks and tree search." Nature 529.7587 (2016): 484-489.

AlphaZero: without human knowledge

- Self-play

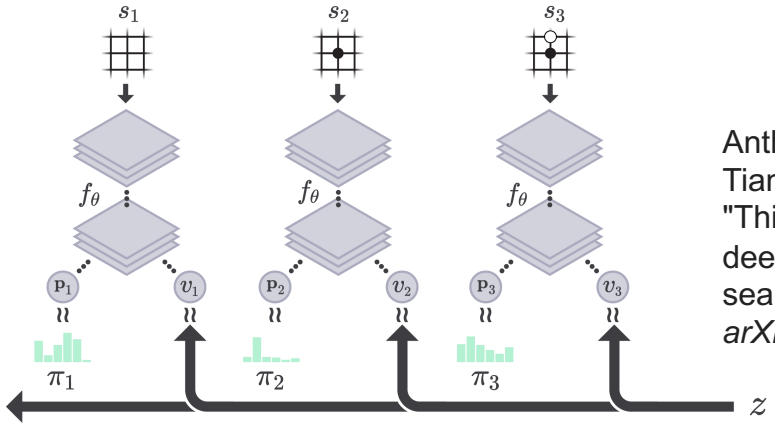


- Add an additional neural network outputing policy ***p*** and value *v* by imitating MCTS

$$(\mathbf{p}, v) = f_\theta(s)$$

Loss:

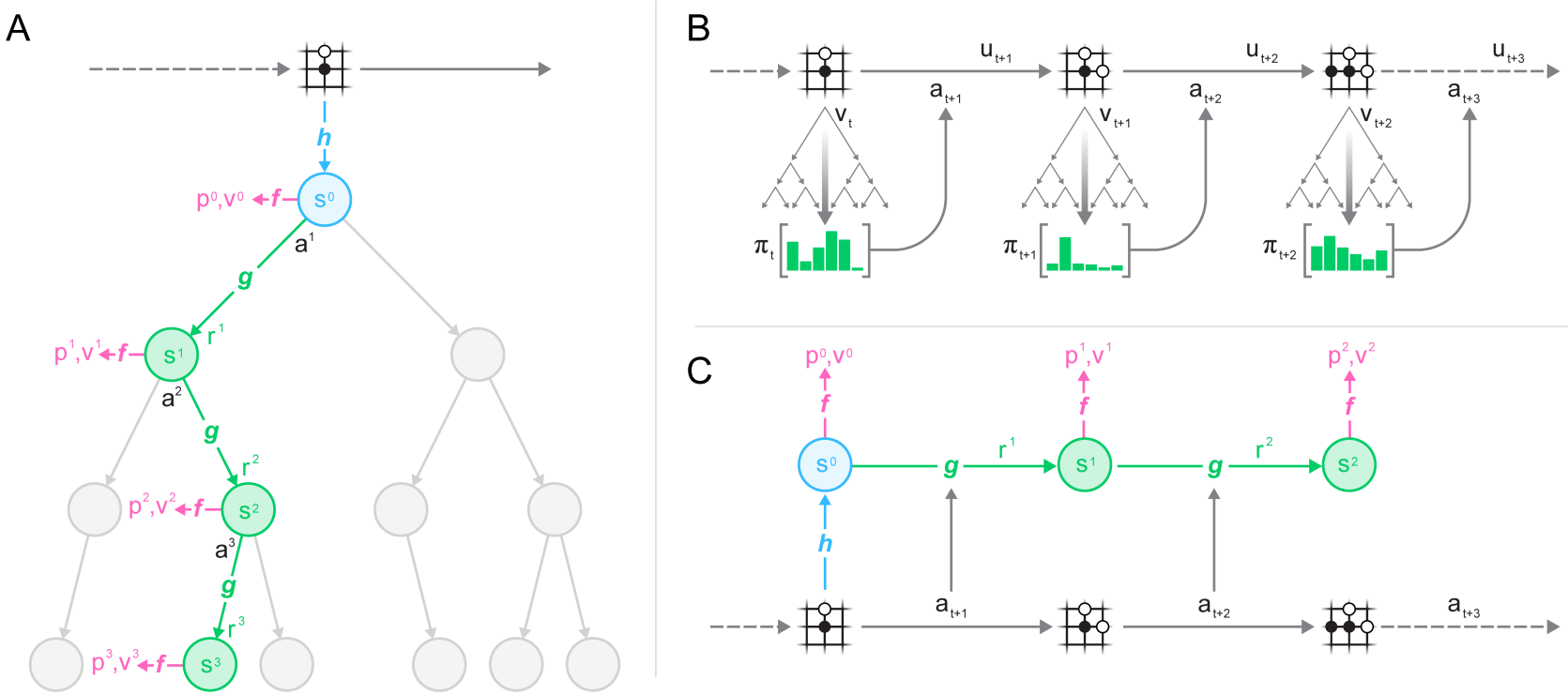
$$l = (z - v)^2 - \boldsymbol{\pi}^\top \log \mathbf{p} + c||\theta||^2$$



Silver, David, et al. "Mastering the game of go without human knowledge." *nature* 550.7676 (2017): 354-359.

Anthony, Thomas, Zheng Tian, and David Barber. "Thinking fast and slow with deep learning and tree search." *arXiv preprint arXiv:1705.08439* (2017).

MuZero: learning the transition



- (A) MCTS with a learned transition, (B) Self-play, and (C) Build up a hidden state s^k to represent the state and base on it to build a transition model (to run MCTS in (A)) and also make prediction about the value and optimise the policy

Schrittwieser, Julian, et al. "Mastering atari, go, chess and shogi by planning with a learned model." *Nature* 588.7839 (2020): 604-609.

References

- Reinforcement learning slides are largely based on
  - Sutton, Richard S., and Andrew G. Barto. *Reinforcement learning: An introduction*. Vol. 1. No. 1. Cambridge: MIT press, 1998.
  - Weinan Zhang's lecture notes at Shanghai Jiaotong University
- Szepesvári, Csaba. "Algorithms for reinforcement learning." *Synthesis lectures on artificial intelligence and machine learning* 4.1 (2010): 1-103.
- Neto, Gonçalo. "From single-agent to multi-agent reinforcement learning: Foundational concepts and methods." *Learning theory course* (2005).

Deep Reinforcement Learning and Game

- Sutton, Richard S., and Andrew G. Barto. Reinforcement learning: An introduction. MIT press, 1998.
- Neumann, L. J., & Morgenstern, O. (1947). Theory of games and economic behavior (Vol. 60). Princeton: Princeton university press.
- 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.
- Silver, David, et al. "Mastering the game of Go with deep neural networks and tree search." Nature 529.7587 (2016): 484-489.
- Russell, Stuart, Peter Norvig, and Artificial Intelligence. "A modern approach." Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs 25 (1995): 27.
- Browne, Cameron B., et al. "A survey of monte carlo tree search methods." Computational Intelligence and AI in Games, IEEE Transactions on 4.1 (2012): 1-43.
- Kocsis, Levente, and Csaba Szepesvári. "Bandit based monte-carlo planning." Machine Learning: ECML 2006. Springer Berlin Heidelberg, 2006. 282-293.
- Coulom, Rémi. "Efficient selectivity and backup operators in Monte-Carlo tree search." Computers and games. Springer Berlin Heidelberg, 2006. 72-83.