Reinforcement Learning

Lyle Ungar

With images by Sutton & Barto and slides by
Heejin Jeong and Steven Chen

RL Definition
- state, action, reward, policy…

RL algorithms
- model-based, model free

Deep RL: AlphaGo/AlphaZero

Thanksgiving week
- no pods
- no class Wed/Fri
- no OH Thurs-Sat

Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm

Starting from random play, and given no domain knowledge except the game rules, AlphaZero achieved within 24 hours a superhuman level of play in the games of chess and shogi (Japanese chess) as well as Go, and convincingly defeated a world-champion program in each case.

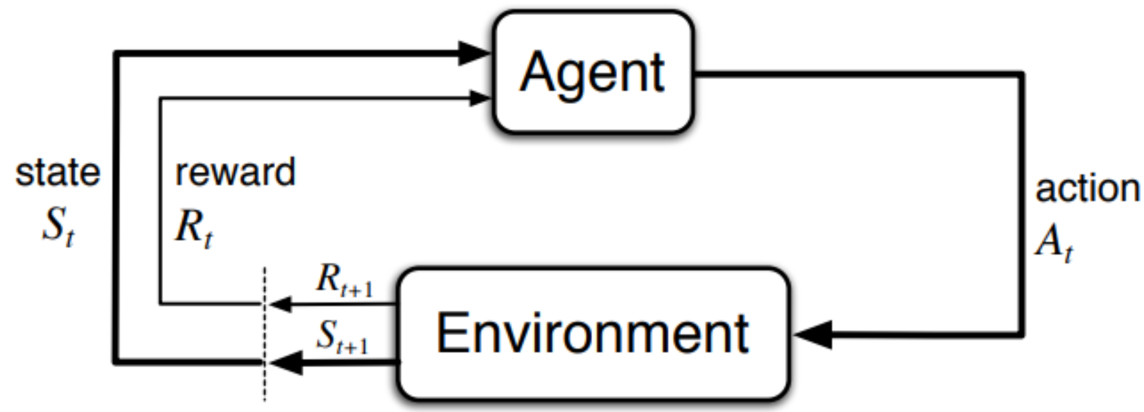
2017: https://arxiv.org/pdf/1712.01815.pdf

Outline, which won't make sense yet

- **What is Reinforcement Learning?**
- **Model-based RL**
  - Markov Decision Process (MDP)
  - Dynamic Programming
- **Model-free RL**
  - TD methods; Q-Learning
  - Exploration-Exploitation Trade-off
  - On- and off-policy learning
  - Monte Carlo Methods
- **Deep RL**
  - AlphaGo, AlphaZero

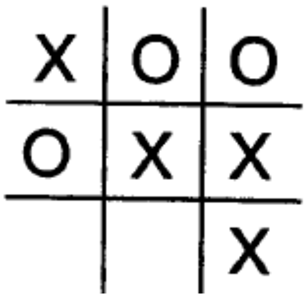
Reinforcement Learning Idea

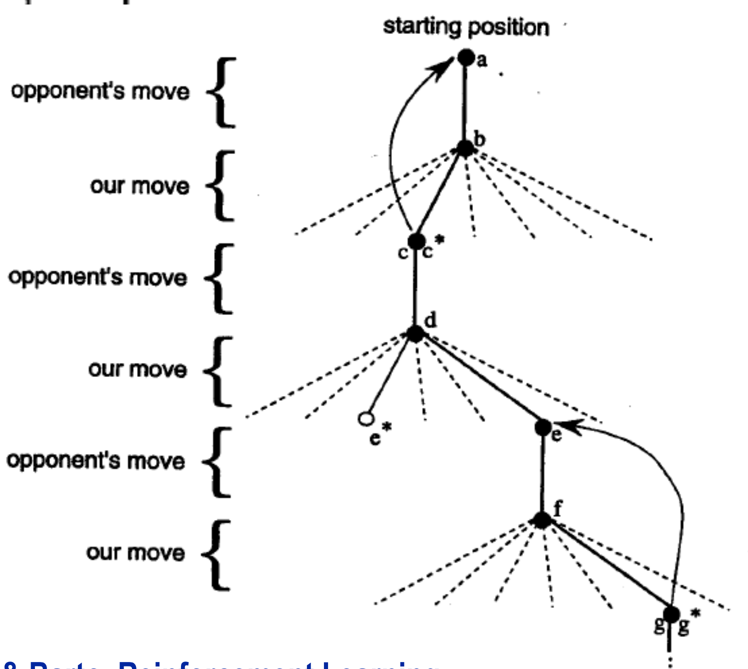
Learn a function (policy) that maximizes an agent's long-term reward in an environment



From Sutton *Reinforcement Learning: An Introduction* (2016 draft)

Tic-Tac-Toe Example





Sutton & Barto, Reinforcement Learning

- **State** ***s***
  - Current board position
- **Action** *a*
  - Move
  - Possible actions depend on state
- **Policy** *p(a|**s**)*
  - Given state, what action to take
- **Reward** *r(**s**,a)*
  - -1/0/1 for lose/tie/win
  - 0 for all intermediate states

- **Exploration policy**
  - Search to find out what happens and how good each state is.
- **Exploitation policy**
  - Use what was learned to do well.

Examples of RL

- Games
- Robotics
- Bidding
- Showing ads
- Optimizing chemical reactions
- Chatbot conversation

Stanford Autonomous Helicopter

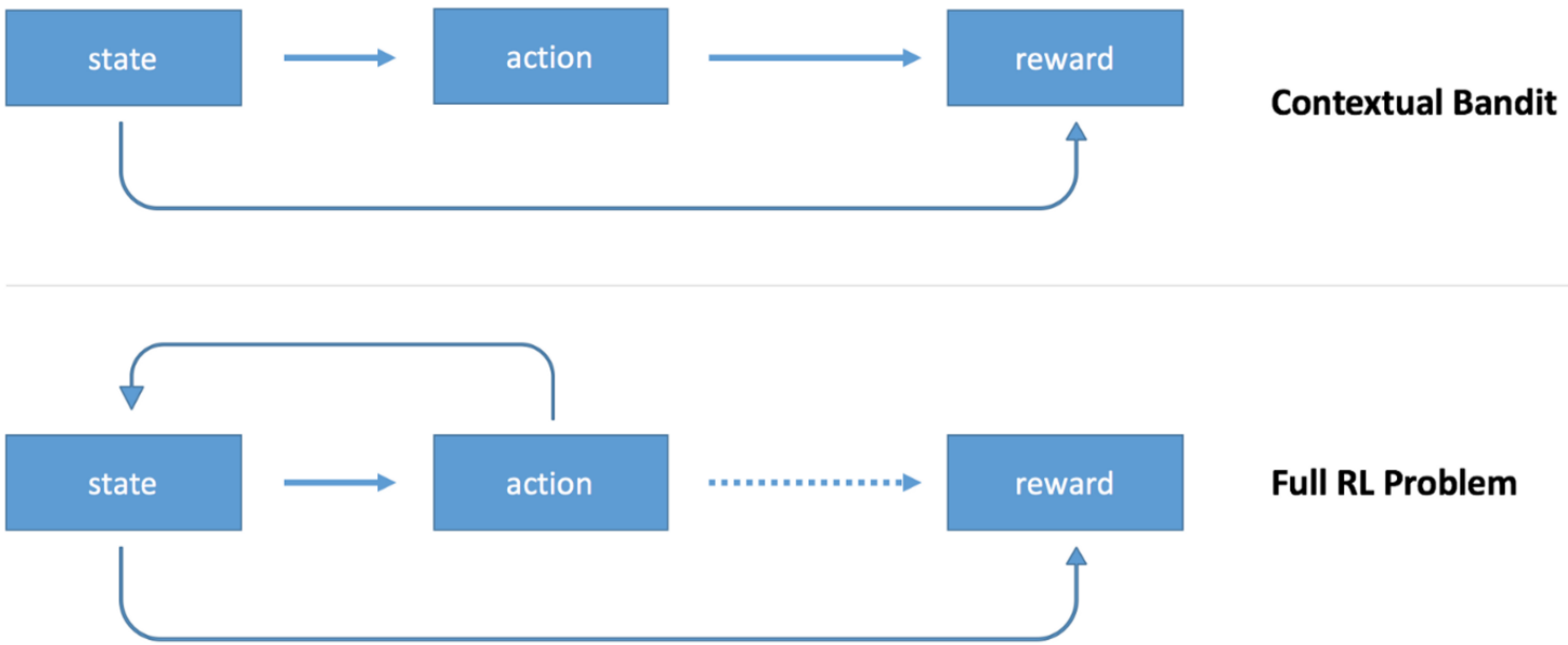
https://www.youtube.com/watch?v=M-QUkgk3HyE

https://towardsdatascience.com/applications-of-reinforcement-learning-in-real-world-1a94955bcd12

RL Challenges

- **Never see the result of actions not taken**
- **Never told what the best action was**
- **Often a long sequence of actions before you discover consequences of the actions**
  - E.g., win or lose game only after moves are complete

Contextual Bandits



Source: https://medium.com/emergent-future/simple-reinforcement-learning-with-tensorflow-part-0-q-learning-with-tables-and-neural-networks-d195264329d0

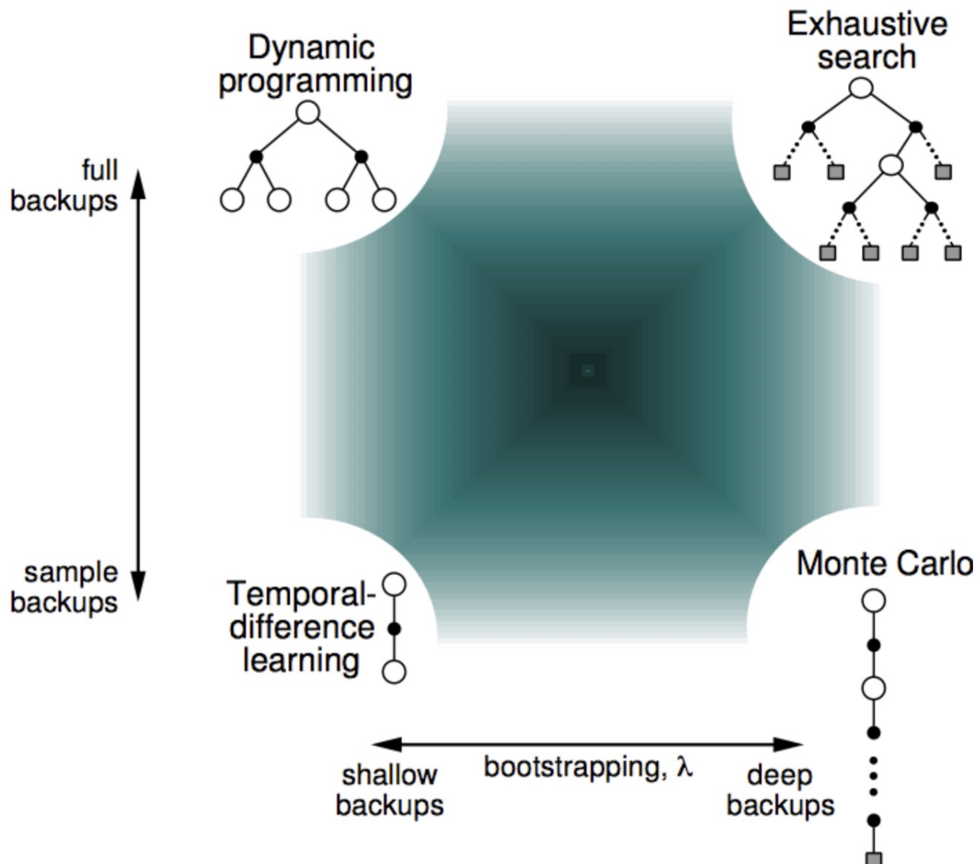
RL Types

- **Model based**
  - Explicitly learn $p(s_{t+1}|s_t, a_t)$, $r(s_t, a_t)$
  - Markov Decision Process (MDP)
  - Then figure out policy
- **Model free**
  - Learn expected value of each state, $V(s_t)$, given a policy
  - Learn expected value of each state and action, $Q(s_t, a_t)$
  - Learn an optimal policy, while learning V or Q
  - State can be discrete or real, V and Q can be neural nets

Overview of RL Strategies

Response to all possible actions

Model-based



Response to one action

Model-free

One-step ahead

Search to end

From David Silver UCL Course on RL: http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html

Temporal Differences: TD(0)

- Learn the value $V_p(s)$ of each state under a policy $p(s)$
- This will allow us to iterate
  - Find a better policy p' given a value function $V_p(s)$
  - Find a more accurate value function $V_{p'}(s)$ for the new policy

TD(0): Mouse in Maze Example

A mouse (or robot) is placed in a maze

- On each trial, starts on a lettered square
- Can move to any adjacent square, except for the maroon one.
- If land in a lettered square, nothing happens.
- If land in Food get fruit loops (+1) and leave maze.
- If land in Shock get a mild shock (-1) and leave maze.
- Initially no knowledge.

A	B	C	Food
D		E	Shock
J	G	H	I

Simplest model-free RL method: "Temporal Difference"

Mouse in Maze (‘gridworld’) Example

Goal: learn optimal policy e.g. by learning value of every square

- Initial values all set to 0
- On each trial, move through maze until exit.
- Update values of squares as you leave them.
- Do many trials to learn values of every square.

TD(0) is model free: update immediately rather than at the end of the ’episode’.

A 0	B 0	C 0	Food 0
D 0		E 0	Shock 0
J 0	G 0	H 0	I 0

Tabular TD(0) for estimating v_π

Input: the policy π to be evaluated
Algorithm parameter: step size $\alpha \in (0, 1]$
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(\textit{terminal}) = 0$

Loop for each episode:
    Initialize S
    Loop for each step of episode:
        $A \leftarrow$ action given by π for S
        Take action A, observe R, S'
        $V(S) \leftarrow V(S) + \alpha\big[R + \gamma V(S') - V(S)\big]$
        $S \leftarrow S'$
    until S is terminal

Source: Reinforcement Learning: An Introduction (Sutton, R., Barto A.)

Mouse in Maze Example

- Update rule for value *V(s)* of square *s* you just left, when entering square *s'*
  - $\Delta V = c\ (\ V(s') - V(s) + R(s)\ )$
  - *V(s)* and *V(s')* are the values of the two squares before the update.
  - After update $V(s) = V(s) + \Delta V$.
  - *R(s)* is the reward you get when leaving square s.
  - The constant *c* is the learning rate.

A 0	B 0	C 0	Food 0
D 0		E 0	Shock 0
J 0	G 0	H 0	I 0

Mouse in Maze Example

Trial 1: A ➔ B ➔ C ➔ Food ➔ Get reward 1 and exit

- Define V(exit) = 0, always.
- $\Delta V = 0.5\,(\,V(s') - V(s) + R(s)\,)$
- New value of A: $V(A) + 0.5\,(\,V(B) - V(A) + R(A)\,) = 0$
- New value of B: $V(B) + 0.5\,(\,V(C) - V(B) + R(B)\,) = 0$
- New value of C: $V(C) + 0.5\,(\,V(Food) - V(C) + R(C)\,) = 0$
- New value of Food: $V(Food) + 0.5\,(V(Exit)-V(Food) + R(Food)) = 0.5$

A 0	B 0	C 0	Food 0
D 0		E 0	Shock 0
J 0	G 0	H 0	I 0

Values before trial

➔

A 0	B 0	C 0	Food 0.5
D 0		E 0	Shock 0
J 0	G 0	H 0	I 0

Values after trial

Mouse in Maze Example

Trial 2: A → B → C → Food → Get reward 1 and exit

- New value of A: V(A) + 0.5 (V(B) - V(A) + R(A)) = 0
- New value of B: V(B) + 0.5 (V(C) - V(B) + R(B)) = 0
- New value of C: V(C) + 0.5 (V(Food) - V(C) + R(C)) = 0.25
- New value of Food: V(Food) + 0.5 (V(Exit)-V(Food) + R(Food)) = 0.75

A 0	B 0	C 0	Food 0.5
D 0		E 0	Shock 0
J 0	G 0	H 0	I 0

Values before trial

→

A 0	B 0	C 0.25	Food 0.75
D 0		E 0	Shock 0
J 0	G 0	H 0	I 0

Values after trial

Mouse in Maze Example

- After many trials learn values

A 0.812	B 0.868	C 0.918	Food 1.00
D 0.762		E 0.660	Shock -1.00
J 0.705	G 0.655	H 0.611	I 0.388

Values after convergence

What is implicit in these values?

A policy

Mouse in Maze Example

- What would the value of A be under an optimal policy with no discounting and deterministic motion?

A 0	B 0	C 0	Food 0
D 0		E 0	Shock 0
J 0	G 0	H 0	I 0

Using TD(0)

- **We showed how to learn the values of states under a policy using a recurrence relationship**

$$V(S) \leftarrow V(S) + \alpha\left[R + \gamma V(S') - V(S)\right]$$

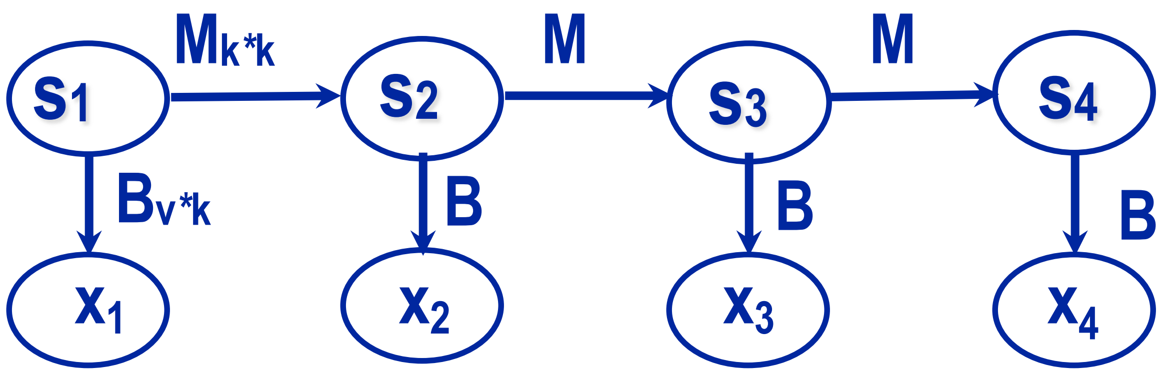
- **We'll later also iterate**
  - Learn the values of states under a policy
  - Improve the policy given those values

Markov Decision Process (MDP)

Model-based RL

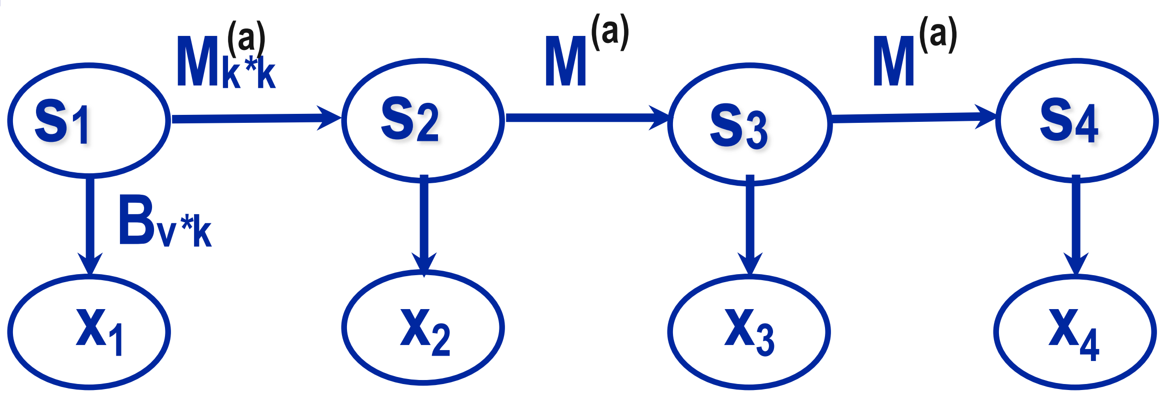
MDPs generalize Markov Models

- **MM**



M = Markov transition matrix
B = emission matrix

- **MDP**

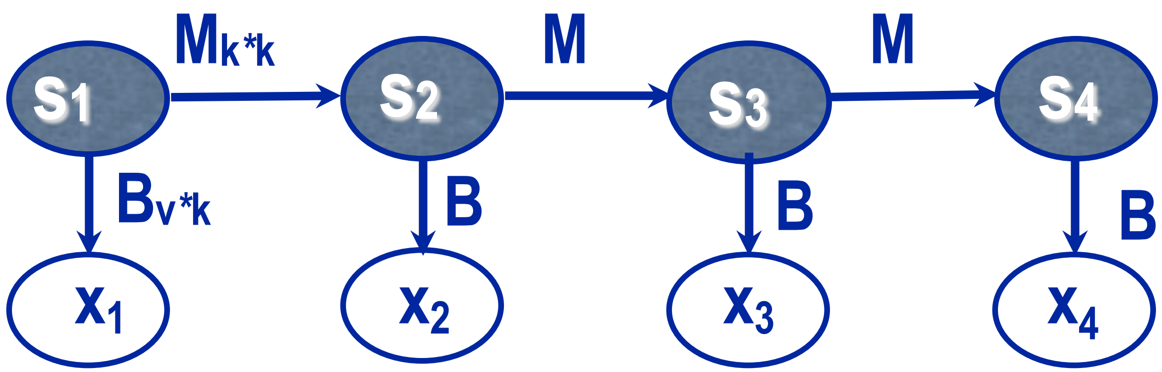


$\mathbf{M}^{(a)}$ Different transition matrix for each action, a

Emission, $\mathbf{x}_t$, includes reward, R_t

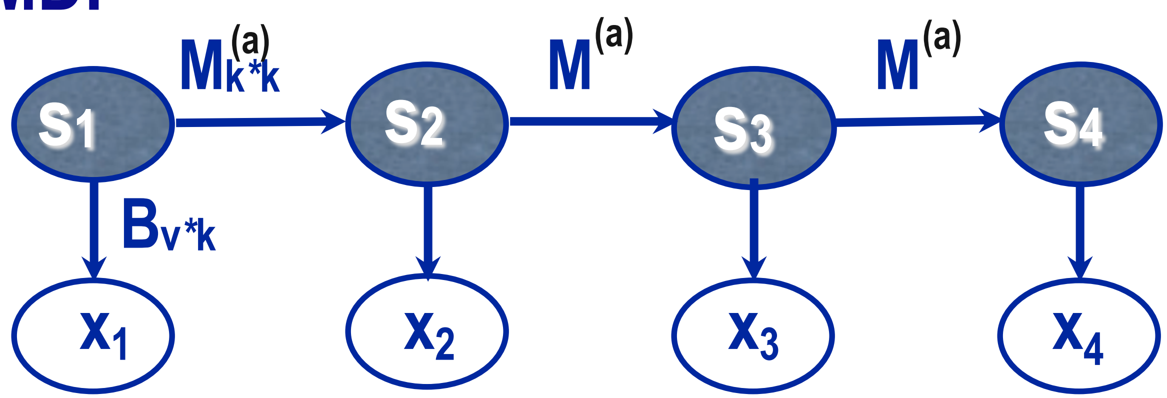
POMDPs generalize HMMs

- **HMM**



M = Markov transition matrix
B = emission matrix

- **POMDP**

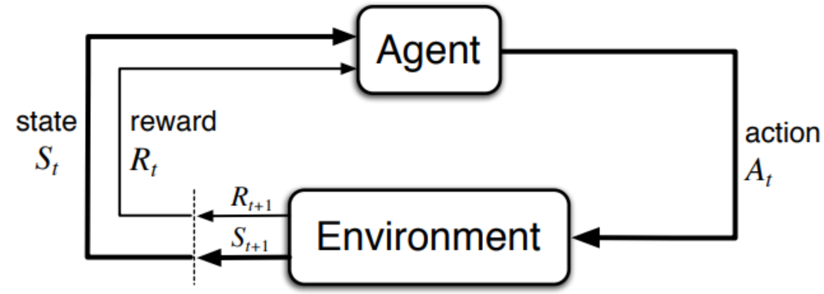


$\mathbf{M}^{(a)}$ Different transition matrix for each action, a

Emission, $\mathbf{x}_t$, includes reward, R_t

MDP Example

- **State:** agent position
- **Action:** up, down, left, right
  - excluding actions that cause collisions
- **Transition:** where you actually move (depends on state and action)
- **Reward:**
  - 0 - have not reached exit
  - 1 - reached good exit
  - -1 - reached bad exit



From Sutton *Reinforcement Learning: An Introduction* (2016 draft)

A 0	B 0	C 0	Food 0
D 0		E 0	Shock 0
J 0	G 0	H 0	I 0

Reward given after exiting

MDP Specification

Joint distribution $p(s', r|s, a) = Pr\{S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a\}$ can be used to specify MDP

Traditional specification of MDP is 5-tuple $(\mathcal{S}, \mathcal{A}(\cdot), p(\cdot|\cdot,\cdot), r(\cdot,\cdot,\cdot), \gamma)$ where

- $\mathcal{S}$ is a finite set of states
- $\mathcal{A}(s)$ is a finite set of actions
- $p(s'|s, a) = Pr(s_{t+1} = s'|S_t = s, A_t = a) = \sum_{r \in \mathcal{R}} p(s', r|s, a)$
- $r(s, a, s') = \mathbb{E}\left[R_{t+1}|S_t = s, A_t = a, S_{t+1} = s'\right] = \frac{\sum_{r \in \mathcal{R}} rp(s', r|s, a)}{p(s'|s, a)}$
- $\gamma \in [0, 1]$ is the discount factor

Goal: Find policy $a_t = \pi(s_t)$ that maximizes long term return

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{k+t+1}$$

Notation summary

- s_t **state**
- $V(s_t)$ **value**
- $a_t = \pi(s_t)$ **action (and policy π)**
- γ **discount factor**
- $r(s_t,a_t,s_{t+1})$ **reward** (usually simply $r(s_{t+1})=R_{t+1}$)
- G_t **expected discounted reward ('return')**
- $p(s_{t+1}|s_t,a_t)$ **model**

MDP generalize to NNets

- $\mathbf{s}_t$ **state** – a vector
- $\mathbf{a}_t$ **action** – a vector
- $V(\mathbf{s}_t)$ **value** – a nonlinear function of $\mathbf{s}_t$
- $p(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t)$ **model –** a nonlinear function of $\mathbf{s}_t$ and $\mathbf{a}_t$
  - Often deterministic: $s_{t+1} = f(s_t,a_t)$

Policy, Value, and Q Values

Policy Specific

- **Policy (could be stochastic)**: $\pi(a|s)$
- **Value**:

$$v_\pi(s) = \mathbb{E}_\pi\left[G_t | S_t = s\right]$$
$$= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s\right]$$

- **Q value**:

$$q_\pi(s,a) = \mathbb{E}_\pi\left[G_t | S_t = s, A_t = a\right]$$
$$= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s A_t = a\right]$$

Optimal

- **Policy (deterministic)**:

$$\pi^*(s) = \operatorname*{argmax}_a q_*(s,a)$$

- **Value**:

$$v_*(s) = \max_\pi v_\pi(s)$$
$$= \max_a q_*(s,a)$$

- **Q value**:

$$q_*(s,a) = \max_\pi q_\pi(s,a)$$

Questions

- What is *V(A)*?
- What is *R(A)*?
- What is *q(A, move to D)*?
- What is $\pi^*(A)$?
- What are possible reasons that *V(A) < 1*?

A 0.812	B 0.868	C 0.918	Food 1.00
D 0.762		E 0.660	Shock -1.00
J 0.705	G 0.655	H 0.611	I 0.388

Bellman's Equation

$$
\begin{aligned}
v_\pi(s) &= \mathbb{E}_\pi\left[G_t | S_t = s\right] \\
&= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s\right] \\
&= \mathbb{E}_\pi\left[R_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k R_{t+k+2} | S_t = s\right] \\
&= \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)\left[r + \gamma \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+2} | S_{t+1} = s'\right]\right] \\
&= \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)\left[r + \gamma v_\pi(s')\right], \forall s \in \mathcal{S}
\end{aligned}
$$

Recurrence relation for Value

Bellman's Equation

Bellman's Equation: Holds for all policies $\pi(a|s)$

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)\left[r + \gamma v_\pi(s')\right], \forall s \in \mathcal{S}$$

$$q_\pi(s,a) = \sum_{s',r} p(s',r|s,a)\left[r + \gamma v_\pi(s')\right], \forall s \in \mathcal{S}, \forall a \in \mathcal{A}(s)$$

Bellman's Optimality Equation: Holds for optimal policies $\pi^*(s)$

$$v_*(s) = \max_{a \in \mathcal{A}(s)} \sum_{s',r} p(s',r|s,a)\left[r + \gamma v_*(s')\right], \forall s \in \mathcal{S}$$

$$q_*(s,a) = \sum_{s',r} p(s',r|s,a)\left[r + \gamma \max_{a'} q_*(s')\right], \forall s \in \mathcal{S}, \forall a \in \mathcal{A}(s)$$

Bellman's Equation

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \left[r + \gamma v_\pi(s')\right], \forall s \in \mathcal{S}$$

TD(0)

$$V(S) \leftarrow V(S) + \alpha\left[R + \gamma V(S') - V(S)\right]$$

Model-based RL:
"Dynamic Programming"

Interleave:
Policy Evaluation: Estimate v_π using Bellman's equation
Policy Improvement: Improve π using v_π

Policy Evaluation

Compute v_π for an arbitrary policy π

Turn *Bellman's Equation* into an update rule to find a fixed point

Randomly initialize initial approximation v_0

$$v_{k+1}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)\left[r + \gamma v_k(s')\right]$$

Bellman's Equation shows that $v_k = v_\pi$ is a fixed point for this update rule

Sequence $\{v_k\} \to v_\pi$ as $k \to \infty$.

Policy Improvement

Greedily update policy $\pi(s) \rightarrow \pi'(s)$

Initialize a random policy π_0

$$\pi'(s) = \underset{a}{\text{argmax}} \sum_{s',r} p(s', r|s, a) \left[r + \gamma v_{\pi}(s')\right]$$

Policy gives a strictly better policy except when original policy is already optimal

Policy Iteration

Policy iteration (using iterative policy evaluation)

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Repeat
   $\quad \Delta \leftarrow 0$
   $\quad$ For each $s \in \mathcal{S}$:
   $\quad\quad v \leftarrow V(s)$
   $\quad\quad V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s))\big[r + \gamma V(s')\big]$
   $\quad\quad \Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   $\quad$ *old-action* $\leftarrow \pi(s)$
   $\quad \pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r|s, a)\big[r + \gamma V(s')\big]$
   $\quad$ If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

Assuming deterministic policy $\pi(s)$

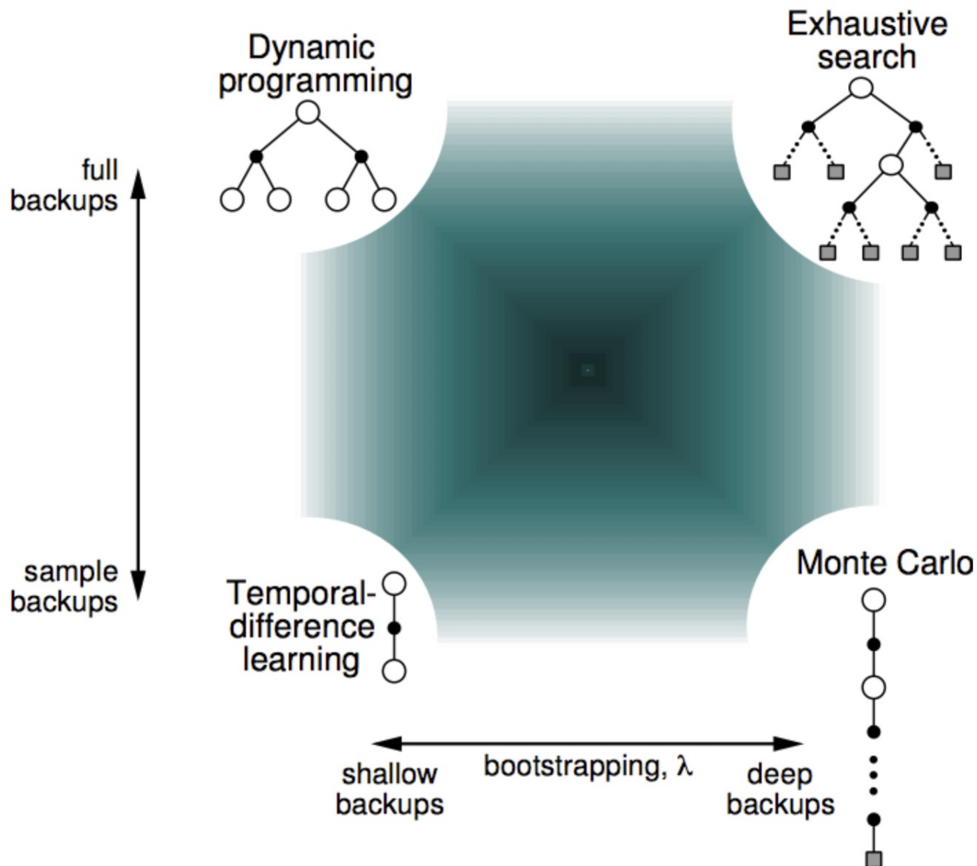
From Sutton *Reinforcement Learning: An Introduction* (2016 draft)

End of part I

Overview of RL Strategies

Response to all possible actions

Response to one possible actions



Model-based

Model-free

One-step ahead

Search to end

From David Silver UCL Course on RL: http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html

Model-free RL

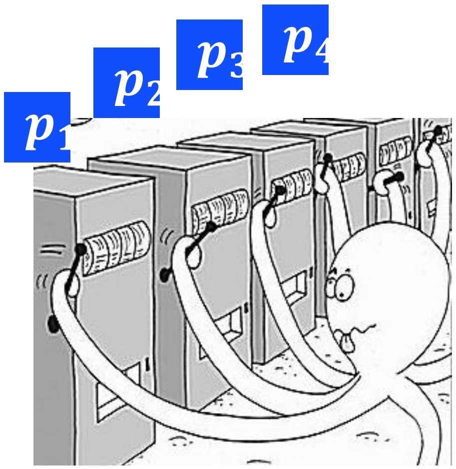
On policy

- SARSA (State-Action-Reward-State-Action)

Off policy

- Q-learning

Exploration-Exploitation Trade-off



Should I select the best arm based on my current knowledge?
Or should I explore other arms?

On- or *off*- policy control

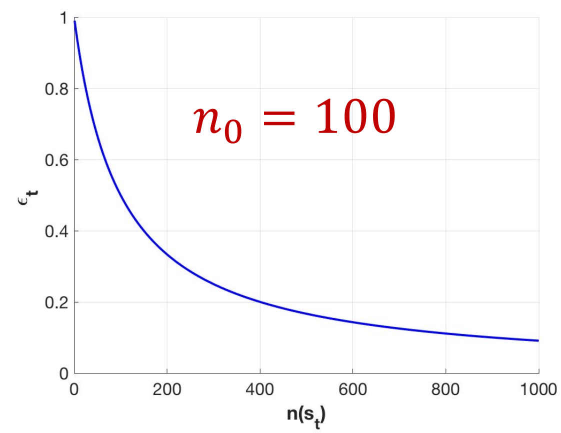
- **Target policy, $\boldsymbol{\pi}$**: policy that we want to update
- **Action (behavior) policy, $\boldsymbol{\mu}$**: policy for choosing an action
- **On-policy Control**
  - Learn policy π using experience sampled from target policy π
  - $(\mu = \pi)$
- **Off-policy Control**
  - Learn policy π using experience sampled from different policy μ
  - $(\mu \neq \pi)$
  - Sometimes: safe exploration or learn by observing others

ϵ-greedy Exploration

- **Continual exploration**
  - With probability ϵ, perform a randomly selected action
  - With probability $1-\epsilon$, perform a greedy action
- **Use ϵ-greedy policy μ with respect to Q_π to improve it**
- **Annealing: time-varying $\epsilon = \epsilon_t$**

$$\epsilon_t = \frac{n_0}{n_0 + visits(s_t)}$$

An annealing schedule



SARSA (State-Action-Reward-State-Action)

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Initialize $Q(s, a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, 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

Source: Introduction to Reinforcement learning by Sutton and Barto —Chapter 6

On-policy, model-free

Q-LEARNING

Temporal Difference (TD) Learning

Off-policy, model-free

Temporal Difference (TD) Prediction

- TD learns from current predictions rather than waiting until termination
- **TD(0): One-step look ahead**
  - $V(s_t) \leftarrow V(s_t) + \alpha(\underbrace{r_t + \gamma V(s_{t+1})}_{\text{TD } \textit{target}} - V(s_t))$

Q-learning: Off-policy TD(0)

- **On experience** $< s_t, a_t, r_t, s_{t+1} >$ with greedy target policy π

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \Big(\underbrace{r_t + \gamma \underline{Q\big(s_{t+1}, \pi(s_{t+1})\big)}}_{\text{TD target}} - Q(s_t, a_t) \Big)$$

α **: Learning rate**

$$= \max_{a' \in A} Q(s_{t+1}, a') = V^{\pi}(s_{t+1})$$

- **Convergence is guaranteed for discrete S, A if:**
  - $\sum_t \alpha_t = \infty,\ \sum_t \alpha_t^2 < \infty\ (\alpha \in (0,1))$
  - All *(s,a)* pairs are visited infinitely often

*Proof in [Watkins & Dayan 1992]

Q-learning : Off-policy TD(0)

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

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

Source: Introduction to Reinforcement learning by Sutton and Barto — Chapter 6

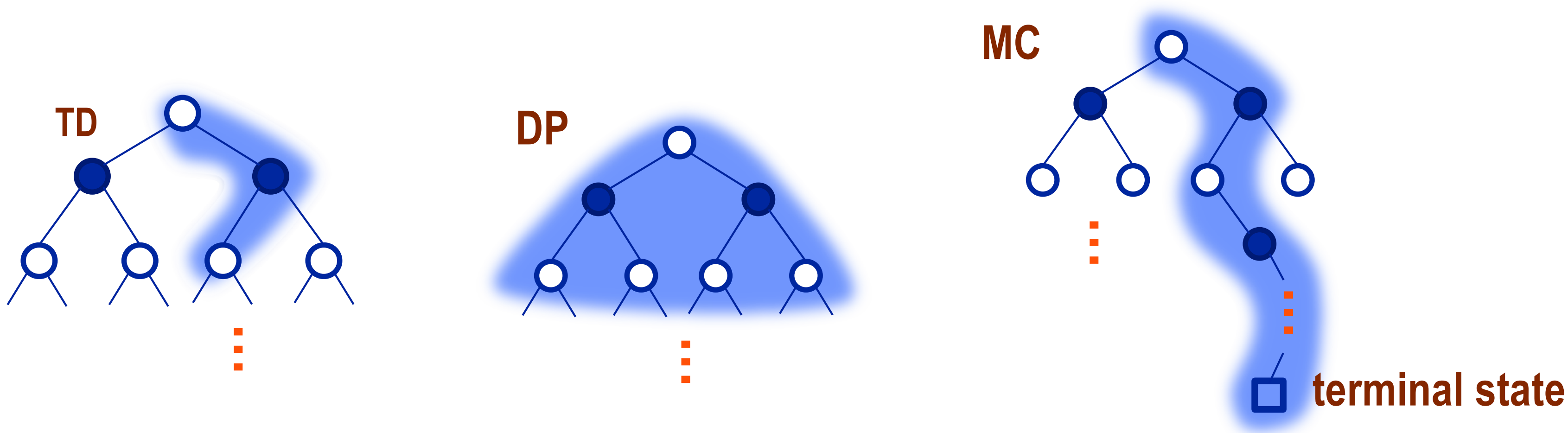
* When your subsequent state $s_{t+1} = S'$ is a terminal state, your ”expected future total reward” is just the immediate reward, $r_t + \gamma r_{t+1} + \cdots$

MONTE CARLO RL

Monte Carlo (MC) Methods in RL

- **Estimate expected reward by sampling**
  - avoids full search
  - defined for episodic tasks

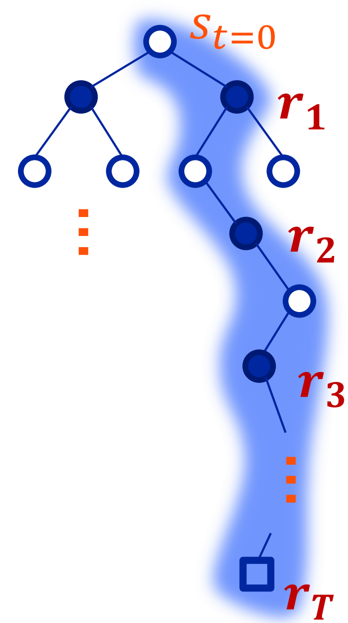
$$\mathbb{E}_\pi\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k+1} | S_t = s\right]$$



Monte Carlo (MC) Prediction

- ***Return,*** $G_t = r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{T-1} r_{t+T}$

  In MC, use empirical mean return starting from s_t or (s_t, a_t) instead of expected return for $V^\pi(s_t)$ or $Q^\pi(s, a)$

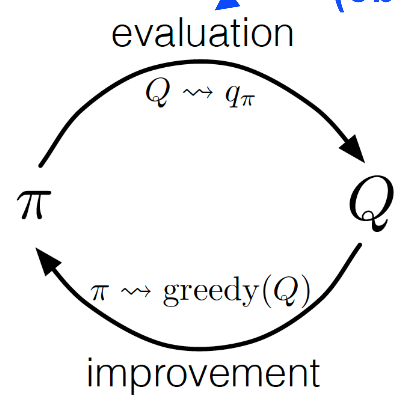


- $V^\pi(s)$ = average of the returns following all the visits to s in a set of episodes

- $Q^\pi(s, a)$ = average of the returns following all the visits to (s, a) in a set of episodes

Monte Carlo Updates

Sample an episode following the current action policy (obtain a return, G_t, of the episode)



Update the action value with the average of $[G_1, G_2, \cdots, G_N]$

*Image from Sutton *Reinforcement Learning: An Introduction* (2016 draft)

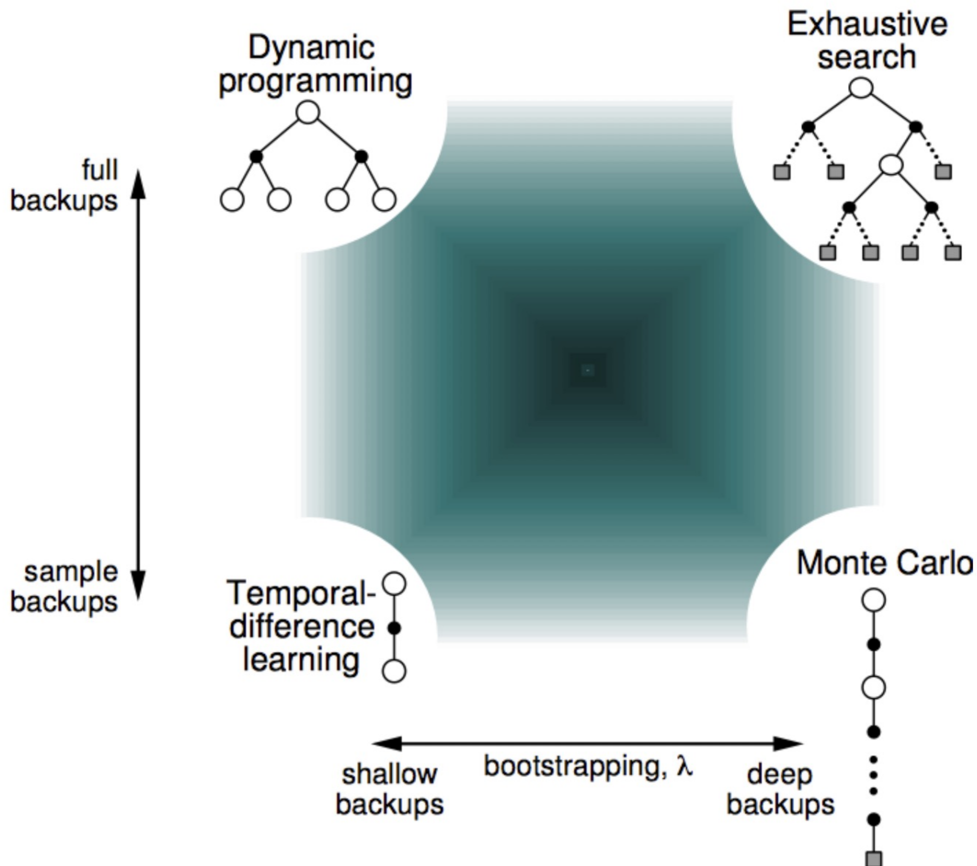
Monte Carlo vs. Q-learning

- **MC: High Variance, Low Bias**
  - Less sensitive to initial Q values
- **Q-learning (TD): Low Variance, High Bias**
  - Online learning is possible. We wait only one time step!
  - For applications with **long episodes**: delaying all learning until an episode's end is too slow
  - Needed for **non-episodic** (continuing) tasks
  - In practice, **TD methods converge faster than constant α MC methods** on stochastic tasks

Summary

Response to all possible actions

Response to one possible actions



Model-based

Model-free

One-step ahead

Search to end

From David Silver UCL Course on RL: http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html

What you should know

- S, A, V, Q, R, γ, G
- MDP, POMDP
- Exploration/exploitation
- Model based vs. model free RL
- On policy / off policy
- Q-learning (TD)
- Search (e.g. for games)
  - Shallow vs deep; Complete vs. Monte Carlo