Machine learning

Reinforcement Learning

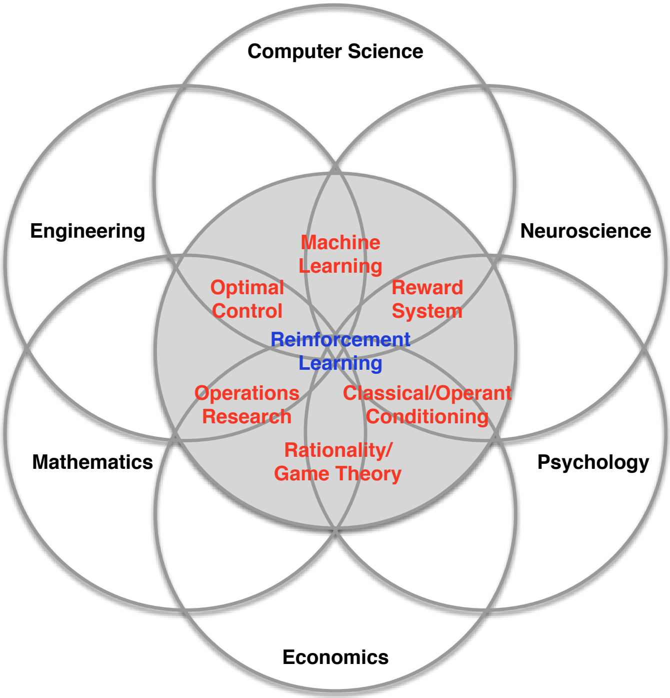
Hamid Beigy

Sharif University of Technology

May 20, 2023

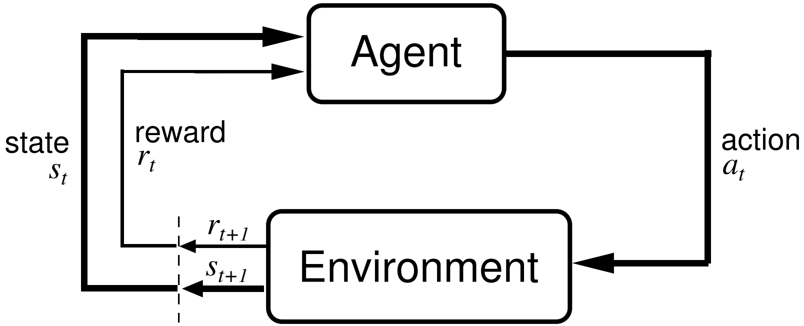
Table of contents

Introduction


Computer Science
Engineering
Neuroscience
Machine Learning
Optimal Control
Reward System
Reinforcement Learning
Operations Research
Classical/Operant Conditioning
Mathematics
Rationality/ Game Theory
Psychology
Economics

- Reinforcement learning is what to do (how to map situations to actions) so as to maximize a scalar reward/reinforcement signal
- The learner is not told which actions to take as in supervised learning, but discover which actions yield the most reward by trying them.
- The trial-and-error and delayed reward are the two most important feature of reinforcement learning.
- Reinforcement learning is defined not by characterizing learning algorithms, but by characterizing a learning problem.
- Any algorithm that is well suited for solving the given problem, we consider to be a reinforcement learning.
- One of the challenges that arises in reinforcement learning and other kinds of learning is tradeoff between exploration and exploitation.

- A key feature of reinforcement learning is that it explicitly considers the whole problem of a goal-directed agent interacting with an uncertain environment.



- Experience is a sequence of observations, actions, rewards.

$$o_1, r_1, a_1, \ldots, a_{t-1}, o_t, r_t$$

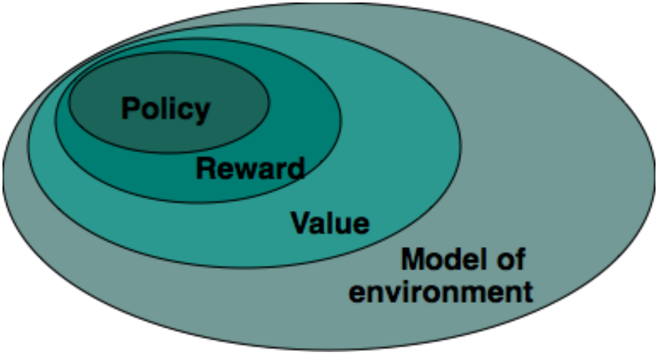
- The state is a summary of experience

$$s_t = f(o_1, r_1, a_1, \ldots, a_{t-1}, o_t, r_t)$$

- In a fully observed environment

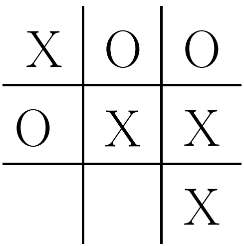
$$s_t = f(o_t)$$

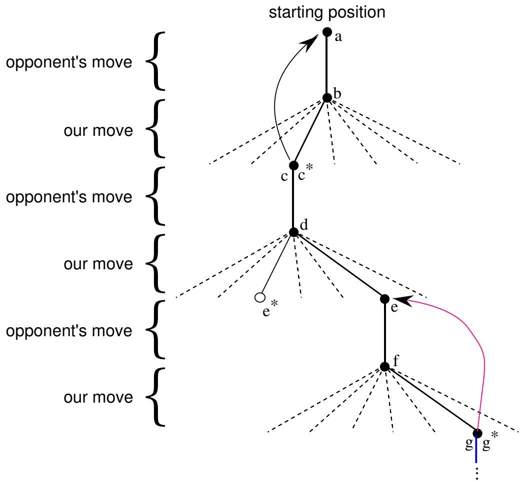
- Policy : A policy is a mapping from received states of the environment to actions to be taken (what to do?).
- Reward function: It defines the goal of RL problem. It maps each state-action pair to a single number called reinforcement signal, indicating the goodness of the action. (what is good?)
- Value : It specifies what is good in the long run. (what is good because it predicts reward?)
- Model of the environment (optional): This is something that mimics the behavior of the environment. (what follows what?)



An example : Tic-Tac-Toe

- Consider a two-playes game (Tic-Tac-Toe)

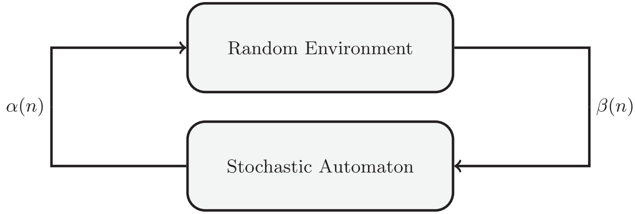




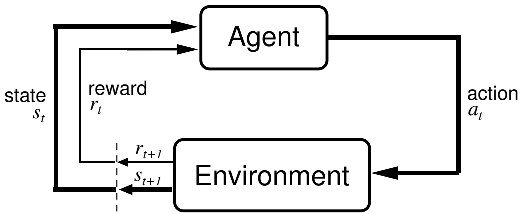
- Consider the following updating

$$V(s) \leftarrow V(s) + \alpha[V(s') - V(s)]$$

- Non-associative reinforcement learning : The learning method that does not involve learning to act in more than one state.



- Associative reinforcement learning : The learning method that involves learning to act in more than one state.



Non-associative reinforcement learning

- Consider that you are faced repeatedly with a choice among n different options or actions.
- After each choice, you receive a numerical reward chosen from a stationary probability distribution that depends on the action you selected.
- Your objective is to maximize the expected total reward over some time period.
- This is the original form of the $n-$armed bandit problem called a slot machine.

- Consider some simple methods for estimating the values of actions and then using the estimates to select actions.
- Let the true value of action a denoted as $Q^*(a)$ and its estimated value at t^{th} play as $Q_t(a)$.
- The true value of an action is the mean reward when that action is selected.
- One natural way to estimate this is by averaging the rewards actually received when the action was selected.
- In other words, if at the t^{th} play action a has been chosen k_a times prior to t, yielding rewards $r_1, r_2, \ldots, r_{k_a}$, then its value is estimated to be

$$Q_t(a) = \frac{r_1 + r_2 + \ldots + r_{k_a}}{k_a}$$

- Greedy action selection : This strategy selects the action with highest estimated action value.

$$a_t = \underset{a}{\operatorname{argmax}}\ Q_t(a)$$

- $\epsilon-$greedy action selection : This strategy selects the action with highest estimated action value most of time but with small probability ϵ selects an action at random, uniformly, independently of the action-value estimates.
- Softmax action selection : This strategy selects actions using the action probabilities as a graded function of estimated value.

$$p_t(a) = \frac{\exp^{Q_t(a)/\tau}}{\sum_b \exp^{Q_t(b)/\tau}}$$

- Environment represented by a tuple $< \underline{\alpha}, \underline{\beta}, \underline{C} >$,
  1. $\underline{\alpha} = \{\alpha_1, \alpha_2, \ldots, \alpha_r\}$ shows a set of inputs,
  2. $\underline{\beta} = \{0, 1\}$ represents the set of values that the reinforcement signal can take,
  3. $\underline{C} = \{c_1, c_2, \ldots, c_r\}$ is the set of penalty probabilities, where $c_i = Prob[\beta(k) = 1 | \alpha(k) = \alpha_i]$.
- A variable structure learning automaton is represented by triple $< \beta, \alpha, T >$,
  1. $\beta = \{0, 1\}$ is a set of inputs,
  2. $\alpha = \{\alpha_1, \alpha_2, \ldots, \alpha_r\}$ is a set of actions,
  3. T is a learning algorithm used to modify action probability vector $\underline{p}$.

- In linear reward-ϵpenalty algorithm ($L_{R-\epsilon P}$) updating rule for p is defined as

$$p_j(k+1) = \begin{cases} p_j(k) + a \times [1 - p_j(k)] & \text{if} \quad i = j \\ p_j(k) - a \times p_j(k) & \text{if} \quad i \neq j \end{cases}$$

when $\beta(k) = 0$ and

$$p_j(k+1) = \begin{cases} p_j(k) \times (1 - b) & \text{if} \quad i = j \\ \frac{b}{r-1} + p_j(k)(1 - b) & \text{if} \quad i \neq j \end{cases}$$

when $\beta(k) = 1$.

- Parameters $0 < b \ll a < 1$ represent step lengths.
- When $a = b$, we call it linear reward penalty(L_{R-P}) algorithm.
- When $b = 0$, we call it linear reward inaction(L_{R-I}) algorithm.

- In stationary environments, average penalty received by automaton is

$$M(k) = E[\beta(k)|p(k)] = Prob[\beta(k) = 1|p(k)] = \sum_{i=1}^{r} c_i p_i(k).$$

- A learning automaton is called expedient if

$$\lim_{k \to \infty} E[M(k)] < M(0)$$

- A learning automaton is called optimal if

$$\lim_{k \to \infty} E[M(k)] = \min_i c_i$$

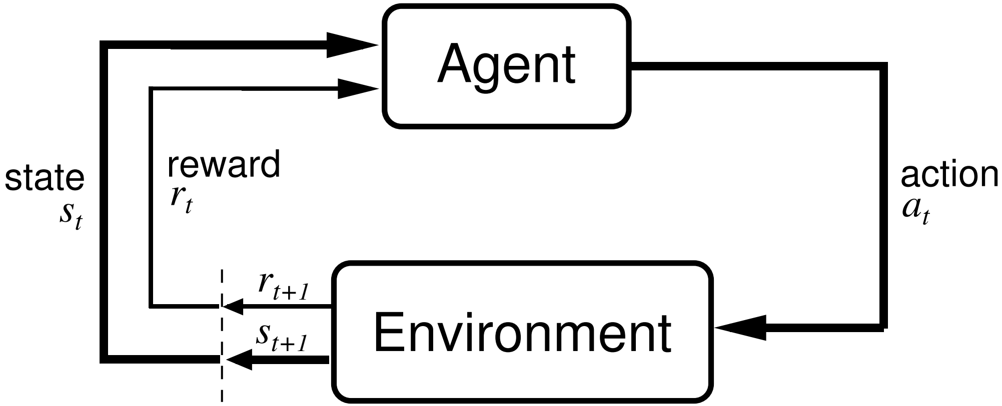
- A learning automaton is called $\epsilon-$optimal if

$$\lim_{k \to \infty} E[M(k)] < \min_i c_i + \epsilon$$

  for arbitrary $\epsilon > 0$

Associative reinforcement learning

The learning method that involves learning to act in more than one state.



Goals,rewards, and returns

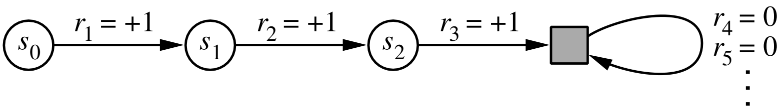
- In reinforcement learning, the goal of the agent is formalized in terms of a special reward signal passing from the environment to the agent.
- The agent's goal is to maximize the total amount of reward it receives. This means maximizing not immediate reward, but cumulative reward in the long run.
- How might the goal be formally defined?
- In episodic tasks the return, R_t, is defined as

$$R_t = r_1 + r_2 + \ldots + r_T$$

- In continuous tasks the return, R_t, is defined as

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

- The unified approach

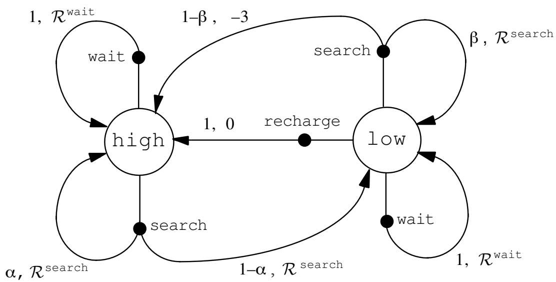


Markov decision process

- A RL task satisfing the Markov property is called a Markov decision process (MDP).
- If the state and action spaces are finite, then it is called a finite MDP.
- A particular finite MDP is defined by its state and action sets and by the one-step dynamics of the environment.

$$
\begin{aligned}
P^a_{ss'} &= Prob\{s_{t+1} = s' | s_t = s, a_t = a\} \\
\mathcal{R}^a_{ss'} &= E[r_{t+1} | s_t = s, a_t = a, s_{t+1} = s']
\end{aligned}
$$

- Recycling Robot MDP



- Let in state s action a is selected with probability of $\pi(s, a)$.
- Value of state s under a policy π is the expected return when starting in s and following π thereafter.

$$\begin{aligned} V^\pi(s) &= E_\pi\{R_t | s_t = s\} = E_\pi\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \middle| s_t = s\right\} \\ &= \sum_{\pi} \pi(s, a) \sum_{s'} P_{ss'}^a \left[\mathcal{R}_{ss'}^a + \gamma V^\pi(s')\right]. \end{aligned}$$

- Value of action a in state s under a policy π is the expected return when starting in s taking action a and following π thereafter.

$$Q^\pi(s, a) = E_\pi\{R_t | s_t = s, a_t = a\} = E_\pi\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \middle| s_t = s, a_t = a\right\}$$

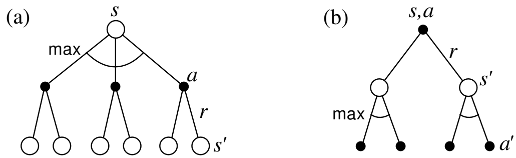
- Policy π is better than or equal of π' iff for all s $V^{\pi}(s) \geq V^{\pi'}(s)$.
- There is always at least one policy that is better than or equal to all other policies. This is an optimal policy.
- Value of state s under the optimal policy ($V^*(s)$) equals

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

- Value of action a in state s under the optimal policy ($Q^*(s,a)$ equals

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a)$$

- Backup diagram for V^* and Q^*



1. Model-based RL
   1.1 Build a model of the environment.
   1.2 Plan (e.g. by lookahead) using model.
2. Value-based RL
   2.1 Estimate the optimal value function $Q^*(s, a)$
   2.2 This is the maximum value achievable under any policy
3. Policy-based RL
   3.1 Search directly for the optimal policy π^*.
   3.2 This is the policy achieving maximum future reward.

Model based methods

- The key idea of DP is the use of value functions to organize and structure the search for good policies.
- We can easily obtain optimal policies once we have found the optimal value functions, or , which satisfy the Bellman optimality equations:

$$\begin{aligned} V^*(s) &= \max_a E\{r_{t+1} + \gamma V^*(s_{t+1}) | s_t = s, a_t = a\} \\ &= \max_a \sum_{s'} P^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V^*(s')\right]. \end{aligned}$$

- Value of action a in state s under a policy π is the expected return when starting in s taking action a and following π thereafter.

$$\begin{aligned} Q^*(s,a) &= E\{r_{t+1} + \gamma \max_{a'} Q^*(s_{t+1}, a') | s_t = s, a_t = a\} \\ &= \sum_{s'} P^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma \max_{a'} Q^*(s', a')\right]. \end{aligned}$$

- Policy iteration is an iterative process

$$\pi_0 \xrightarrow{E} V^{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} V^{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \ldots\ldots \xrightarrow{I} \pi^* \xrightarrow{E} V^*$$

- Policy iteration has two phases : policy evaluation and improvement.
- In policy evaluation, we compute state or state-action value functions

$$\begin{aligned} V^\pi(s) &= E_\pi\{R_t | s_t = s\} = E_\pi\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \middle| s_t = s\right\} \\ &= \sum_{\pi} \pi(s,a) \sum_{s'} P^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V^\pi(s')\right]. \end{aligned}$$

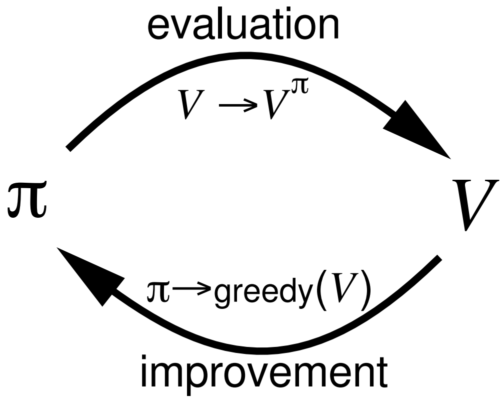
- In policy improvement, we change the policy to obtain a better policy

$$\begin{aligned} \pi'(s) &= \underset{a}{\operatorname{argmax}}\ Q^\pi(s,a) \\ &= \underset{a}{\operatorname{argmax}} \sum_{s'} P^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V^\pi(s')\right]. \end{aligned}$$

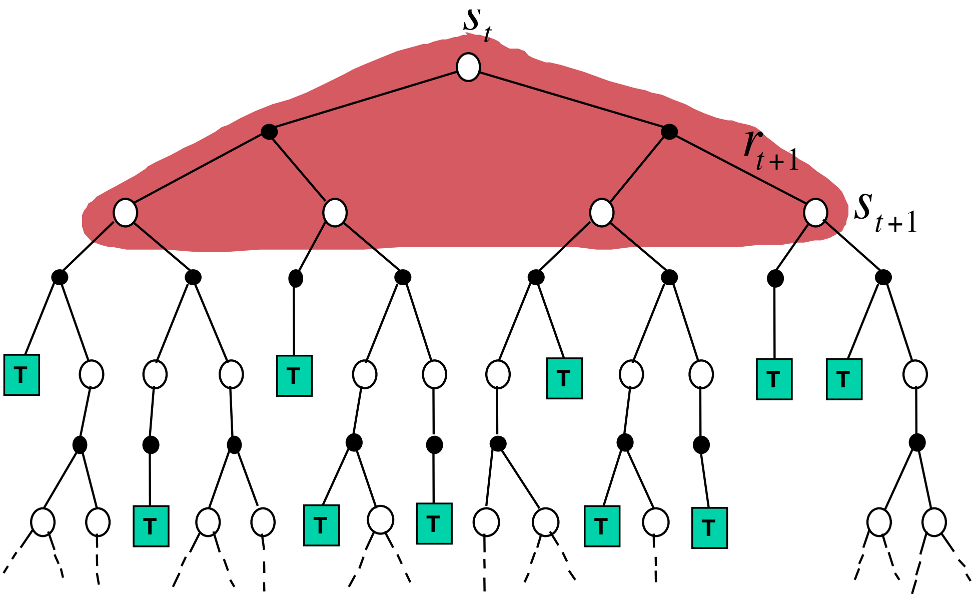
- In value iteration we have

$$\begin{aligned} V_{k+1}(s) &= \max_a E\{r_{t+1} + \gamma V_k(s_{t+1}) | s_t = s, a_t = a\} \\ &= \max_a \sum_{s'} P^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V^(_k s') \right]. \end{aligned}$$

- Generalized policy iteration



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



Value-based methods

- These methods lean policy function implicitly.
- These methods first learn a value function $Q(s, a)$.
- Then infer policy $\pi(s, a)$ from $Q(s, a)$.
- Examples
  - Monte-carlo methods
  - Q-learning
  - SARSA
  - TD(λ)

Monte Carlo methods

- MC methods learn directly from episodes of experience.
- MC is model-free: no knowledge of MDP transitions / rewards
- MC learns from complete episodes
- MC uses the simplest possible idea: value = mean return
- Goal: learn V_π from episodes of experience under policy π

$$S_1 \xrightarrow[R_1]{\alpha_1} S_2 \xrightarrow[R_2]{\alpha_2} S_3 \xrightarrow[R_3]{\alpha_3} S_4 \ldots \xrightarrow[R_{k-1}]{\alpha_{k-1}} S_k$$

- The return is the total discounted reward:

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

- The value function is the expected return:

$$V_\pi(s) = E_\pi[G_t | S_t = s]$$

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

- To evaluate state s
- The first 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

$$V(s) = \frac{S(s)}{N(s)}$$

- By law of large numbers,

$$V(s) \rightarrow v_\pi(s)$$

as

$$N(s) \rightarrow \infty$$

Every-Visit Monte-Carlo Policy Evaluation

- To evaluate state s
- 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

$$V(s) = \frac{S(s)}{N(s)}$$

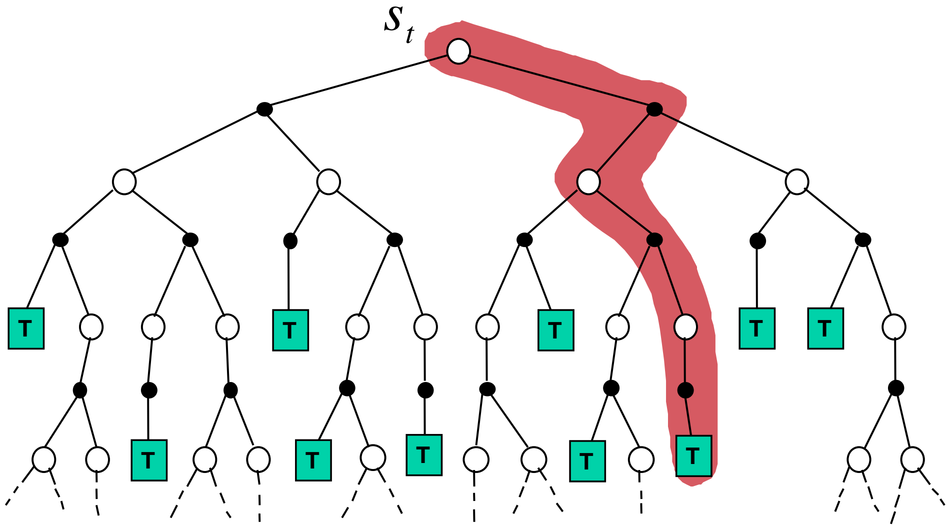
- By law of large numbers,

$$V(s) \rightarrow v_\pi(s)$$

as

$$N(s) \rightarrow \infty$$

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



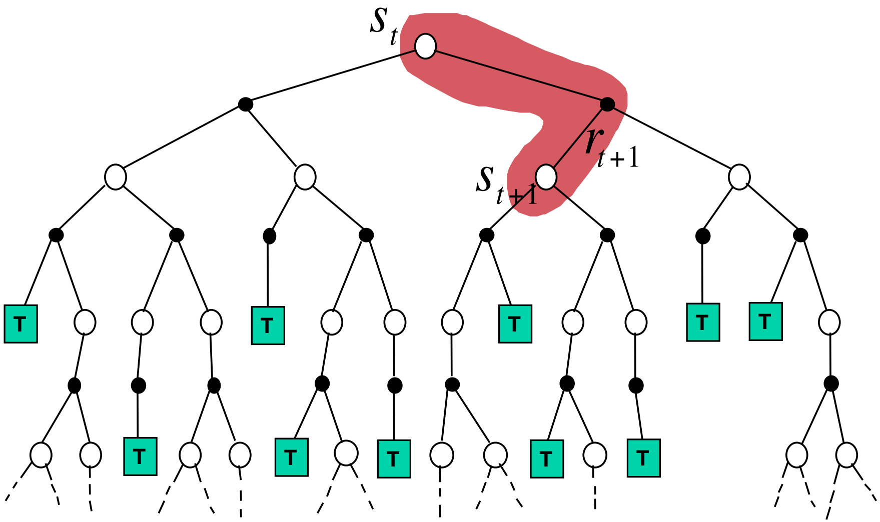
Value-based methods

Temporal-difference methods

- TD learning is a combination of Monte Carlo ideas and dynamic programming (DP) ideas.
- Like Monte Carlo methods, TD methods can learn directly from raw experience without a model of the environment's dynamics.
- Like DP, TD methods update estimates based in part on other learned estimates, without waiting for a final outcome (they bootstrap).
- Monte Carlo methods wait until the return following the visit is known, then use that return as a target for $V(s_t)$ while TD methods need wait only until the next time step.
- The simplest TD method, known as TD(0), is

$$V(s_t) \leftarrow V(s_t) + \alpha \left[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)\right]$$

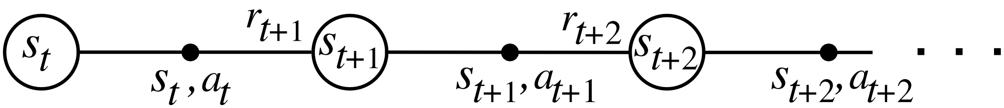
$$V(s_t) \leftarrow V(s_t) + \alpha \left[r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \right]$$



- Algorithm for TD(0)

```
Initialize V(s) arbitrarily, π to the policy to be evaluated
Repeat (for each episode):
.    Initialize s
.    Repeat (for each step of episode):
.    .    a ← action given by π for s
.    .    Take action a; observe reward, r, and next state, s'
.    .    V(s) ← V(s) + α [r + γV(s') − V(s)]
.    .    s ← s'
.    until s is terminal
```

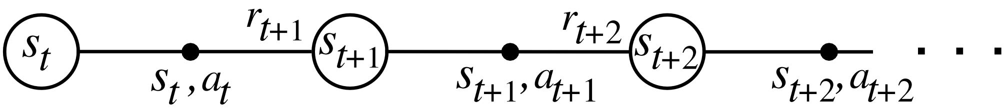
- An episode consists of an alternating sequence of states and state-action pairs:



- SARSA, which is an on policy, updates values using

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right]$$

- An episode consists of an alternating sequence of states and state-action pairs:



- Q-learning, which is an off policy, updates values using

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right]$$

Policy-based methods

- In policy-based learning, there is no value function.
- The policy $\pi(s, a)$ is parametrized by vector θ ($\pi(s, a; \theta)$).
- Explicitly learn policy $\pi(s, a; \theta)$ that implicitly maximize reward over all policies.
- Given policy $\pi(s, a; \theta)$ with parameters θ, find best θ.
- How do we measure the quality of a policy $\pi(s, a; \theta)$?
- Let objective function be $J(\theta)$.
- Find policy parameters θ that maximize $J(\theta)$.
- Sample algorithm: REINFORCE

Policy-based methods versus value-based methods

- Advantages of policy-based methods over value-based methods
  - Usually, computing Q-values is harder than picking optimal actions
  - Better convergence properties
  - Effective in high dimensional or continuous action spaces
  - Can benefit from demonstrations
  - Policy subspace can be chosen according to the task
  - Exploration can be directly controlled
  - Can learn stochastic policies
- Disadvantages of policy-based methods over value-based methods
  - Typically converge to a local optimum rather than a global optimum
  - Evaluating a policy is typically data inefficient and high variance

Reading

1. Chapters 1-6 of Reinforcement Learning: An Introduction (Sutton and Barto 2018).

References i

Sutton, Richard S. and Andrew G. Barto (2018). *Reinforcement Learning: An Introduction*. Second edition. The MIT Press.

Questions?