Reinforcement Learning Tutorial

Bill Podlaski

isiCNI2018
10 Jan 2018

Outline

1. **Intro: what is reinforcement learning (RL)?**
    - Markov Decision Processes (MDPs)
    - Gridworld
2. **Dynamic Programming (DP) methods**
    - Policy evaluation and policy iteration
3. **Monte Carlo (MC) methods**
4. **Temporal Difference (TD) methods**
    - SARSA and Q-learning
5. **N-step and lambda returns**
6. **Value function approximation (VFA)**
    - Mountain Car example with linear approximation

Acknowledgements

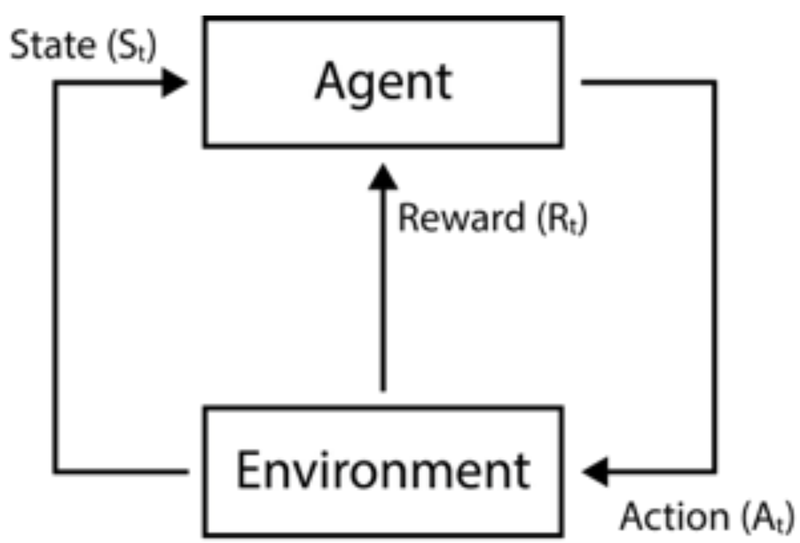
David Silver

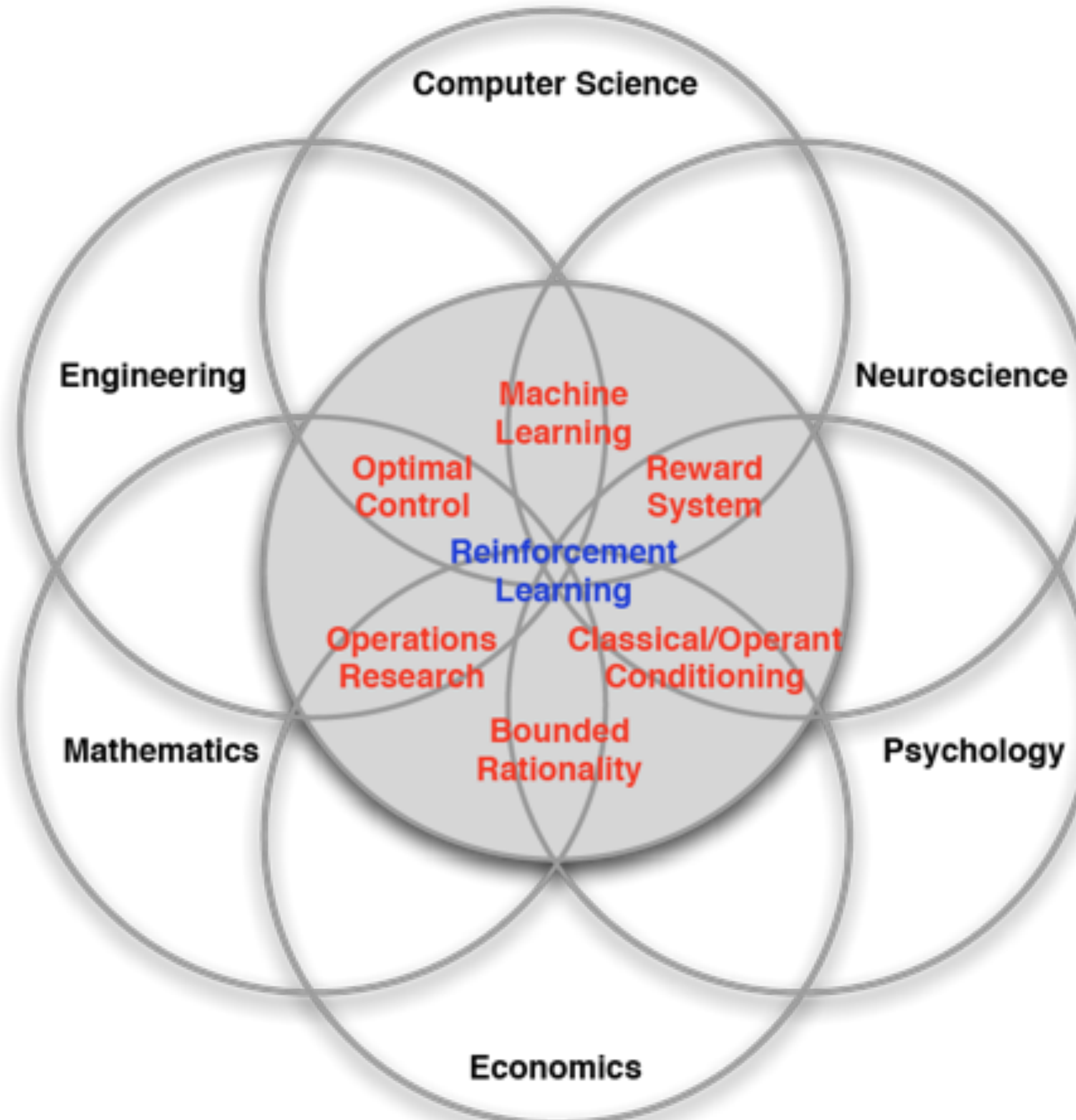
http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html

Sutton & Barto

What is reinforcement learning?

- Agent-oriented learning — learning by interacting with environment to achieve a goal



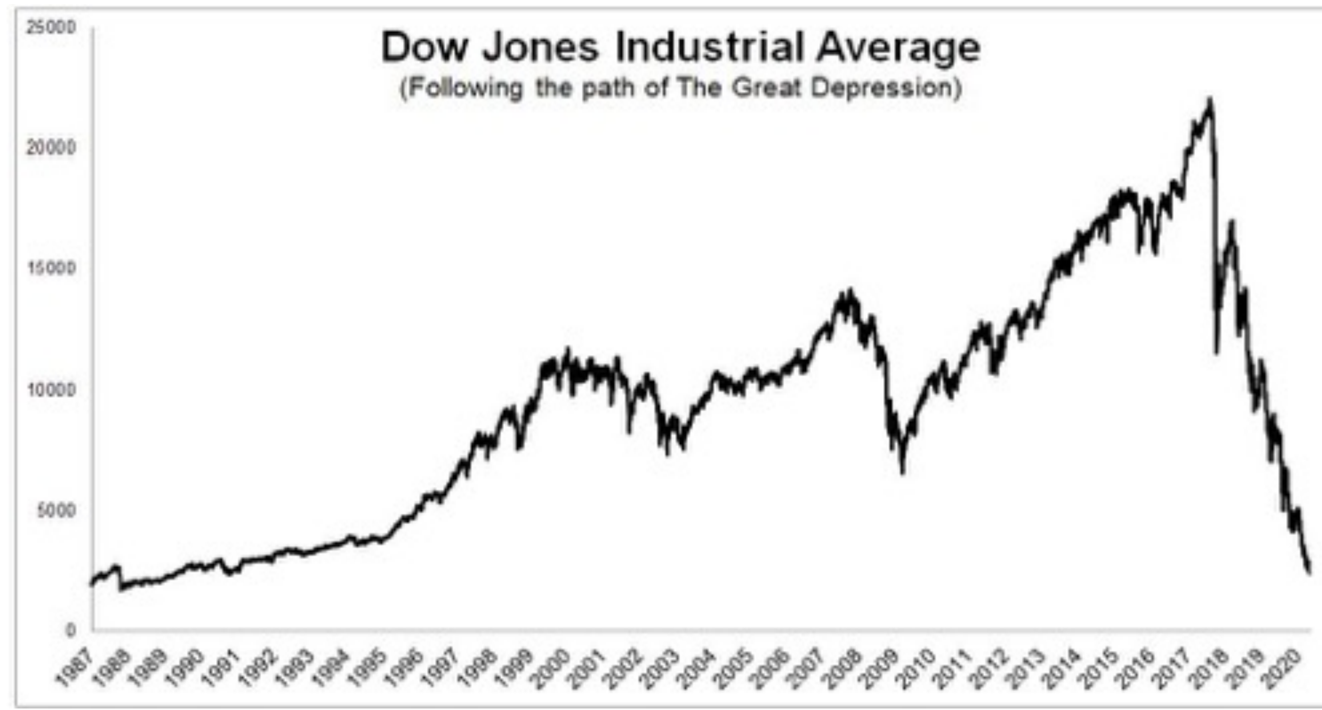


David Silver, 2015

- No supervisor, only reward (scalar readout)
- Feedback is not instantaneous
- **Reward hypothesis:** all goals can be described by the maximization of expected cumulative reward

Examples of reinforcement learning

https://www.centraltelegraph.com.au



https://www.marketwatch.com

https://www.gettyimages.com

https://apkmos.com

Markov Decision Processes (MDPs)

- A Markov Decision Process is an environment with rewards (R) and actions (A), in which sequences of states (S) have the **Markov property**, i.e., "the future is independent of the past given the present": $\Pr[s_{t+1}|s_1, ..., s_t] = \Pr[s_{t+1}|s_t]$

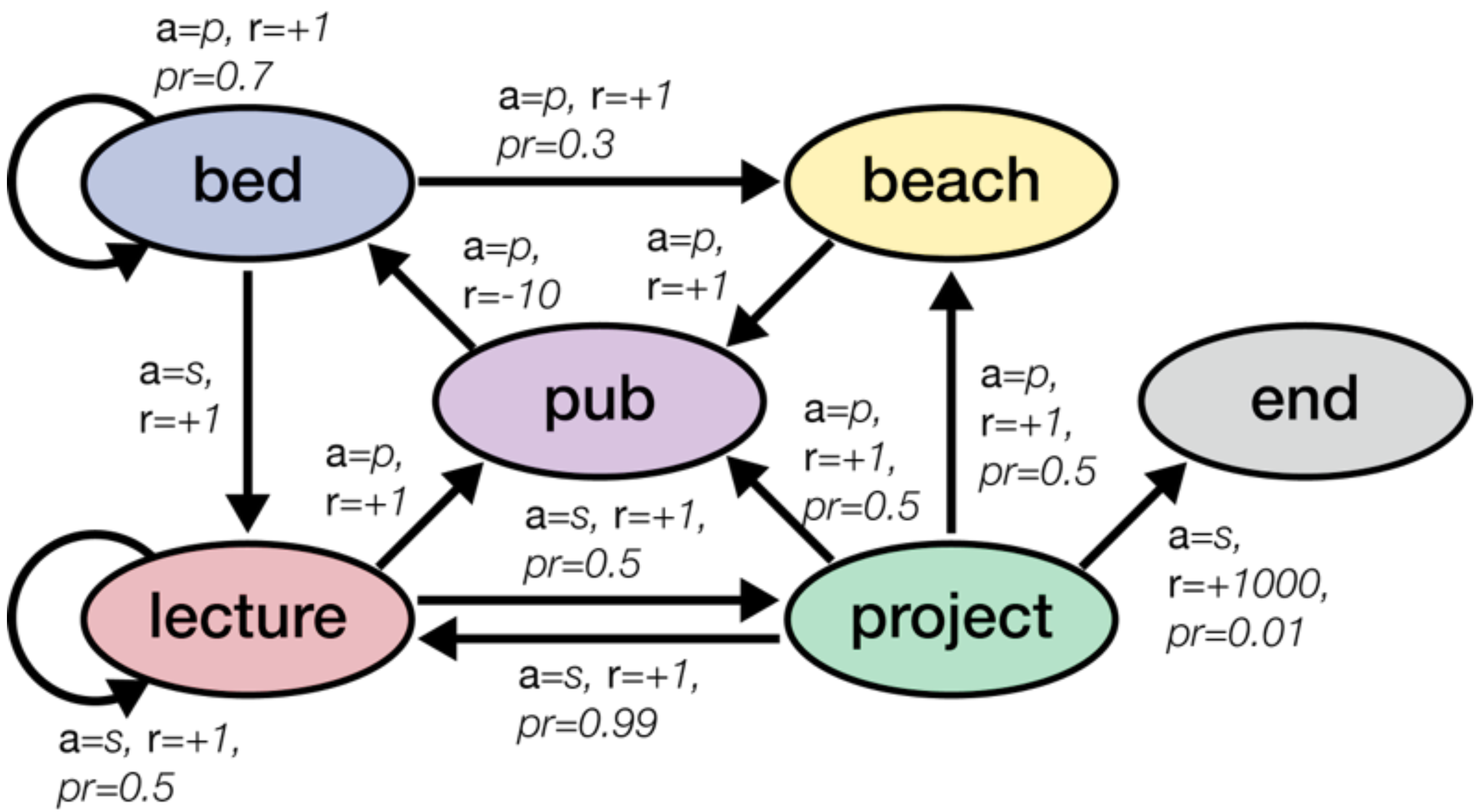
- MDPs are defined by the following variables: $\{\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma\}$
  - $\mathcal{S}$ is a finite set of states
  - $\mathcal{A}$ is a finite set of actions
  - $\mathcal{P}$ defines state transition probabilities (model)

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

  - $\mathcal{R}$ is a reward function: $\mathcal{R}^a_s = \mathbb{E}[R_{t+1}|S_t = s, A_t = a]$
  - $\gamma \in [0, 1]$ is a discount factor (immediate vs future reward)

reward upon leaving state s taking action a

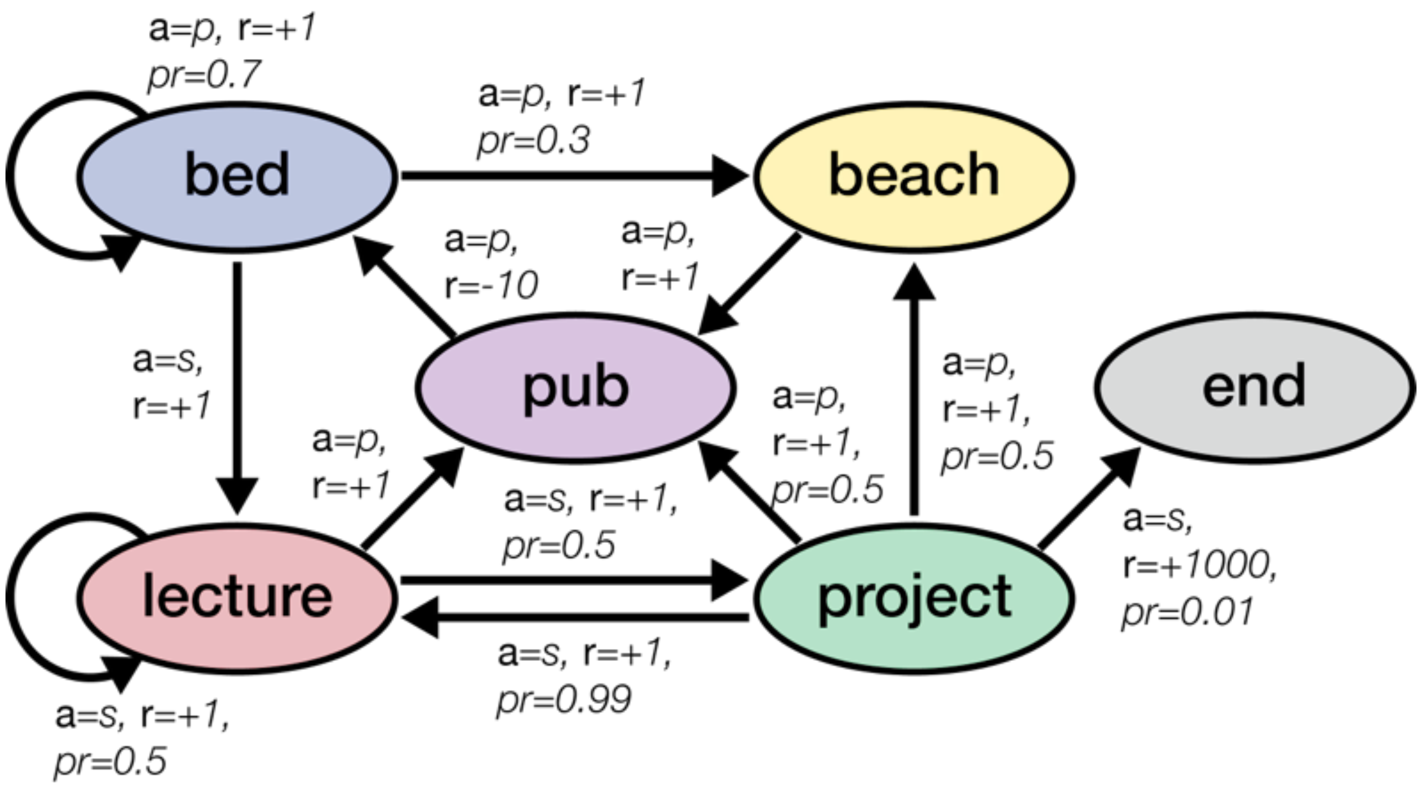
Imbizo student MDP



States = {lecture, bed, pub, beach, project, end}

Actions = {study (s), party (p)}

Imbizo student MDP



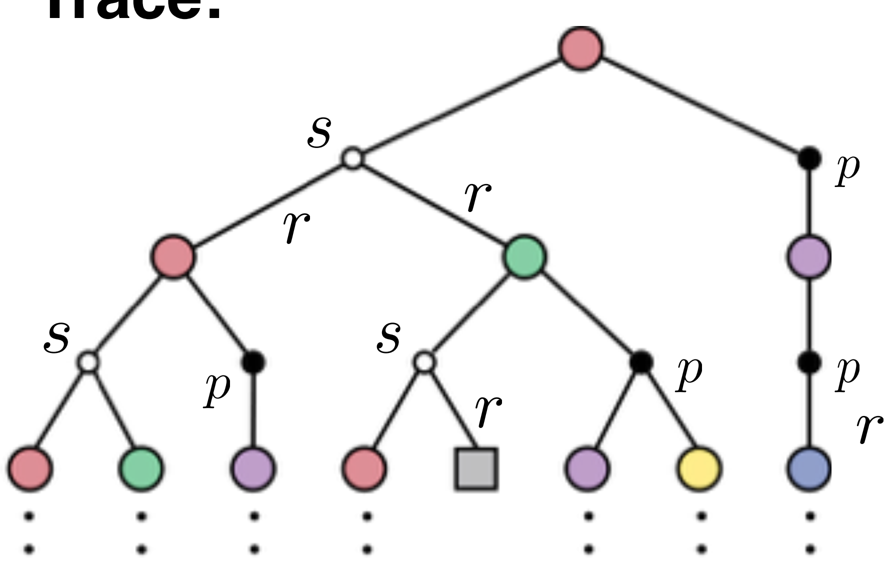
	lecture	bed	pub	beach	project	end
lecture	0.5				0.5	
bed	1					
pub						
beach						
project	.99					.01
end						

a=study

	lecture	bed	pub	beach	project	end
lecture			1			
bed		0.7		0.3		
pub		1				
beach			1			
project			0.5	0.5		
end						

a=party

Trace:



Example episodes:

$$\tau_1 = [\{Lecture, s, +1\}, \{Project, s, +1000\}, \{End\}]$$

$$\tau_2 = [\{Lecture, p, +1\}, \{Pub, p, -10\}, \{Bed, s, +1\}, \{Lecture, s, +1\}, \{Project, s, +1000\}, \{End\}]$$

$$\tau_3 = [\{Lecture, s, +1\}, \{Lecture, s, +1\}, \{Project, s, +1\}, \{Lecture, s, +1\}, \{Project, s, +1000\}, \{End\}]$$

Gains, policies and values

- Agents can learn about the expected rewards from different states (**values**) based on what they gained from individual episodes

- The Gain (or return) G_t of an episode is the total discounted reward from step t:

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

why discount?

- Based on the values of different states, agents can learn which actions lead to more reward (**policy**).

- A policy (π) is:

a distribution over actions given states **OR** a deterministic function of state

$$\pi[a|s] = \Pr[A_t = a | S_t = s] \qquad \pi(s) = a$$

Gains, policies and values

- The state-value function v(s) is the expected return starting from state s and following policy π:

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

- The state-action value q(s,a) is the expected return starting from state s and taking action a, then following policy π:

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

- Example policies based on value functions:

greedy: $$\pi(s) = \arg\max_{a \in \mathcal{A}} q_\pi(s,a)$$

ϵ-greedy: $$\pi[a|s] = \begin{cases} \epsilon/m + 1 - \epsilon & \text{if } a = \arg\max_{a \in \mathcal{A}} q(s,a) \\ \epsilon/m & \text{otherwise} \end{cases}$$

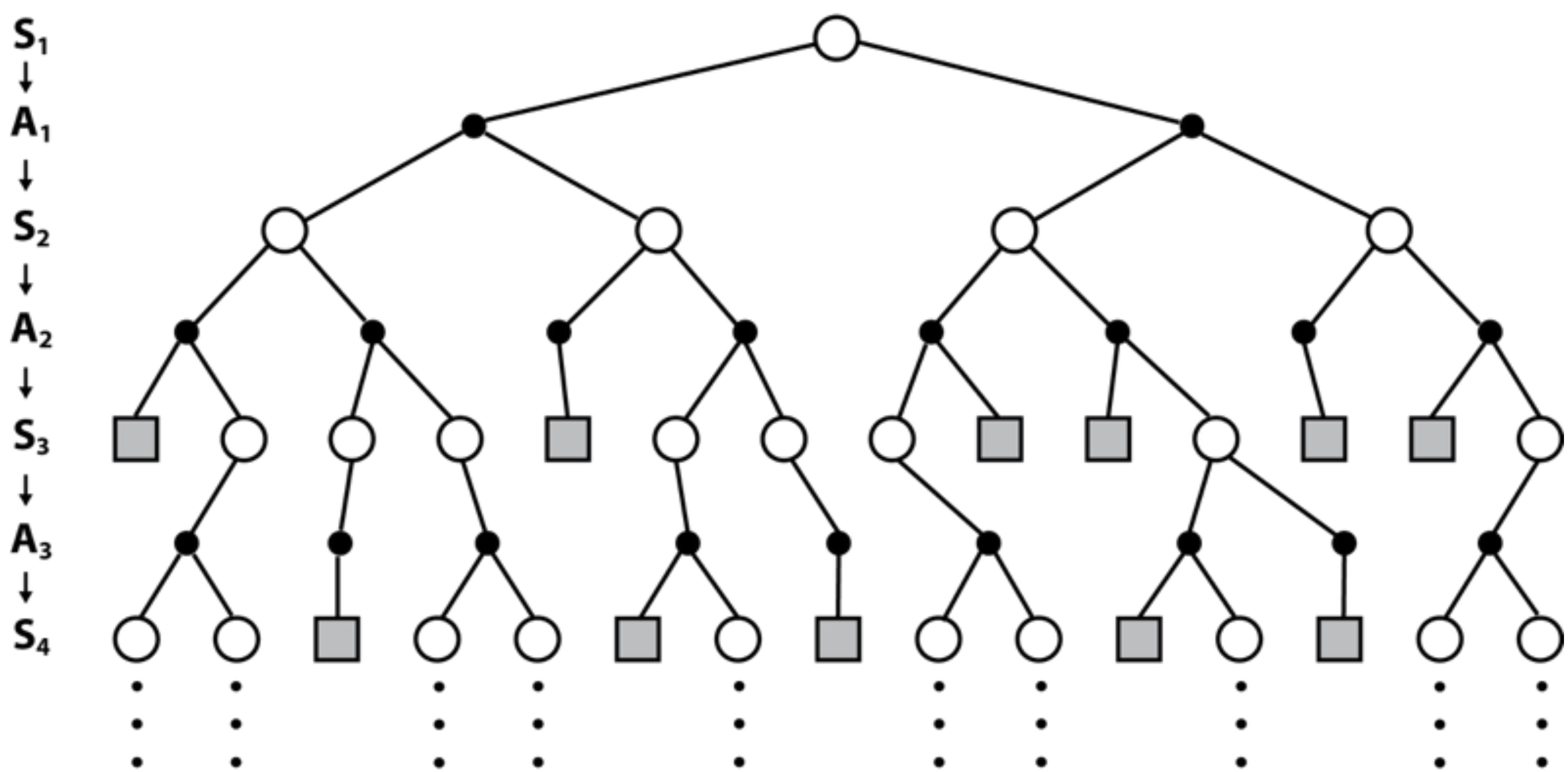
Goals of reinforcement learning for Markov Decision Processes (MDPs)

- **1) Prediction:** learn the value of each state (under a particular policy)
    - Given an MDP $\{\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma\}$, and policy π, what is the corresponding state-value function $v_\pi(s)$? (or $q_\pi(s, a)$)
- **2) Control:** optimize the policy to maximize reward
    - Given an MDP $\{\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma\}$, what is the optimal policy? (and optimal value function)
- How should the agent act in practice? — Exploration vs. exploitation

How to evaluate decision traces of MDPs

1) **Brute force** — calculate value by performing a weighted sum over all possible traces

$$v_\pi(s) = \mathbb{E}[G_t | S_t = s] = \sum_{\tau \in \mathrm{MDP}} p(\tau | \pi, S_t = s) G(\tau)$$



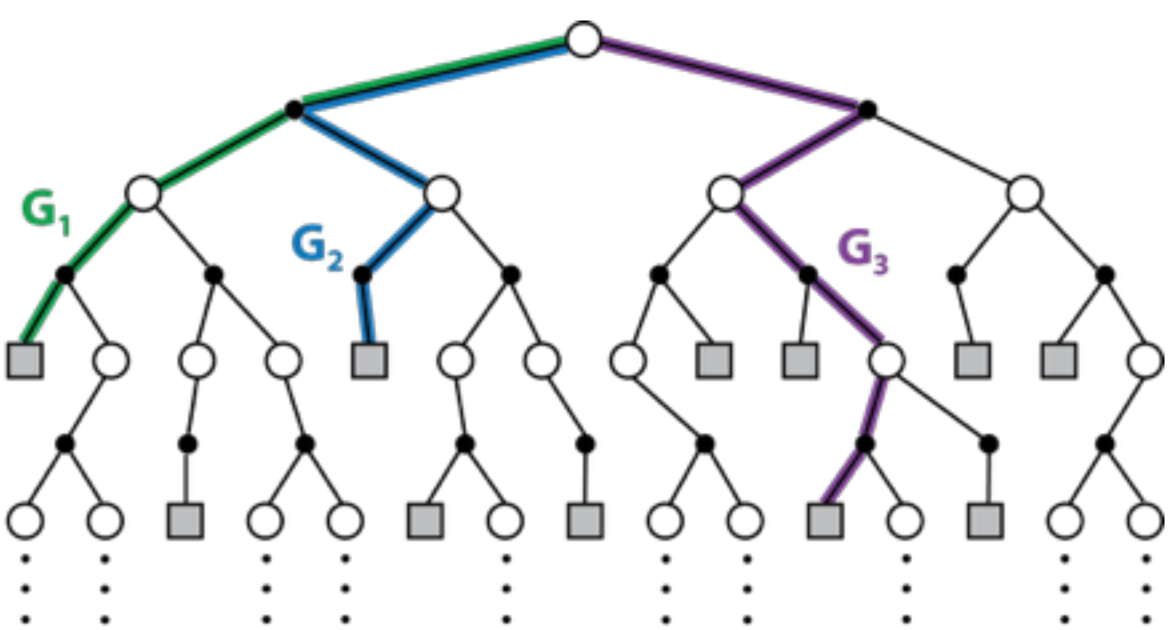
How to evaluate decision traces of MDPs

2) **Sampling** — estimate value by taking the empirical average of sampled episodes

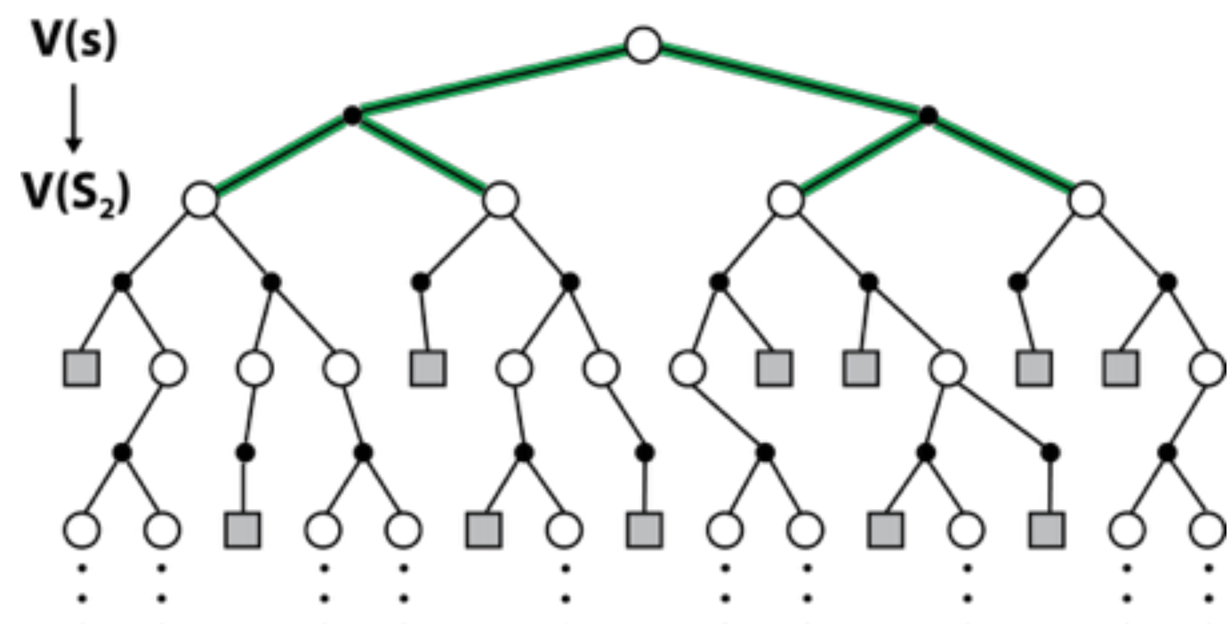
$$v_\pi(s) = \frac{1}{N}\sum_{i=1}^{N} G(\tau_i), \text{ for } S_0 = s$$

3) **Bootstrapping** — estimate value by using values of successor states

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

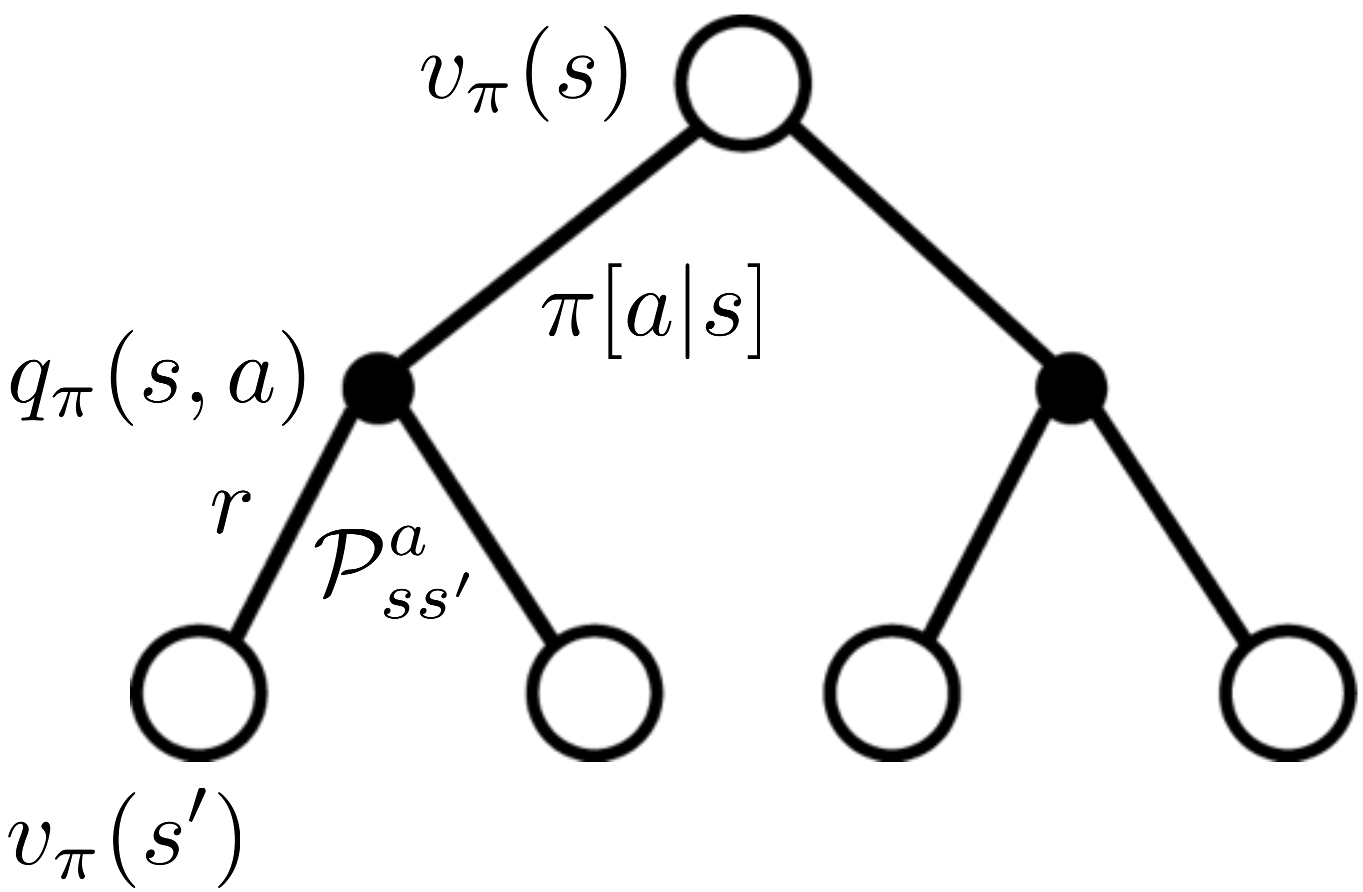


sampling



bootstrapping

Bootstrapping and the Bellman Expectation Equations

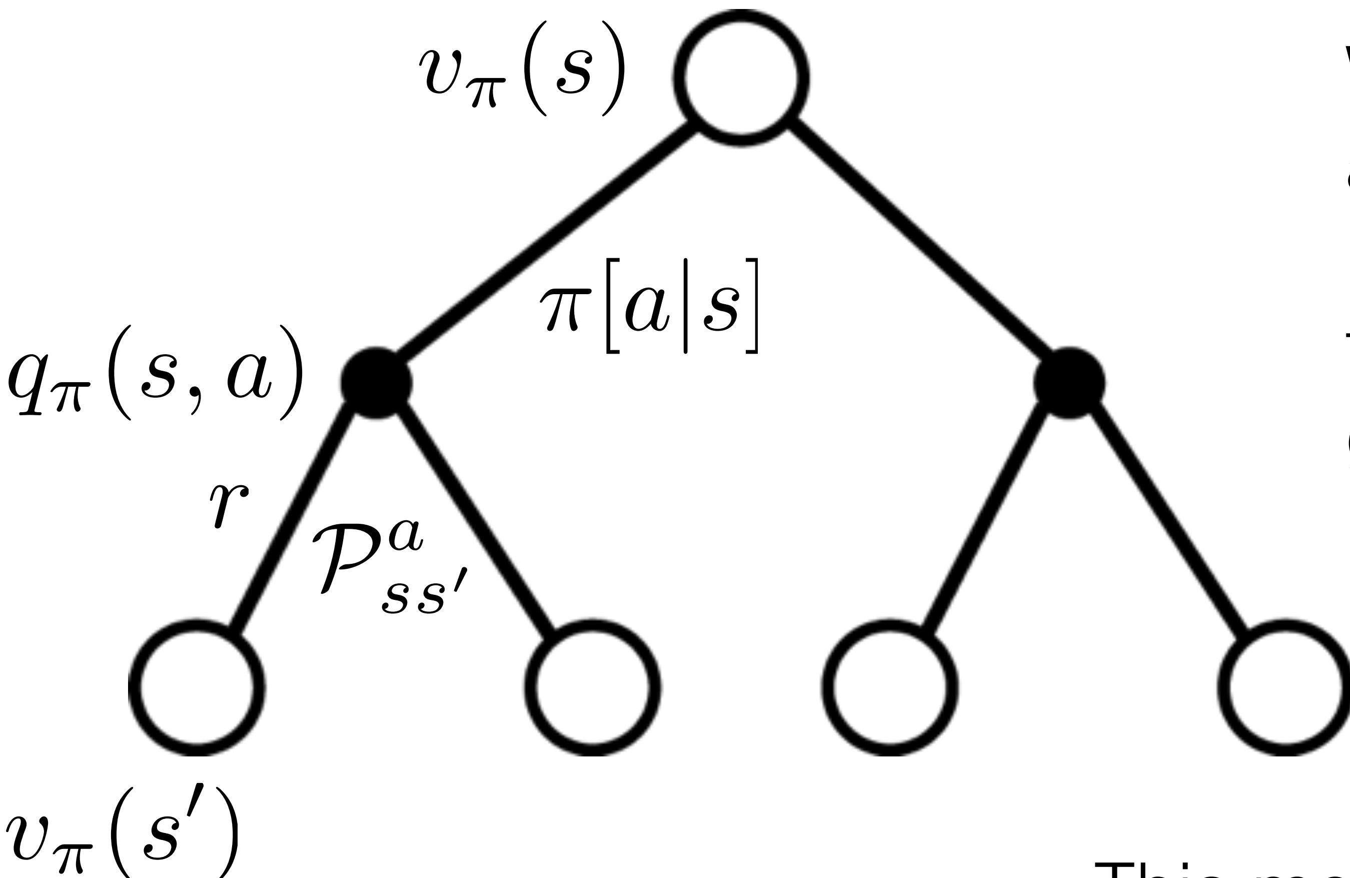


$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) q_\pi(s,a)$$

$$q_\pi(s,a) = \mathcal{R}^a_s + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}^a_{ss'} v_\pi(s')$$

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(R^a_s + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}^a_{ss'} v_\pi(s') \right)$$

Bootstrapping and the Bellman Expectation Equations



IMPORTANT POINT:

When a fixed policy π is used to make actions for a Markov Decision Process, it becomes a simple Markov Chain with transition probabilities between states governed by $\mathcal{P}^\pi_{ss'}$ where

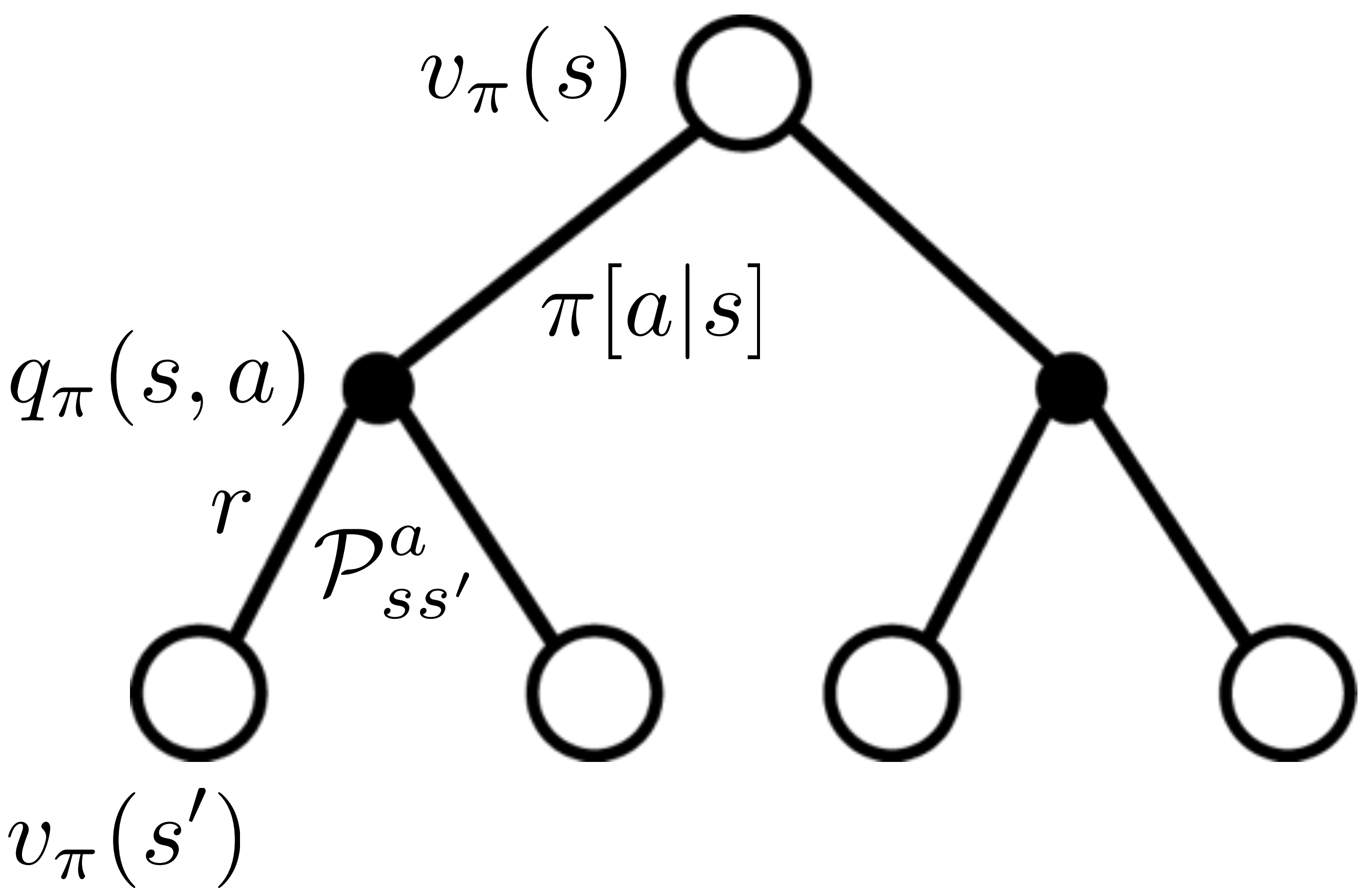
$$\mathcal{P}^\pi_{ss'} = \sum_{a \in \mathcal{A}} \pi(a|s) \mathcal{P}^a_{ss'}$$

This means that the value function can also be defined in terms of $\mathcal{P}^\pi_{ss'}$:

$$v_\pi(s) = R^\pi_s + \gamma \sum_{s' \in \mathcal{S}} P^\pi_{ss'} v_\pi(s')$$

$$\mathcal{R}^\pi_s = \sum_{a \in \mathcal{A}} \pi(a|s) \mathcal{R}^a_s$$

Bootstrapping and the Bellman Expectation Equations



$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) q_\pi(s,a)$$

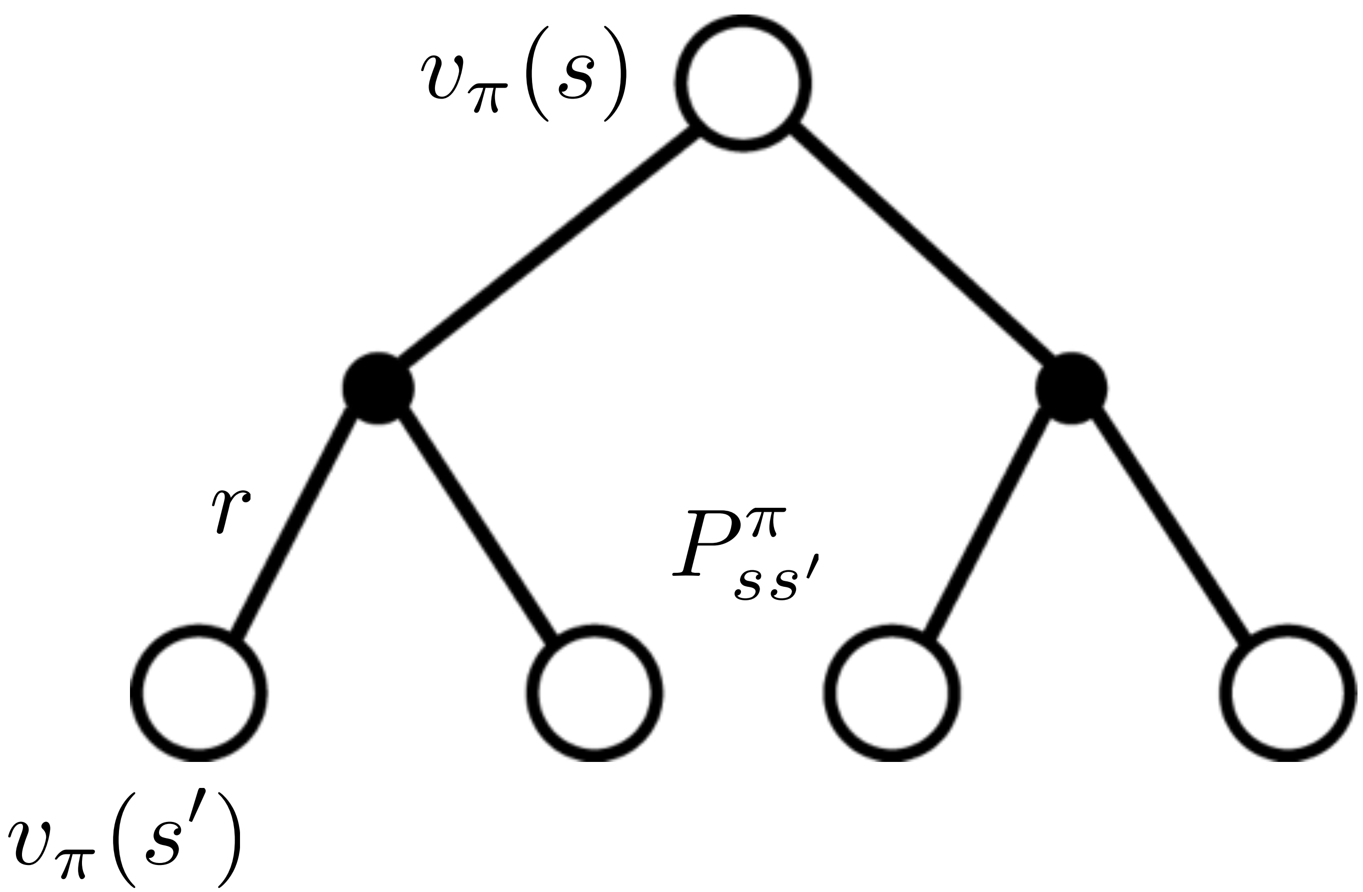
$$q_\pi(s,a) = \mathcal{R}^a_s + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}^a_{ss'} v_\pi(s')$$

$$v_\pi(s) = R^\pi_s + \gamma \sum_{s' \in \mathcal{S}} P^\pi_{ss'} v_\pi(s')$$

$$\mathcal{P}^\pi_{ss'} = \sum_{a \in \mathcal{A}} \pi(a|s) \mathcal{P}^a_{ss'}$$

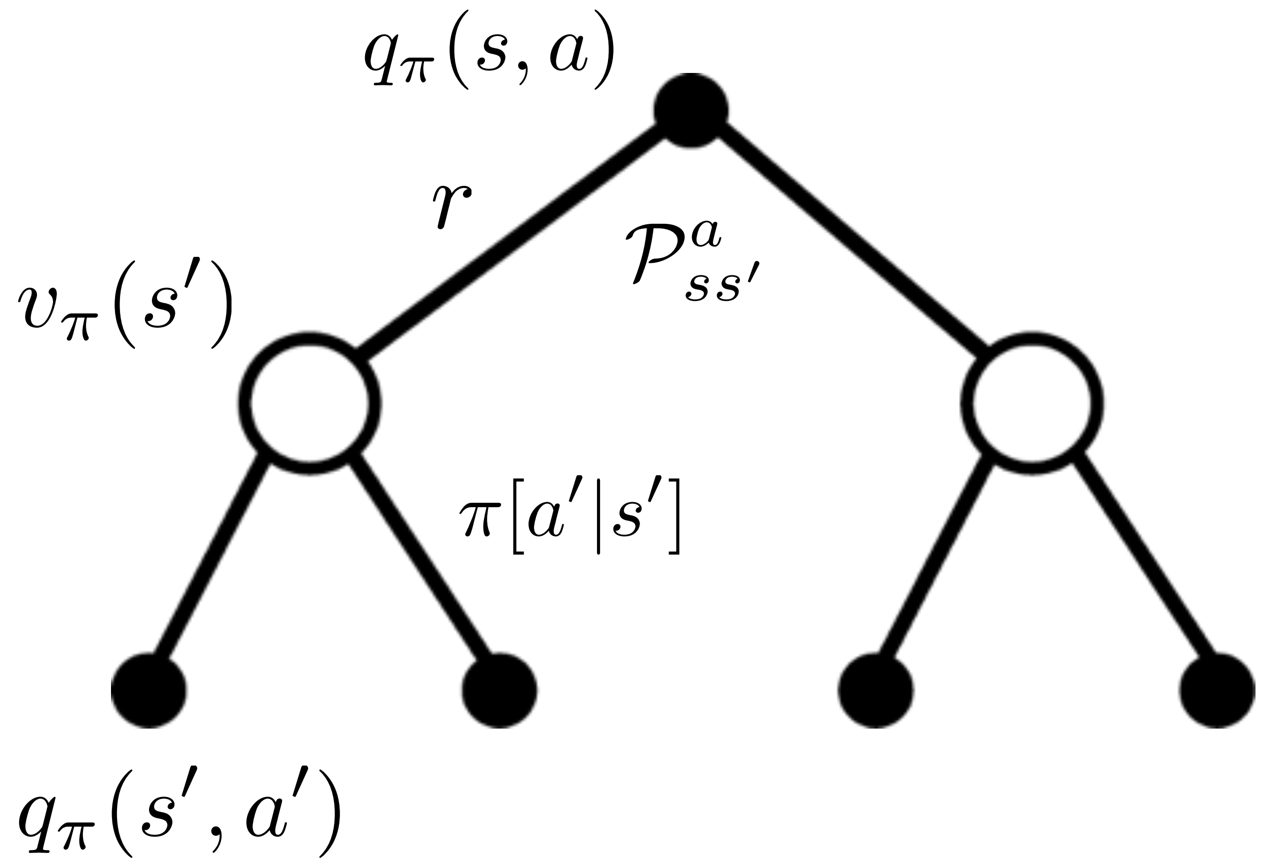
$$\mathcal{R}^\pi_s = \sum_{a \in \mathcal{A}} \pi(a|s) \mathcal{R}^a_s$$

Bootstrapping and the Bellman Expectation Equations



$$v_\pi(s) = R^\pi_s + \gamma \sum_{s' \in \mathcal{S}} P^\pi_{ss'} v_\pi(s')$$

Bootstrapping and the Bellman Expectation Equations

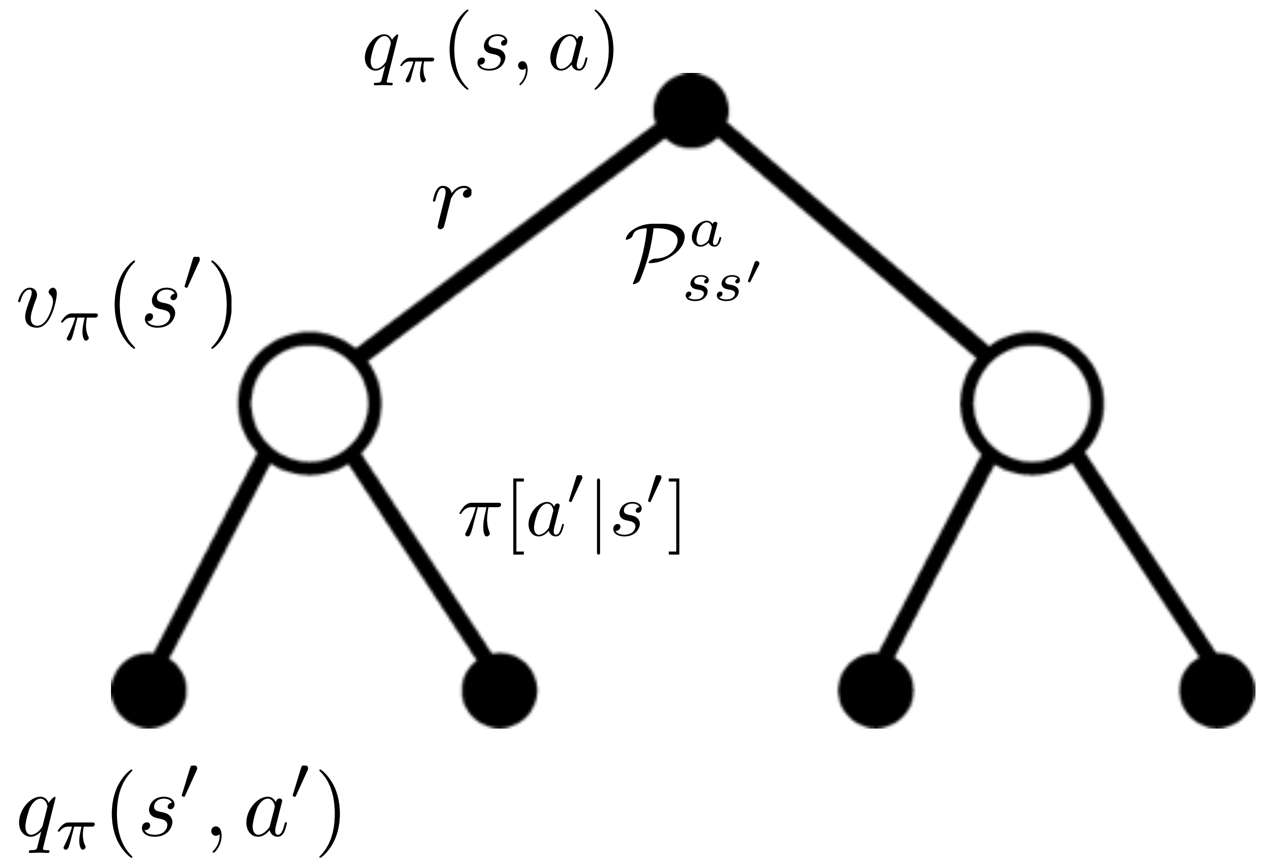


$$q_\pi(s,a) = \mathcal{R}^a_s + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}^a_{ss'} v_\pi(s')$$

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) q_\pi(s,a)$$

$$q_\pi(s,a) = \mathcal{R}^a_s + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}^a_{ss'} \sum_{a' \in \mathcal{A}} \pi(a'|s') q_\pi(s',a')$$

Bootstrapping and the Bellman Expectation Equations



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

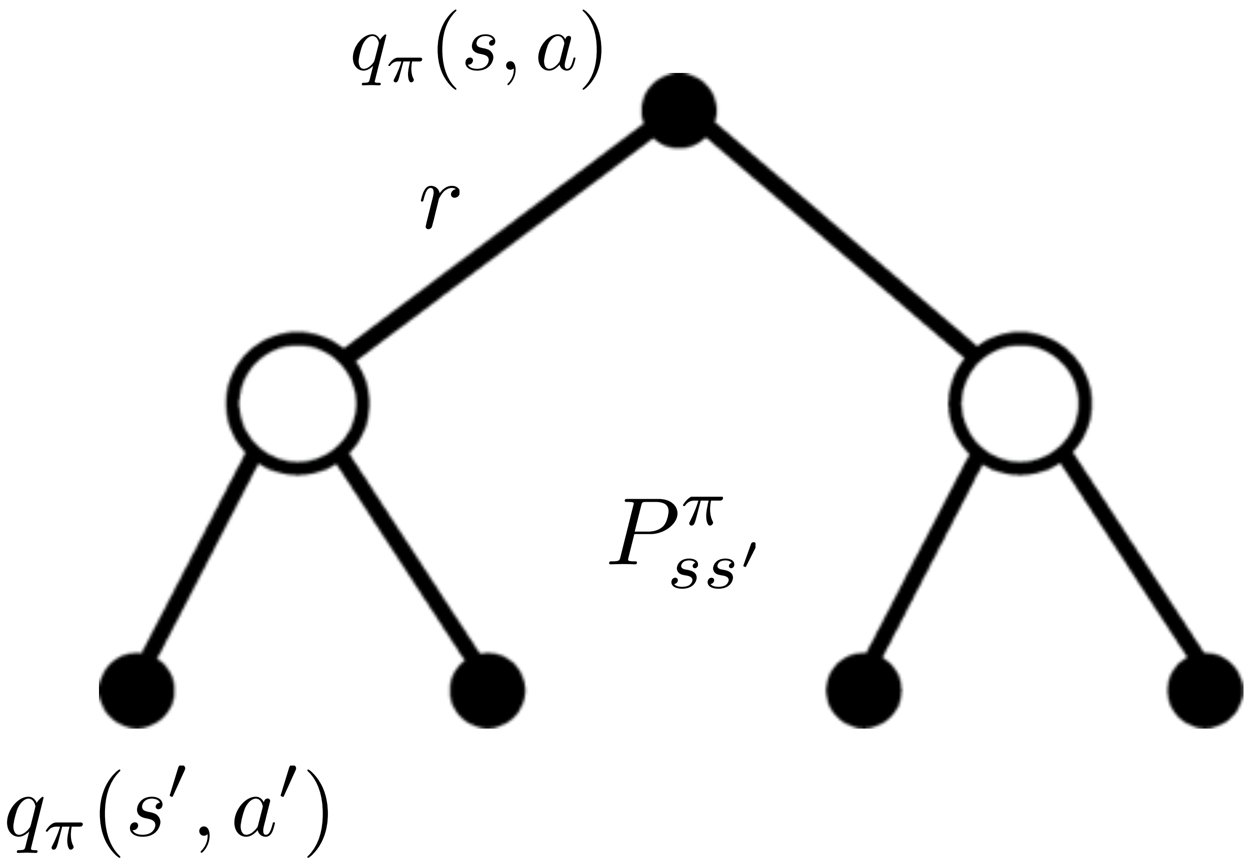
$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) q_\pi(s,a)$$

$$q_\pi(s,a) = R_s^\pi + \gamma \sum_{a' \in \mathcal{A}} P_{ss'}^\pi q_\pi(s',a')$$

$$\mathcal{P}_{ss'}^\pi = \sum_{a \in \mathcal{A}} \pi(a|s) \mathcal{P}_{ss'}^a$$

$$\mathcal{R}_s^\pi = \sum_{a \in \mathcal{A}} \pi(a|s) \mathcal{R}_s^a$$

Bootstrapping and the Bellman Expectation Equations



$$q_\pi(s,a) = R^\pi_s + \gamma \sum_{a' \in \mathcal{A}} P^\pi_{ss'} q_\pi(s',a')$$

The Bellman Optimality Equations

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

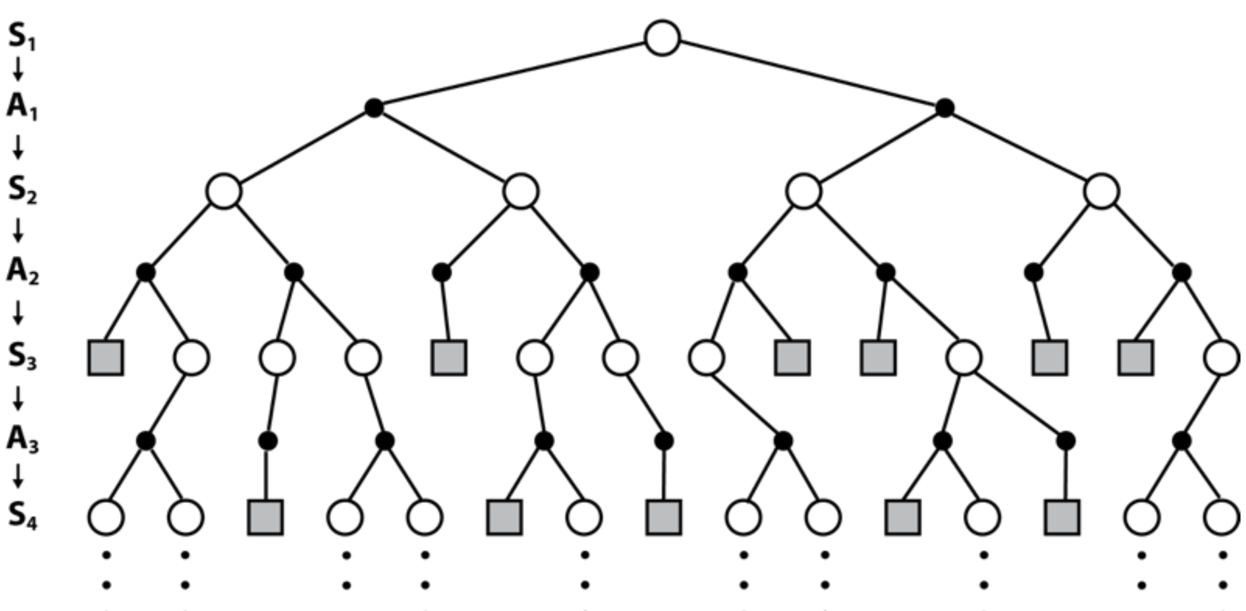
- This specifies the best possible performance in the MDP
- But how to reach it? i.e., how can we obtain the optimal policy?
- **One idea:** if we know the optimal state-value or state-action-value function, then obtaining a good policy is easy:

$$\pi_*(a|s) = \begin{cases} 1 & \text{if } a = \arg\max\limits_{a \in \mathcal{A}} \; q_*(s,a) \\ 0 & \text{otherwise} \end{cases}$$

★ **Value-based RL: If we can develop ways of improving our value functions, then we can get the optimal policy for free**

Summary so far

- MDPs are environments with states, decision and rewards
- Agents learn values of states or state-actions (prediction) and optimize policies (control)
- Value-based RL: learn a good value function, use it to choose actions
- Prediction: brute force vs sampling vs bootstrapping — Bellman expectation equations
- Control: Bellman optimality equations



Bellman expectation equations:

$$v_\pi(s) = R_s^\pi + \gamma \sum_{s' \in \mathcal{S}} P_{ss'}^\pi v_\pi(s')$$

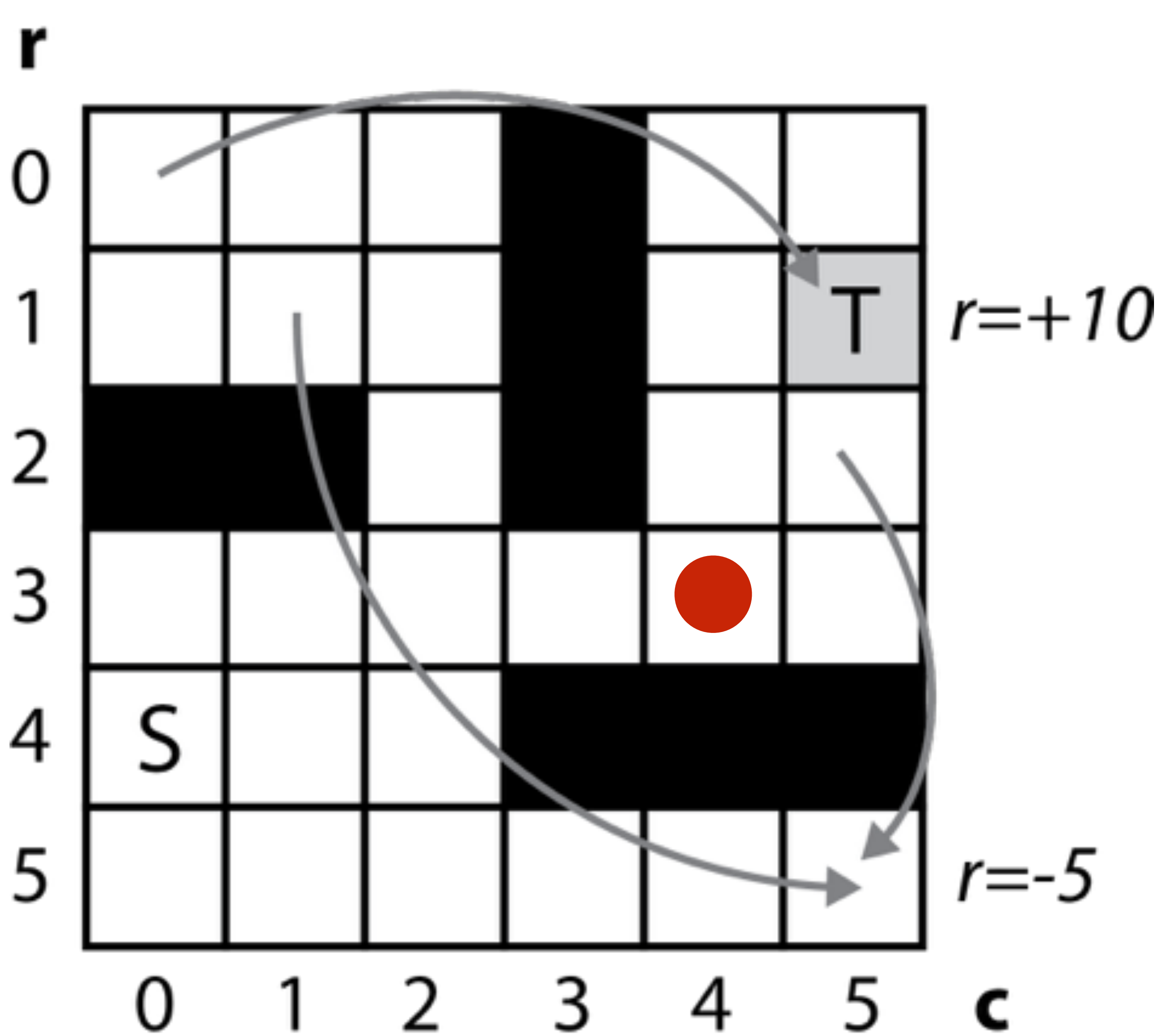
$$q_\pi(s,a) = R_s^\pi + \gamma \sum_{a' \in \mathcal{A}} P_{ss'}^\pi q_\pi(s',a')$$

Bellman optimality equations:

$$v_*(s) = \max_\pi v_\pi(s)$$

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

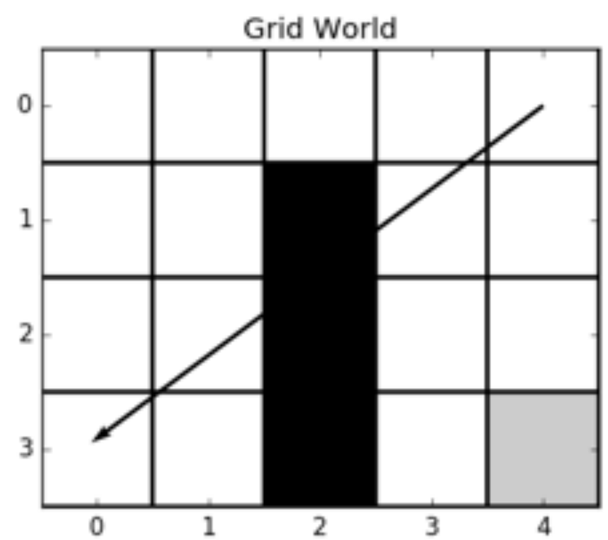
Running example: Gridworld



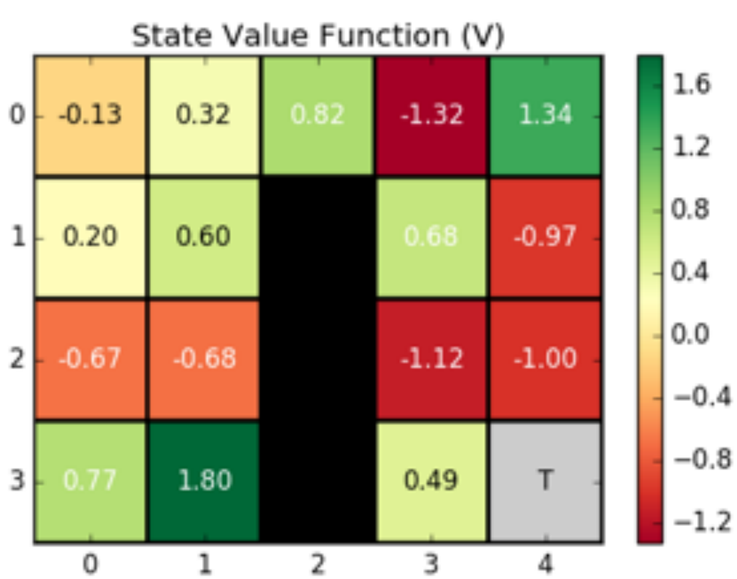
- Shape = (nrows=6,ncols=6)
- State = (3,4)
- Start state = (4,0)
- Terminal state(s) = [(1,5)]
- Obstacle(s) = [(2,0),(2,1),(0,3),(1,3),(2,3),(4,3),(4,4),(4,5)]
- Jumps = {(0,0):(1,5), (1,1):(5,5),(2,5):(5,5)}
- Actions = {Down, Up, Right, Left, Jump}
  *(jump states only allow jump actions)
- Rewards = {(1,5):10, (5,5):-5}
  (entering state)

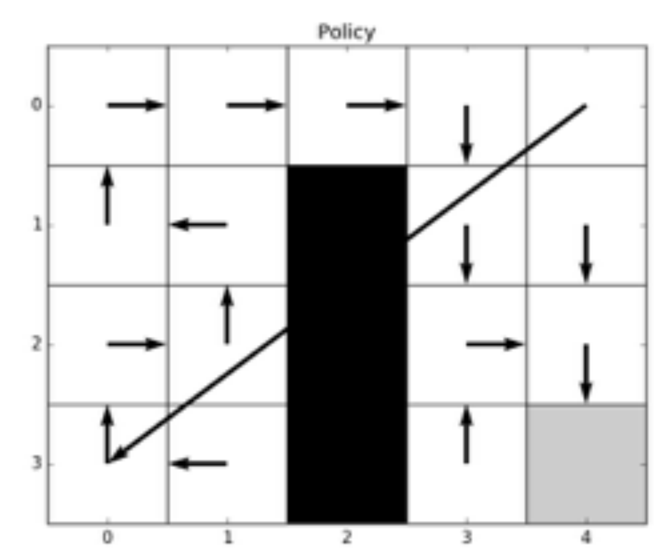
Gridworld setup

- **Programming: do question 1 of assignment**



- Initialize a gridworld and RL agent
- Run episode and observe states, actions and rewards
- Plot the v(s), q(s,a) and policy





Overview of methods

- Model-based vs. model-free:
  - Model-based methods: requires full knowledge of the MDP (states and transitions between states) — e.g., dynamic programming (DP)

$$v_\pi(s) = R_s^\pi + \gamma \sum_{s' \in \mathcal{S}} P_{ss'}^\pi v_\pi(s')$$

  - Model-free methods: does not require knowledge of MDP, information is acquired through sampling trajectories — e.g., Monte Carlo (MC) and temporal difference (TD)

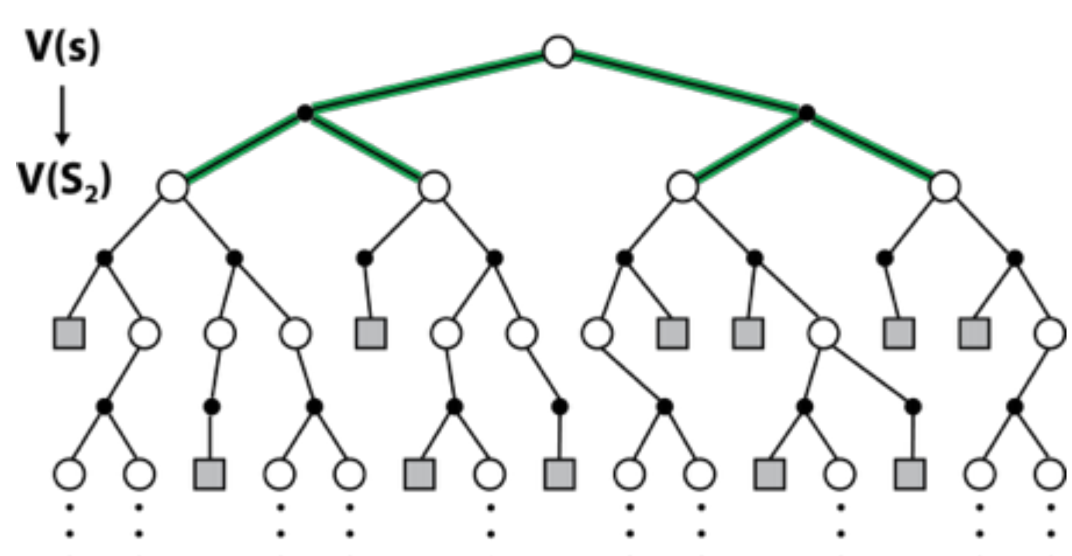
$$v_\pi(s) = ?$$

Dynamic programming (DP)

- Assumes full knowledge of the Markov Decision Process
- Prediction with **policy evaluation**: using the Bellman Expectation equation, we can analytically calculate the state-value function for a particular policy:

$$v_{k+1}(s) = R_s^\pi + \gamma \sum_{s' \in \mathcal{S}} P_{ss'}^\pi v_k(s') \text{ for all } s$$

matrix form: $\mathbf{v}^{k+1} = \mathcal{R}^\pi + \gamma \mathcal{P}^\pi \mathbf{v}^k$



note: can be done with v or q

Prediction with policy evaluation

- **Programming: do question 2.1 of assignment**

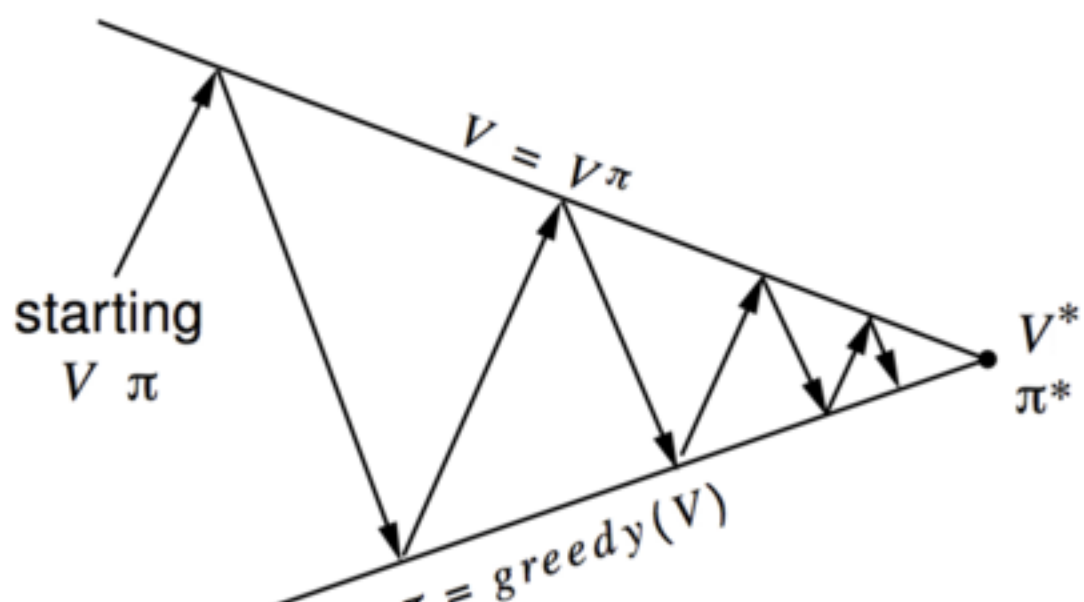
Algorithm 1 Policy Evaluation

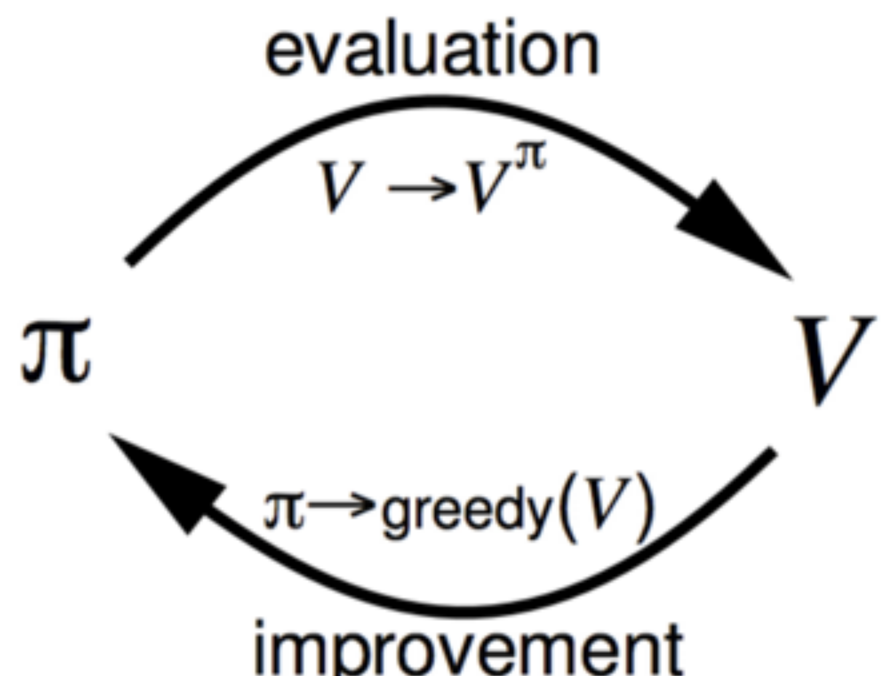
procedure POLICYEVAL(γ)
  Initialize $v(s) = 0$ for all states s
  while loop: $\Delta > 0.0001$
    for loop: for all states s in $\mathcal{S}$
      $v'(s) \leftarrow \sum_a \pi(a|s) \sum_{s'} P^a_{ss'} [R^a_{ss'} + \gamma v(s')]$
      (note: this is equivalent to $v(s) \leftarrow R^{\pi}_{ss'} + \gamma P^{\pi}_{ss'} v(s')$)
    end loop
    $\Delta \leftarrow \max_s |v'(s) - v(s)|$
    $v(s) \leftarrow v'(s)$ for all s
  end loop

Dynamic programming (DP)

- Control with **policy iteration**: once the state-value function is calculated through *policy evaluation*, the policy can be updated by acting greedily with respect to the value of subsequent states (*policy improvement*)

$$\pi_g(s) = \arg\max_{a \in \mathcal{A}} \; q_\pi(s,a) = \arg\max_{a \in \mathcal{A}} \; v_\pi(s') \text{ for } s \xrightarrow{a} s'$$





David Silver, 2015

Control with policy iteration

- **Programming: do question 2.2 of assignment**

Algorithm 2 Policy Improvement

procedure POLICYIMPROVE
  Initialize matrix $P^{\pi}(s, s') = 0$ for all pairs of states s, s'
  for loop: for all states s in $\mathcal{S}$
    Initialize max_v $= -1e6$ and max_v_state $= 0$
    for loop: for all states s' in $\mathcal{S}$
      for loop: for all actions a in $\mathcal{A}$
        if $P^{a}_{ss'} > 0$ **and** $v(s') >$ max_v:
          max_v $\leftarrow v(s')$
          max_v_state $\leftarrow s'$
        end if
      end for loop
    end for loop
    $P^{\pi}(s,\text{max\_v\_state}) \leftarrow 1$
  end loop

Algorithm 3 Policy Iteration

procedure POLICYITER(γ)
  Initialize pStable = False
  while loop: (while pStable is False)
    $P^{\pi}_{\text{old}} \leftarrow P^{\pi}$
    POLICYEVAL(γ)
    POLICYIMPROVE()
    if P^{π}_{old} equals P^{π}:
      pStable = True
    end if
  end loop

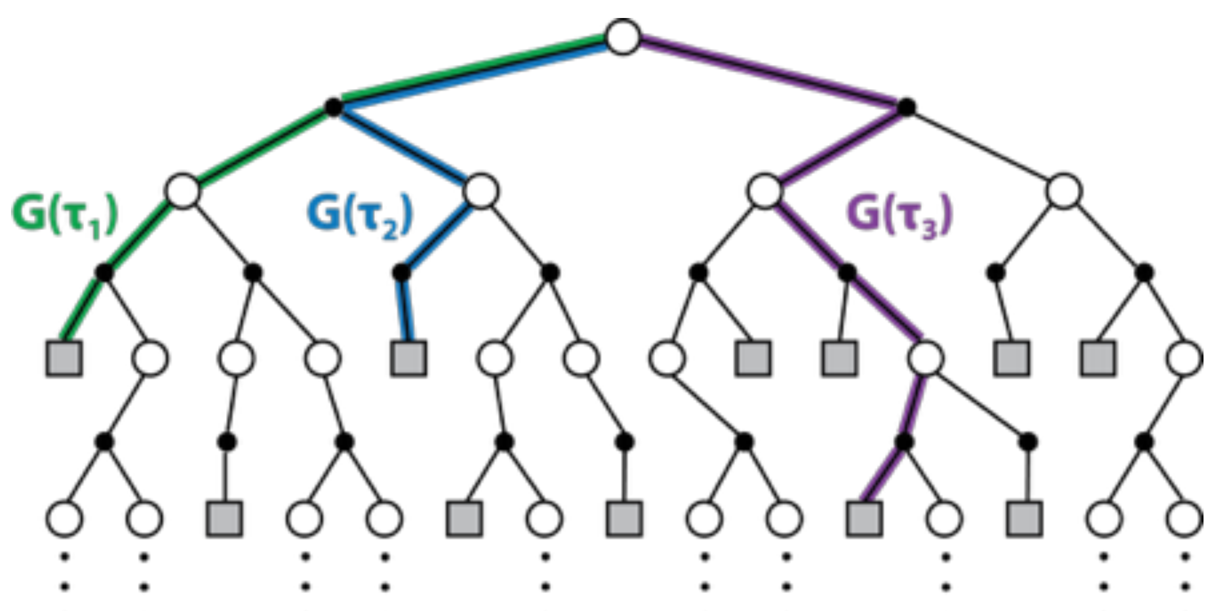
Monte Carlo (MC) methods

- **Model-free:** Learn directly from episodes of experience — upon episode completion, update the value of all states (or state-actions) visited — **sampling**

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

- Prediction: run many episodes and estimate empirical value of a state as empirical mean of the return starting from that state

$$v_\pi(s) \approx V(s) = \frac{1}{N} \sum_{i=1}^{N} G_t(\tau_i), \text{ for } S_t = s$$



incremental:

$$V^k(s) = V^{k-1}(s) + \frac{1}{k}(G_t(\tau_k) - V^{k-1}(s))$$

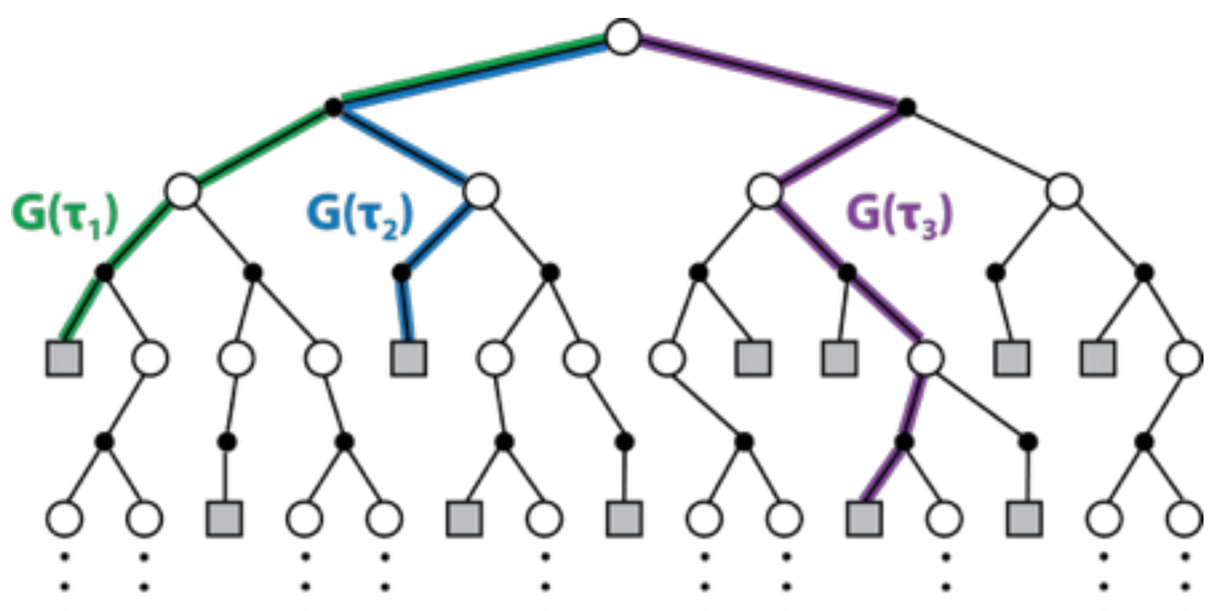
Monte Carlo (MC) methods

- **Model-free:** Learn directly from episodes of experience — upon episode completion, update the value of all states (or state-actions) visited — **sampling**

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

- Prediction: run many episodes and estimate empirical value of a state as empirical mean of the return starting from that state

$$v_\pi(s) \approx V(s) = \frac{1}{N}\sum_{i=1}^{N} G_t(\tau_i), \text{ for } S_t = s$$



incremental:

$$V^k(s) = V^{k-1}(s) + \alpha(G_t(\tau_k) - V^{k-1}(s))$$

Monte Carlo (MC) control

- Doing policy improvement with state-value function v(s) requires knowledge of the world — so we can use state-action-value function q(s,a) instead.

$$\pi_g(s) = \arg\max_{a \in \mathcal{A}} q_\pi(s,a) = \arg\max_{a \in \mathcal{A}} v_\pi(s') \text{ for } s \xrightarrow{a} s'$$

- Naive approach: policy iteration with MC estimation and greedy policy improvement — why might this be a problem?

- Problem: greedy policy improvement doesn't allow for enough exploration

- Solution: epsilon-greedy policy

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

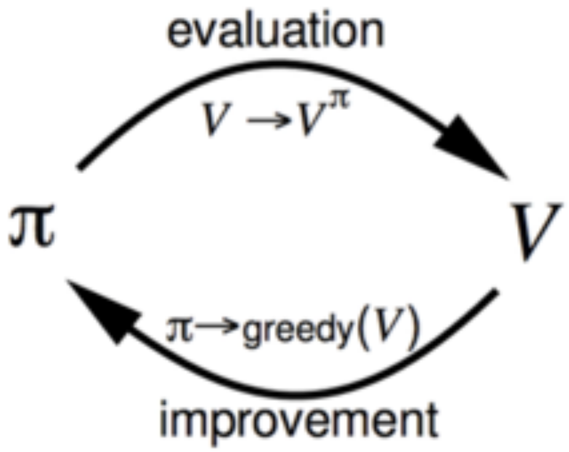
Monte Carlo (MC) summary

- **Evaluation:** for each episode, update the estimate of the value

$$Q^k(s,a) = Q^{k-1}(s,a) + \alpha(G_t(\tau_k) - Q^{k-1}(s,a))$$

- **Improvement:** update policy to be ϵ -greedy w.r.t Q-value function

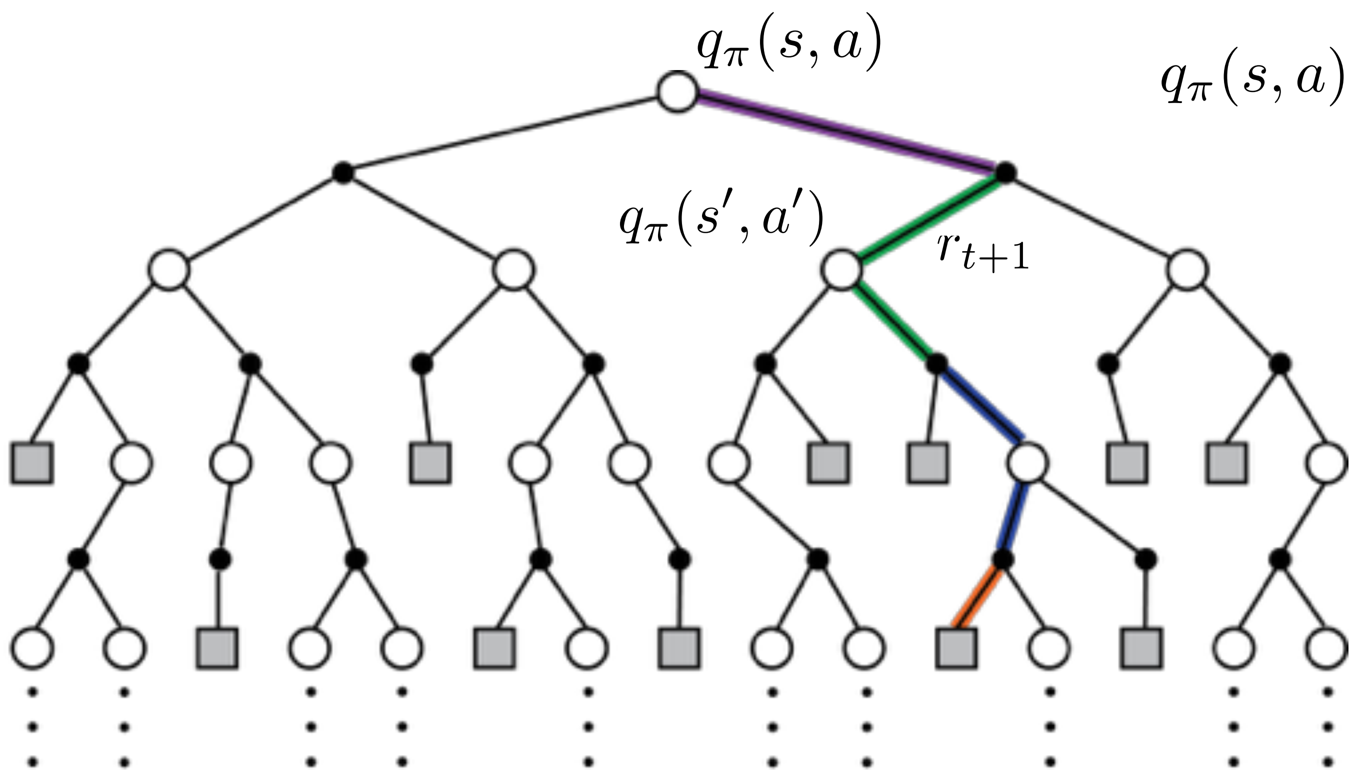
$$\pi[a|s] = \begin{cases} \epsilon/m + 1 - \epsilon & \text{if } a = \arg\max_{a \in \mathcal{A}} q(s,a) \\ \epsilon/m & \text{otherwise} \end{cases}$$



For later: If you finish all of the problems, you can try to implement MC methods…

Temporal Difference (TD) learning

- Monte Carlo (MC): value estimation from samples of episodes, but learning is only done at the end of the episode.
- Dynamic programming (DP): efficient value estimation based on bootstrapping values of successor states, but requires knowledge of the world.
- Temporal difference (TD) learning combines efficient value estimation by bootstrapping along with sampling from episodes.

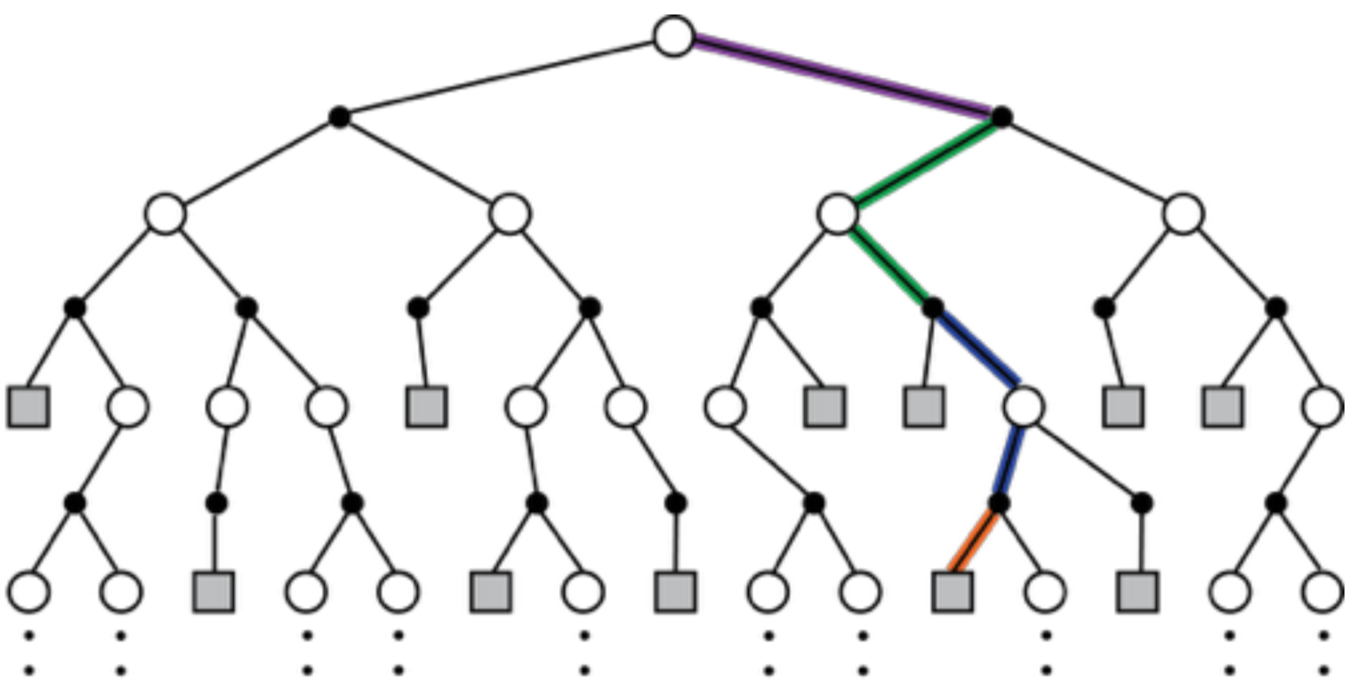


$$q_\pi(s, a) = \mathbb{E}_\pi[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) | S_t = s, A_t = a]$$

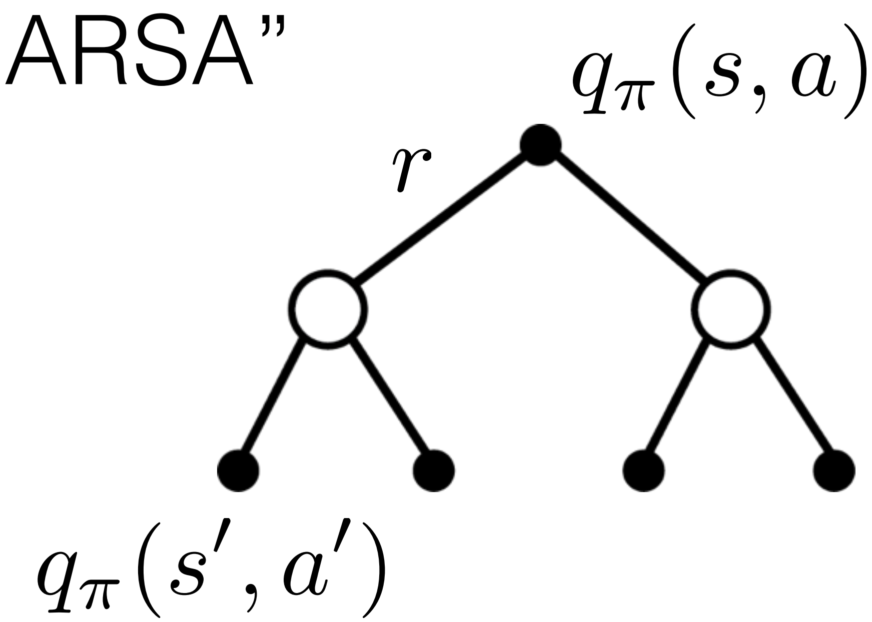
Prediction with TD-estimation

- Incremental update of value with prediction:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\underbrace{(r + \gamma Q(s',a') - Q(s,a))}_{\delta_t \text{ (TD-error)}}$$



“SARSA”



Prediction with TD-estimation

- **Programming: do question 3.1 of assignment**

Algorithm 4 Temporal Difference Policy Evaluation

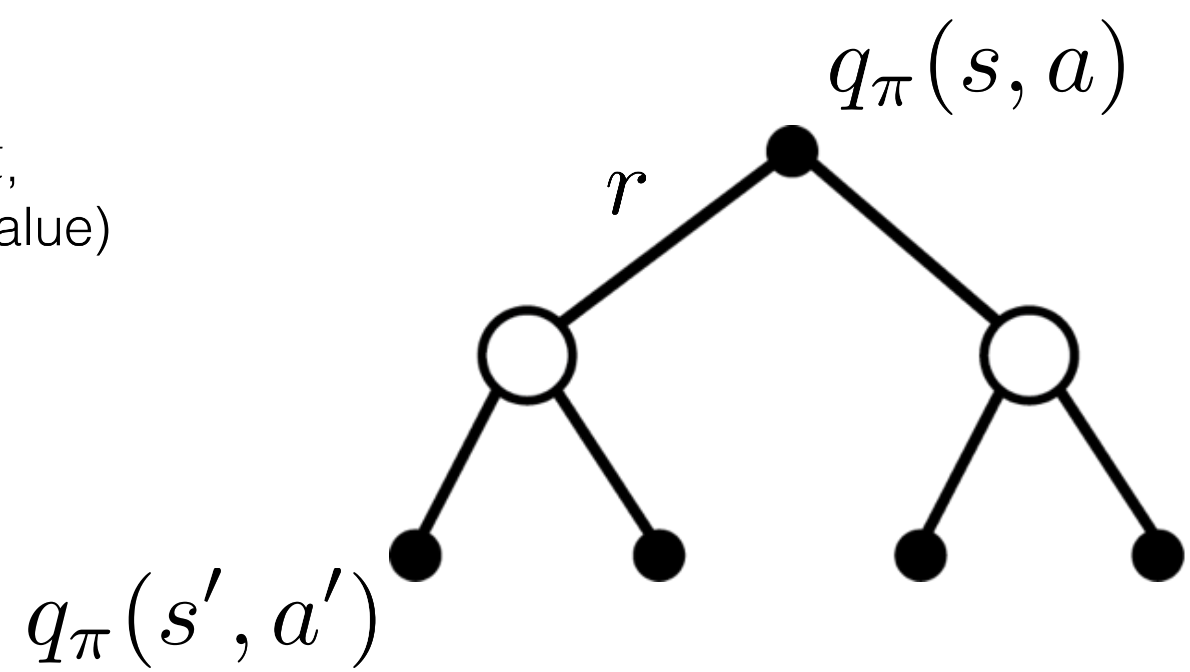
procedure TDPolicyEval(γ, α, ntrials)
    Initialize $Q(s, a)$ arbitrarily for all states s (e.g., $Q(s, a) = 0$)
    for loop: for $i = 1$ to ntrials (number of episodes)
        Initialize in starting state s
        Choose action a from state s according to policy π
        while loop: while terminal state has not been reached
            Take action a, observe reward r and new state s'
            Choose action a' from state s' according to policy π
            $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)]$
            $s \leftarrow s', a \leftarrow a'$
        end loop (when terminal state is reached)
    end loop

Control with TD: SARSA algorithm

- Evaluation: TD-estimation
- Improvement: epsilon-greedy policy

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

(choose a random action with probability ϵ,
otherwise choose the action with the highest value)



Control with TD: SARSA algorithm

- **Programming: do question 3.2 of assignment**

Algorithm 5 SARSA Policy Evaluation

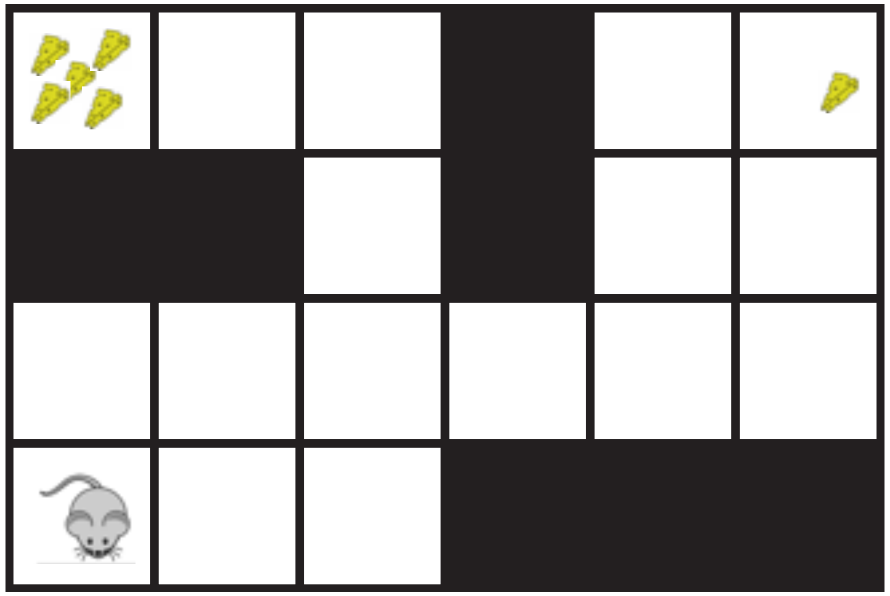
```
procedure SARSA_POLICYEVAL(γ, α, ntrials)
    Initialize Q(s,a) arbitrarily for all states s (e.g., Q(s,a) = 0)
    for loop: for i = 1 to ntrials (number of episodes)
        Initialize in starting state s
        Choose action a from state s derived from Q (e.g., ϵ-greedy)
        while loop: while terminal state has not been reached
            Take action a, observe reward r and new state s′
            Choose action a′ from state s′ derived from Q (e.g., ϵ-greedy)
            Q(s,a) ← Q(s,a) + α[r + γQ(s′,a′) − Q(s,a)]
            s ← s′, a ← a′
        end loop (when terminal state is reached)
    end loop
```

Algorithm 6 SARSA Policy Iteration

```
procedure SARSA_POLICYITER(γ, α, ntrials)
    Initialize pStable = False
    Initialize a_old arbitrarily
    while loop: (while pStable is False)
        SARSA_POLICYEVAL(γ, α, ntrials)
        if : a_old equals arg max_a Q(s,a)
            pStable = True
        a_old = arg max_a Q(s,a) for all states s
        end if
    end loop
```

On-policy vs off-policy

- For Monte Carlo and Temporal Difference methods, we often choose a non-optimal policy in order to ensure enough exploration.
- Off-policy methods allow us to optimize a policy while following a different one.
- Why might we want to do this?
- On-policy methods typically perform better than off-policy methods, but off-policy methods typically find a better policy.



Off-policy TD learning: Q-learning

- The agent has two policies — the behavior policy $\mu(s)$ (e.g., ϵ-greedy), and the optimized policy $\pi(s)$ (e.g., greedy)

- The next action is chosen using the behavior policy, but Q-values are updated using the optimized policy.

$$Q(s, a_\mu) \leftarrow Q(s, a_\mu) + \alpha(r + \gamma Q(s', a'_\pi) - Q(s, a_\mu))$$

- This allows us to optimize a greedy policy (as in DP methods), but we don't have to worry about exploration (due to ϵ-greedy behavior policy)

Off-policy TD learning: Q-learning

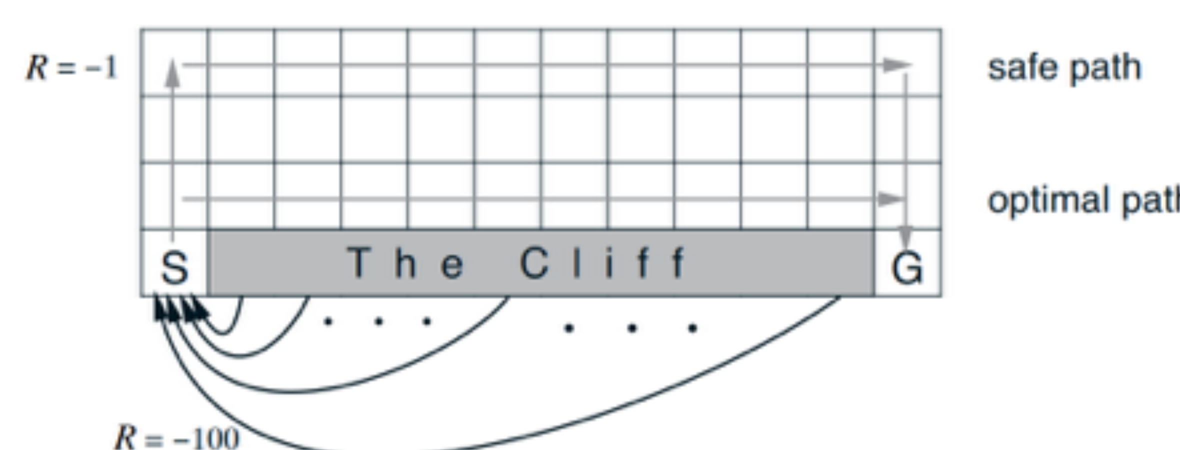
- **Programming: do question 4 of assignment**

Algorithm 7 Q-learning Policy Evaluation

procedure Q_POLICYEVAL(γ, α, ntrials)
  Initialize $Q(s, a)$ arbitrarily for all states s (e.g., $Q(s, a) = 0$)
  for loop: for $i = 1$ to ntrials (number of episodes)
    Initialize in starting state s
    Choose action a from state s derived from Q and behavior policy μ
    (e.g., ϵ-greedy)
    while loop: while terminal state has not been reached
      Take action a, observe reward r and new state s'
      Choose action a' from state s' derived from Q and behavior policy μ
      (e.g., ϵ-greedy)
      Choose action a'' from state s' derived from Q and optimized policy π
      (e.g., greedy)
      $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma Q(s', a'') - Q(s, a)]$
      $s \leftarrow s', a \leftarrow a'$ (note that behavior policy is followed)
    end loop (when terminal state is reached)
  end loop

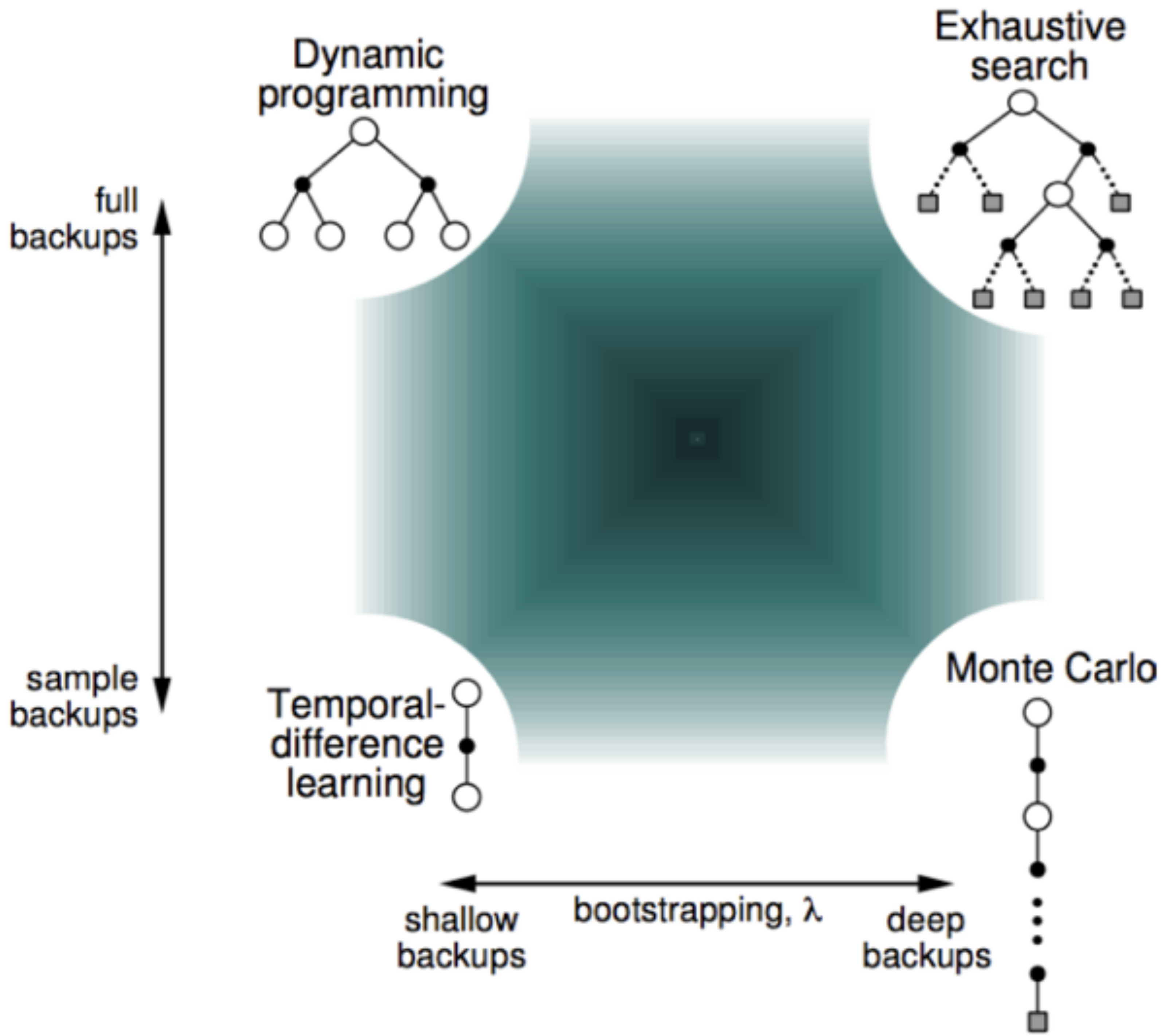
Algorithm 8 Q-learning Policy Iteration

procedure Q_POLICYITER(γ, α, ntrials)
  Initialize pStable = False
  Initialize $\mathbf{a}_{\text{old}}$ arbitrarily
  while loop: (while pStable is False)
    Q_POLICYEVAL(γ, α, ntrials)
    if : $\mathbf{a}_{\text{old}}$ equals $\arg\max_a Q(s, a)$
      pStable = True
    $\mathbf{a}_{\text{old}} = \arg\max_a Q(s, a)$ for all states s
    end if
  end loop



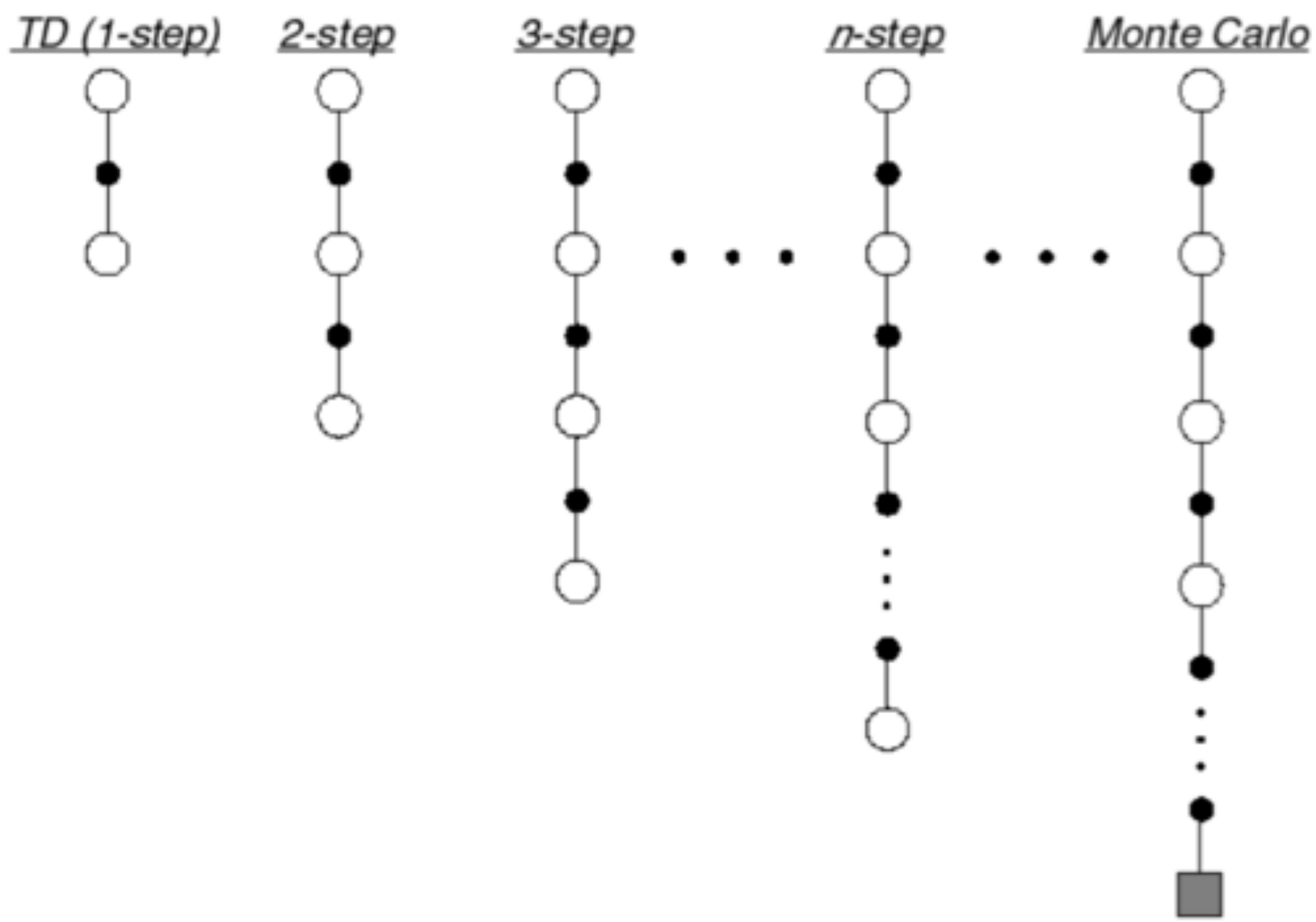
Sutton & Barto

Unified view of RL

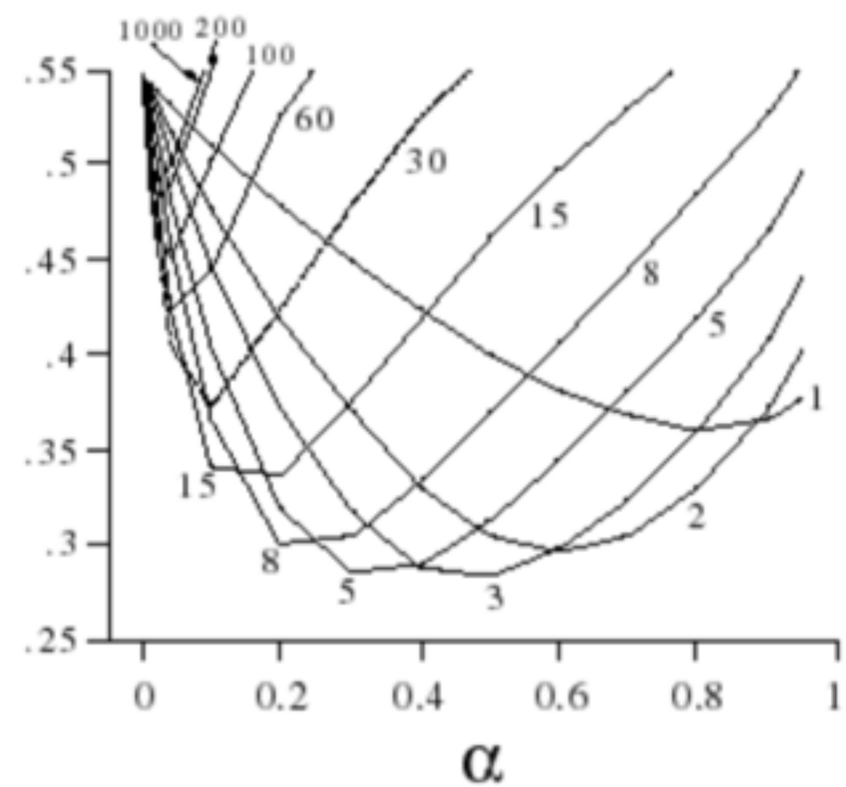


Sutton & Barto

N-step methods and the lambda return



How many steps to choose?



$n = 1$ (TD) $G_t^{(1)} = R_{t+1} + \gamma V(S_{t+1})$

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

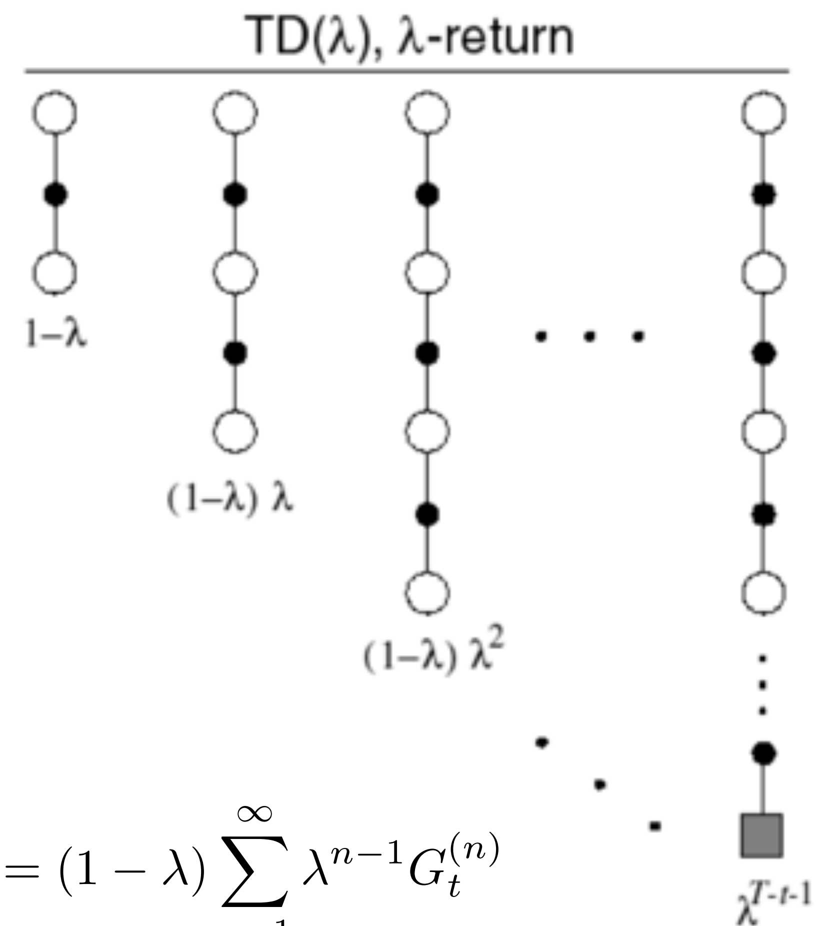
$\vdots$ $\vdots$

$n = \infty$ (MC) $G_t^{(\infty)} = R_{t+1} + \gamma R_{t+2} + ... + \gamma^{T-1} R_T$

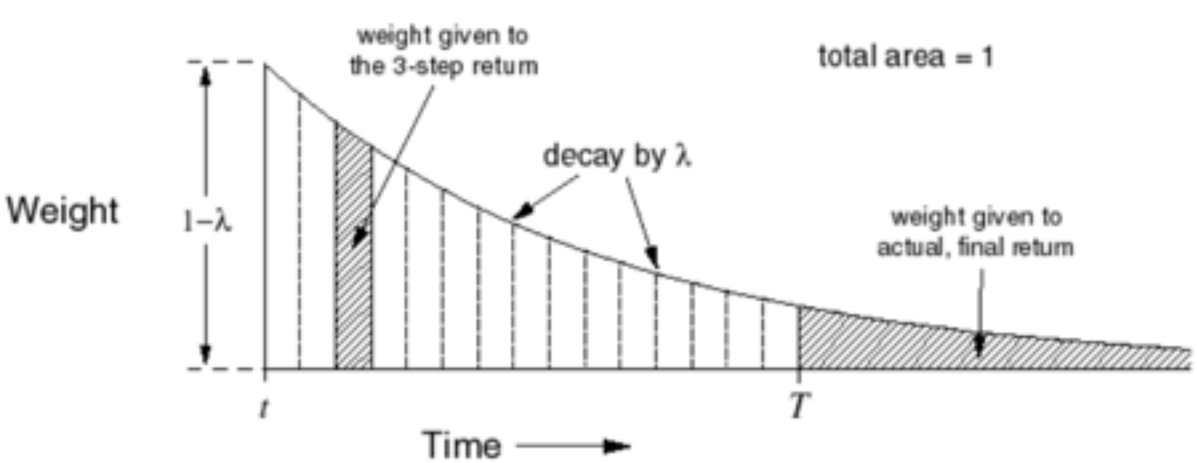
Sutton & Barto

David Silver, 2015

N-step methods and the lambda return

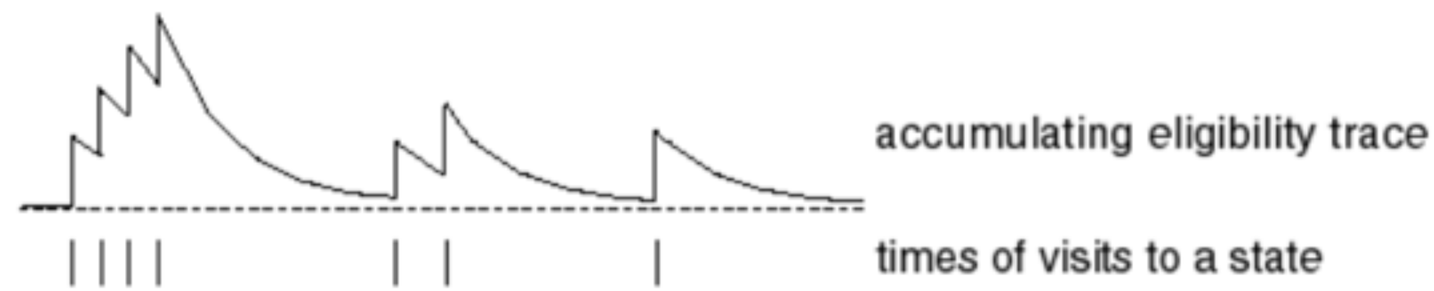


$$G_t^{\lambda} = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} G_t^{(n)}$$



How to implement?

eligibility traces:



$$E_t(s,a) = \gamma\lambda E_{t-1}(s,a) + \mathbf{1}(S_t = s, A_t = a)$$

$$\delta_t = R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)$$

$$Q(s,a) \leftarrow Q(s,a) + \alpha\delta_t E_t(s,a) \text{ for all } s, a$$

David Silver, 2015

SARSA(λ) for gridworld

- **Programming: do question 5 of assignment**

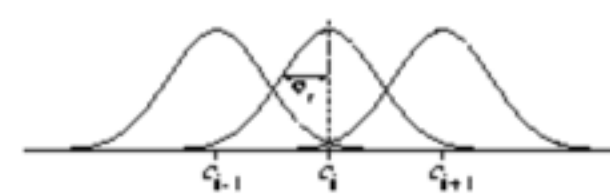
Algorithm 9 SARSA(λ) Policy Evaluation

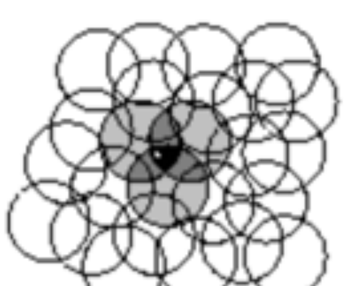
procedure SARSA_λ_EVAL(γ, α, ntrials,λ)
    Initialize $Q(s, a) = 0$ for all states s and actions a
    Initialize $E(s, a) = 0$ for all states s and actions a
    for loop: for $i = 1$ to ntrials (number of episodes)
        Initialize in starting state s
        Choose action a from state s derived from Q (e.g., ϵ-greedy)
        while loop: while terminal state has not been reached
            Take action a, observe reward r and new state s'
            Choose action a' from state s' derived from Q (e.g., ϵ-greedy)
            $\delta_t \leftarrow r + \gamma Q(s', a') - Q(s, a)$
            $E(s, a) \leftarrow E(s, a) + 1$
            $Q(s, a) \leftarrow Q(s, a) + \alpha \delta_t E(s, a)$
            $E(s, a) \leftarrow \gamma \lambda E(s, a)$
            $s \leftarrow s', a \leftarrow a'$
        end loop (when terminal state is reached)
    end loop

Scaling up: value function approximation

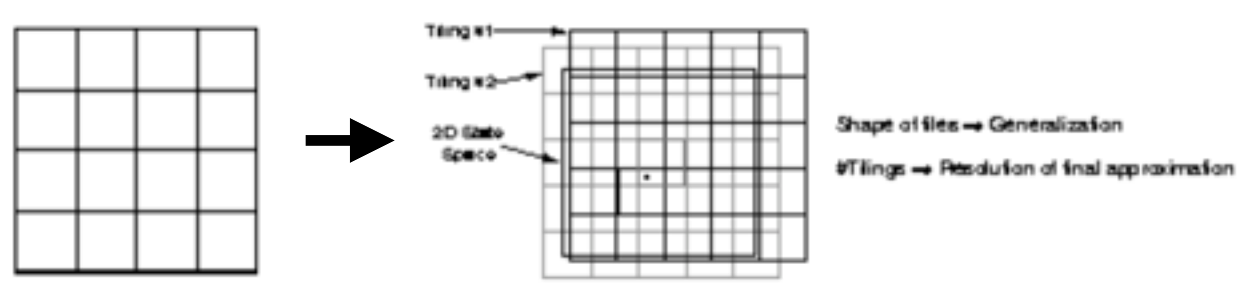
- How to deal with large state spaces, or continuous state spaces?
- Use function approximation: $\hat{q}(s, a, \mathbf{w}) \approx q_\pi(s, a)$
- E.g., linear combination of features, neural network, etc.

radial basis function



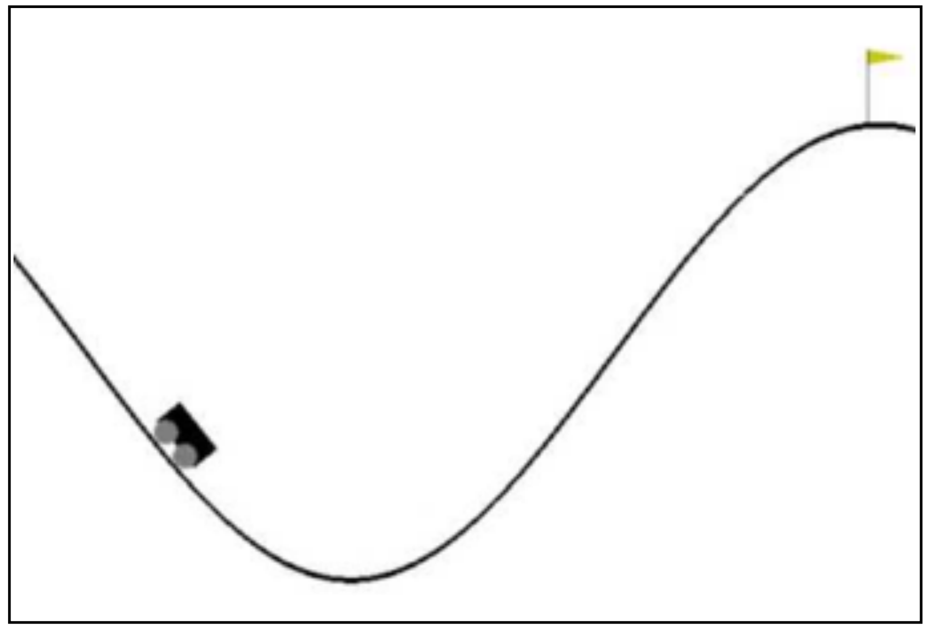


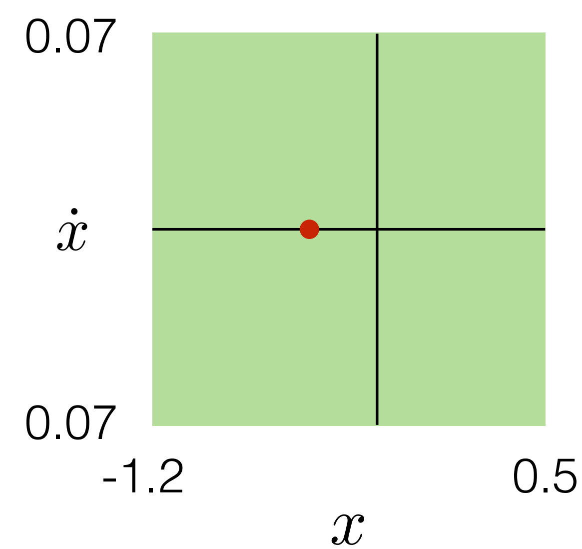
coarse coding



tile coding

Example: the mountain car problem



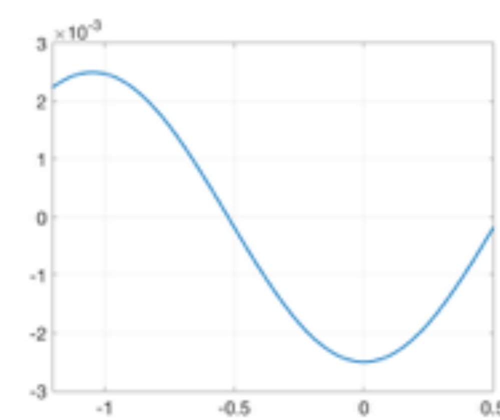


- State space: $s = (x, v), \; x : \text{position}, v : \text{velocity}$

- Actions: $\mathcal{A} = \{-1, 0, +1\}$

- Goal: get to the top of the hill

- Dynamics: $$\dot{x}_{t+1} = \text{bound}_{\dot{x}}[\dot{x}_t + 0.001a_t - 0.0025\cos(3x_t)]$$
$$x_{t+1} = \text{bound}_x[x_t + \dot{x}_{t+1}]$$
$$-1.2 \leq x_{t+1} \leq 0.5, -0.07 \leq \dot{x}_{t+1} \leq 0.07$$

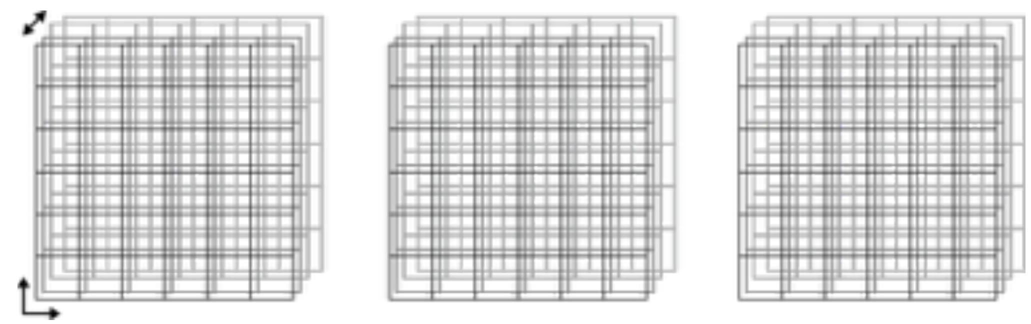


Linear value function approximation by stochastic gradient descent

- Represent state by a feature vector

$$\phi(S, A) = [\phi_1(S, A), ..., \phi_n(S, A)]^T$$

tile-coding



$$\mathcal{A} = \{-1, 0, 1\}$$

- State-action value function is a weighted combination of feature vectors

$$\hat{q}(S, A, \mathbf{w}) = \phi(S, A)^T \mathbf{w} = \sum_{j=1}^{n} \phi_j(S, A) w_j$$

- Objective function:

$$J(\mathbf{w}) = \mathbb{E}_\pi[(q_\pi(S, A) - \hat{q}(S, A, \mathbf{w}))^2] = \mathbb{E}_\pi[(q_\pi(S, A) - \phi(S, A^T)\mathbf{w})^2]$$

Linear value function approximation by stochastic gradient descent

$$J(\mathbf{w}) = \mathbb{E}_\pi[(q_\pi(S,A) - \hat{q}(S,A,\mathbf{w}))^2] = \mathbb{E}_\pi[(q_\pi(S,A) - \phi(S,A^T)\mathbf{w})^2]$$

Stochastic gradient descent (SGD): $\Delta\mathbf{w} = \alpha(q_\pi(S,A) - \hat{q}(S,A,\mathbf{w}))\phi(S,A)$

Problem: we don't know the function $q_\pi(S,A)$

TD: $\Delta\mathbf{w} = \alpha(R + \gamma\hat{q}(S',A',\mathbf{w}) - \hat{q}(S,A,\mathbf{w}))\phi(S,A)$

TD(λ): $\delta_t = R_{t+1} + \gamma\hat{q}(S_{t+1},A_{t+1},\mathbf{w}) - \hat{q}(S_t,A_t,\mathbf{w})$

$$E_t = \gamma\lambda E_{t-1} + \phi(S_t)$$

$$\Delta\mathbf{w} = \alpha\delta_t E_t$$

Learning the mountain car with SARSA(λ) and tile-coding function approximation

- **Programming: do question 6.2 of assignment**

Algorithm 11 SARSA(λ) Policy Evaluation with Function Approximation

procedure SARSA_λ_EVAL(γ, α, ntrials,λ)
  Initialize $w_j = 0$ for all features ϕ_j
  Initialize $E_j = 0$ for all features ϕ_j (eligibility traces)
  for loop: for $i = 1$ to ntrials (number of episodes)
    Initialize in starting state s
    Choose action a from state s derived from Q (e.g., ϵ-greedy)
    while loop: while terminal state has not been reached
      Take action a, observe reward r and new state s'
      Choose action a' from state s' derived from Q (e.g., ϵ-greedy)
      $\delta_t \leftarrow r + \gamma Q(s', a') - Q(s, a)$
      $E_j \leftarrow E_j + \phi_j(s, a)$ for all features $\phi_j(s, a)$
      $Q_j \leftarrow Q_j + \alpha \delta_t E_j$
      $s \leftarrow s', a \leftarrow a'$ (note that behavior policy is followed)
    end loop (when terminal state is reached)
  end loop

Performance and divergence

- Rich Sutton's deadly triad: the risk of divergence emerges when we combine these three things:

    1. Function approximation

    2. Bootstrapping (TD learning)

    3. Off-policy learning (Q-learning)

- One hacky solution: experience replay — see DQNs, AlphaGo

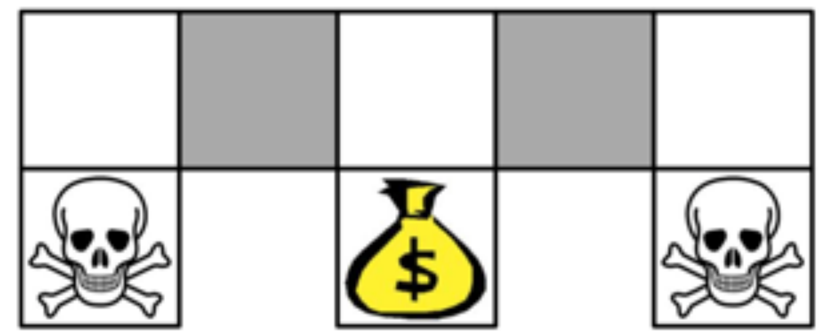
★We still don't have a good understanding of how these affect performance

Other methods

- Value-based RL: dynamic programming, Monte Carlo, temporal difference

- Value-based vs policy-based methods

$$v_*(s) = \max_\pi v_\pi(s)$$



- Policy-based and combined methods: policy gradient & actor-critic methods
    - Optimize the policy directly. Doesn't require having a value function at all, though one can be included.
    - Why? — policy may be simpler to represent, optimal policy may be stochastic (can't be done with value-based methods)

Back to neuroscience

- Looking for signatures of RL in the brain:
  - Model free — Dopamine prediction error signal (basal ganglia, VTA)
  - Model based — May involve spatial representation in the hippocampus and parts of cortex

Scaling up: RL toolboxes

Tensorflow agents

Pytorch - RL tutorials

OpenAI baselines and gym (on github)

Acknowledgements and more info

Sutton & Barto

David Silver
http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html