RISE
Reconfigurable and Intelligent Systems Engineering

An Introduction to Reinforcement Learning and Multi-arm Bandits

Explore-Exploit Dilemma

B. Ravindran

Reconfigurable and Intelligent Systems (RISE) Group

Department of Computer Science and Engineering

Indian Institute of Technology Madras

Carnegie Mellon
School of Computer Science

Deep Reinforcement Learning and Control

Markov Decision Processes

Lecture 2, CMU 10703

Katerina Fragkiadaki

Learning to Control

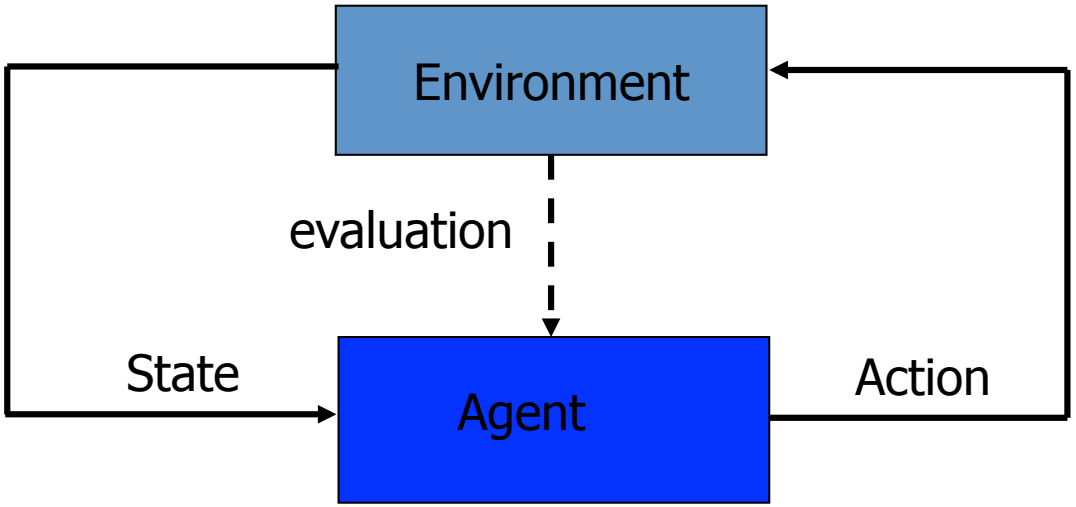
- Familiar models of machine learning
  - Supervised: Classification, Regression, etc.
  - Unsupervised: Clustering, Frequent patterns, etc.

- How did you learn to cycle?
  - Neither of the above
  - Trial and error!
  - Falling down hurts!

Reinforcement Learning

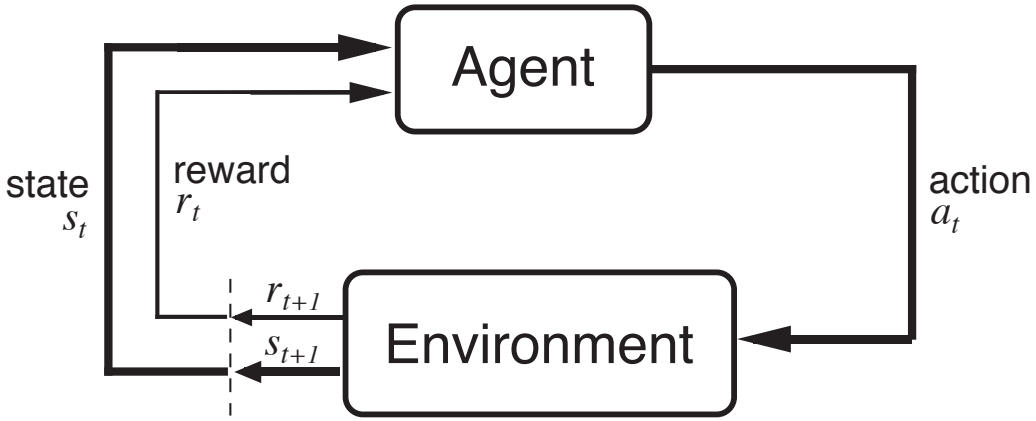
- A trial-and-error learning paradigm
  - Rewards and Punishments
- Not just an algorithm but a new paradigm in itself
- Learn about a system through interaction
- Inspired by behavioural psychology!

RL Framework



- Learn from close interaction
- Stochastic environment
- Noisy delayed scalar evaluation
- Maximize a measure of long term performance

The Agent-Environment Interface



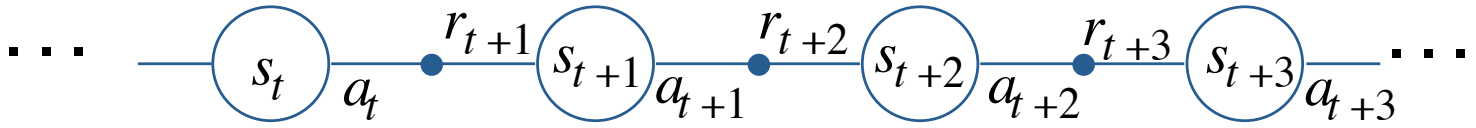
Agent and environment interact at discrete time steps: $t = 0, 1, 2, \ldots$

Agent observes state at step t: $s_t \in S$

produces action at step t : $a_t \in A(s_t)$

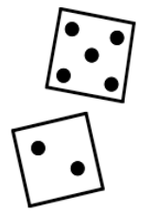
gets resulting reward: $r_{t+1} \in \Re$

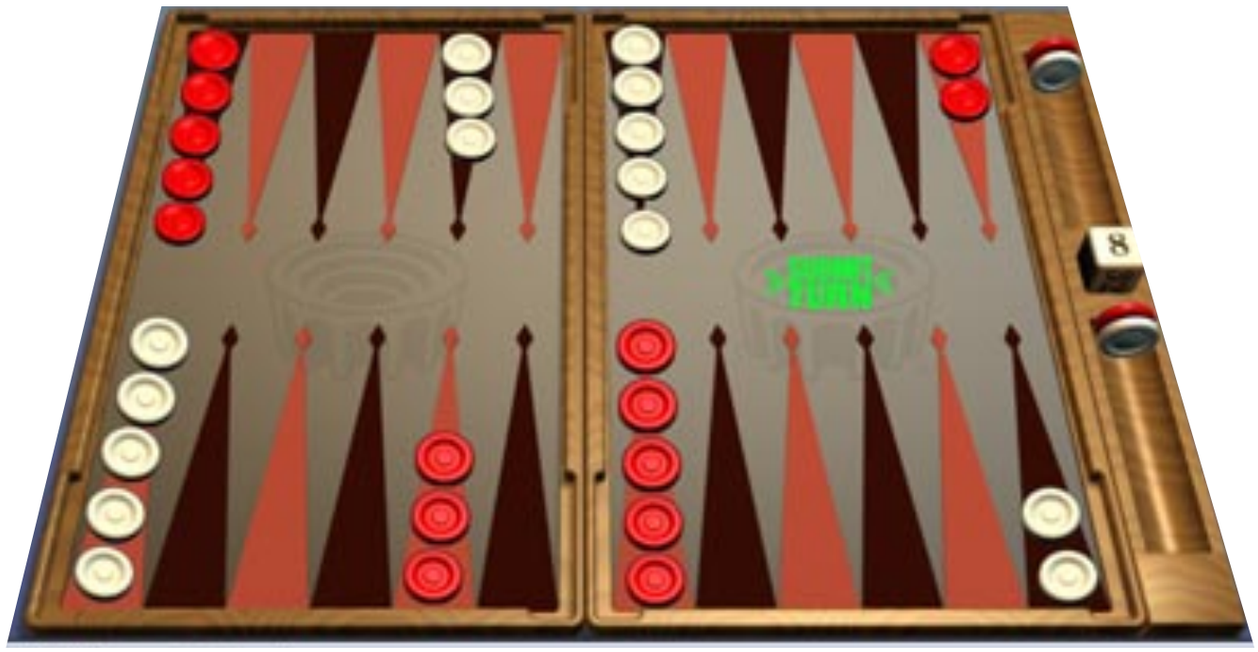
and resulting next state: s_{t+1}



Backgammon

- States: Configurations of the playing board (≈1020)
- Actions: Moves
- Rewards:
  - win: +1
  - lose: –1
  - else: 0



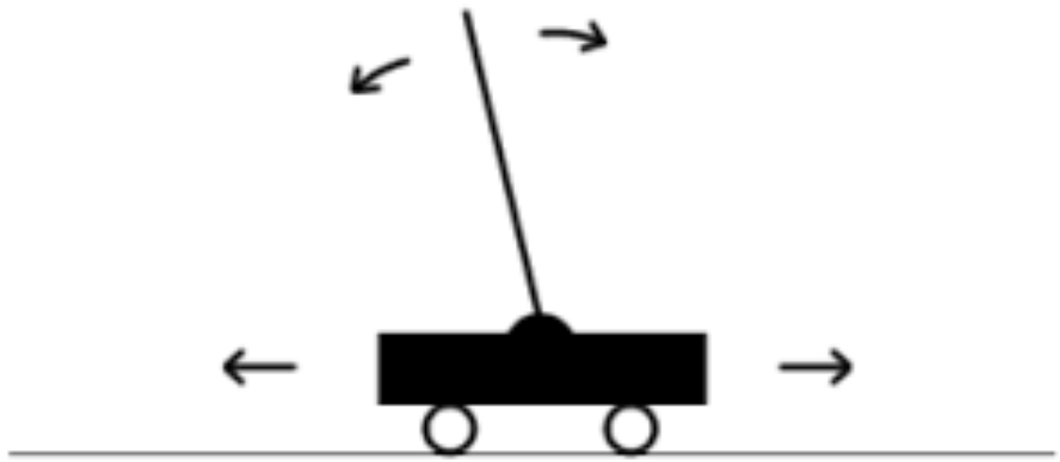


Visual Attention

- States: Road traffic, weather, time of day
- Actions: Visual glimpses from mirrors/cameras/front
- Rewards:
  - +1 safe driving, not over-tired
  - -1: honking from surrounding drivers

Cart Pole

- States: Pole angle and angular velocity
- Actions: Move left right
- Rewards:
  - 0 while balancing
  - -1 for imbalance



Single Pole Balancing

Markov Decision Process

A **Markov Decision Process** is a tuple $(\mathcal{S}, \mathcal{A}, T, r, \gamma)$

- $\mathcal{S}$ is a finite set of states
- $\mathcal{A}$ is a finite set of actions
- T is a state transition probability function

$$T(s'|s,a) = \mathbb{P}[S_{t+1} = s' | S_t = s, A_t = a]$$

- r is a reward function

$$r(s,a) = \mathbb{E}[R_{t+1} | S_t = s, A_t = a]$$

- γ is a discount factor $\gamma \in [0, 1]$

States

- A state captures whatever information is available to the agent at step t about its environment. The state can include immediate "sensations," highly processed sensations, and structures built up over time from sequences of sensations, memories etc.

- A state should summarize past sensations so as to retain all "essential" information, i.e., it should have the **Markov Property**:

$$\mathbb{P}[R_{t+1} = r, S_{t+1} = s' | S_0, A_0, R_1, ..., S_{t-1}, A_{t-1}, R_t, S_t, A_t] = \mathbb{P}[R_{t+1} = r, S_{t+1} = s' | S_t, A_t]$$

for all $s' \in \mathcal{S}, r \in \mathcal{R}$, and all histories

- We should be able to throw away the history once state is known

States

- A state captures whatever information is available to the agent at step t about its environment. The state can include immediate “sensations,” highly processed sensations, and structures built up over time from sequences of sensations, memories etc.
- Example: What would you expect to be the state information of a self driving car?

An agent cannot be blamed for missing information that is unknown, but for forgetting relevant information.

States

- A state captures whatever information is available to the agent at step t about its environment. The state can include immediate “sensations,” highly processed sensations, and structures built up over time from sequences of sensations, memories etc.
- How would you expect to be the state information of a vacuum-cleaner robot ?

Dynamics

- How the state changes given the actions of the agent
- Model based: dynamics are known or are estimated
- Model free: we do not know the dynamics of the MDP

Since in practice the dynamics are unknown, the state representation should be such that is easily predictable from neighboring states

Rewards

Definition: The *return* G_t is the total discounted reward from time-step t

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

- The objective in RL is to maximize long-term future reward
- That is, to choose A_t so as to maximize $R_{t+1}, R_{t+2}, R_{t+3}, \ldots$
- Episodic tasks - finite horizon VS continuous tasks - infinite horizon
- In episodic tasks we can consider undiscounted future rewards

Agent Learns a Policy

Definition: A policy π is a distribution over actions given states,

$$\pi(a|s) = \mathbb{P}[A_t = a|S_t = s]$$

- A policy fully defines the behavior of an agent
- MDP policies depend on the current state (not the history)
- i.e. Policies are stationary (time-independent)

  $A_t \sim \pi(\cdot|S_T), \forall t > 0$

Solving Markov Decision Processes

- Find the optimal policy
- Prediction: For a given policy, estimate value functions of states and states/action pairs
- Control: Estimate the value function of states and state/action pairs for the optimal policy.

Value Functions

	state values	action values
prediction	v_π	q_π
control	v_*	q_*

- Value functions measure the goodness of a particular state or state/action pair: how good is for the agent to be in a particular state or execute a particular action at a particular state. Of course that depends on the policy!
- Optimal value functions measure the best possible goodness of states or state/action pairs under any policy.

Value Functions are Cumulative Expected Rewards

Definition: The *state-value function* $\mathrm{v}_\pi(s)$ of an MDP is the expected return starting from state s, and then following policy π

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

The *action-value function* $q_\pi(s, a)$ is the expected return starting from state s, taking action a, and then following policy π

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

Optimal Value Functions are Best Achievable Cumulative Expected Rewards

- **Definition:** The *optimal state-value function* $\mathrm{v}_*(s)$ is the maximum value function over all policies

$$\mathrm{v}_*(s) = \max_\pi \mathrm{v}_\pi(s)$$

- The *optimal action-value function* $q_*(s, a)$ is the maximum action-value function over all policies

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

Value Functions are Cumulative Expected Rewards

- **Definition:** The *state-value function* $\mathrm{v}_\pi(s)$ of an MDP is the expected return starting from state s, and then following policy π

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

The *action-value function* $q_\pi(s, a)$ 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

$$
\begin{aligned}
v_\pi(s) &= \mathbb{E}_\pi[G_t \mid S_t = s] \\
&= \mathbb{E}_\pi\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots \mid S_t = s\right] \\
&= \mathbb{E}_\pi\left[R_{t+1} + \gamma\left(R_{t+2} + \gamma R_{t+3} + \ldots\right) \mid S_t = s\right] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s]
\end{aligned}
$$

The value function can be decomposed into two parts:

- Immediate reward R_{t+1}
- Discounted value of successor state $\gamma \mathrm{v}(S_{t+1})$

Bellman Expectation Equation

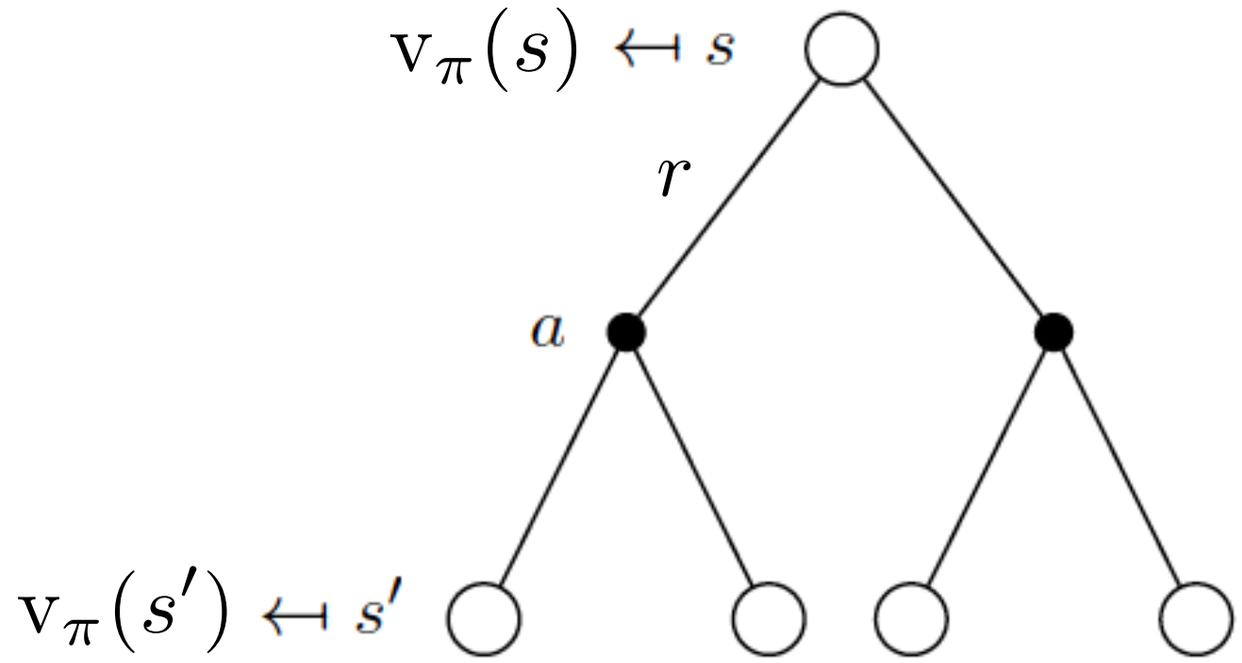
The state-value function can be decomposed into immediate reward plus discounted value of successor state,

$$v_\pi(s) = \mathbb{E}_\pi \left[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s \right]$$

The action-value function can similarly be decomposed,

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

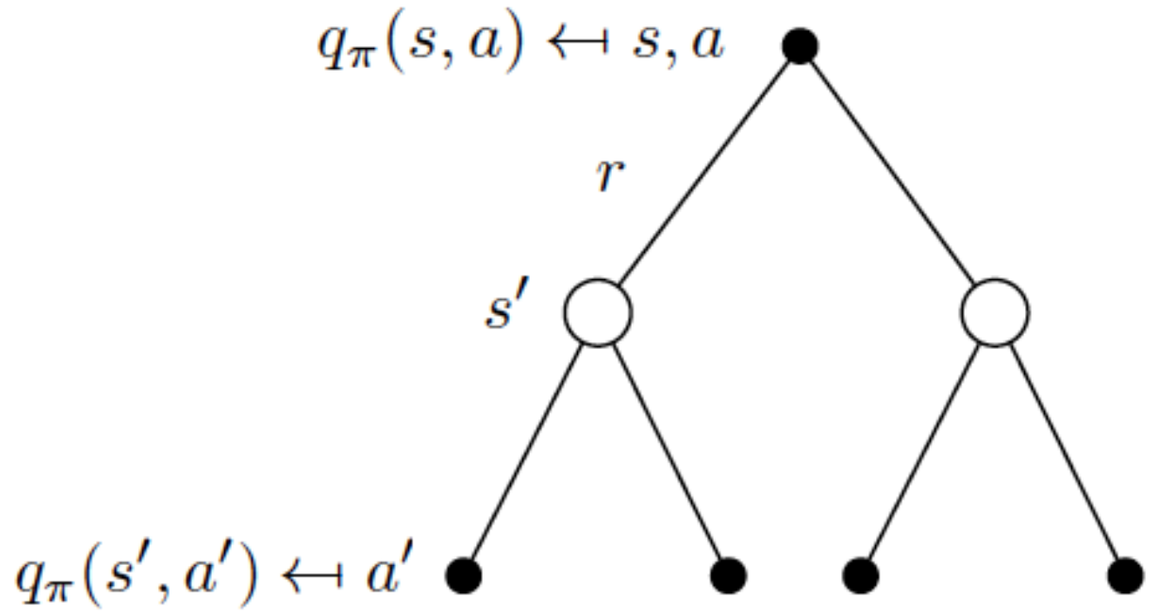
Looking Inside the Expectations



$$\mathrm{v}_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(r(s,a) + \gamma \sum_{s' \in S} T(s'|s,a) \mathrm{v}_\pi(s') \right)$$

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

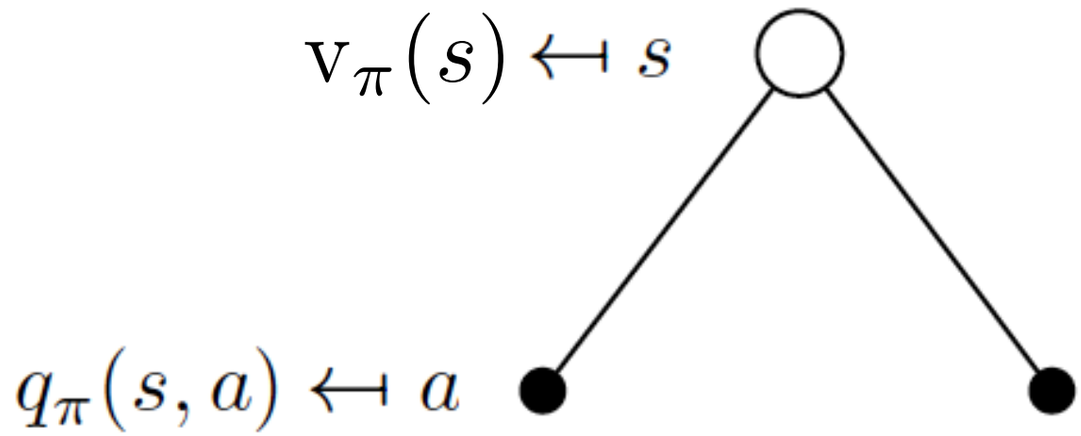
Looking Inside the Expectations



$$q_\pi(s,a) = r(s,a) + \gamma \sum_{s' \in S} T(s'|s,a) \sum_{a' \in \mathcal{A}} \pi(a'|s') q_\pi(s',a')$$

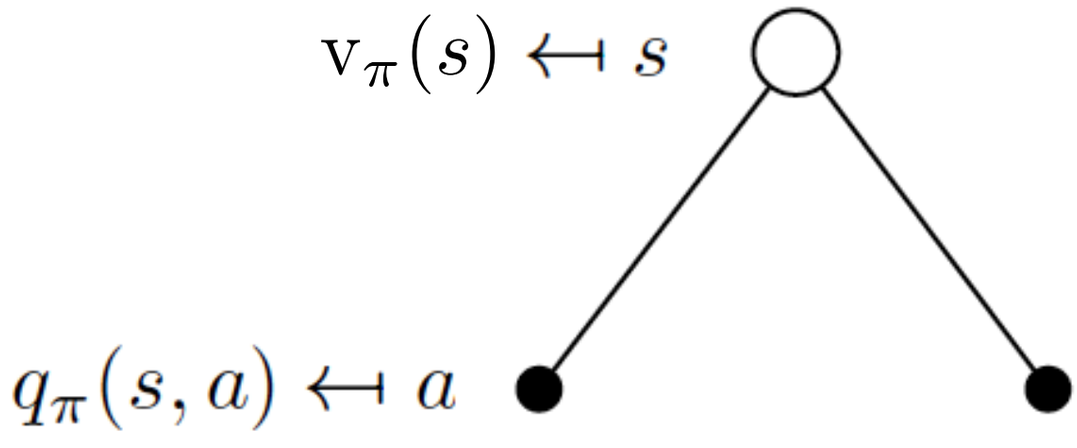
$$\mathrm{q}_\pi(s,a) = \mathbb{E}_\pi[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) | S_t = s, A_t = a]$$

State and State/Action Value Functions



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

State and State/Action Value Functions



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

$$q_\pi(s,a) = r(s,a) + \gamma \sum_{s' \in S} T(s'|s,a) \sum_{a' \in \mathcal{A}} \pi(a'|s') q_\pi(s',a')$$

$$q_\pi(s,a) = r(s,a) + \gamma \sum_{s' \in S} T(s'|s,a) \mathrm{v}_\pi(s')$$

Linear system of Equations

The Bellman expectation equation can be expressed concisely using the induced MRP,

$$v_\pi = \mathcal{R}^\pi + \gamma \mathcal{P}^\pi v_\pi$$

with direct solution

$$v_\pi = (I - \gamma \mathcal{P}^\pi)^{-1} \mathcal{R}^\pi$$

Optimal Value Functions

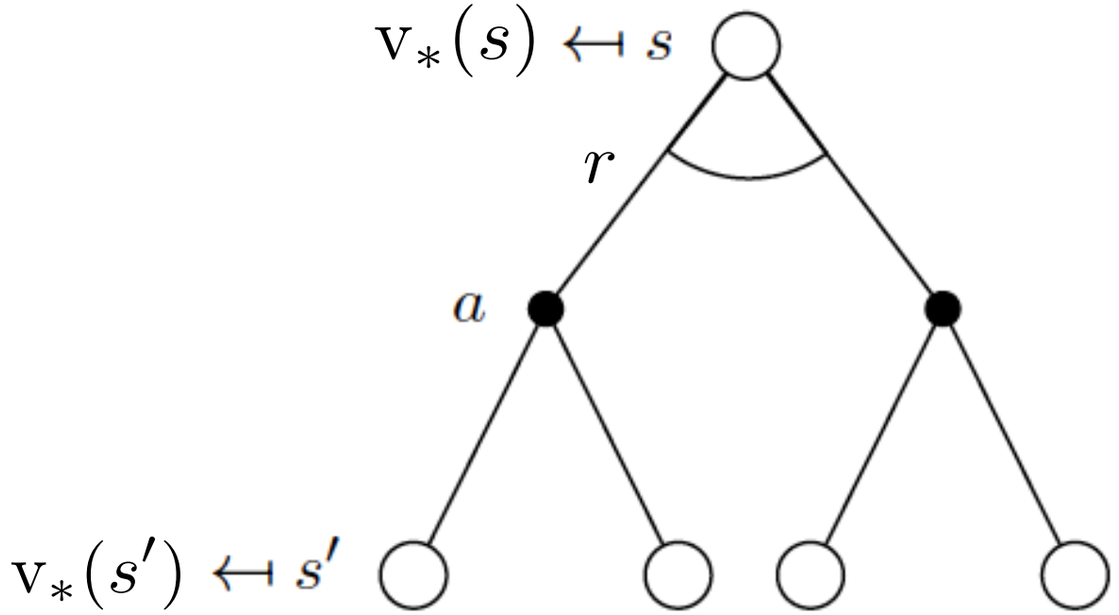
Definition: The *optimal state-value function* $\mathrm{v}_*(s)$ is the maximum value function over all policies

$$\mathrm{v}_*(s) = \max_\pi \mathrm{v}_\pi(s)$$

The *optimal action-value function* $q_*(s,a)$ is the maximum action-value function over all policies

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

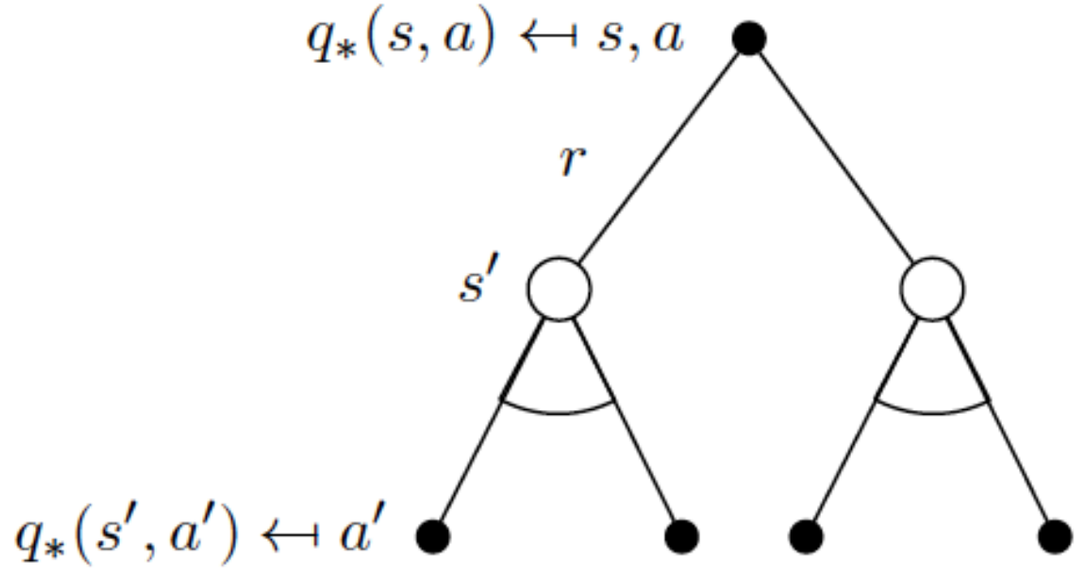
Bellman Optimality Equations for State Value Functions



$$\mathrm{v}_*(s) \leftarrow \max_{a \in \mathcal{A}} r(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s'|s,a)\mathrm{v}_*(s')$$

Principle of Optimality: An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision. (See Bellman, 1957, Chap. III.3).

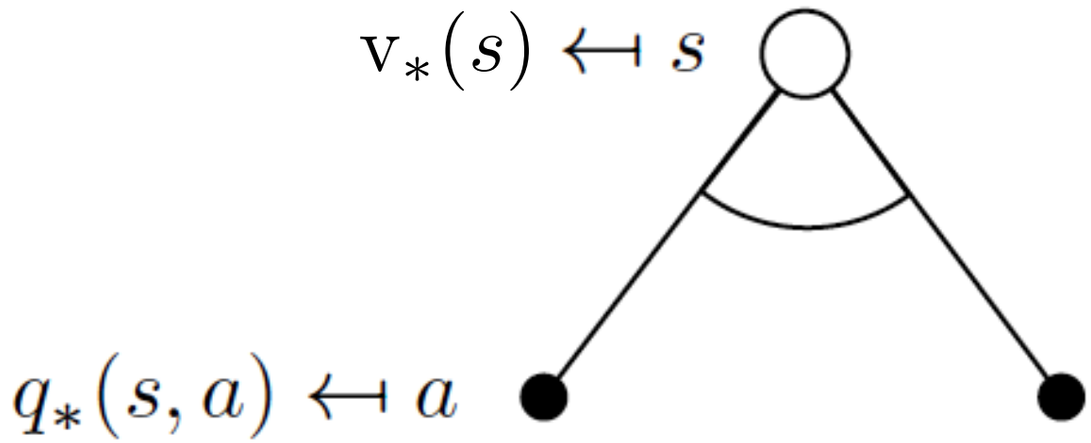
Bellman Optimality Equations for State/Action Value Functions



$$q_*(s,a) = r(s,a) + \gamma \sum_{s' \in S} T(s'|s,a) \max_{a'} q_*(s',a')$$

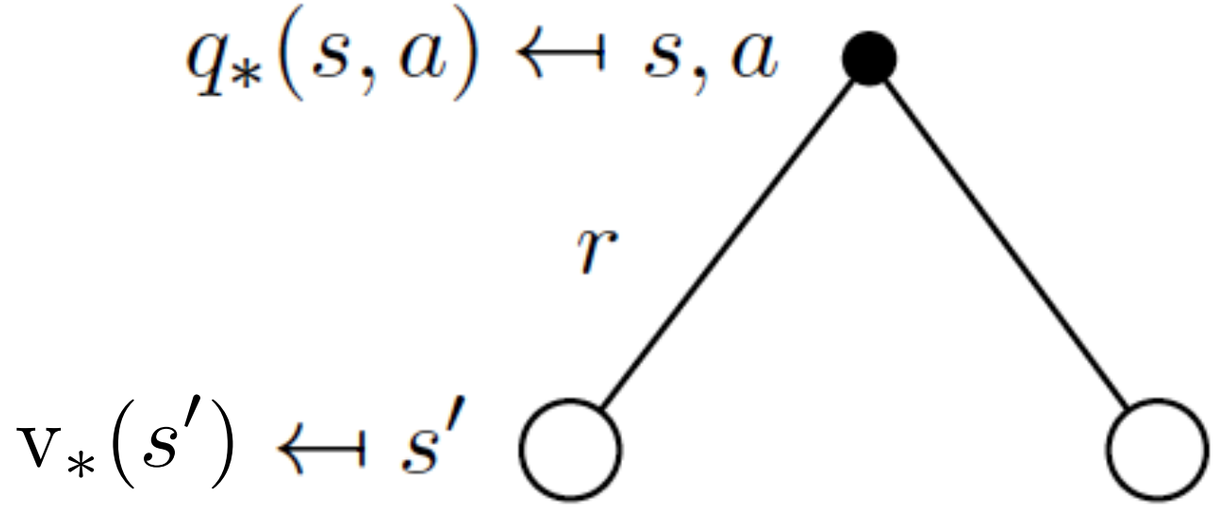
Relating Optimal State and Action Value Functions

The optimal value functions are recursively related by the Bellman optimality equations:



$$\mathrm{v}_*(s) = \max_a q_*(s,a)$$

Relating Optimal State and Action Value Functions



$$q_*(s,a) = r(s,a) + \gamma \sum_{s' \in S} T(s'|s,a)\mathrm{v}_*(s')$$

Optimal Policy

Define a partial ordering over policies

$$\pi \geq \pi' \text{ if } \mathrm{v}_\pi(s) \geq \mathrm{v}_{\pi'}(s), \forall s$$

Theorem: For any Markov Decision Process

- There exists an optimal policy π_* that is better than or equal to all other policies, $\pi_* \geq \pi, \forall \pi$
- All optimal policies achieve the optimal value function, $\mathrm{v}_{\pi_*}(s) = \mathrm{v}_*(s)$
- All optimal policies achieve the optimal action-value function, $q_{\pi_*}(s, a) = q_*(s, a)$

From Optimal State Value Functions to Optimal Policies

- An optimal policy can be found from $\mathrm{v}_*(s)$ and the model dynamics using one step look ahead, that is, acting greedily w.r.t. $\mathrm{v}_*(s)$

$$\mathrm{v}_*(s) = \max_a r(s,a) + \gamma \sum_{s' \in S} T(s'|s,a)\mathrm{v}_*(s')$$

From Optimal Action Value Functions to Optimal Policies

An optimal policy can be found by maximizing over $q_*(s, a)$

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

- There is always a deterministic optimal policy for any MDP
- If we know $q_*(s, a)$ we immediately have the optimal policy

Solving the Bellman Optimality Equation

- Finding an optimal policy by solving the Bellman Optimality Equation requires the following:
  - accurate knowledge of environment dynamics;
  - we have enough space and time to do the computation;
  - the Markov Property.
- How much space and time do we need?
  - polynomial in number of states (tabular methods)
  - BUT, number of states is often huge
  - So exhaustive sweeps of the state space are not possible

Solving the Bellman Optimality Equation

- We usually have to settle for approximations.
- Approximate dynamic programming has been introduced by D. P. Bertsekas and J. N. Tsitsiklis with the use of artificial neural networks for approximating the Bellman function. This is an effective mitigation strategy for reducing the impact of dimensionality by replacing the memorization of the complete function mapping for the whole space domain with the memorization of the sole neural network parameters.

Approximation and Reinforcement Learning

- RL methods: Approximating Bellman optimality equations
- Balancing reward accumulation and system identification (model learning) in case of unknown dynamics
- The on-line nature of reinforcement learning makes it possible to approximate optimal policies in ways that put more effort into learning to make good decisions for frequently encountered states, at the expense of less effort for infrequently encountered states. This is not the case, e.g., in control.

Summary

- Markov Decision Processes
- Value functions and Optimal Value functions
- Bellman Equations

So far finite MDPs with known dynamics