Reinforcement Learning

Sindhu P R
Stochastic Systems Lab

Undergraduate Summer School-2016
Dept. of Computer Science and Automation
Indian Institute of Science

July 5, 2016

Checkmate !

Animal Foraging

- Capability of taking decisions under different scenarios seen in animals and especially birds
- Example: Bird foraging

Sequential Decision Making

Features

- Decision making in stages
- Step taken affects future stages
- Rationality: An "entity" takes decisions to maximize a pre-defined utility
- Aim: Find a sequence of decisions to achieve the goal

Can we mimic this capability in algorithms ?

Sequential Decision Making

Features

- Decision making in stages
- Step taken affects future stages
- Rationality: An "entity" takes decisions to maximize a pre-defined utility
- Aim: Find a sequence of decisions to achieve the goal

Can we mimic this capability in algorithms ?

Sequential Decision Making

Features

- Decision making in stages
- Step taken affects future stages
- Rationality: An "entity" takes decisions to maximize a pre-defined utility
- Aim: Find a sequence of decisions to achieve the goal

Can we mimic this capability in algorithms ?

Sequential Decision Making

Features

- Decision making in stages
- Step taken affects future stages
- Rationality: An "entity" takes decisions to maximize a pre-defined utility
- Aim: Find a sequence of decisions to achieve the goal

Can we mimic this capability in algorithms ?

Sequential Decision Making

Features

- Decision making in stages
- Step taken affects future stages
- Rationality: An "entity" takes decisions to maximize a pre-defined utility
- Aim: Find a sequence of decisions to achieve the goal

Can we mimic this capability in algorithms ?

Sequential Decision Making

Features

- Decision making in stages
- Step taken affects future stages
- Rationality: An "entity" takes decisions to maximize a pre-defined utility
- Aim: Find a sequence of decisions to achieve the goal

Can we mimic this capability in algorithms ?

Mathematical Framework

Problem Modeling

- Agent (learner): Entity that observes and acts
- Components
- Dynamics

Mathematical Framework

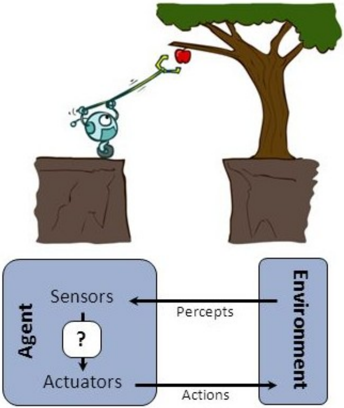
Problem Modeling

- Agent (learner): Entity that observes and acts
- Components:
- Dynamics:

Mathematical Framework

Problem Modeling

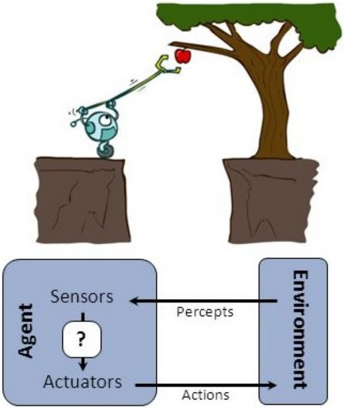
- Agent (learner): Entity that observes and acts
- Components:



Mathematical Framework

Problem Modeling

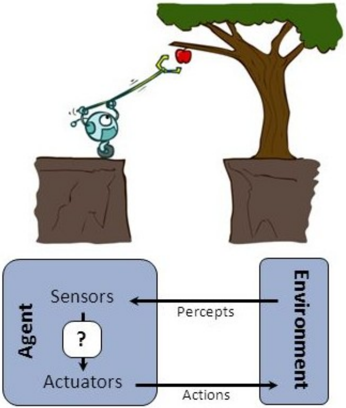
- Agent (learner): Entity that observes and acts
- Components:
  - Set of states (or configurations) $s \in S$
  - Feasible actions (or decisions) $a(s) \in A$ in every state s
- Dynamics:
  - Action changes state
  - Obtain reward for every action, $R(s, a, s_{next})$



Mathematical Framework

Problem Modeling

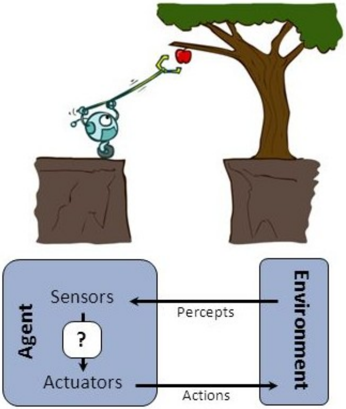
- Agent (learner): Entity that observes and acts
- Components:
  - Set of states (or configurations) $s \in S$
  - Feasible actions (or decisions) $a(s) \in A$ in every state s
- Dynamics:



Mathematical Framework

Problem Modeling

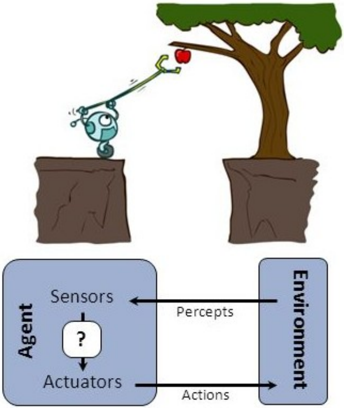
- Agent (learner): Entity that observes and acts
- Components:
  - Set of states (or configurations) $s \in S$
  - Feasible actions (or decisions) $a(s) \in A$ in every state s
- Dynamics:
  - Action changes state
  - Obtain reward for every action, $R(s, a, s_{next})$



Mathematical Framework

Problem Modeling

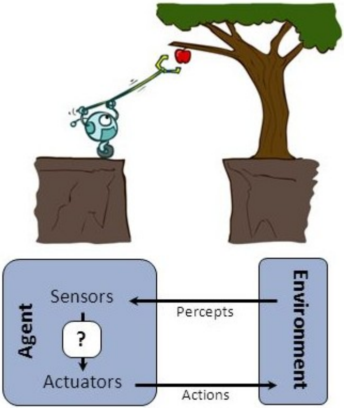
- Agent (learner): Entity that observes and acts
- Components:
  - Set of states (or configurations) $s \in S$
  - Feasible actions (or decisions) $a(s) \in A$ in every state s
- Dynamics:
  - Action changes state
  - Obtain reward for every action, $R(s, a, s_{next})$



Mathematical Framework

Problem Modeling

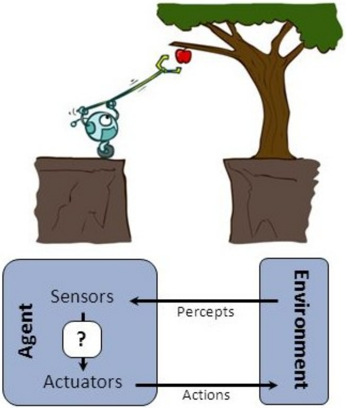
- Agent (learner): Entity that observes and acts
- Components:
  - Set of states (or configurations) $s \in S$
  - Feasible actions (or decisions) $a(s) \in A$ in every state s
- Dynamics:
  - Action changes state
  - Obtain reward for every action, $R(s, a, s_{next})$



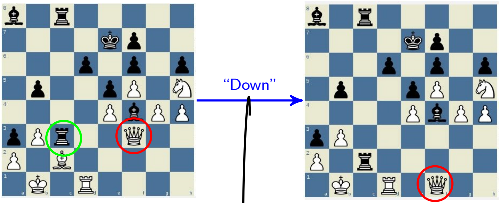
Mathematical Framework

Problem Modeling

- Agent (learner): Entity that observes and acts
- Components:
  - Set of states (or configurations) $s \in S$
  - Feasible actions (or decisions) $a(s) \in A$ in every state s
- Dynamics:
  - Action changes state
  - Obtain reward for every action, $R(s, a, s_{next})$



Example: Chess



State s_1

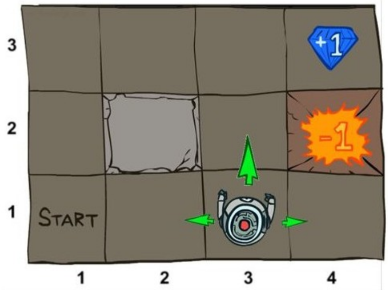
State s_2

$$R\left(s_1, \text{"}Down'', s_2\right) = -1$$

Optimal Decision making under uncertainty

Example:

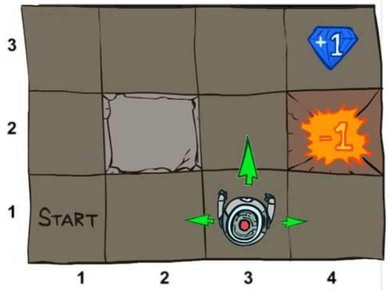
- Maze-like problem:
  - Agent lives in a grid
  - Wall blocks the agent's path
- Noisy movement: actions do not always go as planned:
- Agent receives small rewards each time step and big rewards at the end
- Goal: Maximize expected sum of rewards



Optimal Decision making under uncertainty

Example:

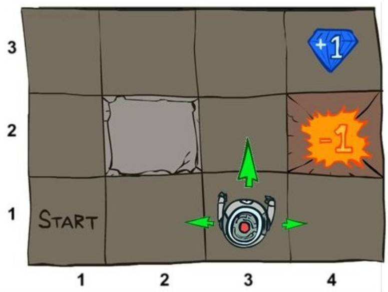
- Maze-like problem:
  - Agent lives in a grid
  - Wall blocks the agent's path
- Noisy movement: actions do not always go as planned:
  - 90% of the time action North takes the agent north
  - 10% of the time, North takes the agent west;
- Agent receives small rewards each time step and big rewards at the end
- Goal: Maximize expected sum of rewards



Optimal Decision making under uncertainty

Example:

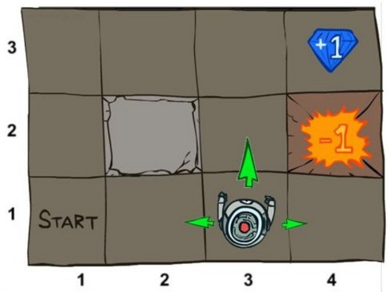
- Maze-like problem:
  - Agent lives in a grid
  - Wall blocks the agent's path
- Noisy movement: actions do not always go as planned:
  - 90% of the time action North takes the agent north
- Agent receives small rewards each time step and big rewards at the end
- Goal: Maximize expected sum of rewards



Optimal Decision making under uncertainty

Example:

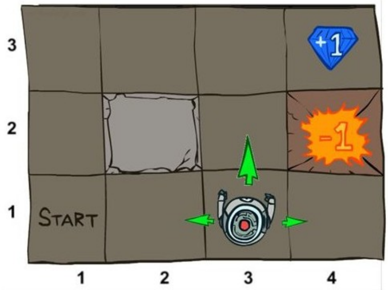
- Maze-like problem:
  - Agent lives in a grid
  - Wall blocks the agent's path
- Noisy movement: actions do not always go as planned:
  - 90% of the time action North takes the agent north
  - 10% of the time, North takes the agent west
- Agent receives small rewards each time step and big rewards at the end
- Goal: Maximize expected sum of rewards



Optimal Decision making under uncertainty

Example:

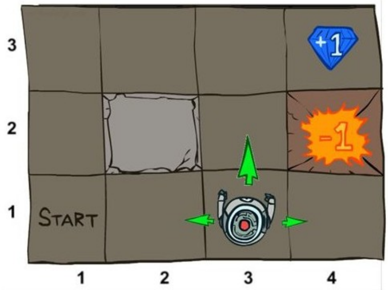
- Maze-like problem:
  - Agent lives in a grid
  - Wall blocks the agent's path
- Noisy movement: actions do not always go as planned:
  - 90% of the time action North takes the agent north
  - 10% of the time, North takes the agent west
- Agent receives small rewards each time step and big rewards at the end
- Goal: Maximize expected sum of rewards



Optimal Decision making under uncertainty

Example:

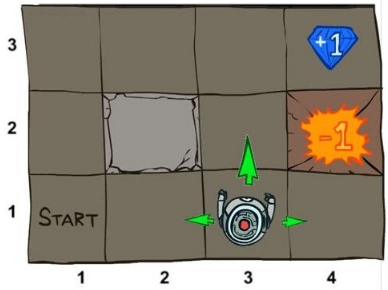
- Maze-like problem:
  - Agent lives in a grid
  - Wall blocks the agent's path
- Noisy movement: actions do not always go as planned:
  - 90% of the time action North takes the agent north
  - 10% of the time, North takes the agent west
- Agent receives small rewards each time step and big rewards at the end
- Goal: Maximize expected sum of rewards



Optimal Decision making under uncertainty

Example:

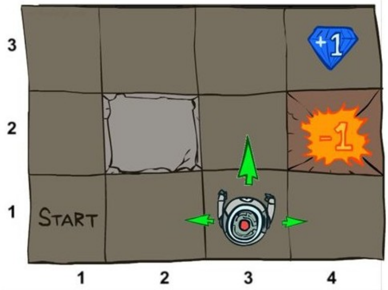
- Maze-like problem:
  - Agent lives in a grid
  - Wall blocks the agent's path
- Noisy movement: actions do not always go as planned:
  - 90% of the time action North takes the agent north
  - 10% of the time, North takes the agent west
- Agent receives small rewards each time step and big rewards at the end
- Goal: Maximize expected sum of rewards



Optimal Decision making under uncertainty

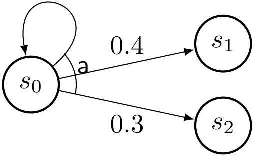
Example:

- Maze-like problem:
  - Agent lives in a grid
  - Wall blocks the agent's path
- Noisy movement: actions do not always go as planned:
  - 90% of the time action North takes the agent north
  - 10% of the time, North takes the agent west
- Agent receives small rewards each time step and big rewards at the end
- Goal: Maximize expected sum of rewards



Markov Decision Process

- Model sequential decision making problem under uncertainty as a Markov Decision Process (MDP)
- MDP $= <$ set of states S, actions A, reward function R and a transition function $P >$
- $P(s, a, s_{next})$: Probability of moving to state s_{next} from state s under action a



Policy

- In deterministic problems, we want a sequence of actions from start to goal
- For MDP, we want an optimal policy $\pi^* : S \to A$
- Policy π gives an action for each state at each time
- Optimal Policy: Gives maximum expected sum of rewards (if followed)

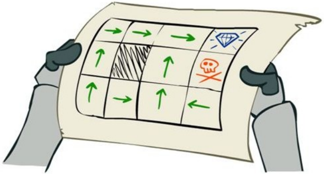
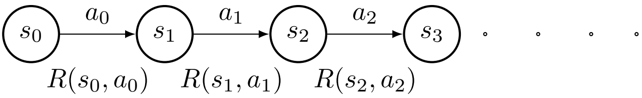


Figure: Optimal policy when $R = -3$ for all s

Value Functions



- State Value Function V^π: indicates how good or bad a state is under policy π

  $$V^\pi(s_0) = \mathbb{E}\left[R(s_0, \pi(s_0)) + \gamma R(s_1, \pi(s_1)) + \gamma^2 R(s_2, \pi(s_2)) + \ldots\right]$$

- Action Value Function Q^π: indicates how good or bad an action is for a state when policy π is followed

  $$Q^\pi(s_0, a_0) = \mathbb{E}\left[R(s_0, a_0) + \gamma R(s_1, \pi(s_1)) + \gamma^2 R(s_2, \pi(s_2)) + \ldots\right]$$

- Optimal Value Function: $V^{\pi^*} : S \rightarrow \mathrm{R}$

Dynamic Programming

- V^{π^*}: Computed using iterative dynamic programming principle
- At iteration k, we get an estimate V_k of V^{π^*}

$$V_k(s) = \max_a P(s, a, s_{next}) \left[R(s, a, s') + \gamma V_k(s_{next}) \right]$$
$$= \max_a Q_k(s, a)$$

- $V_{k+1}(s) \leftarrow V_k(s) \ \forall s \in S$
- $V_k \rightarrow V^{\pi^*}$ as $k \rightarrow \infty$
- Why discounting (γ)?: Prefer rewards now to rewards later

Reinforcement Learning

Twist: P ✗, R ✗

- We do not know which states are good
- No knowledge what actions do

Basic Idea:

- Observe the state and take an action
- Receive feedback in the form of rewards
- Must (learn to) act so as to maximize expected sum of discounted rewards

Reinforcement Learning

Twist: P ✗, R ✗

- We do not know which states are good
- No knowledge what actions do

Basic Idea:

- Observe the state and take an action
- Receive feedback in the form of rewards
- Must (learn to) act so as to maximize expected sum of discounted rewards

Reinforcement Learning

Twist: P ✗, R ✗

- We do not know which states are good
- No knowledge what actions do

Basic Idea:

- Observe the state and take an action
- Receive feedback in the form of rewards
- Must (learn to) act so as to maximize expected sum of discounted rewards

Reinforcement Learning

Twist: P ✗, R ✗

- We do not know which states are good
- No knowledge what actions do

Basic Idea:

- Observe the state and take an action
- Receive feedback in the form of rewards
- Must (learn to) act so as to maximize expected sum of discounted rewards

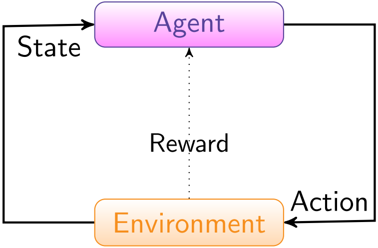
Reinforcement Learning

Twist: P ✗, R ✗

- We do not know which states are good
- No knowledge what actions do

Basic Idea:

- Observe the state and take an action
- Receive feedback in the form of rewards
- Must (learn to) act so as to maximize expected sum of discounted rewards



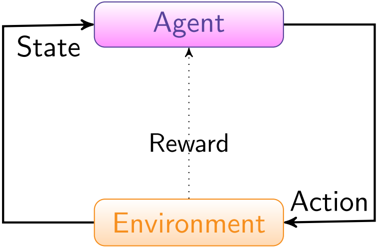
Reinforcement Learning

Twist: P ✗, R ✗

- We do not know which states are good
- No knowledge what actions do

Basic Idea:

- Observe the state and take an action
- Receive feedback in the form of rewards
- Must (learn to) act so as to maximize expected sum of discounted rewards



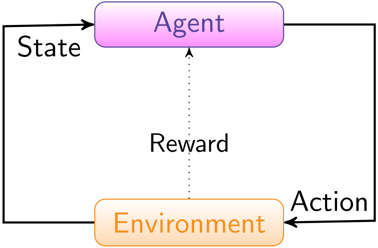
Reinforcement Learning

Twist: P ✗, R ✗

- We do not know which states are good
- No knowledge what actions do

Basic Idea:

- Observe the state and take an action
- Receive feedback in the form of rewards
- Must (learn to) act so as to maximize expected sum of discounted rewards



Reinforcement Learning

- Learning is based on observed samples of outcomes !
- Still assume a Markov Decision Process:
  - A set of states $s \in S$
  - A set of actions $a \in A$
  - States evolve according to $T(s, a, s')$
  - Agent receives reward according to $R(s, a, s')$
- Still looking for a policy $\pi(\cdot)$
- Trial-and-error: Must try actions at all states to learn

Reinforcement Learning

- Learning is based on observed samples of outcomes !
- Still assume a Markov Decision Process:
  - A set of states $s \in S$
  - A set of actions $a \in \mathcal{A}$
  - States evolve according to [illegible]
  - Agent receives reward according to [illegible]
- Still looking for a policy $\pi(\cdot)$
- Trial-and-error: Must try actions at all states to learn

Reinforcement Learning

- Learning is based on observed samples of outcomes !
- Still assume a Markov Decision Process:
  - A set of states $s \in S$
  - A set of actions $a \in A$
  - States evolve according to $P(s, a, s')$
  - Agent receives reward according to $R(s, a, s')$
- Still looking for a policy $\pi(\cdot)$
- Trial-and-error: Must try actions at all states to learn

Reinforcement Learning

- Learning is based on observed samples of outcomes !
- Still assume a Markov Decision Process:
  - A set of states $s \in S$
  - A set of actions $a \in A$
  - States evolve according to $P(s, a, s')$
  - Agent receives reward according to $R(s, a, s')$
- Still looking for a policy $\pi(\cdot)$
- Trial-and-error: Must try actions at all states to learn

Reinforcement Learning

- Learning is based on observed samples of outcomes !
- Still assume a Markov Decision Process:
  - A set of states $s \in S$
  - A set of actions $a \in A$
  - States evolve according to $P(s, a, s')$
  - Agent receives reward according to $R(s, a, s')$
- Still looking for a policy $\pi(\cdot)$
- Trial-and-error: Must try actions at all states to learn

Reinforcement Learning

- Learning is based on observed samples of outcomes !
- Still assume a Markov Decision Process:
  - A set of states $s \in S$
  - A set of actions $a \in A$
  - States evolve according to $P(s, a, s')$
  - Agent receives reward according to $R(s, a, s')$
- Still looking for a policy $\pi(\cdot)$
- Trial-and-error: Must try actions at all states to learn

Reinforcement Learning

- Learning is based on observed samples of outcomes !
- Still assume a Markov Decision Process:
  - A set of states $s \in S$
  - A set of actions $a \in A$
  - States evolve according to $P(s, a, s')$
  - Agent receives reward according to $R(s, a, s')$
- Still looking for a policy $\pi(\cdot)$
- Trial-and-error: Must try actions at all states to learn

Reinforcement Learning

- Learning is based on observed samples of outcomes !
- Still assume a Markov Decision Process:
  - A set of states $s \in S$
  - A set of actions $a \in A$
  - States evolve according to $P(s, a, s')$
  - Agent receives reward according to $R(s, a, s')$
- Still looking for a policy $\pi(\cdot)$
- Trial-and-error: Must try actions at all states to learn

Reinforcement Learning

- Learning is based on observed samples of outcomes !
- Still assume a Markov Decision Process:
  - A set of states $s \in S$
  - A set of actions $a \in A$
  - States evolve according to $P(s, a, s')$
  - Agent receives reward according to $R(s, a, s')$
- Still looking for a policy $\pi(\cdot)$
- Trial-and-error: Must try actions at all states to learn

Learning without P and R

$$V_k(s) = \max_a \cancel{P(s, a, s_{next})} \left[\cancel{R(s, a, s')} + \gamma V_k(s_{next})\right]$$

- Idea: Obtain R from samples
- Receive sample (s, a, s_{next}, r)
  Training:
- Initialize $Q_0(s, a) = 0$, $\forall (s, a)$ pairs
- Compute estimate Q_k of Q^{π^*} iteratively using many such samples

$$Q_k(s, a) = (1 - \alpha) Q_k(s, a) + \alpha \left[r + \gamma \max_b Q_k(s_{next}, b) \right]$$

- As $k \to \infty$, $Q_k \to Q^{\pi^*}$
- Good action for a state s is one that has highest Q value

RL in Action - Miniature Helicopter Control

Challenges in Helicopter Control

- Unstable
- Complicated dynamics - air flow, blade dynamics
- Noisy(inexact) estimates of position, orientation, velocity

MDP Modeling

- State $s =$ (Position, orientation, velocity, angular velocities)
- Action $a =$ (Movement direction,acceleration)
- Reward function is quadratic based on change in position and orientation

Summary

- RL is used for optimal sequential decision making under uncertainty
- Control without human intervention
- Applications: Engineering, Artificial Intelligence
- Some other applications: Traffic light control, control of drones

Thank You