Artificial Intelligence - Introduction to Reinforcement Learning

ENSISA 2A

Ali El Hadi Ismail Fawaz

ENSISA, Université Haute-Alsace

October 14, 2024

Une école d'ingénieurs de l'Université de Haute-Alsace

Overview

Overview

In this course you will learn:

Overview

In this course you will learn:

- What is a Reinforcement Learning model, i.e. a Sequential Decision Model and its different components

Overview

In this course you will learn:

- What is a Reinforcement Learning model, i.e. a Sequential Decision Model and its different components
- What is a Markov Decision Model and its properties

Overview

In this course you will learn:

- What is a Reinforcement Learning model, i.e. a Sequential Decision Model and its different components
- What is a Markov Decision Model and its properties
- How to solve model-based Reinforcement Learning models using Dynamic Programming

Overview

In this course you will learn:

- What is a Reinforcement Learning model, i.e. a Sequential Decision Model and its different components
- What is a Markov Decision Model and its properties
- How to solve model-based Reinforcement Learning models using Dynamic Programming
- How to solve model-free Reinforcement Learning models using Incremental On-Off policy algorithms

Overview

In this course you will learn:

- What is a Reinforcement Learning model, i.e. a Sequential Decision Model and its different components
- What is a Markov Decision Model and its properties
- How to solve model-based Reinforcement Learning models using Dynamic Programming
- How to solve model-free Reinforcement Learning models using Incremental On-Off policy algorithms
- Course and slides are based on:

Overview

In this course you will learn:

- What is a Reinforcement Learning model, i.e. a Sequential Decision Model and its different components
- What is a Markov Decision Model and its properties
- How to solve model-based Reinforcement Learning models using Dynamic Programming
- How to solve model-free Reinforcement Learning models using Incremental On-Off policy algorithms
- Course and slides are based on:
  - An Introduction to Reinforcement Learning, Sutton and Barto, 2nd Edition, 2018: Chapters 5, 6, 7

Overview

In this course you will learn:

- What is a Reinforcement Learning model, i.e. a Sequential Decision Model and its different components
- What is a Markov Decision Model and its properties
- How to solve model-based Reinforcement Learning models using Dynamic Programming
- How to solve model-free Reinforcement Learning models using Incremental On-Off policy algorithms
- Course and slides are based on:
  - An Introduction to Reinforcement Learning, Sutton and Barto, 2nd Edition, 2018: Chapters 5, 6, 7
  - Slides of Dr. Mireille Sarkiss, Telecom SudParis, Institut Polytechnique de Paris, Lectures 1-2

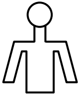
Sequential Decision Making

Sequential Decision Making

Suppose the following scenario

Sequential Decision Making

Suppose the following scenario

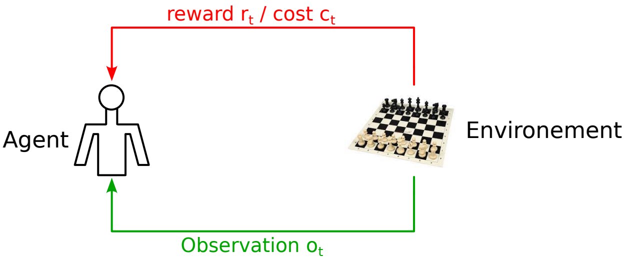
Agent 

 Environement

An agent wants to learn an environment's behavior, in this case, a game of chess

Sequential Decision Making

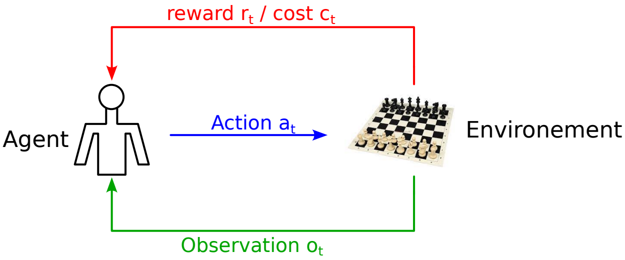
Suppose the following scenario



The environment reports a reward r_t or cost c_t to the agent as well as an observation o_t.

Sequential Decision Making

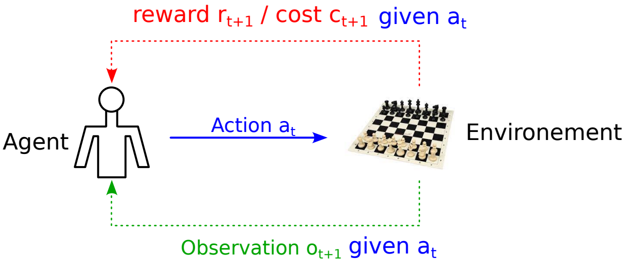
Suppose the following scenario



The agent then takes a decision given the information provided previously and applies the action a_t.

Sequential Decision Making

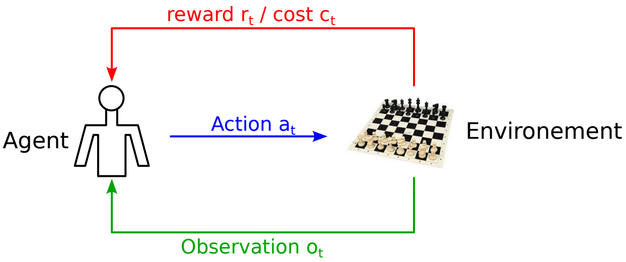
Suppose the following scenario



The environment then reports, given action a_t, an observation o_{t+1} and a reward r_{t+1} or cost c_{t+1}.

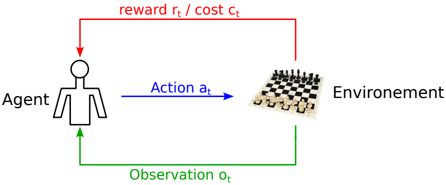
Sequential Decision Making

Suppose the following scenario

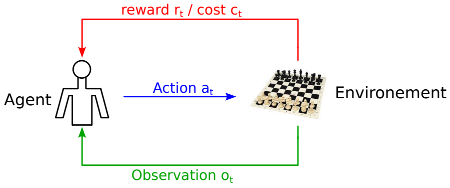


The goal of the agent is to choose the best actions that maximizes or minimizes the reward or cost.

History and State



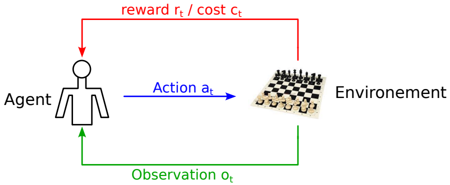
History and State



- History: sequence of past observation, actions and rewards up to the agent's decision time t

$$h_t = (o_1, r_1, a_1), (o_2, r_2, a_2), \ldots, (o_t, r_t)$$

History and State

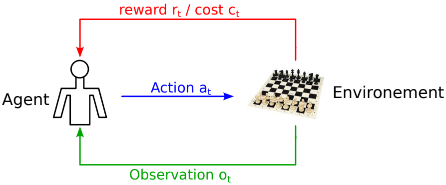


- History: sequence of past observation, actions and rewards up to the agent's decision time t

$$h_t = (o_1, r_1, a_1), (o_2, r_2, a_2), \ldots, (o_t, r_t)$$

- Agent takes decisions given the history.

History and State



- History: sequence of past observation, actions and rewards up to the agent's decision time t

$$h_t = (o_1, r_1, a_1), (o_2, r_2, a_2), \ldots, (o_t, r_t)$$

- Agent takes decisions given the history.
- A state is produced given the history, function of history: $s_t = f(h_t)$.

Agent and Environment States

Agent and Environment States

- Environment state s_t^e: environment's private representation (rules, connections, bad moves etc.)

Agent and Environment States

- Environment state s_t^e: environment's private representation (rules, connections, bad moves etc.)
  - It is the set of information used to understand how the environment generates observations and reward/cost.

Agent and Environment States

- Environment state s_t^e: environment's private representation (rules, connections, bad moves etc.)
  - It is the set of information used to understand how the environment generates observations and reward/cost.
  - Usually not visible to the agent.

Agent and Environment States

- Environment state s_t^e: environment's private representation (rules, connections, bad moves etc.)
  - It is the set of information used to understand how the environment generates observations and reward/cost.
  - Usually not visible to the agent.
- Agent's state s_t^a: agent's private representation (how the agent thinks and analysis etc.)

Agent and Environment States

- Environment state s_t^e: environment's private representation (rules, connections, bad moves etc.)
  - It is the set of information used to understand how the environment generates observations and reward/cost.
  - Usually not visible to the agent.
- Agent's state s_t^a: agent's private representation (how the agent thinks and analysis etc.)
  - It is the set of information the agent uses to take decisions/

Agent and Environment States

- Environment state s_t^e: environment's private representation (rules, connections, bad moves etc.)
  - It is the set of information used to understand how the environment generates observations and reward/cost.
  - Usually not visible to the agent.
- Agent's state s_t^a: agent's private representation (how the agent thinks and analysis etc.)
  - It is the set of information the agent uses to take decisions/
  - Function of history $s_t^a = f(h)$.

Agent and Environment States

- Environment state s_t^e: environment's private representation (rules, connections, bad moves etc.)
  - It is the set of information used to understand how the environment generates observations and reward/cost.
  - Usually not visible to the agent.
- Agent's state s_t^a: agent's private representation (how the agent thinks and analysis etc.)
  - It is the set of information the agent uses to take decisions/
  - Function of history $s_t^a = f(h)$.
- A process changes in a:

Agent and Environment States

- Environment state s_t^e: environment's private representation (rules, connections, bad moves etc.)
  - It is the set of information used to understand how the environment generates observations and reward/cost.
  - Usually not visible to the agent.
- Agent's state s_t^a: agent's private representation (how the agent thinks and analysis etc.)
  - It is the set of information the agent uses to take decisions/
  - Function of history $s_t^a = f(h)$.
- A process changes in a:
  - Deterministic way: given an action, one single observation and reward/cost are generated (ex: robotics)

Agent and Environment States

- Environment state s_t^e: environment's private representation (rules, connections, bad moves etc.)
  - It is the set of information used to understand how the environment generates observations and reward/cost.
  - Usually not visible to the agent.
- Agent's state s_t^a: agent's private representation (how the agent thinks and analysis etc.)
  - It is the set of information the agent uses to take decisions/
  - Function of history $s_t^a = f(h)$.
- A process changes in a:
  - Deterministic way: given an action, one single observation and reward/cost are generated (ex: robotics)
  - Stochastic way: given an action, many possible observations and rewards/costs can be generated (ex: card game)

Components of Sequential Decision Process

Components of Sequential Decision Process

Agents the following components:

Components of Sequential Decision Process

Agents the following components:

- Policy π: agent's behavior (actions) function

Components of Sequential Decision Process

Agents the following components:

- Policy π: agent's behavior (actions) function
- Value function: Given a policy that the agent follows, how good is each state and/or action over a long run.

Components of Sequential Decision Process

Agents the following components:

- Policy π: agent's behavior (actions) function
- Value function: Given a policy that the agent follows, how good is each state and/or action over a long run.
- Model: The agent's understanding of the environment i.e. how the environment responds to a given action provided by the agent.

Policy

Policy

- A policy π determines the decision making (next action) of the agent.

Policy

- A policy π determines the decision making (next action) of the agent.
- A policy π maps the state space to the action space: $\pi : s \rightarrow a$

Policy

- A policy π determines the decision making (next action) of the agent.
- A policy π maps the state space to the action space: $\pi : s \rightarrow a$
- A deterministic policy: $\pi(s) = a$

Policy

- A policy π determines the decision making (next action) of the agent.
- A policy π maps the state space to the action space: $\pi : s \rightarrow a$
- A deterministic policy: $\pi(s) = a$
- Stochastic policy: $\pi(a|s) = P(a_t = a|s_t = s)$

Value Function

- The value function generates the future reward if a given policy is followed.

Value Function

- The value function generates the future reward if a given policy is followed.
- Utility: evaluates the goodness/badness of a policy on a long run.

Value Function

- The value function generates the future reward if a given policy is followed.
- Utility: evaluates the goodness/badness of a policy on a long run.
- The value function V_π that follows the policy π is evaluated as follows:

$$V_\pi(s) = \mathbb{E}_\pi[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+2} + \ldots | s_t = s]$$

Value Function

- The value function generates the future reward if a given policy is followed.
- Utility: evaluates the goodness/badness of a policy on a long run.
- The value function V_π that follows the policy π is evaluated as follows:

$$V_\pi(s) = \mathbb{E}_\pi[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+2} + \ldots | s_t = s]$$

  - γ is a discount factor $0 \leq \gamma \leq 1$

Value Function

- The value function generates the future reward if a given policy is followed.
- Utility: evaluates the goodness/badness of a policy on a long run.
- The value function V_π that follows the policy π is evaluated as follows:

$$V_\pi(s) = \mathbb{E}_\pi[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+2} + \ldots | s_t = s]$$

  - γ is a discount factor $0 \leq \gamma \leq 1$
  - γ controls which is more important: immediate or future reward/cost.

Model

Model

- A model predicts what will the environment do given an agent's action.

Model

- A model predicts what will the environment do given an agent's action.
- A transition model predicts the new agent state:

$$P_{ss'}^{a} = P(s_{t+1} = s' | s_t = s, a_t = a)$$

Model

- A model predicts what will the environment do given an agent's action.
- A transition model predicts the new agent state:

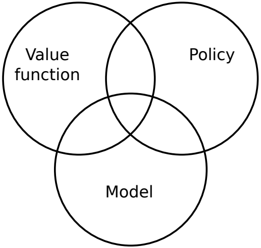
$$P_{ss'}^{a} = P(s_{t+1} = s' | s_t = s, a_t = a)$$

- A reward model predicts the next immediate reward:

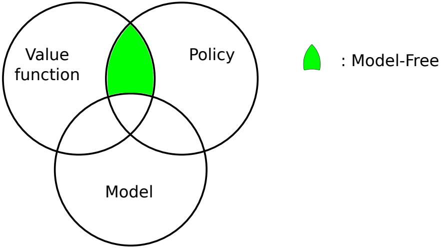
$$R_s^a = R(s,a) = \mathbb{E}[r_{t+1} | s_t = s, a_t = a]$$

Types of Agents

Types of Agents

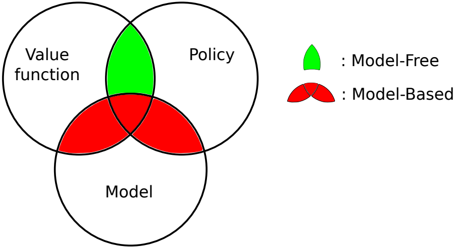


Types of Agents



A model-free agent will optimize the policy without trying to learn the environment rules and state information.

Types of Agents



A model-based agent will optimize the policy by learning the model's behavior and state information.

Sequential Decision Making - Reinforcement Learning

Sequential Decision Making - Reinforcement Learning

- Environment is initially unknown.

Sequential Decision Making - Reinforcement Learning

- Environment is initially unknown.
- Agent interacts with environment to understand its behavior (by taking decisions).

Sequential Decision Making - Reinforcement Learning

- Environment is initially unknown.
- Agent interacts with environment to understand its behavior (by taking decisions).
- Using trial and error approach, the agent optimizes its policy.

Sequential Decision Making - Reinforcement Learning

- Environment is initially unknown.
- Agent interacts with environment to understand its behavior (by taking decisions).
- Using trial and error approach, the agent optimizes its policy.
- Two main concepts in RL, Exploitation and Exploration:

Sequential Decision Making - Reinforcement Learning

- Environment is initially unknown.
- Agent interacts with environment to understand its behavior (by taking decisions).
- Using trial and error approach, the agent optimizes its policy.
- Two main concepts in RL, Exploitation and Exploration:
  - Exploitation: select the action that maximizes the reward given past experience.

Sequential Decision Making - Reinforcement Learning

- Environment is initially unknown.
- Agent interacts with environment to understand its behavior (by taking decisions).
- Using trial and error approach, the agent optimizes its policy.
- Two main concepts in RL, Exploitation and Exploration:
  - Exploitation: select the action that maximizes the reward given past experience.
  - Exploration: discover more possibilities to find more information about the environment's behavior to avoid doing a mistake in the future.

Exploitation/Exploitation Trade Off

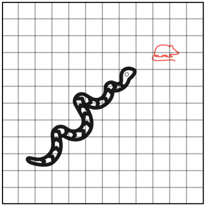
Exploitation/Exploitation Trade Off

Agent
Simulator

Consider an agent being a computer bot.

Exploitation/Exploitation Trade Off

Agent Simulator



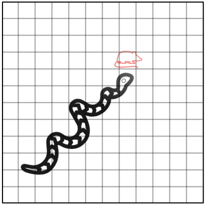
Snake Game

The agent, at time t, is trying to win a snake game by understanding the environment's behavior.

Exploitation/Exploitation Trade Off

Hmm instead of going up I will try to go right this time for future use

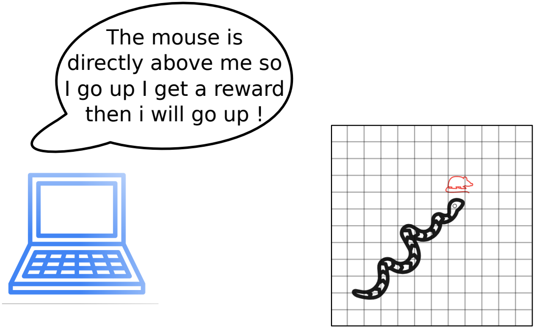
Agent Simulator



Snake Game

The agent when exploring is trying to do an action to explore.

Exploitation/Exploitation Trade Off



Agent Simulator

Snake Game

The agent when exploiting is trying to take a decision that maximizes the reward.

Markov Decision Process (MDP)

MDP and Markov Property

Definition: Markov State

A state s_t is Markov if and only if:

$$P(s_{t+1}|s_t) = P(s_{t+1}|s_t, s_{t-1}, \ldots, s_1)$$

MDP and Markov Property

Definition: Markov State

A state s_t is Markov if and only if:

$$P(s_{t+1}|s_t) = P(s_{t+1}|s_t, s_{t-1}, \ldots, s_1)$$

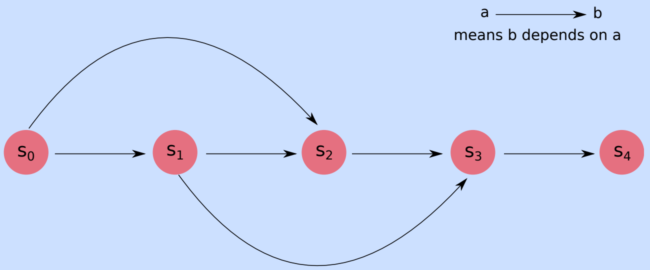
a $\longrightarrow$ b
means b depends on a

MDP and Markov Property

Definition: Markov State

A state s_t is Markov if and only if:

$$P(s_{t+1}|s_t) = P(s_{t+1}|s_t, s_{t-1}, \ldots, s_1)$$

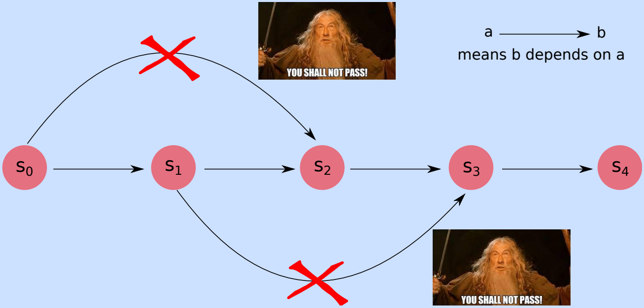


MDP and Markov Property

Definition: Markov State

A state s_t is Markov if and only if:

$$P(s_{t+1}|s_t) = P(s_{t+1}|s_t, s_{t-1}, \dots, s_1)$$



MDP and Markov Property

MDP and Markov Property

- MDP is a classical form of representing a decision making problem.

MDP and Markov Property

- MDP is a classical form of representing a decision making problem.
- For finite number of states N, a state transition matrix P defines the transition probabilities from all states s to all successor states s'.

MDP and Markov Property

- MDP is a classical form of representing a decision making problem.
- For finite number of states N, a state transition matrix P defines the transition probabilities from all states s to all successor states s'.
- State transition matrix P where only the rows sum up to 1.

$$P = [P_{ss'}] \begin{bmatrix} P_{s_1 s_1} & P_{s_1 s_2} & \dots & P_{s_1 s_N} \\ P_{s_2 s_1} & P_{s_2 s_2} & \dots & P_{s_2 s_N} \\ \dots & \dots & \dots & \dots \\ P_{s_N s_1} & P_{s_N s_2} & \dots & P_{s_N s_N} \end{bmatrix}$$

MDP and Markov Property

- MDP is a classical form of representing a decision making problem.
- For finite number of states N, a state transition matrix P defines the transition probabilities from all states s to all successor states s'.
- State transition matrix P where only the rows sum up to 1.

$$P = [P_{ss'}] \begin{bmatrix} P_{s_1 s_1} & P_{s_1 s_2} & \dots & P_{s_1 s_N} \\ P_{s_2 s_1} & P_{s_2 s_2} & \dots & P_{s_2 s_N} \\ \dots & \dots & \dots & \dots \\ P_{s_N s_1} & P_{s_N s_2} & \dots & P_{s_N s_N} \end{bmatrix}$$

- Rows sum up to 1 but columns do not (proof next slide).

MDP and Markov Property

proof:

MDP and Markov Property

proof:

- Rows sum up to 1:

MDP and Markov Property

proof:

- Rows sum up to 1:

$$\text{Sum of row } r = \sum_{i=1}^{N} P_{s_r s_i}(s_{t+1} = s_i | s_t = s_r)$$

MDP and Markov Property

proof:

- Rows sum up to 1:

$$\text{Sum of row } r = \sum_{i=1}^{N} P_{s_r s_i}(s_{t+1} = s_i | s_t = s_r)$$
$$= \sum_{i=1}^{N} \frac{P_{s_r s_i}(s_{t+1} = s_i, s_t = s_r)}{P(s_t = s_r)} \quad \textit{Bias Rule}$$

MDP and Markov Property

proof:

- Rows sum up to 1:

$$\begin{aligned}
\text{Sum of row } r &= \sum_{i=1}^{N} P_{s_r s_i}(s_{t+1} = s_i | s_t = s_r) \\
&= \sum_{i=1}^{N} \frac{P_{s_r s_i}(s_{t+1} = s_i, s_t = s_r)}{P(s_t = s_r)} \quad \textit{Bias Rule} \\
&= \frac{\sum_{i=1}^{N} P_{s_r s_i}(s_{t+1} = s_i, s_t = s_r)}{P(s_t = s_r)}
\end{aligned}$$

MDP and Markov Property

proof:

- Rows sum up to 1:

$$
\begin{aligned}
\text{Sum of row } r &= \sum_{i=1}^{N} P_{s_r s_i}(s_{t+1}=s_i|s_t=s_r) \\
&= \sum_{i=1}^{N} \frac{P_{s_r s_i}(s_{t+1}=s_i, s_t=s_r)}{P(s_t=s_r)} \quad \textit{Bias Rule} \\
&= \frac{\sum_{i=1}^{N} P_{s_r s_i}(s_{t+1}=s_i, s_t=s_r)}{P(s_t=s_r)} \\
&= \frac{P(s_t=s_r)}{P(s_t=s_r)} \quad \textit{Marginal Distribution} \\
&= 1
\end{aligned}
$$

MDP and Markov Property

proof:

- Columns do not sum up to 1:

$$\text{Sum of column } c = \sum_{i=1}^{N} P_{s_i s_c}(s_{t+1} = s_c | s_t = s_i)$$

$$= \sum_{i=1}^{N} \frac{P_{s_i s_c}(s_{t+1} = s_c, s_t = s_i)}{P(s_t = s_i)} \quad \textit{Bias Rule}$$

Nothing can be done here as the sum includes both numerator and denominator.

Markov Process

Markov Process

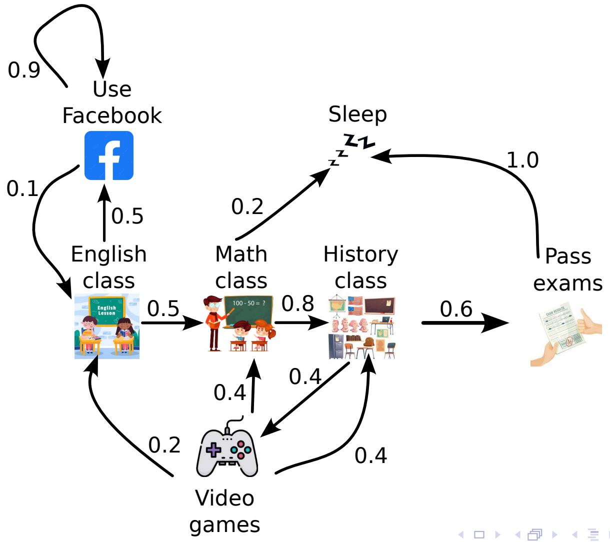
Definition: Markov Process or Markov Chain

A Markov Process or Chain is a tuple $< S, P >$ where:

- S is a finite set of states
- P is a state transition probability matrix

Markov Process

Example: Student Markov Chain



Markov Reward Process (MRP)

Markov Reward Process (MRP)

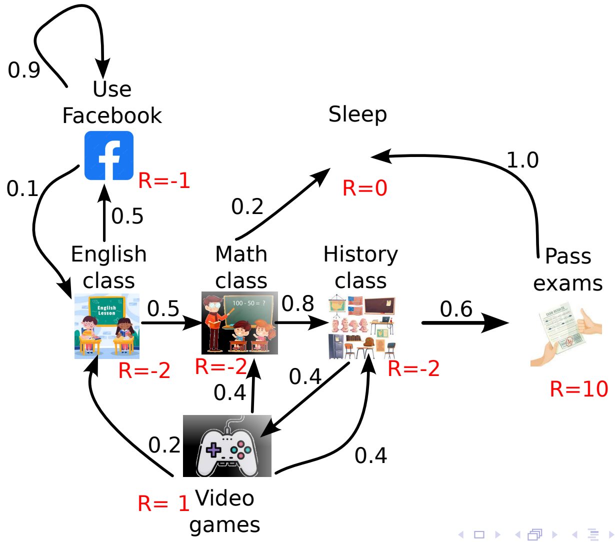
Definition: Markov Reward Process

A Markov Reward Process is a tuple $< S, P, R, \gamma >$ where:

- S is a finite set of states
- P is a state transition probability matrix
- R is a reward function
- γ is a discount factor

Markov Reward Process (MRP)

Example: Student MRP



Properties of MRP

Properties of MRP

Definition: Horizon

The number of time steps T in each episode, can be finite or ∞.

Properties of MRP

Definition: Horizon

The number of time steps T in each episode, can be finite or ∞.

Definition: Return

The return G_t at time t of an MRP is the total discounted reward from t to T.

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

Properties of MRP

Definition: Horizon

The number of time steps T in each episode, can be finite or ∞.

Definition: Return

The return G_t at time t of an MRP is the total discounted reward from t to T.

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

Definition: State Value Function

As defined before, slide 10. Its the expected return starting from state s:

$$V(s) = \mathbb{E}[G_t | s_t = s]$$

Bellman Equation for MRP

Bellman Equation for MRP

The Bellman equation decomposes the state value function of an MRP into two parts:

Bellman Equation for MRP

The Bellman equation decomposes the state value function of an MRP into two parts:

- Immediate reward

Bellman Equation for MRP

The Bellman equation decomposes the state value function of an MRP into two parts:

- Immediate reward
- Discounted value of successor state

Bellman Equation for MRP

The Bellman equation decomposes the state value function of an MRP into two parts:

- Immediate reward
- Discounted value of successor state

$$\begin{aligned} V(s) &= \mathbb{E}[r_{t+1} + \gamma V(s_{t+1}) | s_t = s] \text{ or,} \\ V(s) &= R_s + \gamma \sum_{s' \in S} P_{ss'} V(s') \end{aligned} \tag{1}$$

Bellman Equation for MRP

The Bellman equation decomposes the state value function of an MRP into two parts:

- Immediate reward
- Discounted value of successor state

$$V(s) = \mathbb{E}[r_{t+1} + \gamma V(s_{t+1}) | s_t = s] \text{ or,}$$
$$V(s) = R_s + \gamma \sum_{s' \in S} P_{ss'} V(s') \tag{1}$$

proof:

Bellman Equation for MRP

The Bellman equation decomposes the state value function of an MRP into two parts:

- Immediate reward
- Discounted value of successor state

$$V(s) = \mathbb{E}[r_{t+1} + \gamma V(s_{t+1}) | s_t = s] \text{ or,}$$
$$V(s) = R_s + \gamma \sum_{s' \in S} P_{ss'} V(s') \tag{1}$$

proof:

$$V(s) = \mathbb{E}[G_t | s_t = s]$$

Bellman Equation for MRP

The Bellman equation decomposes the state value function of an MRP into two parts:

- Immediate reward
- Discounted value of successor state

$$
\begin{aligned}
V(s) &= \mathbb{E}[r_{t+1} + \gamma V(s_{t+1}) | s_t = s] \text{ or,} \\
V(s) &= R_s + \gamma \sum_{s' \in S} P_{ss'} V(s')
\end{aligned} \tag{1}
$$

proof:

$$
\begin{aligned}
V(s) &= \mathbb{E}[G_t | s_t = s] \\
&= \mathbb{E}[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots | s_t = s]
\end{aligned}
$$

Bellman Equation for MRP

The Bellman equation decomposes the state value function of an MRP into two parts:

- Immediate reward
- Discounted value of successor state

$$V(s) = \mathbb{E}[r_{t+1} + \gamma V(s_{t+1})|s_t = s] \text{ or,}$$
$$V(s) = R_s + \gamma \sum_{s' \in S} P_{ss'} V(s') \tag{1}$$

proof:

$$\begin{aligned} V(s) &= \mathbb{E}[G_t|s_t = s] \\ &= \mathbb{E}[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots |s_t = s] \\ &= \mathbb{E}[r_{t+1} + \gamma.(r_{t+2} + \gamma r_{t+3} + \dots)|s_t = s] \end{aligned}$$

Bellman Equation for MRP

The Bellman equation decomposes the state value function of an MRP into two parts:

- Immediate reward
- Discounted value of successor state

$$\begin{aligned} V(s) &= \mathbb{E}[r_{t+1} + \gamma V(s_{t+1}) | s_t = s] \text{ or,} \\ V(s) &= R_s + \gamma \sum_{s' \in S} P_{ss'} V(s') \end{aligned} \tag{1}$$

proof:

$$\begin{aligned} V(s) &= \mathbb{E}[G_t | s_t = s] \\ &= \mathbb{E}[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots | s_t = s] \\ &= \mathbb{E}[r_{t+1} + \gamma.(r_{t+2} + \gamma r_{t+3} + \ldots) | s_t = s] \\ &= \mathbb{E}[r_{t+1} + \gamma G_{t+1} | s_t = s] \end{aligned}$$

Bellman Equation for MRP

The Bellman equation decomposes the state value function of an MRP into two parts:

- Immediate reward
- Discounted value of successor state

$$V(s) = \mathbb{E}[r_{t+1} + \gamma V(s_{t+1}) | s_t = s] \text{ or,}$$
$$V(s) = R_s + \gamma \sum_{s' \in S} P_{ss'} V(s') \tag{1}$$

proof:

$$\begin{aligned} V(s) &= \mathbb{E}[G_t | s_t = s] \\ &= \mathbb{E}[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots | s_t = s] \\ &= \mathbb{E}[r_{t+1} + \gamma.(r_{t+2} + \gamma r_{t+3} + \ldots) | s_t = s] \\ &= \mathbb{E}[r_{t+1} + \gamma G_{t+1} | s_t = s] \\ &= \mathbb{E}[r_{t+1} + \gamma V(s_{t+1}) | s_t = s] \end{aligned}$$

Bellman Equation for MRP

The Bellman equation decomposes the state value function of an MRP into two parts:

- Immediate reward
- Discounted value of successor state

$$\begin{aligned} V(s) &= \mathbb{E}[r_{t+1} + \gamma V(s_{t+1}) | s_t = s] \text{ or,} \\ V(s) &= R_s + \gamma \sum_{s' \in S} P_{ss'} V(s') \end{aligned} \tag{1}$$

proof:

$$\begin{aligned} V(s) &= \mathbb{E}[G_t | s_t = s] \\ &= \mathbb{E}[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots | s_t = s] \\ &= \mathbb{E}[r_{t+1} + \gamma.(r_{t+2} + \gamma r_{t+3} + \dots) | s_t = s] \\ &= \mathbb{E}[r_{t+1} + \gamma G_{t+1} | s_t = s] \\ &= \mathbb{E}[r_{t+1} + \gamma V(s_{t+1}) | s_t = s] \end{aligned}$$

The last step simply uses the rule that says $\mathbb{E}[\mathbb{E}[X|Y]|Z] = \mathbb{E}[X|Z]$ and that

$$\mathbb{E}[V(s_{t+1})|s_t = s] = \mathbb{E}[\mathbb{E}[G_{t+1}|s_{t+1}]|s_t = s] = \mathbb{E}[G_{t+1}|s_t = s]$$

Solving the Bellman Equation for an MRP - Matrix Form

Solving the Bellman Equation for an MRP - Matrix Form

- The goal is to find the optimal value function. The Bellman equation can be re-written in this matrix form:

$$\mathbf{V} = \mathbf{R} + \gamma \mathbf{P} \mathbf{V}$$

, where **V** and **R** are vectors of value/reward for all states and **P** is the transition matrix.

Solving the Bellman Equation for an MRP - Matrix Form

- The goal is to find the optimal value function. The Bellman equation can be re-written in this matrix form:

$$\mathbf{V} = \mathbf{R} + \gamma \mathbf{P}\mathbf{V}$$

  , where $\mathbf{V}$ and $\mathbf{R}$ are vectors of value/reward for all states and $\mathbf{P}$ is the transition matrix.

- Solution of this equation:

$$\begin{aligned} \mathbf{V} &= \mathbf{R} + \gamma \mathbf{P}\mathbf{V} \\ (\mathbf{I} - \gamma \mathbf{P})\mathbf{V} &= \mathbf{R} \\ \mathbf{V} &= (I - \gamma \mathbf{P})^{-1}\mathbf{R} \end{aligned}$$

Solving the Bellman Equation for an MRP - Matrix Form

- The goal is to find the optimal value function. The Bellman equation can be re-written in this matrix form:

$$\mathbf{V} = \mathbf{R} + \gamma \mathbf{P}\mathbf{V}$$

, where $\mathbf{V}$ and $\mathbf{R}$ are vectors of value/reward for all states and $\mathbf{P}$ is the transition matrix.

- Solution of this equation:

$$\begin{aligned} \mathbf{V} &= \mathbf{R} + \gamma \mathbf{P}\mathbf{V} \\ (\mathbf{I} - \gamma \mathbf{P})\mathbf{V} &= \mathbf{R} \\ \mathbf{V} &= (I - \gamma \mathbf{P})^{-1}\mathbf{R} \end{aligned}$$

- Solving this would take $\mathcal{O}(N^3)$ for N states.

Solving the Bellman Equation for an MRP - Matrix Form

- The goal is to find the optimal value function. The Bellman equation can be re-written in this matrix form:

  $$\mathbf{V} = \mathbf{R} + \gamma \mathbf{P}\mathbf{V}$$

  , where **V** and **R** are vectors of value/reward for all states and **P** is the transition matrix.
- Solution of this equation:

  $$\begin{aligned} \mathbf{V} &= \mathbf{R} + \gamma \mathbf{P}\mathbf{V} \\ (\mathbf{I} - \gamma \mathbf{P})\mathbf{V} &= \mathbf{R} \\ \mathbf{V} &= (I - \gamma \mathbf{P})^{-1}\mathbf{R} \end{aligned}$$

- Solving this would take $\mathcal{O}(N^3)$ for N states.
- This is not efficient for large values of N.

Solving the Bellman Equation for an MRP - Matrix Form

- The goal is to find the optimal value function. The Bellman equation can be re-written in this matrix form:

$$\mathbf{V} = \mathbf{R} + \gamma \mathbf{P}\mathbf{V}$$

  , where **V** and **R** are vectors of value/reward for all states and **P** is the transition matrix.

- Solution of this equation:

$$\begin{aligned} \mathbf{V} &= \mathbf{R} + \gamma \mathbf{P}\mathbf{V} \\ (\mathbf{I} - \gamma \mathbf{P})\mathbf{V} &= \mathbf{R} \\ \mathbf{V} &= (I - \gamma \mathbf{P})^{-1}\mathbf{R} \end{aligned}$$

- Solving this would take $\mathcal{O}(N^3)$ for N states.
- This is not efficient for large values of N.
- Better solution: Dynamic programming.

Defining the MDP

Definition: Markov Decision Process

Its a tuple $< S, A, P, R, \gamma >$

Defining the MDP

Definition: Markov Decision Process

Its a tuple $< S, A, P, R, \gamma >$

- S is a finite set of states $s \in S$

Defining the MDP

Definition: Markov Decision Process

Its a tuple $< S, A, P, R, \gamma >$

- S is a finite set of states $s \in S$
- A is a finite set of actions $a \in A$

Defining the MDP

Definition: Markov Decision Process

Its a tuple $< S, A, P, R, \gamma >$

- S is a finite set of states $s \in S$
- A is a finite set of actions $a \in A$
- P is a state transition probability matrix

$$P_{ss'}^{a} = P(s'|s,a) = P(s_{t+1} = s'|s_t = s, a_t = a)$$

Defining the MDP

Definition: Markov Decision Process

Its a tuple $< S, A, P, R, \gamma >$

- S is a finite set of states $s \in S$
- A is a finite set of actions $a \in A$
- P is a state transition probability matrix

$$P^a_{ss'} = P(s'|s,a) = P(s_{t+1} = s'|s_t = s, a_t = a)$$

- R is a reward function

$$R^a_s = R(s_t = a, a_t = a) = \mathbb{E}[r_{t+1}|s_t = s, a_t = a]$$

Defining the MDP

Definition: Markov Decision Process

Its a tuple $< S, A, P, R, \gamma >$

- S is a finite set of states $s \in S$
- A is a finite set of actions $a \in A$
- P is a state transition probability matrix

$$P^a_{ss'} = P(s'|s,a) = P(s_{t+1} = s'|s_t = s, a_t = a)$$

- R is a reward function

$$R^a_s = R(s_t = a, a_t = a) = \mathbb{E}[r_{t+1}|s_t = s, a_t = a]$$

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

Model-Based Reinforcement Learning Methods

Dynamic Programming (DP)

Dynamic Programming (DP)

- Mathematical and algorithmic method solving complex optimization problems.

Dynamic Programming (DP)

- Mathematical and algorithmic method solving complex optimization problems.
- Paradigm proposed by Richard Bellman in 1953 at RAND corporation.

Dynamic Programming (DP)

- Mathematical and algorithmic method solving complex optimization problems.
- Paradigm proposed by Richard Bellman in 1953 at RAND corporation.
- Principle: Breaking complex problems down into sub-problems.

Dynamic Programming (DP)

- Mathematical and algorithmic method solving complex optimization problems.
- Paradigm proposed by Richard Bellman in 1953 at RAND corporation.
- Principle: Breaking complex problems down into sub-problems.

DP is used to solve MDP and achieve optimal policy.

Dynamic Programming (DP)

- Mathematical and algorithmic method solving complex optimization problems.
- Paradigm proposed by Richard Bellman in 1953 at RAND corporation.
- Principle: Breaking complex problems down into sub-problems.

DP is used to solve MDP and achieve optimal policy.

Definition: Principle of Optimality

A policy $\pi(a|s)$ achieves the optimal value from state s, $V_\pi(s) = V^*(s)$ if and only if:

For any state s' reachable from s, π achieves the optimal value from state s',

$$V_\pi(s') = V^*(s')$$

Policy Iteration

Policy Iteration

- A DP algorithm that finds the optimal policy for an MDP.

Policy Iteration

- A DP algorithm that finds the optimal policy for an MDP.
- The algorithm consists of two steps repeated until convergence:

Policy Iteration

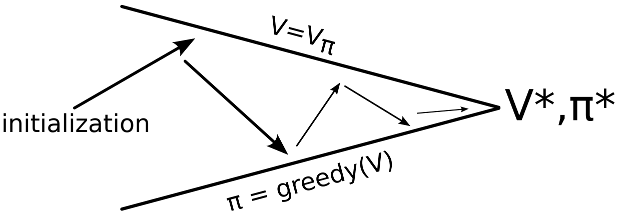
- A DP algorithm that finds the optimal policy for an MDP.
- The algorithm consists of two steps repeated until convergence:
  - Step 1: Evaluate policy π using
    $$V_\pi(s) = \mathbb{E}[r_{t+1} + \gamma r_{t+2} + \ldots | s_t = s] = \mathbb{E}[r_{t+1} + \gamma V_\pi(s_{t+1}) | s_t = s]$$
    meaning until value function becomes unchangeable

Policy Iteration

- A DP algorithm that finds the optimal policy for an MDP.
- The algorithm consists of two steps repeated until convergence:
  - Step 1: Evaluate policy π using
    $$V_\pi(s) = \mathbb{E}[r_{t+1} + \gamma r_{t+2} + \ldots | s_t = s] = \mathbb{E}[r_{t+1} + \gamma V_\pi(s_{t+1}) | s_t = s]$$
    meaning until value function becomes unchangeable
  - Step 2: Improve the policy by acting greedily w.r.t π using one look ahead in the Bellman equation.
    $$\pi' = greedy(V_\pi)$$

Policy Iteration

- A DP algorithm that finds the optimal policy for an MDP.
- The algorithm consists of two steps repeated until convergence:
  - Step 1: Evaluate policy π using
    $$V_\pi(s) = \mathbb{E}[r_{t+1} + \gamma r_{t+2} + \ldots | s_t = s] = \mathbb{E}[r_{t+1} + \gamma V_\pi(s_{t+1}) | s_t = s]$$
    meaning until value function becomes unchangeable
  - Step 2: Improve the policy by acting greedily w.r.t π using one look ahead in the Bellman equation.
    $$\pi' = greedy(V_\pi)$$



Value Iteration

Value Iteration

- A DP algorithm that finds the optimal policy after iteratively have found the optimal value function.

Value Iteration

- A DP algorithm that finds the optimal policy after iteratively have found the optimal value function.
- The algorithm consists on repeating the same step:

$$V(s) = \max_{a \in A}(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a(s'))$$

Value Iteration

- A DP algorithm that finds the optimal policy after iteratively have found the optimal value function.
- The algorithm consists on repeating the same step:

$$V(s) = \max_{a \in A}(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a(s'))$$

- The last step consisting on choosing the policy of the optimal value function:

$$\pi(s) = arg \max_{a \in A}(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a(s'))$$

Policy VS Value Iteration

Table: Comparing Policy and Value Iteration algorithms to solve MDP model-based problems.

Properties	**Policy Iteration**	**Value Iteration**
Initialization	Random policy	Random value function
Complexity	Very complex	Simple
Convergence	Guaranteed	Guaranteed
Number of iterations till convergence	Small number	High number
Speed	Fast	Slow

Model-Free Reinforcement Learning Methods

Model-Free RL

Model-Free RL

- Dynamic programming cannot be used in this case

Model-Free RL

- Dynamic programming cannot be used in this case
- Model-Free RL can learn:

Model-Free RL

- Dynamic programming cannot be used in this case
- Model-Free RL can learn:
  - without knowledge of the environment

Model-Free RL

- Dynamic programming cannot be used in this case
- Model-Free RL can learn:
  - without knowledge of the environment
  - to estimate the value function of an unknown MDP, we call this Model-Free Prediction

Model-Free RL

- Dynamic programming cannot be used in this case
- Model-Free RL can learn:
  - without knowledge of the environment
  - to estimate the value function of an unknown MDP, we call this Model-Free Prediction
  - to optimize the value function of an unknown MDP, we call this Model-Free Control

Model-Free RL methods: Outline

Model-Free RL methods: Outline

- Model-Free Prediction

Model-Free RL methods: Outline

- Model-Free Prediction
  - Monte-Carlo Prediction

Model-Free RL methods: Outline

- Model-Free Prediction
  - Monte-Carlo Prediction
  - Temporal Difference Learning (TD Learning)

Model-Free RL methods: Outline

- Model-Free Prediction
  - Monte-Carlo Prediction
  - Temporal Difference Learning (TD Learning)
  - n-step TD Prediction

Model-Free RL methods: Outline

- Model-Free Prediction
  - Monte-Carlo Prediction
  - Temporal Difference Learning (TD Learning)
  - n-step TD Prediction
  - TD(λ)

Model-Free RL methods: Outline

- Model-Free Prediction
  - Monte-Carlo Prediction
  - Temporal Difference Learning (TD Learning)
  - n-step TD Prediction
  - TD(λ)
- Model-Free Control

Model-Free RL methods: Outline

- Model-Free Prediction
  - Monte-Carlo Prediction
  - Temporal Difference Learning (TD Learning)
  - n-step TD Prediction
  - TD(λ)
- Model-Free Control
  - On-Policy Monte-Carlo Control

Model-Free RL methods: Outline

- Model-Free Prediction
  - Monte-Carlo Prediction
  - Temporal Difference Learning (TD Learning)
  - n-step TD Prediction
  - TD(λ)
- Model-Free Control
  - On-Policy Monte-Carlo Control
  - On-Policy TD Control: SARSA

Model-Free RL methods: Outline

- Model-Free Prediction
  - Monte-Carlo Prediction
  - Temporal Difference Learning (TD Learning)
  - n-step TD Prediction
  - TD(λ)
- Model-Free Control
  - On-Policy Monte-Carlo Control
  - On-Policy TD Control: SARSA
  - Off-Policy Learning: Q-Learning

Model-Free RL methods: Outline

- Model-Free Prediction
  - Monte-Carlo Prediction $\rightarrow$ **Will see in this course**
  - Temporal Difference Learning (TD Learning)
  - n-step TD Prediction
  - TD(λ)
- Model-Free Control
  - On-Policy Monte-Carlo Control
  - On-Policy TD Control: SARSA
  - Off-Policy Learning: Q-Learning $\rightarrow$ **Will see in this course**

Monte-Carlo Simulation

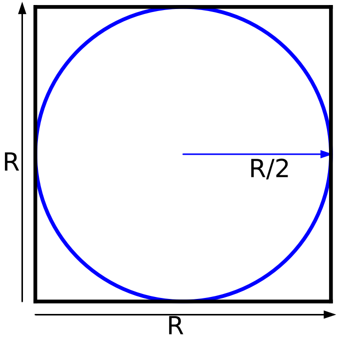
Monte-Carlo Simulation

- An usefull place for a Monte-Carlo method is when we want to estimate the value of π

Monte-Carlo Simulation

- An usefull place for a Monte-Carlo method is when we want to estimate the value of π

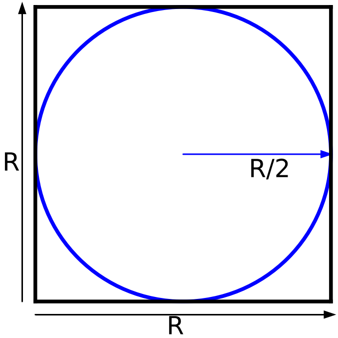
Consider the following example



Monte-Carlo Simulation

- An usefull place for a Monte-Carlo method is when we want to estimate the value of π

Consider the following example

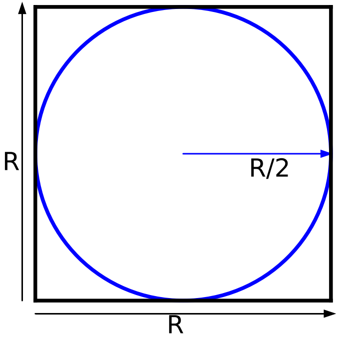


- The area of the square is: $A_{sq} = R^2$

Monte-Carlo Simulation

- An usefull place for a Monte-Carlo method is when we want to estimate the value of π

Consider the following example



- The area of the square is: $A_{sq} = R^2$
- The area of the circle is: $A_{cr} = \frac{\pi R^2}{4}$

Monte-Carlo Simulation

To estimate the value of π, we can randomly choose points in the $2D$ grid, but guaranteed to be inside the square.

Monte-Carlo Simulation

To estimate the value of π, we can randomly choose points in the $2D$ grid, but guaranteed to be inside the square.
The probability of a point to be inside the circle is:

$$P = \frac{A_{cr}}{A_{sq}} = \frac{\pi}{4}$$

Monte-Carlo Simulation

To estimate the value of π, we can randomly choose points in the $2D$ grid, but guaranteed to be inside the square.
The probability of a point to be inside the circle is:

$$P = \frac{A_{cr}}{A_{sq}} = \frac{\pi}{4}$$

This means that $\pi = 4 * P$

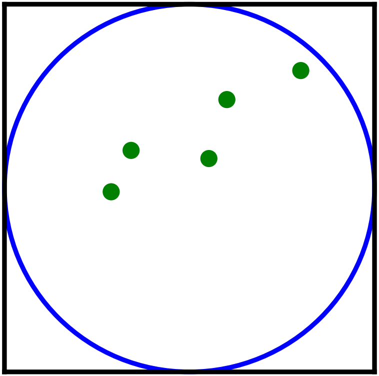
Monte-Carlo Simulation

To estimate the probability P of a point being in the circle, with enough generated points we have:

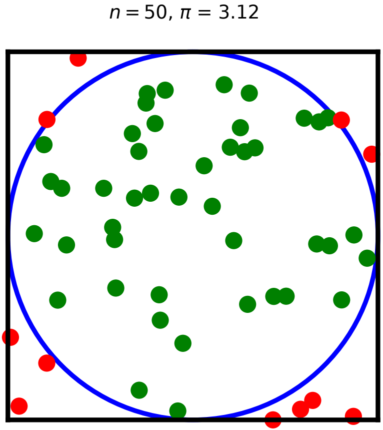
$$P = \frac{\textit{number of points landing inside the circle}}{\textit{total number of generated points}}$$

Monte-Carlo Simulation

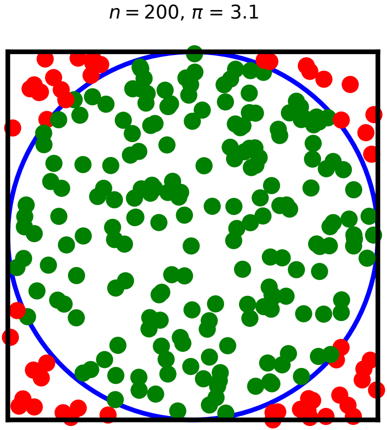
$n = 5$, $\pi = 4.0$



Monte-Carlo Simulation

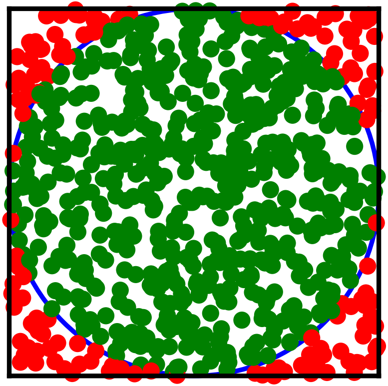


Monte-Carlo Simulation



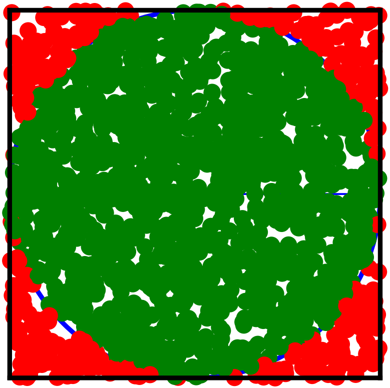
Monte-Carlo Simulation

$n = 500$, $\pi = 3.264$



Monte-Carlo Simulation

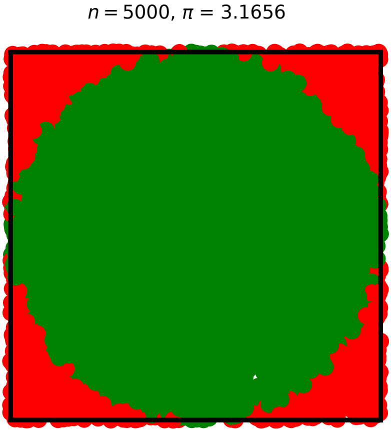
$n = 1000$, $\pi = 3.22$



Monte-Carlo Simulation

$n = 5000$, $\pi = 3.1656$

Monte-Carlo Simulation



If we keep generating points we will converge to almost 3.14

Monte-Carlo Prediction

Monte-Carlo Prediction

- MC is a model-free algorithm so no knowledge of the environment exists, i.e. transition matrix and rewards.

Monte-Carlo Prediction

- MC is a model-free algorithm so no knowledge of the environment exists, i.e. transition matrix and rewards.
- MC requires only experience which is gained by interacting with the environment.

Monte-Carlo Prediction

- MC is a model-free algorithm so no knowledge of the environment exists, i.e. transition matrix and rewards.
- MC requires only experience which is gained by interacting with the environment.
- MC learns from episodes hence simulated experience.

Monte-Carlo Prediction

- MC is a model-free algorithm so no knowledge of the environment exists, i.e. transition matrix and rewards.
- MC requires only experience which is gained by interacting with the environment.
- MC learns from episodes hence simulated experience.
- MC can only learn on complete episodes, i.e. the episode of states, actions, rewards should terminate.

Monte-Carlo Prediction

- MC is a model-free algorithm so no knowledge of the environment exists, i.e. transition matrix and rewards.
- MC requires only experience which is gained by interacting with the environment.
- MC learns from episodes hence simulated experience.
- MC can only learn on complete episodes, i.e. the episode of states, actions, rewards should terminate.
- The last property is a must, without it, MC does not work.

Monte-Carlo Prediction

- MC is a model-free algorithm so no knowledge of the environment exists, i.e. transition matrix and rewards.
- MC requires only experience which is gained by interacting with the environment.
- MC learns from episodes hence simulated experience.
- MC can only learn on complete episodes, i.e. the episode of states, actions, rewards should terminate.
- The last property is a must, without it, MC does not work.
- There exists multiple MC Prediction methods (all of them estimate the value function with the arithmetic mean):

Monte-Carlo Prediction

- MC is a model-free algorithm so no knowledge of the environment exists, i.e. transition matrix and rewards.
- MC requires only experience which is gained by interacting with the environment.
- MC learns from episodes hence simulated experience.
- MC can only learn on complete episodes, i.e. the episode of states, actions, rewards should terminate.
- The last property is a must, without it, MC does not work.
- There exists multiple MC Prediction methods (all of them estimate the value function with the arithmetic mean):
    - First-Visit MC Policy Evaluation

Monte-Carlo Prediction

- MC is a model-free algorithm so no knowledge of the environment exists, i.e. transition matrix and rewards.
- MC requires only experience which is gained by interacting with the environment.
- MC learns from episodes hence simulated experience.
- MC can only learn on complete episodes, i.e. the episode of states, actions, rewards should terminate.
- The last property is a must, without it, MC does not work.
- There exists multiple MC Prediction methods (all of them estimate the value function with the arithmetic mean):
  - First-Visit MC Policy Evaluation
  - Every-Visit MC Policy Evaluation

Monte-Carlo Prediction

- MC is a model-free algorithm so no knowledge of the environment exists, i.e. transition matrix and rewards.
- MC requires only experience which is gained by interacting with the environment.
- MC learns from episodes hence simulated experience.
- MC can only learn on complete episodes, i.e. the episode of states, actions, rewards should terminate.
- The last property is a must, without it, MC does not work.
- There exists multiple MC Prediction methods (all of them estimate the value function with the arithmetic mean):
  - First-Visit MC Policy Evaluation
  - Every-Visit MC Policy Evaluation
  - Incremental MC

First-Visit MC Policy Evaluation

Algorithm 1 First-Visit Monte-Carlo Prediction

Input: Policy π, int E the number of episodes
Output: Value function V_π

Initialize #visits $N(s) = 0$ and Returns $G(s) = 0$ $\forall s \in S$
for $e = 1$ to E **do**
  Generate using π an episode $s_0, a_0, r_1, s_1, a_1, r_2, \ldots, s_{T-1}, a_{T-1}, a_T$
  $g \leftarrow 0$
  for $t = T - 1$ to $t = 0$ **do**
    $g \leftarrow g + r_{t+1}$
    if state s_t is not in the sequence $s_0, s_1, \ldots, s_{t-1}$ **then**
      $G(s_t) \leftarrow G(s_t) + g$
      $N(s_t) \leftarrow N(s_t) + 1$
    end if
  end for
end for
return $V(s) \leftarrow \dfrac{G(s)}{N(s)} \forall s \in S$

Every-Visit MC Policy Evaluation

Algorithm 2 Every-Visit Monte-Carlo Prediction

Input: Policy π, $int\ E$ the number of episodes
Output: Value function V_π

Initialize #visits $N(s) = 0$ and Returns $G(s) = 0\ \forall s \in S$
Initialize $S(s) = 0\ \forall s\ inS$
for $e = 1$ to E **do**
  Generate using π an episode $s_0, a_0, r_1, s_1, a_1, r_2, \ldots, s_{T-1}, a_{T-1}, a_T$
  $g \leftarrow$ Return from time-step t to horizon T
  for $t = T - 1$ to $t = 0$ **do**
    $G(s_t) \leftarrow G(s_t) + g$
    $N(s_t) \leftarrow N(s_t) + 1$
  end for
end for
return $V(s) \leftarrow \dfrac{G(s)}{N(s)} \forall s \in S$

Incremental Monte-Carlo Updates

Incremental Monte-Carlo Updates

- Update $V(s)$ incrementally after episode $s_0, a_0, r_1, s_1, a_1, r_2, \ldots, s_{T-1}, a_{T-1}, r_T$

Incremental Monte-Carlo Updates

- Update $V(s)$ incrementally after episode $s_0, a_0, r_1, s_1, a_1, r_2, \ldots, s_{T-1}, a_{T-1}, r_T$
- Incremental mean: Given V_n and $G_{t,n}$ the n_{th} estimation of V and the n_{th} Return:

Incremental Monte-Carlo Updates

- Update $V(s)$ incrementally after episode $s_0, a_0, r_1, s_1, a_1, r_2, \ldots, s_{T-1}, a_{T-1}, r_T$
- Incremental mean: Given V_n and $G_{t,n}$ the n_{th} estimation of V and the n_{th} Return:

$$V_n(s_t) = \frac{1}{n} \sum_{i=1}^{n} G_{t,i}$$

Incremental Monte-Carlo Updates

- Update $V(s)$ incrementally after episode $s_0, a_0, r_1, s_1, a_1, r_2, \ldots, s_{T-1}, a_{T-1}, r_T$
- Incremental mean: Given V_n and $G_{t,n}$ the n_{th} estimation of V and the n_{th} Return:

$$\begin{aligned} V_n(s_t) &= \frac{1}{n}\sum_{i=1}^{n} G_{t,i} \\ &= \frac{1}{n}(G_{t,n} + \sum_{i=1}^{n-1} G_{t,i}) \end{aligned}$$

Incremental Monte-Carlo Updates

- Update $V(s)$ incrementally after episode $s_0, a_0, r_1, s_1, a_1, r_2, \ldots, s_{T-1}, a_{T-1}, r_T$
- Incremental mean: Given V_n and $G_{t,n}$ the n_{th} estimation of V and the n_{th} Return:

$$
\begin{aligned}
V_n(s_t) &= \frac{1}{n}\sum_{i=1}^{n} G_{t,i} \\
&= \frac{1}{n}(G_{t,n} + \sum_{i=1}^{n-1} G_{t,i}) \\
&= \frac{1}{n}(G_{t,n} + (n-1)\frac{1}{n-1}\sum_{i=1}^{n-1} G_{t,i})
\end{aligned}
$$

Incremental Monte-Carlo Updates

- Update $V(s)$ incrementally after episode $s_0, a_0, r_1, s_1, a_1, r_2, \ldots, s_{T-1}, a_{T-1}, r_T$
- Incremental mean: Given V_n and $G_{t,n}$ the n_{th} estimation of V and the n_{th} Return:

$$
\begin{aligned}
V_n(s_t) &= \frac{1}{n}\sum_{i=1}^{n} G_{t,i} \\
&= \frac{1}{n}(G_{t,n} + \sum_{i=1}^{n-1} G_{t,i}) \\
&= \frac{1}{n}(G_{t,n} + (n-1)\frac{1}{n-1}\sum_{i=1}^{n-1} G_{t,i}) \\
&= \frac{1}{n}(G_{t,n} + (n-1)V_{n-1}(s_t))
\end{aligned}
$$

Incremental Monte-Carlo Updates

- Update $V(s)$ incrementally after episode $s_0, a_0, r_1, s_1, a_1, r_2, \ldots, s_{T-1}, a_{T-1}, r_T$
- Incremental mean: Given V_n and $G_{t,n}$ the n_{th} estimation of V and the n_{th} Return:

$$\begin{aligned}
V_n(s_t) &= \frac{1}{n}\sum_{i=1}^{n} G_{t,i} \\
&= \frac{1}{n}(G_{t,n} + \sum_{i=1}^{n-1} G_{t,i}) \\
&= \frac{1}{n}(G_{t,n} + (n-1)\frac{1}{n-1}\sum_{i=1}^{n-1} G_{t,i}) \\
&= \frac{1}{n}(G_{t,n} + (n-1)V_{n-1}(s_t)) \\
V(s) &= V(s) + \frac{1}{N(s)}(G_t - V(s))
\end{aligned}$$

Incremental Monte-Carlo Updates

- Update $V(s)$ incrementally after episode $s_0, a_0, r_1, s_1, a_1, r_2, \ldots, s_{T-1}, a_{T-1}, r_T$
- Incremental mean: Given V_n and $G_{t,n}$ the n_{th} estimation of V and the n_{th} Return:

$$\begin{aligned}
V_n(s_t) &= \frac{1}{n}\sum_{i=1}^{n} G_{t,i} \\
&= \frac{1}{n}(G_{t,n} + \sum_{i=1}^{n-1} G_{t,i}) \\
&= \frac{1}{n}(G_{t,n} + (n-1)\frac{1}{n-1}\sum_{i=1}^{n-1} G_{t,i}) \\
&= \frac{1}{n}(G_{t,n} + (n-1)V_{n-1}(s_t)) \\
V(s) &= V(s) + \frac{1}{N(s)}(G_t - V(s))
\end{aligned}$$

- New Estimate = Old Estimate + Step $*$ (Target $-$ Old Estimate)

Incremental Monte-Carlo Updates

Incremental Monte-Carlo Updates

- Usually to avoid having to save the number of visits, a parameter $\alpha \in [0, 1]$ is used instead

Incremental Monte-Carlo Updates

- Usually to avoid having to save the number of visits, a parameter $\alpha \in [0, 1]$ is used instead
- $V(s) = V(s) + \alpha(G_t - V(s))$

Incremental Monte-Carlo Updates

- Usually to avoid having to save the number of visits, a parameter $\alpha \in [0, 1]$ is used instead
- $V(s) = V(s) + \alpha(G_t - V(s))$
- This is done incrementally while going through the episode.

Off Policy Control: Q-Learning

Off Policy Control: Q-Learning

Definition: State-Action Value function

The state-action value function $q_\pi(s,a)$ of an MDP is the expected Return for when starting in a state s, and taking action a following the policy π.

$$Q_\pi(s,a) = \mathbb{E}_\pi[G_t | s_t = s, a_t = a]$$
$$V_\pi(s) = \sum_{a \in A} \pi(a|s).Q(s,a)$$

Off Policy Control: Q-Learning

Definition: State-Action Value function

The state-action value function $q_\pi(s, a)$ of an MDP is the expected Return for when starting in a state s, and taking action a following the policy π.

$$Q_\pi(s, a) = \mathbb{E}_\pi[G_t | s_t = s, a_t = a]$$
$$V_\pi(s) = \sum_{a \in A} \pi(a|s).Q(s, a)$$

Bellman Expectation Equation for state-action value function:

$$Q_\pi(s, a) = \mathbb{E}_\pi[r_{t+1} + \gamma Q_\pi(s_{t+1}, a_{t+1}) | s_t = s, a_t = a] \tag{2}$$

Bellman Optimality Equation for state-action value function:

$$Q_\pi(s_t, a_t) = R_{s_t}^{a_t} + \gamma \max_{a \in A} Q_\pi(s_{t+1}, a) \tag{3}$$

Off Policy Q-Learning

Off Policy Q-Learning

- Lets try to approximate the optimal state-action value function Q^* using the Bellman optimality equation.

Off Policy Q-Learning

- Lets try to approximate the optimal state-action value function Q^* using the Bellman optimality equation.
- We can define the Bellman error as follows:

$$Bellman\_error = \underbrace{R_{s_t}^{a_t} + \gamma \max_{a \in A} Q(s_{t+1}, a)}_{\textit{Bellman equation approximation}} - Q(s_t, a_t)$$

Off Policy Q-Learning

- Lets try to approximate the optimal state-action value function Q^* using the Bellman optimality equation.
- We can define the Bellman error as follows:

$$Bellman\_error = \underbrace{R_{s_t}^{a_t} + \gamma \max_{a \in A} Q(s_{t+1}, a)}_{\textit{Bellman equation approximation}} - Q(s_t, a_t)$$

- Q-Learning consists on repeatedly adjusting Q to minimize the Bellman error.

Off Policy Q-Learning

- Lets try to approximate the optimal state-action value function Q^* using the Bellman optimality equation.
- We can define the Bellman error as follows:

$$Bellman\_error = \underbrace{R_{s_t}^{a_t} + \gamma \max_{a \in A} Q(s_{t+1}, a)}_{\textit{Bellman equation approximation}} - Q(s_t, a_t)$$

- Q-Learning consists on repeatedly adjusting Q to minimize the Bellman error.
- For each tuple $s_t, a_t, r_{t+1}, s_{t+1}$ in a given episode, we apply the following:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha(r_{t+1} + \gamma \max_{a \in A} Q(s_{t+1}, a) - Q(s_t, a_t))$$

Off Policy Q-Learning

- Lets try to approximate the optimal state-action value function Q^* using the Bellman optimality equation.
- We can define the Bellman error as follows:

$$Bellman\_error = \underbrace{R_{s_t}^{a_t} + \gamma \max_{a \in A} Q(s_{t+1}, a)}_{\textit{Bellman equation approximation}} - Q(s_t, a_t)$$

- Q-Learning consists on repeatedly adjusting Q to minimize the Bellman error.
- For each tuple $s_t, a_t, r_{t+1}, s_{t+1}$ in a given episode, we apply the following:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha(r_{t+1} + \gamma \max_{a \in A} Q(s_{t+1}, a) - Q(s_t, a_t))$$

- α is the learning rate $\alpha \in [0, 1]$

Exploration-Exploitation Trade off Q-Learning

Exploration-Exploitation Trade off Q-Learning

- The Q-Learning algorithm will only learn from states and actions it visits, so there is no visit to new sub-optimal space which reduces the exploration.

Exploration-Exploitation Trade off Q-Learning

- The Q-Learning algorithm will only learn from states and actions it visits, so there is no visit to new sub-optimal space which reduces the exploration.
- The agent should be able to explore sub-optimal states in order increase its exploration of states and actions.

Exploration-Exploitation Trade off Q-Learning

- The Q-Learning algorithm will only learn from states and actions it visits, so there is no visit to new sub-optimal space which reduces the exploration.
- The agent should be able to explore sub-optimal states in order increase its exploration of states and actions.
- Solution: $\epsilon - greedy$ approach ($\epsilon \in [0, 1]$):

Exploration-Exploitation Trade off Q-Learning

- The Q-Learning algorithm will only learn from states and actions it visits, so there is no visit to new sub-optimal space which reduces the exploration.
- The agent should be able to explore sub-optimal states in order increase its exploration of states and actions.
- Solution: $\epsilon - greedy$ approach ($\epsilon \in [0, 1]$):
  - With probability $1 - \epsilon$ choose the optimal action that maximizes the Q function

Exploration-Exploitation Trade off Q-Learning

- The Q-Learning algorithm will only learn from states and actions it visits, so there is no visit to new sub-optimal space which reduces the exploration.
- The agent should be able to explore sub-optimal states in order increase its exploration of states and actions.
- Solution: $\epsilon - greedy$ approach ($\epsilon \in [0, 1]$):
  - With probability $1 - \epsilon$ choose the optimal action that maximizes the Q function
  - With probability ϵ choose a random action

The Q-Learning step-by-step Algorithm

Algorithm 3 Q-Learning Algorithm

Initialize $Q(s, a) \forall s \in S, a \in A$ randomly and $Q(terminal\_state, .) = 0$
for $e = 1$ to E **do**
  Initialize s
  while $s \neq terminal\_state$ **do**
    Choose a from s using $\epsilon - greedy$
    Take action a and observe s' and r
    $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a' \in A} Q(s', a') - Q(s, a)]$
  end while
end for
return Q

Take-Home Message of this Introduction Course

Take-Home Message of this Introduction Course

- Reinforcement Learning is a way of training based on a reward system

Take-Home Message of this Introduction Course

- Reinforcement Learning is a way of training based on a reward system
- RL is simply "the science of sequential decision making"

Take-Home Message of this Introduction Course

- Reinforcement Learning is a way of training based on a reward system
- RL is simply "the science of sequential decision making"
- RL consists on a smart Agent interacting with an Environment in order to learn how it works resulting in maximization of the reward

Take-Home Message of this Introduction Course

- Reinforcement Learning is a way of training based on a reward system
- RL is simply "the science of sequential decision making"
- RL consists on a smart Agent interacting with an Environment in order to learn how it works resulting in maximization of the reward
- Depending on the case scenario, a choice of algorithms should be done to solve such a problem