

Introduction to Reinforced Learning

Bordeaux

Milan R. Rapać

Chair of Automatic Control
Computing and Control Department
Faculty of Technical Sciences
University of Novi Sad
Novi Sad • Serbia

October 2, 2023

Outline

- 1 Introduction
- 2 Markov Decision Processes (MDPs)
- 3 Bellman Equation
- 4 RL: Model-Based Tabular Methods
- 5 RL: Tabular Monte Carlo Methods
- 6 RL: Temporal-Difference Methods
- 7 Approximate Reinforcement Learning

Introduction

- 1 Introduction
 - What is it? Where is it applied?
 - Selling Points
 - How does it relate to the (rest of) Control Theory?
- 2 Markov Decision Processes (MDPs)
- 3 Bellman Equation
- 4 RL: Model-Based Tabular Methods
- 5 RL: Tabular Monte Carlo Methods
- 6 RL: Temporal-Difference Methods
- 7 Approximate Reinforcement Learning

Introduction

What is it? Where is it applied?

The Map of Control Theory

Linear

- model reference adaptive
- iterative learning control
- optimal
- lqr
- full state feedback K
- pid
- loop shaping
- gain scheduling
- bang-bang
- backstepping
- genetic algorithms
- fuzzy control
- intelligent

Nonlinear

- sliding mode
- backstepping

Robust

- H_{∞}
- Lyapunov based control
- Lyapunov stability
- block diagrams
- bode plots
- nichols chart

Planning

- sine
- step
- impulse
- mapping
- holonomic
- nonholonomic
- redundant
- constraints
- optimal
- stability
- nonminimum phase
- root locus
- nyquist plots
- phase plane

System Analysis

- performance
- stability
- margins
- SAFETY
- controllability
- observability
- $C_m = [B, AB, A^2B]$
- $O_m = [C, CA, CA^2]$

Modeling & Simulation

- transfer functions
- minimum realizations
- system id
- linearization
- particle
- sigma-point
- kalman filter
- observer
- sensor fusion
- calibration
- GPS
- Camera
- IMU

State Estimation

- filtering
- tracking
- observer
- kalman filter
- sigma-point
- particle

Hybrid System

- minimum realizations
- system id
- linearization
- particle
- sigma-point
- kalman filter
- observer
- sensor fusion
- calibration
- GPS
- Camera
- IMU

Predictive

- model predictive control
- robust mpc
- predictive
- linear mpc
- reinforcement learning
- explore vs exploit

Intelligent

- reinforcement learning
- explore vs exploit

Central Hub: Control Methods

- continuous
- discrete
- frequency
- feedback
- feedforward
- laplace
- D2C
- C2D

Other Notes:

- gang of six
- controllability
- observability
- $C_m = [B, AB, A^2B]$
- $O_m = [C, CA, CA^2]$
- linear state space
- $\dot{x} = Ax + Bu$
- $y = Cx + Du$
- nonlinear state space
- $\frac{dx}{dt} = f(x, u)$
- $y = g(x, u)$
- hybrid system
- simulation
- linearization
- particle
- sigma-point
- kalman filter
- observer
- sensor fusion
- calibration
- GPS
- Camera
- IMU
- filtering
- tracking
- observer
- kalman filter
- sigma-point
- particle
- linearization
- particle
- sigma-point
- kalman filter
- observer
- sensor fusion
- calibration
- GPS
- Camera
- IMU
- filtering
- tracking
- observer
- kalman filter
- sigma-point
- particle
- linearization
- particle
- sigma-point
- kalman filter
- observer
- sensor fusion
- calibration
- GPS
- Camera
- IMU
- filtering
- tracking
- observer
- kalman filter
- sigma-point
- particle
- linearization
- particle
- sigma-point
- kalman filter
- observer
- sensor fusion
- calibration
- GPS
- Camera
- IMU
- filtering
- tracking
- observer
- kalman filter
- sigma-point
- particle
- linearization
- particle
- sigma-point
- kalman filter
- observer
- sensor fusion
- calibration
- GPS
- Camera
- IMU
- filtering
- tracking
- observer
- kalman filter
- sigma-point
- particle
- linearization
- particle
- sigma-point
- kalman filter
- observer
- sensor fusion
- calibration
- GPS
- Camera
- IMU
- filtering
- tracking
- observer
- kalman filter
- sigma-point
- particle
- linearization
- particle
- sigma-point
- kalman filter
- observer
- sensor fusion
- calibration
- GPS
- Camera
- IMU
- filtering
- tracking
- observer
- kalman filter
- sigma-point
- particle
- linearization
- particle
- sigma-point
- kalman filter
- observer
- sensor fusion
- calibration
- GPS
- Camera
- IMU
- filtering
- tracking
- observer
- kalman filter
- sigma-point
- particle
- linearization
- particle
- sigma-point
- kalman filter
- observer
- sensor fusion
- calibration
- GPS
- Camera
- IMU
- filtering
- tracking
- observer
- kalman filter
- sigma-point
- particle
- linearization
- particle
- sigma-point
- kalman filter
- observer
- sensor fusion
- calibration
- GPS
- Camera
- IMU
- filtering
- tracking
- observer
- kalman filter
- sigma-point
- particle
- linearization
- particle
- sigma-point
- kalman filter
- observer
- sensor fusion
- calibration
- GPS
- Camera
- IMU
- filtering
- tracking
- observer
- kalman filter
- sigma-point
- particle
- linearization
- particle
- sigma-point
- kalman filter
- observer
- sensor fusion
- calibration
- GPS
- Camera
- IMU
- filtering
- tracking
- observer
- kalman filter
- sigma-point
- particle
- linearization
- particle
- sigma-point
- kalman filter
- observer
- sensor fusion
- calibration
- GPS
- Camera
- IMU
- filtering
- tracking
- observer
- kalman filter
- sigma-point
- particle
- linearization
- particle
- sigma-point
- kalman filter
- observer
- sensor fusion
- calibration
- GPS
- Camera
- IMU
- filtering
- tracking
- observer
- kalman filter
- sigma-point
- particle
- linearization
- particle
- sigma-point
- kalman filter
- observer
- sensor fusion
- calibration
- GPS
- Camera
- IMU
- filtering
- tracking
- observer
- kalman filter
- sigma-point
- particle
- linearization
- particle
- sigma-point
- kalman filter
- observer
- sensor fusion
- calibration
- GPS
- Camera
- IMU
- filtering
- tracking
- observer
- kalman filter
- sigma-point
- particle
- linearization
- particle
- sigma-point
- kalman filter
- observer
- sensor fusion
- calibration
- GPS
- Camera
- IMU
- filtering
- tracking
- observer
- kalman filter
- sigma-point
- particle
- linearization
- particle
- sigma-point
- kalman filter
- observer
- sensor fusion
- calibration
- GPS
- Camera
- IMU
- filtering
- tracking
- observer
- kalman filter
- sigma-point
- particle
- linearization
- particle
- sigma-point
- kalman filter
- observer
- sensor fusion
- calibration
- GPS
- Camera
- IMU
- filtering
- tracking
- observer
- kalman filter
- sigma-point
- particle
- linearization
- particle
- sigma-point
- kalman filter
- observer
- sensor fusion
- calibration
- GPS
- Camera
- IMU
- filtering
- tracking
- observer
- kalman filter
- sigma-point
- particle
- linearization
- particle
- sigma-point
- kalman filter
- observer
- sensor fusion
- calibration
- GPS
- Camera

The Map of Control Theory

central hub: control methods (continuous, discrete, frequency)

modeling & simulation: transfer functions ($G(s) = \frac{b^s}{s^2 + 2\gamma\omega s + \omega^2}$), minimum realizations, linearization ($G(s) = \frac{1}{s^2 + 1}$), particle, sigma-point, kalman filter, observer, tracking, sensor fusion, calibration, GPS, camera, imu, y = $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} y + b$

state estimation: state estimation, transfer functions, minimum realizations, linearization, particle, sigma-point, kalman filter, observer, tracking, sensor fusion, calibration, GPS, camera, imu, y = $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} y + b$

planning: planning, mapping, holonomic, nonholonomic, redundant, constraints, optimal, sine, step, impulse, feedback, time, frequency, laplace, feedforward, discrete, continuous

robust: robust, lyapunov based control, loop shaping, pid, hinf, full state feedback, linear, lead-lag, gain scheduling, bang-bang, backstepping, filtering, tracking

optimal: optimal, sine, step, impulse, mapping, holonomic, nonholonomic, redundant, constraints, optimal, sine, step, impulse

adaptive: adaptive, extremum-seeking, iterative learning control, model reference adaptive, again!

predictive: predictive, model predictive control, robust mpc, linear mpc

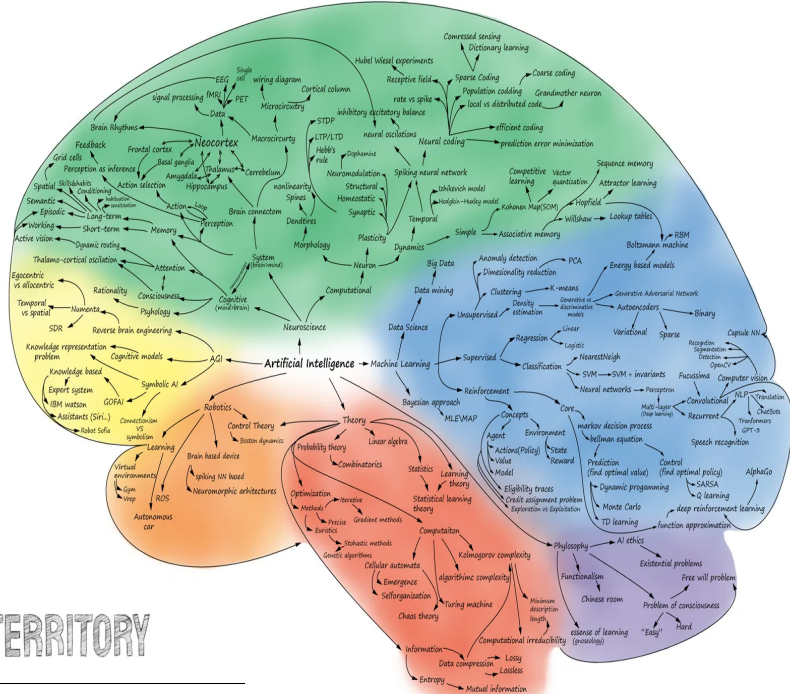
nonlinear: nonlinear, sliding mode, bang-bang, backstepping

linear: linear, lead-lag, gain scheduling, bang-bang

intelligent: intelligent, genetic algorithms, fuzzy control, reinforcement learning, explore vs exploit

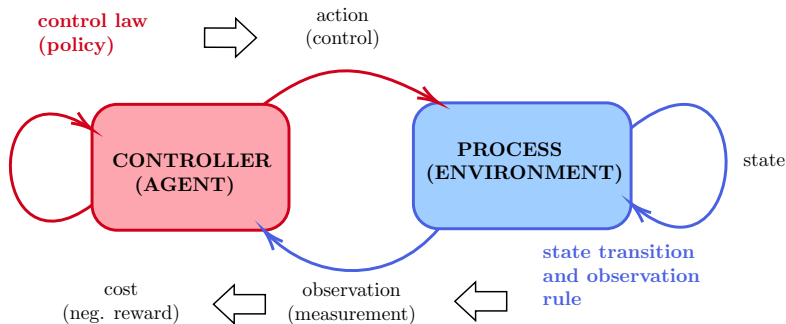
other topics: lyapunov stability, block diagrams, bode plots, nichols chart, root locus, nyquist plots, phase plane, stability, nonminimum phase, margins, SAFETY, controllability, observability, $C_m = [B, AB, A^2B]$, $O_m = [C, CA, CA^2]$, gang of six, simulation, system id, linearization, minimum realizations, transfer functions, sigma-point, kalman filter, observer, tracking, sensor fusion, calibration, GPS, camera, imu, y = $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} y + b$

credit: brian douglas © 2020 (M: Engineering Media)



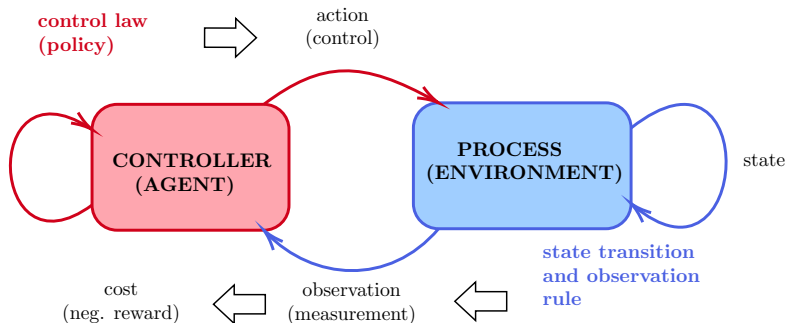
What is Reinforcement Learning?

RL is a framework for **sequential decision making**.



What is Reinforcement Learning?

RL is a framework for **sequential decision making**.



Reinforcement Learning is

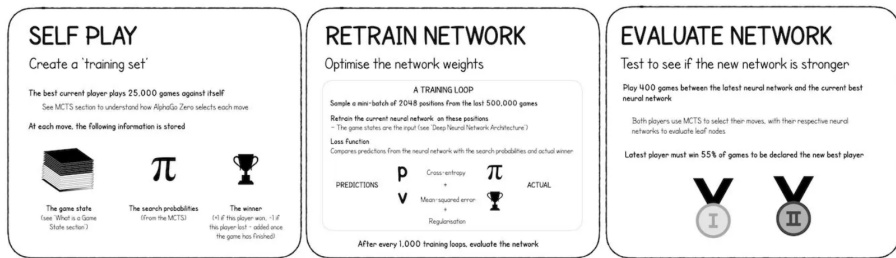
a **model-free** framework for solving **optimal control** problems stated as **Markov Decision Processes (MDPs)**.

Selected Application: Go RL Go!

... or how RL reappeared in headlines

In 2016, the computer program **AlphaGo** captured the world's attention when it **defeated the legendary Go player Lee Sedol**. The ancient board game of Go is one of the most complex games ever devised, *with more possible board configurations than atoms in the universe*. It was a longstanding grand challenge for artificial intelligence and AlphaGo's 4-1 win was considered by many to be a decade ahead of its time. The system was invented by **DeepMind**, co-founded by scientist Demis Hassabis. Five months earlier, AlphaGo had beaten European champion Fan Hui, becoming *the first program to defeat a professional player*.

The training pipeline for AlphaGo Zero consists of three stages, executed in parallel



<https://medium.com/applied-data-science/alphago-zero-explained-in-one-diagram-365f5abf67e0>

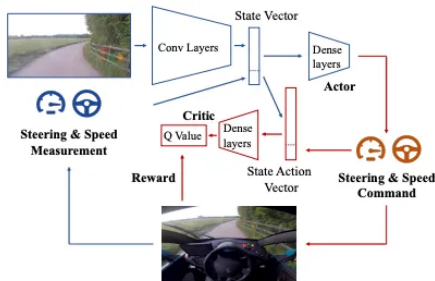
Selected Application: Autonomous Driving

Learning to drive in a day

In 2018 RL was used **to train an agent to drive from scratch**.

The agent was able to both drive and navigate successfully.

The agent was **later also successfully tested in the field**.

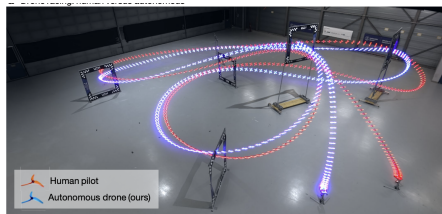


The original paper is freely available, and can be found [here](#). Further discussion can be found [here](#) and in the following [video](#).

Selected Application: Drone Racing

Going head-to-head against a champion...

First-person view (FPV) drone racing is a televised sport in which professional competitors pilot high-speed aircraft through a 3D circuit. Each pilot sees the environment from the perspective of their drone by means of video streamed from an onboard camera.



In 2013, a group of researchers from Zurich introduced **Swift** – an autonomous system that can race physical vehicles at the level of the human world champions.

Swift uses deep Reinforcement Learning agent trained using a combination of simulated data and data collected from the physical world.

The original paper can be accessed using [here](#).

Further Reading

... including lists of some potentially interesting applications

Interesting recent review articles:

[[Li, 2018](#)]

[[Arulkumaran et al., 2017](#)]

[[Nian et al., 2020](#)]

An excellent and very popular reference [[Sutton and Barto, 2018](#)].

Some other informative resources:

Medium: 9 Awesome Applications of Reinforcement Learning

Medium: RL Applications

Introduction

Selling Points

Some properties of RL I

If I wanted to **sell** RL to you, I would say that:

- RL is **general**. It can uniformly handle
 - ▶ processes of strongly nonlinear and/or stochastic **dynamics**;
 - ▶ time-varying processes;
 - ▶ arbitrarily complex **cost** functions (performance indices).
- RL is **model-free**. It does not require explicit model of system dynamics, or even an explicit expression for the cost.
- RL inherently operates in **discrete time** – it can be directly implemented on a digital computer.

Some properties of RL II

If I wanted to **sell** RL to you, I would say that:

- RL provides **an alternative perspective to**, and **widens areas of application of** much of the modern control theory.
- RL provides a direct link between control theory and **artificial intelligence**, in two ways:
 - ▶ It enables systematic applications of AI methods within control system design;
 - ▶ It highlights a methodology enabling application of control theory to many demanding applications – traditionally in the AI domain;
- RL provides a direct link between control theory and **game theory**, **multi-agent systems** theory, etc.

Introduction

How does it relate to the (rest of) Control Theory?

[illegible]

Engineering Media

A note on terminology

When an AI researcher says \mathfrak{X} control engineer should understand \mathfrak{Y}

they talk about ...	but it actually means ...
agent / decision-maker	controller
environment	plant / process / system / object
gains, rewards, penalties, losses	performance indices
model-based RL	optimal control
off-line model-learning RL	off-line identification + optimal control
on-line model-learning RL	indirect adaptive control
model-free RL	direct adaptive control
completely observable	all process states are directly measured
partially observable	only the outputs are measured

Markov Decision Processes (MDPs)

1 Introduction

2 Markov Decision Processes (MDPs)

- The Model
- Decision Making using State and Action Values
- Exploration vs. Exploitation

3 Bellman Equation

4 RL: Model-Based Tabular Methods

5 RL: Tabular Monte Carlo Methods

6 RL: Temporal-Difference Methods

7 Approximate Reinforcement Learning

Markov Decision Processes (MDPs)

The Model

Markov Decision Processes

MDP is a model of the **environment (process)**. It tells us ...

- ... how the **state** of the process is changing in reaction to the applied **(control) actions**
- ... what **observations** the agent (controller) may receive

Based on the received observation, the agent (controller) evaluates assesses the **immediate (short-term)** effects of the applied action. In the AI community, it is often said that the agent evaluates a **reward**, while in the control community one often speaks of a **penalty**. The two positions are philosophically different, but essentially equivalent

Deterministic MDP

$$s^+ = f(s, a)$$

$$r = h(a, u)$$

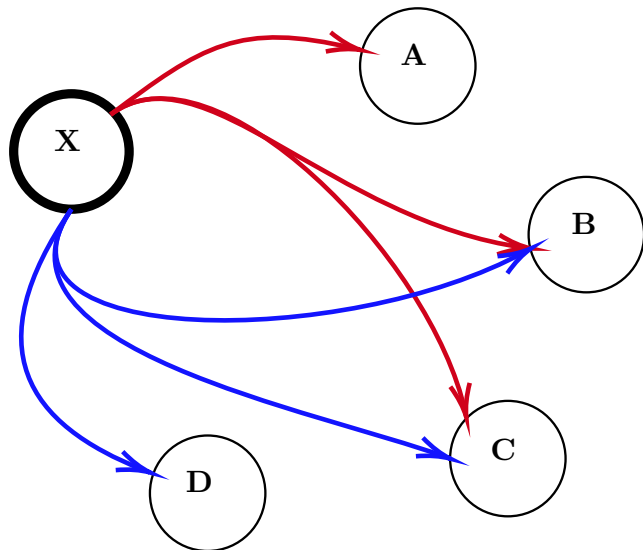
Stochastic MDP

$$p(s^+, r | s, a) =$$

$$\mathbb{P} \{ S^+ = s^+, R = r | S = s, A = a \}$$

MDP as a graph

... a conceptual representation



Gain

The goal is never to optimize short-term returns. We are always interested in optimizing measures of long-term success: maximizing long-term **gains** (or minimizing long-term **losses**).

Let r_0, r_1, \dots be a sequence of rewards obtained in subsequent time instances. The **gain** is defined as

Deterministic MDP

$$g = \sum_{k=0}^T \gamma^k r_k$$

Stochastic MDP

$$g = \mathbb{E} \left\{ \sum_{k=0}^T \gamma^k R_k \right\}$$

where the **discount factor** γ (typically chosen between 0 and 1) accounts for the fact that high rewards expected in the far future are often favored less than substantially smaller rewards to be received immediately.

Decision Policy

... a.k.a. Control Law

Decision policy describes actions of the **controller**: it tells us how controller will act *in each possible observed state* of the environment.

Deterministic Policy

$$a = \pi(s)$$

Stochastic Policy

$$\pi(a|s) = \mathbb{P}\{A = a | S = s\}$$

One can apply both deterministic and stochastic policies regardless of the nature of the environment (to both stochastic and deterministic MDPs).

Gain revisited

as a function of the initial state and the decision policy

Given an MDP in a certain initial state s_0 controlled by an agent applying policy π , one can observe a sequence of applied actions, environment states and rewards.

deterministic case : $s_0 \rightarrow a_0 \rightarrow r_0, s_1 \rightarrow a_1 \rightarrow r_1, s_2 \rightarrow \dots$

stochastic case : $s_0 \rightarrow A_0 \rightarrow R_0, S_1 \rightarrow A_1 \rightarrow R_1, S_2 \rightarrow \dots$

Once the initial state is fixed, together with the decision policy, these sequences are fixed as well, and so is the gain! Therefore, it is possible to associate a map between initial state and decision policies with the resulting gain for the given MDP.

$$g_{\pi}(s) = \mathbb{E}_{\pi} \left\{ \sum_{k=0}^T \gamma^k R_k \mid S_0 = s \right\}$$

Markov Decision Processes (MDPs)

Decision Making using State and Action Values

The Value

of a state, or of an action in state, for the given policy

State-Value Function for policy

The value of a state s for the policy π is the gain that a controller following decision policy π attains when the environment starts from the initial state s .

$$v_{\pi}(s) = g_{\pi}(s)$$

Action-Value Function for policy

The value of an action a in state s for the policy π is the gain that a controller attains when the environment starts from state s when the initial action a and the policy π is followed afterward

$$q_{\pi}(s, a) = g \text{ when } s_0 = s, a_0 = a \text{ and policy } \pi \text{ is used for } k \geq 1$$

The Optimal Values

of a state, or of an action in state

Let \mathcal{P} denote the set of all possible policies (regardless if they are deterministic or stochastic).

Optimal State-Value

The optimal value of a state is the maximal value of that state under any policy:

$$v^*(s) = \max_{\pi \in \mathcal{P}} v_{\pi}(s)$$

Optimal State-Action Value

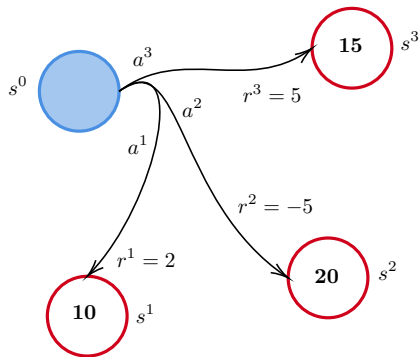
The optimal value of an action within a state is the maximal value of that action within that state under any policy:

$$q^*(s, a) = \max_{\pi \in \mathcal{P}} q_{\pi}(s, a)$$

Decision Making using Optimal State Values

Assume that I know values of all states, and assuming that I know that the environment is in state s_0 ,

- How to choose the best action?
- What else do I need to know in order to be able to choose?



$\gamma = 0.9$				
a	s^+	$v(s^+)$	r	g
a^1	s^1	10	2	11
a^2	s^2	20	-5	13
a^3	s^3	15	5	18.5

$$a^* = \pi^*(s) \in \operatorname{argmax}_{a \in \mathcal{A}} \{h(s, a) + \gamma v^*(f(s, a))\}$$

Decision Making using Optimal State-Action Values

Assume that I know values of all actions in all states, and assuming that I know that the environment is in state s_0 ,

- How to choose the best action?
- What else do I need to know in order to be able to choose?

$\gamma = 0.9$	
a	$q(s, a)$
a^1	11
a^2	13
a^3	18.5

$$a^* = \pi^*(s) \in \operatorname{argmax}_{a \in \mathcal{A}} q^*(s, a)$$

Decision Making using Values

Concluding remarks...

Arguably, state values are easier to understand in comparison to the state-action values, however **state-action values (q -values) are more convenient for decision making purposes.**

- q -values store all relevant information for decision making. The explicit model is not needed!
- It is not possible to decide based on the state-values alone. In this case, an explicit model is also necessary.
- Storing q -values is more expensive than storing v -values, however in most practical cases the states are abundant while the actions are relatively few, so the practical difference is actually not that significant.

Markov Decision Processes (MDPs)

Exploration vs. Exploitation

Greedy and Optimal Policy

Given an *arbitrary* (not necessarily optimal) state value function v , or state action value function q , the **greedy policy** is defined as

$$a^{\text{greedy}} = \pi_v^{\text{greedy}}(s) \in \operatorname{argmax}_{a \in \mathcal{A}} \{h(s, a) + \gamma v(f(s, a))\}$$

$$a^{\text{greedy}} = \pi_q^{\text{greedy}}(s) \in \operatorname{argmax}_{a \in \mathcal{A}} q(s, a)$$

Optimal Policy

The greedy policy with respect to the optimal value functions v^* and q^* is the optimal policy.

Exploration vs. Exploitation

If the estimated values of v and q are not optimal, the greedy policy with respect to them need not be good, nor even reasonable.

Even worse, by consistently following a non-optimal greedy policy we keep revisiting the same combinations of states and actions over-and-over again. We continue to **exploit** existing **incomplete knowledge** of the system, without any attempt to **explore**, to **experiment**, to advance our understanding, and possibly to **find better** actions in certain situations.

In situations when we are **learning** (because we do not know the system sufficiently, or we know it but anticipate that it will change in the future) **completely exploitative (greedy) policy should be avoided**, and **explorative component must be added** to the decision-making process.

Random Policy

... as an example of a **completely explorative policy**

Random policy is suitable in situations where the rewards (and gain) are completely neglected, and the goal is not to maximize the gain but to learn as much as possible about the environment (i.e. about the controlled system).

$$\pi^{\text{random}}(s) = \text{choose } a \text{ for } \mathcal{A} \text{ randomly with uniform probability}$$

ε -greedy Policy

... as an example of a **balanced policy**

In reality, one is interested in **simultaneously** maximizing the gain **and** exploring the environment. This is a multi-criteria optimization problem, and it is necessary to establish a **tradeoff** between two **conflicting** goals.

$$\pi^\varepsilon(s) = \begin{cases} \pi^{\text{random}}(s) & \text{with probability } \varepsilon \\ \pi^{\text{greedy}}(s) & \text{with probability } 1 - \varepsilon \end{cases}$$

Bellman Equation

- 1 Introduction
- 2 Markov Decision Processes (MDPs)
- 3 Bellman Equation**
 - Finding Policy Values in the Deterministic Case
 - Finding Policy Values in the Stochastic Case
 - Finding Optimal Values and Optimal Policies
- 4 RL: Model-Based Tabular Methods
- 5 RL: Tabular Monte Carlo Methods
- 6 RL: Temporal-Difference Methods
- 7 Approximate Reinforcement Learning

Bellman Equation & Dynamic Programming

What is it? What problem does it solve?

Bellman Equation

Bellman Equation (in its various forms) is the equation which, if solved, would specify optimal action of every possible state. Any policy satisfying the Bellman Equation is an optimal policy.

Dynamic Programming

The term dynamic programming is often used in somewhat different contexts, however in the present context it will be used solely to indicate a group of techniques for finding an optimal policy by solving Bellman Equation.

Bellman Equation

Finding Policy Values in the Deterministic Case

Deterministic Bellman Equation

... or at least the seed from which it will be derived

$$g_k = \sum_{i=0}^T \gamma^i r_{k+i} = r_0 + \sum_{i=1}^T \gamma^i r_{k+i} = r_0 + \gamma \sum_{i=0}^T \gamma^i r_{k+1+i}$$

$$g_k = r_k + \gamma g_{k+1}$$

$$g_{\pi}(s) = r + \gamma g_{\pi}(s^+)$$

A recursive formula for state values (Deterministic Case)

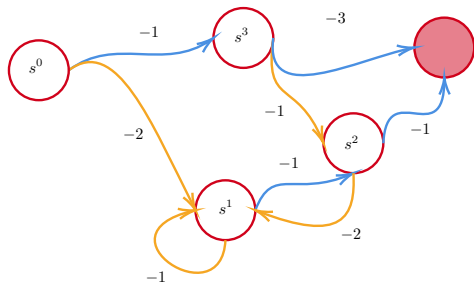
$$v_{\pi}(s) = h(s, \pi(s)) + \gamma v_{\pi}(f(s, \pi(s)))$$

Example

Write Bellman Equations and evaluate State Values

POLICY

Use blue action.



Bellman Equation

$$v^0 = -1 + \gamma v^3$$

$$v^1 = -1 + \gamma v^2$$

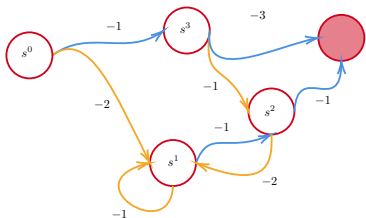
$$v^2 = -1 + \gamma v^{\text{term}}$$

$$v^3 = -3 + \gamma v^{\text{term}}$$

Bellman Equations for a given policy are always a set of linear equations!

Evaluation of State Values for a Given Policy

... by solving a linear matrix equation



$$\underbrace{\begin{bmatrix} v^0 \\ v^1 \\ v^2 \\ v^3 \end{bmatrix}}_{\mathbf{v}} = \gamma \underbrace{\begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}}_{\mathbf{A}} \underbrace{\begin{bmatrix} v^0 \\ v^1 \\ v^2 \\ v^3 \end{bmatrix}}_{\mathbf{v}} + \gamma \underbrace{\begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}}_{\mathbf{r}} v^{\text{term}} + \underbrace{\begin{bmatrix} -1 \\ -1 \\ -1 \\ -3 \end{bmatrix}}_{\mathbf{r}}$$

Direct Solution

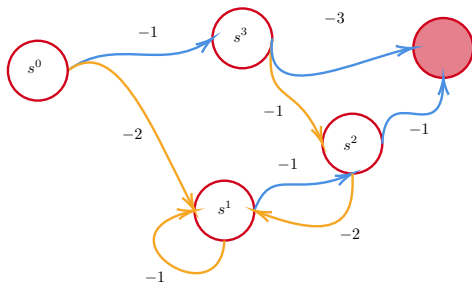
$$\mathbf{v} = (\mathbf{I} - \gamma \mathbf{A})^{-1} \mathbf{r}$$

Iterative Solution

$$\mathbf{v}^{k+1} = \gamma \mathbf{A} \mathbf{v}^k + \mathbf{r}, \quad \mathbf{v}^0 \in \text{rnd}$$

Evaluation of State Values for a Given Policy

... by backtracking from the terminal state



- choose $v^{\text{term}} = 0$
- Find all states from which the terminal state can be reached directly (s^2 and s^3)

$$v^2 = -1 + \gamma v^{\text{term}} = -1$$

$$v^3 = -3 + \gamma v^{\text{term}} = -3$$

- Find a state from which it is possible to directly reach a state of known value (s^0 and s^1)

$$v^0 = -1 + \gamma v^3 = -3.7$$

$$v^1 = -1 + \gamma v^2 = -1.9$$

Bellman Equation

... for the **state-action value** of an arbitrary **given policy** in the **deterministic case**

$$g_k = r_k + \gamma g_{k+1}$$

$$q_\pi(s_k, a_k) = r_k + \gamma q_\pi(s_{k+1}, a_{k+1})$$

A recursive formula for state-action values (Deterministic Case)

$$q_\pi(s, a) = h(s, a) + \gamma q_\pi(f(s, a), \pi(f(s, a)))$$

Note that this is still a set of linear equations (although in more unknown variables compared to the equations used to evaluate state values).

Bellman Equation

Finding Policy Values in the Stochastic Case

Bellman Equation for evaluating State-Values

... of a given policy in the stochastic case

$$g_{\pi}(s) = \mathbb{E}_{\pi} \left\{ \sum_{i=0}^T \gamma^i R_{k+i} \mid S_0 = s \right\} = \mathbb{E}_{\pi} \left\{ R_0 + \gamma \sum_{i=1}^T \gamma^{i-1} R_i \mid S_0 = s \right\}$$

$$g_{\pi}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \mathbb{E}_{\pi} \left\{ R_0 + \gamma \sum_{i=1}^T \gamma^{i-1} R_i \mid S_0 = s, A_0 = a \right\}$$

$$g_{\pi}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{\substack{s^+ \in \mathcal{S} \\ r \in \mathcal{R}}} p(s^+, r|s, a) \mathbb{E}_{\pi} \left\{ R_0 + \gamma \sum_{i=1}^T \gamma^{i-1} R_i \mid S_0 = s, A_0 = a, S_1 = s^+, R_0 = r \right\}$$

$$g_{\pi}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{\substack{s^+ \in \mathcal{S} \\ r \in \mathcal{R}}} p(s^+, r|s, a) \left[r + \gamma \mathbb{E}_{\pi} \left\{ \sum_{i=1}^T \gamma^{i-1} R_i \mid S_1 = s^+ \right\} \right]$$

A recursive formula for state values (Stochastic Case)

$$v_{\pi}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{\substack{s^+ \in \mathcal{S} \\ r \in \mathcal{R}}} p(s^+, r|s, a) [r + \gamma v_{\pi}(s^+)]$$

Bellman Equation for evaluating State-Action values

... of a given policy in the stochastic case

$$q_{\pi}(s, a) = \mathbb{E}_{\pi} \left\{ R_0 + \gamma \sum_{i=1}^T \gamma^{i-1} R_i \mid S_0 = s, A_0 = a \right\}$$

$$q_{\pi}(s, a) = \sum_{\substack{s^+ \in \mathcal{S} \\ r \in \mathcal{R}}} p(s^+, r | s, a) \mathbb{E}_{\pi} \left\{ R_0 + \gamma \sum_{i=1}^T \gamma^{i-1} R_i \mid S_0 = s, A_0 = a, S_1 = s^+, R_0 = r \right\}$$

$$q_{\pi}(s, a) = \sum_{\substack{s^+ \in \mathcal{S} \\ r \in \mathcal{R}}} p(s^+, r | s, a) \left[r + \gamma \mathbb{E}_{\pi} \left\{ \sum_{i=1}^T \gamma^{i-1} R_i \mid S_1 = s^+ \right\} \right]$$

$$q_{\pi}(s, a) = \sum_{\substack{s^+ \in \mathcal{S} \\ r \in \mathcal{R}}} p(s^+, r | s, a) \left[r + \gamma \sum_{a^+ \in \mathcal{A}} \pi(a^+ | s^+) \mathbb{E}_{\pi} \left\{ \sum_{i=1}^T \gamma^{i-1} R_i \mid S_1 = s^+, A_1 = a^+ \right\} \right]$$

A recursive formula for state values (Stochastic Case)

$$q_{\pi}(s, a) = \sum_{\substack{s^+ \in \mathcal{S} \\ r \in \mathcal{R}}} p(s^+, r | s, a) \left[r + \gamma \sum_{a^+ \in \mathcal{A}} \pi(a^+ | s^+) q_{\pi}(s^+, a^+) \right]$$

Bellman Equation

Finding Optimal Values and Optimal Policies

Bellman Equation

... for the state value of the optimal policy in the deterministic case

Since optimal policy is a policy, the recursive formula for computing v_π with $\pi \equiv \pi^*$ must hold

$$v_{\pi^*}(s) = h(s, \pi^*(s)) + \gamma v_{\pi^*}(f(s, \pi^*(s)))$$

Since v^* is optimal, the right hand side must be maximal among all possible policies

$$v_{\pi^*}(s) = \max_{\pi \in \mathcal{P}} \{g(s, \pi(s)) + \gamma v_{\pi^*}(f(s, \pi(s)))\}$$

Deterministic Bellman Equation for State Values

$$v^*(s) = \max_{a \in \mathcal{A}} \{g(s, a) + \gamma v^*(f(s, a))\}$$

Bellman Equation

... for the **state value** of the **optimal policy** in the **stochastic case**

Since optimal policy is a policy, the recursive formula for computing $v^* \equiv v_{\pi^*}$ holds

$$v_{\pi^*}(s) = \sum_{a \in \mathcal{A}} \pi^*(a|s) \sum_{\substack{s^+ \in \mathcal{S} \\ r \in \mathcal{R}}} p(s^+, r|s, a) [r + \gamma v_{\pi^*}(s^+)]$$

Since v^* is optimal, the right hand side must be maximal among all possible policies

$$v_{\pi^*}(s) = \max_{\pi \in \mathcal{P}} \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{\substack{s^+ \in \mathcal{S} \\ r \in \mathcal{R}}} p(s^+, r|s, a) [r + \gamma v_{\pi^*}(s^+)]$$

Stochastic Bellman Equation for State Values

$$v^*(s) = \max_{a \in \mathcal{A}} \sum_{\substack{s^+ \in \mathcal{S} \\ r \in \mathcal{R}}} p(s^+, r|s, a) [r + \gamma v^*(s^+)]$$

Bellman Equation

... for the state-action values of the optimal policy in the deterministic case

Since optimal policy is a policy, the recursive formula for computing q_π with $\pi \equiv \pi^*$ must hold

$$q_{\pi^*}(s, a) = h(s, a) + \gamma q_{\pi^*}(f(s, a), \pi^*(f(s, a)))$$

Since v^* is optimal, the right hand side must be maximal among all possible policies

$$q_{\pi^*}(s, a) = \max_{\pi \in \mathcal{P}} \{h(s, a) + \gamma q_\pi(f(s, a), \pi(f(s, a)))\}$$

Deterministic Bellman Equation for State-Action Values

$$q^*(s, a) = h(s, a) + \gamma \max_{a^+ \in \mathcal{A}} q^*(f(s, a), a^+)$$

Bellman Equation

... for the state-action values of the optimal policy in the stochastic case

Since optimal policy is a policy, the recursive formula for computing q_π with $\pi \equiv \pi^*$ must hold

$$q_{\pi^*}(s, a) = \sum_{\substack{s^+ \in \mathcal{S} \\ r \in \mathcal{R}}} p(s^+, r | s, a) \left[r + \gamma \sum_{a^+ \in \mathcal{A}} \pi^*(a^+ | s^+) q_{\pi^*}(s^+, a^+) \right]$$

Since v^* is optimal, the right hand side must be maximal among all possible policies

$$q_{\pi^*}(s, a) = \sum_{\substack{s^+ \in \mathcal{S} \\ r \in \mathcal{R}}} p(s^+, r | s, a) \left[r + \gamma \max_{\pi \in \mathcal{P}} \sum_{a^+ \in \mathcal{A}} \pi^*(a^+ | s^+) q_{\pi^*}(s^+, a^+) \right]$$

Stochastic Bellman Equation for State-Action Values

$$q^*(s, a) = \sum_{\substack{s^+ \in \mathcal{S} \\ r \in \mathcal{R}}} p(s^+, r | s, a) \left[r + \gamma \max_{a^+ \in \mathcal{A}} q^*(s^+, a^+) \right]$$

Bellman Equation

... and its various beautiful forms :)

Bellman Equation for State Values

$$v^*(s) = \max_{a \in \mathcal{A}} \{g(s, a) + \gamma v^*(f(s, a))\}$$

$$v^*(s) = \max_{a \in \mathcal{A}} \sum_{\substack{s^+ \in \mathcal{S} \\ r \in \mathcal{R}}} p(s^+, r | s, a) [r + \gamma v^*(s^+)]$$

Bellman Equation for State-Action Values

$$q^*(s, a) = h(s, a) + \gamma \max_{a^+ \in \mathcal{A}} q^*(f(s, a), a^+)$$

$$q^*(s, a) = \sum_{\substack{s^+ \in \mathcal{S} \\ r \in \mathcal{R}}} p(s^+, r | s, a) \left[r + \gamma \max_{a^+ \in \mathcal{A}} q^*(s^+, a^+) \right]$$

Due to the appearance of the \max operator, the Bellman equation is a set of coupled nonlinear equations. Direct, **closed-form solution is no longer possible**, not even in principle.

RL: Model-Based Tabular Methods

- 1 Introduction
- 2 Markov Decision Processes (MDPs)
- 3 Bellman Equation
- 4 RL: Model-Based Tabular Methods**
 - Value Iteration
 - Policy Iteration
- 5 RL: Tabular Monte Carlo Methods
- 6 RL: Temporal-Difference Methods
- 7 Approximate Reinforcement Learning

Introduction to Tabular Methods

Tabular methods of reinforcement learning work well for problems with **finite and relatively small state-action spaces**. In these cases, it is reasonable to represent state value function v as a vector \mathbf{v} , and the state-action function q as a matrix \mathbf{Q} .

RL: Model-Based Tabular Methods

Value Iteration

Introduction to Value Iteration methods

Value iteration algorithms exploit the fact that the Bellman Equation has a form of a **fixed-point formula**

$$\mathbf{v} = \mathcal{T}_v \{ \mathbf{v} \}$$

$$\mathbf{Q} = \mathcal{T}_Q \{ \mathbf{Q} \}$$

These methods start from an initial guess (\mathbf{v}^0 or \mathbf{Q}^0) and proceed to construct better-and-better approximations by transforming Bellman identity into an iterative procedure

$$\mathbf{v}^{k+1} \leftarrow \mathcal{T}_v \{ \mathbf{v}^k \}$$

$$\mathbf{Q}^{k+1} \leftarrow \mathcal{T}_Q \{ \mathbf{Q}^k \}$$

The procedure stops as soon as two successive approximations become close enough (i.e. if $\| \mathbf{v}^{k+1} - \mathbf{v}^k \| \leq \varepsilon$ or $\| \mathbf{Q}^{k+1} - \mathbf{Q}^k \| \leq \varepsilon$, for some sufficiently small $\varepsilon > 0$.)

Finally, the greedy policy with respect to the found value function is considered as a sufficiently good approximation to the optimal policy.

Value Iteration for the State Value Function

Algorithm 1 v -iteration algorithm

Require: $f, \gamma, \varepsilon > 0$

Require: \mathbf{v}^0

$\mathbf{v} \leftarrow \mathbf{v}^0$

repeat

for $s \in \mathcal{S}$ **do**

$$\mathbf{v}(s)^+ \leftarrow \max_{a \in \mathcal{A}} \sum_{\substack{s^+ \in \mathcal{S} \\ r \in \mathcal{R}}} p(s^+, r | s, a) [r + \gamma \mathbf{v}(s^+)]$$

end for

$$\delta \leftarrow \|\mathbf{v}^+ - \mathbf{v}\|_\infty$$

$$\mathbf{v} \leftarrow \mathbf{v}^+$$

until $\delta > \varepsilon$

return $\mathbf{v}, \pi = \pi_{\mathbf{v}}^{\text{greedy}}$

Value Iteration for the State-Action Value Function

Algorithm 2 Q -iteration algorithm

Require: $f, \gamma, \varepsilon > 0$

Require: Q^0

$Q \leftarrow Q^0$

repeat

for $s \in \mathcal{S}, a \in \mathcal{A}$ **do**

$$Q(s, a)^+ \leftarrow \sum_{r \in \mathcal{R}} p(s^+, r | s, a) [r + \gamma \max_{a^+ \in \mathcal{A}} Q(s^+, a^+)]$$

end for

$$\delta \leftarrow \|Q^+ - Q\|_\infty$$

$$Q \leftarrow Q^+$$

until $\delta > \varepsilon$

return $Q, \pi = \pi_Q^{\text{greedy}}$

RL: Model-Based Tabular Methods

Policy Iteration

Introduction to Policy Iteration methods

Policy iteration methods work directly on policies: they start from an initial policy π^0 and proceed to construct better-and-better policies.

In each iteration of the policy iteration procedure, a value function is constructed based on the given policy (this operation is relatively cheap). Then, the greedy policy with respect to that value function is used as the policy in the subsequent iteration.

$$\pi^0 \rightarrow \mathbf{Q}^0 \rightarrow \pi^1 \rightarrow \mathbf{Q}^1 \rightarrow \dots$$

The procedure stops when two identical consecutive policies are encountered.

Policy Iteration for the State Value Function

Algorithm 3 Policy iteration algorithm using \mathbf{v}

Require: f, γ

Require: π^0

$\pi \leftarrow \pi^0$

repeat

 Evaluate \mathbf{v}_π

$\pi^+ \leftarrow \pi_{\mathbf{v}_\pi}^{\text{greedy}}$

 TERMINATE $\leftarrow \pi^+ = \pi$

$\pi \leftarrow \pi^+$

until TERMINATE

return π

Value Iteration for the State-Action Value Function

Algorithm 4 Policy iteration algorithm using Q

Require: f, γ

Require: π^0

$\pi \leftarrow \pi^0$

repeat

Evaluate Q_π

$\pi^+ \leftarrow \pi_{Q_\pi}^{\text{greedy}}$

TERMINATE $\leftarrow \pi^+ = \pi$

$\pi \leftarrow \pi^+$

until **TERMINATE**

return π

RL: Tabular Monte Carlo Methods

- 1 Introduction
- 2 Markov Decision Processes (MDPs)
- 3 Bellman Equation
- 4 RL: Model-Based Tabular Methods
- 5 RL: Tabular Monte Carlo Methods**
- 6 RL: Temporal-Difference Methods
- 7 Approximate Reinforcement Learning

Introduction to Monte Carlo Methods

Monte Carlo approaches are characterized by the fact that the agent (controller) **learns** (near-) optimal decision policy through **off-line** experience: actual or simulated.

In general, the term Monte Carlo technique may be applied to any algorithm that uses randomization (or stochastic mean) to estimate certain quantities. We use the term Monte Carlo RL in the sense defined by [Sutton and Barto, 2018] – as an off-line approximate technique for solving RL problem.

If an agent uses simulated experience in a Monte Carlo learning scheme, then the model is actually needed. However, the required model is just a **simulator**, which is considerably easier to construct compared to the closed-form (stochastic) MDP required to formulate and solve the Bellman Equation.

Monte Carlo methods operate by averaging returns of a sequence of collected episodes (actual or simulated). **They are incremental in the sense that the active decision policy is updated on completion of one or more episodes.** During each episode, the decision policy remains fixed.

Monte Carlo Evaluation of State Values

Algorithm 5 Monte Carlo for obtaining v

```
function MONTECARLOEVALUATEV( simulator,  $\pi$ ,  $\gamma$ , EPOCHS_NO)
  For each  $s \in \mathcal{S}$  initialize  $\text{returns}(s) \leftarrow$  empty list
  for  $k \in 1..EPOCHS\_NO$  do
     $s^0 \leftarrow$  random
    episode  $\leftarrow$  simulator( $\pi, s^0$ )
    for  $s \in$  episode do
       $g \leftarrow$  gain obtained after (first) occurrence of  $s$ 
      Append  $g$  to  $\text{returns}(s)$ 
    end for
  end for
  return  $v(s) = \text{average returns}(s)$  for every  $s \in \mathcal{S}$ 
end function
```

Monte Carlo Evaluation of State-Action Values

Algorithm 6 Monte Carlo for obtaining Q

function MONTECARLOEVALUATEQ(simulator, π , γ , EPOCHS_NO)

For each $s \in \mathcal{S}$, $a \in \mathcal{A}$ initialize $\text{returns}(s, a) \leftarrow$ empty list

for $k \in 1..\text{EPOCHS_NO}$ **do**

$s^0 \leftarrow$ random

$a^0 \leftarrow$ random

episode \leftarrow simulator(π, s^0, a^0)

for $(s, a) \in$ episode **do**

$g \leftarrow$ gain obtained after (first) occurrence of s

Append g to $\text{returns}(s, a)$

end for

end for

return $Q(s, a) =$ average $\text{returns}(s, a)$ for every $s \in \mathcal{S}$, $a \in \mathcal{A}$

end function

Monte Carlo Control Algorithm

Algorithm 7 Monte Carlo RL

Require: π^0 , γ , simulator, EPOCHS_NO

$\pi \leftarrow \pi^0$

repeat

$Q \leftarrow \text{MonteCarloEvaluateQ}(\text{simulator}, \pi, \gamma, \text{EPOCHS_NO})$

$\pi^+ \leftarrow \pi_Q^{\text{greedy}}$

 TERMINATE $\leftarrow \pi = \pi^+$

$\pi \leftarrow \pi^+$

until TERMINATE

return π

Towards incremental RL algorithms

One of the main objections to the Monte Carlo RL (as presented here) is that we need many episodes in order to estimate **Q**-matrix.

Even worse, as soon as we change the decision policy we need to re-estimate the value function again! We loose all previous knowledge.

We would like to have an **incremental** learning algorithm, one that will *accumulate new pieces of information and integrate it into existing knowledge*. **Incremental Monte Carlo** and **Temporal-Difference Algorithms** (to be introduced next) are such procedures.

$$Q(s, a) \leftarrow Q(s, a) + \text{update based on the new evidence/data}$$

How to update the existing knowledge?

Surprisingly, a **filtering** problem :)

Suppose that for some combination of state s and action a we have an existing estimate $Q(s, a)$, and that we observe an episode in which following this same state-action combination we observe gain equal to g . **How should we update our Q -estimate?** What should be new estimate of the State-Action Value?

The answer, off course, depends strongly on how certain we are in the existing estimate and how “noisy” (or “uncertain”) is the observation, but **a fairly general procedure would be to take a weighted average of the existing estimate and the new data**, i.e. to choose $\alpha \in (0, 1)$ and

$$Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha g$$

update based on the new evidence/data

$$Q(s, a) \leftarrow Q(s, a) + \overbrace{\alpha(g - Q(s, a))}$$

Incremental Monte Carlo Control Algorithm

Algorithm 8 Incremental Monte Carlo RL

Require: \mathbf{Q}^0 , γ , simulator, α

$\mathbf{Q} \leftarrow \mathbf{Q}^0$

repeat

 episode \leftarrow simulator(π, s^0, a^0)

for $(s, a) \in$ episode **do**

$g \leftarrow$ gain obtained after (first) occurrence of s

$\mathbf{Q}(s, a) \leftarrow \mathbf{Q}(s, a) + \alpha(g - \mathbf{Q}(s, a))$

end for

 TERMINATE $\leftarrow \pi = \pi^+$

$\pi \leftarrow \pi^+$

until TERMINATE

return π

RL: Temporal-Difference Methods

- 1 Introduction
- 2 Markov Decision Processes (MDPs)
- 3 Bellman Equation
- 4 RL: Model-Based Tabular Methods
- 5 RL: Tabular Monte Carlo Methods
- 6 RL: Temporal-Difference Methods**
 - Q-Learning
 - SARSA Algorithm
- 7 Approximate Reinforcement Learning

Introduction to Temporal Difference Methods

Similar to the Incremental Monte Carlo procedure, Temporal Difference methods are incremental: they update existing value estimates based on newly acquired experience.

Unlike the Incremental Monte Carlo, **TD methods do not wait for the completion of the episode in order to make an update**, they do it on step-by-step basis (or sometimes after several steps). These methods are therefore also applicable to non-episodic tasks (such as control problems).

But, how does one update Q -values after just one decision step, when the gain possibly depends on many future steps? The idea is to **improve estimate based on other estimates!** This is called **bootstrapping!**

RL: Temporal-Difference Methods

Q-Learning

Q-Learning Explained

... as an **off-policy** temporal difference method

Q-Learning was proposed in 1992 by [Watkins and Dayan, 1992]. It is perhaps the most popular RL algorithm.

Q-Learning starts from the Bellman Equation for the optimal policy

$$q(s, a) = r + \gamma \max_{a^+ \in \mathcal{A}} q(s^+, a^+)$$

and turns it into incremental rule for updating Q -value estimate.

Q-Learning update rule

$$\mathbf{Q}(s, a) \leftarrow \mathbf{Q}(s, a) + \alpha \left[r + \gamma \max_{a^+ \in \mathcal{A}} \mathbf{Q}(s^+, a^+) - \mathbf{Q}(s, a) \right]$$

Q-Learning Algorithm

Algorithm 9 Q-Learning Algorithm

Require: \mathbf{Q}^0 , γ , `one_step_sim`, α

$\mathbf{Q} \leftarrow \mathbf{Q}^0$

repeat

$s \leftarrow \text{init}$

$a \leftarrow \text{init}$

repeat

$(s^+, r, \text{terminate}) \leftarrow \text{one_step_sim}(s, a)$

$\mathbf{Q}(s, a) \leftarrow \mathbf{Q}(s, a) + \alpha [r + \gamma \max_{a \in \mathcal{A}} \mathbf{Q}(s^+, a^+) - \mathbf{Q}(s, a)]$

$a \leftarrow \text{exploratory action based on } \mathbf{Q}$

until terminate

until stopping condition is satisfied

return \mathbf{Q}

RL: Temporal-Difference Methods

SARSA Algorithm

SARSA Explained 1/2

... or “Modified Connectionist Q-Learning” (MCQ-L)

SARSA algorithm was introduced in 1994 (MCQ-L) by [Rummery and Niranjan, 1994]. This original name is essentially never used, and the name **SARSA** is used throughout the literature (see e.g. [Sutton and Barto, 2018])

$$s^0 \xrightarrow{a^0, r^1} s^1 \xrightarrow{a^1, r^2} s^2 \xrightarrow{a^2, r^3} \dots$$

$$\mathbf{S} \rightarrow \mathbf{A} \rightarrow \mathbf{R}, \mathbf{S}^+ \rightarrow \mathbf{A}^+$$

The main question we ask in SARSA is: *If I observe the above sequence, how should I update my estimate of the Q-function?*

SARSA Explained 2/2

... as an **on-policy** temporal difference method

We saw that, upon receiving a new information g , an incremental update procedure has the generic form

$$\mathbf{Q}(s, a) \leftarrow \mathbf{Q}(s, a) + \alpha [g - \mathbf{Q}(s, a)]$$

We know that for any given policy (and therefore also for the policy currently in use) the following Bellman Equation holds

$$q(s, a) = r + \gamma q(s^+, a^+)$$

SARSA update rule

$$\mathbf{Q}(s, a) \leftarrow \mathbf{Q}(s, a) + \alpha [r + \gamma \mathbf{Q}(s^+, a^+) - \mathbf{Q}(s, a)]$$

Since the Q -function is evaluated for the policy that is actually being used, SARSA is an on-policy algorithm.

SARSA Algorithm

Algorithm 10 SARSA Algorithm

Require: \mathbf{Q}^0 , γ , `one_step_sim`, α

$\mathbf{Q} \leftarrow \mathbf{Q}^0$

repeat

$s \leftarrow \text{init}$

$a \leftarrow \text{init}$

repeat

$(s^+, r, \text{terminate}) \leftarrow \text{one_step_sim}(s, a)$

$a^+ \leftarrow \text{exploratory action based on } \mathbf{Q}$

$\mathbf{Q}(s, a) \leftarrow \mathbf{Q}(s, a) + \alpha [r + \gamma \mathbf{Q}(s^+, a^+) - \mathbf{Q}(s, a)]$

$a \leftarrow a^+$

until terminate

until stopping condition is satisfied

return \mathbf{Q}

Approximate Reinforcement Learning

- 1 Introduction
- 2 Markov Decision Processes (MDPs)
- 3 Bellman Equation
- 4 RL: Model-Based Tabular Methods
- 5 RL: Tabular Monte Carlo Methods
- 6 RL: Temporal-Difference Methods
- 7 Approximate Reinforcement Learning**

Introduction to Approximate RL 1/2

Tabular methods (discussed up to this point) require that Q -functions and policies are **exactly represented**.

This is possible, but only when the state-action space has tractably many entries.

In many practically interesting examples – an in all typical control problems – state-action space is continuous, and none of the described algorithms can be applied directly.

It is therefore more convenient to use parameterized Q -functions – $Q(s, a, \theta)$ – and parameterized policies – $\pi(x, \pi)$.

In this way, although the action-space is potentially intractably big, the parametric space remains an Euclidean space of (relatively) small dimension. It is well known that search (optimization) can be performed in an effective manner in such spaces.

Introduction to Approximate RL 2/2

Approximate Q-iterations

The main idea in all approximate methods is to parameterize the Action-Value Function so as to minimize (with respect to θ) the estimation cost

$$J = \frac{1}{2} \sum_{(s,a) \in \text{experience}} (g(s,a) - Q(s,a,\theta))^2$$

where $g(s,a)$ is an assessment (an estimate) of the gain obtained following the encounter of state-action point (s,a) .

Different approximate methods differ w.r.t. **when is the policy updated** (after multiple episodes, one episode, several actions, each action) and w.r.t. **how is the assessment of the gain obtained**.

Introduction to On-Line Approximate RL

In On-Line Approximate RL, the cost function is evaluated often using **just the latest sample**

$$J = \frac{1}{2}(g(s, a) - Q(s, a, \theta))^2$$

and it is optimized using the **Gradient Descent Rule** (a.k.a. the MIT Rule): update parameters in the direction of negative gradient (when minimizing).

$$\theta_{k+1} = \theta_k - \alpha \nabla_{\theta} J = \theta_k + \alpha (g(s, a) - Q(s, a, \theta)) \nabla_{\theta} Q$$

Approximate Q-Learning

Algorithm 11 Approximate Q-Learning Algorithm

Require: θ^0 , γ , one_step_sim, α

$\theta \leftarrow \theta^0$

repeat

$s \leftarrow \text{init}$

$a \leftarrow \text{init}$

repeat

$(s^+, r, \text{terminate}) \leftarrow \text{one_step_sim}(s, a)$

$\theta \leftarrow \theta + \alpha [r + \gamma \max_{a \in \mathcal{A}} Q(s^+, a^+, \theta) - Q(s, a, \theta)] \nabla_{\theta} Q(s, a, \theta)$

$a \leftarrow \text{exploratory action based on } Q$

until terminate

until stopping condition is satisfied

return θ

Approximate SARSA

Algorithm 12 Approximate SARSA Algorithm

Require: θ^0 , γ , one_step_sim, α

$\theta \leftarrow \theta^0$

repeat

$s \leftarrow \text{init}$

$a \leftarrow \text{init}$

repeat

$(s^+, r, \text{terminate}) \leftarrow \text{one_step_sim}(s, a)$

$a \leftarrow \text{exploratory action based on } Q$

$\theta \leftarrow \theta + \alpha [r + \gamma Q(s^+, a^+, \theta) - Q(s, a, \theta)] \nabla_{\theta} Q(s, a, \theta)$

$a \leftarrow a^+$

until terminate

until stopping condition is satisfied

return θ

References I



Arulkumaran, K., Deisenroth, M. P., Brundage, M., and Bharath, A. A. (2017).

Deep reinforcement learning: A brief survey.
IEEE Signal Processing Magazine, 34(6):26–38.



Li, Y. (2018).

Deep reinforcement learning: An overview.



Nian, R., Liu, J., and Huang, B. (2020).

A review on reinforcement learning: Introduction and applications in industrial process control.



Computers & Chemical Engineering, 139:106886.



Rummery, G. A. and Niranjan, M. (1994).

On-line Q-learning using connectionist systems.

References II

-  Sutton, R. S. and Barto, A. G. (2018).
Reinforcement Learning: An Introduction.
The MIT Press, second edition.
-  Watkins, C. J. C. H. and Dayan, P. (1992).
Q-learning.
Machine Learning, 8(3):279–292.