Reinforcement Learning, Lecture 1: Introduction [1]

Lecturer: Jim Mainprice, PhD

Interim Professor & Research Group Leader
Machine Learning and Robotics Lab

University of Stuttgart
Germany

April 22, 2020

[1] Many slides adapted from R. Sutton's course, David Silver's course, as well as previous RL courses given at U. of Stuttgart by D. Hennes, M. Toussaint, H. Ngo, and V. Ngo.

Outline

Contact

- **Contact**
  - jim.mainprice@ipvs.uni-stuttgart.de
  - philipp.kratzer@ipvs.uni-stuttgart.de
- **Website (Announcements!!!)**
  https://ipvs.informatik.uni-stuttgart.de/mlr/reinforcement-learning-ss-20/
- **Ilias course (Material)**
  https://ilias3.uni-stuttgart.de/goto_Uni_Stuttgart_crs_1878872.html

Dates

Lecture

- Tuesdays, 15:45-17:15
- First lecture on 21.04 - live on YouTube:
  `https://www.youtube.com/channel/UCVe0W0-tDpHaTuxLB8DmV2g`
- Lectures will be available on Ilias's course page

Tutorials

- Wednesday, 14:00 - 15:30
- First tutorial on 29.04
- As long as the university is closed, tutorials will be held online via Webex
- Depending on the number of students we might have a second tutorial slot

Exercises Requirements

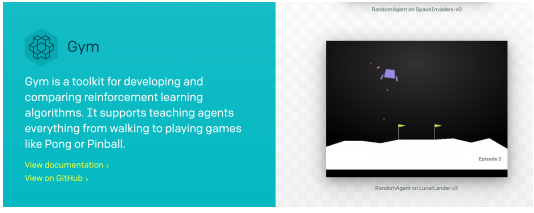
- Weekly exercise sheets (usually 10 sheets total)
- **Written and programming** exercises
- Each exercise sheet will have $\sim$ 10 credits
- 50% of the total credits are required

Exercises Submission

- Solutions **uploaded to Ilias one day before** the tutorial session
- Group submissions of up to 3 students are allowed
- Code should run out of the box, otherwise it will not be graded
- Assignments will only be checked to ensure an attempt at solving has been made (no detailed individual feedback)
- The submitted solutions will be discussed in the tutorial sessions
- Submissions as a zip archive with
  - written as a **single pdf** file
  - progra. as a **single python** source file

Programming

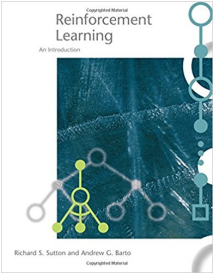
- Programming will be done in python3
- Sourcecode for the exercises will be published via github `https://github.com/humans-to-robots-motion/rl-course`
- We will use **openai gym** (`https://gym.openai.com/`)
- Only the dependencies of python libraries (e.g. numpy, matplotlib, scipy, openai gym) should be used
- Submit clean and commented code!

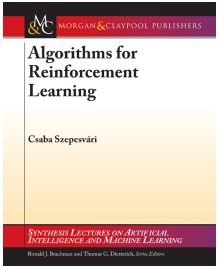


Announcements

- This week (tomorrow): no tutorials!
- First exercise sheet available in Ilias (due 28.04.)
- Next week, lecture online same time!

Literature

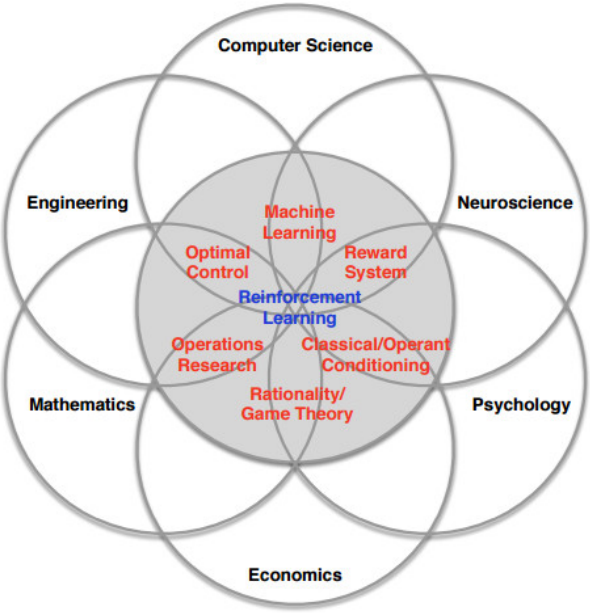


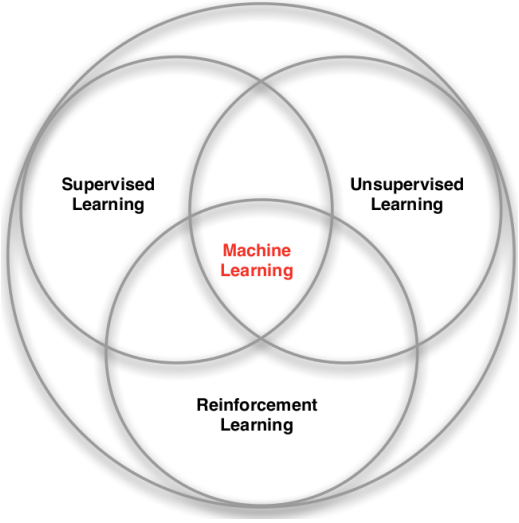


- *Reinforcement Learning: An Introduction* (2nd ed.) by Richard Sutton and Andrew Barto
  `http://incompleteideas.net/book/bookdraft2017nov5.pdf`
- *Algorithms for Reinforcement Learning* by Cesba Szepesvari
  `https://sites.ualberta.ca/~szepesva/papers/RLAlgsInMDPs.pdf`

What is reinforcement learning?

- General-purpose framework for *decision-making*
- Autonomous agent that *interacts* with its environments
  Learning through interaction
- Improving over time through *trial & error*
- **Agent** with the capacity to **act**
- Each **action** influences the future state
- Success is measured by a scalar **reward** signal
- Goal: **select actions to maximise future reward**


Computer Science
Engineering
Neuroscience
Machine Learning
Optimal Control
Reward System
Reinforcement Learning
Operations Research
Classical/Operant Conditioning
Rationality/ Game Theory
Mathematics
Psychology
Economics


Supervised Learning
Unsupervised Learning
Machine Learning
Reinforcement Learning

The term "reinforcement learning"

- The term *"Reinforcement learning"* may refer to
  - a type of **problem**
  - the class of **solution methods** that work well on RL problems
  - the **research field** that studies RL problems and RL methods

It is important not to confuse the first two!

Characteristics of reinforcement learning

What makes reinforcement learning differnet from other machine learning paradigms?

- There is no supervisor, only a *reward* signal
- Feedback is (often) delayed, non instantaneous
- Time really matters (sequential, non i.i.d data)
- Agent's actions affect the subsequent data it receives

Examples of reinforcement learning

- **Fly stunt manoeuvres with a RC helicopter**
- **Learn to flip pancakes**
- Play boardgames (e.g., Backgammon, Go, Chess)
- Manage investment portfolios
- **Play Atari games at super human level**
- **Learning to walk**

Rewards

- A *reward* R_t is a scalar feedback signal
- *Only* feedback provided to the agent,
  no explicit teacher
- May indicate how well agent's last action was
- The agent's job is to maximize its expected cumulative reward
  over some (possibly) infinite horizon
- **Examples:**
  - winning or loosing a game (e.g., Backgammon, Go, . . .)
  - increaseing/decreasing score (e.g., video games)
  - earning/loosing money (e.g., portfolio management)
  - following a desired trajectory vs. crashing (e.g., robotic control)
  - . . .

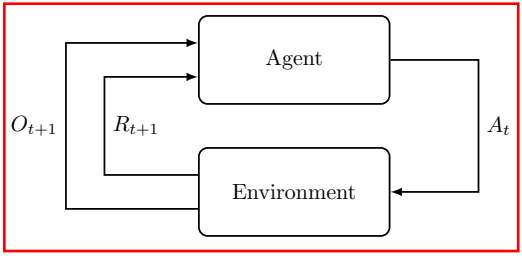
Can we describe *all* goals by the maximization of expected cumulative reward?

Sequential decision making

- Goal: *select actions to maximize total future reward*
- Actions may have long term consequences
- Reward may be delayed
- E.g., it may be better to sacrifice immediate reward to gain more long-term reward
- Examples:
  - A financial investment (may take months to mature)
  - Refueling a helicopter (might prevent a crash in several hours)
  - Blocking opponent moves (might help winning chances many moves from now)

Interaction loop

- At each step t, the agent
  - receives observation O_t
  - receives scalar reward R_t
  - executes action A_t
- The environment
  - receives action A_t
  - emits observation O_{t+1}
  - emits scalar reward R_{t+1}
- t increments at environment step



History and state

- The **history** is the sequence of observations, actions, rewards

$$H_t = O_1, R_1, A_1, \ldots, A_{t-1}, O_t, R_t$$

- i.e. all observable variables up to time t
- i.e. the sensorimotor stream of a robot or embodied agent
- What happens next depends on the history:
  - The agent selects actions
  - The environment selects observations/rewards
- **State** is the sufficient information used to determine what happens next
- Formally, the state is a function of the history:

$$S_t = f(H_t)$$

Information state

An **information state** (a.k.a. *Markov* state) contains all useful information from the history.

A state S_t is Markov if and only if

$$\Pr\{S_{t+1} \mid S_t\} = \Pr\{S_{t+1} \mid S_1, \ldots, S_t\}$$

The future is independent of the past given the present

$$H_{1:t} \to S_t \to H_{t+1:\infty}$$

- Once the state is known, the history may be thrown away, i.e. the state is a sufficient statistic of the future
- The history H_t is Markov

Fully and partially observable environments

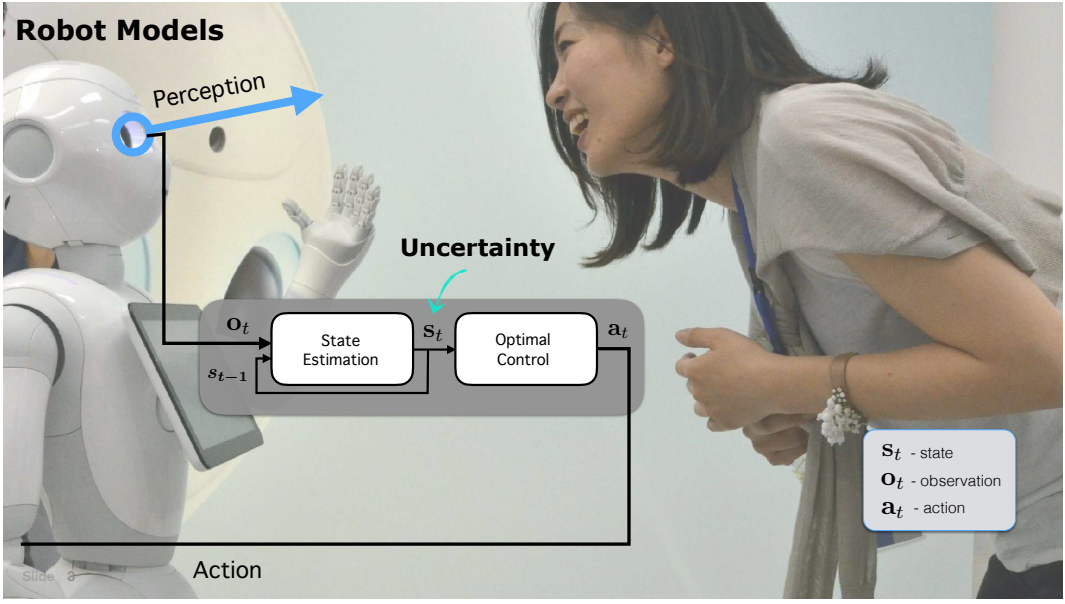
- If the agent directly observes the Markov state,
  we call the interaction model a **Markov Decision Process** (MDP)
- If the agent indirectly observes the environment state,
  we call it a **Partially Observable Markov Decision Process** (POMDP)
- Many (if not all) real world examples are POMDPs
- Examples:
  - a robot with camera vision isn't told its absolute location
  - a trading agent only observes current prices
  - a poker playing agent only observes public cards

Building blocks of RL agents

- **Policy:** agent's behavior
- **Value function:** how good is (*a given action in*) a given state?
- **Model:** agent's representation of the environment

Policy

- Defines the agent's behavior
- Maps from state to action
- **Deterministic policy:** $a = \pi(s)$
- **Stochastic policy:** $\pi(a \mid s) = \Pr\{A_t = a \mid S_t = s\}$


Robot Models
Perception
Uncertainty
$\mathbf{o}_t$
s_{t-1}
State Estimation
$\mathbf{s}_t$
Optimal Control
$\mathbf{a}_t$
Action
$\mathbf{s}_t$ - state
$\mathbf{o}_t$ - observation
$\mathbf{a}_t$ - action
Slide 3

Self-Driving Car

Actuators

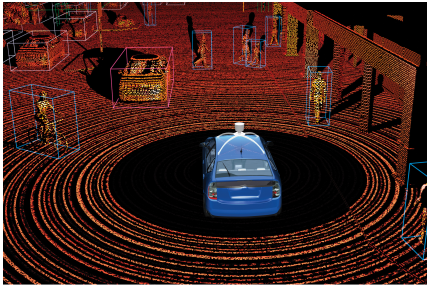
1. Steering $\dot{\theta}_t$
2. Accelerator $\ddot{x}_t$

Sensors

1. Wheel encoders
2. Velodyne LiDAR™ (Laser radar)
3. Cameras, IMUs, GPS, ...

State

1. Angle of the wheels θ_t
2. Speed $\dot{x}_t$
3. Location of self in the world x_t
4. Location of pedestrians and other cars



Google™ car

Robot Control Policy

To act a robot needs a map from state $\mathbf{s}_t$ to an action $\mathbf{a}_t$

- Policy $\pi : \mathbf{s}_t \mapsto \mathbf{a}_t$

When a robot walks

- Path $\sim$1 Hz
- Balance $\sim$1000 Hz

Action process with different time horizon are used

- Planning (global)
- Control (local)

$$\pi(\mathbf{s}_t) = \mathbf{a}_t$$

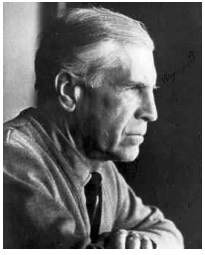
Asimo^{TM} robot

Notation

$\mathbf{s}_t$ – state
$\mathbf{a}_t$ – action

$\mathbf{x}_t$ – state
$\mathbf{u}_t$ – action управление

Richard Bellman



Lev Pontryagin

[source Levine 2018]

Value function

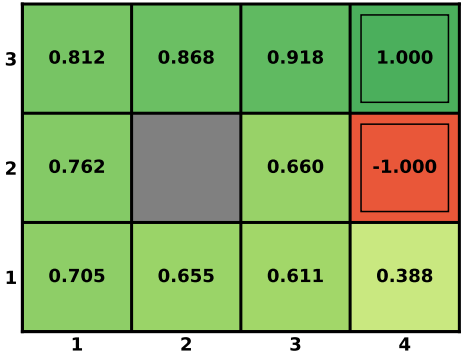
- Value function is a *prediction of future reward*

Value Function = Cost-to-go
= Cost (for infinite horizon problems)

$$v_\pi(s) = \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots \mid S_t = s]$$

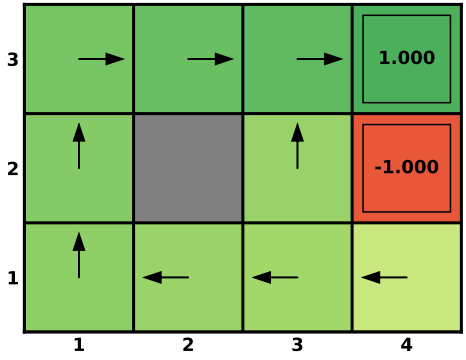
- Used to evaluate the goodness/badness of states
- And thus to select between actions

Example: grid world

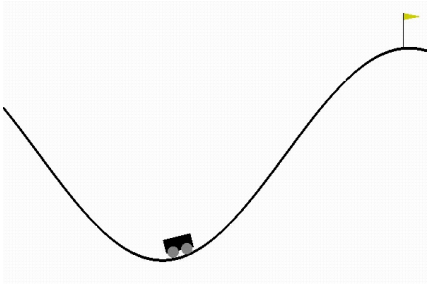


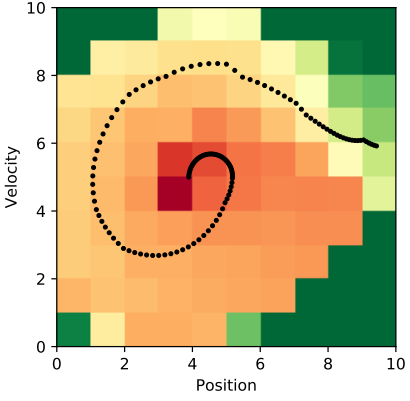
- Rewards: 0, $+1$, -1
- Actions: N, E, S, W
- States: agent's location

Example: grid world



Example: mountain car





Model

- A **model** predicts what the environment will do next
  - the next state s'
  - the next (immediate) reward r

$$p(s', r \mid s, a) = \Pr\left\{S_{t+1} = s', R_{t+1} = r \mid S_t = s, A_t = a\right\}$$

Many flavours of reinforcement learning

model-based $S_t, R_t, A_t, S_{t+1} \ldots \rightarrow p(s' \mid s, a), r(s, a, s') \rightarrow v(s) \rightarrow \pi(s)$

model-free

value-based $S_t, R_t, A_t, S_{t+1} \ldots \rightarrow q(s, a) \rightarrow \pi(s)$

policy-based $S_t, R_t, A_t, S_{t+1} \ldots \rightarrow \pi(s)$

actor-critic $S_t, R_t, A_t, S_{t+1} \ldots \rightarrow q(s, a), \pi(s)$

imitation learn. $\left\{(S_{1:T}, A_{1:T}, R_{1:T})^i\right\}_{i=1}^n \rightarrow \pi(s)$

learning dynamic programming

Learning or planning?

- **Reinforcement Learning:**
  - the environment is (initially) unknown
  - the agent interacts with the environment
  - the agent improves its policy
- **Planning:**
  - a model of the environment is known
  - the agent performs computations with its model (without any actual interaction)
  - the agent improves its policy
- There are two types of problems
  - Prediction: Determine a value function
  - Control: Determine an optimal policy

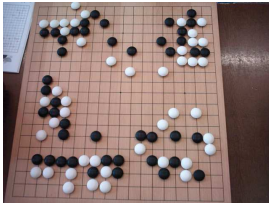
Exploration vs. exploitation

- Reinforcement learning is *trial & error* learning
- The agent should discover a good policy
  - from its experiences of the environment
  - without losing too much reward along the way
- **Exploration** finds more information about the environment
- **Exploitation** exploits known information to maximise reward
- Examples:
  - **Dining:** go to your favorite restaurant vs. try something new
  - **Advertisment:** place a new advert vs. the most relevant
  - **Mars rover:** sample a new location vs. sample best so far
  - **Game playing:** play a new move vs. the move that worked in the past

Success of reinforcement learning

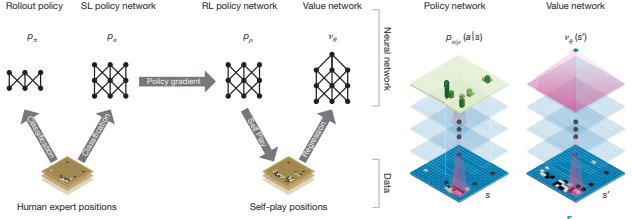
- **Games:**
  - Backgammon (Tesauro, 1994)
  - Deep RL playing Atari (2014)
  - AlphaGo (2016)
- **Operations research:**
  - Inventory Management (Van Roy, Bertsekas, Lee, & Tsitsiklis, 1996)
  - Dynamic Channel Allocation (e.g. Singh & Bertsekas, 1997)
  - Investment portfolio managment
  - Online advertisements
- **Robotics:**
  - Helicopter control (Ng 2003, Abbeel & Ng 2006)
  - Bi-pedal walking
  - Grasping

Success AlphaGo



[source wikipedia]

- AlphaGo [Silver 16]
  - Train a policy $p_\sigma(\mathbf{a}|\mathbf{s})$ network by SL/BC
  - Improve the policy by RL and self-play
  - Train a value network $v_\theta(\mathbf{s}')$
- $p_\sigma(\mathbf{a}|\mathbf{s})$ is a 13-layer DNN with alternating convolutions and ReLUs, with output soft-max layer (probabilities over $\mathbf{a}$)



[source Silver 16]

Tabular solution methods

- *State* and *action spaces* are small enough to be represented as arrays, or *tables*
- Methods can often find *exact* solutions, i.e., **optimal value function** or **optimal policy**
- Later: function approximation & policy search

Multi-armed bandits: RL problems with only one state

- *Nonassociative* setting: only the action space is relevant
- Finite bandit (k-armed bandit) $\rightarrow$ finite action space
- Training information:
  - *evaluate* actions taken
  - **not** *instructs* of "correct" action

k-armed bandit

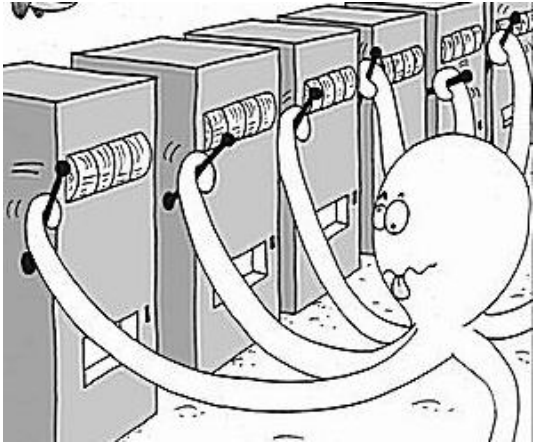


image credits: Microsoft Research

- There are k actions (machines)
- Each machine returns a reward from a stationary probability distribution
- Objective is to maximize the expected total reward, collected over the first T trials

Value

V(x) is a function only of the states
Q(x,a) is a function of action and states

- Each action a has an expected or mean reward, the **value**:

$$q_*(a) = \mathbb{E}[R_t \mid A_t = a]$$

- If you would know the true value q_* the next choice would be trivial
- Estimate of the action-value at time step t: $Q_t(a)$

Exploration vs. exploitation

- At each time step t there is (at least) one action that maximizes Q_t, called the *greedy* action:

$$A_t = \arg\max_a Q_t(a)$$

- Exploitation: selecting *greedy* action
- Exploration: selecting *nongreedy* action
  - improving estimate of the nongreedy action's value
  - reward lower in the short run
  - potentially much higher in the long run
- What is better? What does it depend on?
  - current action-value estimates
  - uncertainties
  - number of remaining steps

Estimating action-values

- *Sample average* method:

$$Q_t(a) = \frac{\sum_{i=1}^{t-1} R_i \cdot \mathbb{1}_{A_i=a}}{\sum_{i=1}^{t-1} \mathbb{1}_{A_i=a}}$$

sum of rewards that that action gave us

number of times we picked that action

- Next action is greedy:

$$A_t = \arg\max_a Q_t(a)$$

- $\mathbb{1}$ is the indicator function,
  i.e., 1 if predicate is true, else 0

- If denominator is zero, set $Q_t(a) = 0$

ϵ-greedy action selection

- Simple idea to force continued exploration
- With probability $1 - \epsilon$ take the *greedy* action
- With probability ϵ take a random action
- All actions are chosen with non-zero probability

Course Outline

- Exact Methods
  1. Introduction
  2. Markov Decision Processes
  3. Dynamic Programming
  4. Monte Carlo Methods
  5. Temporal Differences Learning
- Approximate Methods
  1. Integrated Planning & Learning
  2. Function Approximation
  3. Policy Gradient
  4. Hierarchical Reinforcement Learning
  5. Inverse Reinforcement Learning

Announcements

- This week (tomorrow): no tutorials!
- First exercise sheet available in Ilias (due 28.04.)
- Next week, lecture online same time!