Reinforcement Learning
Lecture 1: Introduction [1]

Lecturer: Prof. Dr. Mathias Niepert

Institute for Aritifical Intelligence
Machine Learning and Simulation Lab

University of Stuttgart
Germany

imprs-is

April 11, 2024

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

Outline

Admin

Contact

- **Contact**
  - Main teaching assistant: Vinh Tong
  - E-mail: `vinh.tong@ipvs.uni-stuttgart.de`
  - Please only use e-mail for questions not appropriate for the forum – use the Ilias forum for all questions about lecture material, organization, etc.
- **Ilias course (General Information + Material + Forum)**
  `https://ilias3.uni-stuttgart.de/goto_Uni_Stuttgart_crs_3649799.html`
  - Students are encouraged to use the forum and discuss lecture content with each other

Lecture Dates

Lecture

- Thursdays, 14:00 - 15:30, room V38.03
- Lectures will be recorded and available on Ilias's course page
- Due to holidays no lecture on
  - Christi Himmelfahrt: 9. May 2023
  - Pfingstferien: 23. May 2023
  - Fronleichnam: 30. May 2023
  - No tutorial session if no lecture and, therefore, no homework assignment!

Tutorial Session Dates

Tutorials

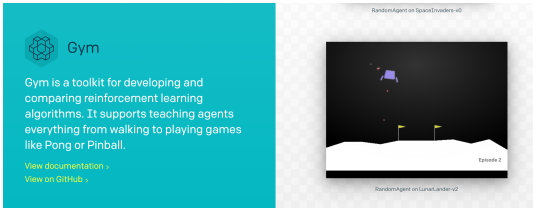
- Group 1: Monday 9:45 - 11:15, U32.139
- Group 2: Monday 11:30 - 13:00, U32.139
- Group 3: Thursday 9:45 - 11:15, U32.139
- Group 5: Wednesday 15:45 - 17:15, U32.139
- Group 6: Thursday 9:45 - 11:15, **U32.138**
- Group 8: Wednesday 14:00 - 15:30, U32.139
- First tutorials on Monday 22.04.
- No tutorials on holidays - please switch to other groups on these days
- Tutorial sessions are not recorded

Exercises

- Weekly exercise sheets (usually 9 sheets total)
- published one day after the lecture, due 9 days later, the Sunday before Monday tutorials – upload on Ilias!
- Each exercise sheet will have $\sim$ 10 credits
- **Written and programming** exercises
- Students join their exercise group to discuss the exercises
- Group solutions of **three** students are required
- Tutors provide no written solutions
- Exercises will be (partially) graded

Programming

- Programming will be done in python3
- Sourcecode for the exercises will be published via Ilias
- 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
- Write clean and commented code!



Exam Admission Criteria

1. Exercises are graded by tutors
2. Students need to obtain **50% of the total credits** to get the "Schein"
3. During tutorials, you will be called randomly to demonstrate your solution
4. If you are not present the first time, you get one "strike" but don't have to do anything
5. If you get a second strike you need a proper excuse
6. No need to send notes to tutors before the tutorials!
7. **Scheine from previous semesters are valid!**

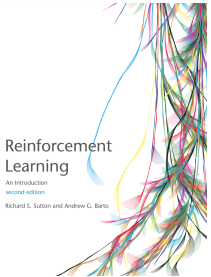
Grading

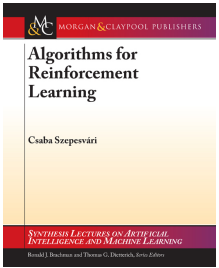
- The final grade is determined by the final exam

Announcements

- This week and next week: no tutorials!
- First exercise sheet will be uploaded to Ilias tomorrow (due 21.04.)
- Next week, lecture at same time!

Literature





- *Reinforcement Learning: An Introduction* (2nd ed.)
  by Richard Sutton and Andrew Barto
  `http://incompleteideas.net/book/RLbook2020.pdf`
- *Algorithms for Reinforcement Learning* by Cesba Szepesvari
  `https://sites.ualberta.ca/~szepesva/papers/RLAlgsInMDPs.pdf`

Reinforcement Learning Problem

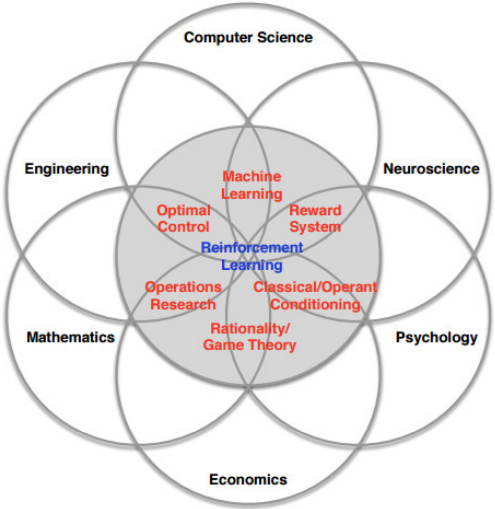
What is reinforcement learning?

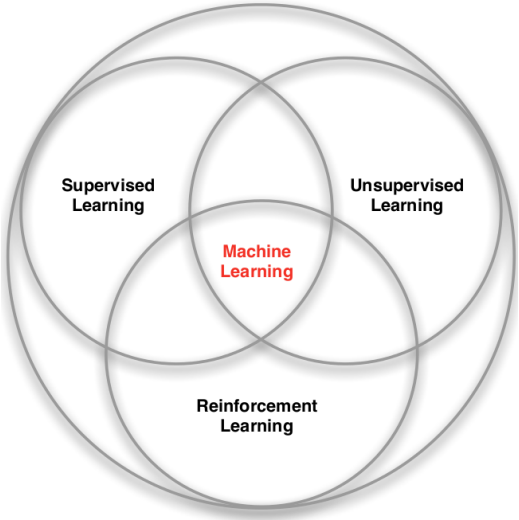
- General-purpose framework for *decision-making*
- Autonomous agent that *interacts* with its environments
  Learning through interaction
- Improving over time through *trial & error*
- **Agent** with the capacity to **act**
- Each **action** influences the future state
- Success is measured by a scalar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

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!


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

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**
- Fine-tune a large language model such as ChatGPT

Rewards

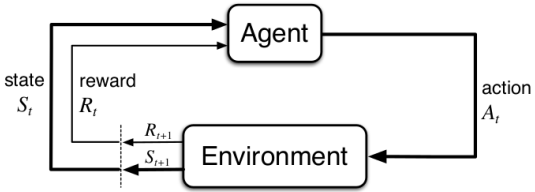
- A *reward* R_t is a scalar feedback signal
- *Only* feedback provided to the agent, no explicit teacher
- May indicate how well agent's last action was
- The agent's job is to maximize its expected cumulative reward over some (possibly) infinite horizon
- **Examples:**
  - winning or loosing a game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

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 representation of state S_t
  - executes action A_t
  - receives scalar reward R_{t+1}
- The environment
  - receives action A_t
  - emits scalar reward R_{t+1}
  - emits observation S_{t+1}
- t increments at environment step



History and state

- The **history** is the sequence (trajectory) of states (observations), actions, rewards

$$H_t = S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{t-1}, A_{t-1}, R_t$$

- i.e. all observable variables (the state representation) up to time t
- e.g. 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:

$$S_t = f(H_t)$$

The Markov Property

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

Reinforcement Learning Agent

Building blocks of RL agents

- **Policy:** agent's behavior
- **Value function:** the expected reward ("goodness") of state under a given policy
- **Action-value function:** the expected reward ("goodness") of an action in a state under a given policy
- **Model:** agent's representation of the environment

Policy

- Defines the agent's behavior
- (Probabilistic) map from states to actions
- **Deterministic policy:** $a = \pi(s)$
- **Stochastic policy:** $\pi(a \mid s) = \Pr\{A_t = a \mid S_t = s\}$

Self-Driving Car

Actuators

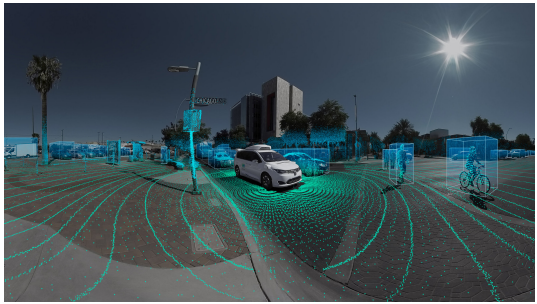
1. Steering $\dot{\theta}_t$
2. Accelerator $\ddot{x}_t$

Sensors

1. Wheel encoders
2. Velodyne LiDAR^{TM} (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



Waymo^{TM} 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

Value function

- Value function is an *expectation of future reward*

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

- it is a function of a state s under policy π
- Used to evaluate the goodness/badness of states
- And thus to select between actions

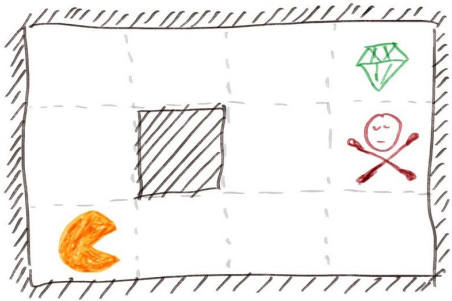
Action-Value function

- the action-value function defines the value (expected future reward) of taking action a in state s under policy π (and then following π)

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

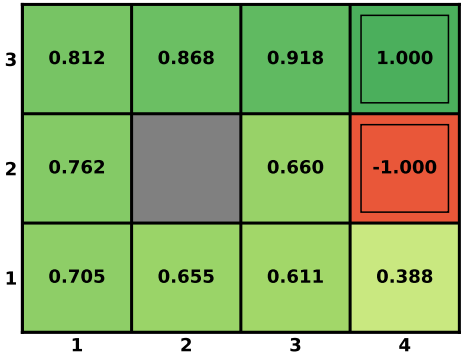
- Used to evaluate the goodness/badness of an action taken in a particular state under the policy
- Can also be used to select between actions

Example: grid world



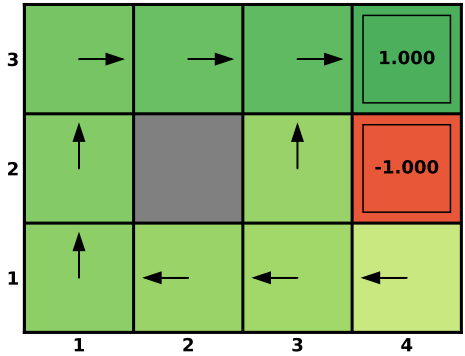
- Rewards: -0.01, $+1$, -1
- Actions: N, E, S, W
- States: agent's location

Example: grid world

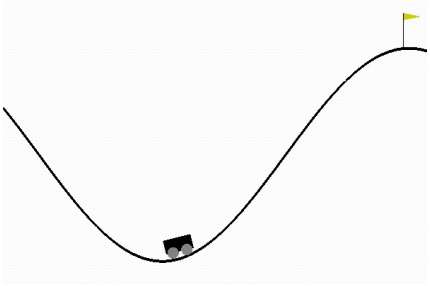


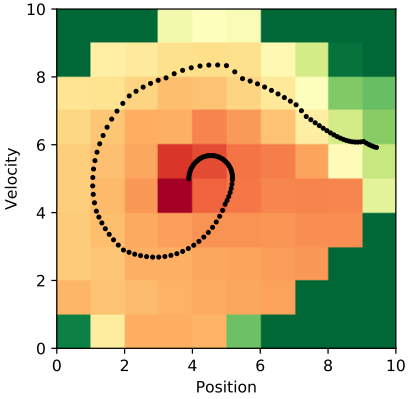
- Rewards: -0.01, $+1$, -1
- Actions: N, E, S, W
- States: agent's location

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

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

- "True" model is also called the transition function of the environment
- Model-based RL tries to identify an (approximate) model of the environment

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

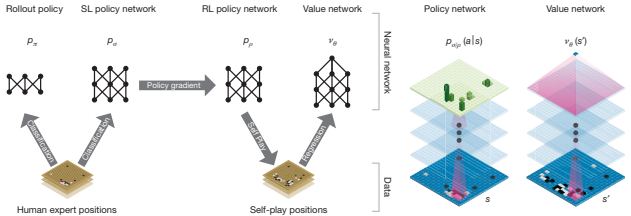
Success of reinforcement learning

- **Games:**
  - Backgammon (Tesauro, 1994)
  - Deep RL playing Atari (2014)
  - AlphaGo (2016)
- **Operations research:**
  - Investment portfolio managment
  - Online advertisements
- **Robotics:**
  - Helicopter control (Ng 2003, Abbeel & Ng 2006)
  - Bi-pedal walking
  - Grasping
- **Generative AI:**
  - RLHF to fine-tune LLMs

Success AlphaGo

[source wikipedia]

- AlphaGo [Silver 16]
  - Train a policy $p_\sigma(\mathbf{a}|\mathbf{s})$ network
  - 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 $\mathbf{a}$)

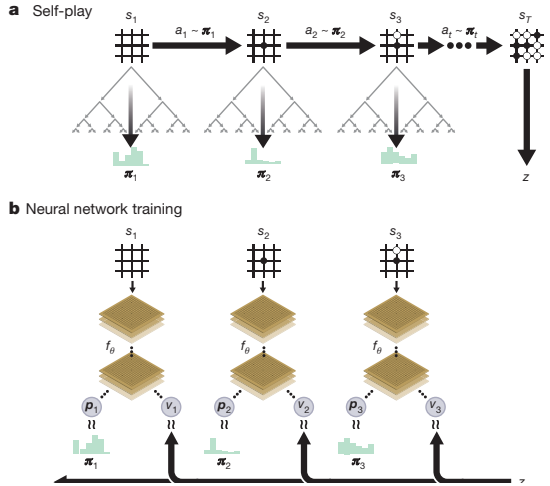


[source Nature Silver 16]

Success AlphaGo Zero

AlphaGo Zero [Silver 17]

- Start *tabula rasa*
- Policy improvement: MCTS
- Policy evaluation: Self Play
  Learn From Win/Loss

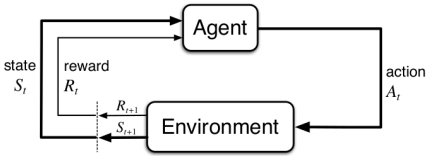


[source Nature Silver 17]

Bandits

What is reinforcement learning?

- Agent observes the **state**
- Agent chooses an **action**
- Agent gets a **reward**
- Aim is to learn a **policy**: what action to choose
  in a given state in order to get maximum *long-term* reward
- Problems are reduced to three signals being passed back and forth



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**

k-armed bandit

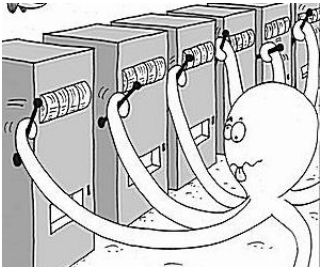


image credits: Microsoft Research

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

Value

- 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 action value q_* **for every** a, 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
- **Greedy policy**:

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

- 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

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

Estimating action-values

- *Sample average* method:

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

- If $n(a) = 0$, set $Q_t(a) = 0$
- As $n(a) \to \infty, Q_t(a) = q_*(a)$
- We sometimes write $\hat{Q}(a)$ for the estimate
- What is the difference between $q_*(a)$ and $\max_a Q_t(a)$?
  - $q_*(a)$ is the true value of a
  - $\max_a Q_t(a)$ is the greedy action value at time t

Announcements

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 and next week: no tutorials!
- First exercise sheet will be uploaded to Ilias tomorrow (due 21.04.)
- Next week, lecture at same time!