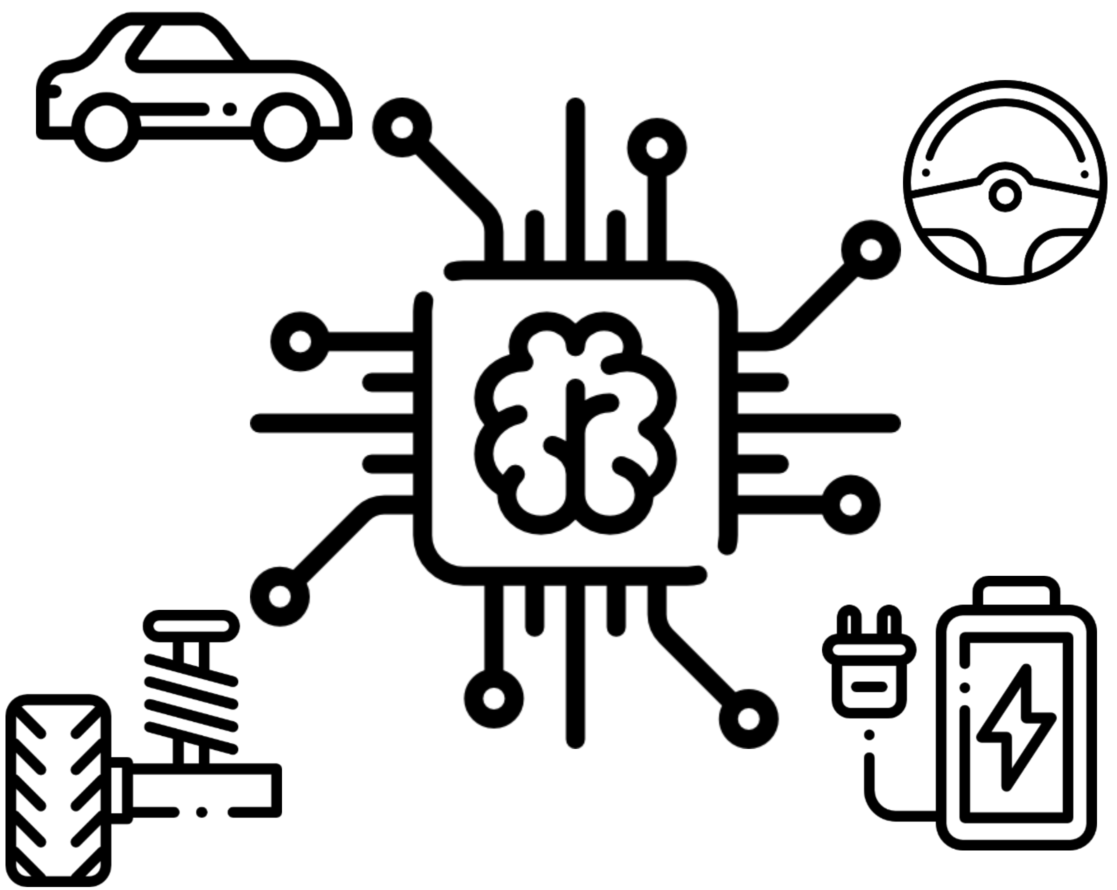
Lehrstuhl für Fahrzeugtechnik
Fakultät für Maschinenwesen
Technische Universität München

Artificial Intelligence in Automotive Technology

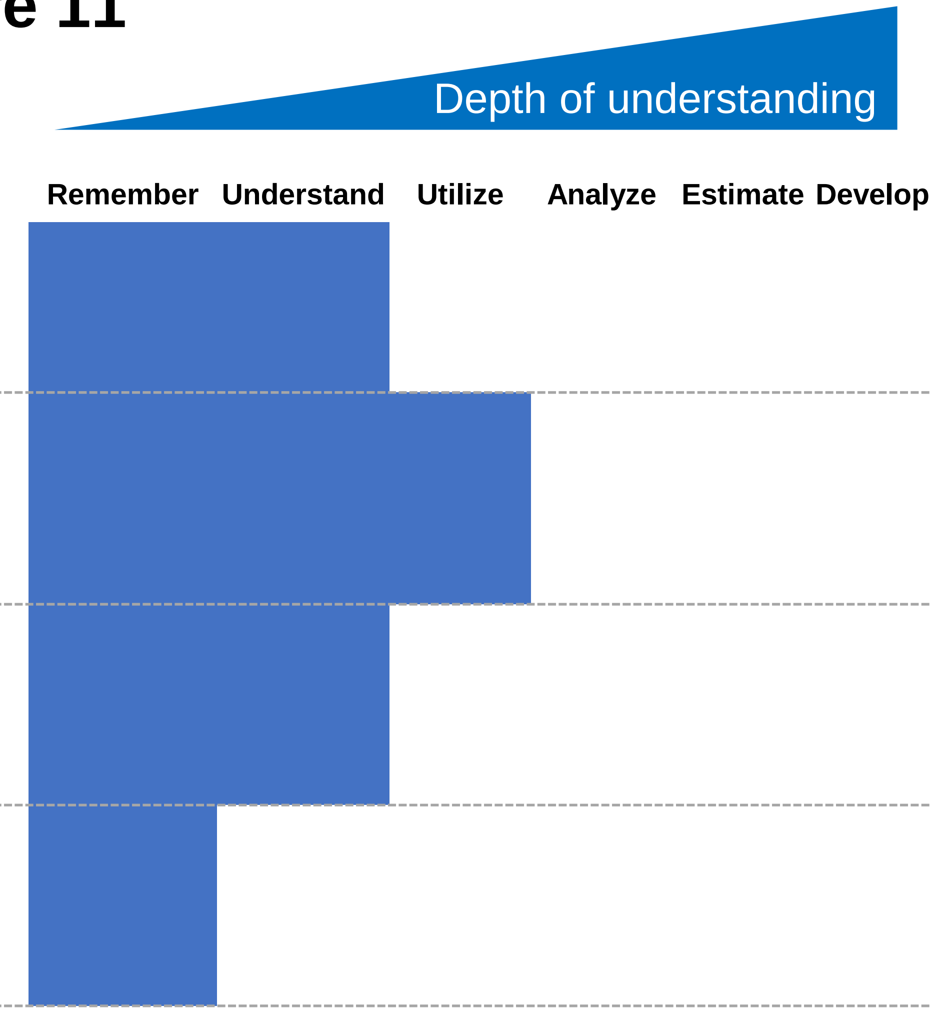
Johannes Betz / Prof. Dr.-Ing. Markus Lienkamp / Prof. Dr.-Ing. Boris Lohmann



Lecture Overview

Overall Introduction for the Lecture 18.10.2018 – Betz Johannes	**6 Pathfinding: From British Museum to A*** 29.11.2018 – Lennart Adenaw	**11 Reinforcement Learning** 17.01.2019 – Christian Dengler
1 Introduction: Artificial Intelligence 18.10.2018 – Betz Johannes	**P6:** 29.11.2018 – Lennart Adenaw	**P11** 17.01.2019 – Christian Dengler
P1: 18.10.2018 – Betz Johannes	**7 Introduction: Artificial Neural Networks** 06.12.2018 – Lennart Adenaw	**12 AI-Development** 24.01.2019 – Johannes Betz
2 Perception 25.10.2018 – Betz Johannes	**P7** 06.12.2018 – Lennart Adenaw	**P12** 24.01.2019 – Johannes Betz
P2: 25.10.2018 – Betz Johannes	**8 Deep Neural Networks** 13.12.2018 – Jean-Michael Georg	**13 Free Discussion** 31.01.2019 – Betz/Adenaw
3 Supervised Learning: Regression 08.11.2018 – Alexander Wischnewski	**P8** 13.12.2018 – Jean-Michael Georg	
P3: 08.11.2018 – Alexander Wischnewski	**9 Convolutional Neural Networks** 20.12.2018 – Jean-Michael Georg	
4 Supervised Learning: Classification 15.11.2018 – Jan-Cedric Mertens	**P9** 20.12.2018 – Jean-Michael Georg	
P4: 15.11.2018 – Jan-Cedric Mertens	**10 Recurrent Neural Networks** 10.01.2019 – Christian Dengler	
5 Unsupervised Learning: Clustering 22.11.2018 – Jan-Cedric Mertens	**P10** 10.01.2019 – Christian Dengler	

Objectives of the lecture 11



Understand **which kind of problems** reinforcement learning (RL) can tackle.

Understand the concept of a **value function** and **action-value function** in discrete state and action spaces.

Understand the **basic RL methods** in discrete state and action space.

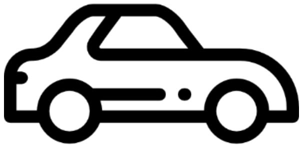
Understand the **basic policy gradient** for continuous state and actions space.

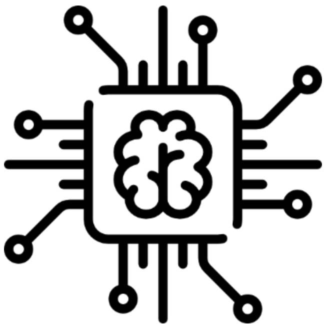
Reinforcement Learning

Johannes Betz / Prof. Dr. Markus Lienkamp / Prof. Dr. Boris Lohmann
(Christian Dengler, M. Sc.)

Agenda

1. Terminology and Concept
    1.1 Terminology and problem definition
    1.2 Motivation for RL in engineering
2. RL in discrete state- and action-spaces
    2.1 Markov decision processes
    2.2 Value-Function, Q-learning etc.
3. RL in continuous state- and action-spaces
    3.1 Overview of methods
    3.2 Connection to Optimal Control
    3.3 Exploration in the action space
    3.4 Exploration in parameter space (Optional)

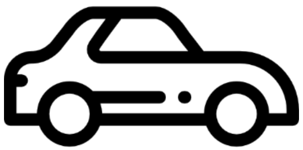


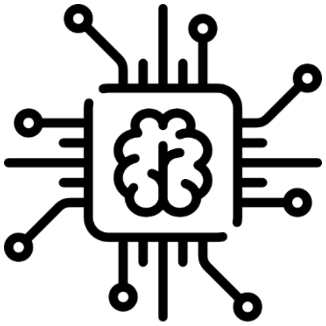


Reinforcement Learning
Johannes Betz / Prof. Dr. Markus Lienkamp / Prof. Dr. Boris Lohmann
(Christian Dengler, M. Sc.)

Agenda

1. Terminology and Concept
    - **1.1 Terminology and problem definition**
    - 1.2 Motivation for RL in engineering
2. RL in discrete state- and action-spaces
    - 2.1 Markov decision processes
    - 2.2 Value-Function, Q-learning etc.
3. RL in continuous state- and action-spaces
    - 3.1 Overview of methods
    - 3.2 Connection to Optimal Control
    - 3.3 Exploration in the action space
    - 3.4 Exploration in parameter space (Optional)





1.1 Terminology and problem definition

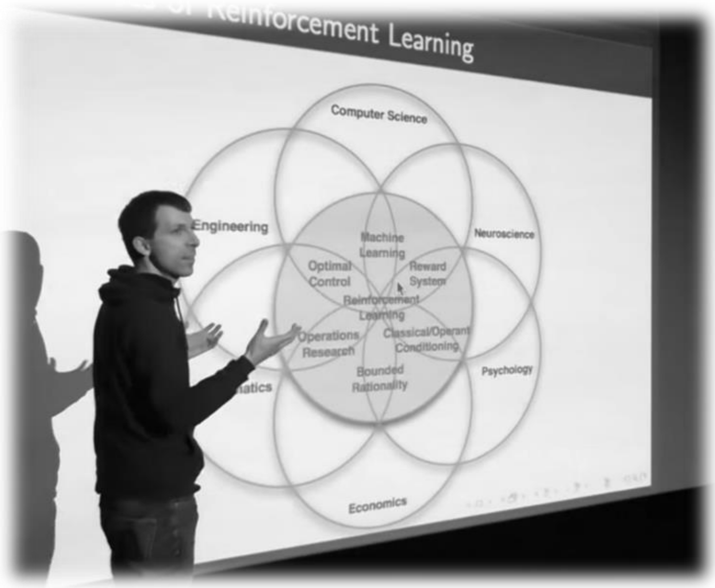
Revision

- Supervised Learning
  - Learning on labeled data, e.g. image classification using labeled dataset and a deep neural network
- Unsupervised Learning
  - Learning on unlabeled data, e.g. clustering using K-means on a database of customers
- Reinforcement Learning
  - ?

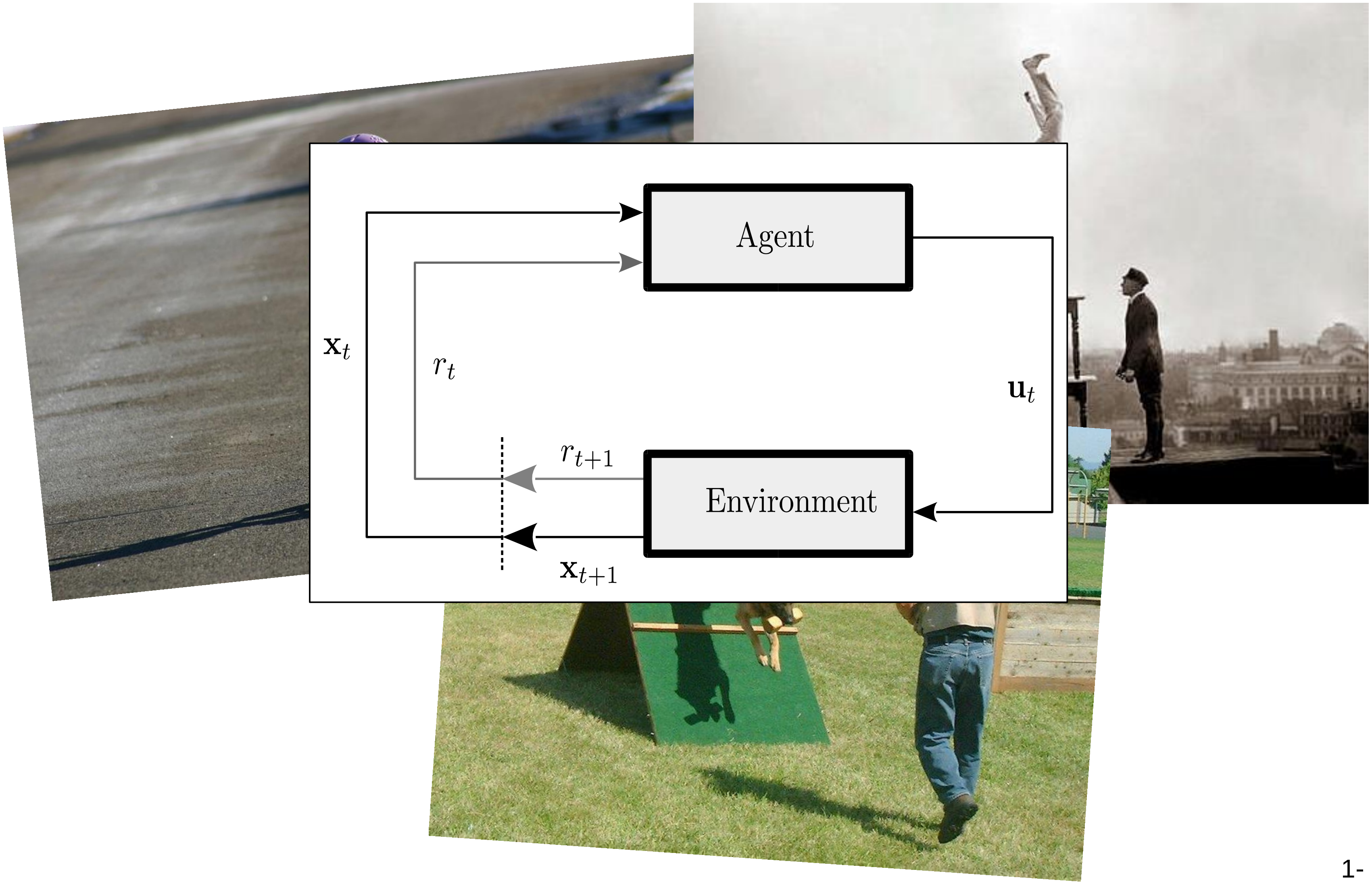
1.1 Terminology and problem definition

"... So what is that problem? It's essentially the science of decision making. I guess that's what makes it so general and so interesting across many many fields ... It's trying to understand the optimal way to make decisions..."

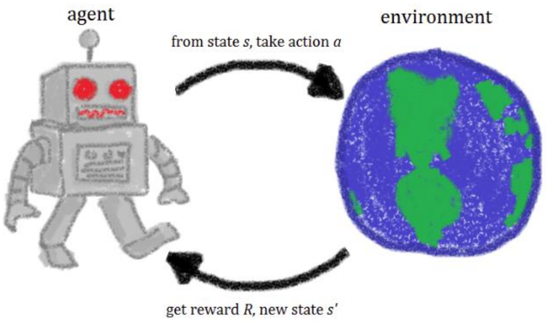
David Silver, DeepMind, 2015



1.1 Terminology and problem definition



1.1 Terminology and problem definition



- **Agent**:
  The decision taking unit, in our case a computer executing a policy/strategy.
- **Environment**:
  Everything outside of the agent. This would in theory include the universe, but it is usually sufficient to only consider a small part, e.g. a space in proximity of the agent.
- **Reward**:
  A scalar signal that the agent receives, which depends on how it is performing in the environment. In our case, we will design what is rewarded.

1.1 Terminology and problem definition

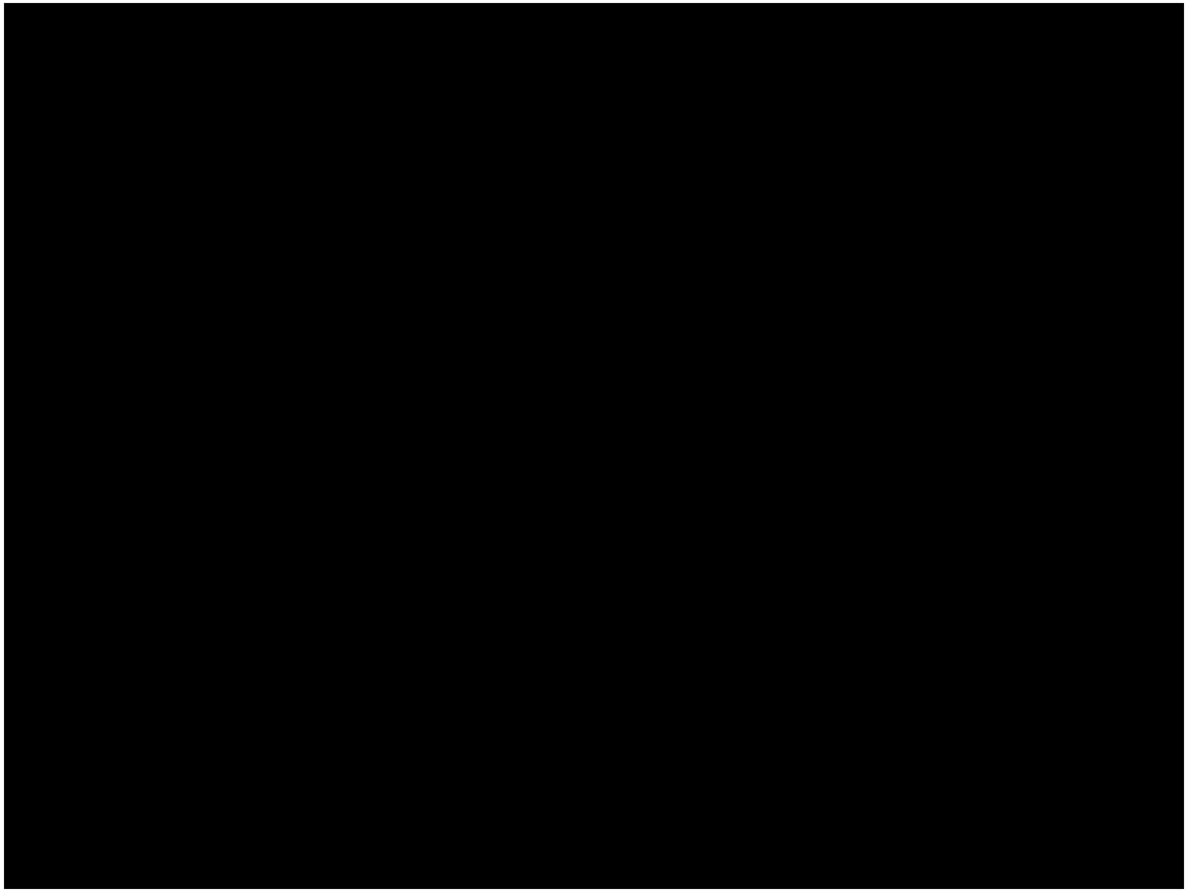
- **State**:
  A signal describing the *environment* (or at least the important part), e.g. the positions and velocities of the limbs of a robot. The state is often assumed Markovian (full information).
- **Action**:
  The *agent* decides on an action, following its policy
- **Goal of RL:**
  Train the agent in a way that it receives as much reward as possible.

1.1 Terminology and problem definition

Example

- **Agent**:
- **Environment**:
- **Reward**:
- **State**:
- **Action**:

1.1 Terminology and problem definition



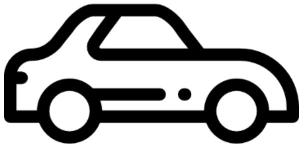
Mnih et al.: Human-level control through deep reinforcement learning, Nature, 2015

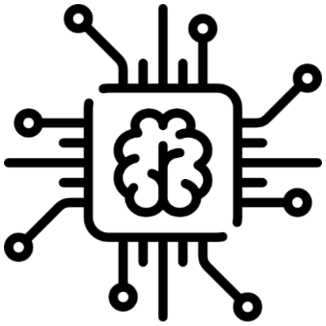
1.1 Terminology and problem definition

Heess et al.: Emergence of Locomotion Behaviours in Rich Environments, *CoRR* , 2017

Reinforcement Learning
Johannes Betz / Prof. Dr. Markus Lienkamp / Prof. Dr. Boris Lohmann
(Christian Dengler, M. Sc.)

Agenda

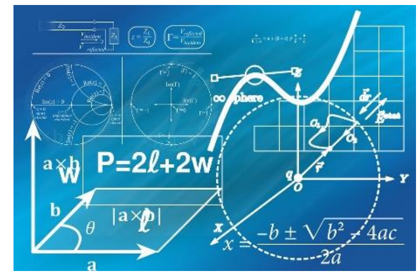




1.2 Motivation for RL in engineering

Classical engineering approach for automatic control

1. Understand the problem and derive a simplified model



2. Use the simplified model equations and some analytical tools to derive a control law

3. Apply the control law

1.2 Motivation for RL in engineering

Classical engineering approach for automatic control

- Understand the problem and derive a simplified model
  - Problems can be very complex, thus for complex problems it can be very hard to deduce a model that accurately describes reality.
- Use the simplified model equations and some analytical tools to derive a control law
  - This approach works well for most simple problems, for general nonlinear models in high state-space, this is often very tricky and requires a team of experts on this specific problem.
- Apply the control law
  - The control law is executed as is, any change in the system needs a manual change in the control law by experts.

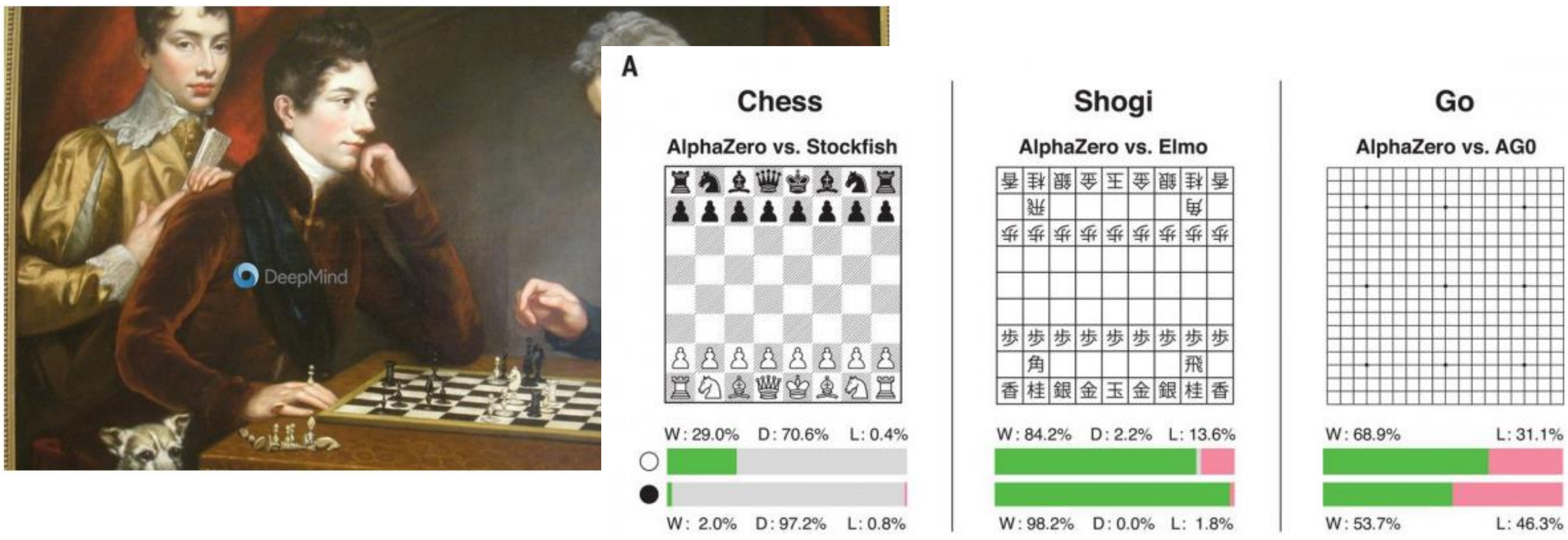
1.2 Motivation for RL in engineering

Interaction of mechanics, electronics and *software*

1.2 Motivation for RL in engineering

DeepMind's AlphaZero AI is the new champion in chess, shogi, and Go

by IVAN MEHTA — 11 days ago in ARTIFICIAL INTELLIGENCE



https://thenextweb.com/artificial-intelligence/2018/12/07/deepminds-alphazero-ai-is-the-new-champion-in-chess-shogi-and-go/

1.1 + 1.2 Wrap up

- Reinforcement Learning describes the high level **idea of learning to make good decisions** by **repeating** a task and receiving a **reward signal**. It is not an algorithm!
- The specific algorithm then depends on the task. E.g. do we want to learn to play a game or control a robot?
- The RL setup includes an **agent** and his **environment**. The agent takes **actions**, and perceives/receives the **state** and receives a **reward**.
- Can lead to better decision making than manual human designs → promising also for engineering (topic of research).

1.3 Depth of the topic and scope of the lecture

- Lecture by David Silverman (Google Deepmind),**15h material**
  http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html

- Lecture by Sergey Levine (UC Berkeley), **>20h material**
  http://rail.eecs.berkeley.edu/deeprlcourse/

- **Requires** knowledge of **basic probability theory**.

→ Focus here on the most simple case of making decisions in discrete states and actions.

1.3 Depth of the topic and scope of the lecture

Basic probability theory for discrete variables

- **Probability mass function** (PMF) $P(x)$, dice example:
  x random variable, describes the number of rolled eyes.

$$P(x=1)=\tfrac{1}{6}$$
$$P(x=2)=\tfrac{1}{6}$$
$$\vdots$$
$$P(x=6)=\tfrac{1}{6}$$

$$\sum_i P(x=\mathrm{x}_i)=1$$

$$P(x=\mathrm{x}_i):=P(\mathrm{x}_i)$$

- Sampling from a PMF: $\mathrm{x} \sim P(x)$
  →actually throwing the dice and observing the result.

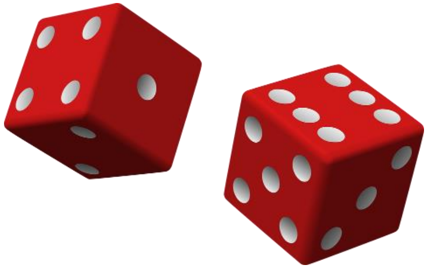
1.3 Depth of the topic and scope of the lecture

Basic probability theory for discrete variables

- **Probability mass function** (PMF), 2 dice example $P(x, y)$.

  x random variable: 1 if (sum of eyes)>5, else 0

  y random variable: 1 if atleast one dice rolled a 5, else 0



$$P(x=0, y=0) = \frac{10}{36} = \frac{5}{18}$$
$$P(x=0, y=1) = 0$$
$$P(x=1, y=0) = \frac{15}{36}$$
$$P(x=1, y=1) = \frac{11}{36}$$

1.3 Depth of the topic and scope of the lecture

Basic probability theory for discrete variables

- **Probability mass function** (PMF), 2 dice example $P(x, y)$.

  x random variable: 1 if (sum of eyes)>5, else 0

  y random variable: 1 if atleast one dice rolled a 5, else 0

Dice1\Dice2	1	2	3	4	5	6
1	2	3	4	5	6	7
2	3	4	5	6	7	8
3	4	5	6	7	8	9
4	5	6	7	8	9	10
5	6	7	8	9	10	11
6	7	8	9	10	11	12

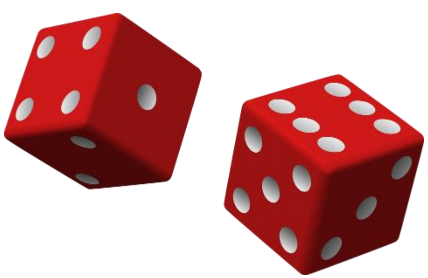
1.3 Depth of the topic and scope of the lecture

Basic probability theory for discrete variables

- **Conditional probability**, dice example $P(x|y)$.

  x random variable: 1 if sum of eyes>5, else 0

  y random variable: 1 if atleast one dice rolled a 5, else 0



$$P(x=0|y=0) = \frac{15}{25} = \frac{3}{5}$$

$$P(x=1|y=0) = \frac{10}{25} = \frac{2}{5}$$

We know that there is no 5!

1.3 Depth of the topic and scope of the lecture

Basic probability theory for discrete variables

- Expected value, dice example:

$$\mathbb{E}^P[x] = \sum_i P(x_i)\; x_i$$
$$= \tfrac{1}{6} \cdot 1 + \tfrac{1}{6} \cdot 2 + \ldots$$
$$= \frac{21}{6} = 3.5$$

- Useful relations:

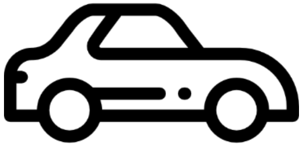
$$P(x) = \sum_i P(x, \mathrm{y}_i)$$

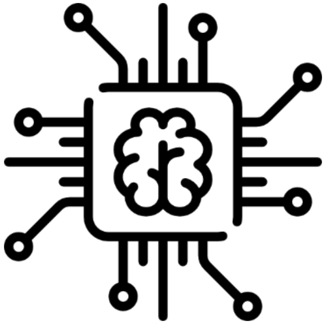
$$P(x, y) = P(x|y) \cdot P(y)$$

Reinforcement Learning
Johannes Betz / Prof. Dr. Markus Lienkamp / Prof. Dr. Boris Lohmann
(Christian Dengler, M. Sc.)

Agenda

1. Terminology and Concept
   - 1.1 Terminology and problem definition
   - 1.2 Motivation for RL in engineering
2. RL in discrete state- and action-spaces
   - **2.1 Markov decision processes**
   - 2.2 Value-Function, Q-learning etc.
3. RL in continuous state- and action-spaces
   - 3.1 Overview of methods
   - 3.2 Connection to Optimal Control
   - 3.3 Exploration in the action space
   - 3.4 Exploration in parameter space (Optional)



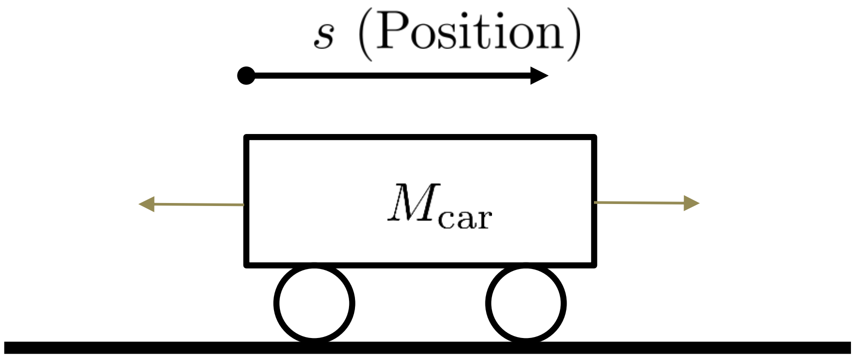


2.1 Markov-decision-process

Markov State

A state is Markov, if

$$P(x_{t+1}|x_t) = P(x_{t+1}|x_t, x_{t-1}, \ldots, x_0)$$



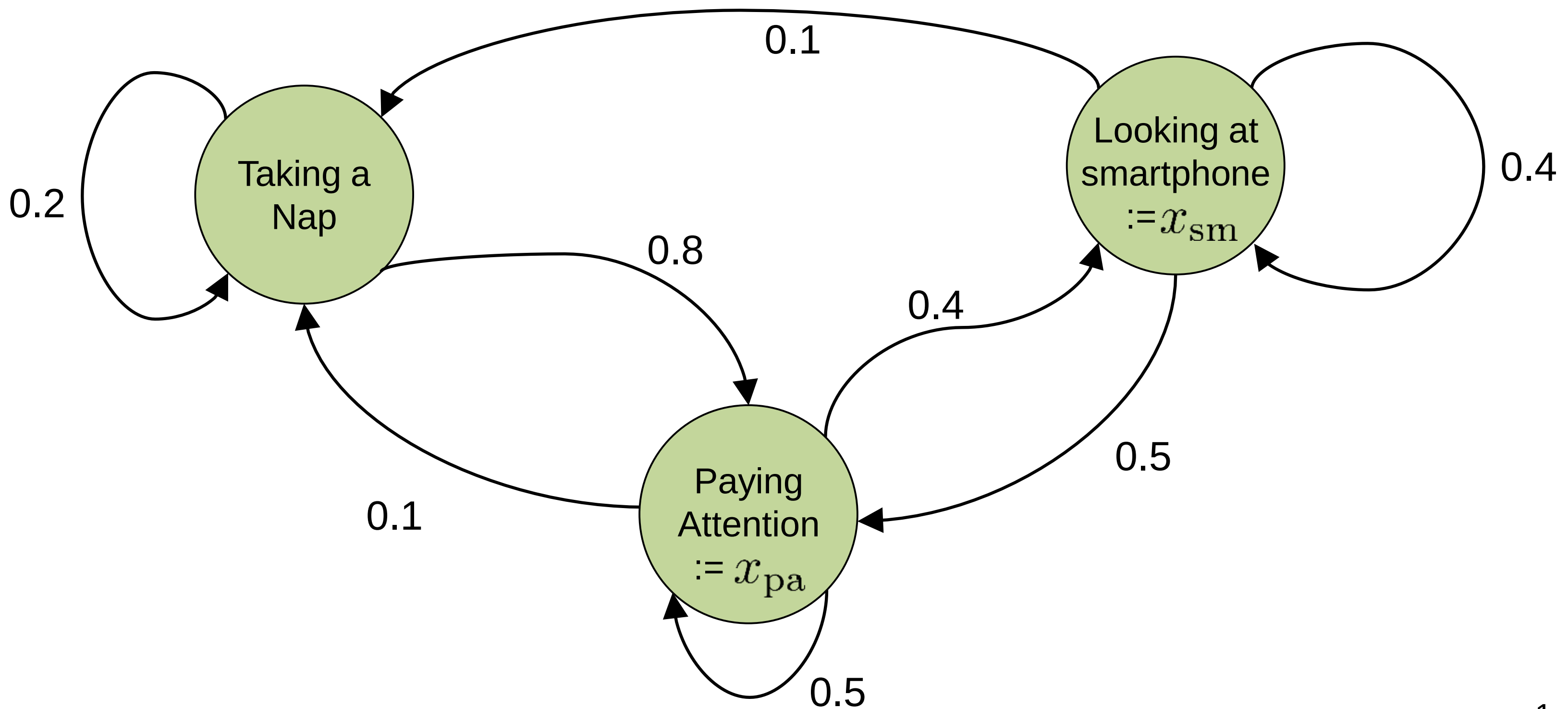
$$x_t = s$$

$$x_t = \begin{bmatrix} s \\ \dot{s} \end{bmatrix}$$

2.1 Markov-decision-process

Markov-process

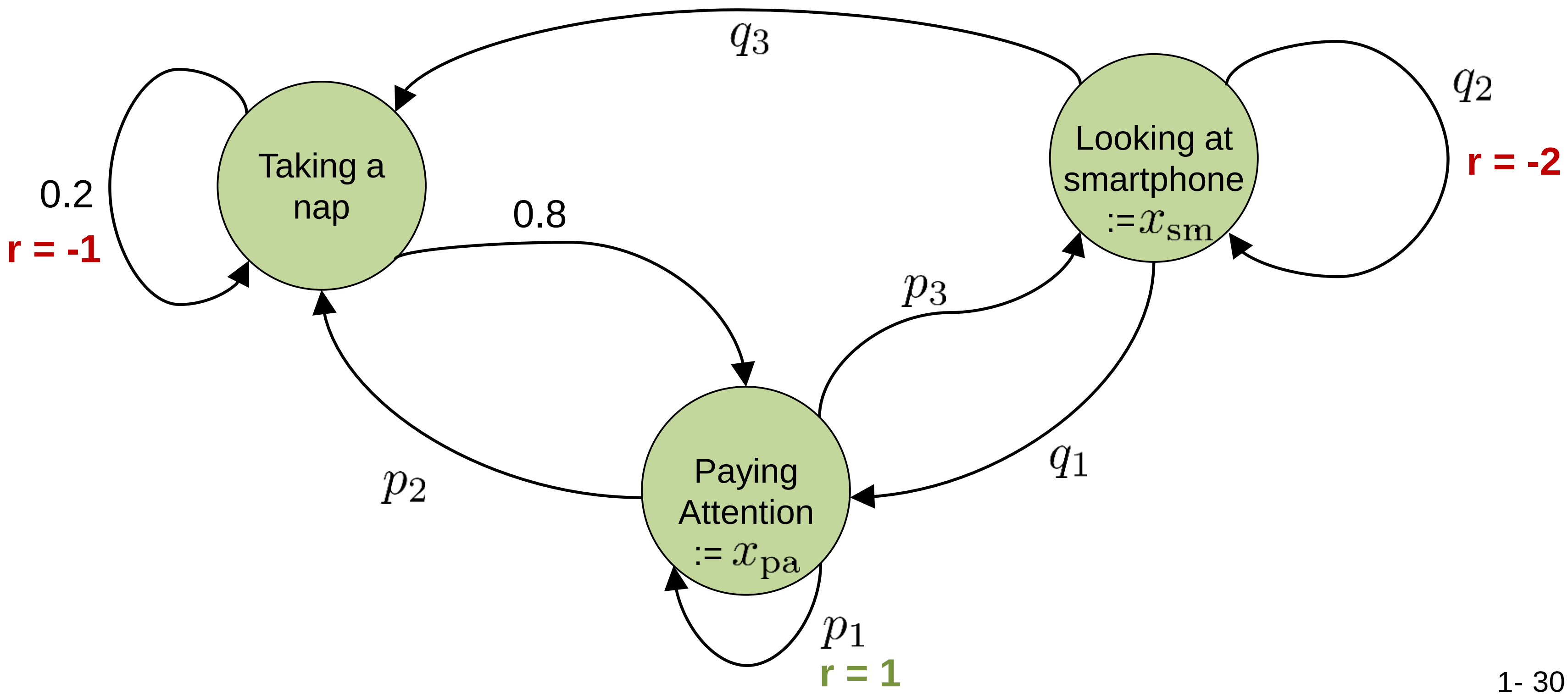
A Markov-process is sequence of random states with the Markov property.



2.1 Markov-decision-process

Markov-decision-process

A Markov-decision-process is a Markov-process with additional rewards, and the possibility to affect transition probabilities.



2.1 Markov-decision-process

Markov-decision-process

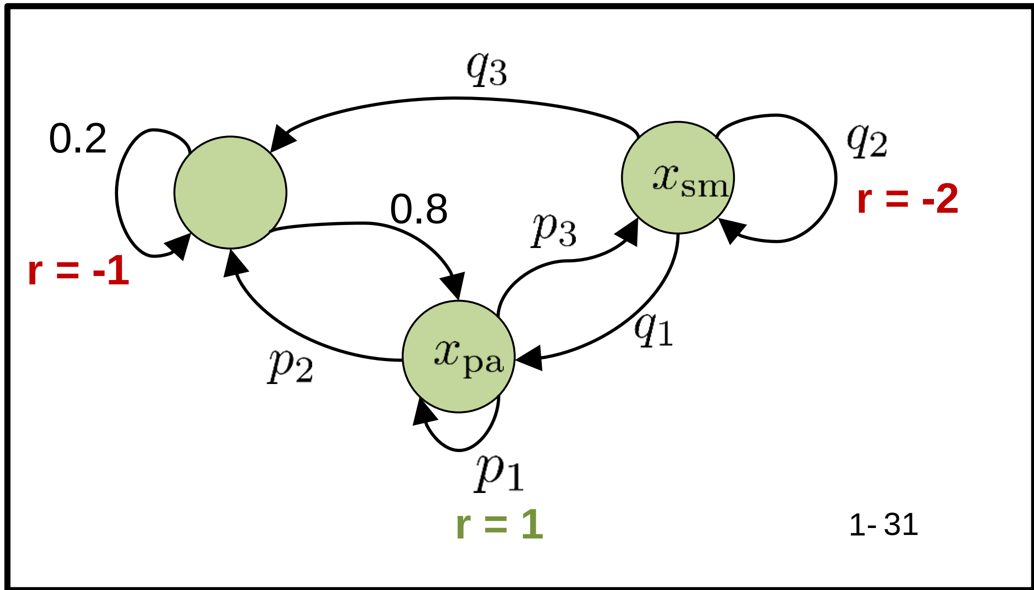
- **Goal**:
  - Find strategy which maximizes future rewards, i.e.:

Find probabilities $p_1, p_2, p_3, q_1, q_2, q_3$

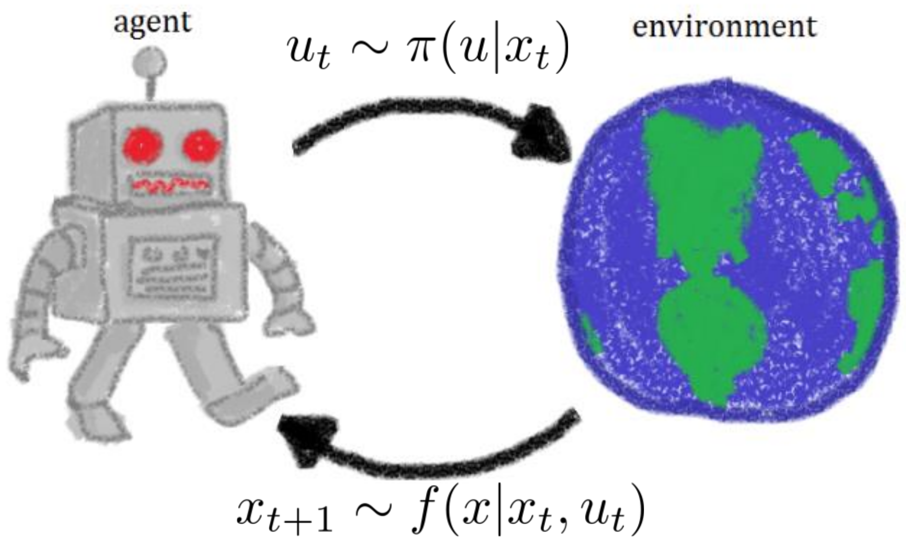
$$\sum_i p_i = 1, \sum_i q_i = 1$$

Maximizing :

$$\sum_{t=0}^{\infty} \gamma^t \cdot r_t \; ; \; \gamma \in [0,1]$$



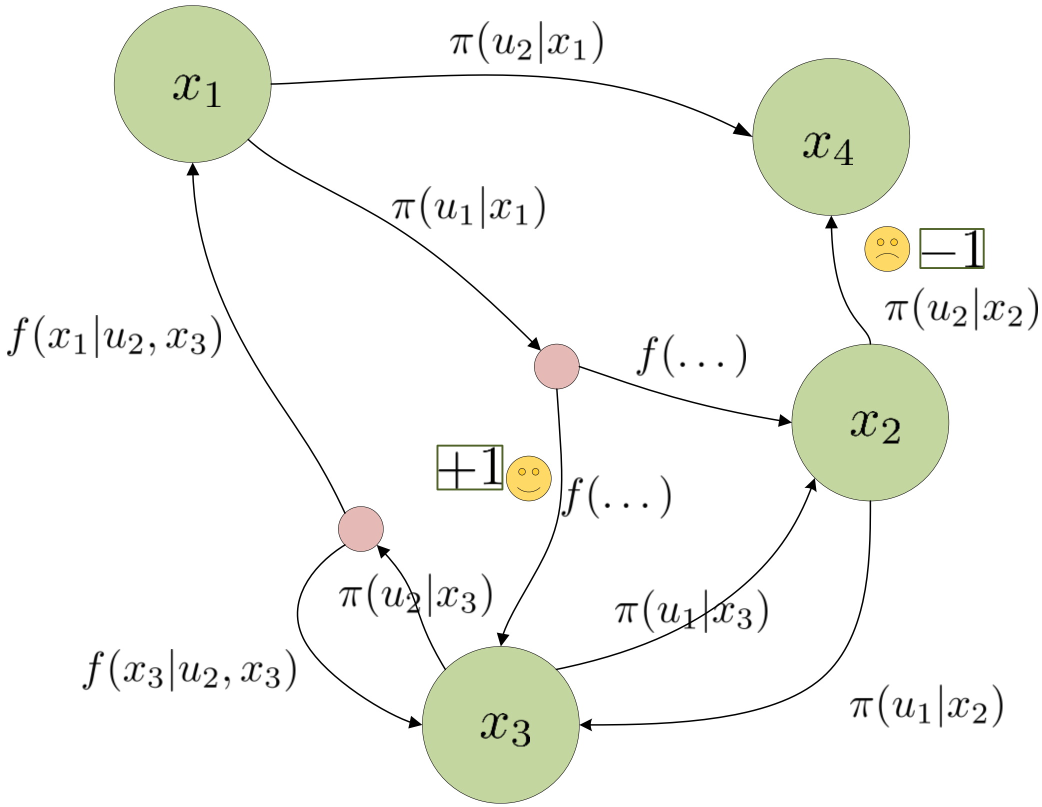
2.1 Markov-decision-process



- **Legend**:
  - State: x
  - Action: u
  - Policy: $\pi(u|x_t)$
  - Behavior of the environment: $f(x|x_t, u)$

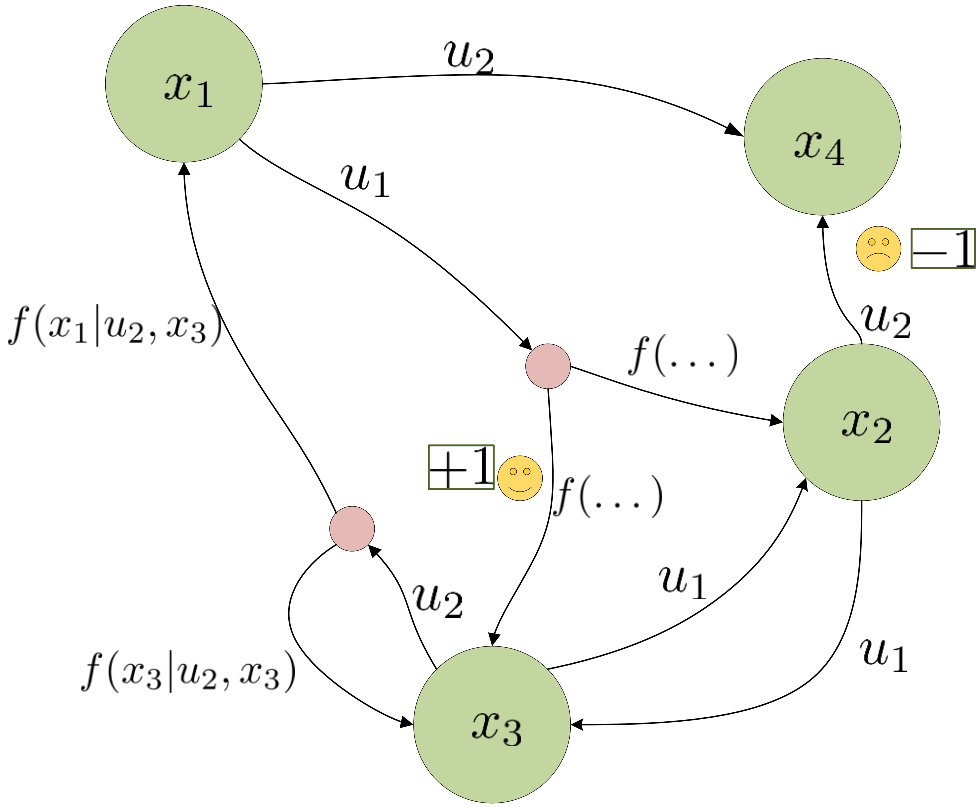
- **Assumptions**:
  - $\pi(u|x_t)$ and $f(x|x_t, u)$ are discrete probability distributions.
  - x is a Markovian state.

2.1 Markov-decision-process



- **Goal**:
  - Find strategy $\pi(u|x)$ which maximize rewards

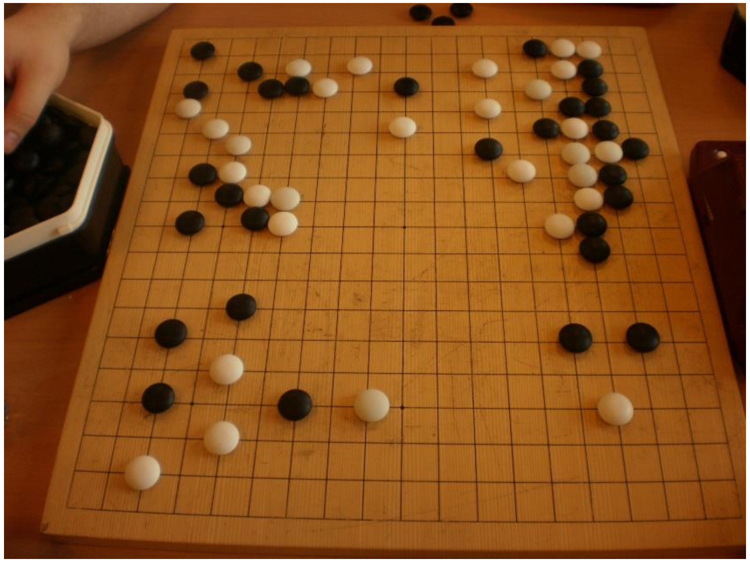
2.1 Markov-decision-process



- **Goal**:
  - Find strategy $\pi(u|x)$ which maximize rewards

2.1 Markov-decision-process

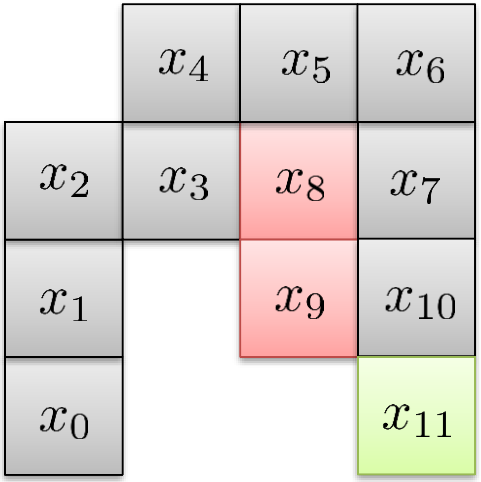
Example of discrete MDP’s



2.1 Markov-decision-process

Example: Grid World

- 12 states/positions
- 4 actions per state: go **up**, **down left**, **right** (Hitting a wall is possible and means no movement)
- Different reward depending on the state.
  - -1 when moving to a grey or green state
  - -2 when moving to a red state
- Initial state x_0 (One always starts here)
- Absorbing state x_{11} (episode finished, 0 reward from here on)

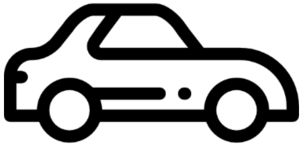


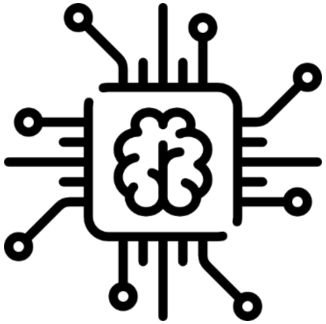
Reinforcement Learning

Johannes Betz / Prof. Dr. Markus Lienkamp / Prof. Dr. Boris Lohmann
(Christian Dengler, M. Sc.)

Agenda

1. Terminology and Concept
    - 1.1 Terminology and problem definition
    - 1.2 Motivation for RL in engineering
2. RL in discrete state- and action-spaces
    - 2.1 Markov decision processes
    - **2.2 Value-Function, Q-learning etc.**
3. RL in continuous state- and action-spaces
    - 3.1 Overview of methods
    - 3.2 Connection to Optimal Control
    - 3.3 Exploration in the action space
    - 3.4 Exploration in parameter space (Optional)





2.2 Value function, Q-learning etc

Definitions

Value Function

Function depending on the state and a policy. The function returns the expected future reward, starting in a state x and then always following a policy π.

Action Value Function

Function depending on the state, the next action and a policy. The function returns the expected future reward, starting in a state x, then choosing action u and afterwards following policy π.

2.2 Value function, Q-learning etc

Value function

Value Function

Function depending on the state and a policy. The function returns the expected future reward, starting in a state x and then always following a policy π .

$$V^{\pi}(x) = \mathbb{E}^{\pi}\left[\sum_{\tau=t}^{\infty} \gamma^{\tau-t} r_{\tau+1} | x_t = x\right]$$

$0 < \gamma <= 1$ discount factor

2.2 Value function, Q-learning etc

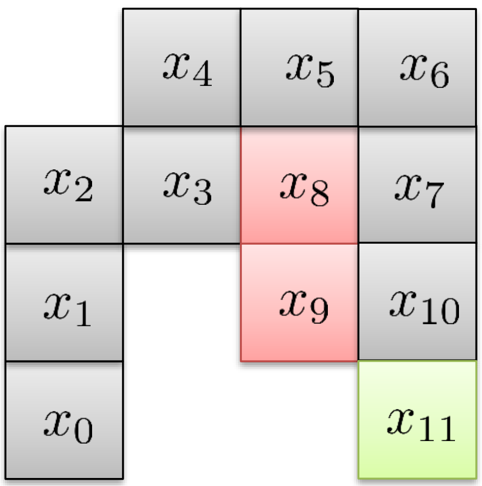
Value function

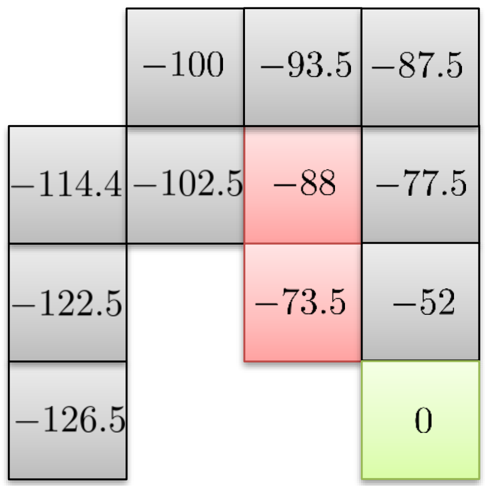
$$V^{\pi}(x) = \mathbb{E}^{\pi}\left[\sum_{\tau=t}^{\infty} \gamma^{\tau-t} r_{\tau+1} | x_t = x\right]$$

$0 < \gamma <= 1$ discount factor

2.2 Value function, Q-learning etc

Grid World, Value Function





$V^{\pi_1}(x)$

V(x) in PC Memory:

x_0	-126.5
x_1	-122.5
$\vdots$	
x_{10}	-52
x_{11}	0

Uniform random strategy: $\pi_1(\uparrow, x) = \pi_1(\leftarrow, x) = \pi_1(\downarrow, x) = \pi_1(\rightarrow, x) = \frac{1}{4}$

2.2 Value function, Q-learning etc

Policy evaluation using the Bellman equation

The value function can be determined if all probabilities are known (transitions + policy) by iterating the Bellman equation for all states.

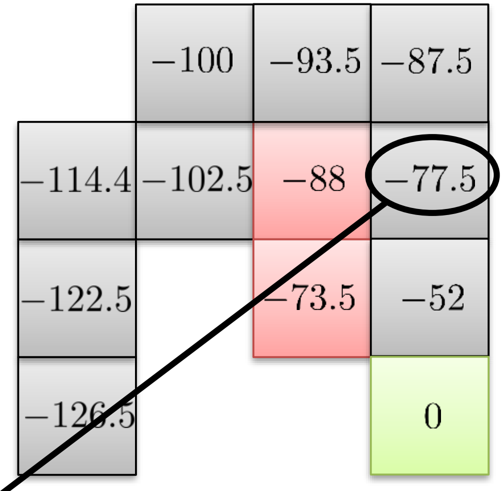
until convergence of V:

    for all states x:

        $V^{\pi}_{k+1}(x) = \sum_u \pi(u|x_t) \cdot (r_{t+1} + \gamma V^{\pi}_k(x_{t+1}))\,|x_t = x$

    end

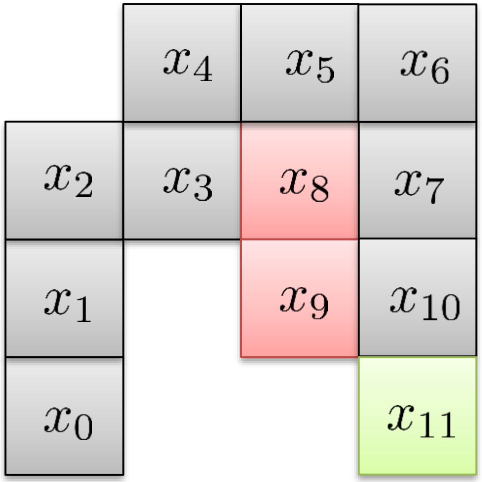
end



2.2 Value function, Q-learning etc

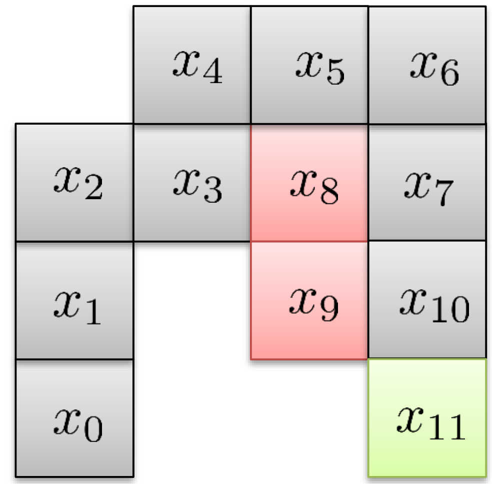
Policy evaluation using the Bellman equation

For small MDP one could just solve a system of equations instead of doing it iteratively. The equations are the Bellman equation for each state, and the unknowns are the value function at the states.



2.2 Value function, Q-learning etc

Policy evaluation using the Bellman equation



Solve for $V(x_0), \dots V(x_{10})$:

$$V(x_0) = \frac{3}{4}(-1 + V(x_0)) + \frac{1}{4}(-1 + V(x_1))$$

$$V(x_1) = \frac{1}{4}(-1 + V(x_0)) + \frac{2}{4}(-1 + V(x_1)) + \frac{1}{4}(-1 + V(x_2))$$

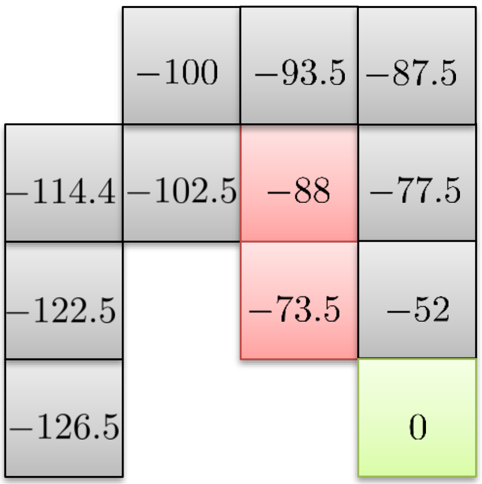
$$\vdots$$

$$V(x_{10}) = \frac{1}{4}(-2 + V(x_9)) + \frac{1}{4}(-1 + V(x_7)) + \frac{1}{4}(-1 + V(x_{10})) + \frac{1}{4}(0 + V(x_{11}))$$

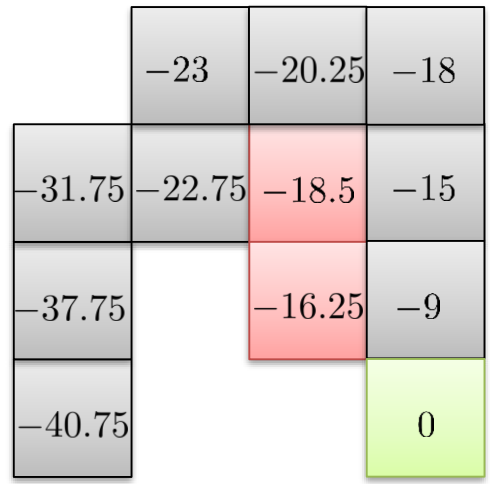
$$V(x_{11}) = 0$$

2.2 Value function, Q-learning etc

Grid World, Value Function



$V^{\pi_1}(x)$



$V^{\pi_2}(x)$

Uniform random strategy: $\pi_1(\uparrow, x) = \pi_1(\leftarrow, x) = \pi_1(\downarrow, x) = \pi_1(\rightarrow, x) = \frac{1}{4}$

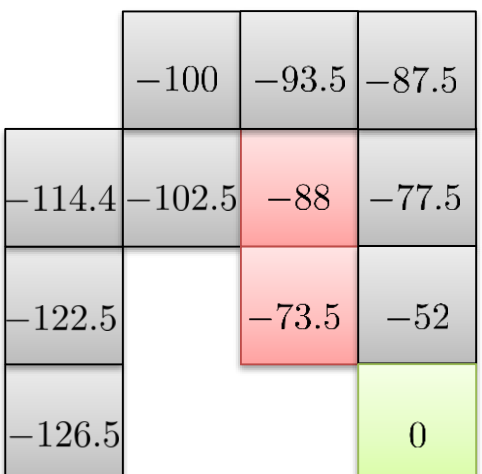
Different Strategy: $\pi_2(\uparrow, x) = \pi_2(\downarrow, x) = \pi_2(\rightarrow, x) = \frac{1}{3}, \quad \pi_2(\leftarrow, x) = 0$

2.2 Value function, Q-learning etc

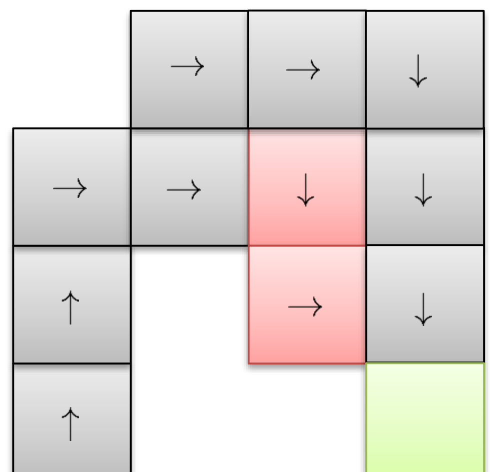
Policy improvement

In order to **improve** the policy and get more reward, one creates a new **deterministic policy**, choosing in every state the action with the most expected future reward, according to the **old V(x)**

$$u_{\text{greedy}} = \arg\max_u \mathbb{E}[r_{t+1} + \gamma V^{\pi}(x_{t+1}) | x_t = x, u_t = u]$$



$V^{\pi_1}(x)$

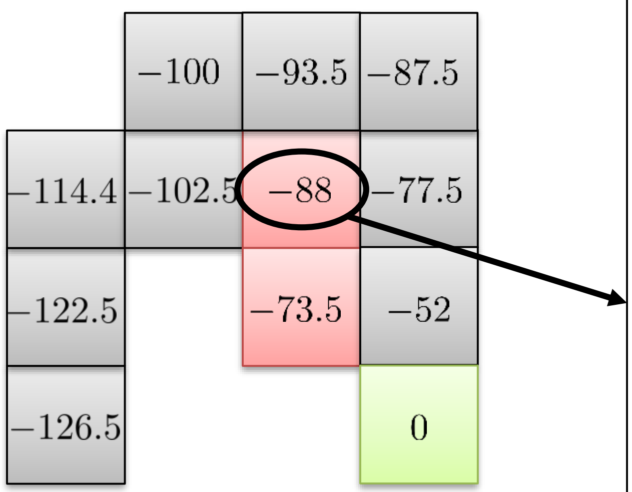


$\pi_2(x)$

2.2 Value function, Q-learning etc

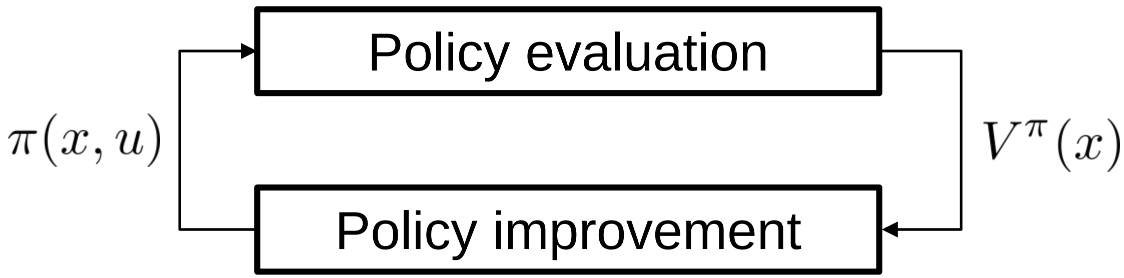
Policy improvement

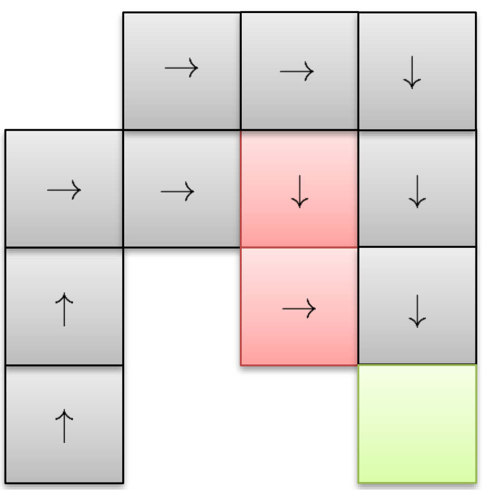
$$u_{\text{greedy}} = \arg\max_u \mathbb{E}[r_{t+1} + \gamma V^{\pi}(x_{t+1}) | x_t = x, u_t = u]$$



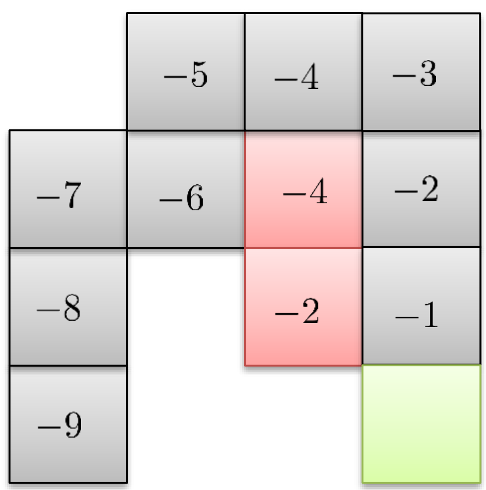
$V^{\pi_1}(x)$

2.2 Value function, Q-learning etc

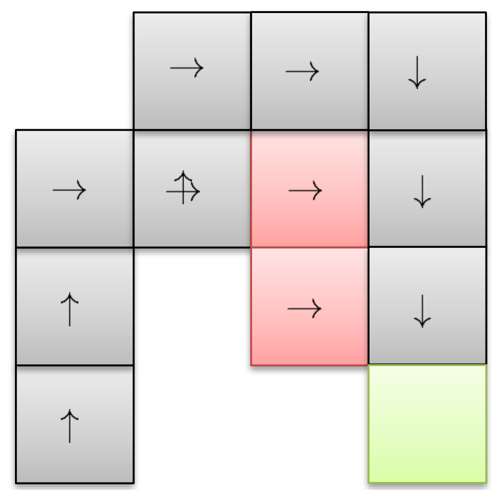




$\pi_2(x)$



$V^{\pi_2}(x)$



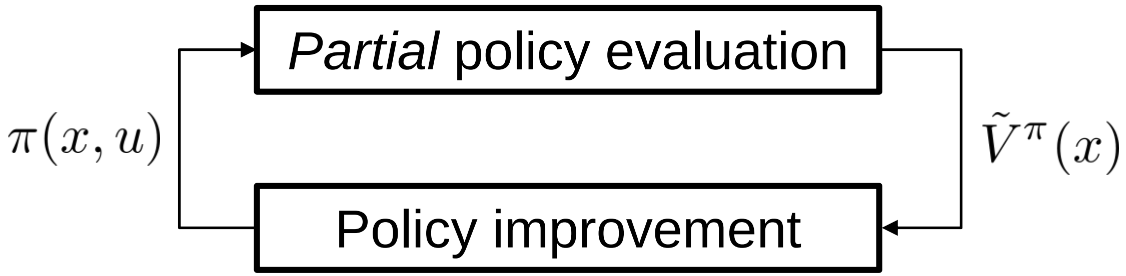
$\pi_3(x)$

2.2 Value function, Q-learning etc

This is guaranteed to converge to the optimal policy, however for simple MDP‘s with known transitions, there are much more efficient algorithms.

2.2 Value function, Q-learning etc

Reducing Computation Time: Generalized Policy iteration



It is not always necessary to let the value function converge, improvements on the policy can be made earlier.

2.2 Value function, Q-learning etc

Definitions

Action Value Function

Function depending on the state, the next action and a policy. The function returns the expected future reward, starting in a state x, then choosing action u and afterwards following policy π.

$$Q^\pi(x,u) = \mathbb{E}^\pi \left[\sum_{\tau=t}^{\infty} \gamma^{\tau-t} r_{\tau+1} \; | x_t = x, \; u_t = u \right]$$

$$= \mathbb{E}^\pi \left[r_{t+1} + \gamma V(x_{t+1}) \; | x_t = x, \; u_t = u \right]$$

2.2 Value function, Q-learning etc

Definitions

Action Value Function

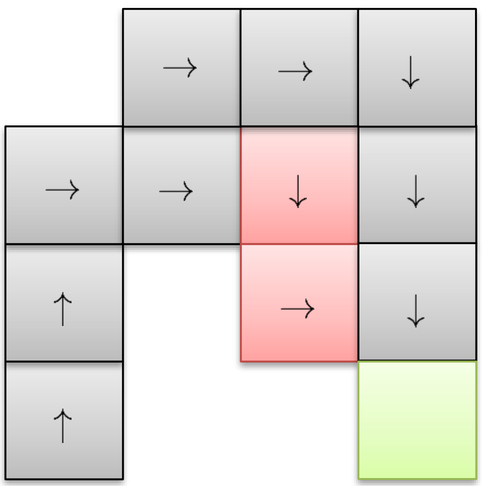
Function depending on the state, the next action and a policy. The function returns the expected future reward, starting in a state x, then choosing action u and afterwards following policy π.

$$Q^{\pi}(x, \pi(x)) = V^{\pi}(x)$$

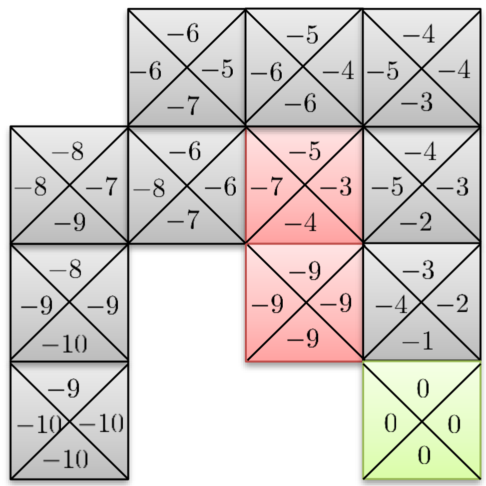
$$\begin{aligned} Q^{\pi}(x,u) &= \mathbb{E}^{\pi}\left[\sum_{\tau=t}^{\infty} \gamma^{\tau-t} r_{\tau+1} \;|x_t = x,\; u_t = u\right] \\ &= \mathbb{E}^{\pi}\left[r_{t+1} + \gamma Q(x_{t+1}, \pi(x_{t+1}))\;|x_t = x,\; u_t = u\right] \end{aligned}$$

2.2 Value function, Q-learning etc

Grid World, Action Value Function



$\pi_2(x)$



$Q^{\pi_2}(x,u)$

Q(x, u) in PC Memory:

	↑	←	↓	→
x_0	-9	-10	-10	-10
x_1	-8	-9	-10	-9
$\vdots$		$\vdots$		
x_{10}	-3	-4	-1	-2
x_{11}	0	0	0	0

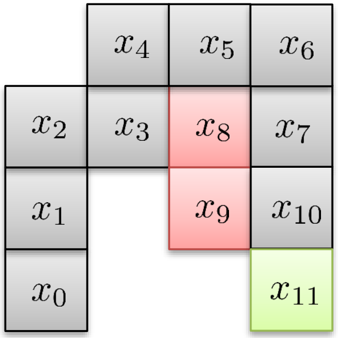
2.2 Value function, Q-learning etc

Why Q-function instead of Value Function

- **Advantage:**
  - Contains all the information needed to do the policy improvement. No need to know the transition probabilities!!!

$$u_{\text{greedy}}(x) = \arg\max_u \mathbb{E}[r_{t+1} + \gamma V^{\pi}(x_{t+1}) | x_t = x, u_t = u]$$

$$u_{\text{greedy}}(x) = \arg\max_u Q(x, u)$$

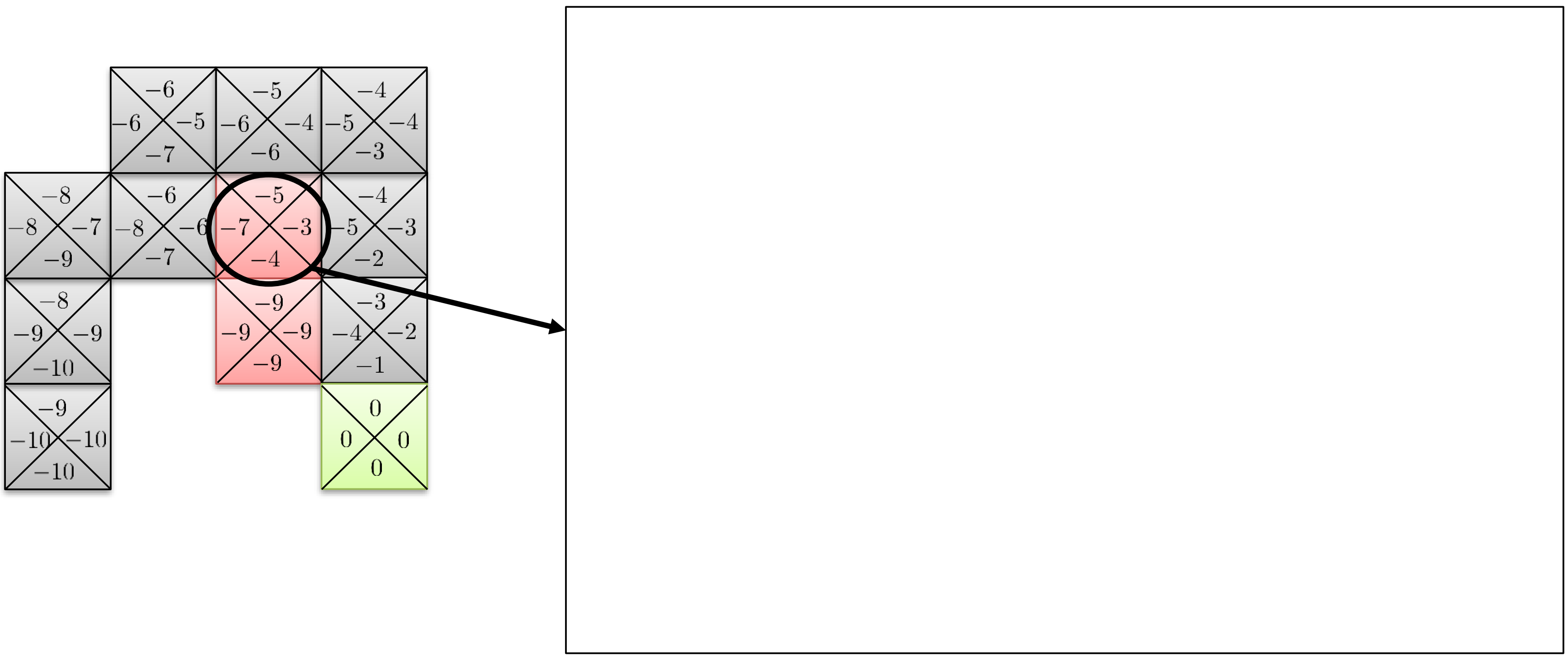


- **Disadvantage:**
  - More memory required and needs more time to be trained.

2.2 Value function, Q-learning etc

Grid World, Action Value Function

$$u_{\text{greedy}}(x) = \arg\max_u Q(x,u)$$



2.2 Value function, Q-learning etc

Short wrap up

- Using the Bellman equation we can learn the value or action-value function. (e.g. iterative or system of equations)
- Once we have value or action-value function, we can improve the policy
  - If we used the value function, we need to know the transition dynamics also.
  - If we use the action-value function, we can just read the best value (no need to know the transition dynamics), but we need more memory.

2.2 Value function, Q-learning etc

Model free learning

- **So far**, we assumed to know the transition dynamics (where do we end up if we chose $\leftarrow$ in state x?).

until convergence of V:
  for all states x:
    $V^{\pi}_{k+1}(x) = \sum_u \pi(u|x_t) \cdot (r_{t+1} + V^{\pi}_k(x_{t+1}))\,|x_t = x$
  end
end

- If we don‘t have the model, we can use **data from interactions** with the MDP. We assume the data was generated by π (on-policy).

Iterate over all data tuples $(x_t, u_t, r_{t+1}, x_{t+1})$:
  $V^{\pi}_{k+1}(x_t) = (1-\alpha) \cdot V^{\pi}_k(x_t) + \alpha \cdot (r_{t+1} + \gamma V^{\pi}_k(x_{t+1}))$
  or
  $Q^{\pi}_{k+1}(x_t, u_t) = (1-\alpha) \cdot Q^{\pi}_k(x_t, u_t) + \alpha \cdot (r_{t+1} + \gamma Q^{\pi}_k(x_{t+1}, u_{t+1}))$
end

Do this if you want to improve later

2.2 Value function, Q-learning etc

Model free learning

- Necessary assumptions for convergence:
  - **All states and actions** have a non-zero probability of **being visited**. Problem if we chose a greedy policy, we need to explore other actions (and states) too!
  - The learning rate is decreasing
  - We learn an infinite amount of time

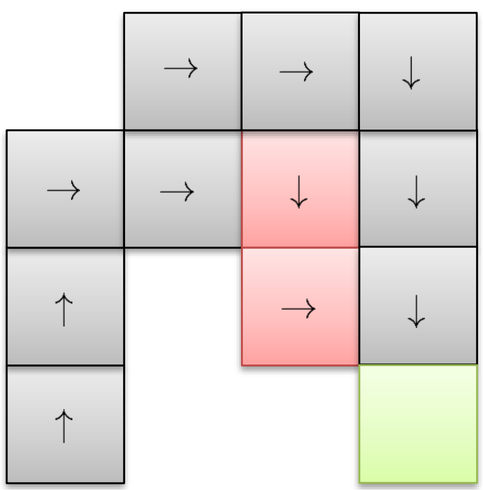
$$\sum_{k=0}^{\infty} \alpha_k = \infty \; ; \; \sum_{k=0}^{\infty} \alpha_k^2 < \infty$$

- In practice not as bad, the assumptions can be relaxed and results still be good.

2.2 Value function, Q-learning etc

Model free learning

- Necessary assumptions for convergence:
  - **All states and actions** have a non-zero probability of **being visited**. Problem if we chose a greedy policy, we need to explore other actions (and states) too!



Policy evaluation

Iterate over all data tuples $(x_t, u_t, r_{t+1}, x_{t+1})$:

$$Q^{\pi}_{k+1}(x_t, u_t) = (1-\alpha)\cdot Q^{\pi}_{k}(x_t, u_t) + \alpha\cdot(r_{t+1} + \gamma Q^{\pi}_{k}(x_{t+1}, u_{t+1}))$$

end

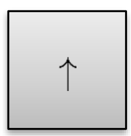
Policy improvement

$$u_{\text{greedy}}(x) = \arg\max_{u} Q(x, u)$$

2.2 Value function, Q-learning etc

Model free learning

- How to handle the assumptions:
  - Do greedy update, but give all other actions a small probability too.
    $\pi(u_{greed}|x_t) = 1 - \epsilon$ all other actions share probability ϵ (chose e.g. 0.1).
    This is called ϵ **-greedy** policy.



$$\pi(x, \uparrow) = 1 - \epsilon \ , \quad \pi(x, \leftarrow) = \pi(x, \downarrow) = \pi(x, \rightarrow) = \frac{\epsilon}{3}$$

  - The learning rate **can be** reduced during training, but sometimes keeping it constant is enough. It‘s like with learning rates for NN.
  - As we saw for generalized policy iteration, we don‘t need full convergence of the value function anyway to do an update, so we just **stop at some point**.

2.2 Value function, Q-learning etc

Combining it all: Q-Learning

- Learning without a model. Does policy improvement and evaluation in one step. Also need exploration, ϵ-greedy is common.

Policy evaluation

Iterate over all data tuples $(x_t, u_t, r_{t+1}, x_{t+1})$:

$$Q^{\pi}_{k+1}(x_t, u_t) = (1-\alpha)\cdot Q^{\pi}_k(x_t, u_t) + \alpha \cdot (r_{t+1} + \gamma Q^{\pi}_k(x_{t+1}, u_{t+1}))$$

end

Policy improvement

$$u_{\text{greedy}}(x) = \arg\max_u Q(x, u)$$

Q-learning

$$Q_{k+1}(x_t, u_t) = (1-\alpha)\cdot Q_k(x_t, u_t) + \alpha\left(r_{t+1} + \max_u Q_k(x_{t+1}, u)\right)$$

2.2 Value function, Q-learning etc

Combining it all: Q-Learning

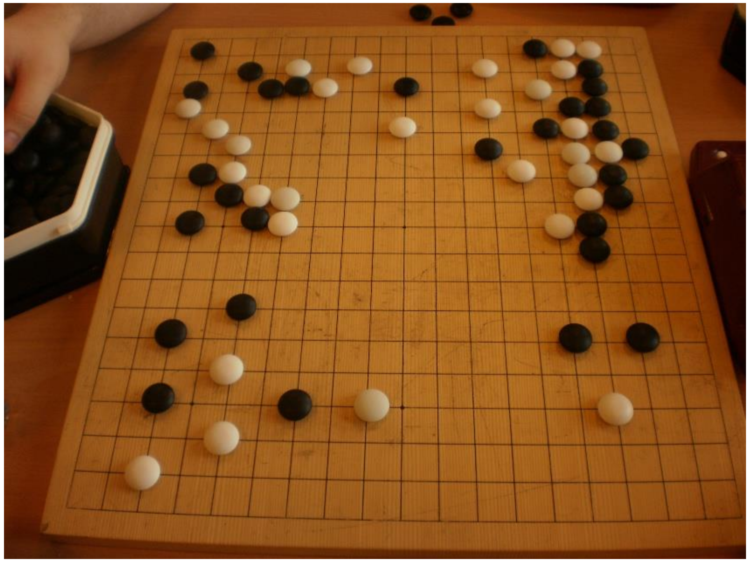
- Learning without a model. Does policy improvement and evaluation in one step. Also need exploration, ϵ-greedy is common.

$$Q_{k+1}(x_t, u_t) = (1-\alpha) \cdot Q_k(x_t, u_t) + \alpha \left(r_{t+1} + \max_u Q_k(x_{t+1}, u) \right)$$

```
initialise a table of Q values (e.g. random)
provide epsilon, alpha, x0, x_end
for i = 1:N_episodes
    x = x0
    while x!=x_end
        u = eps_greedy(x, epsilon)
        # Interaction with the environment, take action a
        # receive next state x2 and reward r
        Q(x, u) = (1-alpha)*Q(x, u) + alpha*(r + max_u2(Q(x2, u2)))
        x = x2
    end
end
```

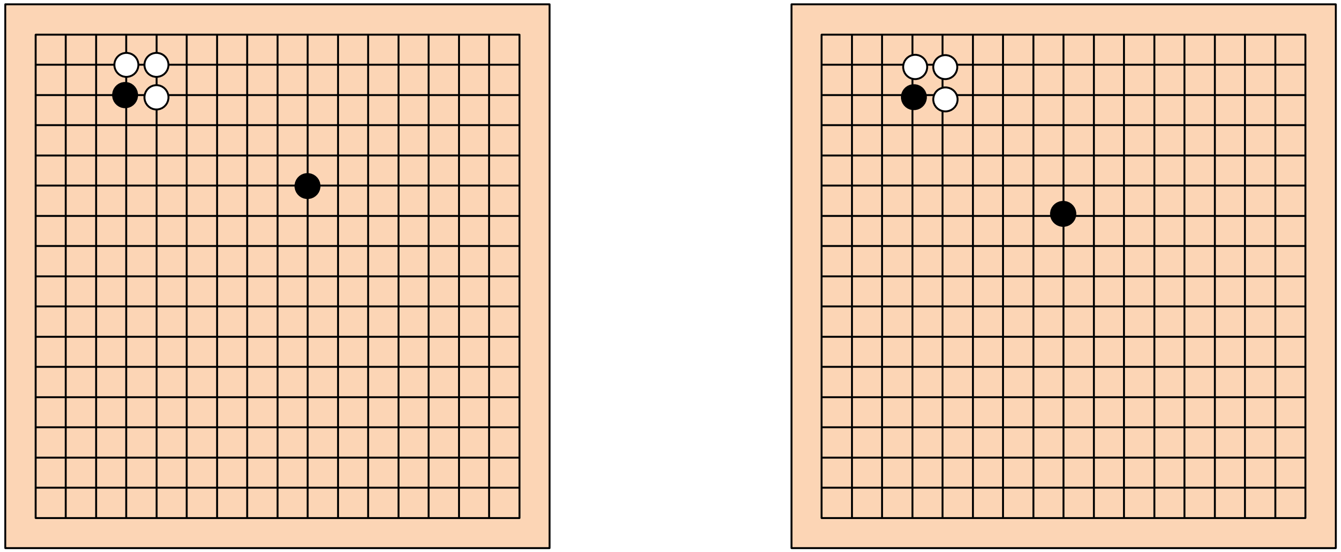
2.2 Value function, Q-learning etc

Example of discrete MDP's



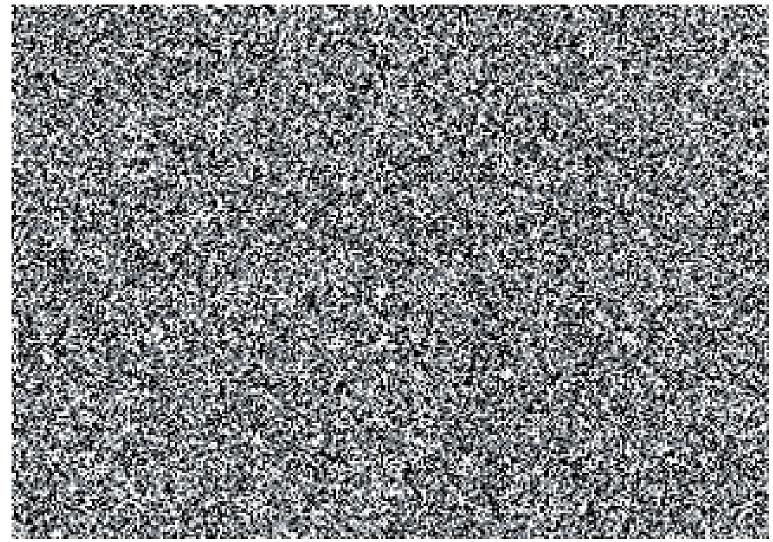
Number of states $> 2 \cdot 10^{170}$!
How much RAM do you have?

2.2 Value function, Q-learning etc



Do we really need to save one float for every state and action? Some states can be very similar! **Generalize over large state and action spaces!**

2.2 Value function, Q-learning etc



Also **some states** are simply **irrelevant** as they have very low or zero probability!

Couple these methods with (Deep) NN -> Find Q-function and/or policy on relevant states only and learn to generalize!

2.2 Value function, Q-learning etc

Wrap up

1. Learn value function or action-value function of current policy using the Bellman equation
2. Use 1. to improve.
3. Repeat.

- Learning the value function or action-value function requires to visit all states →deterministic policy problematic →epsilon-greedy
- No need to learn the value function to full convergence, can do update step earlier
- **Q-learning** can be used to learn the optimal greedy policy using state transitions from any policy → algorithm of choice in discrete MDPs

2.2 Value function, Q-learning etc

What I expect you to know for the exam from chapter 2.2

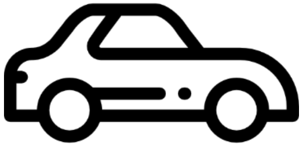
1. The Bellman equation
2. How to compute the value function in discrete MDP’s
3. How to compute the action-value function in discrete MDP’s
4. How to get a new greedy policy given a value or action-value function.
5. Calculate a Q-learning update step.
6. Understand why a deterministic policy in a deterministic environment does not work for learning.

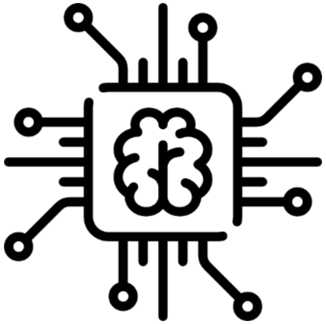
Reinforcement Learning

Johannes Betz / Prof. Dr. Markus Lienkamp / Prof. Dr. Boris Lohmann
(Christian Dengler, M. Sc.)

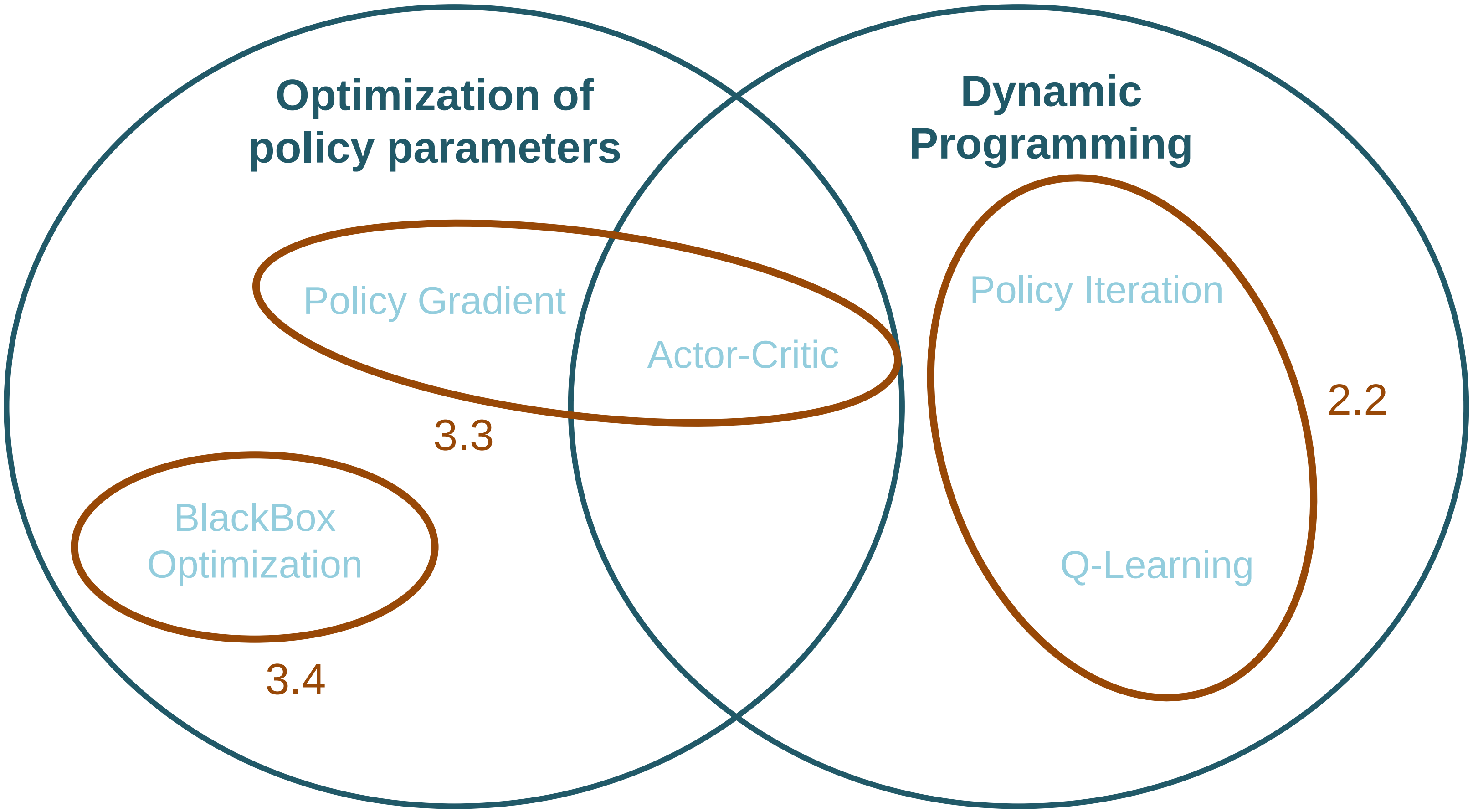
Agenda

1. Terminology and Concept
    - 1.1 Terminology and problem definition
    - 1.2 Motivation for RL in engineering
2. RL in discrete state- and action-spaces
    - 2.1 Markov decision processes
    - 2.2 Value-Function, Q-learning etc.
3. RL in continuous state- and action-spaces
    - **3.1 Overview of methods**
    - 3.2 Connection Optimal Control
    - 3.3 Exploration in the action space
    - 3.4 Exploration in parameter space (Optional)





3.1 Overview of methods



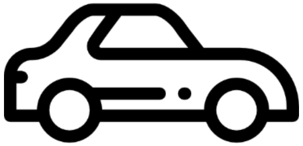
Inspired by „Schulman, J.: Optimizing expectations: from deep reinforcement learning to stochatic computation graphs.“

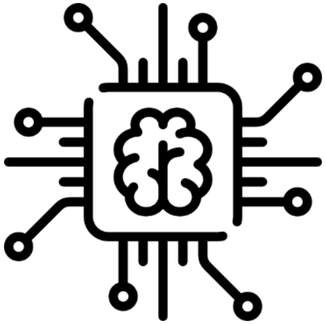
Reinforcement Learning

Johannes Betz / Prof. Dr. Markus Lienkamp / Prof. Dr. Boris Lohmann
(Christian Dengler, M. Sc.)

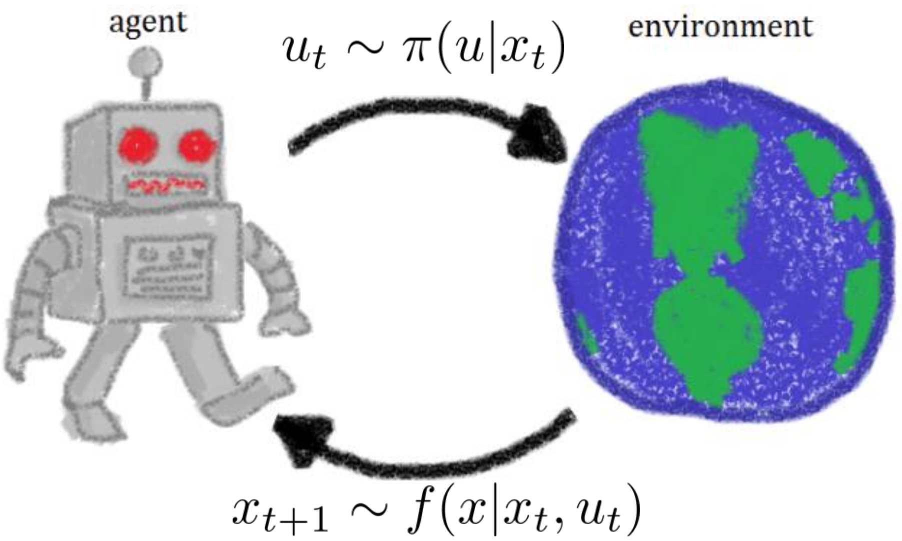
Agenda

1. Terminology and Concept
   - 1.1 Terminology and problem definition
   - 1.2 Motivation for RL in engineering
2. RL in discrete state- and action-spaces
   - 2.1 Markov decision processes
   - 2.2 Value-Function, Q-learning etc.
3. RL in continuous state- and action-spaces
   - 3.1 Overview of methods
   - **3.2 Connection to Optimal Control**
   - 3.3 Exploration in the action space
   - 3.4 Exploration in parameter space (Optional)





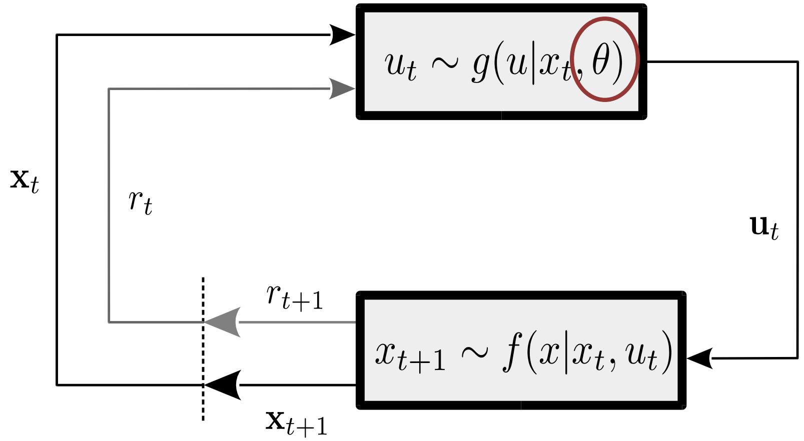
3.1 Connection to Optimal Control



- **Legend**:
  - State: x
  - Action: u
  - Policy: $\pi(u, x)$
  - Behavior of the environment: $f(x_{t+1}, x_t, u)$

- **Changes**:
  - $\pi(u, x)$ and $f(x_{t+1}, x_t, u)$ are continuous probability distributions.

3.1 Connection to Optimal Control



Find the **best parameters** for the policy/control law.

Optimization Problem:

$$\boldsymbol{\theta}^* = \arg\max_{\theta} \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(\mathbf{x}_t, \mathbf{u}_t) | \mathbf{x}_0 \sim d_0(\mathbf{x})\right]$$

$$s.t.\ \mathbf{x}_{t+1} \sim f(\mathbf{x}|\ \mathbf{x}_t, \mathbf{u}_t)$$

$$\mathbf{u}_t \sim \pi(\mathbf{u}|\ \mathbf{x}_t, \boldsymbol{\theta})$$

$$0 < \gamma < 1$$

3.1 Connection to Optimal Control

Optimal Control, Linear Case, LQR control

- Model description

$$x_{t+1} = Ax_t + Bu_t$$

- Optimization Problem (analytical solution: see Wikipedia)

$$F^* = \arg\min_F \sum_{t=0}^{\infty} \left(x_t^T Q x_t + u_t^T R u_t \right)$$

$$s.t.\ x_{t+1} = Ax_t + Bu_t$$

$$u_t = Fx_t$$

- Same problem, minimizing cost instead of maximizing reward. Only for linear model and quadratic costs. Solution independent of starting state.

3.1 Connection to Optimal Control

Nonlinear Case: Model Predictive Control

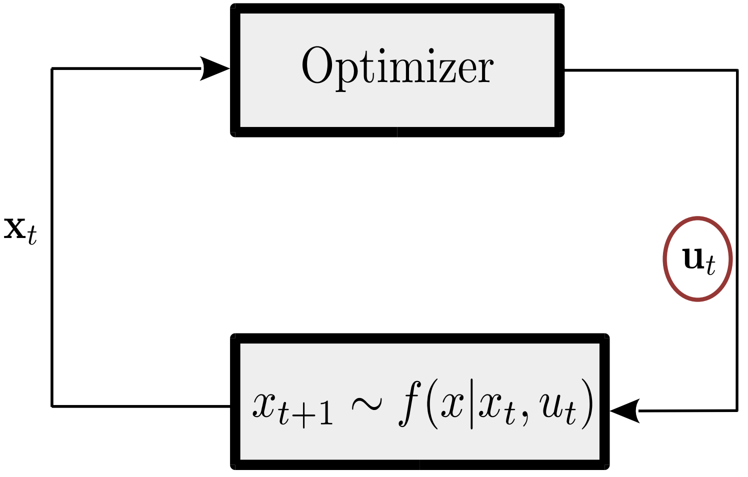
- Model description:

$$x_{t+1} = f(x_t, u_t)$$

- Optimization Problem solved at each t:

$$U = u_0, u_1, \dots, u_T$$
$$X = x_0, x_1, \dots, x_T$$

$$U = \arg\min_U \sum_{t=0}^{T} l(x_t, u_t) + V(x_T)$$
$$s.t. \; x_{t+1} = f(x_t, u_t)$$
$$g(X, U) \leq 0$$



3.1 Connection to Optimal Control

Wrap up

- Optimal Control ⊂ Reinforcement Learning

Control people	Machine Learning people
Minimize costs	Maximize rewards
Optimize the control inputs each time. (Unless the model is linear)	Optimize the policy/control law parameters.
Uncertainties in nonlinear MPC problematic.	Can deal somewhat with uncertainty.
Usual background in electical or mechanical engineering	Usual background in computer science, informatics, mathematics.

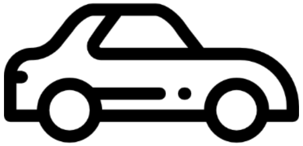
Reinforcement Learning

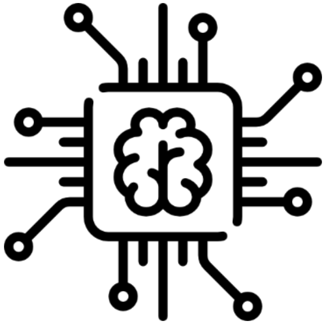
Johannes Betz / Prof. Dr. Markus Lienkamp / Prof. Dr. Boris Lohmann

(Christian Dengler, M. Sc.)

Agenda

1. Terminology and Concept
    - 1.1 Terminology and problem definition
    - 1.2 Motivation for RL in engineering
2. RL in discrete state- and action-spaces
    - 2.1 Markov decision processes
    - 2.2 Value-Function, Q-learning etc.
3. RL in continuous state- and action-spaces
    - 3.1 Overview of methods
    - 3.2 Connection to Optimal Control
    - **3.3 Exploration in the action space**
    - 3.4 Exploration in parameter space (Optional)





3.2 Exploration in action space

- Policy based reinforcement learning is an **optimization problem**.
- You should know by now, that we can optimize parameters with respect to a cost/reward function if we can get the **gradient** (or atleast a stochastic version of it).

Problem:

$$\boldsymbol{\theta}^* = \arg\max_{\theta} \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(\mathbf{x}_t, \mathbf{u}_t) | \mathbf{x}_0 \sim d_0(\mathbf{x})\right]$$

$$s.t. \ \ldots$$

Gradient?:

$$J(\boldsymbol{\theta}) := \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(\mathbf{x}_t, \mathbf{u}_t) | \mathbf{x}_0 \sim d_0(\mathbf{x})\right]$$

$$\nabla_\theta J(\boldsymbol{\theta}) = \ldots$$

3.2 Exploration in action space

Policy Gradient, basics

- Expected Value:

$$\mathbb{E}_{x \sim p(x)}[f(\mathbf{x})] = \int p(\mathbf{x}) f(\mathbf{x}) d\mathbf{x}$$

- Usefull Identity:

$$\nabla_\theta \log(p(\theta)) = \frac{1}{p(\theta)} \nabla_\theta p(\theta)$$

$$\Leftrightarrow \nabla_\theta p(\theta) = p(\theta) \nabla_\theta \log(p(\theta))$$

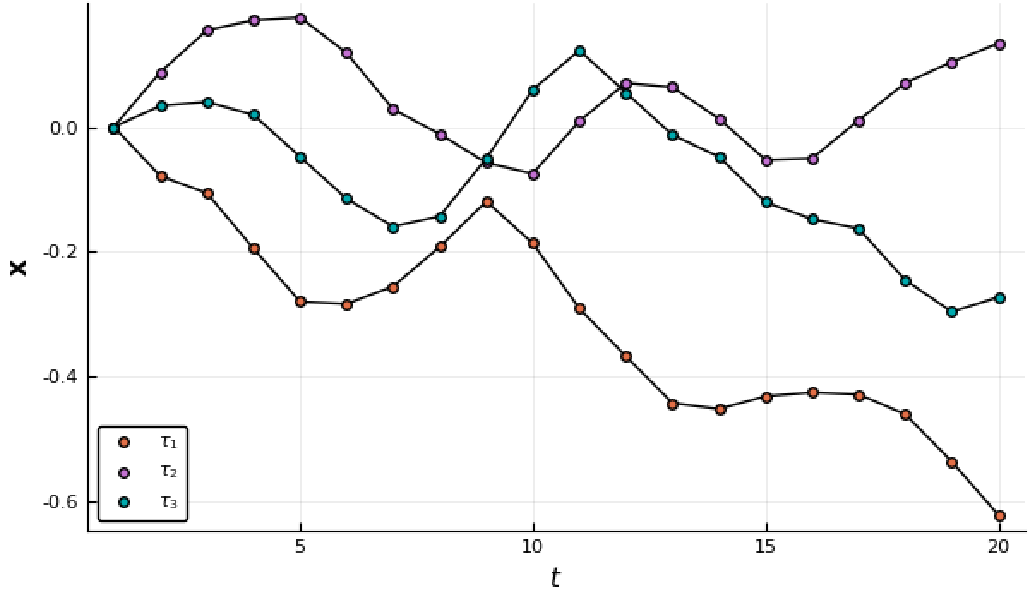
- Log property: $\log(a \cdot b) = \log(a) + \log(b)$

3.2 Exploration in action space

Policy Gradient

- Let $\boldsymbol{\tau}$ be the random variable describing a trajector. We can sample from it through simulation

$$\boldsymbol{\tau} = \left[\mathbf{x}_0, \mathbf{u}_0, \mathbf{x}_1, \mathbf{u}_1, \ldots\right]$$



3.2 Exploration in action space

Policy Gradient

- Sampled future reward, short notation

$$\sum_{t=0}^{\infty} \gamma^t r(\mathbf{x}_t, \mathbf{u}_t) := r(\boldsymbol{\tau})$$

- Objective Function $J(\boldsymbol{\theta}) := \mathbb{E}_{\tau \sim p_\theta(\tau)}\left[r(\boldsymbol{\tau})\right]$

$$= \int p_\theta(\boldsymbol{\tau}) r(\boldsymbol{\tau}) d\boldsymbol{\tau}$$

$$\nabla_\theta J(\boldsymbol{\theta}) = \int \nabla_\theta p_\theta(\boldsymbol{\tau}) r(\boldsymbol{\tau}) d\boldsymbol{\tau}$$

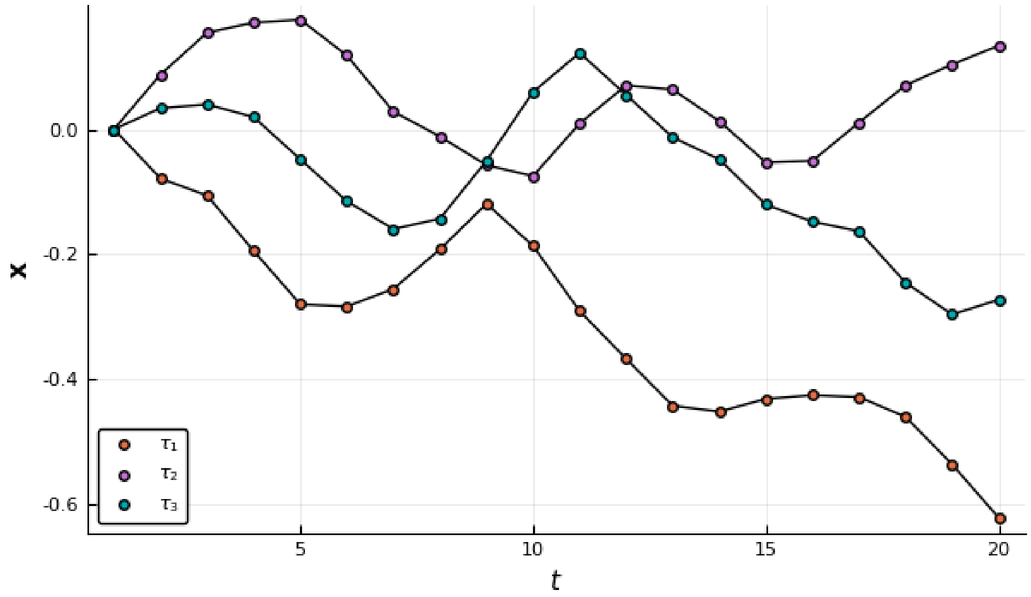
Inspired by UC Berkley: http://rail.eecs.berkeley.edu/deeprlcourse/index.html

3.2 Exploration in action space

Policy Gradient

$$J(\boldsymbol{\theta}) := \mathbb{E}_{\tau \sim p_\theta(\tau)}\left[r(\boldsymbol{\tau})\right] = \int p_\theta(\boldsymbol{\tau}) r(\boldsymbol{\tau}) d\boldsymbol{\tau}$$

$$\nabla_\theta J(\boldsymbol{\theta}) = \int \nabla_\theta p_\theta(\boldsymbol{\tau}) r(\boldsymbol{\tau}) d\boldsymbol{\tau}$$



Inspired by UC Berkley: http://rail.eecs.berkeley.edu/deeprlcourse/index.html

3.2 Exploration in action space

Policy Gradient

$$\nabla_\theta J(\boldsymbol{\theta}) = \int \nabla_\theta p_\theta(\boldsymbol{\tau}) r(\boldsymbol{\tau}) d\boldsymbol{\tau}$$
$$= \int p_\theta(\boldsymbol{\tau}) \nabla_\theta \log(p_\theta(\boldsymbol{\tau})) r(\boldsymbol{\tau}) d\boldsymbol{\tau}$$
$$= \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\nabla_\theta \log(p_\theta(\boldsymbol{\tau})) r(\boldsymbol{\tau}) \right]$$

- We can build this expectation by sampling.
  But what is $\nabla_\theta \log(p_\theta(\boldsymbol{\tau}))$?

3.2 Exploration in action space

Policy Gradient

$$p_\theta(\boldsymbol{\tau}) = p_\theta(\mathbf{x}_0, \mathbf{u}_0, \mathbf{x}_1, \mathbf{u}_1, \ldots)$$

$$p_\theta(\boldsymbol{\tau}) = p(\mathbf{x}_0) \prod_{t=1}^{\infty} \pi_\theta(\mathbf{u}_t|\mathbf{x}_t) p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)$$

$$\log(p_\theta(\boldsymbol{\tau})) = \cancel{\log(p(\mathbf{x}_0))} + \sum_{t=1}^{\infty} \log(\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)) + \cancel{\log(p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t))}$$

$$\nabla_\theta \log(p_\theta(\boldsymbol{\tau})) = \nabla_\theta \sum_{t=1}^{\infty} \log(\pi_\theta(\mathbf{u}_t|\mathbf{x}_t))$$

3.2 Exploration in action space

Policy Gradient

- Policy Gradient:

$$\nabla_\theta J(\boldsymbol{\theta}) = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[r(\boldsymbol{\tau}) \sum_{t=0}^{\infty} \nabla_\theta \log(\pi_\theta(u_t | x_t)) \right]$$

- However we can‘t sample an infinitely large random variable τ.

3.2 Exploration in action space

Policy Gradient

- **Possibility 1**: sample only first T states.

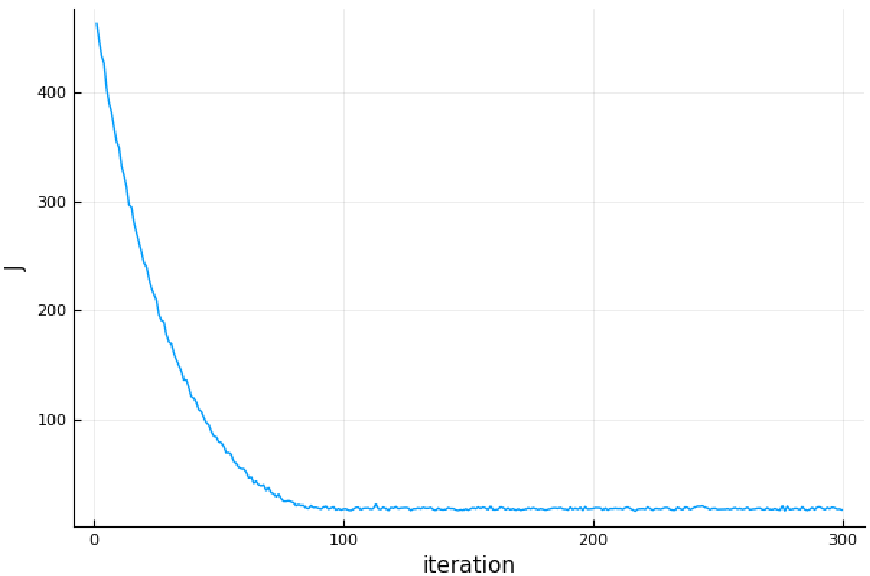
"Reinforce" algorithm:

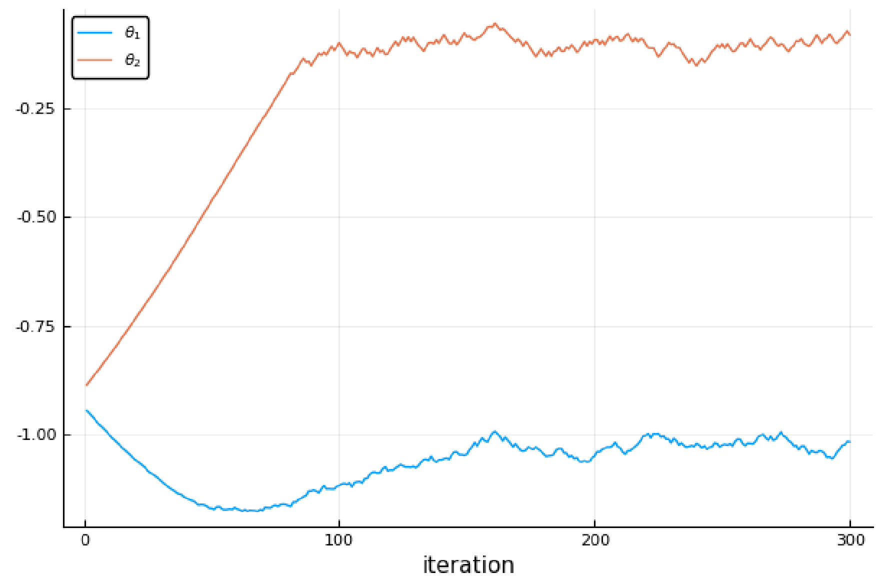
1. **Initialise π_θ randomly or make a nice guess**
2. **Simulate N trajectories $\tau_{(i=1:N)} = [x_0, u_0, r_1, \ldots r_{T-1}, x_T, u_T]_i$**
3. **Calculate gradient $\nabla_\theta J = \frac{1}{N} \sum_{i=1}^{N} \left(\sum_t \nabla \log \pi_\theta(u_{i,t} | x_{i,t}) \cdot \sum_t r_{i,t} \right)$**
4. **Update parameters $\boldsymbol{\theta} = \boldsymbol{\theta} + \alpha \nabla_\theta J$**
5. **Unless converged/max iters done, goto 2**

3.2 Exploration in action space

Reinforce

- Model: $x_{t+1} = 1.1 \cdot \mathbb{N}(x_t, 0.1^2) + 0.1 + u_t;\ x_0 \sim \mathbb{N}(0, 2^2)$
- Cost: $c(x,u) = 10x^2 + u^2$
- Hyperparameters: $\alpha = 0.1,\ N = 50, T = 50,\ u_t \sim \mathbb{N}(\theta_1 x_t + \theta_2, 0.1)$





3.2 Exploration in action space

Wrap up

- We can get a **stochastic gradient** of the cost function with respect to the control parameters once we have multiple trajectories.
- We use the gradient for **gradient descent/ascent** = improvement of the policy. Gradient descent when minimizing costs and ascend when maximizing reward.

3.2 Exploration in action space

What I expect you to know for the exam from chapter 3.2

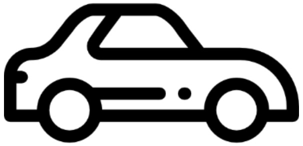
1. Know the steps of the Reinforce algorithm
2. Be able to write down the expression for the policy gradient.
3. Know that the gradient can have very high variance, which can lead to convergence issues.

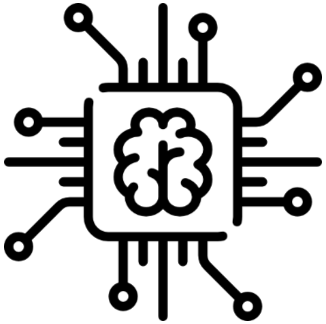
Reinforcement Learning

Johannes Betz / Prof. Dr. Markus Lienkamp / Prof. Dr. Boris Lohmann
(Christian Dengler, M. Sc.)

Agenda

1. Terminology and Concept
    - 1.1 Terminology and problem definition
    - 1.2 Motivation for RL in engineering
2. RL in discrete state- and action-spaces
    - 2.1 Markov decision processes
    - 2.2 Value-Function, Q-learning etc.
3. RL in continuous state- and action-spaces
    - 3.1 Overview of methods
    - 3.2 Connection to Optimal Control
    - 3.3 Exploration in the action space
    - **3.4 Exploration in parameter space (Optional)**





3.4 Exploration in parameter space

Motivation

- Gradient based optimization is efficient if the **gradient** can be evaluated **fast and accurately**, e.g. analytically, in RL this is not the case.
- As we need **multiple samples** $r(\boldsymbol{\tau})$ to estimate one gradient, why not just try different parameters immediately and combine results?
- So far: we have a probability **distribution over actions**, and find the gradient to chose the better actions more often.
- Now: we have a probability **distribution over parameters**, and try to increase the probability of good parameters.

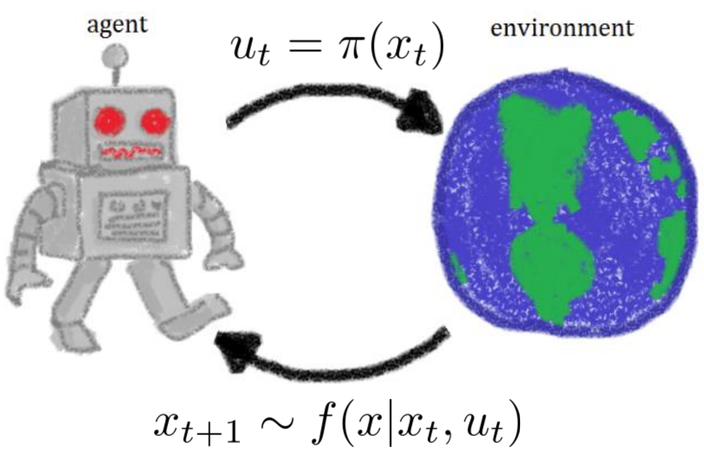
3.4 Exploration in parameter space

Motivation

- We want $r(\boldsymbol{\tau})$ to have low variance → deterministic policy
- No need to learn a Value-Function, continuous-time models possible
- Optimization Problem

$$\boldsymbol{\theta}^* = \arg\max_{\theta} \mathbb{E}\left[\sum_{t=0}^{T} r(\mathbf{x}_t, \mathbf{u}_t) | \mathbf{x}_0 \sim d_0(\mathbf{x})\right]$$

$$\begin{aligned} s.t.\ & \mathbf{x}_{t+1} \sim f(\mathbf{x}|\ \mathbf{x}_t, \mathbf{u}_t) \\ & \mathbf{u}_t = \pi(\mathbf{x}_t, \boldsymbol{\theta}) \\ & 0 < \gamma < 1 \end{aligned}$$



3.4 Exploration in parameter space

Simple ES

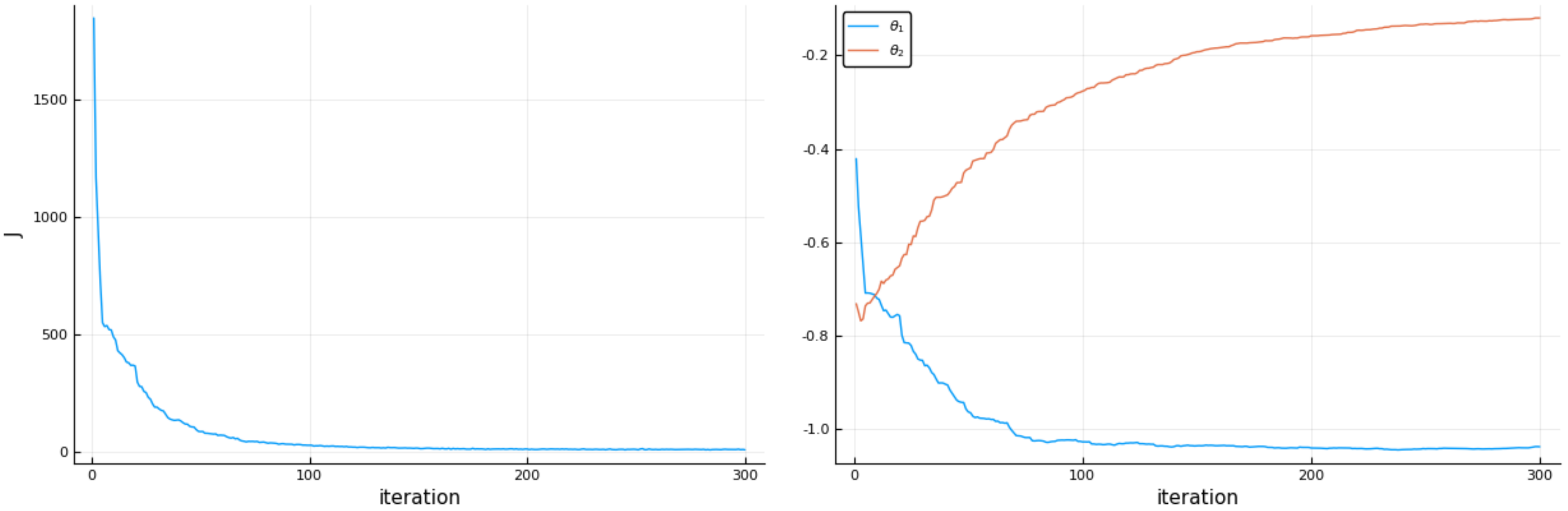
- ~ Random finite differences. Also called *simultaneous perturbation stochastic approximation*, *parameter exploring policy gradient* and others

1. **Initialise π_θ randomly or make a nice guess**
2. **Sample i=1:N random perturbations and sample $r(\boldsymbol{\tau}_{\theta+\epsilon_i})$**
3. **Calculate update $\Delta\boldsymbol{\theta} = \sum_{i=1}^{N} \boldsymbol{\epsilon}_i r(\boldsymbol{\tau}_{\theta+\epsilon_i})$**
4. **Update parameters $\boldsymbol{\theta} = \boldsymbol{\theta} + \alpha\Delta\boldsymbol{\theta}$**
5. **Unless converged/max iters done, goto 2**

Salimans, T.; Ho, J.; Chen, X. & Sutskever, I.
Evolution strategies as a scalable alternative to reinforcement learning
arXiv preprint arXiv:1703.03864, **2017**

3.4 Exploration in parameter space

Simple ES



$$x_{t+1} = 1.1 \cdot \mathbb{N}(x_t, 0.1^2) + 0.1 + u_t;\ \ x_0 \sim \mathbb{N}(0, 2^2);\ \ u_t = \theta_1 x_t + \theta_2$$

- Its still gradient descent! Gradient is built by perturbing parameters this time.

3.4 Exploration in parameter space

Simple ES

- **Problems** with steep gradient/flat regions and high variance **not gone**. Practical application requires some more tweaks, e.g. normalization or fitness shaping, mirroring of the perturbation
- Pros:
  - Only need to store one Float per trajectory, instead of all the states and actions.
  - Easy and efficient to parallelize as simulations are independent.
  - No need to learn a value function.
- Cons:
  - In general more trajectories needed as parameter space is usally much larger than action space →requires more exploration

Salimans, T.; Ho, J.; Chen, X. & Sutskever, I.
Evolution strategies as a scalable alternative to reinforcement learning
arXiv preprint arXiv:1703.03864, **2017**

3.4 Exploration in parameter space

CMA-ES

- Incorporates second order information of the **parameter distribution** → Storage of the covariance matrix.

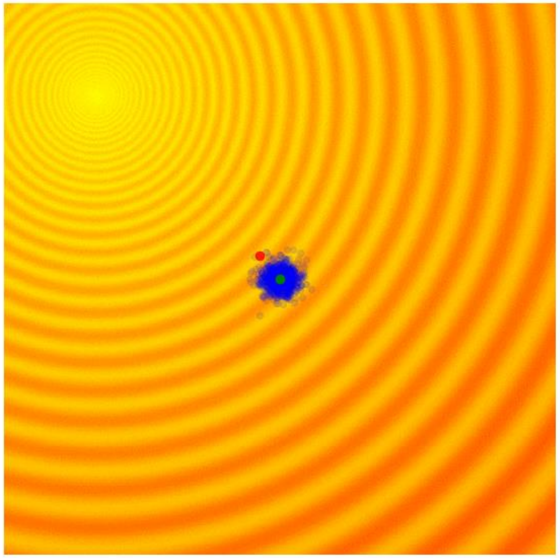
- Unsuited for deep neural networks because of the memory and computation scaling of $\mathcal{O}(N^2)$ where N is the number of parameters. Several researchers adress this though.

$$\boldsymbol{\theta} \sim \mathcal{N}(\mathbf{m}, \mathbf{C})$$
$$\boldsymbol{\theta} \in R^N,\ \mathbf{m} \in R^N,\ \mathbf{C} \in R^{N \times N}$$

Hansen, N.
The CMA Evolution Strategy: A Tutorial
CoRR, **2016**, *abs/1604.00772*

3.4 Exploration in parameter space

CMA-ES



Full algorithm can be found in:

Hansen, N.
The CMA Evolution Strategy: A Tutorial
CoRR, **2016**, *abs/1604.00772*

The steps of the algorithm are:

1. **Initialise π_θ randomly or make a nice guess**
2. **Sample i=1:N parameters $\boldsymbol{\theta}_i \sim \mathcal{N}(\mathbf{m}, \mathbf{C})$ and sample $r(\boldsymbol{\tau}_{\theta_i})$**
3. **Update the mean m using the best returns**
4. **Update a running average of C using the best returns**
5. **Unless converged/max iters done, goto 2**

GIF from: http://blog.otoro.net/2017/10/29/visual-evolution-strategies/

3.4 Exploration in parameter space

Other black-box optimization algorithms

- Natural Evolution Strategies

Wierstra, D.; Schaul, T.; Peters, J. & Schmidhuber, J.
Natural evolution strategies
Evolutionary Computation, 2008. CEC 2008.(IEEE World Congress on Computational Intelligence). IEEE Congress on, **2008**, 3381-3387

- Neuroevolution

Stanley, K. O. & Miikkulainen, R.
Evolving Neural Networks through Augmenting Topologies
Evolutionary Computation, **2002***, 10*, 99-127

- Differential Evolution

Das, S. & Suganthan, P. N.
Differential evolution: a survey of the state-of-the-art
IEEE transactions on evolutionary computation, IEEE, **2011***, 15*, 4-31

Evaluation

Source: https://splinternews.com/12-years-of-steve-jobs-saying-one-more-thing-like-hes-c-1793850336

Evaluation

- In this lecture we are doing in regularly evaluation of each lecture
- We want **your** feedback for every **individual** lecture
- We evaluate the lecture each week
- We give feedback based on the evaluation the week after

Evaluation – Step by Step

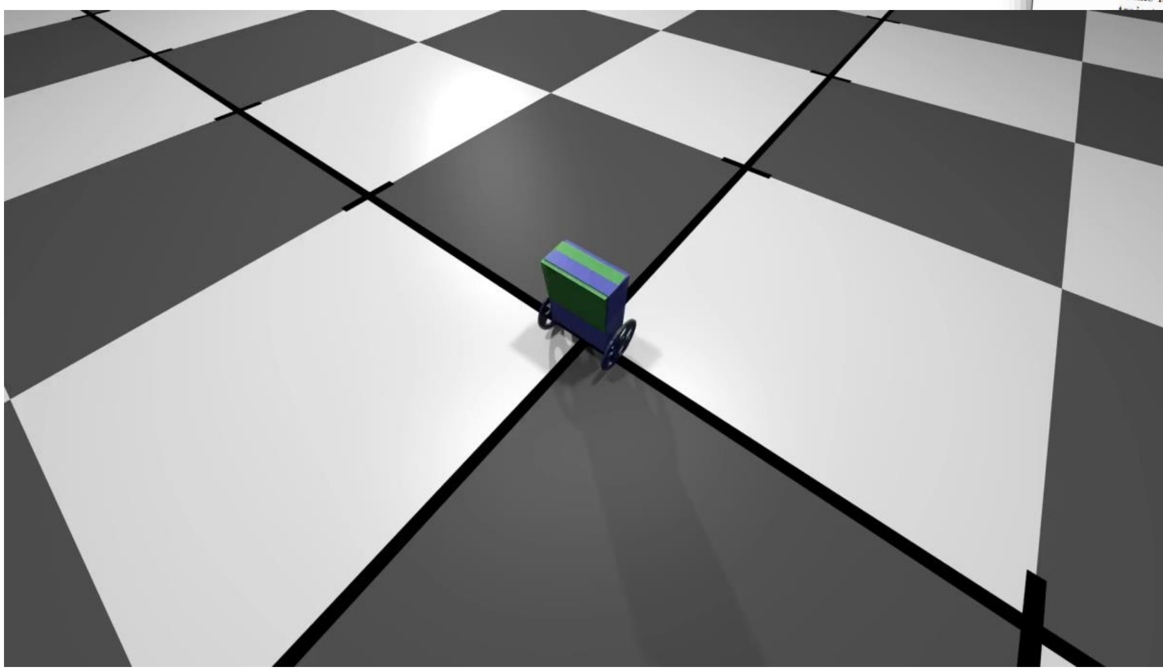
1. Get out your smartphones
2. Open an app for QR-code Reading
3. Read the following QR-code on the right side
4. Open the website
5. Answer the questions
6. Send the evaluation

OR

1. Open the following website in your browser:
   https://evasys.zv.tum.de/evasys/online.php?p=AIAT-11
2. Answer the questions
3. Send the evaluation

RL at the chair of automatic control

If you would love to get deeper into RL:
Master theses will be available starting end of april 2019.
Contact: **c.dengler@tum.de**



Video created by **Nikolas Wilhelm** during his master thesis