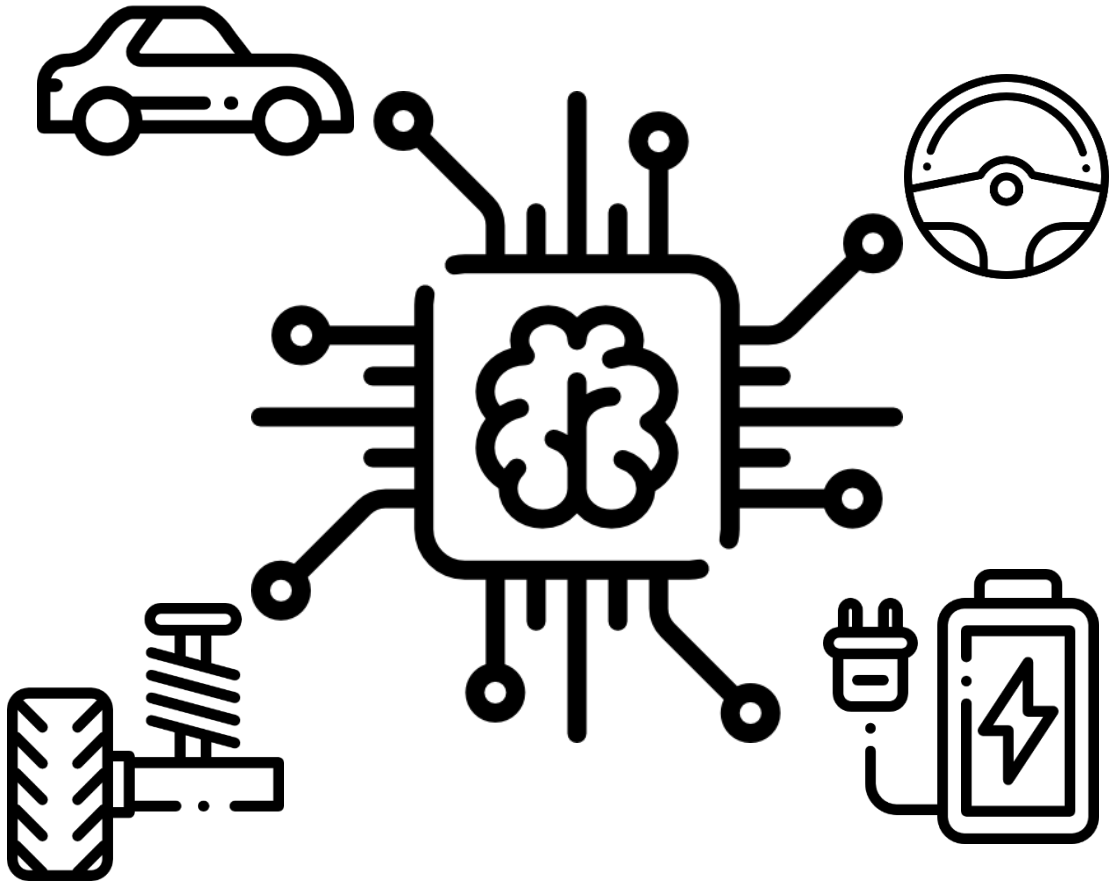
Lehrstuhl für Fahrzeugtechnik
Fakultät für Maschinenwesen
Technische Universität München

Artificial Intelligence in Automotive Technology

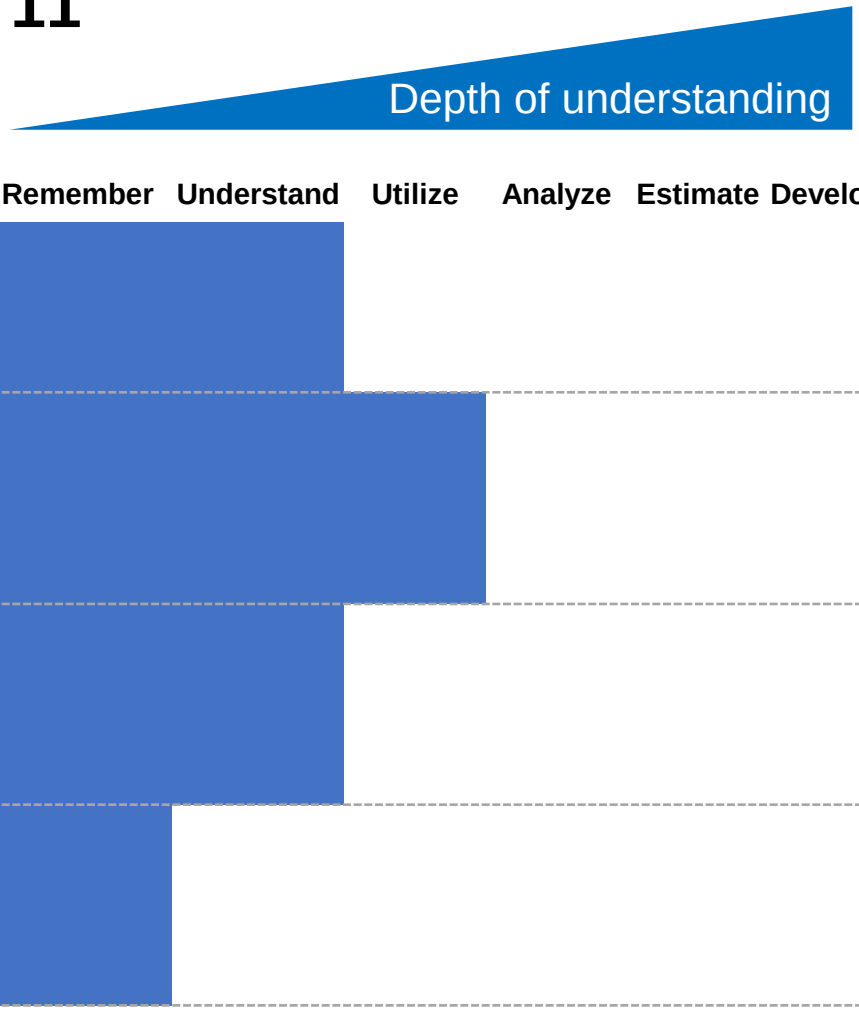
Johannes Betz / Prof. Dr.-Ing. Markus Lienkamp / Prof. Dr.-Ing. Boris Lohmann



Lecture Overview

Lecture 16:15 – 17:45	Practice 17:45 – 18:30
1 Introduction: Artificial Intelligence 17.10.2019 – Johannes Betz	**Practice 1** 17.10.2019 – Johannes Betz
2 Perception 24.10.2019 – Johannes Betz	**Practice 2** 24.10.2019 – Johannes Betz
3 Supervised Learning: Regression 31.10.2019 – Alexander Wischnewski	**Practice 3** 31.10.2019 – Alexander Wischnewski
4 Supervised Learning: Classification 7.11.2019 – Jan Cedric Mertens	**Practice 4** 7.11.2019 – Jan Cedric Mertens
5 Unsupervised Learning: Clustering 14.11.2019 – Jan Cedric Mertens	**Practice 5** 14.11.2019 – Jan Cedric Mertens
6 Pathfinding: From British Museum to A* 21.11.2019 – Lennart Adenaw	**Practice 6** 21.11.2019 – Lennart Adenaw
7 Introduction: Artificial Neural Networks 28.11.2019 – Lennart Adenaw	**Practice 7** 28.11.2019 – Lennart Adenaw
8 Deep Neural Networks 5.12.2019 – Jean-Michael Georg	**Practice 8** 5.12.2019 – Jean-Michael Georg
9 Convolutional Neural Networks 12.12.2019 – Jean-Michael Georg	**Practice 9** 12.12.2019 – Jean-Michael Georg
10 Recurrent Neural Networks 19.12.2019 – Christian Dengler	**Practice 10** 19.12.2019 – Christian Dengler
11 Reinforcement Learning 09.01.2020 – Christian Dengler	**Practice 11** 09.01.2020 – Christian Dengler
12 AI-Development 16.01.2020 – Johannes Betz	**Practice 12** 16.01.2020 – Johannes Betz
13 Guest Lecture: VW Data:Lab 23.01.2020 –	

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.

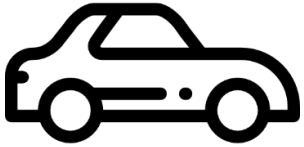
Reinforcement Learning

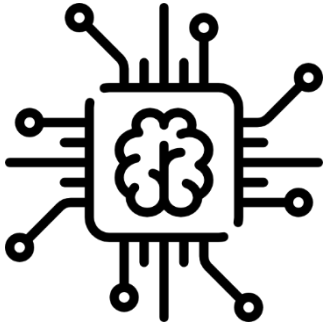
Johannes Betz / Prof. Dr. Markus Lienkamp / Prof. Dr. Boris Lohmann
(Christian Dengler, M. Sc.)

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.





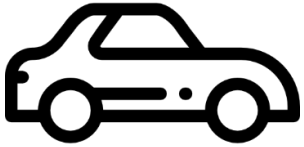
Reinforcement Learning

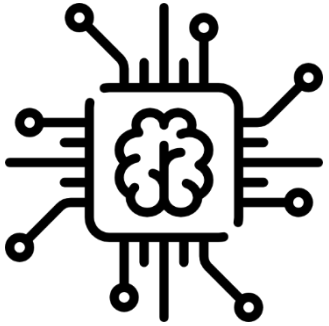
Johannes Betz / Prof. Dr. Markus Lienkamp / Prof. Dr. Boris Lohmann
(Christian Dengler, M. Sc.)

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.





1.1 Terminology and problem definition

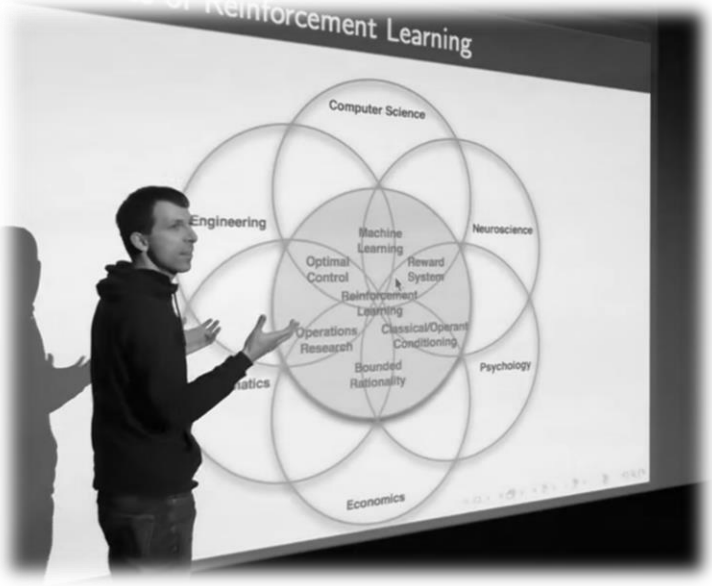
Revision

- 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
  - ?

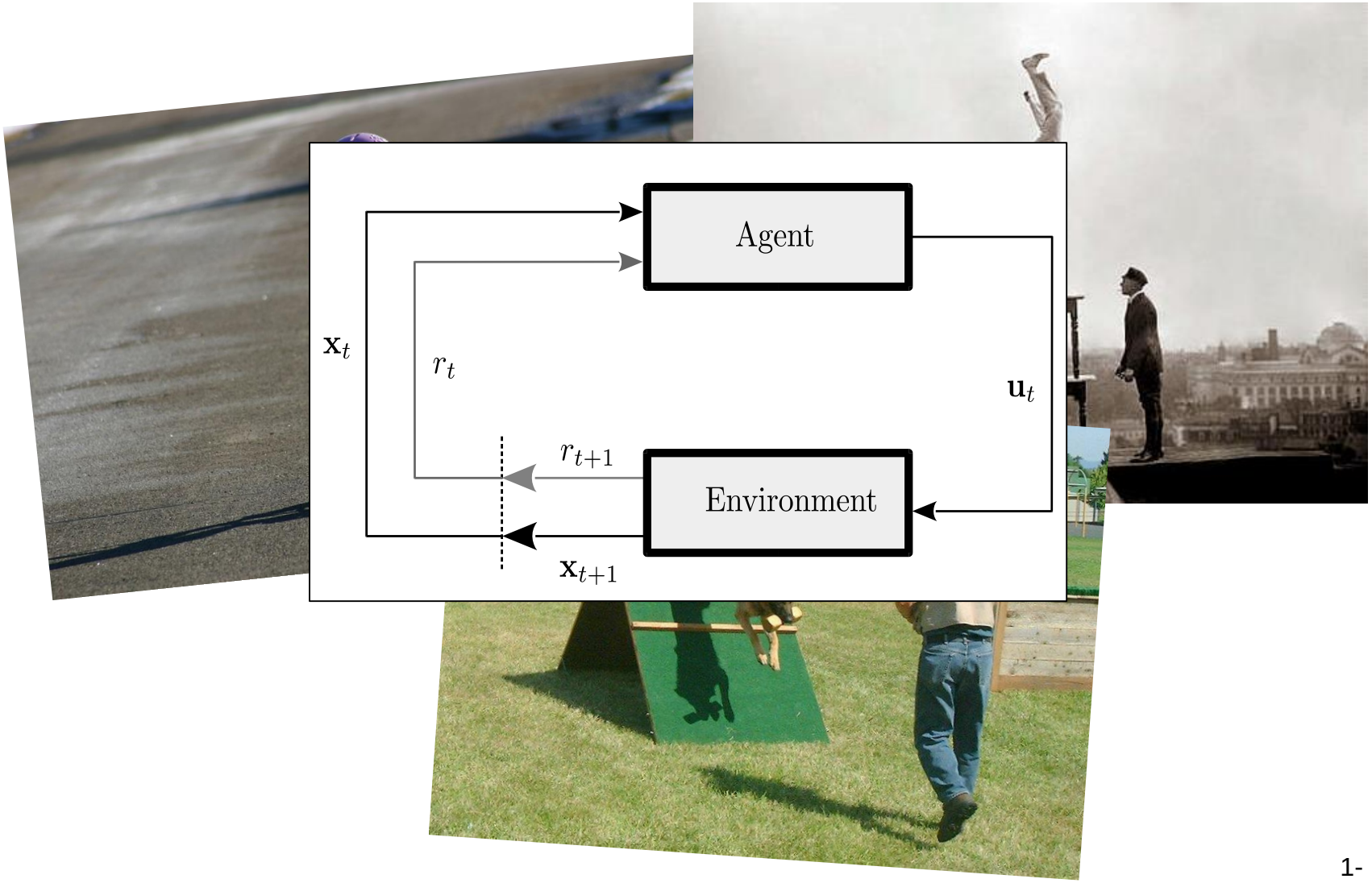
1.1 Terminology and problem definition

"... 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..."

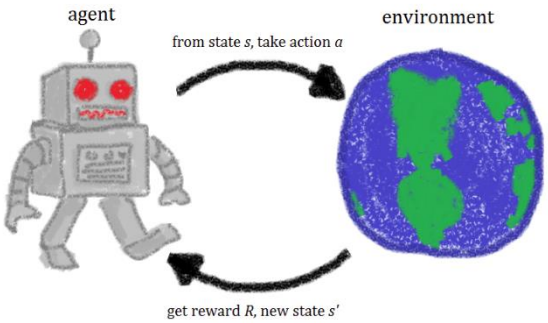
David Silver, DeepMind, 2015



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.

Kommentarfolie

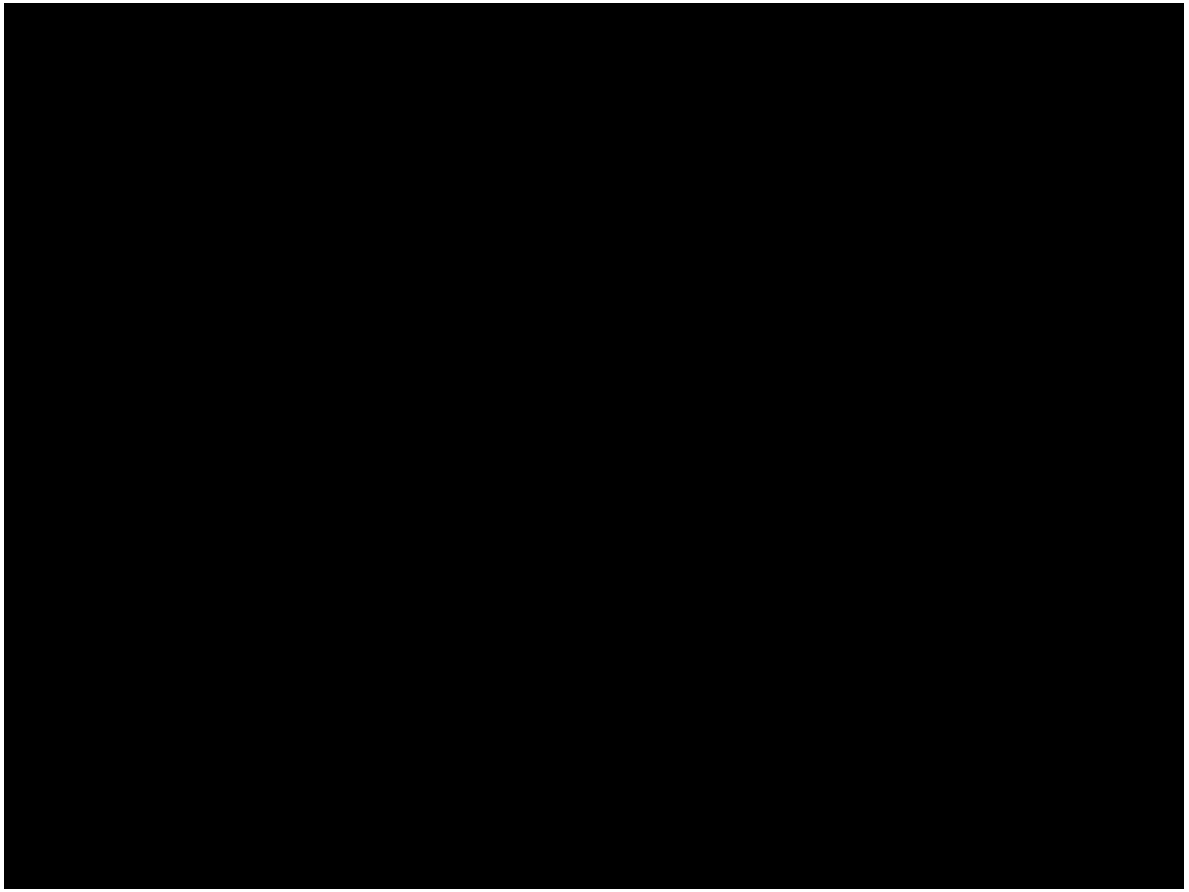
- The term **„Reinforcement learning“** describes **a very general idea of learning from trying out different things and getting a reward**. Many algorithms fall into this cathegory and without clarification it is hard to guess what people did when they say that they used Reinforcement Learning.
- Reinforcement Learning **does not fall into the supervised or unsupervised learning cathegories**, as the goal is not to reproduce or learn features from data, but rather to behave well in a specific environment.
- For notation, we have:
    - Discrete time-step variable: t
    - The state of the environment (e.g. position + velocity of a car) : $\mathbf{x}_t$
    - The action that the agent takes: $\mathbf{u}_t$
    - The reward given to the agent: r_t

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

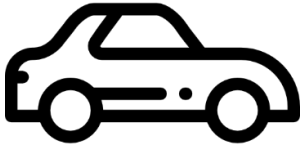
Reinforcement Learning

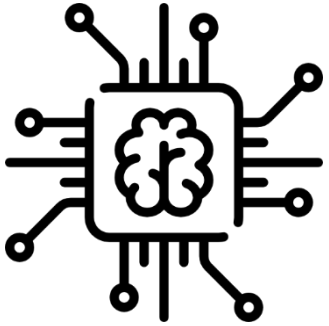
Johannes Betz / Prof. Dr. Markus Lienkamp / Prof. Dr. Boris Lohmann
(Christian Dengler, M. Sc.)

Agenda

1. 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.

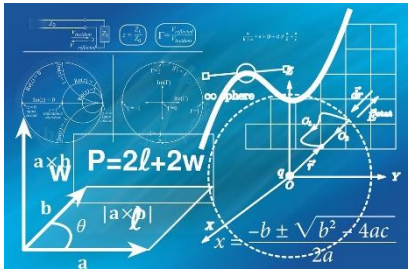




1.2 Motivation for RL in engineering

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.

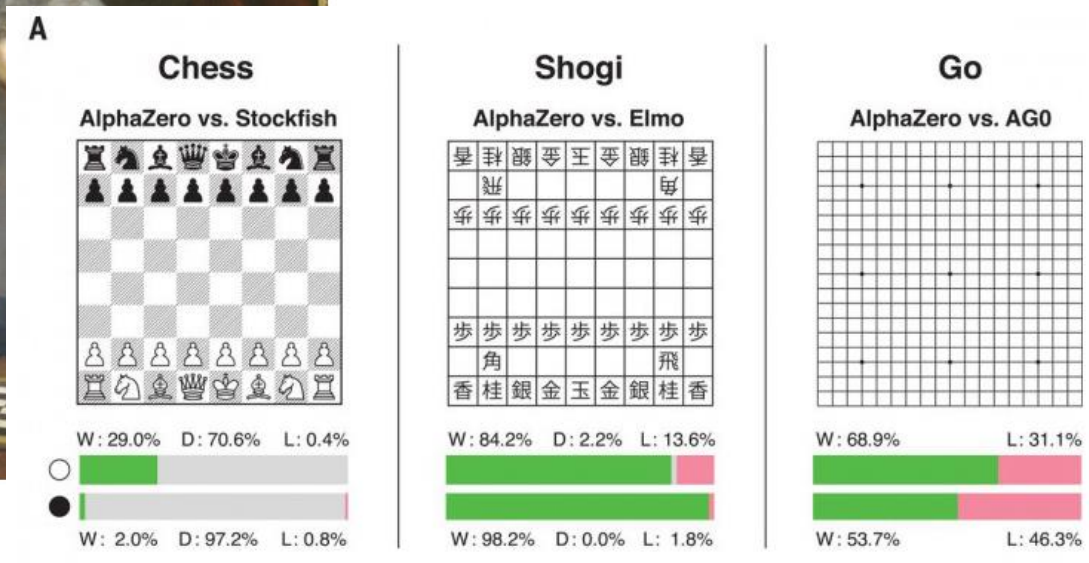
1.2 Motivation for RL in engineering

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 Silver (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.

Kommentarfolie

- Since Reinforcement Learning is a very general idea, the topic is broad and many algorithms have been developed. Algorithms exists for all combinations of stochastic and deterministic dynamics with discrete or continuous state and action spaces. We will consider the most simple case with only discrete state and actions.
- The concept of most algorithms are based on probability theory, thus it is required to have atleast basic knowledge of that topic.

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.

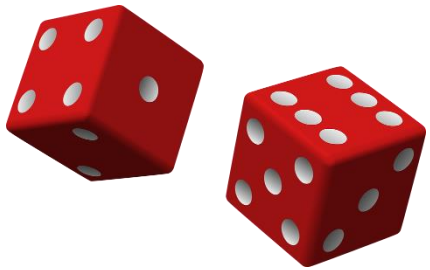
1.3 Depth of the topic and scope of the lecture

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

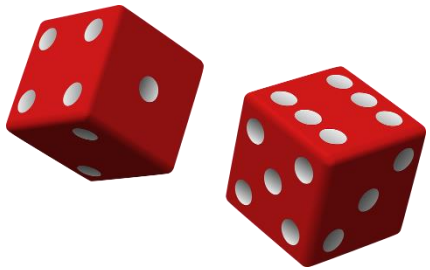
1.3 Depth of the topic and scope of the lecture

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



$$P(x=0|y=0) = \frac{10}{25} = \frac{2}{5}$$
$$P(x=1|y=0) = \frac{15}{25} = \frac{3}{5}$$

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:

$$\begin{aligned}\mathbb{E}^P[x] &= \sum_i P(x_i)\ x_i \\ &= \tfrac{1}{6}\cdot 1 + \tfrac{1}{6}\cdot 2 + \dots \\ &= \tfrac{21}{6} = 3.5\end{aligned}$$

- Useful relations:

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

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

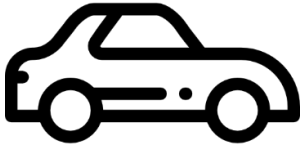
Reinforcement Learning

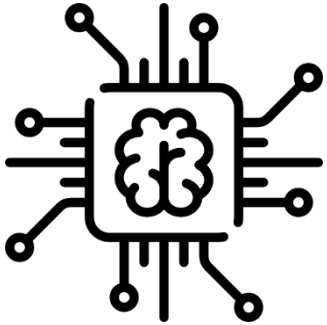
Johannes Betz / Prof. Dr. Markus Lienkamp / Prof. Dr. Boris Lohmann
(Christian Dengler, M. Sc.)

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.



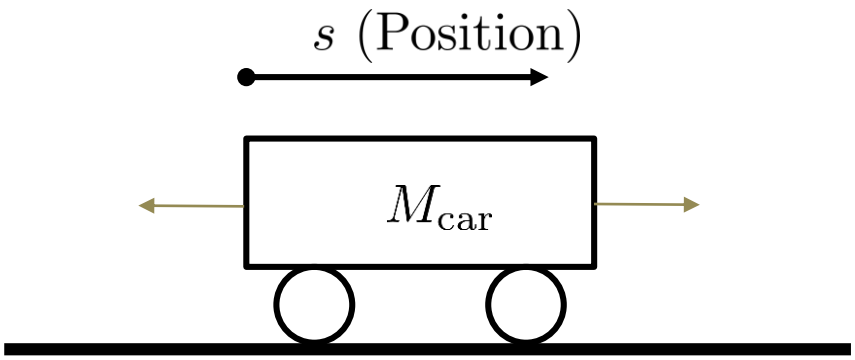


2.1 Markov-decision-process

Markov State

A state is Markov, if

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



$$x_t = s$$

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

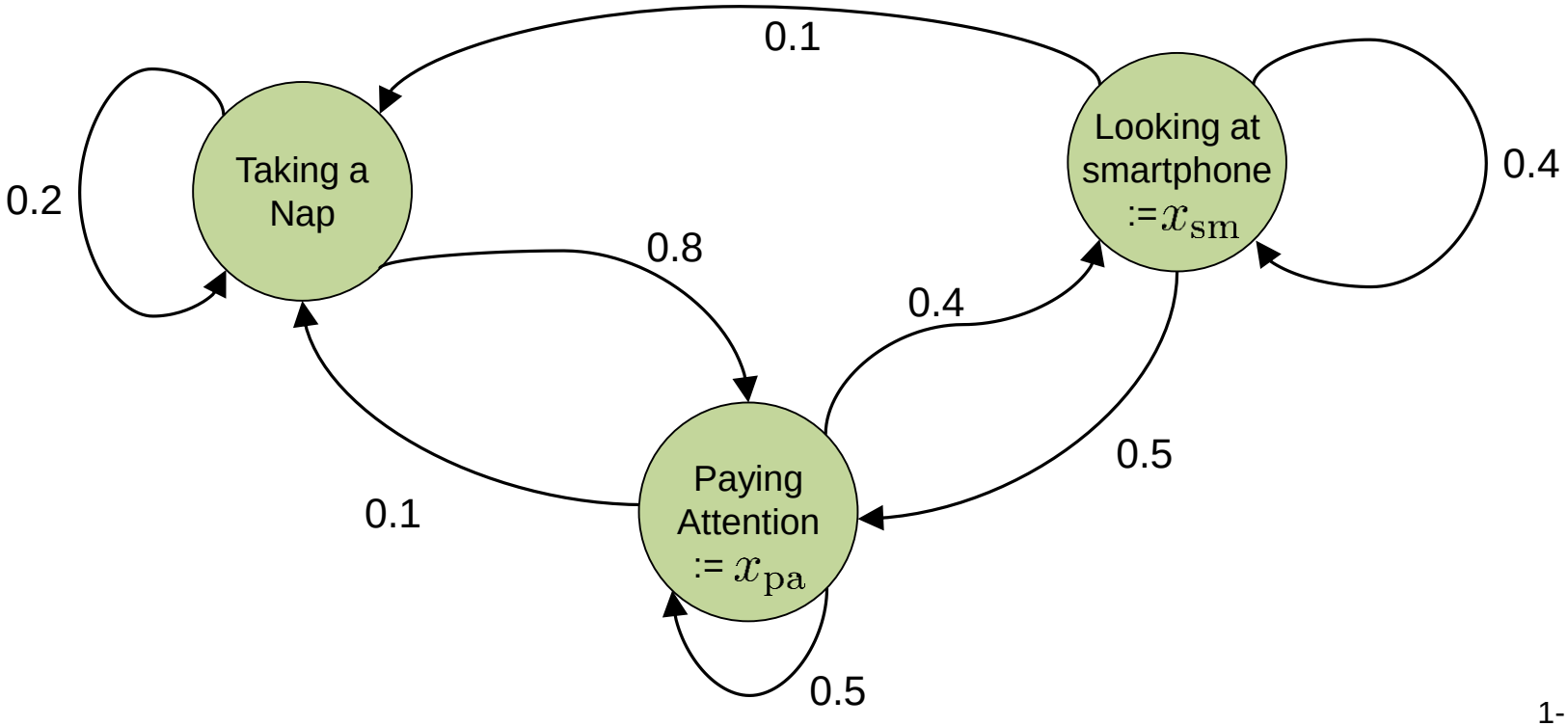
Kommentarfolie

- **A state is Markov if it fully describes the system at a given time t**. The past history does not matter anymore. To clarify this idea, image a cart moving on a 1D plane (no friction, no external forces). If we know the position and velocity of the cart, the next state is fully defined. We simply have to integrate the velocity for the defined time step size and add it to the current position. However if we only know the position at time t, we cannot know the next position at time t+1.
This is represented in the formula on the previous slide, whcih says that the probability of transitioning from one state to the next does not depend on the history $x_{t-1}, \ldots, x_0$.

2.1 Markov-decision-process

Markov-process

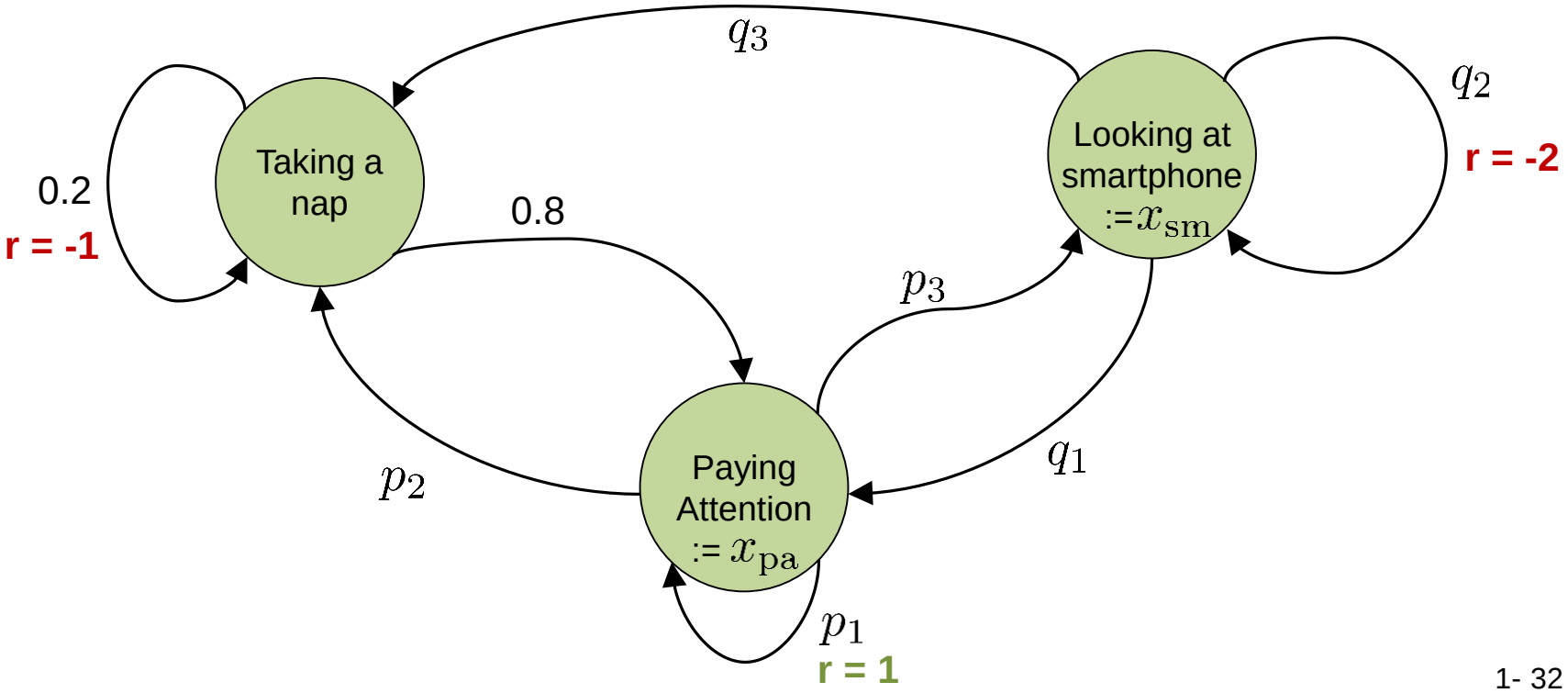
A Markov-process is sequence of random states with the Markov property.



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

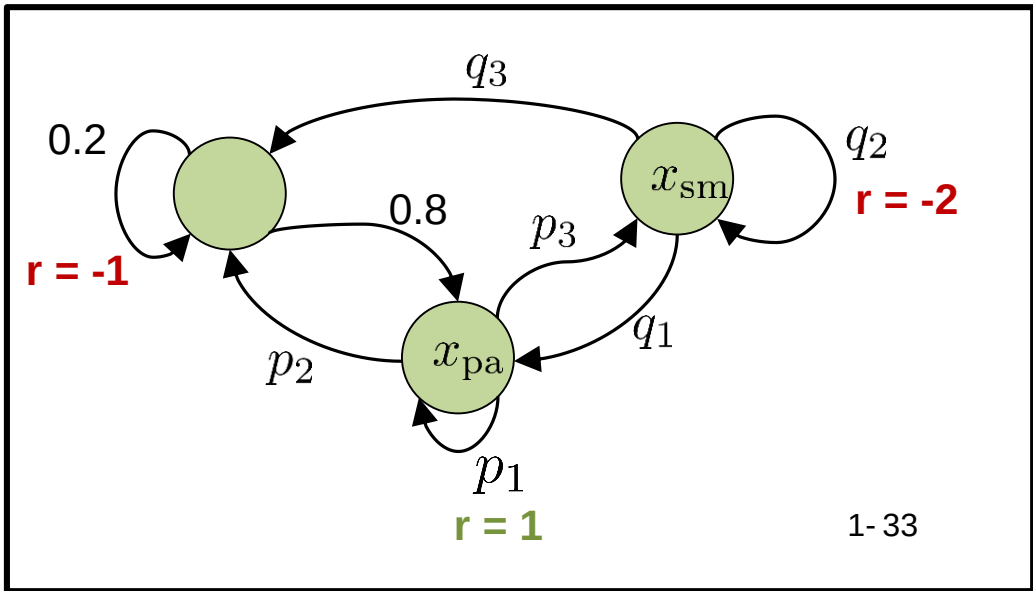
- **Goal**:
  - Find strategy which maximizes future rewards, i.e.:

$$\text{Find probabilities } p_1, p_2, p_3, q_1, q_2, q_3$$

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

$$\text{Maximizing :}$$

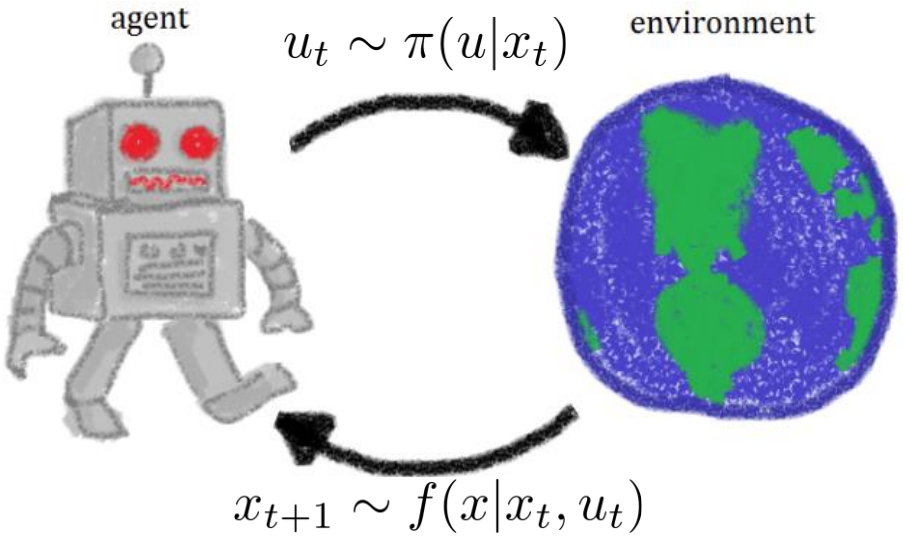
$$\sum_{t=0}^{\infty} \gamma^t \cdot r_t \ ; \ \gamma \in [0, 1]$$



Kommentarfolie

- The goal in our RL setup is to solve a Markov Decision Process, i.e. to find the best startegy which will result in the maximum possible reward.
- The rewards are accumulated in this example for infinite time steps. This could easily result in an accumulated reward of $\pm\infty$. To make the optimization problem well posed, two options exist.
  1) add a discount factor $\gamma \in]0, 1[$, which will make rewards far in the future less important.
  2) change the problem and only add rewards for a fixed and finite number of timesteps $t \in \{0, \ldots, T\}$

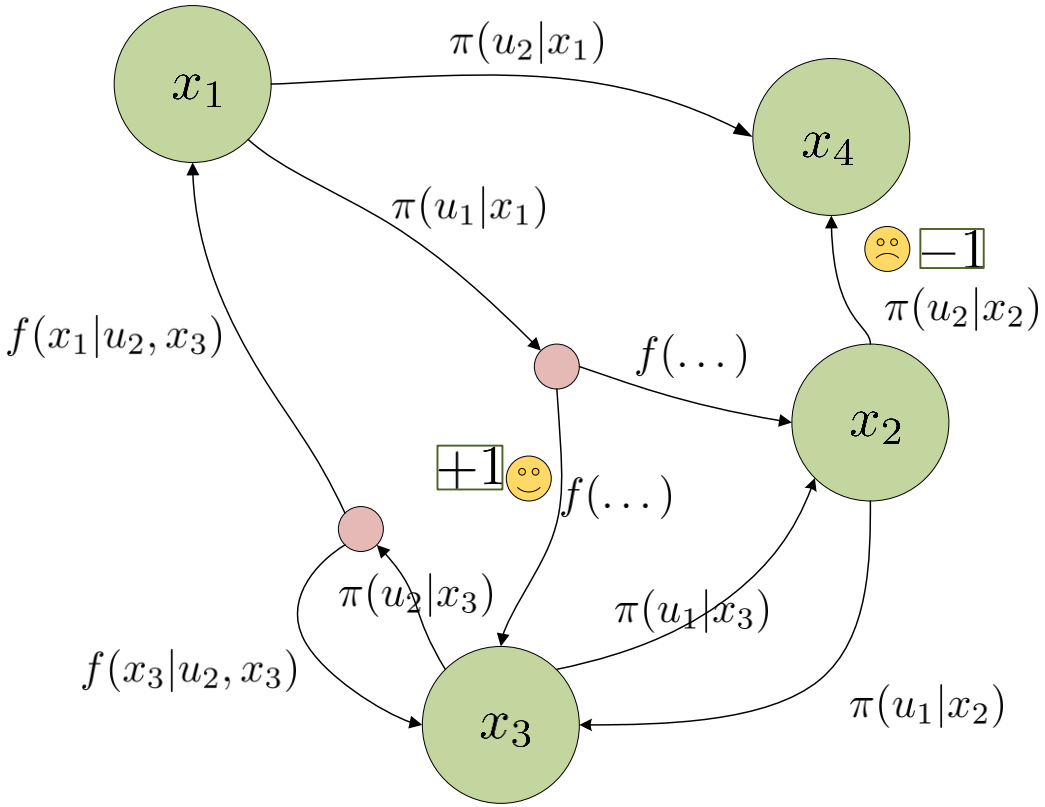
2.1 Markov-decision-process



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

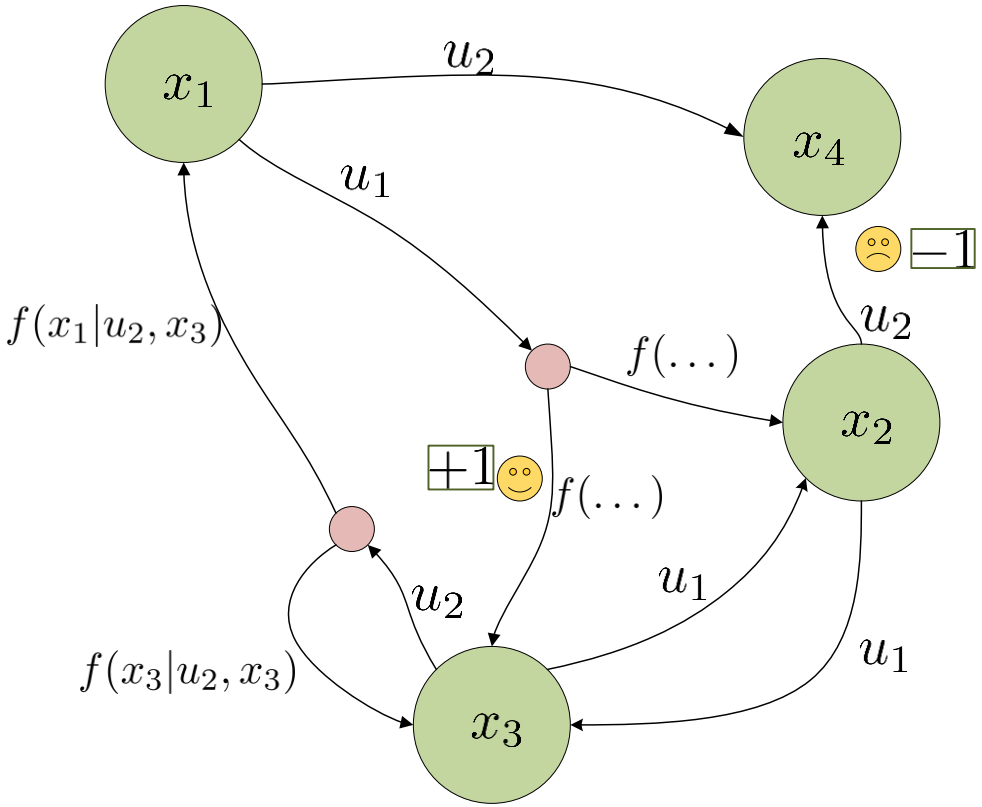
- **Assumptions**:
  - $\pi(u|x_t)$ and $f(x|x_t, u)$ are discrete probability distributions.
  - x is a Markovian state.

2.1 Markov-decision-process



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

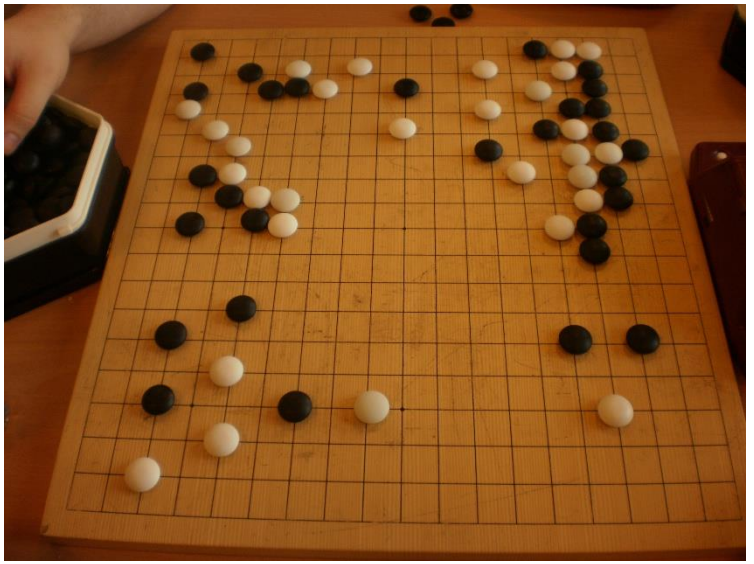
2.1 Markov-decision-process



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

2.1 Markov-decision-process

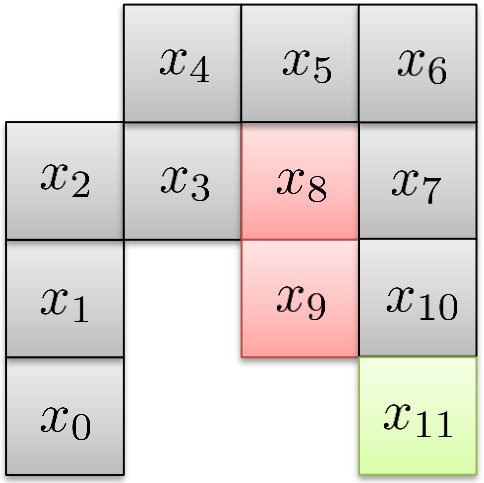
Example of discrete MDP’s



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)



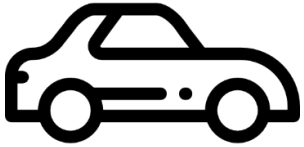
Reinforcement Learning

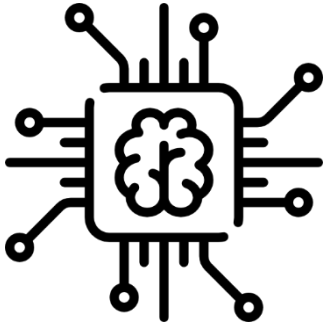
Johannes Betz / Prof. Dr. Markus Lienkamp / Prof. Dr. Boris Lohmann
(Christian Dengler, M. Sc.)

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.**





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

$$V^{\pi}(x) = \mathbb{E}^{\pi}\left[\sum_{\tau=t}^{\infty} \gamma^{\tau-t} r_{\tau+1} | x_t = x\right] \qquad 0 < \gamma <= 1 \quad \text{discount factor}$$

2.2 Value function, Q-learning etc

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

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

Bellmann equation ~1953, after Richard Ernest Bellman

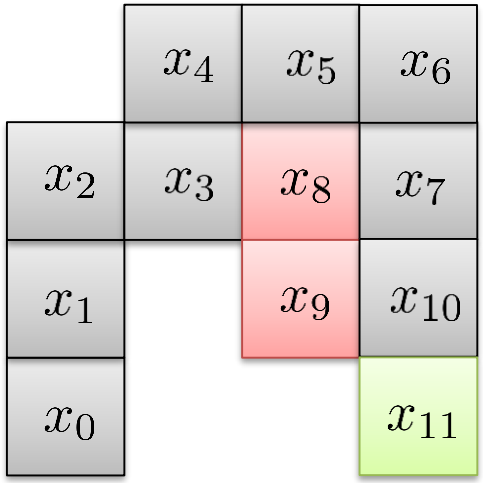
(Dynamic Programming!)

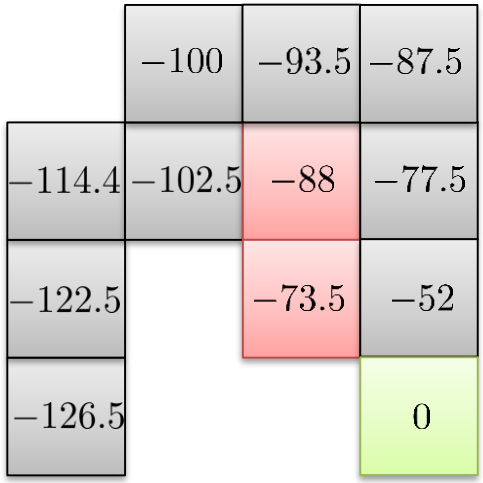
Kommentarfolie

- The Value Function can be expressed in a simple recursive way. This is very helpful for finding the value function, i.e. the V(x) which satisfies the Bellmann equation in every state.
- The value function depends on the state and also on the strategy that we chose.

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
⋮	
x_{10}	-52
x_{11}	0

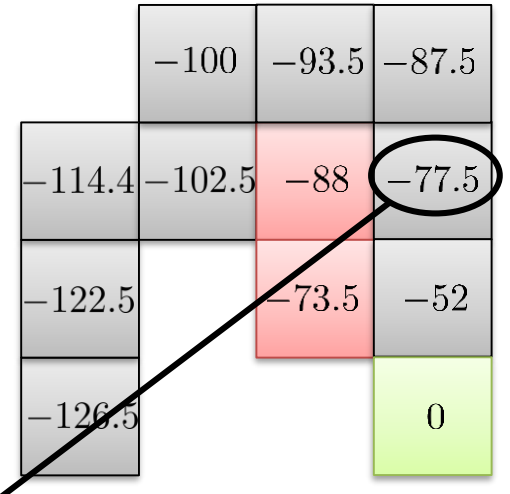
Uniform random strategy: $\pi_1(\uparrow, x) = \pi_1(\leftarrow, x) = \pi_1(\downarrow, x) = \pi_1(\rightarrow, x) = \frac{1}{4}$

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.

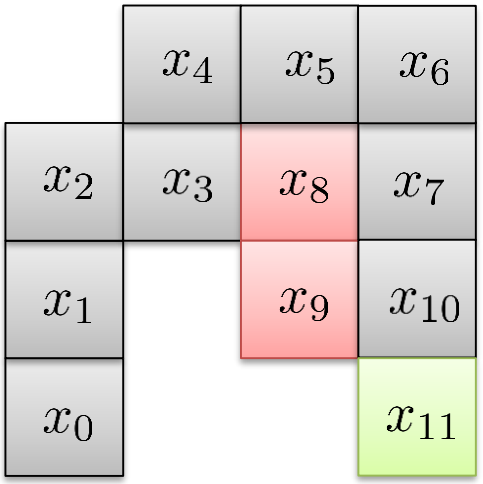
until convergence of V:
  for all states x:
    $V^{\pi}_{k+1}(x) = \sum_u \pi(u|x_t) \cdot (r_{t+1} + \gamma V^{\pi}_k(x_{t+1}))\,|x_t = x$
  end
end



2.2 Value function, Q-learning etc

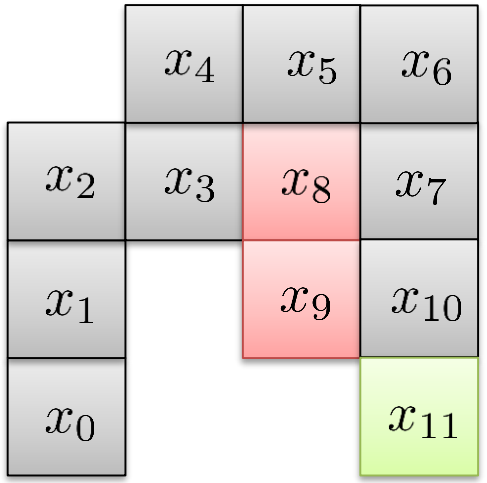
Policy evaluation using the Bellman equation

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

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

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

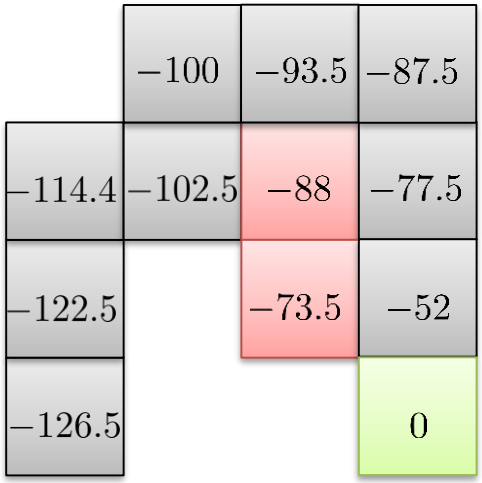
$$\vdots$$

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

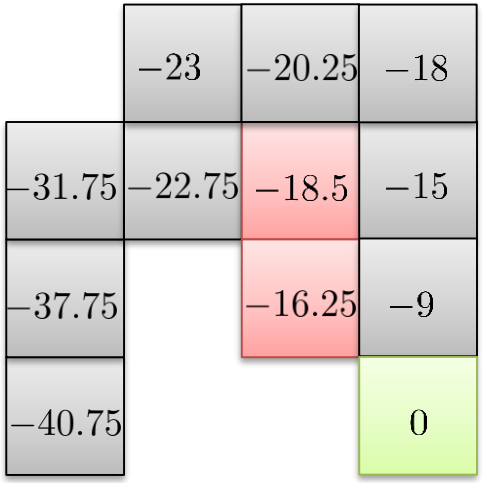
$$V(x_{11}) = 0$$

2.2 Value function, Q-learning etc

Grid World, Value Function



$V^{\pi_1}(x)$



$V^{\pi_2}(x)$

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

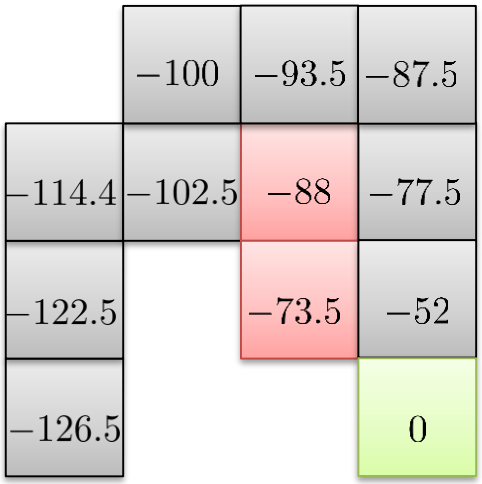
Different Strategy: $\pi_2(\uparrow, x) = \pi_2(\downarrow, x) = \pi_2(\rightarrow, x) = \frac{1}{3}, \quad \pi_2(\leftarrow, x) = 0$

2.2 Value function, Q-learning etc

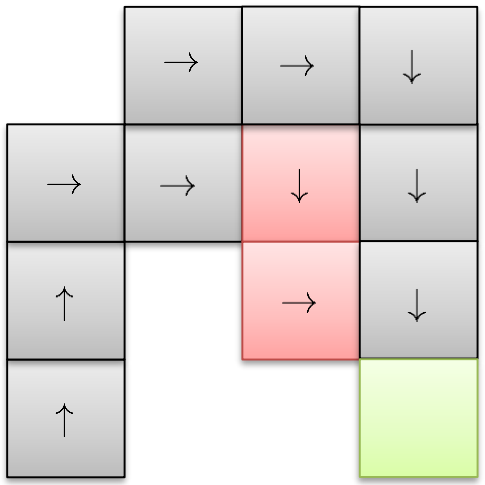
Policy improvement

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)**

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



$V^{\pi_1}(x)$



$\pi_2(x)$

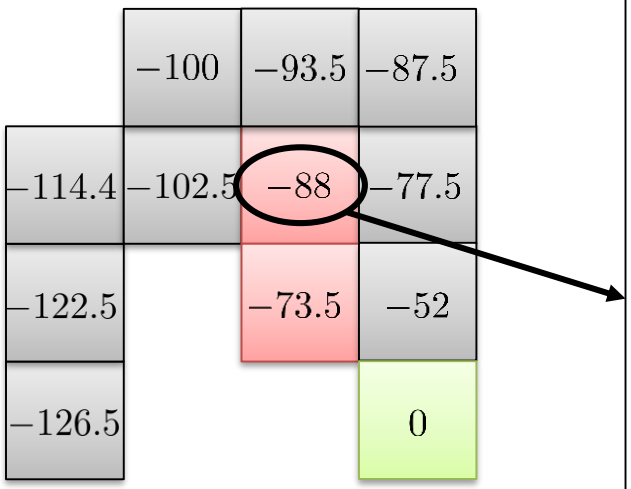
Kommentarfolie

- So far, we have seen an expensive way to compute the value function of a given policy: compute right hand side the Belmann equation for all states to get a new value function and repeat until convergence.
- In order to do this, we need to have a model of the environment and be able to compute the expectation of the right hand side, i.e. we need to know $f(x|x_t, u)$.
- So far we have only seen how to assess a policy, next we will see how to use this information to improve on our policy.

2.2 Value function, Q-learning etc

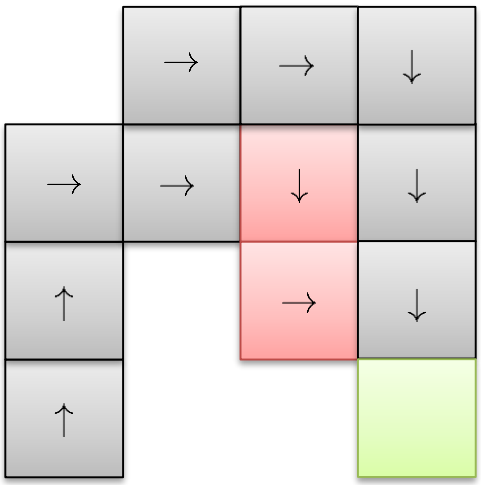
Policy improvement

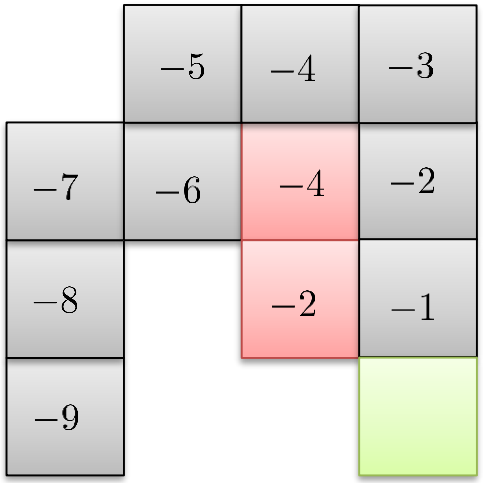
$$u_{\text{greedy}} = \arg\max_u \mathbb{E}[r_{t+1} + \gamma V^{\pi}(x_{t+1}) | x_t = x, u_t = u]$$

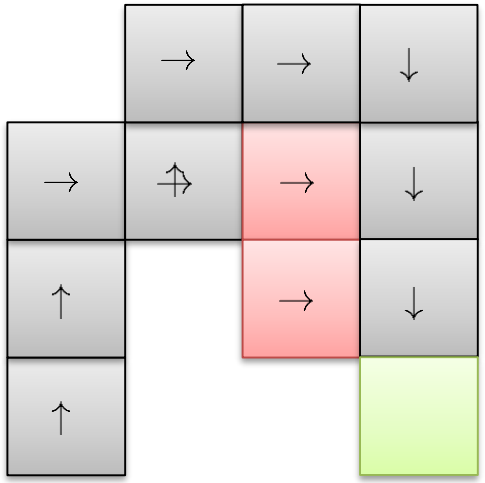


$V^{\pi_1}(x)$

2.2 Value function, Q-learning etc





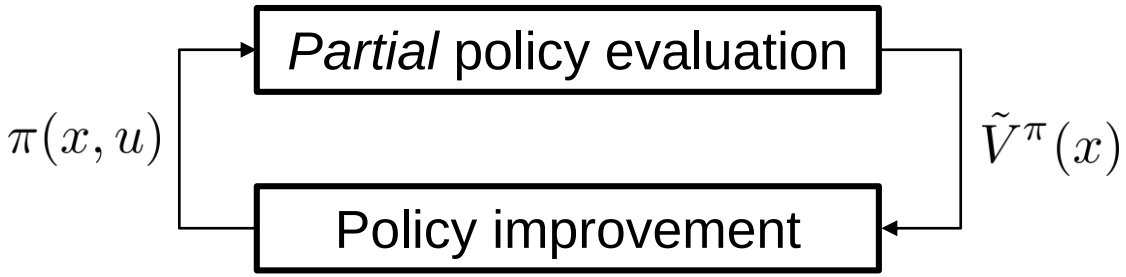


2.2 Value function, Q-learning etc

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.

Kommentarfolie

- By iterating policy evaluation and policy imporvement we can solve our MDP. However, so far we 1) need to know all transition probabilities and 2) need a lot of computations for computing the value function in each state.

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.

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

2.2 Value function, Q-learning etc

Definitions

Action Value Function

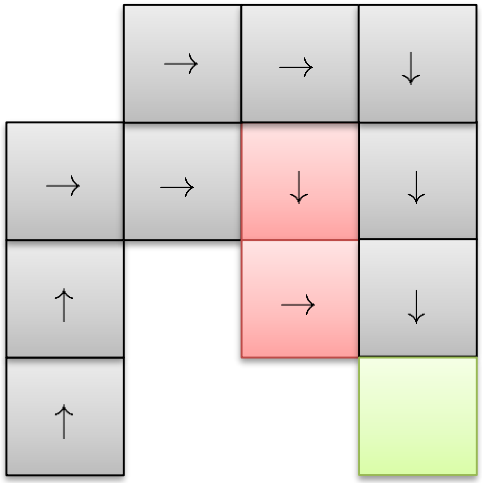
Function depending on the state, the next action and a policy. The function returns the expected future reward, starting in a state x, then choosing action u and afterwards following policy π.

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

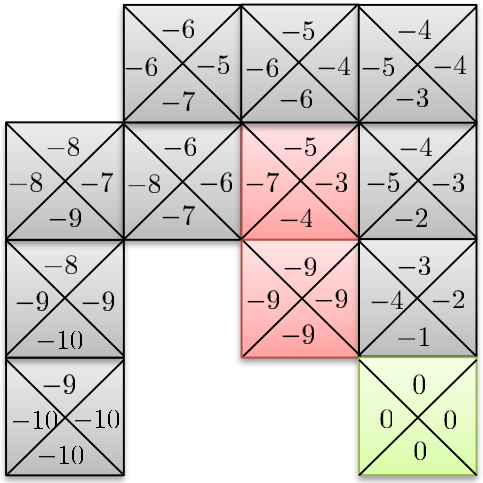
$$\begin{aligned} Q^{\pi}(x,u) &= \mathbb{E}^{\pi}\left[\sum_{\tau=t}^{\infty} \gamma^{\tau-t} r_{\tau+1} \mid x_t = x,\ u_t = u\right] \\ &= \mathbb{E}^{\pi}\left[r_{t+1} + \gamma Q(x_{t+1}, \pi(x_{t+1})) \mid x_t = x,\ u_t = u\right] \end{aligned}$$

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
⋮		⋮		
x_{10}	-3	-4	-1	-2
x_{11}	0	0	0	0

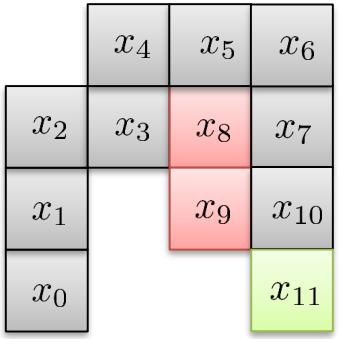
2.2 Value function, Q-learning etc

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!!!

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

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

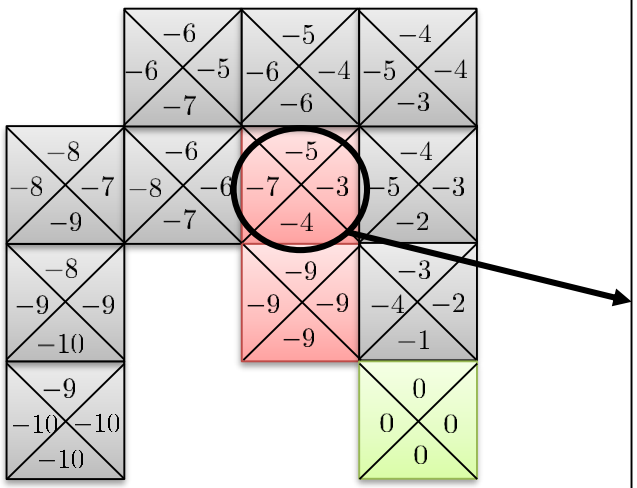


- **Disadvantage:**
  - More memory required and needs more time to be trained.

2.2 Value function, Q-learning etc

Grid World, Action Value Function

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



Kommentarfolie

- The action-value function contains more information than the value function. It expresses the costs to expect when chosing a specific action first, and then following the policy.
- However, for policy improvement, we do not need to know any transition probabilities anymore. We can just read the best action out of the table for each state. However, so far, we still need the transition probabilities in order to learn Q.

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

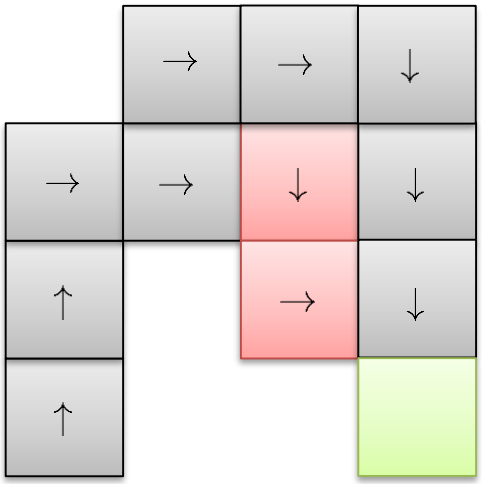
$$\sum_{k=0}^{\infty} \alpha_k = \infty \; ; \quad \sum_{k=0}^{\infty} \alpha_k^2 < \infty$$

- In practice not as bad, the assumptions can be relaxed and results will 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**.

Kommentarfolie

- We can learn the value or action-value function on data from interactions with the environment. However, convergence is conditioned on a few assumptions, the most notable of which is, that we need to explore, i.e. we need to be able to visit all state/action pairs of the MDP.
- After the policy improvement step, we used to have a deterministic policy which is unsuited for learning the Q-function, as exploration is not given. Therefore we introduced the epsilon-greedy policy, which leaves non-zero probability for all actions.

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.

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

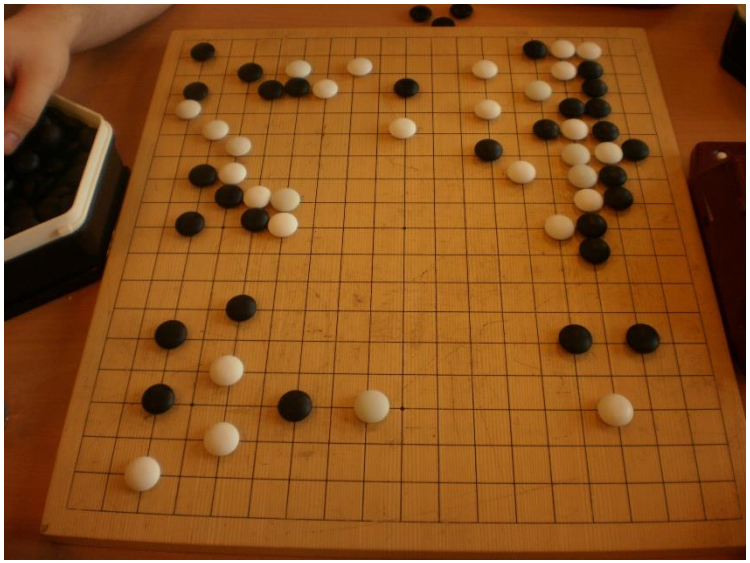
```
initialise a table of Q values (e.g. random)
provide epsilon, alpha, x0, x_end
for i = 1:N_episodes
    x = x0
    while x!=x_end
        u = eps_greedy(x, epsilon)
        # Interaction with the environment, take action a
        # receive next state x2 and reward r
        Q(x, u) = (1-alpha)*Q(x,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

Kommentarfolie

- Q-learning is a widely used algorithm. It combines policy evaluation and policy improvement. It can learn online, without the need to know transition probabilities. It requires some sort of exploration though, e.g. epsilon-greedy.

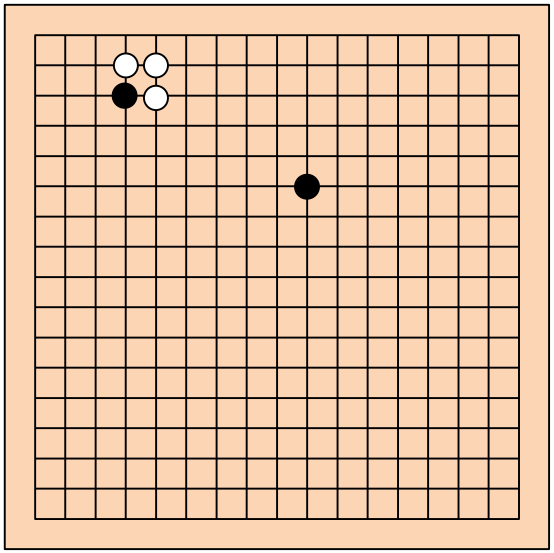
2.2 Value function, Q-learning etc

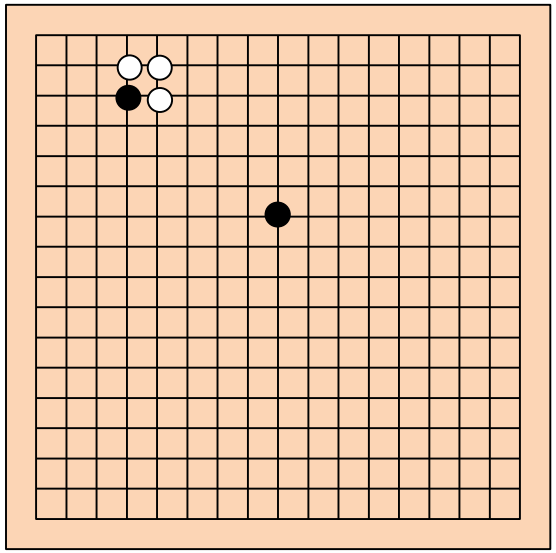
Example of discrete MDP's



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

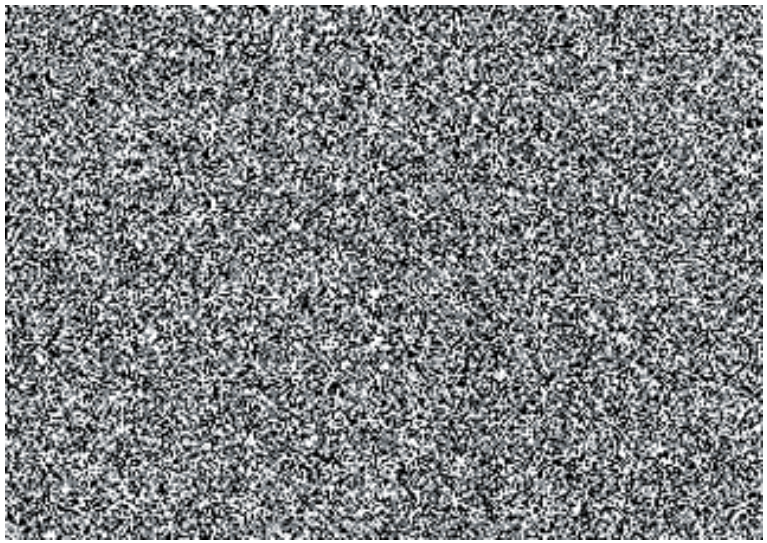
2.2 Value function, Q-learning etc





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!

Kommentarfolie

- While the Q-learning can in theory be used for all problems with discrete states and actions, in practice many problems tend to be too large and require too much memory and computation. In that case, the combination of reinforcement learning techniques with function approximation, e.g. neural networks is promising. This is however out of the scope of this lecture.

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.