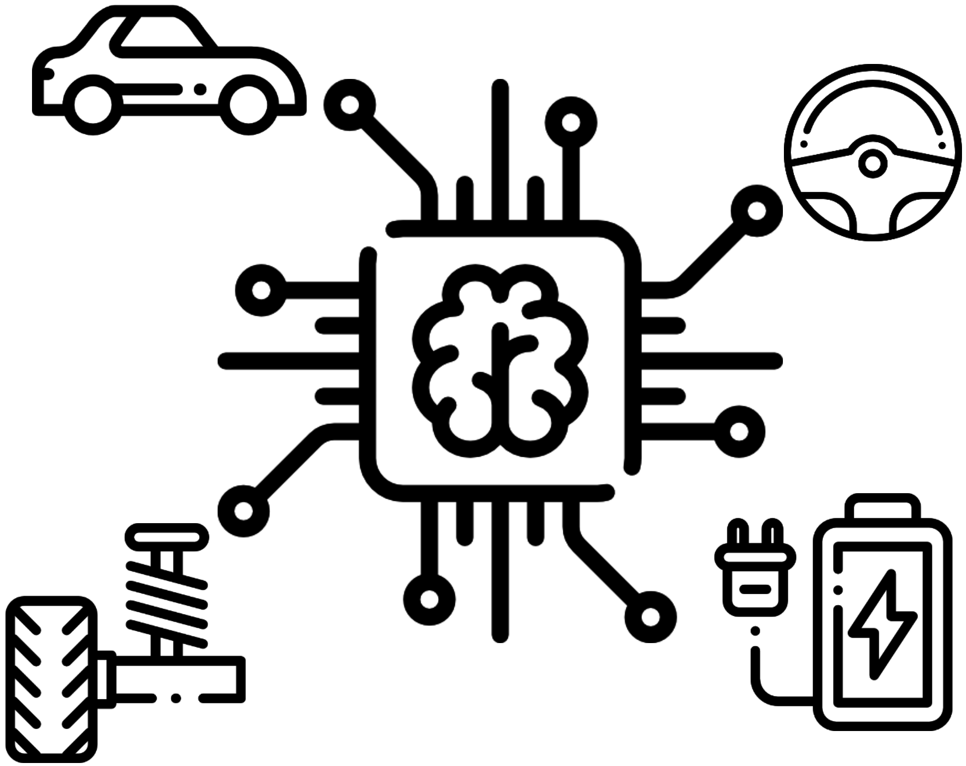
Institute of Automotive Technology
TUM School of Engineering and Design
Technical University of Munich

Artificial Intelligence in Automotive Technology

Maximilian Geißlinger / Fabian Netzler

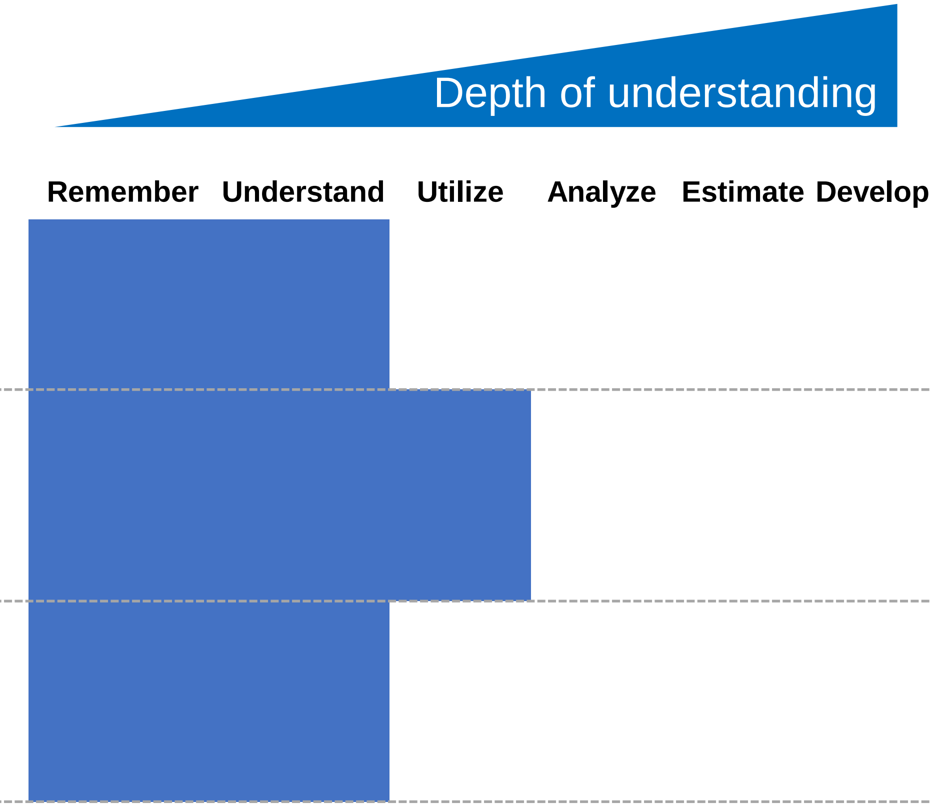
Prof. Dr.-Ing. Markus Lienkamp



Lecture Overview

Lecture 16:15-17:45	Practice 17:45-18:30
1 Introduction: Artificial Intelligence	20.10.2022 – Maximilian Geißlinger
2 Perception	27.10.2022 – Sebastian Huber
3 Supervised Learning: Regression	03.11.2022 – Fabian Netzler
4 Supervised Learning: Classification	10.11.2022 – Andreas Schimpe
5 Unsupervised Learning: Clustering	17.11.2022 – Andreas Schimpe
6 Introduction: Artificial Neural Networks	24.11.2022 – Lennart Adenaw
7 Deep Neural Networks	08.12.2022 – Domagoj Majstorovic
8 Convolutional Neural Networks	15.12.2022 – Domagoj Majstorovic
9 Knowledge Graphs	12.01.2023 – Fabian Netzler
10 Recurrent Neural Networks	19.01.2023 – Matthias Rowold
11 Reinforcement Learning	26.01.2023 – Levent Ögretmen
12 AI-Development	02.02.2023 – Maximilian Geißlinger
13 Guest Lecture	09.02.2023 – to be announced

Objectives of the lecture 11



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

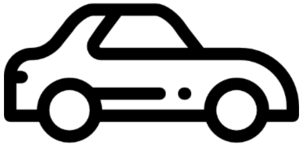
Understand the concept of a **value function** and **action-value function** in discrete state and action spaces.

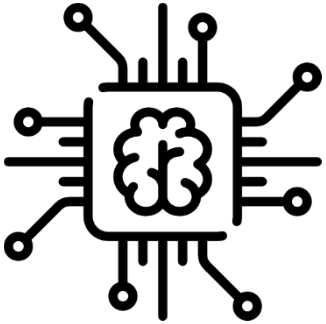
Understand the **basic RL methods** in discrete state and action space.

Reinforcement Learning

Maximilian Geißlinger / Fabian Netzler / Prof. Dr. Markus Lienkamp
(Levent Ögretmen, M. Sc.)

Agenda



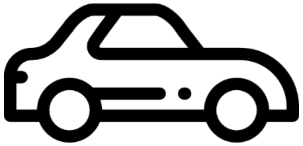


Reinforcement Learning

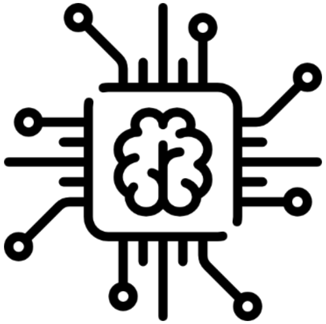
Maximilian Geißlinger / Fabian Netzler / Prof. Dr. Markus Lienkamp
(Levent Ögretmen, M. Sc.)

Agenda

1. Terminology and Concept
   - **<u>1.1 Terminology and problem definition</u>**
   - 1.2 Basic probability theory for discrete variables



2. RL in discrete state- and action-spaces
   - 2.1 Markov decision processes
   - 2.2 Value and Action Value Function
   - 2.3 Policy Iteration
   - 2.4 Q-Learning



1.1 Terminology and problem definition

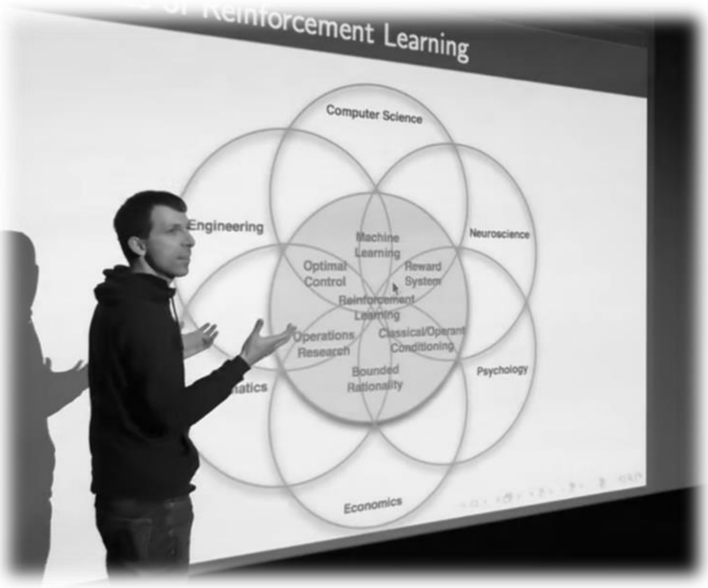
Revision

- 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
  - ?

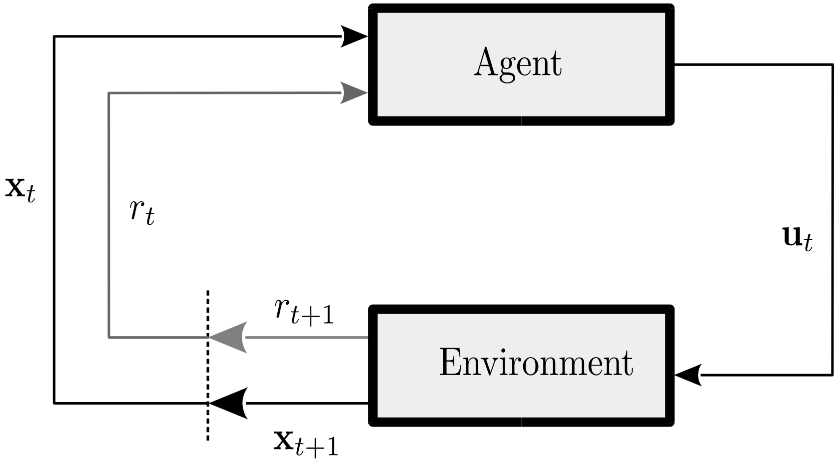
1.1 Terminology and problem definition

"... 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

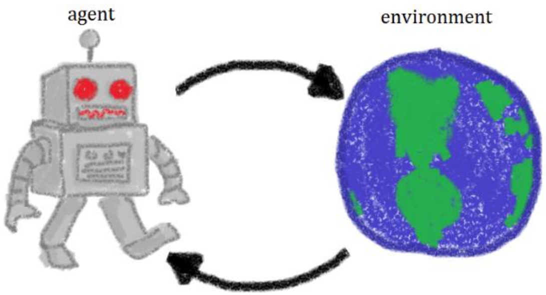


1.1 Terminology and problem definition



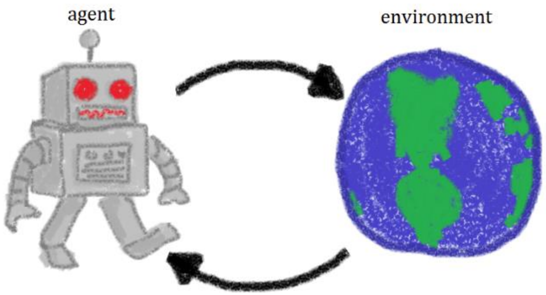
- **Agent**: Dog
- **Environment**: Owner, Obstacles etc.
- **Action $\mathbf{u_t}$**: Movements of Dog
- **State $\mathbf{x_t}$**: Position, Velocity etc. of Dog
- **Reward $\mathbf{r_t}$**: Treats

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

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

1.1 Terminology and problem definition

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).

1.1 Terminology and problem definition

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

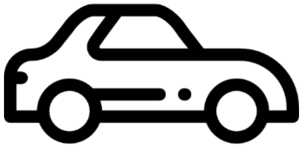
- Lecture by Sergey Levine (UC Berkeley), **>20h material**
  http://rail.eecs.berkeley.edu/deeprlcourse/

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

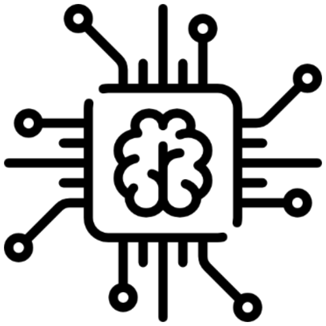
→ Focus here on the most simple case of making decisions in discrete states and actions.

Agenda

1. Terminology and Concept
   - 1.1 Terminology and problem definition
   - **1.2 Basic probability theory for discrete variables**



2. RL in discrete state- and action-spaces
   - 2.1 Markov decision processes
   - 2.2 Value and Action Value Function
   - 2.3 Policy Iteration
   - 2.4 Q-Learning



1.2 Basic probability theory for discrete variables

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

1.2 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

x=1

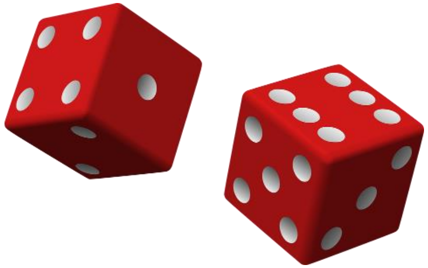
y=1

1.2 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



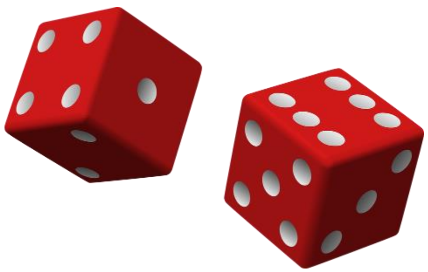
$$P(x=0,y=0)=\frac{10}{36}=\frac{5}{18}$$
$$P(x=0,y=1)=0$$
$$P(x=1,y=0)=\frac{15}{36}$$
$$P(x=1,y=1)=\frac{11}{36}$$

1.2 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!

1.2 Basic probability theory for discrete variables

- Expected value, dice example:

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

- Useful relations:

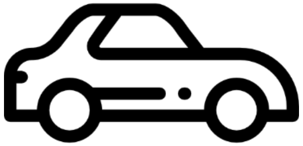
$$P(x) = \sum_i P(x, \mathrm{y}_i)$$
$$P(x, y) = P(x|y) \cdot P(y)$$

Reinforcement Learning

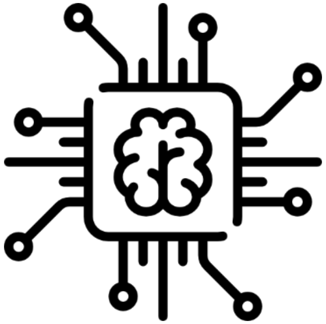
Maximilian Geißlinger / Fabian Netzler / Prof. Dr. Markus Lienkamp
(Levent Ögretmen, M. Sc.)

Agenda

1. Terminology and Concept
    - 1.1 Terminology and problem definition
    - 1.2 Basic probability theory for discrete variables



2. RL in discrete state- and action-spaces
    - **<u>2.1 Markov decision processes</u>**
    - 2.2 Value and Action Value Function
    - 2.3 Policy Iteration
    - 2.4 Q-Learning

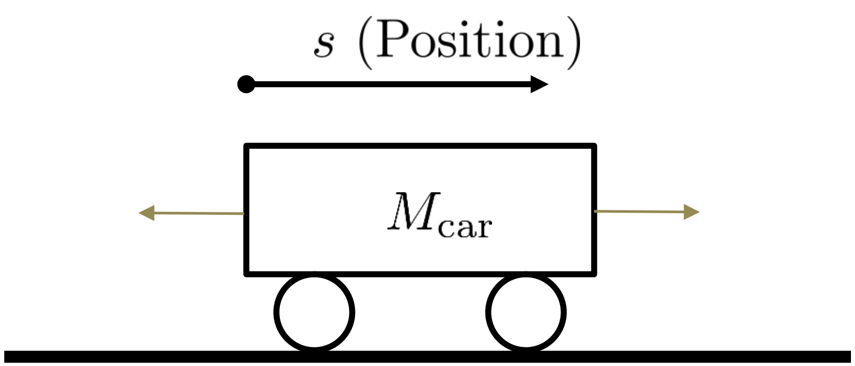


2.1 Markov-decision-process

Markov State

A state is Markov, if

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



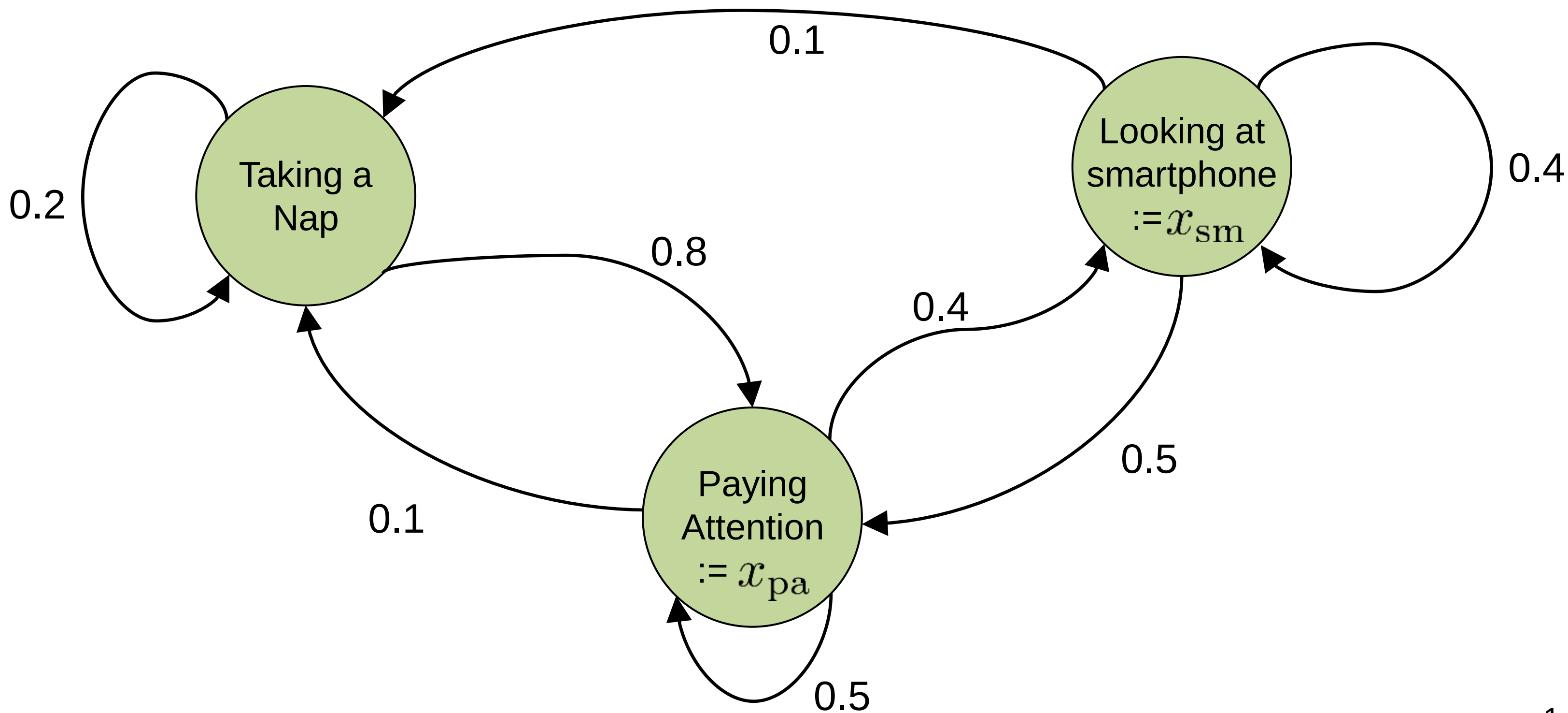
$$x_t \cancel{=} s$$

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

2.1 Markov-decision-process

Markov-process

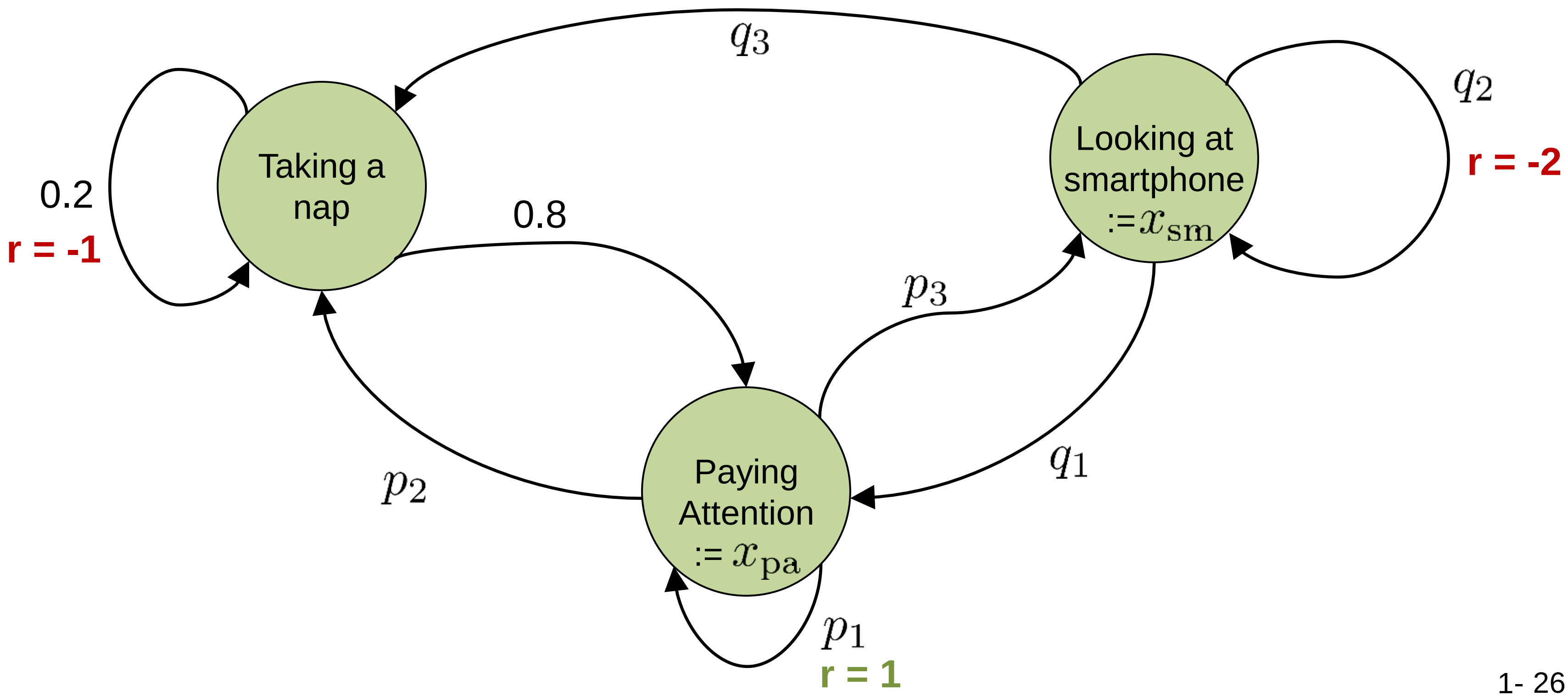
A Markov-process is sequence of random states with the Markov property.



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.:

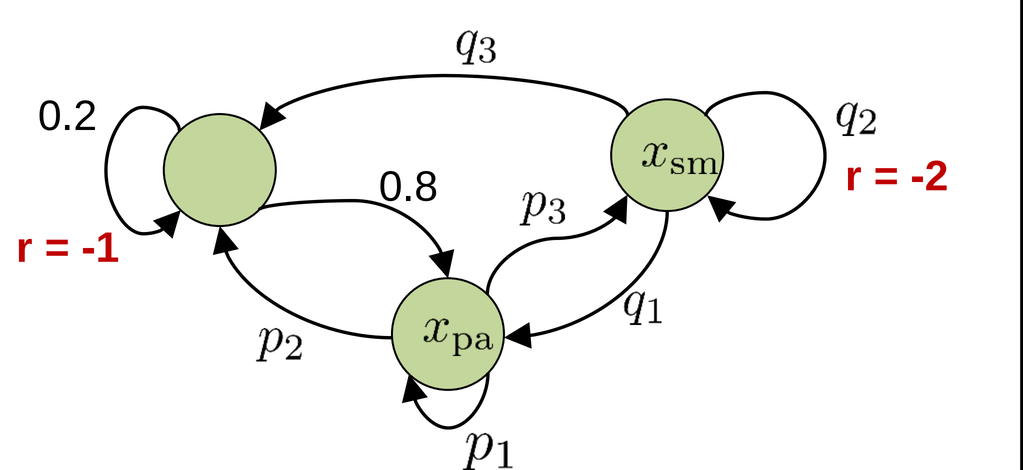
Find probabilities $p_1, p_2, p_3, q_1, q_2, q_3$

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

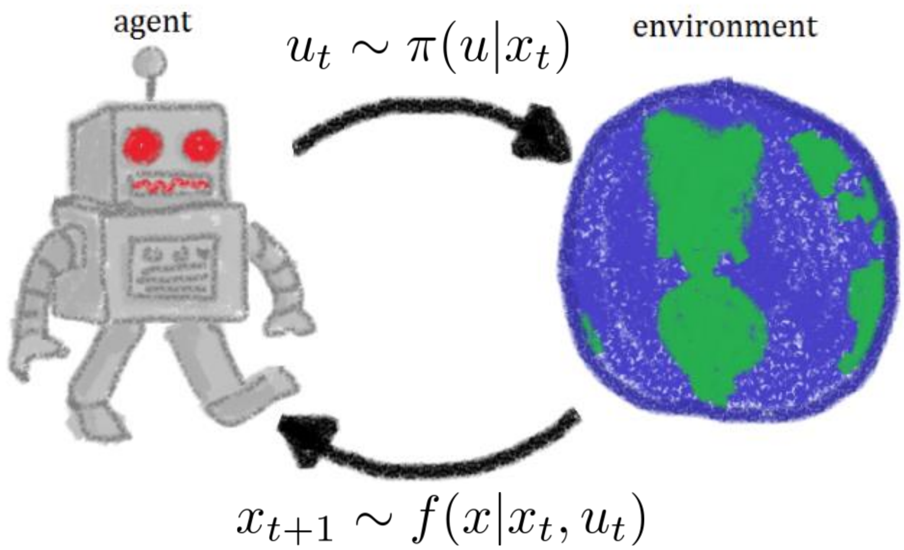
Maximizing :

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

$$= r_0 + \gamma r_1 + \gamma^2 r_2 + \; ...$$



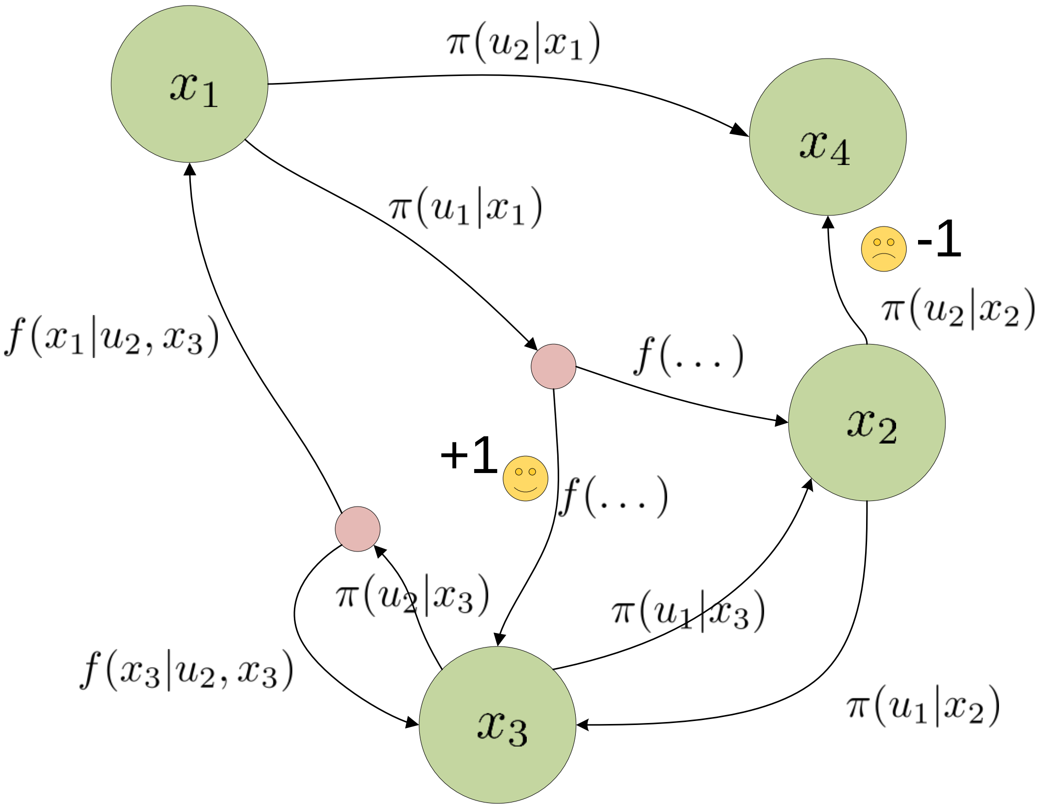
2.1 Markov-decision-process



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

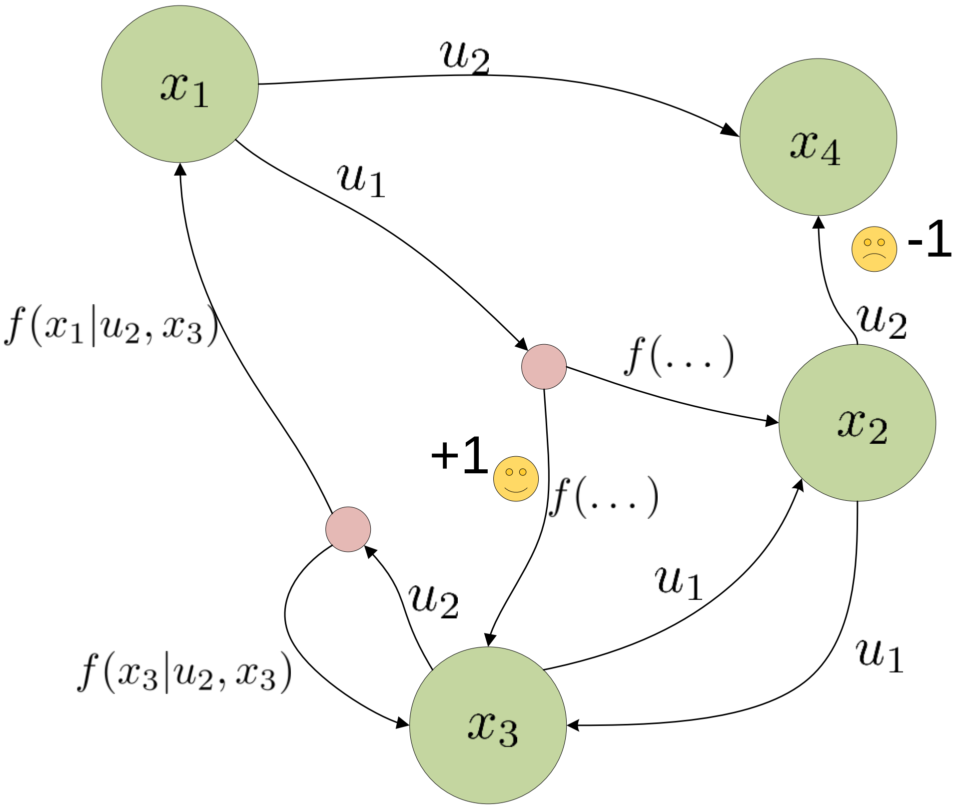
- **Assumptions**:
  - $\pi(u|x_t)$ and $f(x|x_t, u_t)$ are discrete probability distributions.
  - x is a Markovian state.

2.1 Markov-decision-process



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

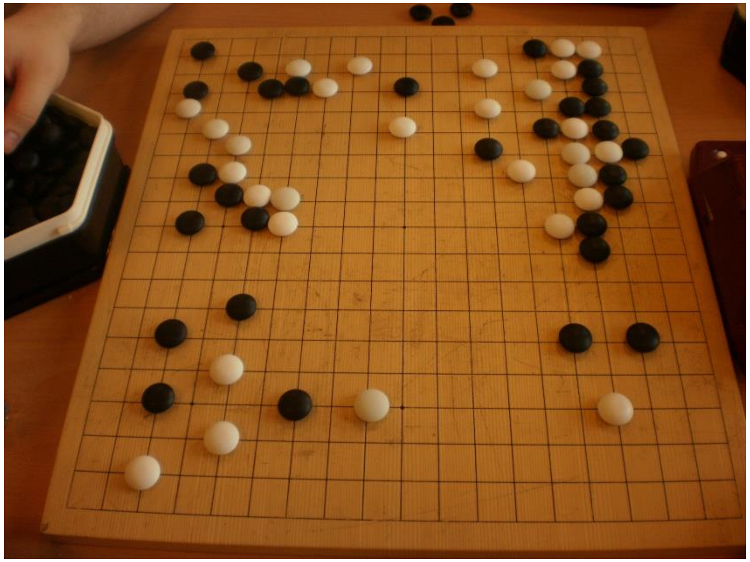
2.1 Markov-decision-process



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

2.1 Markov-decision-process

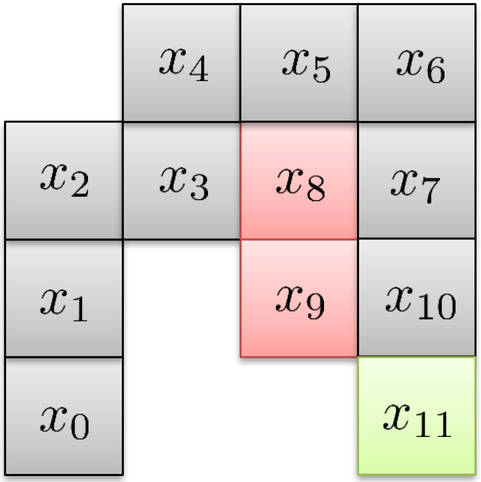
Example of discrete MDP's



2.1 Markov-decision-process

Example: Grid World

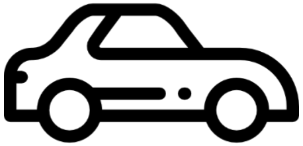
- 12 states/positions
- 4 actions per state: go **up**, **down, left**, **right** (Hitting a wall is possible and means no movement)
- Different reward depending on the state.
  - -1 when moving to a grey or green state
  - -2 when moving to a red state
- Initial state x_0 (One always starts here)
-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



Reinforcement Learning

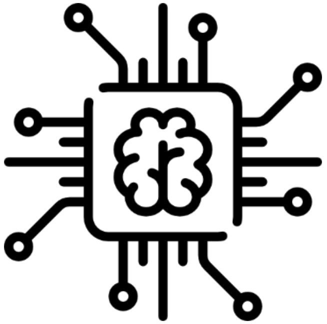
Maximilian Geißlinger / Fabian Netzler / Prof. Dr. Markus Lienkamp
(Levent Ögretmen, M. Sc.)

Agenda



1. Terminology and Concept
    1.1 Terminology and problem definition
    1.2 Basic probability theory for discrete variables

2. RL in discrete state- and action-spaces
    2.1 Markov decision processes
    2.2 Value and Action Value Function
    2.3 Policy Iteration
    2.4 Q-Learning



2.2 Value and Action Value Function

Definitions

Value Function $\boldsymbol{V^{\pi}(x)}$

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 $\boldsymbol{Q^{\pi}(x,u)}$

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 and Action Value Function

Value function

Value Function $V^{\pi}(x)$

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.

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

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

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

Bellmann equation (Richard Bellman)

2.2 Value and Action Value Function

Definitions

Action Value Function $\boldsymbol{Q^{\pi}(x,u)}$

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.

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

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

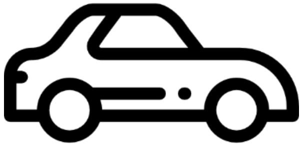
$$Q^{\pi}(x,\pi(x)) = V^{\pi}(x)$$

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

Reinforcement Learning

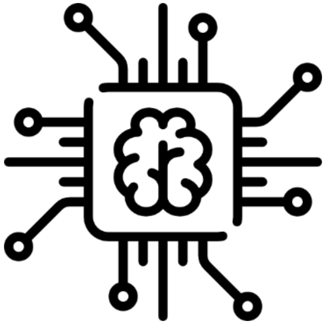
Maximilian Geißlinger / Fabian Netzler / Prof. Dr. Markus Lienkamp
(Levent Ögretmen, M. Sc.)

Agenda



1. Terminology and Concept
    - 1.1 Terminology and problem definition
    - 1.2 Basic probability theory for discrete variables

2. RL in discrete state- and action-spaces
    - 2.1 Markov decision processes
    - 2.2 Value and Action Value Function
    - **<u>2.3 Policy Iteration</u>**
    - 2.4 Q-Learning



2.3 Policy Iteration

Goal: Find optimal policy.

1. Start with random policy $\pi(x, u)$
2. Evaluate policy, i.e. calculate $V^{\pi}(x)$
3. Improve policy

repeat

2.3 Policy Iteration

Goal: Find optimal policy.

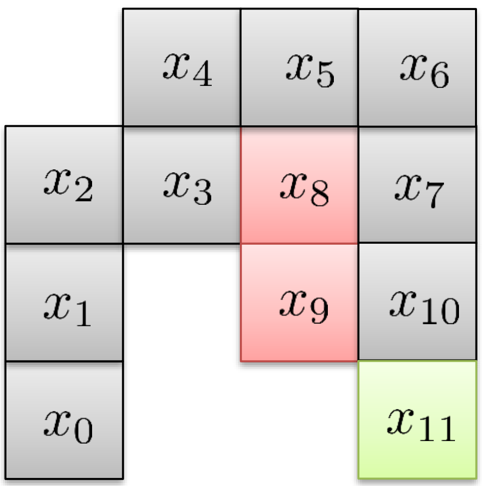
1. Start with random policy $\pi(x,u)$
2. Evaluate policy, i.e. calculate $V^{\pi}(x)$
3. Improve policy

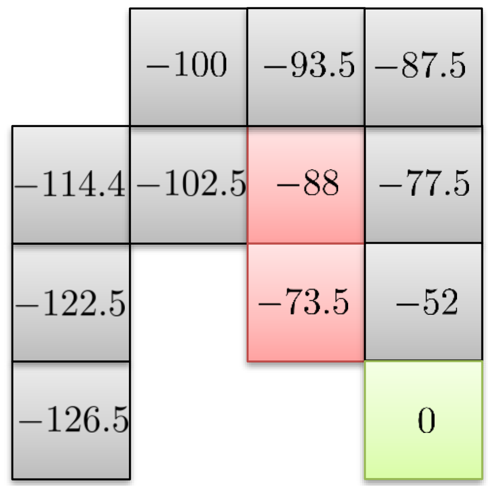
repeat

2.3 Policy Iteration

Grid World, Value Function

$\gamma = 1$





$V^{\pi_1}(x)$

V(x) in PC Memory:

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

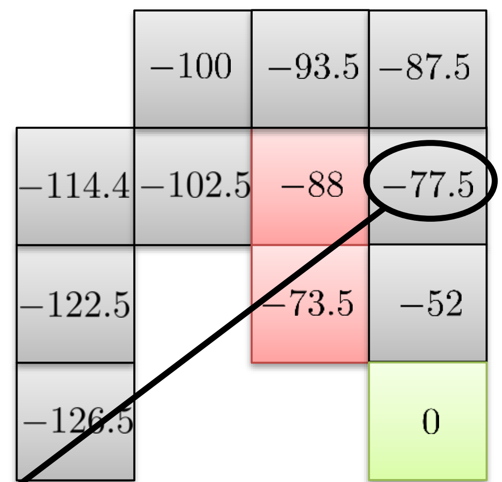
Uniform random strategy: $\pi_1(\uparrow, x) = \pi_1(\leftarrow, x) = \pi_1(\downarrow, x) = \pi_1(\rightarrow, x) = \frac{1}{4}$

2.3 Policy Iteration

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



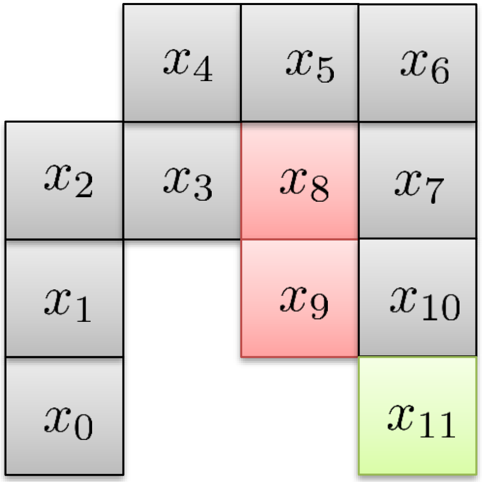
$\gamma = 1$

$$\boxed{V^{\pi_1}_{k+1}(x_7) = \tfrac{1}{4}\cdot(-1-87.5)+\tfrac{1}{4}\cdot(-2-88)+\tfrac{1}{4}\cdot(-1-52)+\tfrac{1}{4}\cdot(-1-77.5) = -77.5}$$

2.3 Policy Iteration

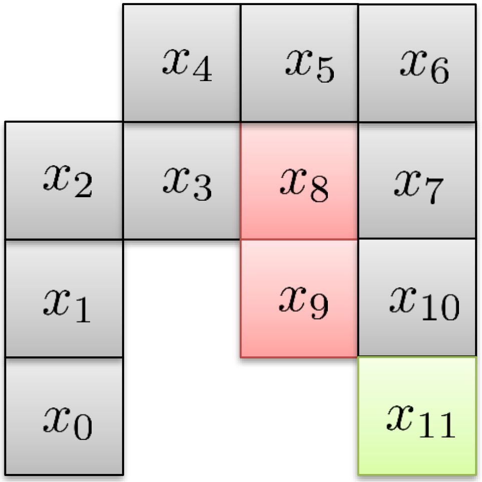
Policy evaluation using the Bellman equation

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.



2.3 Policy Iteration

Policy evaluation using the Bellman equation



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

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

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

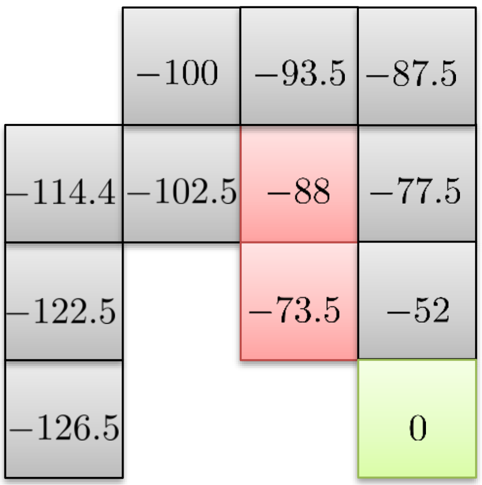
$$\vdots$$

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

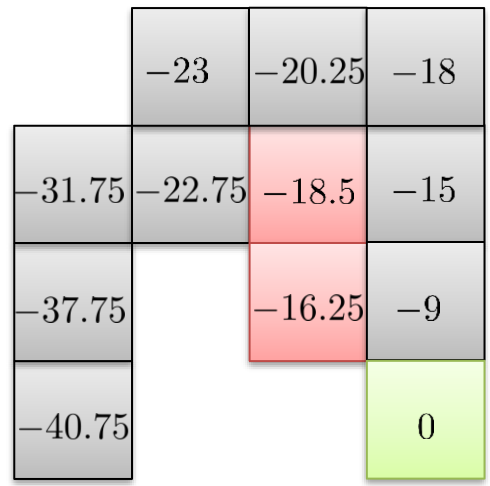
$$V(x_{11}) = 0$$

2.3 Policy Iteration

Grid World, Value Function



$V^{\pi_1}(x)$



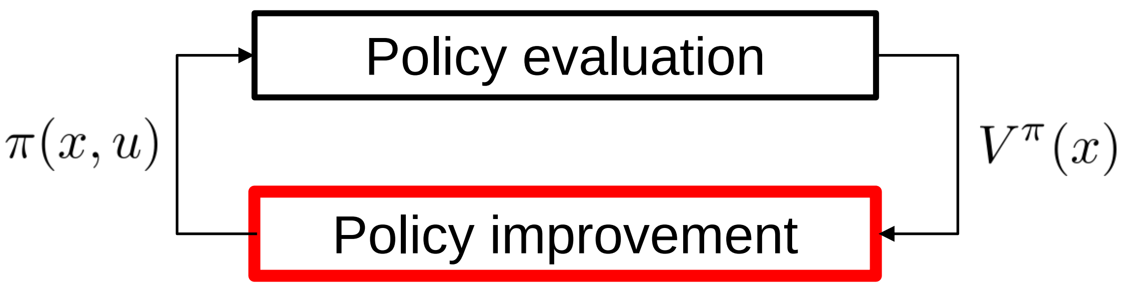
$V^{\pi_2}(x)$

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

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

2.3 Policy Iteration

Goal: Find optimal policy.



1. Start with random policy $\pi(x,u)$
2. Evaluate policy, i.e. calculate $V^\pi(x)$
3. Improve policy

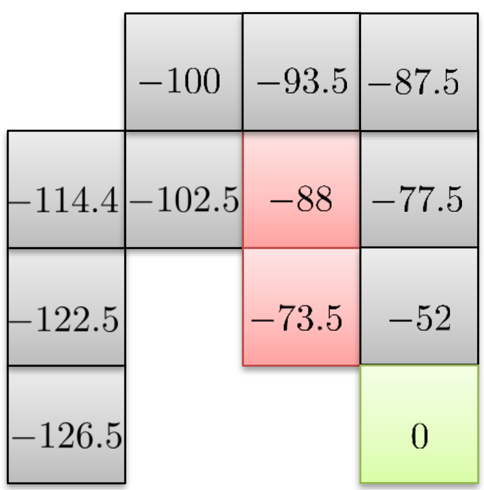
repeat

2.3 Policy Iteration

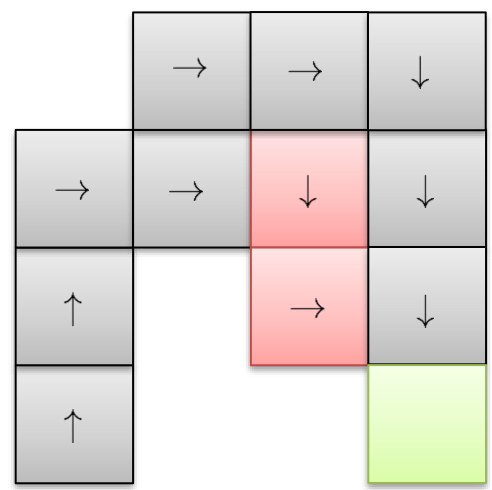
Policy improvement

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)**

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



$V^{\pi_1}(x)$

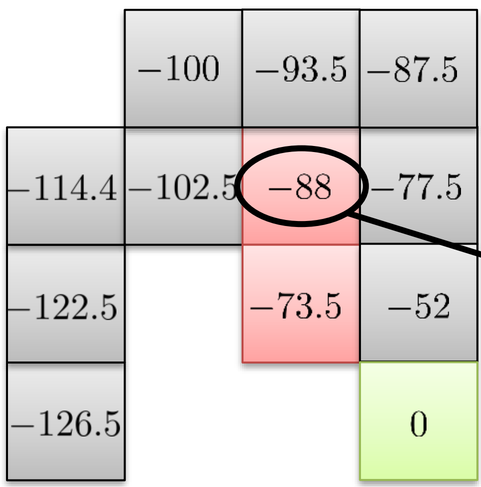


$\pi_2(x)$

2.3 Policy Iteration

Policy improvement

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



$V^{\pi_1}(x)$

$\gamma = 1$

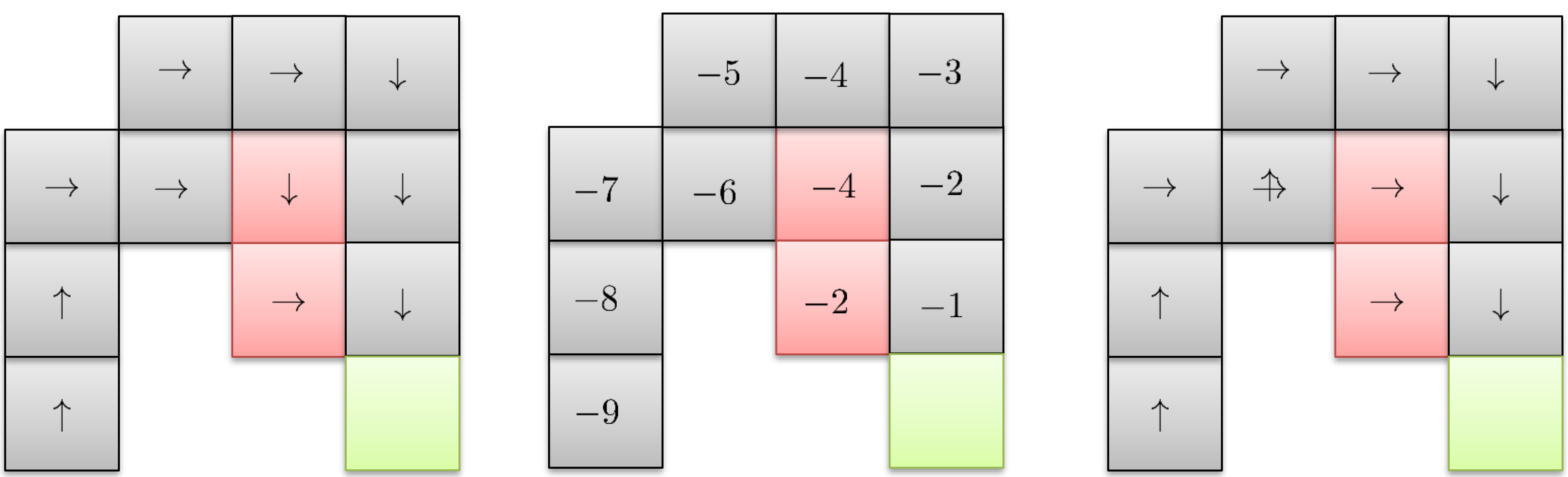
Right: $-1 + \gamma(-77{,}5) = -78{,}5$

Down: $-2 + \gamma(-73{,}5) = \boxed{-75{,}5}$

Left: $-1 + \gamma(-102{,}5) = -103{,}5$

Up: $-1 + \gamma(-93{,}5) = -94{,}5$

2.3 Policy Iteration



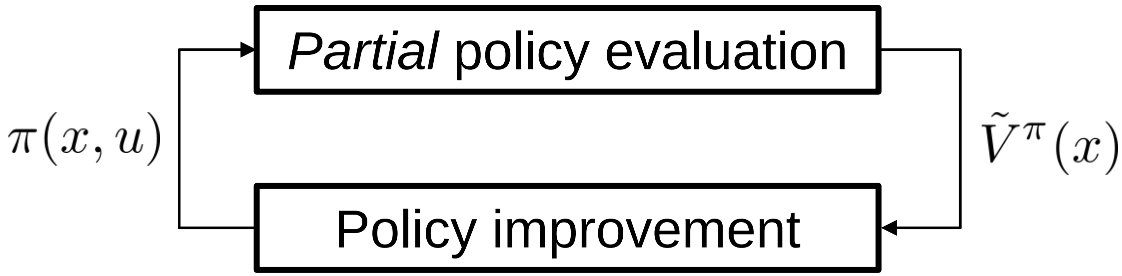
$\pi_2(x)$ $V^{\pi_2}(x)$ $\pi_3(x)$

2.3 Policy Iteration

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

2.3 Policy Iteration

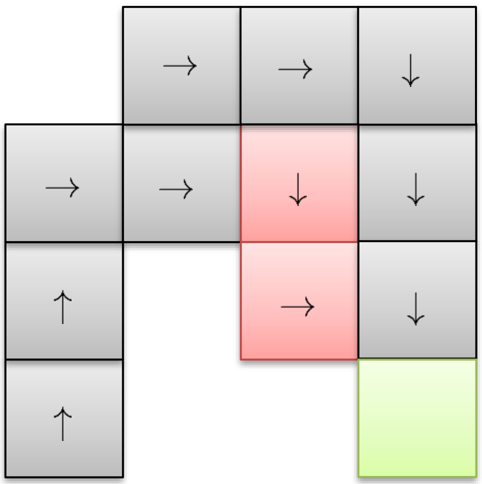
Reducing Computation Time: Generalized Policy iteration



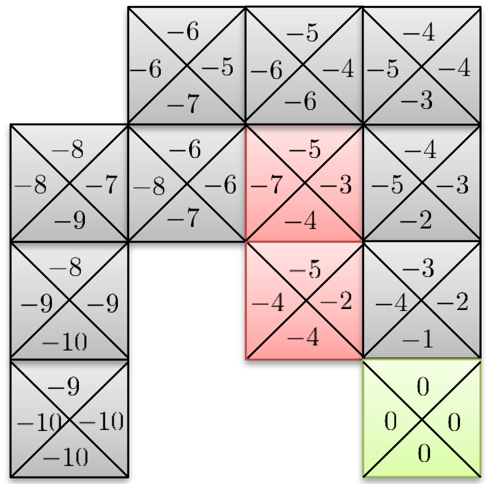
It is not always necessary to let the value function converge, improvements on the policy can be made earlier.

2.3 Policy Iteration

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

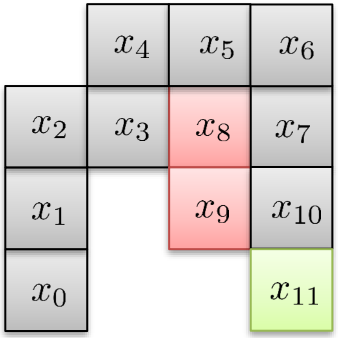
2.3 Policy Iteration

Why Action Value Function instead of Value Function?

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

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

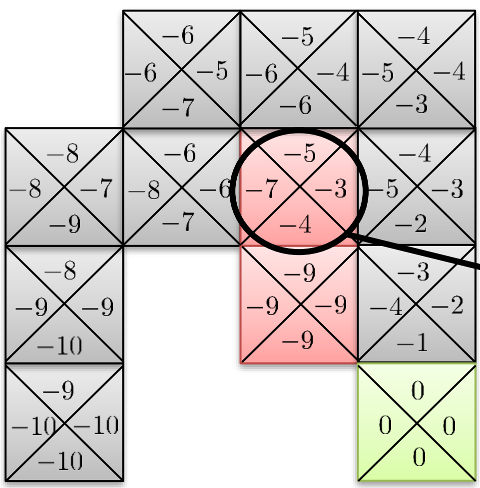
$$u_{\text{greedy}}(x) = \arg\max_u Q(x, u)$$



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

2.3 Policy Iteration

Grid World, Action Value Function



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

Right: $\boxed{-3}$

Down: -4

Left: -7

Up: -5

2.3 Policy Iteration

Short wrap up

- Using the Bellman equation, we can learn the value or action-value function. (e.g. iterative or system of equations)
  - We need to know the transition dynamics.
- Once we have value or action-value function, we can improve the policy
  - If we use the value function, we need to know the transition dynamics.
  - 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.

2.3 Policy Iteration

Model free learning

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

$$
\begin{aligned}
&\textbf{until convergence of } V\textbf{:}\\
&\quad \textbf{for all states } x\textbf{:}\\
&\quad\quad V_{k+1}^{\pi}(x) = \textstyle\sum_u \pi(u|x_t) \cdot (r_{t+1} + V_k^{\pi}(x_{t+1}))\\
&\quad \textbf{end}\\
&\textbf{end}
\end{aligned}
$$

- 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).

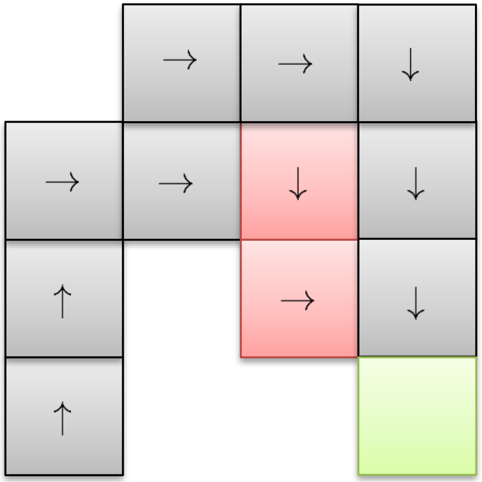
$$
\begin{aligned}
&\textbf{Iterate over all data tuples } (x_t, u_t, r_{t+1}, x_{t+1})\textbf{:}\\
&\quad V_{k+1}^{\pi}(x_t) = (1-\alpha) \cdot V_k^{\pi}(x_t) + \alpha \cdot (r_{t+1} + \gamma V_k^{\pi}(x_{t+1}))\\
&\quad \textbf{or}\\
&\quad Q_{k+1}^{\pi}(x_t, u_t) = (1-\alpha) \cdot Q_k^{\pi}(x_t, u_t) + \alpha \cdot (r_{t+1} + \gamma Q_k^{\pi}(x_{t+1}, u_{t+1}))\\
&\textbf{end}
\end{aligned}
$$

Do this if you want to improve later

2.3 Policy Iteration

Model free learning

- Necessary assumptions for convergence:
  1. **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!
  2. The learning rate is decreasing.
  3. We learn an infinite amount of time.

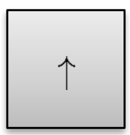


- In practice not as bad, the assumptions can be relaxed and results will still be good.

2.3 Policy Iteration

Model free learning

- How to handle the assumptions:
  1. 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.



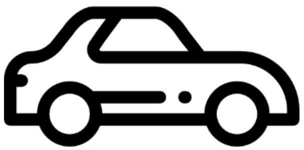
$$\pi(x, \uparrow) = 1 - \epsilon \ , \quad \pi(x, \leftarrow) = \pi(x, \downarrow) = \pi(x, \rightarrow) = \frac{\epsilon}{3}$$

  2. The learning rate **can be** reduced during training, but sometimes keeping it constant is enough. It‘s like with learning rates for NN.
  3. 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**.

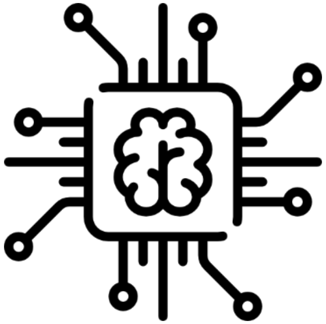
Reinforcement Learning
Maximilian Geißlinger / Fabian Netzler / Prof. Dr. Markus Lienkamp
(Levent Ögretmen, M. Sc.)

Agenda

1. Terminology and Concept
    - 1.1 Terminology and problem definition
    - 1.2 Basic probability theory for discrete variables



2. RL in discrete state- and action-spaces
    - 2.1 Markov decision processes
    - 2.2 Value and Action Value Function
    - 2.3 Policy Iteration
    - **2.4 Q-Learning**



2.4 Q-Learning

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

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

2.4 Q-Learning

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
```

2.4 Q-Learning

Example of discrete MDP's

Number of states $> 2 \cdot 10^{170}$!

Couple these methods with (Deep) Neural Networks

→ Approximate Q-function and/or policy!

Wrap Up

1. Learn value function or action-value function of current policy using the Bellman equation.
2. Use learned function 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

Wrap Up

What I expect you to know for the exam from chapter 2.2 to 2.4

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.