

*** Applied Machine Learning Fundamentals ***

Reinforcement Learning

M. Sc. Daniel Wehner

SAP SE

Winter term 2019/2020



Find all slides on [GitHub](#)

Lecture Overview

Out of scope for this lecture!

Agenda January 27, 2020

① Introduction

What is Reinforcement Learning?
MENACE (Matchbox Educable Noughts and
Crosses Engine)
Reinforcement Learning Formalization

② Algorithms

How to learn the optimal Policy?
Policy Iteration
Q-Learning
SARSA: State-Action-Reward-State-Action

③ Miscellaneous

Exploitation vs. Exploration
Non-Deterministic Rewards and Actions
Temporal Difference Learning
Deep Reinforcement Learning
Demo Applications

④ Wrap-Up

Final Example
Summary
Recommended Literature and further Reading
Meme of the Day

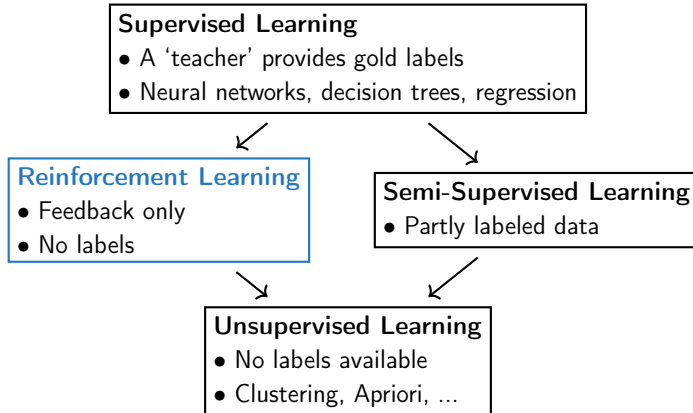
Section:
Introduction



What is Reinforcement Learning?

- Agent \Leftrightarrow (Unknown) Environment
- Goal
 - Learn optimal decisions based on feedback provided by the environment
 - Environment rewards agent for actions but **does not (!)** reveal the correct solution (reward can be positive or negative)
- Applications
 - Games, e. g.:
 - Tic-Tac-Toe: MENACE [*Michie*, 1963]
 - Backgammon: TD-Gammon [*Tesauro*, 1995]
 - Other: E. g. robot control (btw.: *robot_{en}* derived from *robot_{cz}* – labor service)

Dimensions of Learning: Type of Training Information



Difference to other Approaches

① Delayed rewards

- Sequence of actions \Rightarrow Problem of **temporal credit assignment**
- Which actions were good? Which were bad?

② Exploitation vs. exploration

- Should the agent perform actions which are known to be good?
- Or should it try out new (possibly even better) actions?

③ Partially observable states (not everything may be observable)

④ Life-long learning

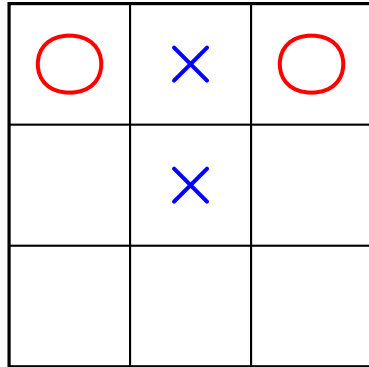
- Environment may not be static, it could change!
- Agents need to adapt to these changes...

Credit Assignment Problem

- **Fundamental problem in reinforcement learning**
 - Especially in games: Reward at the end of the match (have I won or lost?)
 - Central Question: **Which moves contributed to winning / losing?**
 - This problem is referred to as the **credit assignment problem**
- **Solution**
 - **All moves** of the sequence are rewarded / punished
 - The problem is basically ignored in this case
 - After many games the algorithm converges:
 - Bad moves will be reinforced less positively
 - Good moves will be positively reinforced more often

MENACE: Learning to play 'Tic-Tac-Toe'

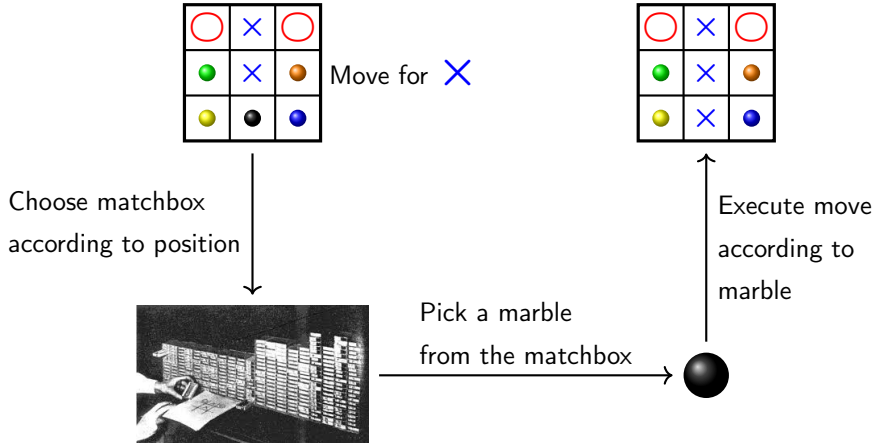
- Introduced by [*Mitchie*, 1963]
- Learns to play 'Tic-Tac-Toe'
- MENACE stands for:
 M
 E
 N
 A
 C
 E
 Matchbox
 Educable
 Noughts
 And
 Crosses
 Engine



MENACE: Setup

- Learns to play 'Tic-Tac-Toe' (reinforcement learning by hand)
- **'Hardware'**
 - 287 matchboxes (1 per playing position)
 - Marbles in nine different colors (one color per field)
 - Initially, each matchbox contains the same amount of marbles of each color
- **Algorithm**
 - Choose matchbox corresponding to current playing position
 - Pick a marble randomly from the matchbox
 - Put marker (× or ○) on the field that corresponds to the color
- **How can good moves be positively reinforced?**

MENACE: Visualized



Reinforcement Learning in MENACE

- **Initialization:** All moves are equally likely \Rightarrow Each matchbox contains the same amount of marbles of each color
- **Learning algorithm**
 - Game **lost**: Marble is removed (negative reinforcement)
 - Game **won**: Marble of corresponding color is added (positive reinforcement)
 - **Remis**: Marbles are put back into the matchbox (no change)

↑↑ Probability of successful moves is **increased**

↓↓ Probability of bad moves is **decreased**

Reinforcement Learning: Formalization

- Definitions
 - $s \in \mathcal{S}$: state space (discrete or continuous)
 - $a \in \mathcal{A}$: action space (discrete or continuous)
 - $s_0 \in \mathcal{S}_0 \subseteq \mathcal{S}$: initial states
 - $\delta : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$: state transition function (deterministic or stochastic)
 - $r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$: reward function
- **Markov property / Markov Decision Process (MDP)**
 - Rewards and state transitions depend on previous state only...
 - ...and not how you got into this state (earlier states and actions don't matter)

Reinforcement Learning: Formalization (Ctd.)

- The agent repeatedly chooses an action according to a **policy** π

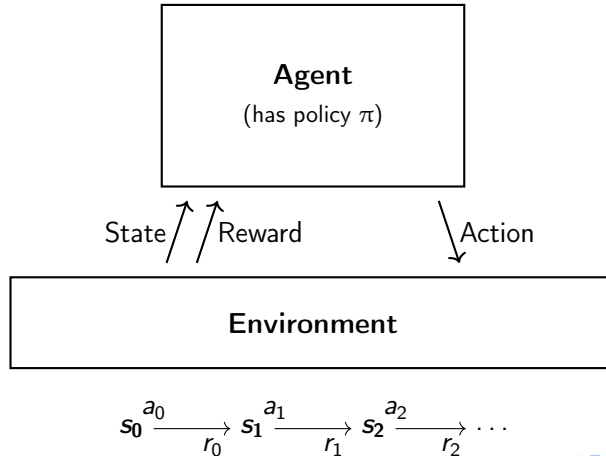
$$\pi(s) = a \quad (1)$$

- This leads the agent into a new state s'

$$s' = \delta(s, \pi(s)) \quad (2)$$

- The agent receives feedback (reinforcement) in some states
 - Not necessarily in all states
 - E. g. in a game there is usually no reward until the end (game is won or lost)

Reinforcement Learning: Formalization (Ctd.)



MENACE Formalization

States	Matchboxes (discrete)
Actions	Possible moves (discrete)
Policy	Prefer actions with higher number of marbles (stochastic)
Reward	Game won / game lost
Transition function	Choose next matchbox according to rules (deterministic)

Task: Find a policy π that maximizes the sum of future rewards ($\hat{= \pi^*}$).

Learning Task

- **Goal:** Maximize the **cumulative reward** for the **trajectory** τ which a policy π generates
 - **Deterministic:** τ is always the same
 - **Stochastic:** τ can differ (random elements included)
- Cumulative reward:

$$R(\pi) = R(\tau^\pi) = \sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \quad (3)$$

- **Discount factor** γ : Immediate rewards are weighted higher, rewards further in the future are discounted (exponentially)

Learning Task (Ctd.)

- Rewards t time steps in the future are discounted exponentially by a factor of γ^t
- Value of γ :
 - $0 \leq \gamma \leq 1$
 - $\gamma = 0$: Only immediate rewards are taken into account
 - $\gamma = 1$: Future rewards are given same emphasis as immediate rewards
- Usually, γ is set to a value close to 1 (0.99, 0.98, 0.95, ...)
- Immediate rewards are usually preferred

How to compute $R(\tau^\pi)$?

$$\begin{aligned}
 R(\tau^\pi) &= \sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \\
 &= r(s_0, a_0) + \gamma r(s_1, a_1) + \gamma^2 r(s_2, a_2) + \dots \\
 &= r(s_0, \pi(s_0)) + \sum_{t=1}^{\infty} \gamma^t r(\delta(s_{t-1}, \pi(s_{t-1})), \pi(s_t)) \\
 &= V^\pi(s_0)
 \end{aligned}$$

- V is called the ‘value’ of the first state
- **Value function:** Reward when starting in state s_0 and following policy π

Optimal Policies and Value Functions

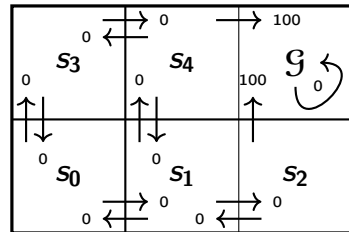
- The optimal policy is denoted by π^*
- The optimal policy has the highest expected value for all states:

$$\begin{aligned}\pi^*(s) &= \arg \max_{\pi} V^{\pi}(s) \quad \forall s \\ &= \arg \max_{a \in \mathcal{A}} r(s, a) + \gamma V^{\pi^*}(\delta(s, a)) \quad \forall s\end{aligned}$$

- Always select the action that maximizes the value function for the next step, when following π^* afterwards
- **Problem:** Recursive definition of π^* requires **perfect knowledge**

An illustrative Example (given perfect Knowledge)

- Each square represents a state
- Each arrow represents a possible action
- Small numbers refer to the reward $r(s, a)$ obtained by taking the action
- In this case: Reward is always set to 0 except when entering the goal state \mathcal{G}
- \mathcal{G} is an absorbing state (impossible to get out again)
- $\gamma = 0.9$



An illustrative Example [given perfect Knowledge] (Ctd.)

In this case we can derive the optimal policy π^* directly, e. g. $s_0 \rightarrow s_3 \rightarrow s_4 \rightarrow \mathcal{G}$

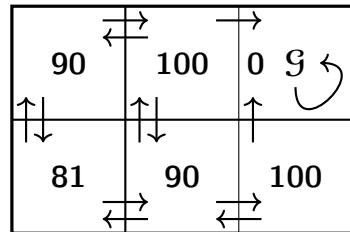
$$V^{\pi^*}(s_0) = 0 + \gamma 0 + \gamma^2 100 + \gamma^3 0 + \dots = \mathbf{81}$$

$$V^{\pi^*}(s_1) = 0 + \gamma 100 + \gamma^2 0 + \gamma^3 0 + \dots = \mathbf{90}$$

$$V^{\pi^*}(s_2) = 100 + \gamma 0 + \gamma^2 0 + \gamma^3 0 + \dots = \mathbf{100}$$

$$V^{\pi^*}(s_3) = 0 + \gamma 100 + \gamma^2 0 + \gamma^3 0 + \dots = \mathbf{90}$$

$$V^{\pi^*}(s_4) = 100 + \gamma 0 + \gamma^2 0 + \gamma^3 0 + \dots = \mathbf{100}$$



Section: Algorithms



How to learn the optimal Policy?

- W/o perfect knowledge it's difficult to learn the optimal policy directly, since training data of the form $\langle s, a \rangle$ is not available (\neq **supervised learning**)
- The only information at the agent's disposal is the sequence of observed immediate rewards:

$$r_0 \longrightarrow r_1 \longrightarrow r_2 \longrightarrow r_3 \longrightarrow r_4 \longrightarrow \dots$$

- What evaluation function should be learned?
 - One obvious choice is V^{π^*}
 - The agent should prefer state s_1 over s_2 if $V^{\pi^*}(s_1) > V^{\pi^*}(s_2)$

How to learn the optimal Policy? (Ctd.)

- The optimal action in state s is action a that maximizes the sum of the immediate reward $r(s, a)$, plus the value of V^{π^*} of the immediate successor, discounted by γ :

$$\pi^*(s) = \arg \max_{a \in \mathcal{A}} r(s, a) + \gamma V^{\pi^*}(\delta(s, a)) \quad (4)$$

- **Problem:**
 - In order to calculate V^{π^*} the agent needs to know $\delta(s, a)$ and $r(s, a)$
 - They may not be known... \Rightarrow Learning V^{π^*} is of no use, since the equation cannot be evaluated
- $\xRightarrow{\text{Remedy}}$ Policy Iteration, Q-Learning, SARSA, ...

Policy Iteration

- Initialize a policy randomly and find its value function and iterate
 - **Policy evaluation**: Find the value function of the current policy
 - **Policy improvement**: Find a better policy based on the current one
- **Policy improvement theorem**

If it is true that selecting the first action in each state according to a policy π' and continuing with policy π is better than always following π , then π' is a better policy than π . $\Rightarrow \pi^0(s) \rightarrow V^{\pi^0}(s) \rightarrow \pi^1(s) \rightarrow V^{\pi^1}(s) \rightarrow \dots \rightarrow \pi^*(s)$

- Select the action that maximizes the value function of the current policy:

$$\pi'(s) = \arg \max_{a \in \mathcal{A}} r(s, a) + \gamma V^{\pi}(\delta(s, a))$$

Q-Learning

- We define a new function $Q(s, a)$
- Its value is the maximum discounted cumulative reward that can be achieved starting from state s and applying action a as the first action

$$Q(s, a) = r(s, a) + \gamma V^{\pi^*}(\delta(s, a)) \quad \text{looks familiar?} \quad (5)$$

- Formula for π^* can be rewritten in terms of $Q(s, a)$:

$$\pi^*(s) = \arg \max_{a \in \mathcal{A}} Q(s, a) \quad (6)$$

- **This rewrite is important:** Learning the Q -function the agent can select optimal actions without any knowledge of r and δ

Q-Learning (Ctd.)

- By always choosing the action with the maximum Q -value...

$$\arg \max_{a \in \mathcal{A}} Q(s, a)$$

- ...we maximize the expected cumulative reward
- What may seem surprising: Local optimal choices lead to a global optimal solution

Learning the Q -function corresponds to learning the optimal policy π^*

Algorithm for Q-Learning

- How can Q be learned?
 - We learn the Q -function by iterative approximation
 - Note the close relationship of Q and V^{π^*} :

$$V^{\pi^*}(s) = \max_{a' \in \mathcal{A}} Q(s, a') \quad (7)$$

- Therefore, $Q(s, a)$ can be expressed in terms of itself (**Bellman equation**):

$$Q(s, a) = r(s, a) + \gamma \max_{a' \in \mathcal{A}} Q(\delta(s, a), a') \quad (8)$$

Algorithm for Q-Learning (Ctd.)

- \hat{Q} denotes the agent's current estimate of the true Q value (which is unknown)
- The Q -function is represented by a table
 - One entry per state-action pair
 - The entry for pair $\langle s, a \rangle$ stores $\hat{Q}(s, a)$
 - Initially, table is filled with random numbers or zeroes
- **Problem:** The table may become large if there are many states and actions

State-Action pair	\hat{Q} value
$\langle s_0, a_0 \rangle$	$\hat{Q}(s_0, a_0)$
$\langle s_0, a_1 \rangle$	$\hat{Q}(s_0, a_1)$
$\langle s_1, a_0 \rangle$	$\hat{Q}(s_1, a_0)$
$\langle s_1, a_1 \rangle$	$\hat{Q}(s_1, a_1)$
...	...

Algorithm for Q-Learning (Ctd.)

- The agent repeatedly observes the current state s , chooses some action a , executes it and observes r as well as the new state s'
- Update the table entry for $\hat{Q}(s, a)$ according to the Bellman equation
- The agent doesn't have to know $\delta(s, a)$ and $r(s, a)$
- Instead, it executes some actions and observes what happens
- In the limit \hat{Q} will converge towards the actual Q -function **iff...**
 - ...the system can be modeled as a deterministic MDP...
 - ...**and** the reward function is bounded...
 - ...**and** each state-action pair is visited infinitely often

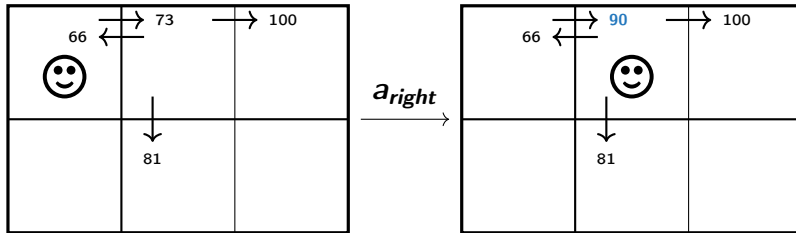
Algorithm for Q-Learning (Ctd.)

Algorithm 1: Learning the Q-function

```

1 foreach  $\langle s, a \rangle$  do // initialize table entries with zeroes
2    $\hat{Q}(s, a) \leftarrow 0$  // Alternative: Initialize with random numbers
3 while true do // endless loop
4   Observe current state  $s$ 
5   Select an action  $a$  and execute it
6   Receive the immediate reward  $r$ 
7   Observe the new state  $s'$ 
8   Update the table entry for  $\hat{Q}(s, a)$ :  $\hat{Q}(s, a) \leftarrow r + \gamma \max_{a' \in \mathcal{A}} \hat{Q}(s', a')$ 
9    $s \leftarrow s'$ 
  
```

Q-Learning: An illustrative Example

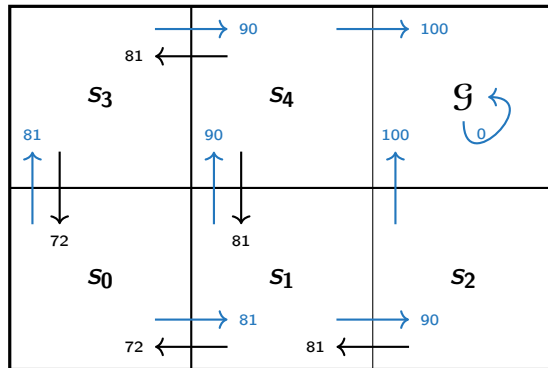


$$\begin{aligned}
 \hat{Q}(s_3, a_{right}) &\leftarrow r + \gamma \max_{a' \in \mathcal{A}} \hat{Q}(s_4, a') \\
 &\leftarrow 0 + 0.9 \max_{a' \in \mathcal{A}} \{66, 81, 100\} \\
 &\leftarrow 90
 \end{aligned}$$

Q-Learning: An illustrative Example (Ctd.)

- Each time the agent makes a move, the Q -value estimates are **propagated backwards** from the new state s' to the old state s
- The immediate reward is used to **augment** the propagated values of \hat{Q}
- Since there is an absorbing goal state \mathcal{G} , there will be a series of episodes
- **One episode:**
 - ① Initialize the agent at a random state
 - ② The episode ends as soon as the goal state \mathcal{G} is reached
 - ③ Repeat until convergence (goto 1)

Q-Learning: An illustrative Example (Ctd.)



SARSA: State-Action-Reward-State-Action

- SARSA is very similar to Q-learning
- Stands for 'State-Action-Reward-State-Action'
- It refrains from calculating the 'max' in the formula
- Instead it uses the current policy (\Rightarrow **on-policy update**)

$$\hat{Q}(s, a) \leftarrow r(s, a) + \gamma \underbrace{\hat{Q}(\delta(s, a), \pi(s'))}_{=s'} \quad (9)$$

Section:
Miscellaneous



Exploitation vs. Exploration

- By always picking the best action the agent runs the risk of overcommitting to actions that were found during early training
- It fails to explore other actions that might be even better
- **EXPLOITATION \Leftrightarrow EXPLORATION**
- Exploitation
 - Use the action the agent assumes to be the best one
 - Approximate the optimal policy
- Exploration
 - 'Optimal action' may be wrong due to approximation errors
 - Try a suboptimal one

Exploitation vs. Exploration (Ctd.)

- **ϵ -greedy** (small $\epsilon \Rightarrow$ exploitation)

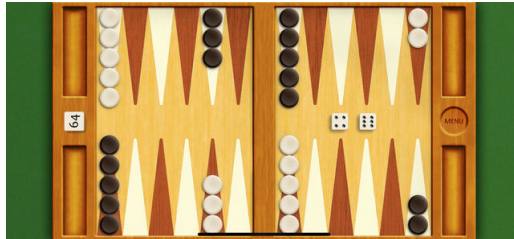
$$p(a_i|s) = \begin{cases} 1 - \epsilon + \frac{\epsilon}{|\mathcal{A}|} & \text{if } a_i = \arg \max_{a \in \mathcal{A}} \hat{Q}(s, a) \\ \frac{\epsilon}{|\mathcal{A}|} & \text{otherwise} \end{cases} \quad (10)$$

- **Softmax** (actions with high \hat{Q} -values get a higher probability)

$$P(a_i|s) = \frac{k^{\hat{Q}(s, a_i)}}{\sum_j k^{\hat{Q}(s, a_j)}} \quad k > 0 \quad (\text{large } k \Rightarrow \text{exploitation}) \quad (11)$$

Non-Deterministic Rewards and Actions

- So far we only considered the deterministic case
- **What if the environment is non-deterministic?**
 - E. g. in Backgammon each move involves a roll of the dice $\Rightarrow \delta(s, a)$ varies
 - Also, the rewards may change $\Rightarrow r(s, a)$ varies



Non-Deterministic Rewards and Actions (Ctd.)

- $\delta(s, a)$ and $r(s, a)$ can be viewed as first producing a probability distribution over the outcomes based on s and a , and then drawing an outcome at random according to this distribution
- If these probability distributions depend solely on s and a (not on earlier states or actions), then the system is called **non-deterministic Markov decision process**

The Q -learning algorithm has to be modified in order to be able to handle the non-deterministic case.

Non-Deterministic Rewards and Actions (Ctd.)

- Restate the agent's objective \Rightarrow Redefine the value V^π of a policy π to be the expected value (solve e. g. by using **Monte Carlo sampling**, choose k):

$$V^\pi(s_0) = \mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, \pi(s_t)) \right] = r(s_0, \pi(s_0)) + \underbrace{\frac{1}{k} \sum_{i=0}^k \sum_{t=1}^{\infty} \gamma^t r(s_t^i, \pi(s_t^i))}_{k \text{ samples}}$$

- The definition of Q has to be updated as well:

$$\begin{aligned} Q(s, a) &= \mathbb{E}[r(s, a) + \gamma V^{\pi^*}(\delta(s, a))] = \mathbb{E}[r(s, a)] + \gamma \mathbb{E}[V^{\pi^*}(\delta(s, a))] \\ &= \mathbb{E}[r(s, a)] + \gamma \sum_{s'} p(s'|s, a) V^{\pi^*}(s') \end{aligned}$$

Non-Deterministic Rewards and Actions (Ctd.)

- We can again re-express Q recursively:

$$Q(s, a) = \mathbb{E}[r(s, a)] + \gamma \sum_{s'} p(s'|s, a) \max_{a'} Q(s', a') \quad (12)$$

- We need a new training rule for the algorithm (the old one fails to converge due to changing rewards...)

$$\hat{Q}_n(s, a) \leftarrow (1 - \alpha_n) \hat{Q}_{n-1}(s, a) + \alpha_n [r + \gamma \max_{a'} \hat{Q}_{n-1}(s', a')] \quad (13)$$

- Where $\alpha_n = \frac{1}{1 + \text{visits}_n(s, a)}$ ($\text{visits}_n(s, a)$ is the number of times the state-action pair has been visited)

Non-Deterministic Rewards and Actions (Ctd.)

- Updates of \hat{Q} are made more gradually than in the deterministic case (average)
- For $\alpha_n = 1$ we get the old training rule
- α_n decreases over time \Rightarrow Updates become smaller in later iterations
- By using α the algorithm ultimately converges
- Q-learning often requires many thousands of training iterations to converge

E. g. TD-Gammon trained for 1.5 million (!) Backgammon games.

Temporal Difference Learning

- Q-learning works by iteratively reducing the discrepancy between Q-value estimates at different time steps (based on a one-step lookahead)
- Q-learning is a special case of **temporal difference learning**

$$\hat{Q}_n(s, a) \leftarrow (1 - \alpha_n) \hat{Q}_{n-1}(s, a) + \alpha_n [r(s, a) + \gamma \max_{a'} \hat{Q}_{n-1}(s', a')]$$

↕ **rewrite the equation**

$$\hat{Q}_n(s, a) \leftarrow \hat{Q}_{n-1}(s, a) + \alpha_n \underbrace{\left[\overbrace{(r(s, a) + \gamma \max_{a'} \hat{Q}_{n-1}(s', a'))}^{\text{after}} - \overbrace{\hat{Q}_{n-1}(s, a)}^{\text{before}} \right]}_{\text{Temporal difference}}$$

Temporal Difference Learning (Ctd.)

- Q-learning performs a one-step lookahead (can be generalized to n steps)
- $Q^{(1)}(s_t, a_t)$ denotes the training value calculated by this one-step lookahead:

$$Q^{(1)}(s_t, a_t) = r_t + \gamma \max_a \hat{Q}(s_{t+1}, a) \quad (14)$$

- For two steps:

$$Q^{(2)}(s_t, a_t) = r_t + \gamma r_{t+1} + \gamma^2 \max_a \hat{Q}(s_{t+2}, a) \quad (15)$$

- For n steps:

$$Q^{(n)}(s_t, a_t) = r_t + \gamma r_{t+1} + \cdots + \gamma^{(n-1)} r_{t+n-1} + \gamma^n \max_a \hat{Q}(s_{t+n}, a) \quad (16)$$

Temporal Difference Learning: $TD(\lambda)$ [Sutton, 1988]

- $TD(\lambda)$ is a method for blending these alternative training estimates
- Introduce a constant λ ($0 \leq \lambda \leq 1$) to combine the estimates from various lookaheads (recursive definition):

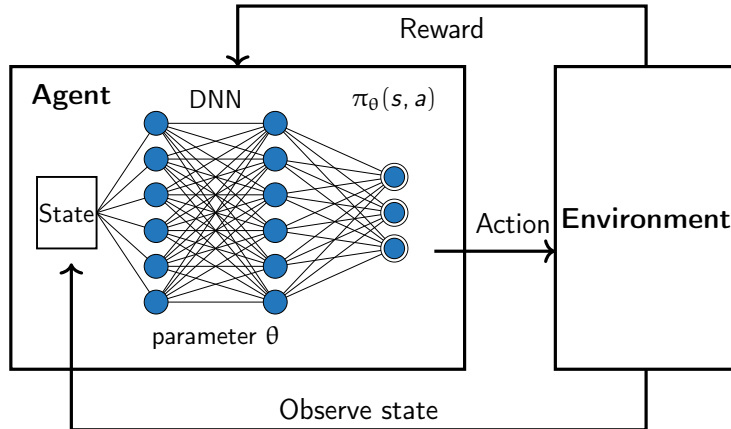
$$Q^\lambda(s_t, a_t) = r_t + \gamma \left[(1 - \lambda) \max_a \hat{Q}(s_t, a_t) + \lambda Q^\lambda(s_{t+1}, a_{t+1}) \right]$$

- If $\lambda = 0$ we get the original training estimate $Q^{(1)}$ (only one-step lookahead)
- If $\lambda = 1$ only the observed rewards r_{t+i} are taken into account
- Increasing values of λ put more emphasis on discrepancies based on more distant lookaheads

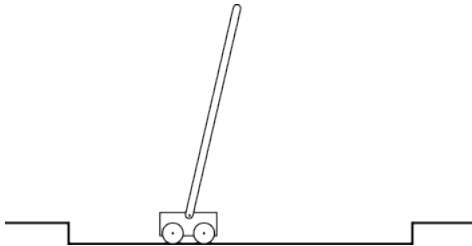
Deep Reinforcement Learning

- As already mentioned: The Q -learning table can become really large if the state space and the action space are really large
- It is not viable to maintain a table of all entries
- Maybe it does not even fit into memory...
- An alternative is needed!
- **Idea:**
 - Use deep learning methods to replace the table
 - E. g. a deep MLP could predict the action to take given the observed state, the reward, ...

Deep Reinforcement Learning (Ctd.)



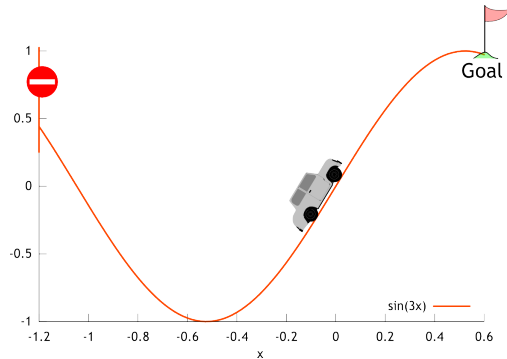
Demo: Cartpole Task



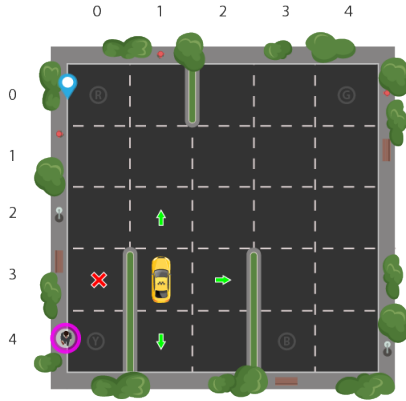
- Move the cart **without** the pole falling over
- Action space \mathcal{A} :
 - Move *LEFT* (0)
 - Move *RIGHT* (1)
- 4D state space \mathcal{S} :
 - Cart position, velocity
 - Angle of pole to cart...
 - ...and its derivative

Demo: Mountain Car Task

- The engine of the car is not strong enough
- You have to go back and forth to acquire momentum
- The car must not hit the wall (left side)
- The car has to reach the top of the hill



Demo: Taxi Task



- Pick up passengers and get them to the drop-off location
- Filled square represents the taxi
- '|' represents a wall; you should not go there ;-)
- Blue letter: **Pick-up location**
- Purple letter: **Drop-off location**
- Taxi turns **green** if a passenger is aboard

Section:
Wrap-Up



Final Example

An agent can move in a simple deterministic world which looks as following:

a	b	c
d	e	f

The agent can move up, down, left or right if the field exists. The state 'f' is an absorbing goal state. If the agent reaches 'f' it receives a reward of one point, otherwise 0 points. The discount factor γ be set to 0.8.

Final Example (Ctd.)

a) Specify the reward function $r(s, a)$ for all state-action pairs.

$r(a, r) = 0$	$r(b, r) = 0$	$r(c, u) = 1$	$r(d, o) = 0$	$r(e, o) = 0$
$r(a, u) = 0$	$r(b, u) = 0$	$r(c, l) = 0$	$r(d, r) = 0$	$r(e, r) = 1$
	$r(b, l) = 0$			$r(e, l) = 0$

Final Example (Ctd.)

b) Calculate the value function $V^\pi(s)$ for all states s given policy π :

→	→	↓
↑	↑	

$$\begin{aligned}
 V^\pi(d) &= \gamma^0 r(d, o) + \gamma^1 r(a, r) + \gamma^2 r(b, r) + \gamma^3 r(c, u) \\
 &= 1 \cdot 0 + 0.8 \cdot 0 + 0.8^2 \cdot 0 + 0.8^3 \cdot 1 = \mathbf{0.512}
 \end{aligned}$$

$V^\pi(a) = 0.640$	$V^\pi(b) = 0.800$	$V^\pi(c) = 1.000$
$V^\pi(d) = 0.512$	$V^\pi(e) = 0.640$	

Final Example (Ctd.)

c) How would **Policy Improvement** change the policy π from task b) for **e**?

Policy Improvement: $\pi(s) = \arg \max_{a \in \mathcal{A}} r(s, a) + \gamma V^\pi(\delta(s, a))$

up: 0.64 (see above)

left: $r(e, l) + \gamma V^\pi(d) = 0 + 0.8 \cdot 0.512 = 0.4096$

right: $r(e, r) = 1 \Leftarrow$ **We would take this one!**

$\Rightarrow \pi(e) = \textit{right}$

Final Example (Ctd.)

d) Think of an optimal way to reach the goal state for each s . Calculate $V^{\pi^*}(s)$.

Optimal routes:

$a \longrightarrow b \longrightarrow c \longrightarrow f$

$a \longrightarrow d \longrightarrow e \longrightarrow f$

$V^{\pi^*}(a) = 0.640$	$V^{\pi^*}(b) = 0.800$	$V^{\pi^*}(c) = 1.000$
$V^{\pi^*}(d) = 0.800$	$V^{\pi^*}(e) = 1.000$	

Final Example (Ctd.)

e) Calculate the optimal Q -values for all possible state-action pairs.

Hint: Using the results from d): $Q(s, a) = r(s, a) + \gamma V^{\pi^*}(s')$

For field **a** we get:

$$Q(a, r) = r(a, r) + \gamma V^{\pi^*}(b) = 0 + 0.8 \cdot 0.8 = 0.64 = Q(a, u)$$

$Q(a, r) = 0.640$	$Q(b, r) = 0.800$	$Q(c, u) = 1.000$	$Q(d, o) = 0.512$	$Q(e, o) = 0.640$
$Q(a, u) = 0.640$	$Q(b, u) = 0.800$	$Q(c, l) = 0.640$	$Q(d, r) = 0.800$	$Q(e, r) = 1.000$
	$Q(b, l) = 0.512$			$Q(e, l) = 0.640$

Summary

- What is reinforcement learning?
 - Learn optimal decisions based on feedback
 - Based on the feedback, the agent only knows how good the action was but not the optimal solution
 - **Credit assignment problem**: Which move(s) is / are responsible for success / failure?
 - Example: MENACE
- Important terms:
 - State s , action a
 - Policy π (*optimal policy* π^*)
 - Reward r (*immediate reward, discounted reward*)

Summary (Ctd.)

- Algorithms
 - Policy iteration
 - Q-learning
 - SARSA
- Exploitation vs. Exploration
 - **Exploitation**: Exploit what is already known to be good
 - **Exploration**: But don't miss to explore new states and actions
- Non-deterministic case ($\delta(s, a)$ and $r(s, a)$ vary)
- Temporal difference learning
- Deep reinforcement learning (using deep learning methods)

Recommended Literature and further Reading I



[1] Machine Learning

Tom Mitchell. McGraw-Hill Science. 1997.

→ [Click here](#), cf. chapter 13



[2] Solving an MDP with Q -Learning from scratch

Venelin Valkov. 2017.

→ [Click here](#)



[3] Hands-On Machine Learning with Scikit-Learn & TensorFlow

Aurélien Géron. O'Reilly. 2017.

→ [Click here](#), cf. chapter 16

Recommended Literature and further Reading II



[4] Machine Learning; An algorithmic perspective

S. Marsland. CRC Press. Chapman & Hall. 2015.

→ [Click here](#), cf. chapter 17



[5] Learning to predict by the methods of temporal differences

R. Sutton. Kluwer Academic Publishers. 1988.

→ [Click here](#)



[6] Machine Learning Lecture

A. Ng. Stanford University. 2008.

→ [Click here](#) (Lecture 16), [Click here](#) (Lecture 17),
[Click here](#) (Lecture 18), [Click here](#) (Lecture 19), [Click here](#) (Lecture 20)

Recommended Literature and further Reading III



[7] OpenAI Gym

OpenAI. 2018.

→ [Click here](#)



[8] Getting started with reinforcement Q-learning

P. Jaiswal. Towards Data Science. 2017.

→ [Click here](#)

Meme of the Day



Thank you very much for the attention!

Topic: *** Applied Machine Learning Fundamentals *** Reinforcement Learning

Term: Winter term 2019/2020

Contact:

M.Sc. Daniel Wehner

SAP SE

daniel.wehner@sap.com

Do you have any questions?