* * * Artificial Intelligence and Machine Learning * * *

Introduction to Reinforcement Learning

Daniel Wehner, M.Sc.

SAP SE / DHBW Mannheim

Summer term 2025

Find all slides on GitHub (DaWe1992/Applied_ML_Fundamentals)

Lecture Overview

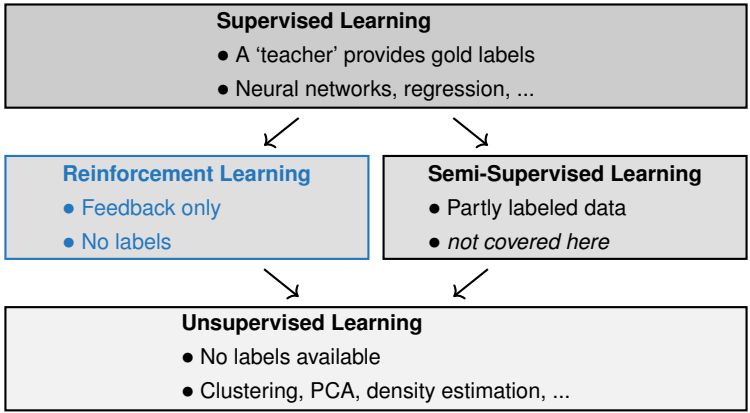
Agenda for this Unit

Section:

Introduction to Reinforcement Learning

What is Reinforcement Learning?
Key Challenges in Reinforcement Learning
MENACE (Matchbox Educable Noughts and Crosses Engine)
Formalization of Reinforcement Learning

Dimensions of Learning: Type of Training Information



What is Reinforcement Learning?

- Agent versus (unknown) environment
- The goal is to learn **optimal decisions based on feedback** provided by the environment
- The environment **rewards** the agent for its actions, but **does not** reveal the correct solution *(rewards can be positive or negative)*
- **Applications**
  - Games
    - Tic-Tac-Toe: MENACE [MICHIE.1963]
    - Backgammon: TD-Gammon [TESAURO.1995]
  - Robot control (*robot*$_{en}$ derived from *robota*$_{cz}$ which translates to 'labor service')

Reward Hypothesis

- **Fundamental assumption:** All goals of an agent can be explained by a **single scalar, called the reward**
- It is the RL practitioners' task to find the right set of rewards from which the agent learns the desired behavior
- This is referred to as **reward shaping**
  (comparable to feature engineering in data science)

Reward hypothesis: What we mean by goals and purposes can be well thought of as the **maximization of the expected value of the cumulative sum of a received scalar signal** *(called reward)*.

Digression into Psychology: Operant Conditioning (B. F. SKINNER)

Reinforcement learning is comparable to operant conditioning *(e. g. when training a dog)*:

	Punishment	**Reinforcement**
Positive	**add** noxious stimulus to decrease behavior	**add** pleasant stimulus to increase behavior
Negative	**remove** pleasant stimulus to decrease behavior	**remove** noxious stimulus to increase behavior

Key Challenges in Reinforcement Learning

1. **Delayed rewards**
   - Usually, there is a sequence of actions, before a reward is given
   - Problem of **temporal credit assignment**: Which actions were good / bad?
2. **Exploitation versus 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**
   - The environment may not be static, it could change!
   - The agent needs to adapt to these changes...

Credit Assignment Problem

Fundamental problem in reinforcement learning:

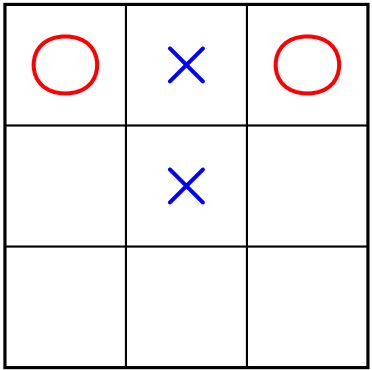
- Especially in games: The reward is given at the end of the match *(have I won or lost?)*
- **Central Question:** Which moves contributed to winning or losing?
- This problem is referred to as the **credit assignment problem**

Simple solution: All moves of the sequence are rewarded or punished equally. After many matches, 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]
- The agent learns to play ‘Tic-Tac-Toe’
- MENACE is short for:

  Matchbox
  Educable
  Noughts
  And
  Crosses
  Engine



MENACE: Setup

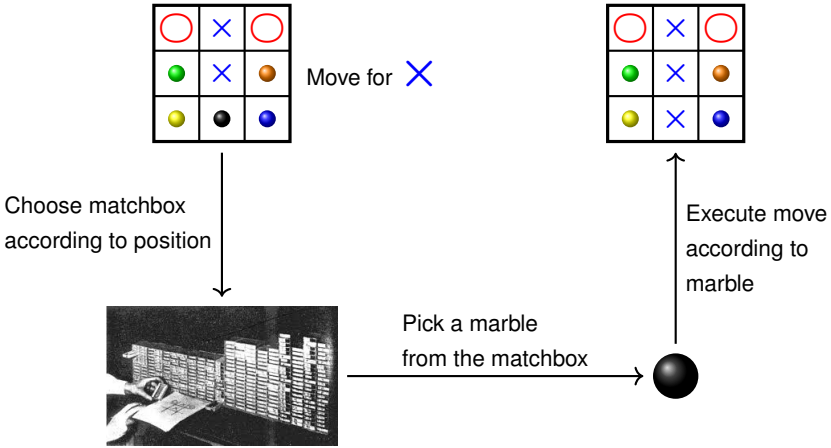
'Hardware':

- 287 matchboxes *(one per playing position)*
- Marbles in nine different colors *(one color per field)*
- Initially, each matchbox contains the same amount of marbles of each color

How to perform a move?

- Choose matchbox corresponding to current playing position
- Pick a marble randomly from the matchbox
- Put marker ($\times$ or $\bigcirc$) on the field that corresponds to its color

MENACE: Visualized



Reinforcement Learning in MENACE

- **Initialization:** Each matchbox contains the same amount of marbles of each color *(all moves are equally likely)*
- **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)*

Result:

Probability of successful moves is **increased** ⇑

Probability of bad moves is **decreased** ⇓

State and Action Spaces, Transition and Reward Functions

Definitions and Notation:

$s \in \mathcal{S}$	State space *(discrete or continuous)*; set of possible states the agent can be in
$s_0 \in \mathcal{S}_0 \subseteq \mathcal{S}$	Set of initial states *('where the agent can be spawned')*
$a \in \mathcal{A}$	Action space *(discrete or continuous)*; set of possible actions
$\delta : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$	State transition function *(deterministic or stochastic)*
$r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$	Reward function

MARKOV property / MARKOV decision process (MDP): Rewards and state transitions depend on the previous state only, and not how you got into this state *(earlier states and actions do not matter)*.

The Policy ϖ

- Based on the current state s, the agent chooses an action a according to a **policy** ϖ *(varpi)*:

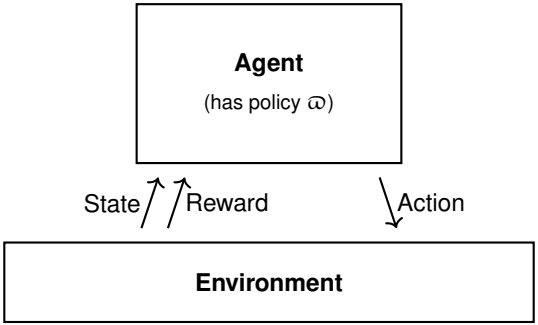
$$\varpi : \mathcal{A} \to \mathcal{A}, \qquad a = \varpi(s) \tag{1}$$

- The agent reaches a new state s' via the state transition function δ:

$$s' = \delta\big(s, \varpi(s)\big) \tag{2}$$

- The policy ϖ is learned by **maximizing the cumulative reward**
- The agent receives a reward in some states *(not necessarily in all states; see temporal credit assignment problem)*

Reinforcement Learning: Formalization (Ctd.)



Learning Task

Goal: Find a policy $\varpi^\star$ which maximizes the sum of future rewards

- Maximize the **cumulative reward** R for the **trajectory** τ^ϖ a policy ϖ generates
  - **Stochastic environment:** τ can differ *(random elements included)*
  - **Deterministic environment:** τ is always the same *(for same initial state)*

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

- The **discount factor** γ gives more weight to immediate rewards, and exponentially less weight to rewards further in the future

Discount Factor γ

- According to equation (3), rewards t time steps in the future are discounted exponentially by a factor of γ^t
- This means we **prefer immediate rewards** over rewards in the distant future
- **Choice of γ:**
  - Possible range: $0 \leqslant \gamma \leqslant 1$
  - $\gamma = 0$: Only immediate rewards are taken into account
    (we define $0^0 := 1$ *in this case)*
  - $\gamma = 1$: Future and immediate rewards are given the same weight
  - Usually, γ is set to a value close to 1, e. g.: 0.99, 0.98, 0.95, ...

Value Function: Value V of a State

$$
\begin{aligned}
R(\tau^{\omega}) &= \sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \\
&= r(s_0, a_0) + \gamma r(s_1, a_1) + \gamma^2 r(s_2, a_2) + \dots \\
&= r\big(s_0, \omega(s_0)\big) + \sum_{t=1}^{\infty} \gamma^t r\Big(\delta\big(s_{t-1}, \omega(s_{t-1})\big), \omega(s_t)\Big) \\
&=: V^{\omega}(s_0) \qquad (4)
\end{aligned}
$$

$V^{\omega}(s_0)$ is called the 'value' of the first state s_0. It represents the cumulative reward obtained when starting in state s_0 and then following policy ω

Optimal Policies and Value Functions

- The optimal policy is denoted by $\varpi^\star$
- The optimal policy has the **highest expected value for all states**:

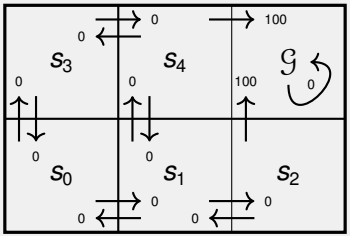
$$\begin{aligned} \varpi^\star(s) &:= \arg\max_\varpi V^\varpi(s) && \forall s \in \mathcal{S} \\ &= \arg\max_{a \in \mathcal{A}} r(s,a) + \gamma V^{\varpi^\star}\big(\delta(s,a)\big) && \forall s \in \mathcal{S} \end{aligned} \tag{5}$$

- Always select the action that maximizes the value function for the next step, when following $\varpi^\star$ afterwards

Problem: Recursive definition of $\varpi^\star$ 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 *(i. e. it is impossible to get out again)*
- $\gamma := 0.9$



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

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

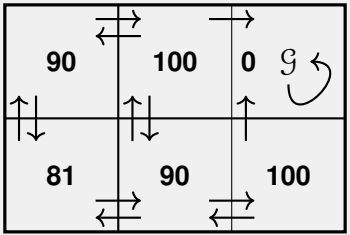
$$V^{\varpi^\star}(s_0) = 0 + \gamma 0 + \gamma^2 100 + \gamma^3 0 + \ldots = \mathbf{81}$$

$$V^{\varpi^\star}(s_1) = 0 + \gamma 100 + \gamma^2 0 + \gamma^3 0 + \ldots = \mathbf{90}$$

$$V^{\varpi^\star}(s_2) = 100 + \gamma 0 + \gamma^2 0 + \gamma^3 0 + \ldots = \mathbf{100}$$

$$V^{\varpi^\star}(s_3) = 0 + \gamma 100 + \gamma^2 0 + \gamma^3 0 + \ldots = \mathbf{90}$$

$$V^{\varpi^\star}(s_4) = 100 + \gamma 0 + \gamma^2 0 + \gamma^3 0 + \ldots = \mathbf{100}$$



Section:

Dynamic Programming Techniques

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

How to learn the optimal Policy?

- **Without perfect knowledge** it is difficult to learn the optimal policy directly, since training data of the form (s, a) is **not available** ($\neq$ supervised learning)
- The only information at the agent's disposal is the **sequence of observed 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 $V^{\omega^\star}$
  - The agent should prefer state s_1 over s_2, if $V^{\omega^\star}(s_1) > V^{\omega^\star}(s_2)$

How to learn the optimal Policy? (Ctd.)

Problem:

- In order to calculate $V^{\varpi^\star}$, the agent needs to know $\delta(s, a)$ and $r(s, a)$
- Basically, $\varpi^\star$ must be known in advace *(deadlock)*

Possible solutions: Use **dynamic programming** techniques like

- Value iteration, policy iteration
- Q-Learning, SARSA

Value Iteration Algorithm

Input: Environment $\mathcal{E} = (\mathcal{S}, \mathcal{A})$, threshold ε

Initialize $V(s) \in \mathbb{R}$ arbitrarily for all $s \in \mathcal{S}$
while $\Delta > \varepsilon$ **do**
  $\Delta \longleftarrow 0$
  foreach $s \in \mathcal{S}$ **do**
    $v \longleftarrow V(s)$ and $V(s) \longleftarrow \max_a \sum_{s',r} p(s', r|s, a)\big[r + \gamma V(s')\big]$
    $\Delta \longleftarrow \max\big(\Delta, |v - V(s)|\big)$
  end
end
return $\varpi^\star$ *such that* $\varpi^\star(s) = \arg\max_a \sum_{s',r} p(s', r|s, a)\big[r + \gamma V(s')\big] \quad \forall s \in \mathcal{S}$

Policy Iteration

- **Basic idea:** Initialize a policy randomly, find its value function, update and iterate
  1. **Policy evaluation**: Find the value function V^{ϖ} of the current policy ϖ
  2. **Policy improvement**: Find a better policy ϖ' based on the current one
- Select the action that **maximizes the value function** of the current policy ϖ:

$$\varpi'(s) = \arg\max_{a \in \mathcal{A}} r(s,a) + \gamma V^{\varpi}\big(\delta(s,a)\big) \tag{6}$$

- This leads to a sequence of **improving policies**:

$$\varpi^0(s) \longrightarrow V^{\varpi^0}(s) \longrightarrow \varpi^1(s) \longrightarrow V^{\varpi^1}(s) \longrightarrow \cdots \longrightarrow \varpi^\star(s)$$

Policy Improvement Theorem

Very important:

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.

Policy Iteration Algorithm (deterministic Case)

Input: Environment $\mathcal{E} = (\mathcal{S}, \mathcal{A})$

1 Initialize $V(s) \in \mathbb{R}$ and $\varpi(s) \in \mathcal{A}$ arbitrarily for all $s \in \mathcal{S}$
2 `policy_stable` $\longleftarrow$ *false*
3 **while** $\neg$`policy_stable` **do**
4 $\quad V_{new} \longleftarrow$ *PolicyEvaluation*$(\varpi, V, \varepsilon = 0.001)$
5 $\quad \varpi$, `policy_stable` $\longleftarrow$ *PolicyImprovement*(V_{new})
6 $\quad V \longleftarrow V_{new}$
7 **end**
8 **return** $\varpi^{\star}$

Sub-Algorithm: PolicyEvaluation

Input: Current policy ϖ, current value function V, threshold ε

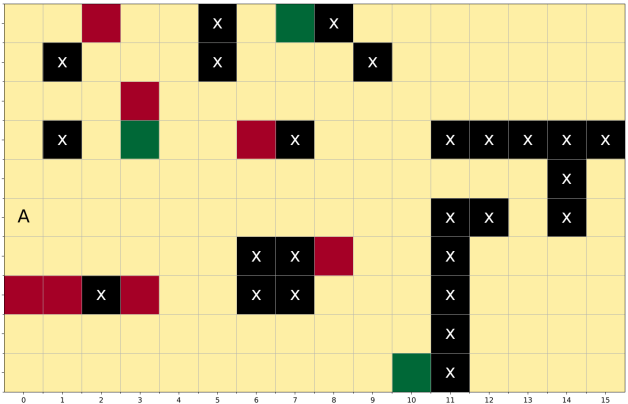
while $\Delta > \varepsilon$ **do**
  $\Delta \longleftarrow 0$
  foreach $s \in \mathcal{S}$ **do**
    $v \longleftarrow V(s)$
    $V(s) \longleftarrow \sum_{s',r} p\big(s', r|s, \varpi(s)\big)\big[r + \gamma V^{\varpi}(s')\big]$
    $\Delta \longleftarrow \max\big(\Delta, |v - V(s)|\big)$
  end
end
return V

Sub-Algorithm: PolicyImprovement

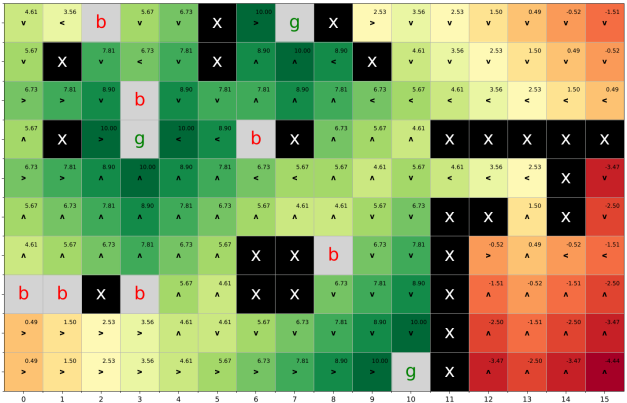
Input: Updated value function V

```
policy_stable ⟵ true
foreach s ∈ 𝒮 do
    old_action ⟵ ϖ(s)
    ϖ(s) ⟵ arg max_a Σ_{s',r} p(s', r|s, a) [r + γ V^ϖ(s')]
    if old_action ≠ ϖ(s) then
        policy_stable ⟵ false
    end
end
return ϖ, policy_stable
```

Policy Iteration Example: Grid World



Policy Iteration Example: Grid World (solved)



Q-Learning

- We introduce a function $Q(s, a)$ which represents the **maximum discounted cumulative reward** that can be achieved starting from state s and performing action a as the first action:

$$Q(s, a) := r(s, a) + \gamma V^{\varpi^\star}\big(\delta(s, a)\big) \tag{7}$$

- Formula (5) for $\varpi^\star$ can be rewritten in terms of $Q(s, a)$:

$$\varpi^\star(s) = \arg\max_{a \in \mathcal{A}} Q(s, a) \tag{8}$$

- Learning the Q-function, the agent can **select optimal actions without any knowledge about r and δ**

Q-Learning (Ctd.)

- We maximize the expected cumulative reward by always choosing the action $a^\star$ with the maximum Q-value:

$$a^\star := \arg\max_{a \in \mathcal{A}} Q(s, a)$$

- What may seem surprising: **Local optimal choices lead to a global optimal solution**

Learning the Q-function corresponds to learning the optimal policy $\varpi^\star$

Algorithm for *Q*-Learning

- We learn the Q-function by **iterative approximation**
- Note the close relationship of Q and $V^{\omega^\star}$:

$$V^{\omega^\star}(s) = \max_{a' \in \mathcal{A}} Q(s, a') \tag{9}$$

- Therefore, $Q(s, a)$ introduced in equation (7) can be expressed in terms of itself:

BELLMAN equation:

$$Q(s, a) = r(s, a) + \gamma \max_{a' \in \mathcal{A}} Q\big(\delta(s, a), a'\big) \tag{10}$$

Algorithm for *Q*-Learning (Ctd.)

- $\widehat{Q}$ denotes the estimate of the true Q-value
- The Q-function is represented by a table in the case of **discrete states and actions**
- Initially, the table is filled with random numbers or zeros

State-action pair	$\widehat{Q}$-value
(s_0, a_0)	$\widehat{Q}(s_0, a_0)$
(s_0, a_1)	$\widehat{Q}(s_0, a_1)$
(s_1, a_0)	$\widehat{Q}(s_1, a_0)$
(s_1, a_1)	$\widehat{Q}(s_1, a_1)$
...	...

Note: The table may become large if there are many state-action pairs. We need other methods **for continuous states and actions** *(e. g. neural networks)*

Algorithm for *Q*-Learning (Ctd.)

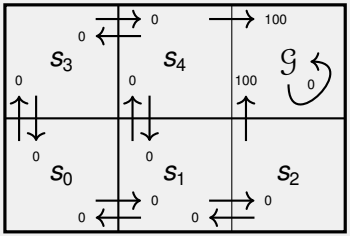
- The agent repeatedly observes the current state s, chooses some action a, executes it and observes r as well as the new state s'
- Update the table entry for $\widehat{Q}(s, a)$ according to the BELLMAN equation (10)
- The agent doesn't have to know $\delta(s, a)$ and $r(s, a)$
- Instead, it executes some actions and observes what happens **(trial-and-error)**
- In the limit, $\widehat{Q}$ will converge towards the actual Q-function **if**...
  - ...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.)

foreach (s, a) **do**
  $\widehat{Q}(s, a) \longleftarrow 0$
end
while *not converged* **do**
  Observe current state s
  Select an action a and execute it
  Receive the immediate reward r
  Observe the new state s'
  Update the table entry for $\widehat{Q}(s, a)$: $\widehat{Q}(s, a) \longleftarrow r + \gamma \max_{a' \in \mathcal{A}} \widehat{Q}(s', a')$
  $s \longleftarrow s'$
end

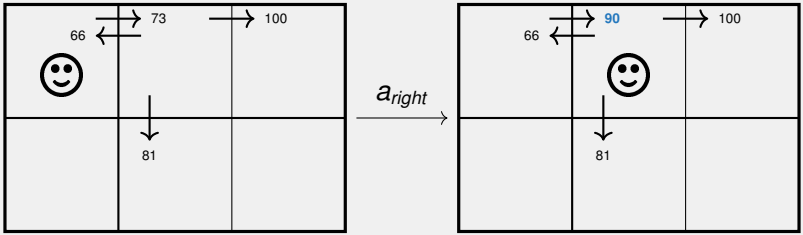
Q-Learning: An illustrative Example

Let us again consider the basic grid world example:



We will now initialize the agent denoted by ☺ in state s_3

Q-Learning: An illustrative Example (Ctd.)



$$\widehat{Q}(s_3, a_{right}) \longleftarrow r + \gamma \max_{a' \in \mathcal{A}} \widehat{Q}(s_4, a')$$

$$\longleftarrow 0 + 0.9 \max_{a' \in \mathcal{A}} \{66, 81, 100\}$$

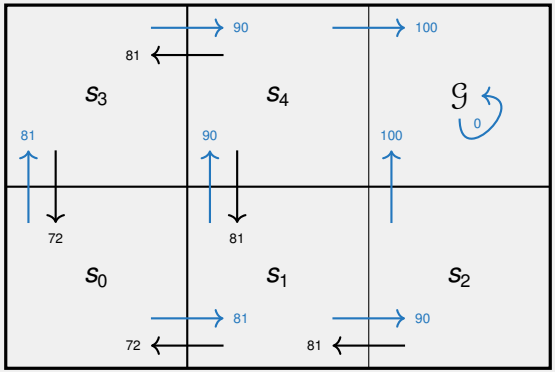
$$\longleftarrow 90$$

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 $\widehat{Q}$
- 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 $\mathcal{G}$ has been reached
  3. Repeat until convergence (go back to ❶)

Q-Learning: An illustrative Example (Ctd.)

Final *Q*-values:



SARSA: State-Action-Reward-State-Action

- The SARSA algorithm is very similar to *Q*-learning
- SARSA is short for '**S**tate-**A**ction-**R**eward-**S**tate-**A**ction'
- It refrains from calculating the 'max' in the BELLMAN equation (10) and instead uses the current policy ϖ *(on-policy update)*

SARSA update rule:

$$\widehat{Q}(s,a) \longleftarrow r(s,a) + \gamma \widehat{Q}\big(\underbrace{\delta(s,a)}_{=s'}, \varpi(s')\big) \tag{11}$$

Section:

Miscellaneous

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

Exploitation vs. Exploration

When always picking the best action, the **agent overcommits to actions** found to be good during early training, but it fails to explore other actions that might be even better

Exploitation:

- Use the action the agent assumes to be the best one
- Approximate the optimal policy

vs.

Exploration:

- 'Optimal action' may be wrong due to approximation errors
- Try a sub-optimal one

There is a trade-off between exploitation and exploration!

Exploitation vs. Exploration (Ctd.)

Solution: Give all actions a certain probability to be chosen. Two possible options:

1. ε**-greedy** (small ε favours exploitation)

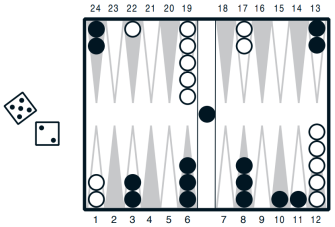
$$p(a|s) = \begin{cases} 1 - \varepsilon + \frac{\varepsilon}{|\mathcal{A}|} & \text{if } a = \arg\max_{a' \in \mathcal{A}} \widehat{Q}(s, a') \\ \frac{\varepsilon}{|\mathcal{A}|} & \text{otherwise} \end{cases} \tag{12}$$

2. **Softmax** (actions with high $\widehat{Q}$-values get a higher probability)

$$p(a|s) = \frac{k^{\widehat{Q}(s,a)}}{\sum_j k^{\widehat{Q}(s,a_j)}} \qquad k > 0 \quad \text{(large } k \text{ favours exploitation)} \tag{13}$$

Non-Deterministic Rewards and Actions

- So far we have only considered the deterministic case
- **What if the environment is non-deterministic, e. g. stochastic?**
  - 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 *(and 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.)

- Set V^{ϖ} to be the expected value (solve e. g. by using **Monte Carlo sampling**):

$$V^{\varpi}(s_0) := \mathbb{E}\left\{\sum_{t=0}^{\infty} \gamma^t r\big(s_t, \varpi(s_t)\big)\right\} = r\big(s_0, \varpi(s_0)\big) + \underbrace{\frac{1}{k}\sum_{i=1}^{k}}_{k \text{ samples}} \sum_{t=1}^{\infty} \gamma^t r\big(s_{t,i}, \varpi(s_{t,i})\big) \tag{14}$$

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

$$\begin{aligned} Q(s,a) &:= \mathbb{E}\big\{r(s,a) + \gamma V^{\varpi^*}\big(\delta(s,a)\big)\big\} \\ &= \mathbb{E}\big\{r(s,a)\big\} + \gamma \mathbb{E}\big\{V^{\varpi^*}\big(\delta(s,a)\big)\big\} \\ &= \mathbb{E}\big\{r(s,a)\big\} + \gamma \sum_{s'} p(s'|s,a) V^{\varpi^*}(s') \end{aligned} \tag{15}$$

Non-Deterministic Rewards and Actions (Ctd.)

- Again, we express the Q-function recursively:

$$Q(s,a) = \mathbb{E}\{r(s,a)\} + \gamma \sum_{s'} p(s'|s,a) \max_{a'} Q(s',a') \tag{16}$$

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

$$\widehat{Q}_n(s,a) \longleftarrow (1-\alpha_n)\widehat{Q}_{n-1}(s,a) + \alpha_n\left[r + \gamma \max_{a'} \widehat{Q}_{n-1}(s',a')\right] \tag{17}$$

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

Non-Deterministic Rewards and Actions (Ctd.)

- Updates of $\widehat{Q}$ are made **more gradually** than in the deterministic case
- For $\alpha_n = 1$ we get the old training rule
- α_n decreases over time which causes the updates become smaller in later iterations
- Due to the average, the Q-learning 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**:

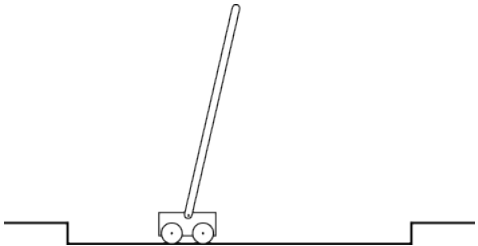
$$\widehat{Q}_n(s,a) \longleftarrow (1-\alpha_n)\widehat{Q}_{n-1}(s,a) + \alpha_n\left[r(s,a) + \gamma \max_{a'} \widehat{Q}_{n-1}(s',a')\right]$$

$\Updownarrow$ **rewrite the equation**

$$\widehat{Q}_n(s,a) \longleftarrow \widehat{Q}_{n-1}(s,a) + \alpha_n\Big[\overbrace{r(s,a) + \gamma \max_{a'} \widehat{Q}_{n-1}(s',a')}^{\text{after}} - \overbrace{\widehat{Q}_{n-1}(s,a)}^{\text{before}}\Big] \quad (18)$$

$$\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad}_{\textbf{Temporal difference}}$$

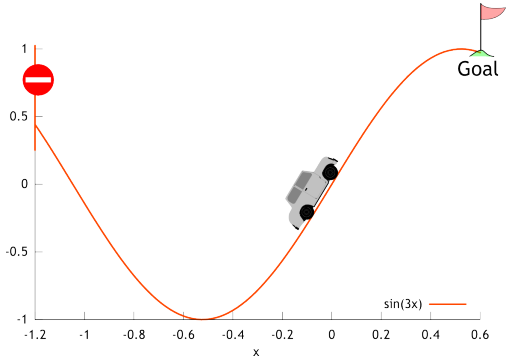
Demo: Cartpole Task



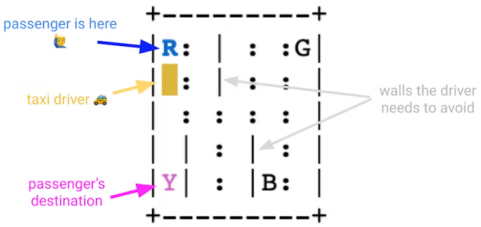
- Move the cart **without** the pole falling over
- Action space $\mathcal{A}$:
  - Move LEFT (0)
  - Move RIGHT (1)
- 4D state space $\mathcal{S}$:
  - Cart position, velocity
  - Angle of pole to cart...
  - ...and its derivative

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 rectangle represents the taxi
- '|' represents a wall which should be avoided
- Blue letter: **Pick-up location**
- Purple letter: **Drop-off location**
- Taxi turns **green** if a passenger is aboard

Section:

Advanced Algorithms

Deep Reinforcement Learning

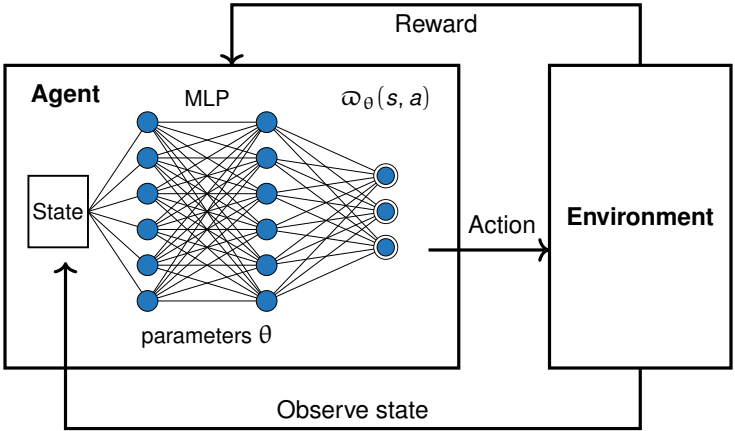
Deep Reinforcement Learning

Keeping track of the Q-function is intractable if state space and/or action space are really large or even continuous. Maybe it does not even fit into memory... We need an alternative!

Idea:

- Use deep learning methods to approximate the policy ϖ
- E. g. an multi-layer perceptron could predict the action to take given the observed state, the reward, etc.

Deep Reinforcement Learning (Ctd.)



Playing Atari with Deep Reinforcement Learning

Volodymyr Mnih **Koray Kavukcuoglu** **David Silver** **Alex Graves** **Ioannis Antonoglou**

Daan Wierstra **Martin Riedmiller**

DeepMind Technologies

{vlad,koray,david,alex.graves,ioannis,daan,martin.riedmiller} @ deepmind.com

Abstract

We present the first deep learning model to successfully learn control policies directly from high-dimensional sensory input using reinforcement learning. The model is a convolutional neural network, trained with a variant of Q-learning, whose input is raw pixels and whose output is a value function estimating future rewards. We apply our method to seven Atari 2600 games from the Arcade Learning Environment, with no adjustment of the architecture or learning algorithm. We find that it outperforms all previous approaches on six of the games and surpasses a human expert on three of them.

https://arxiv.org/abs/1312.5602

Section:

Wrap-Up

Summary
Recommended Literature
Self-Test Questions
Lecture Outlook

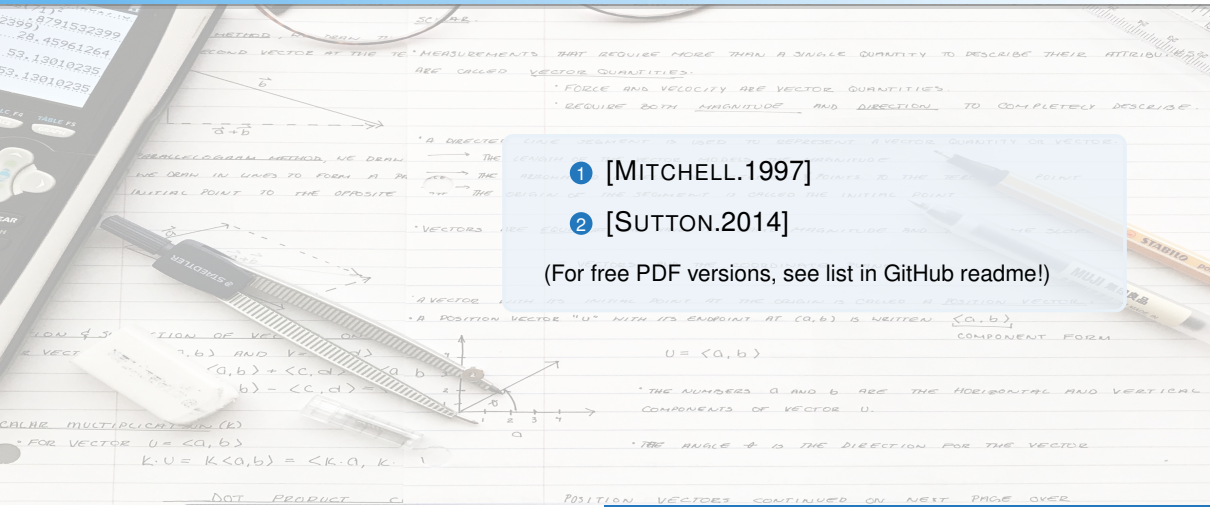
Summary

- **What is reinforcement learning?**
  - Learn optimal decisions based on 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?
  - Simple example: MENACE
- **Important terms:**
  - State s, action a
  - Policy π *(optimal policy* $\pi^\star$*)*
  - Reward r *(immediate reward, discounted reward)*

Summary (Ctd.)

- **Algorithms (dynamic programming)**
  - Value iteration and policy iteration
  - *Q*-learning
  - SARSA
- **Exploitation versus exploration**
  - **Exploitation**: Exploit what is already known to be good
  - **Exploration**: But don't miss to explore new states and actions
- Non-deterministic case *(the functions* $\delta(s, a)$ *and* $r(s, a)$ *are stochastic)*
- Temporal difference learning
- Deep reinforcement learning

Recommended Literature



1. [MITCHELL.1997]
2. [SUTTON.2014]

(For free PDF versions, see list in GitHub readme!)

Self-Test Questions

1. Name and explain some challenges in reinforcement learning!
2. What is a policy? What is a value function?
3. Describe what value iteration is!
4. What is policy iteration? What is the difference to value iteration?
5. What does the policy improvement theorem state?
6. What is Q-Learning and how does it differ from value iteration / policy iteration?
7. Briefly explain what the exploitation / exploration trade-off is about!
8. Name two strategies which ensure that the agent also chooses sub-optimal actions.
9. How do we handle the non-deterministic case?

What's next...?

I	Machine Learning Introduction
II	Optimization Techniques
III	Bayesian Decision Theory
IV	Non-parametric Density Estimation
V	Probabilistic Graphical Models
VI	Linear Regression
VII	Logistic Regression
VIII	Deep Learning
IX	Evaluation
X	Decision Trees
XI	Support Vector Machines
XII	Clustering
XIII	Principal Component Analysis
XIV	Reinforcement Learning
• **XV**	Advanced Regression

Thank you very much for the attention!

*** * * Artificial Intelligence and Machine Learning * * ***

Topic: Introduction to Reinforcement Learning

Term: Summer term 2025

Contact:
Daniel Wehner, M.Sc.
SAP SE / DHBW Mannheim
daniel.wehner@sap.com

Do you have any questions?