Hochschule Hof
University of Applied Sciences

Reinforcement Learning

Applied Deep Learning

Motivation

Hochschule Hof University of Applied Sciences

- Games / AlphaStar/AlphaGo
- Autonomous robots
- NLP

Targets

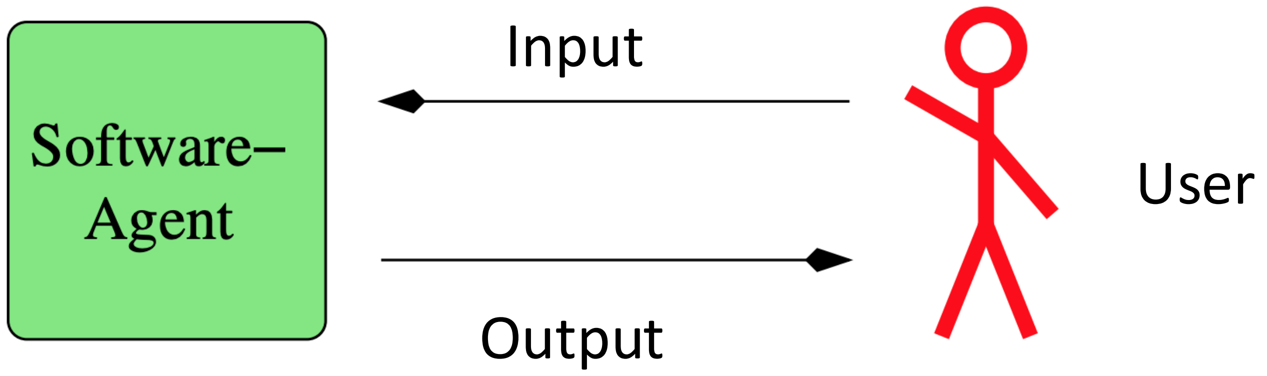
- Know the different types of agents
- Understand the basic mechanism of reinforcement learning
- Know different Deep Reinforcement Learning techniques
- Understanding Q-Learning and Deep-Q-Learning

Agents

- Capture their environment through sensors
- Perform actions in/on the environment through their actuators

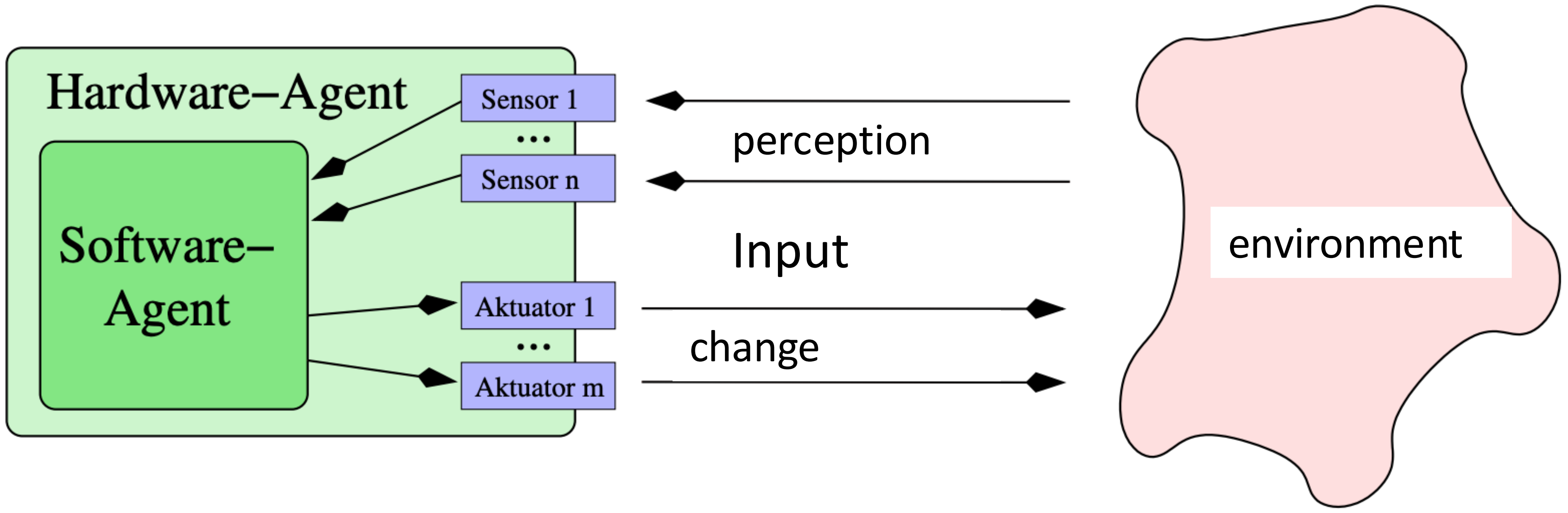
- Software agent
- Hardware agent

Software-Agent



Agents

Hardware Agent (autonomous robot)



Rational agents

- "do the right thing"
- Task environment PEAS
  - Performance measurement
  - Environment
  - Actuators
  - Sensors
- Example: Autonomous vehicle
  - Optimal behavior is often not achievable
  - Not all relevant information can be captured, complexity of the problem too high

Rational vs. omniscient agents

Hochschule Hof
University of Applied Sciences

- An omniscient actor knows the actual effects of his actions.
- In comparison, a rational actor behaves according to his beliefs and knowledge and tries to maximize the expected output.
- Example: If I look both ways before crossing the street and then get hit by a meteorite while crossing, I can hardly be accused of lacking rationality.

Perfectly rational agent

- Rational behavior depends on
  - Performance measures (targets)
  - Perception sequences
  - Knowledge about the environment
  - Possible actions
- For each possible perceptual sequence, a rational agent should choose an action that is expected to maximize its performance measure, given the evidence provided by the perceptual sequence and the integrated knowledge the agent possesses.
- Active perception is necessary to avoid trivialization. The ideal rational agent acts according to the function
  - Perceptual sequence $\times$ world knowledge $\rightarrow$ action

Examples of rational agents

Agent Type	Performance Measure	Environment	Actuators	Sensors
Medical diagnosis system	healthy patient, costs, lawsuits	patient, hospital, stuff	display questions, tests, diagnoses, treatments, referrals	keyboard entry of symptoms, findings, patient's answers
Satellite image analysis system	correct image categorization	downlink from orbiting satellite	display categorization of scene	color pixel arrays
Part-picking robot	percentage of parts in correct bins	conveyor belt with parts, bins	jointed arm and hand	camera, joint angle sensors
Refinery controller	purity, yield, safety	refinery, operators	valves pumps, heaters displays	temperature, pressure, chemical sensors
Interactive English tutor	student's score on test	set of students, testing agency	display exercises, suggestions, corrections	keyboard entry

Environment properties

- Partially/completely observable: Is the entire state of the environment known at any point in time?
- Single / multiple agents: Opposing, cooperative, indifferent
- Deterministic / stochastic: If the state $Z(t+1) = f(Z(t), A)$ is always the same $\rightarrow$ deterministic
- Episodic / sequential: Episodic tasks are atomic and have no influence on later tasks (image classification)
- Static / dynamic: Does the environment change while the agent is planning?
- Discrete / continuous: states and inputs
- Known / unknown: Are all laws known, etc.

Agent structure

- Realization of the ideal mapping through a
- Agent program executed on a
- Architecture that also provides an interface with the environment (perceptions, actions)
- → Agent = Architecture + Program

Table-driven agents

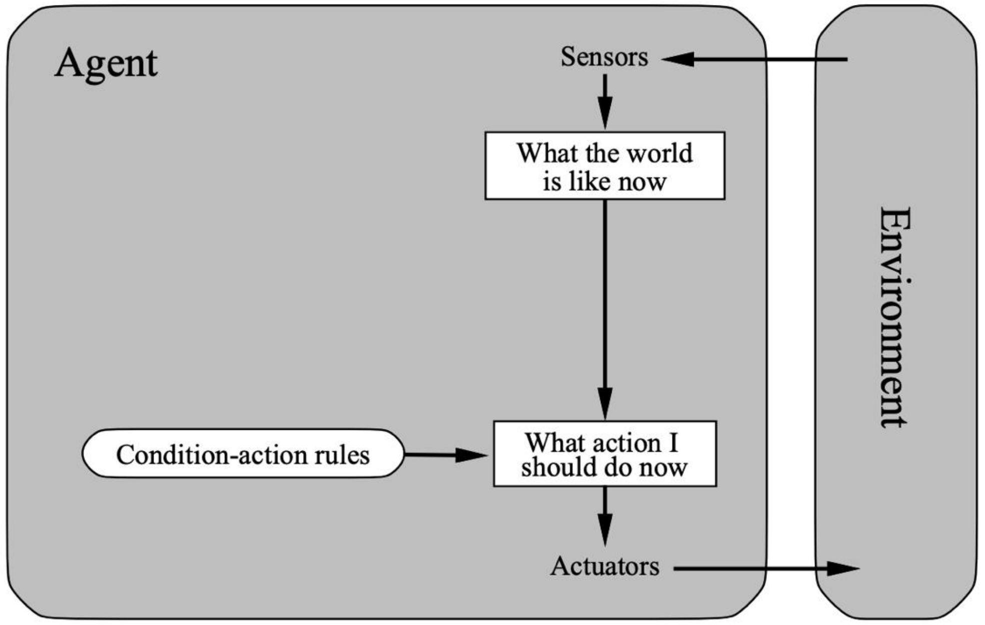
- Keeps a table of possible perceptions and the corresponding actions
- Looks up a sequence of perceptions in the table and chooses the "correct" action
- Problems:
  - The table can become very large and
  - the designer usually takes a long time to specify them (or learn them)
  - virtually impossible

Table-driven agents

- P = possible perceptions
- T = Lifetime
- Lookup table = P^T
- Chess: 10^{150} entries
  - Cannot be saved
  - Cannot be programmed (time)
  - Cannot be learned

Simple reflex agents

- The direct use of perceptions is often not possible because they require a lot of memory (e.g. video images).
- Inputs are therefore often interpreted before decisions are made.
- Condition - rules of action
- Forget the past

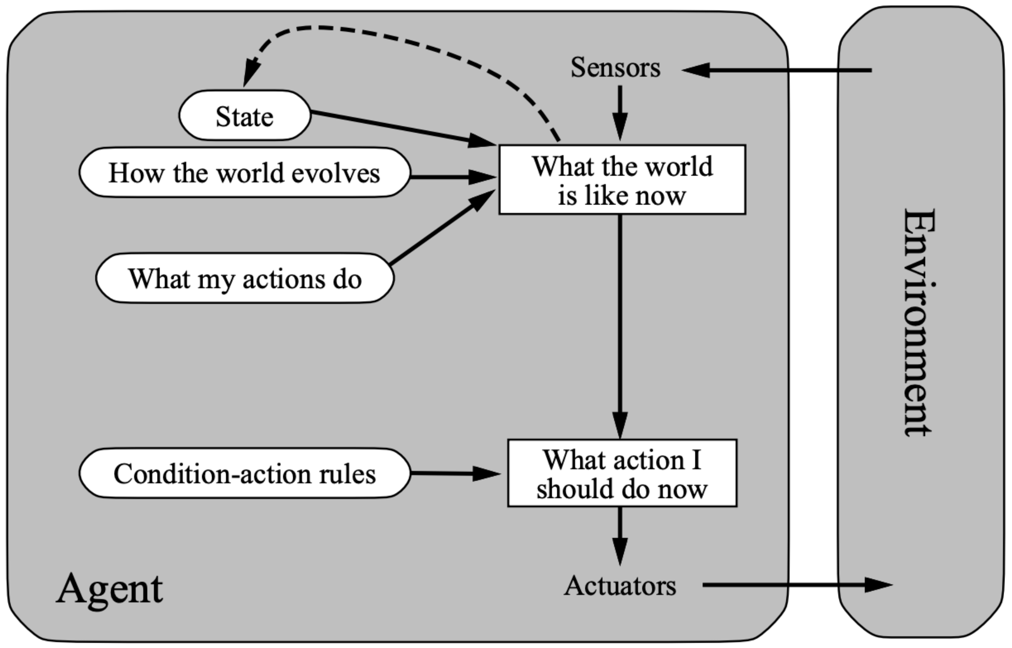


Model based reflex agents

Hochschule Hof University of Applied Sciences

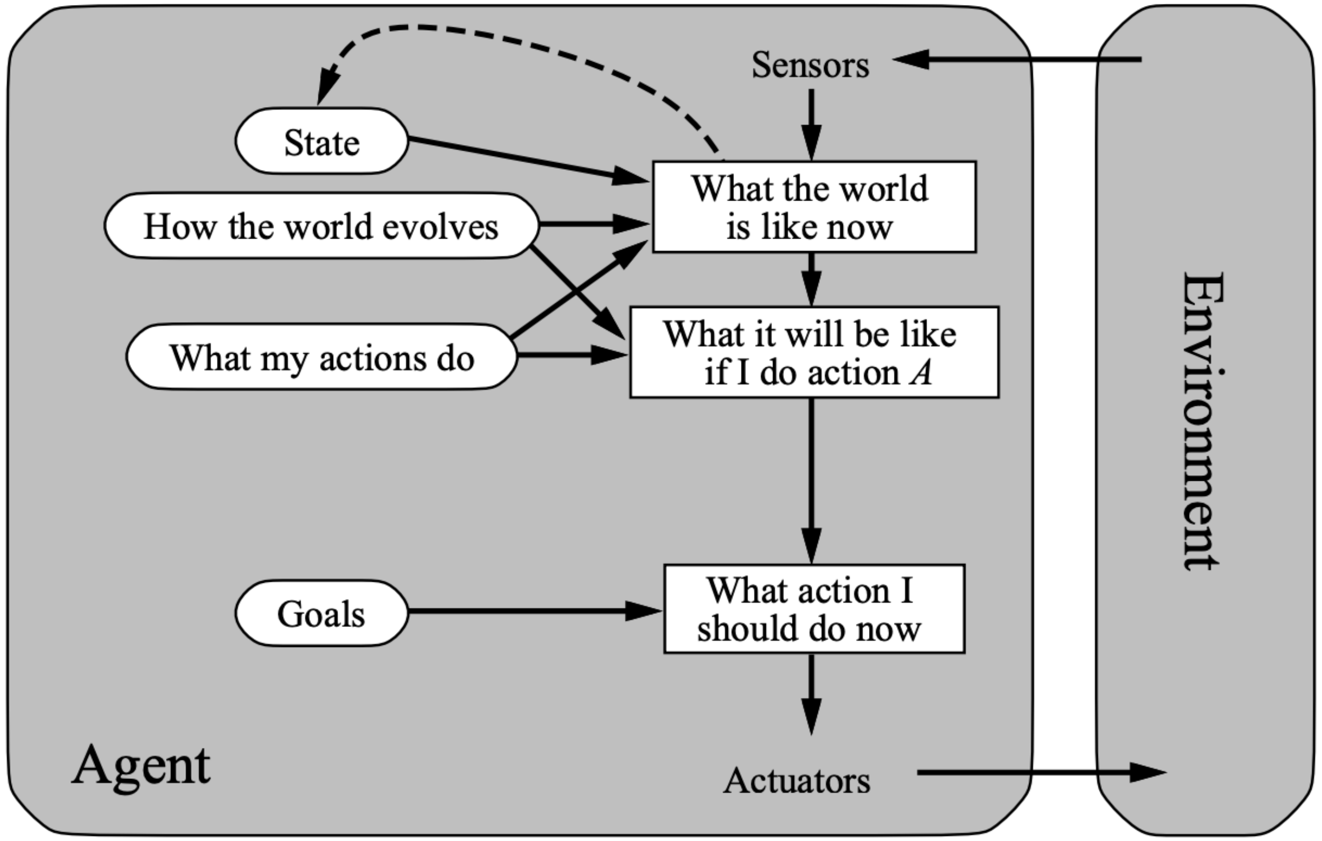
- Tracking the partially visible environment through internal status

In case the agent's history in addition to the actual percept is required to decide on the next action, it must be represented in a suitable form.



Goal-based agents

Hochschule Hof
University of Applied Sciences


Sensors
State
What the world is like now
How the world evolves
What my actions do
What it will be like if I do action *A*
Goals
What action I should do now
Actuators
Agent
Environment

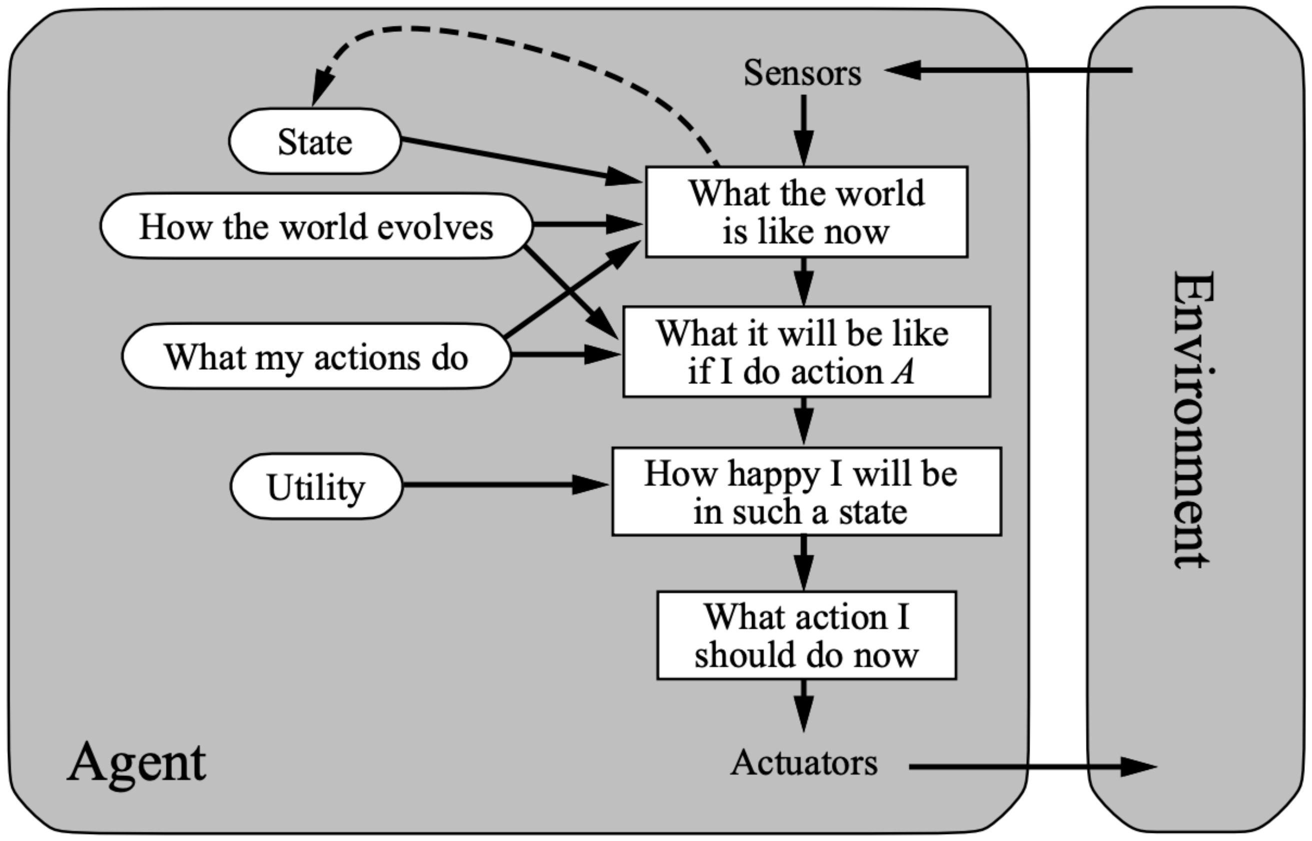
Goal-based agents

- Often, perceptions alone are not enough to decide what to do.
- This is because the correct action depends on the given explicit goals (e.g., go in direction X).
- The model-based, goal-based agents use an explicit representation of goals and take them into account when choosing actions.

Utility-based agents

- Goal-oriented = binary decision (happy or not happy)
- Usually, in a given situation, there are several possible actions that can be taken.
- In such cases, the benefit of the next state reached can be considered to make a decision.
- A utility function maps a state (or sequence of states) to a real number.
- The agent can also use these numbers to weigh the importance of competing goals.
- Benefit = internal performance measure

Utility-based agents

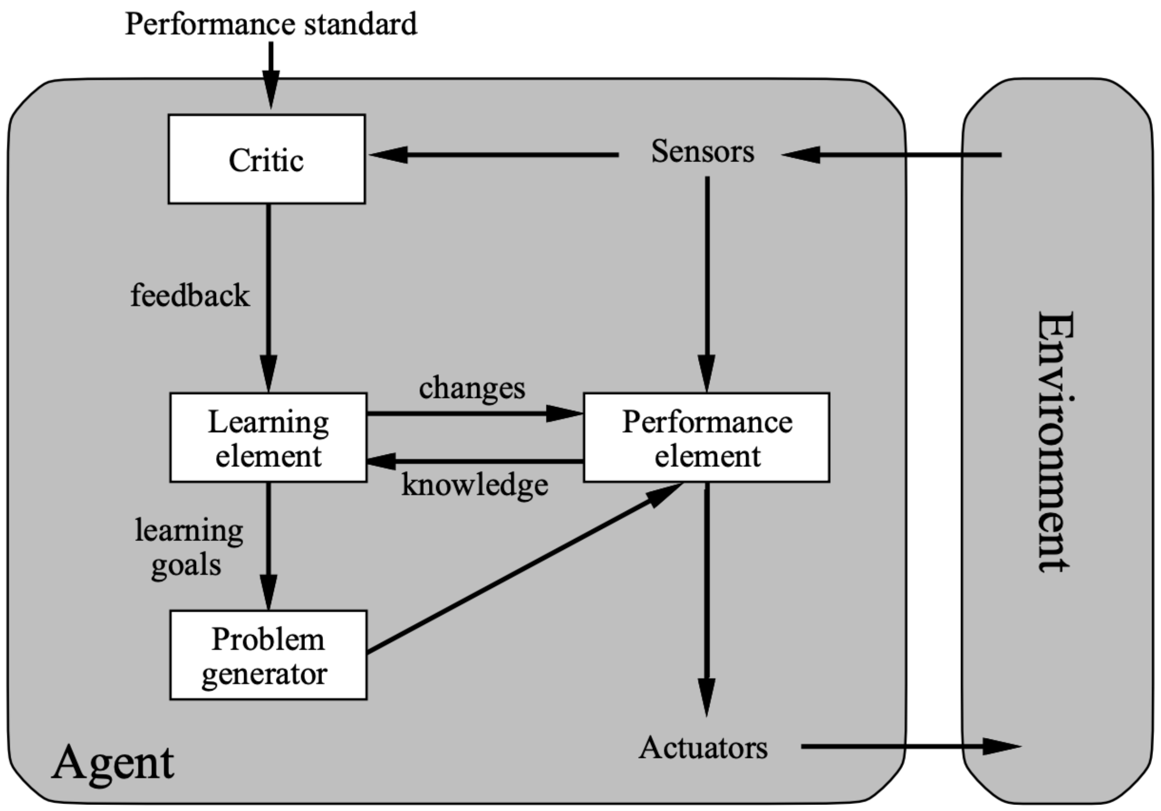

Sensors
State
How the world evolves
What my actions do
Utility
What the world is like now
What it will be like if I do action *A*
How happy I will be in such a state
What action I should do now
Actuators
Agent
Environment

Learning agents

- Learning agents can become more competent over time.
- You can start with an initially empty knowledge base.
- You can work in initially unfamiliar environments.

Learning agents

- Learning element (responsible for improvements)
- Performance element (must select external actions, our previous agent).
- Critic (determines how the performance element should be changed to perform better).
- Problem generator (suggests actions that lead to informative experiences).
- Punishment/reward direct feedback of agent behavior to learn benefits

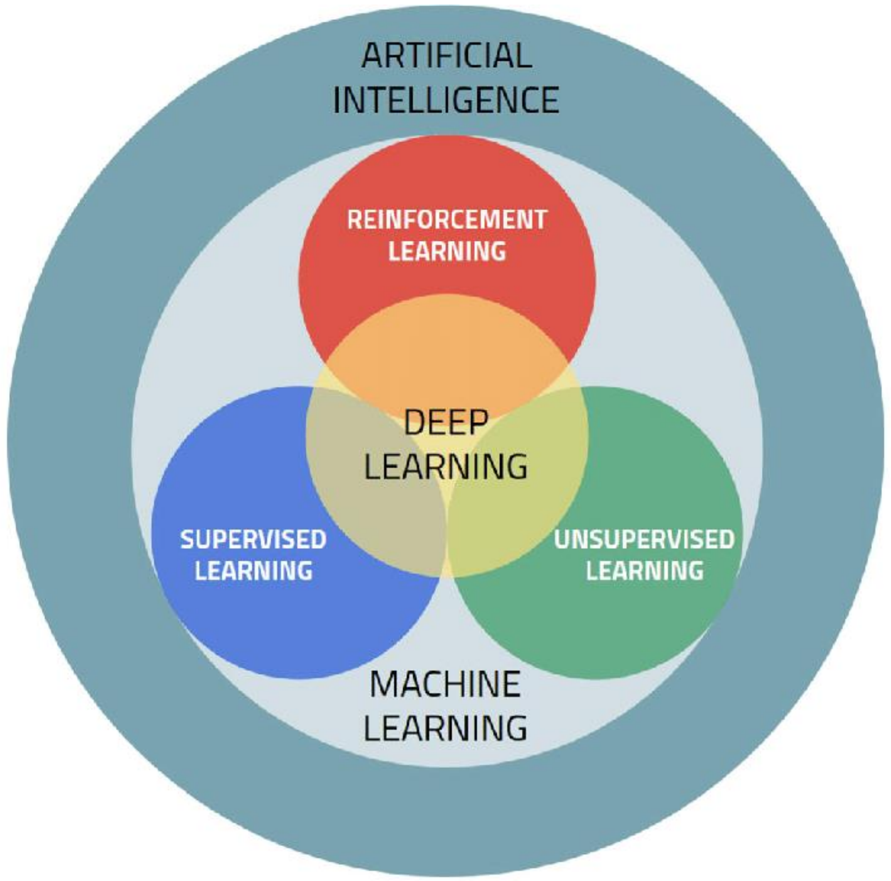


Summary

- An agent is something that perceives and acts.
- It consists of an architecture and an agent program.
- An ideal rational agent always performs the action that maximizes its performance given the perceptual sequence and its knowledge of the environment.
- An agent program maps from a perception to an action.
- There are a variety of designs
    - Reflex agents respond immediately to perceptions.
    - Goal-oriented agents work toward goals.
    - Utility-based agents seek to maximize their rewards.
- Learning agents improve their behavior over time.
- Some environments are more challenging than others.
- Environments that are partially observable, nondeterministic, strategic, dynamic, and continuous, and in which multiple agents operate are the most challenging.

Reinforcement Learning

- Machine learning
- Software agent independently learns a strategy (policy) to maximize rewards received
- Agent has no information which action is the best in a situation
- Agent learns through interaction with the environment
- Reward can also be negative



Reinforcement Learning

Components

- Agent
- Environment (Markov decision process)
- States S
- Actions A
- Rewards R

- Selection of action A_t in S_t according to a policy π $_t$

$$(S_t, A_t) \overset{\pi}{\rightarrow} (S_{t+1}, R_t)$$

- Goal of the agent: Maximization of future expected profit:

$$\mathbb{E}[G_t] = \mathbb{E}\left[\sum_{k=0}^{T} \gamma^k \cdot R_{t+k}\right]$$

with $0 \leq \gamma \leq 1$

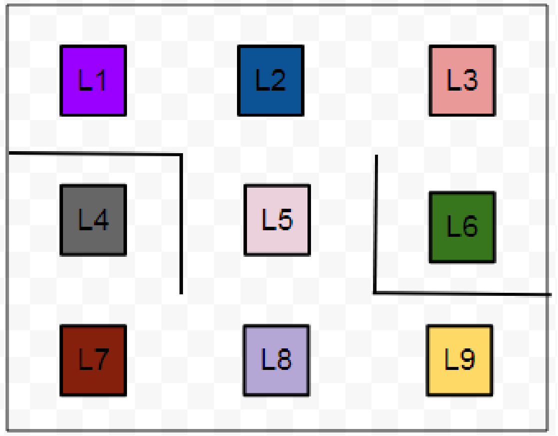
Reinforcement Learning

- Exploitation vs Exploration
  - Agent should maximize profit
  - For this he should choose actions that provide a high reward
  - But to discover such actions he must try new actions
  - Problem unsolved so far

Reinforcement Learning

Example

- Robot in factory
  - S = {0,1,2,...,8}
  - A = {0,1,2,...,8}
- Desired target state: 6



Rewards

	L1	L2	L3	L4	L5	L6	L7	L8	L9
L1	0	1	0	0	0	0	0	0	0
L2	1	0	1	0	1	0	0	0	0
L3	0	1	0	0	0	1	0	0	0
L4	0	0	0	0	0	0	1	0	0
L5	0	1	0	0	0	0	0	1	0
L6	0	0	1	0	0	999	0	0	0
L7	0	0	0	1	0	0	0	1	0
L8	0	0	0	0	1	0	1	0	1
L9	0	0	0	0	0	0	0	1	0

Reinforcement Learning

- Robot to drive from A to green
- Idea: Realization by Footprint from 1s
- Problem: What if the robot starts at B?
- Both directions would have a 1

🟩		
↑		
↑		
↑ B	←	A

🟩		
1		
1		
1	1	1

Reinforcement Learning

Bellmann - Equation

$$V(s)=\max_a (R(s,a)+\gamma V(s'))$$

with

- s = state (space)
- a = action (movement in space)
- s' = next state
- $\boldsymbol{\gamma}$ = discount factor
- $R(s, a)$ = Reward function
- $V(s)$ = value to be in a certain state (footprint)

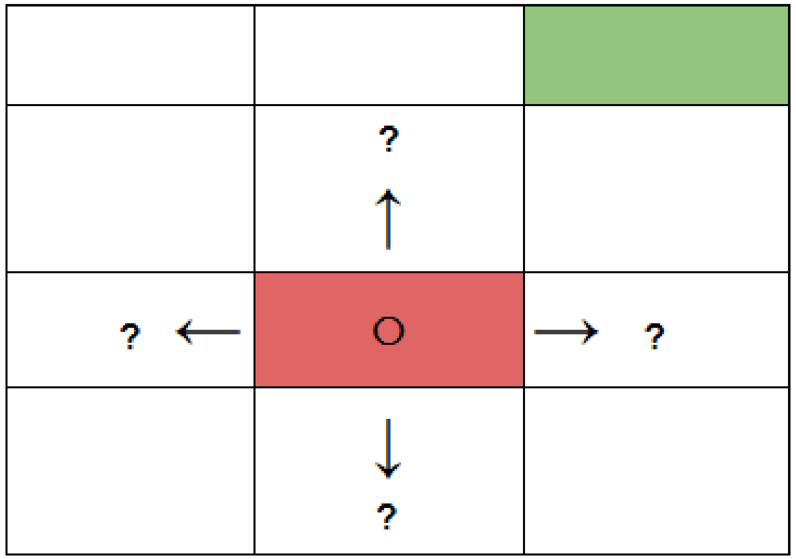
- Example $\boldsymbol{\gamma}$=0.9

🟩		
1		
0.9		
0.81	0.729	Starting point

- Yellow has value 1, since green is closest
- Calculation *:
  $V(s)=\max_a (0 + 0.9*1) = 0.9$

Reinforcement Learning

- Robot has low probability of malfunction
- Result is therefore partly under control of the robot and partly random → MDP



- Introduction of a probability for each result
- From $V(s) = \max_a(R(s,a) + \gamma V(s'))$

  to

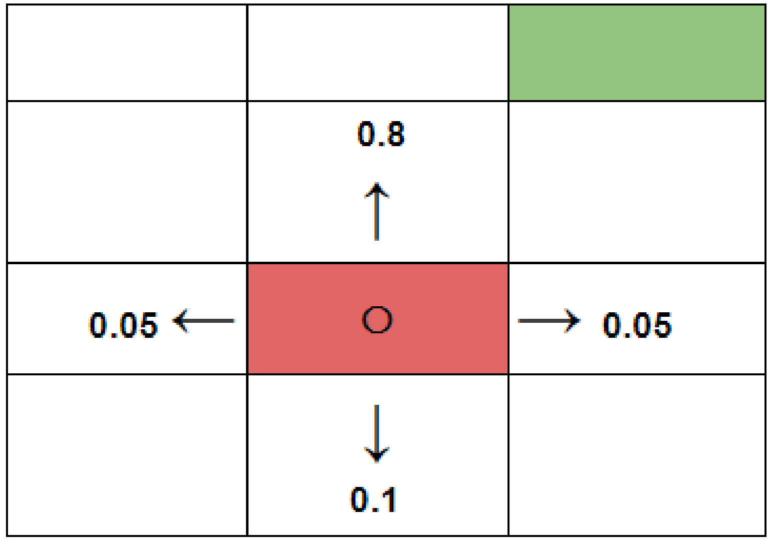
  $$V(s) = \max_a(R(s,a) + \gamma \sum_{s'} P(s,a,s') \cdot V(s'))$$

  with

  P(s, a, s′) - probability with action a to get from s to s'

Reinforcement Learning

- Assumption of the following probabilities for the respective movements



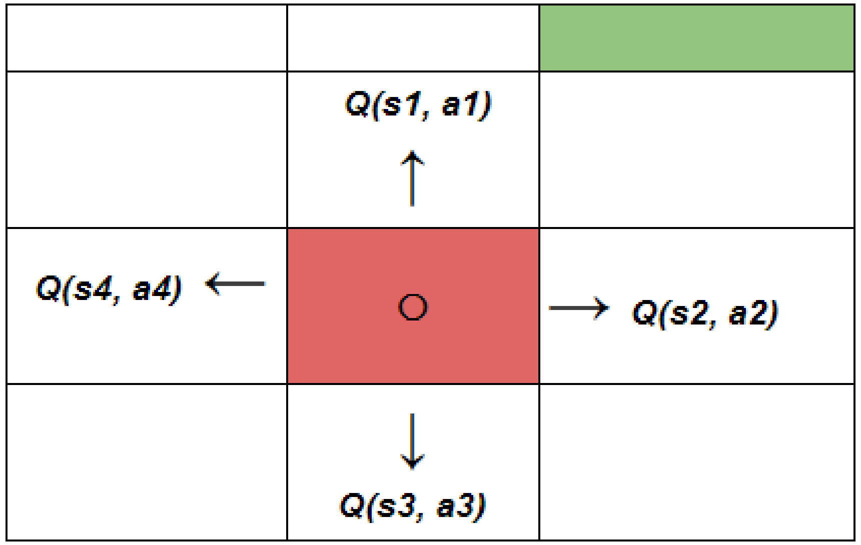
- If we add everything into our previous equation we get

$$V(s) = \max_a (R(s,a) + \gamma((0.8V(room_{up})) + (0.1V(room_{down})) + \ldots))$$

- Footprints have now changed due to the stochastic influence
- Attention: Reward only when the target state is reached!

Reinforcement Learning

- Every action of the robot should generate a reward → living penalty
- Instead of V(s), we introduce quality Q of an action a



- We remove the max from the equation

$$V(s) = \max_a \left(R(s,a) + \gamma \sum_{s'} P(s,a,s') \cdot V(s') \right)$$

And get

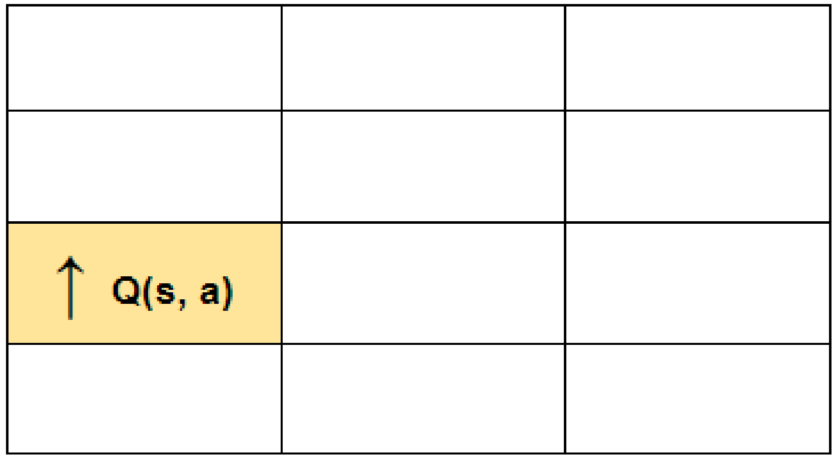
$$R(s,a) + \gamma \sum_{s'} \left(P\left(s,a,s'\right) V\left(s'\right)\right) = Q(s,a)$$

We make one more small adjustment:

$$Q(s,a) = R(s,a) + \gamma \sum_{s'} \left(P\left(s,a,s'\right) \max_{a'} Q\left(s',a'\right) \right)$$

Reinforcement Learning

- Environment changes over time
- Q-Values must be adjusted accordingly → Temporal Differencing
- Ex: Q Value after movement upwards could be different from previous observation



- --> recalculate Q(s,a) and subtract previous value for Q(s,a)

$$TD(a,s) = \underbrace{\boxed{R(s,a) + \gamma \sum_{s'} \left(P(s,a,s') \max_{a'} Q(s',a') \right)}}_{\text{New } \mathbf{Q(s,\ a)}} - Q(s,a)$$

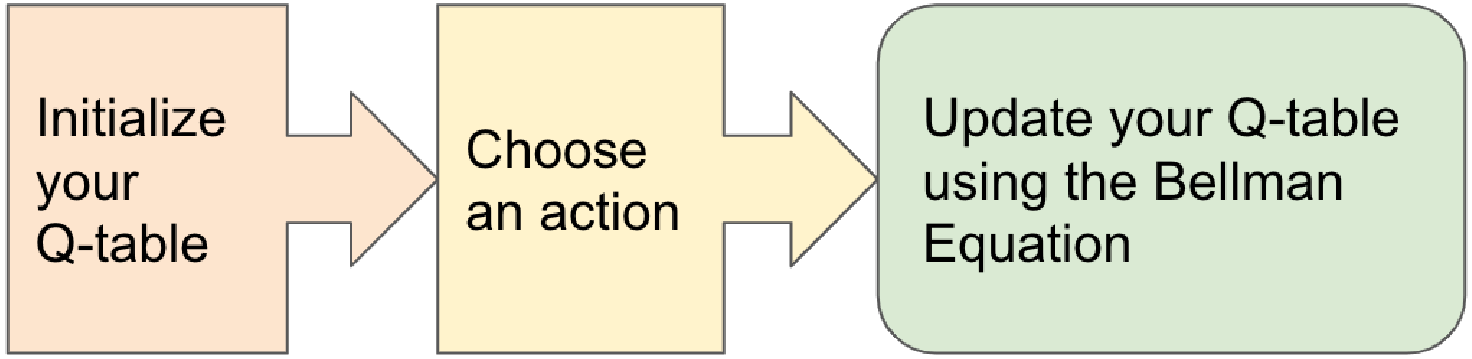
Reinforcement Learning

- New Q(s,a) is updated as

$$Q_t(s,a) = Q_{t-1}(s,a) + \alpha TD_t(a,s)$$

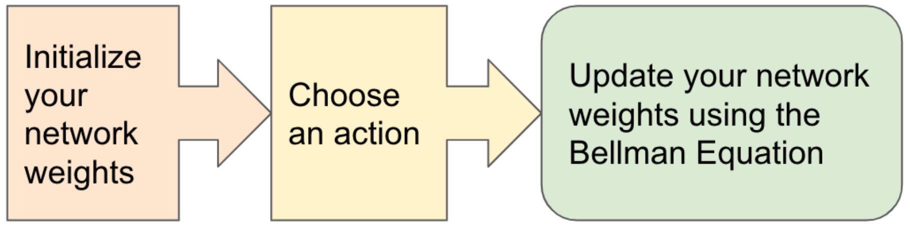
$$\rightarrow Q_t(s,a) = Q_{t-1}(s,a) + \alpha \left(R(s,a) + \gamma \max_{a'} Q\left(s',a'\right) - Q_{t-1}(s,a) \right)$$

with learning rate alpha: how fast robot adapts to changes



Deep Q-Learning

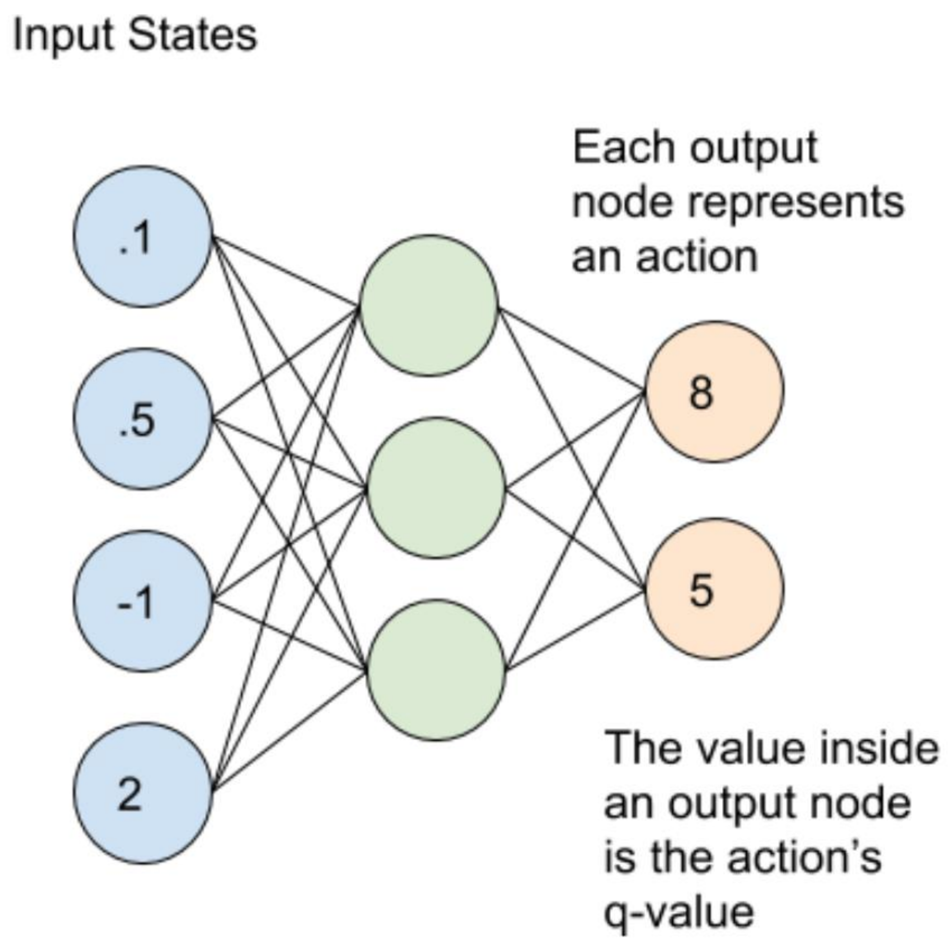
- No more explicit Q-table
- Neural network instead
- Use of two networks
  - Identical structure, but different weights
  - Every x steps weights are copied from the Main Network to the Target Network
  - Leads to more stability


Initialize your network weights
Choose an action
Update your network weights using the Bellman Equation

Deep Q-Learning

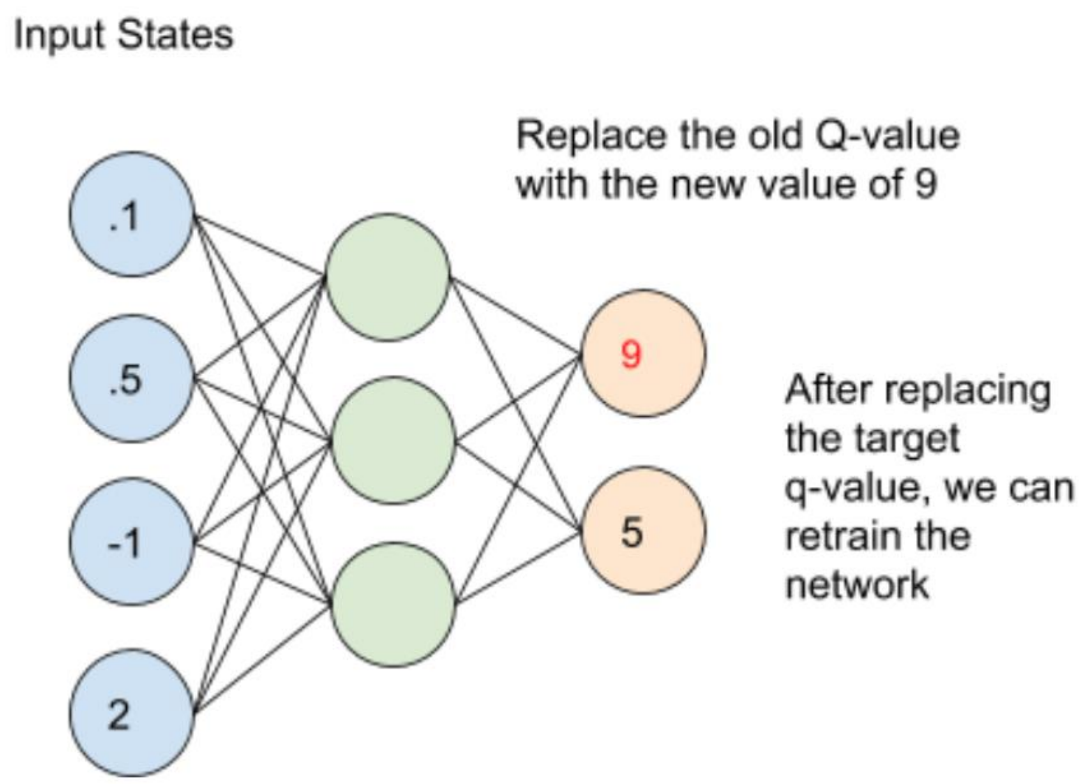
- Instead of mapping (s,a)-> q, we use s -> (q,a)
- Each output neuron corresponds to one action
- The value of the neuron corresponds to the associated q-value
- He initialization
- Huber - Loss

$$L_\delta(a) = \begin{cases} \frac{1}{2}a^2 & \text{for } |a| \le \delta, \\ \delta \cdot \left(|a| - \frac{1}{2}\delta\right), & \text{otherwise.} \end{cases}$$


Input States
.1
.5
-1
2
Each output node represents an action
8
5
The value inside an output node is the action's q-value

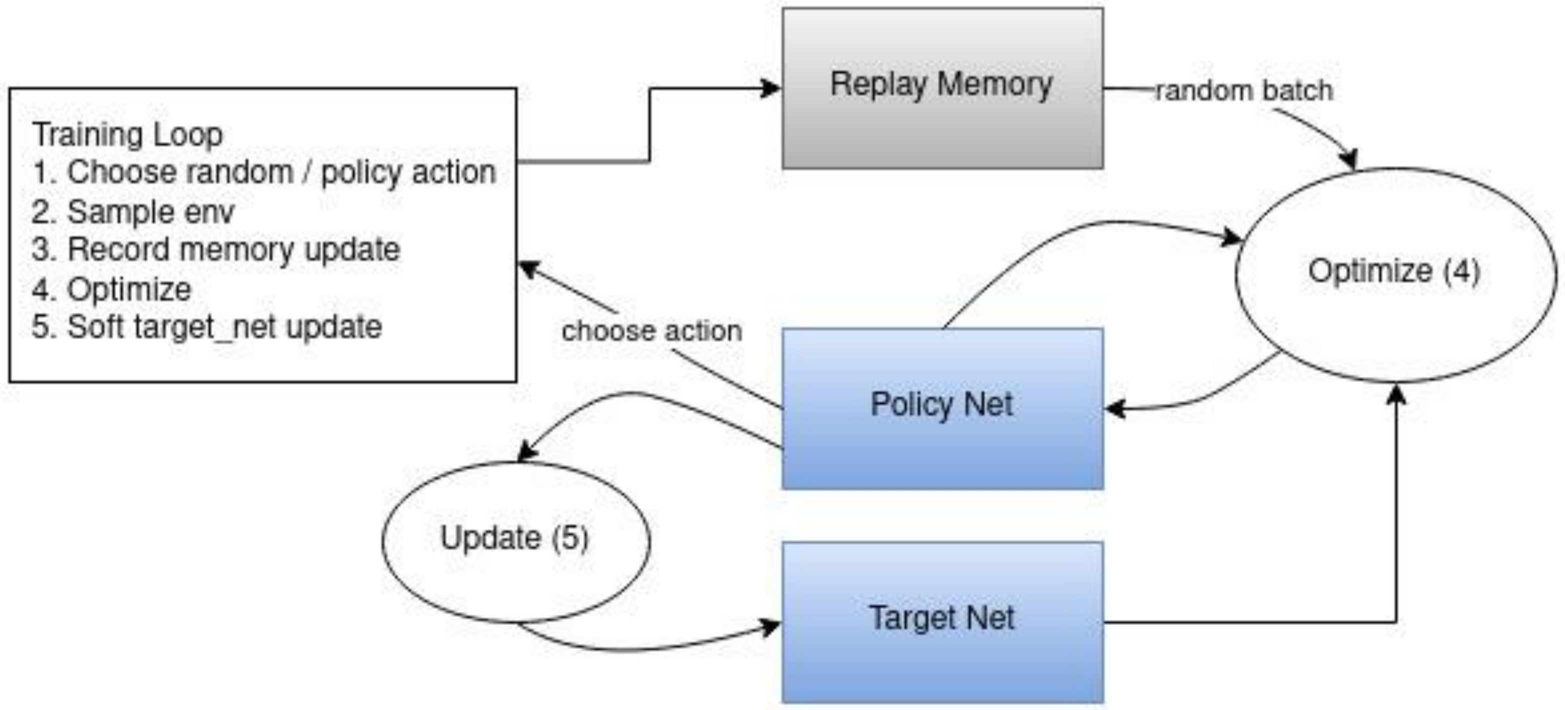
Deep Q-Learning

- Choose next action (highest q-value)
- Update of the weights:
  - Experience Replay
  - e.g. 4-10 steps sampling
  - Then update with
    - Bellmann equation
    - SGD


Input States
Replace the old Q-value with the new value of 9
.1
.5
-1
2
9
5
After replacing the target q-value, we can retrain the network

Hochschule Hof
University of Applied Sciences

Deep Q-Learning



Summary

- There exist different types of agents with different complexities
- Reinforcement learning helps an agent build a policy in a partially observable environment
- Q-Learning uses a table to store all state-action pairs
- DQN is "model-free" → actually stored in the weights of the NN