Reinforcement Learning - Basics

Chair of Automation and Information Systems

Technical University of Munich

Google DeepMind's Deep Q-learning playing Atari Breakout

Source: https://www.youtube.com/watch?v=V1eYniJ0Rnk

What is reinforcement learning?

"… 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

„Reinforcement learning seeks to incentivize computational agents to naturally learn correct decisions by trial and error and to pursue a long term reward."

DeepAI.org

Source: https://deepai.org/machine-learning-glossary-and-terms/reinforcement-learning

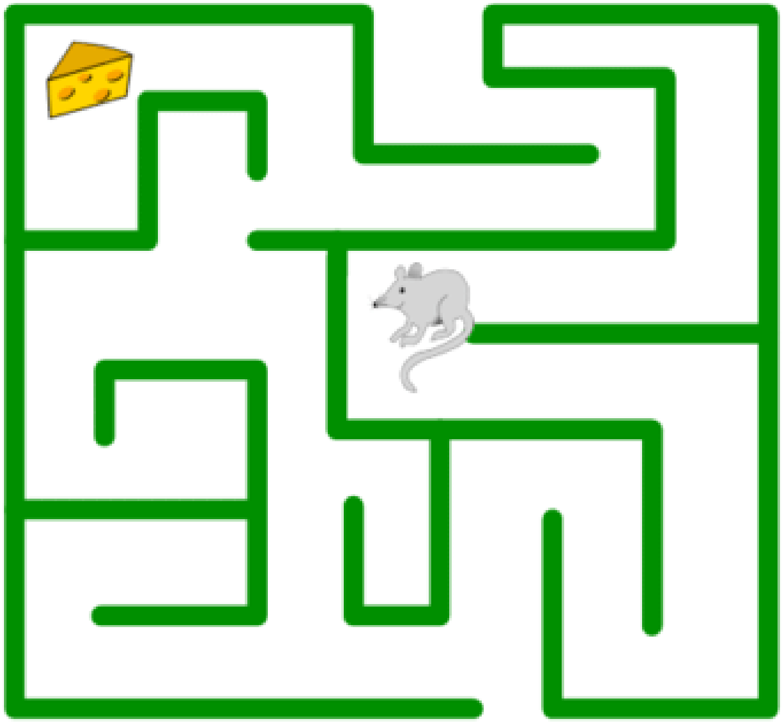
Topics

1. Agent – Environment Model
2. Markov Decision Processes
3. Value Function
4. Action Value Function
5. Q-Learning
6. Deep Q-Learning

Lehrstuhl AIS
Fakultät für Maschinenwesen
Technische Universität München

Agent – Environment Model

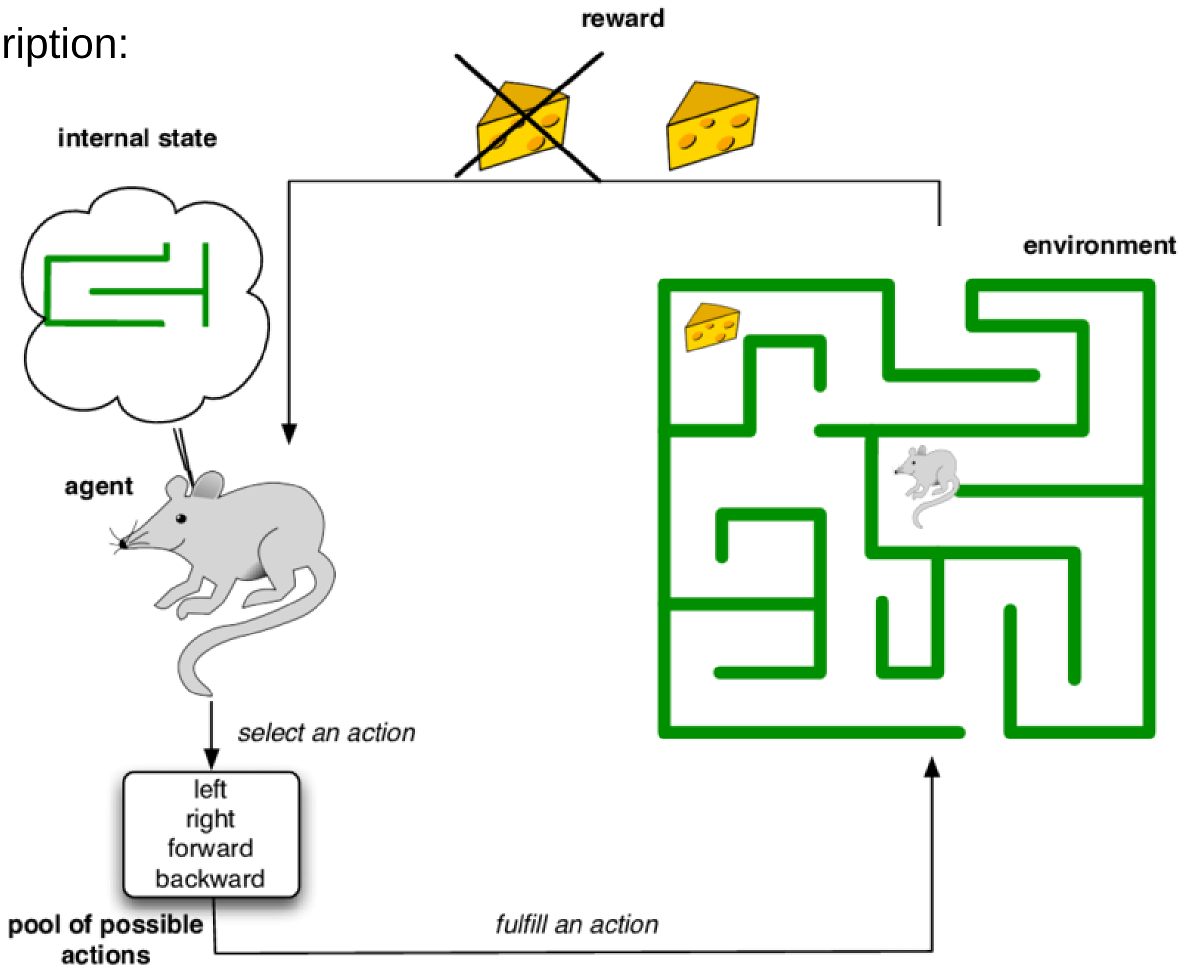
How to get to the cheese?



Source: https://www.researchgate.net/figure/General-reinforcement-learning-model-The-agent-selects-one-action-according-to-its_fig6_235930521

Agent – Environment Model

Problem description:



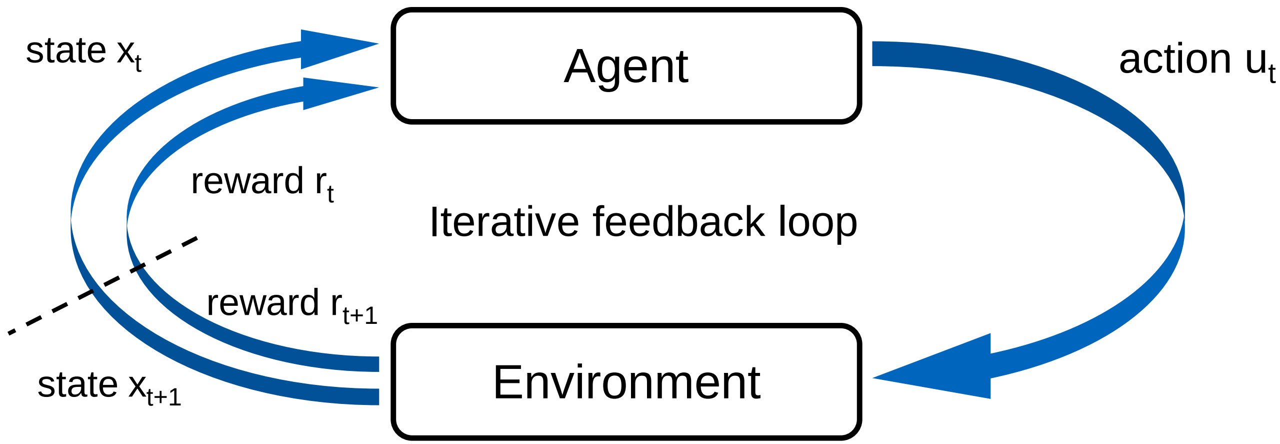
Source: https://www.researchgate.net/figure/General-reinforcement-learning-model-The-agent-selects-one-action-according-to-its_fig6_235930521

Agent – Environment Model

- **Agent:**
  Decision taking unit (a computer executing a policy π)
- **Environment:**
  Everything outside the agent (only consider relevant part for simplicity)
- **Reward r:**
  Scalar signal; Measurement for the agent‘s performance in the environment
- **(Internal) State x:**
  Signal describing the (relevant part) environment; e.g. position of the mouse
- **Action u:**
  Action performed by the agent based on a policy

Goal: Recieve as much reward as possible

Agent – Environment Model



Different notations often get mixed up

	State	Action	Reward
Richard Bellman	$\boldsymbol{s}_t$	$\boldsymbol{a}_t$	$r_{t+1}(\boldsymbol{s}_t, \boldsymbol{a}_t)$
Lev Pontryagin	$\boldsymbol{x}_t$	$\boldsymbol{u}_t$	$c_{t+1}(\boldsymbol{x}_t, \boldsymbol{u}_t)^*$
This course	$\boldsymbol{x}_t$	$\boldsymbol{u}_t$	$r_{t+1}(\boldsymbol{x}_t, \boldsymbol{u}_t)$

* $c_{t+1}(\boldsymbol{x}_t, \boldsymbol{u}_t)$ is a cost function: → $r_{t+1}(\boldsymbol{s}_t, \boldsymbol{a}_t) = -c_{t+1}(\boldsymbol{x}_t, \boldsymbol{u}_t)$

1. Agent – Environment Model
2. **Markov Decision Processes**
3. Value Function
4. Action Value Function
5. Q-Learning
6. Deep Q-Learning

Basic Probability Theory

Let‘s assume a normal dice:

Probability mass function $\boldsymbol{P(x = x_i) = P(x_i)}$:

$$P(x=1) = P(x=2) = P(x=3) = P(x=4) = P(x=5) = P(x=6) = \frac{1}{6}$$

$$\sum_i P(x = x_i) = \sum_i P(x_i) = 1$$

Conditional Probability $\boldsymbol{P(x|y)}$:

Throw 2 times: $\begin{cases} P(2) = \frac{1}{6} \cdot \frac{1}{6} = \frac{1}{36} & \rightarrow \text{P for both throws sum up to 2} \\ P(2|1) = \frac{1}{6} & \rightarrow \text{P for both throws sum up to 2, given the first is a 1} \end{cases}$

Expected value $\mathbb{E}^{\boldsymbol{P}}[\boldsymbol{x}]$:

$$\mathbb{E}^P[x] = \sum_i P(x_i) x_i = \frac{1}{6} \cdot 1 + \frac{1}{6} \cdot 2 + \cdots + \frac{1}{6} \cdot 6 = 3{,}5$$

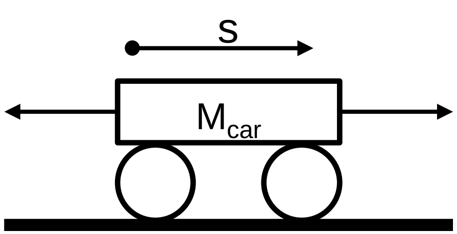
Source: https://www.dreamstime.com/stock-illustration-lucky-white-dice-isolated-white-background-object-image46997728

Markov Decision Processes

- **Markov State:**
  A state has Markov properities, if

$$\underbrace{P(x_{t+1}|x_t)}_{\text{Probability of reaching } x_{t+1} \text{ only given the current state } x_t} = \underbrace{P(x_{t+1}|x_t, x_{t-1}, x_{t-2}, \dots, x_0)}_{\text{Probability of reaching } x_{t+1} \text{ given the current state } x_t \text{ and all past states}}$$

→ A state has Markov properties, if the **past plays no role for describing the system state**



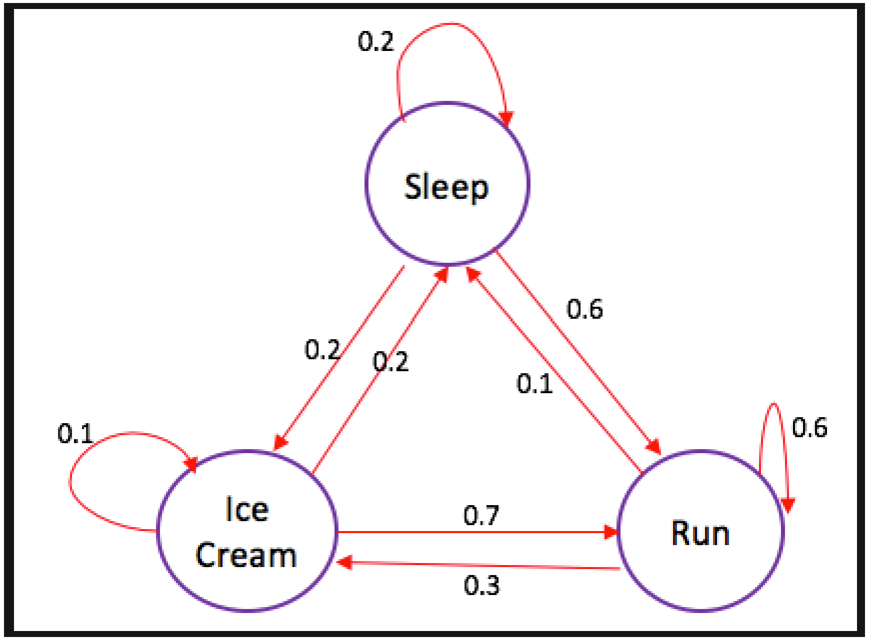
$x_t = s$ not Markov: to calculate v x_t and x_{t-1} are necessary

$x_t = \begin{bmatrix} s \\ \dot{s} \end{bmatrix}$ Markov: v is part of the state

Markov Decision Processes

Markov Process:

Sequence of random states with Markov property



Outgoing probabilities of each state have to sum up to 1:

$\sum_i P_{out,i} = 1$

Source: https://towardsdatascience.com/introduction-to-reinforcement-learning-markov-decision-process-44c533ebf8da

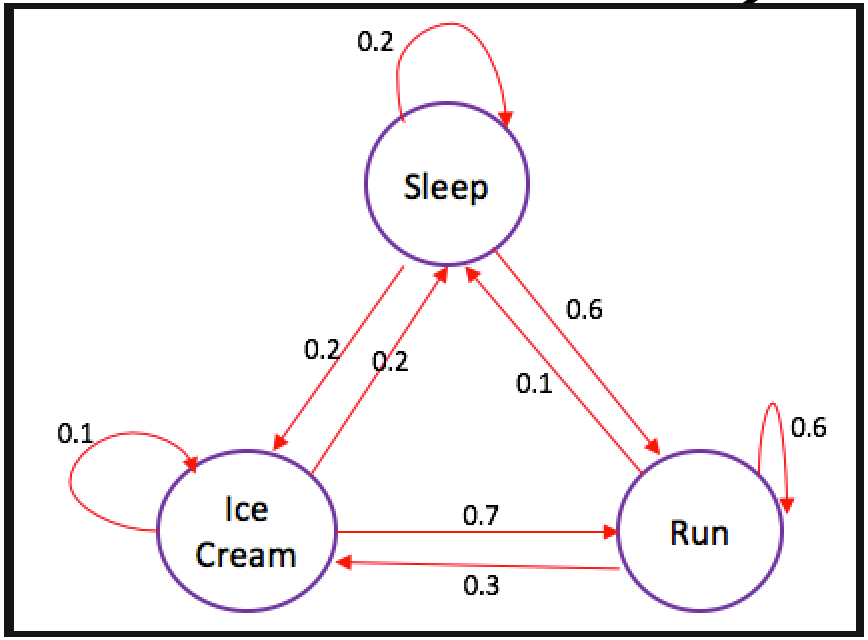
Markov Decision Processes

Markov process
+ rewards
+ possibility to affect transition probabilities
= Markov Decision Process (MDP)

Goal: Maximize future reward

$$\sum_{t=0}^{\infty} \gamma^t \cdot r_t$$

by finding p_i and q_i such that $\sum_i p_i = 1$ and $\sum_i q_i = 1$
$\gamma \in]0,1]$: discount factor → necessary for infinite processes; reward in near future is more important



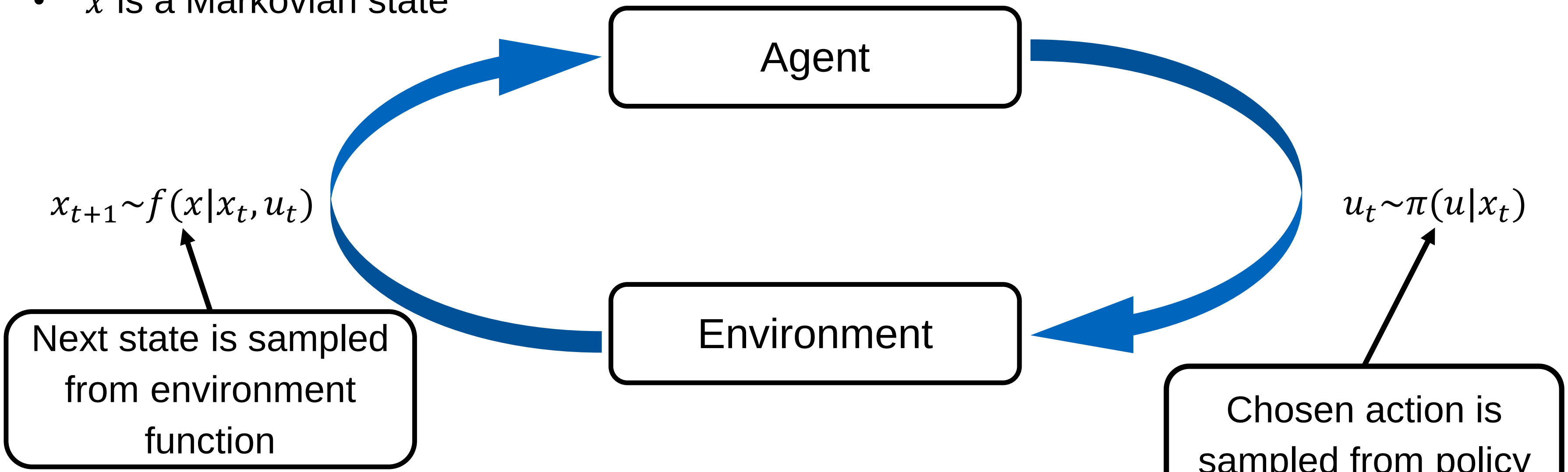
Source: https://towardsdatascience.com/introduction-to-reinforcement-learning-markov-decision-process-44c533ebf8da

Lehrstuhl AIS
Fakultät für Maschinenwesen
Technische Universität München

Markov Decision Processes

Assumptions:

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



Goal: Find strategy $\pi(u|x)$ which maximizes the reward

$\pi(u|x_t)$ is a probability mass function

→ Finding a strategy/policy = Optimizing the probabilities for the possible actions

Value Function

The expected future reward can be compute using the Value function:

$$V^{\pi}(x) = \underbrace{\mathbb{E}^{\pi}}_{\text{Expected reward}}\left[\underbrace{\sum_{\tau=t}^{\infty}}_{\text{for all future timesteps from now on}} \underbrace{\gamma^{\tau-t} r_{\tau+1}}_{\text{regarding the discount factor}} \underbrace{| x_t = x}_{\text{starting in the current state}}\right]$$

→ Function depending on the current state x_t and the policy π

Value Function

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

Extract first summand from sum:

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

$$\underbrace{V^{\pi}(x_t)}_{\text{Expected reward for the current state}} = \mathbb{E}^{\pi}[\underbrace{r_{t+1}}_{\text{Reward for the next transition}} + \underbrace{\gamma V^{\pi}(x_{t+1})}_{\text{Expected reward for the next state}} | x_t = x]$$

Value Function

Richard E. Bellman
1920 – 1984

So called Bellman equation (~1953 by Richard Bellman):

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

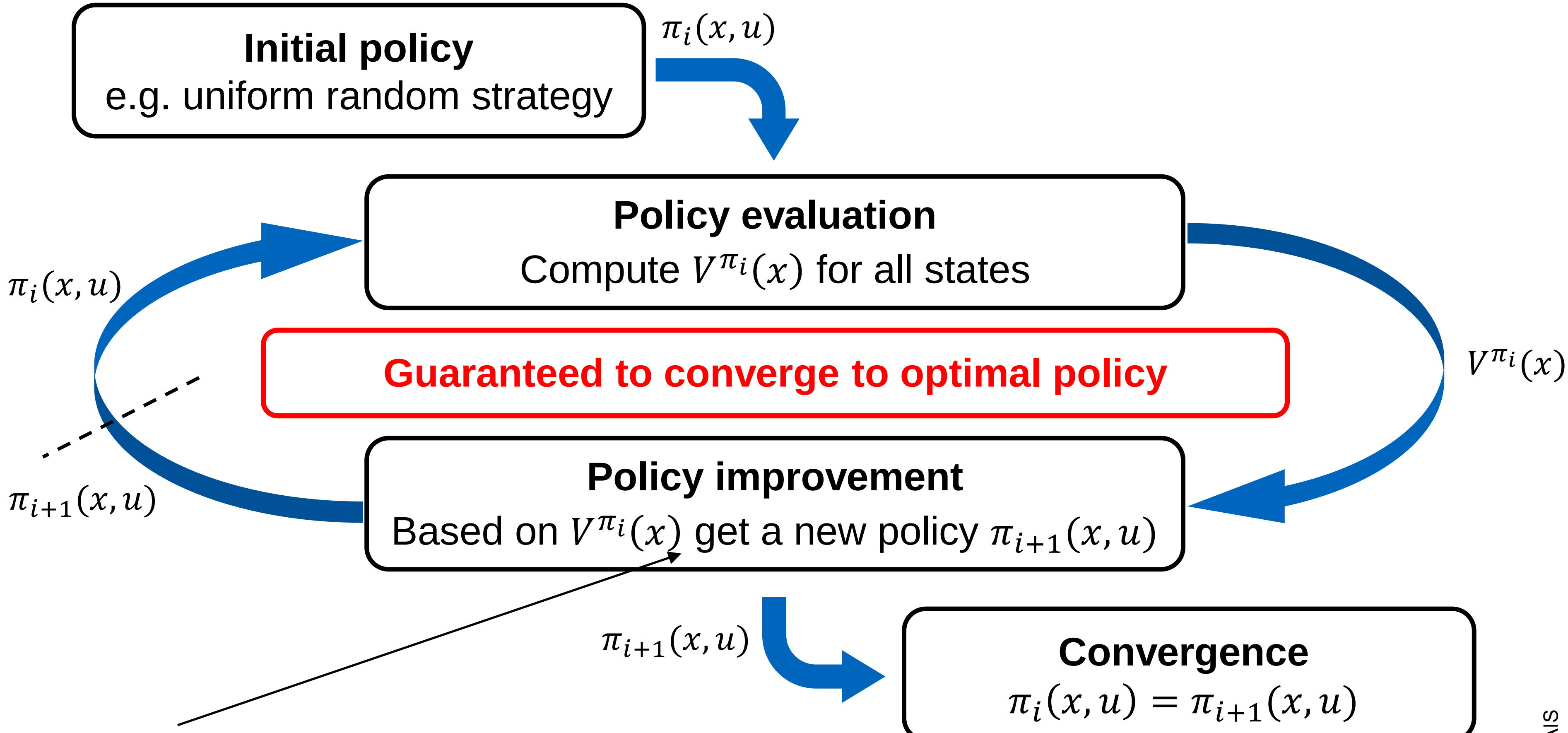
Bellman equation has to be solved for every state of the system

Solving the Bellman equation for a known MDP:
(transitions and policy must be known!)

1) Small MDP: Analytical solution possible
2) Else: Iterating over all states until convergence

Source: https://upload.wikimedia.org/wikipedia/en/7/7a/Richard_Ernest_Bellman.jpg

Value Function



e.g. choose greedy policy:

Lehrstuhl AIS
Fakultät für Maschinenwesen
Technische Universität München

Value Function

Reduce computing time:

→ Improve policy before convergence of value function

→ Worst case: policy stays the same

Initial policy
e.g. uniform random strategy

$\pi_i(x,u)$

***Partial* policy evaluation**
Compute $\tilde{V}^{\pi_i}(x)$ for all states

$\tilde{V}^{\pi_i}(x)$

Policy improvement
Based on $\tilde{V}^{\pi_i}(x)$ get a new policy $\pi_{i+1}(x,u)$

$\pi_{i+1}(x,u)$

$\pi_i(x,u)$

$\pi_{i+1}(x,u)$

Convergence
$\pi_i(x,u) = \pi_{i+1}(x,u)$

Lehrstuhl AIS
Fakultät für Maschinenwesen
Technische Universität München

Value Function – Greedy Algorithm

Definition:

“An algorithm that always takes the best immediate, or local, solution while finding an answer.”

For the Value Function:

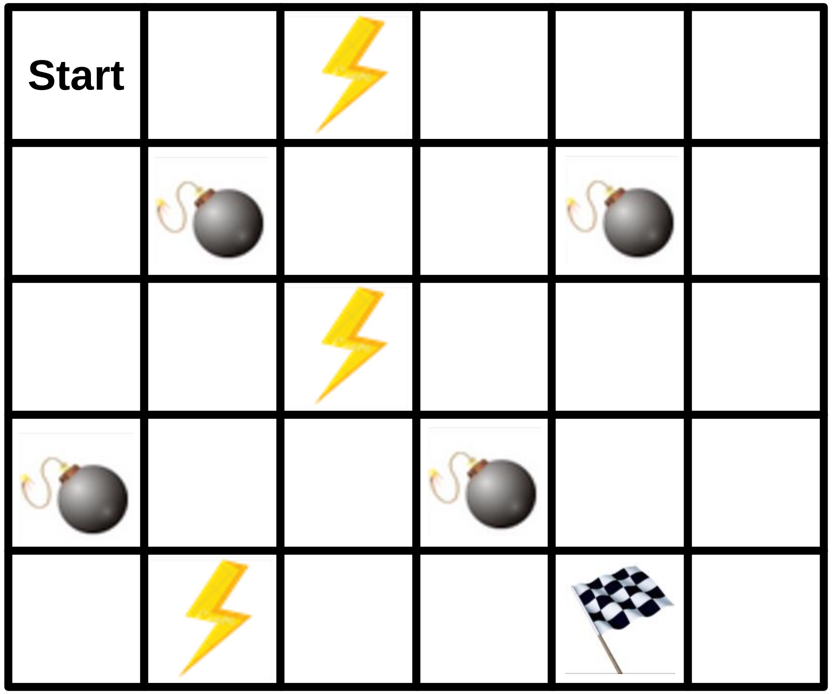
$$u_{\text{greedy}} = \underset{u}{\operatorname{argmax}}\, \mathbb{E}[r_{t+1} + \gamma V^{\pi}(x_{t+1}) | x_t = x, u_t = u]$$

Which leads to:

$$P_{\text{greedy}}\left(u_{\text{greedy}}\right) = 1$$
$$P_{\text{greedy}}\left(u_{\text{non-greedy}}\right) = 0$$

Paul E. Black, "greedy algorithm", in *Dictionary of Algorithms and Data Structures* [online], ed. 2 February 2005, https://www.nist.gov/dads/HTML/greedyalgo.html

Value Function



Example: Grid World

Move from Start to Target with maximum reward

Possible actions:
Move up, down, left, right

Rules:
- Each step costs 1 → r = -1
- Additional r = +1 for stepping on a lightning
- Additional r = -100 for stepping on a bomb and the game ends
- Additional r = +100 for reaching the target and the game ends
- Bumping into the wall counts as a step but the position stays the same

Sources: https://www.mlq.ai/deep-reinforcement-learning-q-learning/
https://pixers.de/fototapeten/wehend-zielflagge-auf-weisem-hintergrund-38483422

Lehrstuhl AIS
Fakultät für Maschinenwesen
Technische Universität München

Value Function

Calculating the Value Function:

Values for uniform random policy
after convergence (started with $V^\pi(x) = 0\ \forall x$)

Check for the green grid position:

=0, finite process

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

-106.161	-104.101	-103.143	-102.003	-100.286	-94.855
-104.221	0	-100.326	-99.578	0	-85.424
-102.501	-99.282	-96.583	-91.985	-68.777	-57.417
0	-95.043	-90.740	0	-21.707	-14.049
-95.076	-87.151	-68.334	-24.111	0	40.975

$$= \pi(\uparrow | x_{\text{green}}) \cdot \left[r_{\text{orange}} + V^\pi(x_{\text{orange}})\right] + \pi(\rightarrow | x_{\text{green}}) \cdot \left[r_{\text{yellow}} + V^\pi(x_{\text{yellow}})\right]$$

$$+ \pi(\downarrow | x_{\text{green}}) \cdot \left[r_{\text{green}} + V^\pi(x_{\text{green}})\right] + \pi(\leftarrow | x_{\text{green}}) \cdot \left[r_{\text{red}} + V^\pi(x_{\text{red}})\right]$$

$$= 0.25 \cdot \underbrace{[(-1 - 100) + 0]}_{\text{bomb}} + 0.25 \cdot \underbrace{[(-1 + 100) + 0]}_{\text{target}} + 0.25 \cdot [-1 + (-24.111)] + 0.25 \cdot [-1 + (-68.334)]$$

$$= -24.111$$

Sources: https://www.mlq.ai/deep-reinforcement-learning-q-learning/
https://pixers.de/fototapeten/wehend-zielflagge-auf-weisem-hintergrund-38483422

Action Value Function

Function depending on the current state, the policy and the next action

Similar to Value Function, but evaluated for each possible decision of a state

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

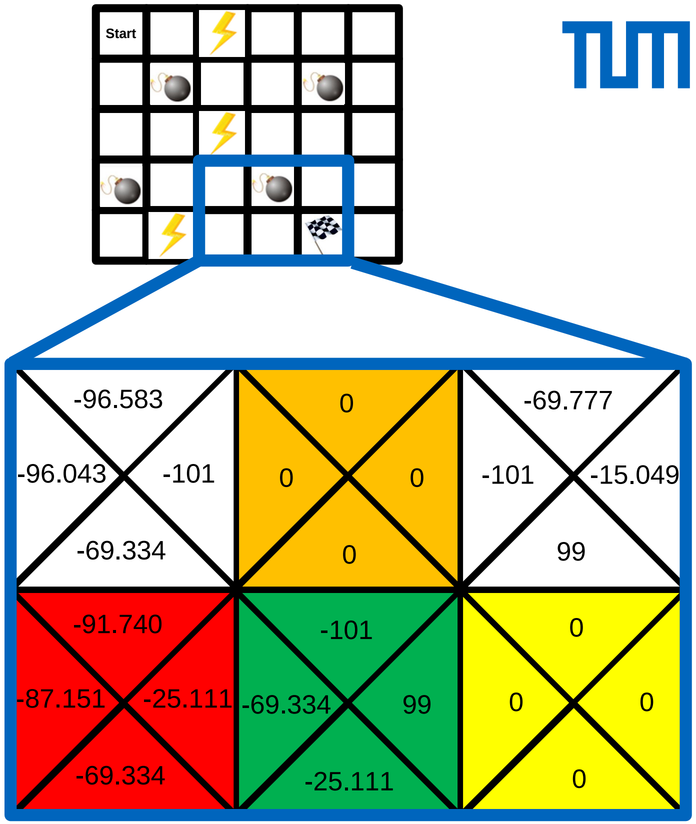
Advantage: No knowledge about transition probabilities of state x_t for policy improvement needed

Disadvantage: More memory required: (one value for each transition of each state) and more time for training

Policy: Choose transition with biggest Action Value Function ($u_{greedy}(x)$)

Application: Same way as Value Function

Action Value Function



Calculating the Action Value Function:

Q-Values for uniform random policy after convergence

Check for action „left“ in the green grid position:

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

$$Q^\pi(x_{\text{green}}, \leftarrow) = \mathbb{E}^\pi[r_{t+1} + \gamma Q(x_{t+1}, \pi(x_{t+1}))|x_{\text{green}} = x, u_{\text{green}} = u]$$

$$= r_{\text{red}} + [\pi(\uparrow|x_{\text{red}}) \cdot Q^\pi(x_{\text{red}}, \uparrow) + \pi(\rightarrow|x_{\text{red}}) \cdot Q^\pi(x_{\text{red}}, \rightarrow) + \pi(\downarrow|x_{\text{red}}) \cdot Q^\pi(x_{\text{red}}, \downarrow) + \pi(\leftarrow|x_{\text{red}}) \cdot Q^\pi(x_{\text{red}}, \leftarrow)]$$

$$= -1 + [0.25 \cdot (-91.740) + 0.25 \cdot (-25.111) + 0.25 \cdot (-69.334) + 0.25 \cdot (-87.151)]$$

$$= -69.334$$

1. Agent – Environment Model
2. Markov Decision Processes
3. Value Function
4. Action Value Function
5. **Q-Learning**
6. Deep Q-Learning

Lehrstuhl AIS
Fakultät für Maschinenwesen
Technische Universität München

Q-Learning

No model of environment

Occuring problems:

- Possible states?
- Possible state transitions?

Learning rate α: influences to what extent newly acquired information overrides old information

Idea:
Agent gets information about the environment by interaction with it

Policy evaluation:

Iterate over all data tuples $(x_t, u_t, r_{t+1}, x_{t+1})$:

$$Q_{k+1}^{\pi}(x_t, u_t) = \underbrace{(1-\alpha)\cdot Q_k^{\pi}(x_t, u_t)}_{\text{Influence of old Q-Value}} + \underbrace{\alpha \cdot \left(r_{t+1} + \gamma Q_k^{\pi}(x_{t+1}, u_{t+1})\right)}_{\text{Influence of new Q-Value}}$$

Policy improvement: $u_{greedy}(x) = \underset{u}{\operatorname{argmax}}\, Q(x, u)$

Q-Learning

Assumptions:

- Non-zero probability for all states of being visited
  (If the probability to reach a state is zero, optimal policy may not be found)
- Decreasing learning rate α
- Infinite learning time

But: Assumptions can be relaxed and you can still observe good results

ϵ-greedy strategy:

- $P(best\ action) = 1 - \epsilon$
- $P(all\ other\ actions) = \frac{\epsilon}{number\ of\ other\ actions}$
- No state has a zero-probability of being visited

Q-Learning

Combination of policy evaluation and policy improvement:

Policy evaluation

$$Q^{\pi}_{k+1}(x_t, u_t) = (1-\alpha)\cdot Q^{\pi}_{k}(x_t, u_t) + \alpha \cdot \left(r_{t+1} + \gamma Q^{\pi}_{k}(x_{t+1}, u_{t+1})\right)$$

Policy improvement

$$u_{greedy}(x) = \underset{u}{\operatorname{argmax}}\, Q(x, u)$$

Q-Learning

$$Q^{\pi}_{k+1}(x_t, u_t) = (1-\alpha)\cdot Q^{\pi}_{k}(x_t, u_t) + \alpha \cdot \left(r_{t+1} + \gamma \cdot \underset{u}{\operatorname{argmax}}\, Q(x_{t+1}, u)\right)$$

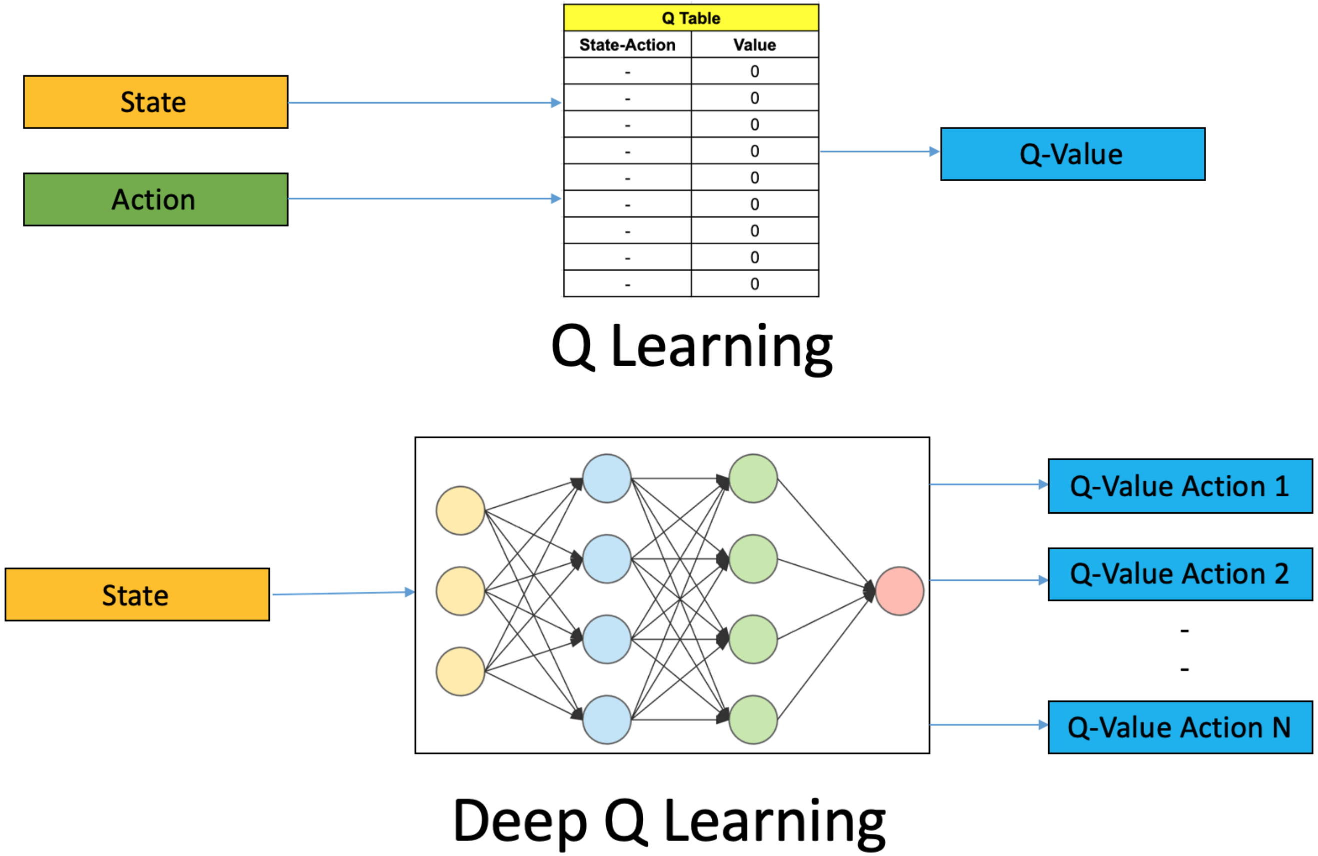
Works well for small environments

Big environments: Too much data to store

1. Agent – Environment Model
2. Markov Decision Processes
3. Value Function
4. Action Value Function
5. Q-Learning
6. **Deep Q-Learning**

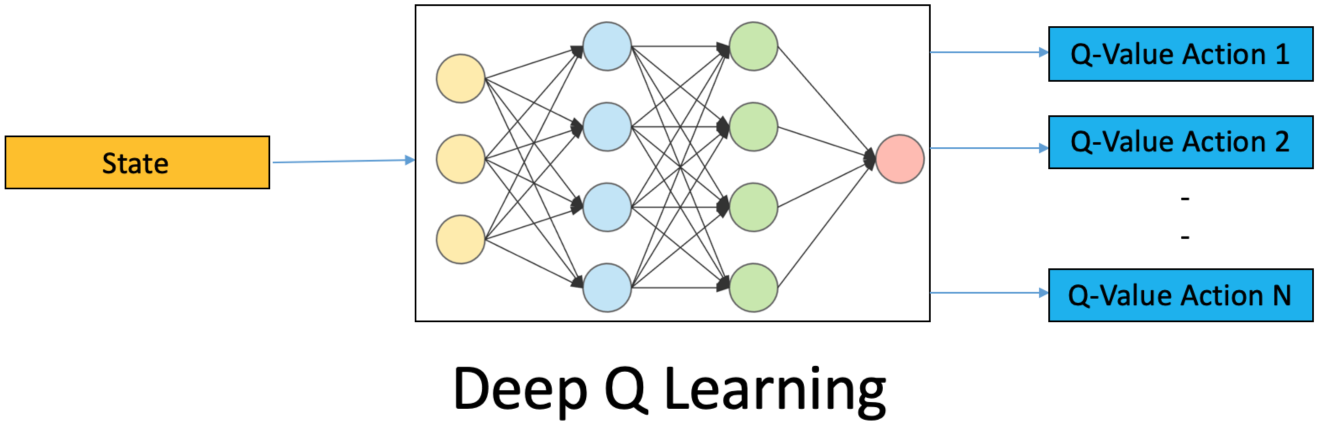
Deep Q-Learning

What is it? And where‘s the difference to Q-Learning?



Source: https://www.analyticsvidhya.com/blog/2019/04/introduction-deep-q-learning-python/

Deep Q-Learning



What should be used as a state?

→ DeepMind playing Breakout: pass the image/screen to the neural net

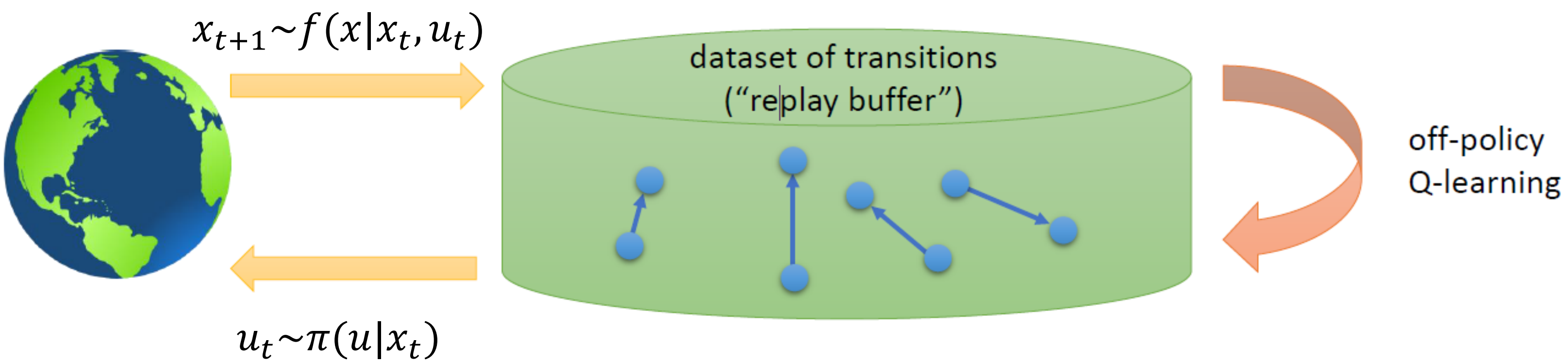
→ All necessary information can be extracted by the NN

→ Often LSTM-NNs are used since data is sequential

Source: https://www.analyticsvidhya.com/blog/2019/04/introduction-deep-q-learning-python/

Deep Q-Learning

Usual setup:
Separate executing and training the policy:



Source: Reinforcement Learning, Lecture 8: Deep RL with Q-Functions, Sergey Levine, UC Berkeley