Research and Innovation Center of Excellence

MSc on Intelligent Critical Infrastructure Systems

Machine Learning

Lecture 12: Reinforcement Learning (part II)

Kleanthis Malialis

Research Associate

KIOS Research and Innovation Center of Excellence

University of Cyprus

University of Cyprus

Imperial College London

FUNDED BY:

Course outline

- **Week 1**
  - Introduction and Preliminaries
- **Week 2**
  - Linear Regression
  - Regularisation, Logistic Regression, SVMs
- **Week 3**
  - Neural Networks and Deep Learning
- **Week 4**
  - Feature Engineering and Evaluation
  - Online Learning
- **Week 5**
  - Unsupervised Learning
- **Week 6**
  - **Reinforcement Learning**
- **Week 7**
  - Monitoring and Control

Recap

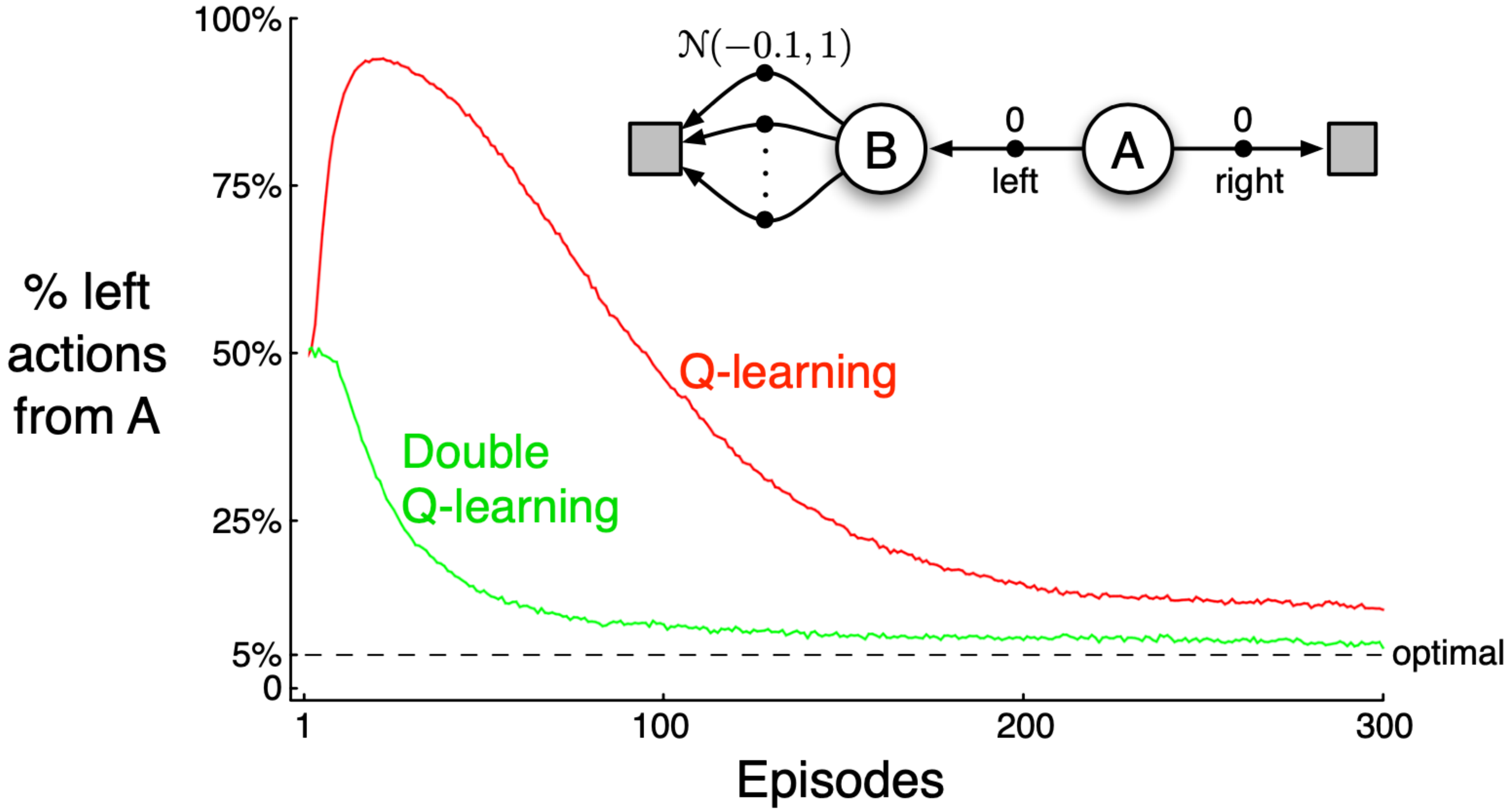
- **Key characteristics of RL**
  - "carrot and stick" approach, i.e., reward / penalty signals
  - sequential decision making / control tasks
  - delayed rewards
- **Markov Decision Process (MDP)**
- **Discounted expected return** and **Value functions**
- **Temporal-Difference (TD) learning**
  - On-policy (SARSA) vs off-policy (Q-learning) control
  - Exploration – exploitation dilemma

Outline

- Tabular solution methods (cont’d)
- Approximate solution methods
- Multi-agent RL
- Imitation learning

TABULAR SOLUTION METHODS (CONT'D)

Maximisation bias



Q-learning

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$$

Old Q-value

Immediate reward

Discount factor

Estimate of optimal future value

Old Q value

Learning rate

Double Q-learning

- Learn **two independent** Q functions, on different sets of experience.
- The first Q-function is used for **action selection**.
- The second Q-function is used for **action evaluation**.

At each time step:

with 50% probability:

$$\mathrm{Q}_1(S_t, A_t) = \mathrm{Q}_1(S_t, A_t) + \alpha[R_{t+1} + \gamma \boldsymbol{Q_2}(S_{t+1}, \mathbf{argmax}_{\boldsymbol{a}} \boldsymbol{Q_1}(\boldsymbol{S_{t+1}}, \boldsymbol{a})) - \mathrm{Q}_1(S_t, A_t)]$$

else:

$$\mathrm{Q}_2(S_t, A_t) = \mathrm{Q}_2(S_t, A_t) + \alpha[R_{t+1} + \gamma \boldsymbol{Q_1}(S_{t+1}, \mathbf{argmax}_{\boldsymbol{a}} \boldsymbol{Q_2}(\boldsymbol{S_{t+1}}, \boldsymbol{a})) - \mathrm{Q}_2(S_t, A_t)]$$

Convergence criteria

- The environment’s states are Markov states.
- **An agent visits all state-action pairs infinitely often.**
- The learning rate reduces to zero.
- The exploration rate reduces to zero (for SARSA).
- Bounded rewards.

Tabular solution methods

- So far, we have represented the value function Q as a **lookup table** with an entry for every state-action pairs (s, a)
- Recall that one of the convergence criteria of Q-learning is for an agent to visit all state-action pairs infinitely often.
- **Problems**:
    - In some cases, it is impossible to satisfy the above criterion, e.g., the game of Go (19x91) has 10^{170} states.
    - Some domains require continuous state and/or action space (e.g., mobile robot).
    - Too many states and/or actions to store in memory.
    - Too slow to learn the value for each state-action.

APPROXIMATE SOLUTION METHODS

RL with function approximation

- Use function approximation to estimate the value function, and generalise from seen states to unseen states.
- The approximate value function is not represented as a table, but as a parameterized functional form.
- Approximate solution methods:
    - Linear methods
    - Deep RL

Linear methods

- The approximate function $v(s;w)$ is a linear function of the weight vector $w = (w_1, w_2, \ldots, w_d)$.

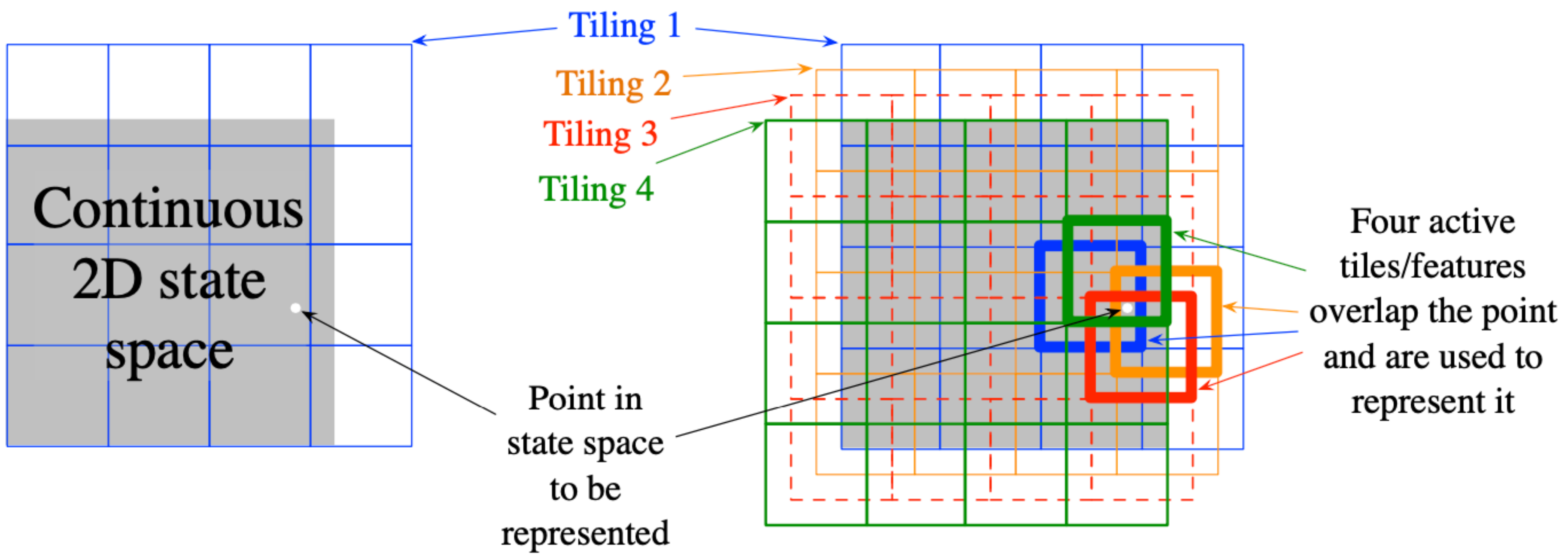
- Corresponding to each state s, there is a real-valued vector $x(s) = (x_1(s), x_2(s), \ldots, x_d(s))$.

- Linear methods approximate the state value function as follows:

$$v(s;w) = w^T x(s) = \sum_{i=1}^{d} w_i x_i(s)$$

Tile Coding

- Tile Coding partitions the state space into **tilings**, each tiling is further partitioned into **tiles** where state feature values are grouped into.
- The number of tilings indicate the degree of **resolution**.
- The size / shape of tiles indicate the nature of **generalisation**.


Tiling 1
Tiling 2
Tiling 3
Tiling 4
Continuous 2D state space
Point in state space to be represented
Four active tiles/features overlap the point and are used to represent it

Tile Coding (2)

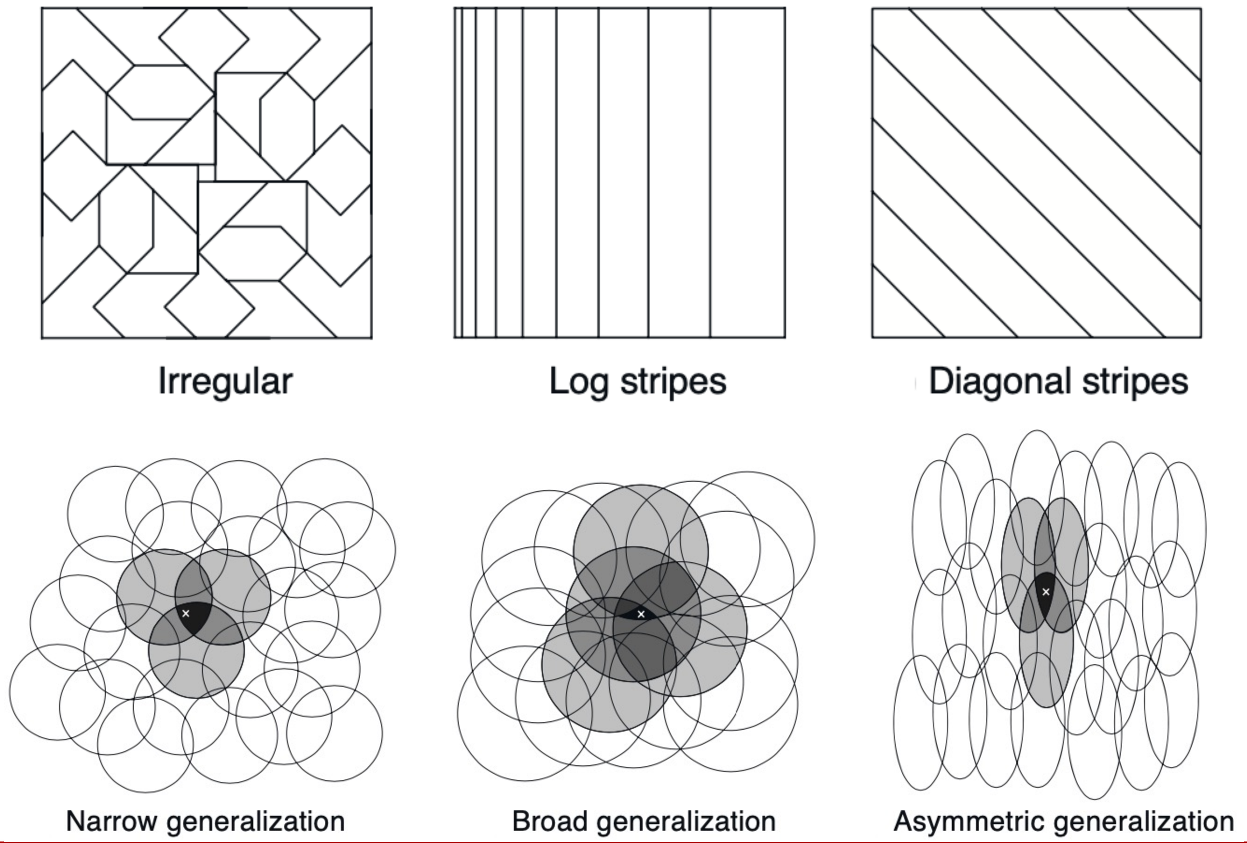
- If the state is inside a tile, then the corresponding feature has the value 1, otherwise the feature is 0. This kind of 0/1-valued feature is called a **binary feature**.
- Tile Coding activates $m < d$ tiles: $TC(s) \to (x_1, x_2, \dots, x_m)$
- The Q-value is now calculated as: $Q(s,a) = Q(TC(s),a) = \sum_{i=1}^{m} Q(x_i,a)$
- The update rule is then:

$$\Delta Q = r + \gamma \max_a Q(s',a) - Q(s,a)$$

$$Q(x_i,a) = Q(x_i,a) + \frac{\alpha}{m}\Delta Q$$

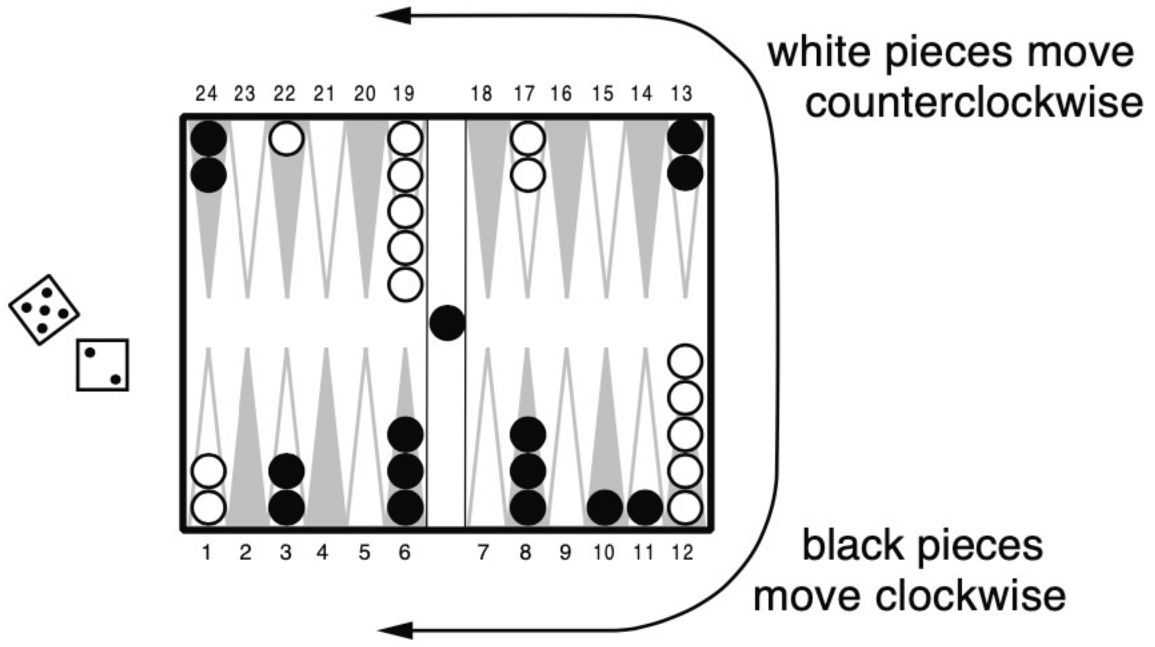
Tile Coding (3)

- Tilings need not be grids, they can be arbitrarily shaped and non-uniform. However, these are rarely used in practise as Tile Coding is flexible and computationally efficient.



Mastering the game of Backgammon (IBM, 1995)

- One of the earliest and most impressive applications of RL to date is **TD-Gammon** by Gerald Tesauro.
- The algorithm combined TD-learning with non-linear function approximation using a multi-layered neural network.

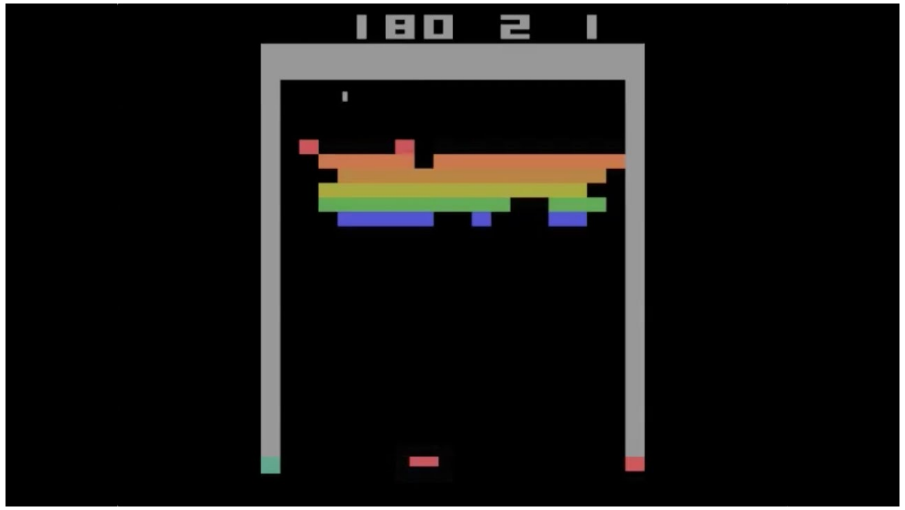

24 23 22 21 20 19
18 17 16 15 14 13
1 2 3 4 5 6
7 8 9 10 11 12
white pieces move counterclockwise
black pieces move clockwise

A backgammon position

Paper

Human-level control of Atari games (DeepMind, 2015)

- DeepMind (U.K.) was bought by Google for $500M in 2014!
- The **Deep Q-Network (DQN)** agent, receiving only the pixels and the game score as inputs, was able to achieve a level comparable to that of a professional human games tester across a set of 49 games, using the same algorithm, network architecture and hyperparameters.



nature
THE INTERNATIONAL WEEKLY JOURNAL OF SCIENCE
LEARNING CURVE
Self-taught AI software attains human-level performance in video games
PAGES 486 & 529
SHARE DATA IN OUTBREAKS
A GIANT IN THE EARLY UNIVERSE
TELEPORTATION FOR TWO

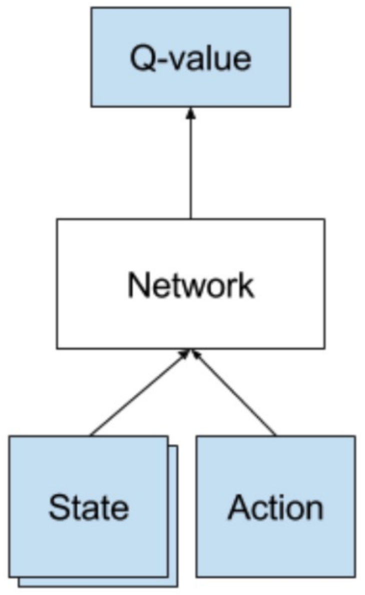
Paper, video

Deep RL

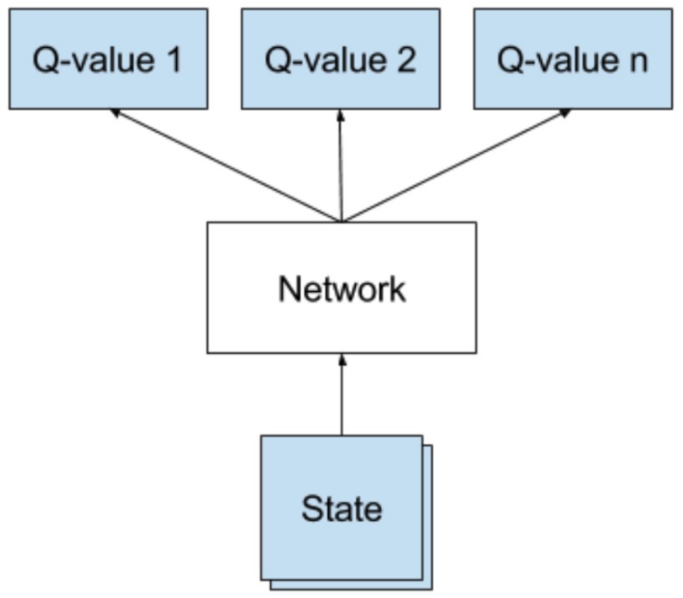
- Neural networks as functions approximators have been around for a long time (e.g., TD-Gammon 1995).
- Key elements introduced by DQN:
  - Neural network architecture
  - Experience replay
  - Target network

DQN: Neural architecture

- **Input**: Current state vector
- **Output**: Contrary to the traditional RL setup, DQN produces a Q-value for each state-action pair in a **single forward pass**.



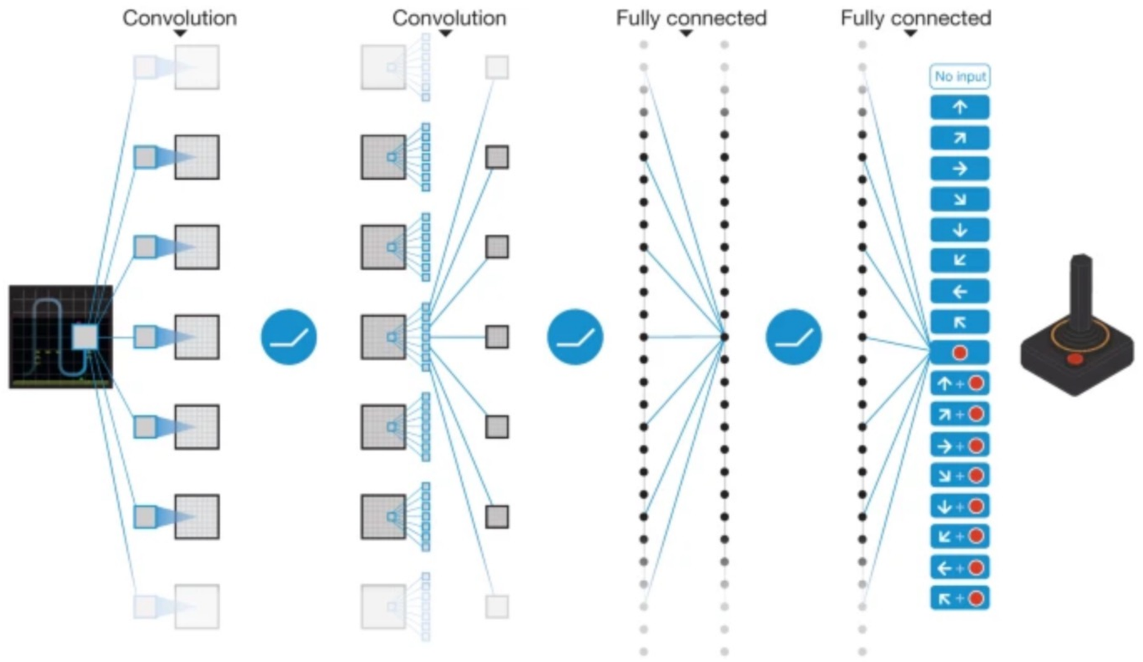
Naive



DQN

DQN: Neural architecture (2)

- **Input**: Current state vector
- **Output**: Contrary to the traditional RL setup, DQN produces a Q-value for each state-action pair in a **single forward pass**.



DQN: Experience replay

- Inspired from biological systems
  - Neuroscientists have discovered that spontaneous recollections, measured directly in the brain, often occur as very fast sequences of multiple memories. Neural “replay” sequences were originally discovered by studying the hippocampus in rats.

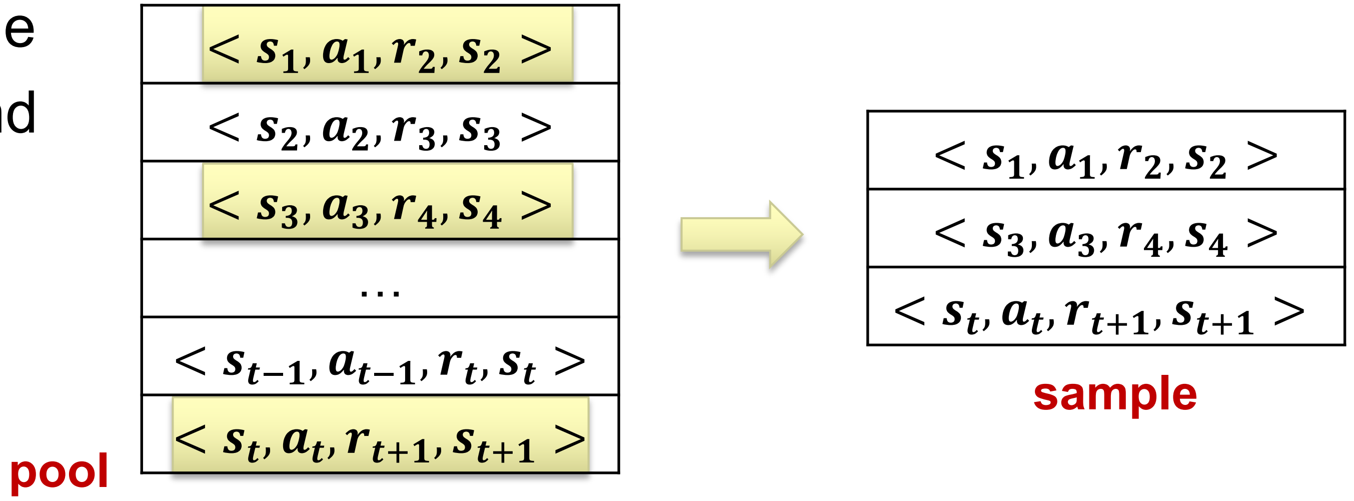
- Experience replay
  - Contrary to the traditional RL setup, all experiences $\mathbf{e}_\mathrm{t} =< s_t, a_t, r_{t+1}, s_{t+1} >$ are stored in a memory / buffer.

$< s_1, a_1, r_2, s_2 >$
$< s_2, a_2, r_3, s_3 >$
$< s_3, a_3, r_4, s_4 >$
$\dots$
$< s_{t-1}, a_{t-1}, r_t, s_t >$
$< s_t, a_t, r_{t+1}, s_{t+1} >$

DQN: Experience replay (2)

- **Advantages**
  - Each experience can potentially be used for many updates, thus being more data efficient.
  - Some experiences may be rare; it'd be useful if we could recall them.

- **Problem**: Sample correlations
  - It causes instability in the NN's performance.
  - **Solution**: Draw a random sample from the pool of experiences, and use that to train the network.



DQN: Loss function

- DQN brings Q-learning closer to supervised learning (backpropagation, stochastic / mini-batch GD) while still allowing it to bootstrap.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[\underbrace{R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)}_{\text{TD error}}]$$

$$L = \frac{1}{2}\Big[\underbrace{R_{t+1} + \gamma \max_a Q(S_{t+1}, a; w)}_{\text{target}} - \underbrace{Q(S_t, A_t; w)}_{\text{prediction}}\Big]^2$$

$$\Delta w = \alpha\left(R_{t+1} + \gamma \max_a Q(S_{t+1}, a; w) - Q(S_t, A_t; w)\right)\nabla_{\mathrm{w}} Q(S_t, A_t; w)$$

DQN: Target network

- **Problem: Non-stationary target**
  - The loss function's dependence on w complicates the process compared to the simpler supervised-learning situation in which the targets do not depend on the parameters being updated. This is another source of instability.

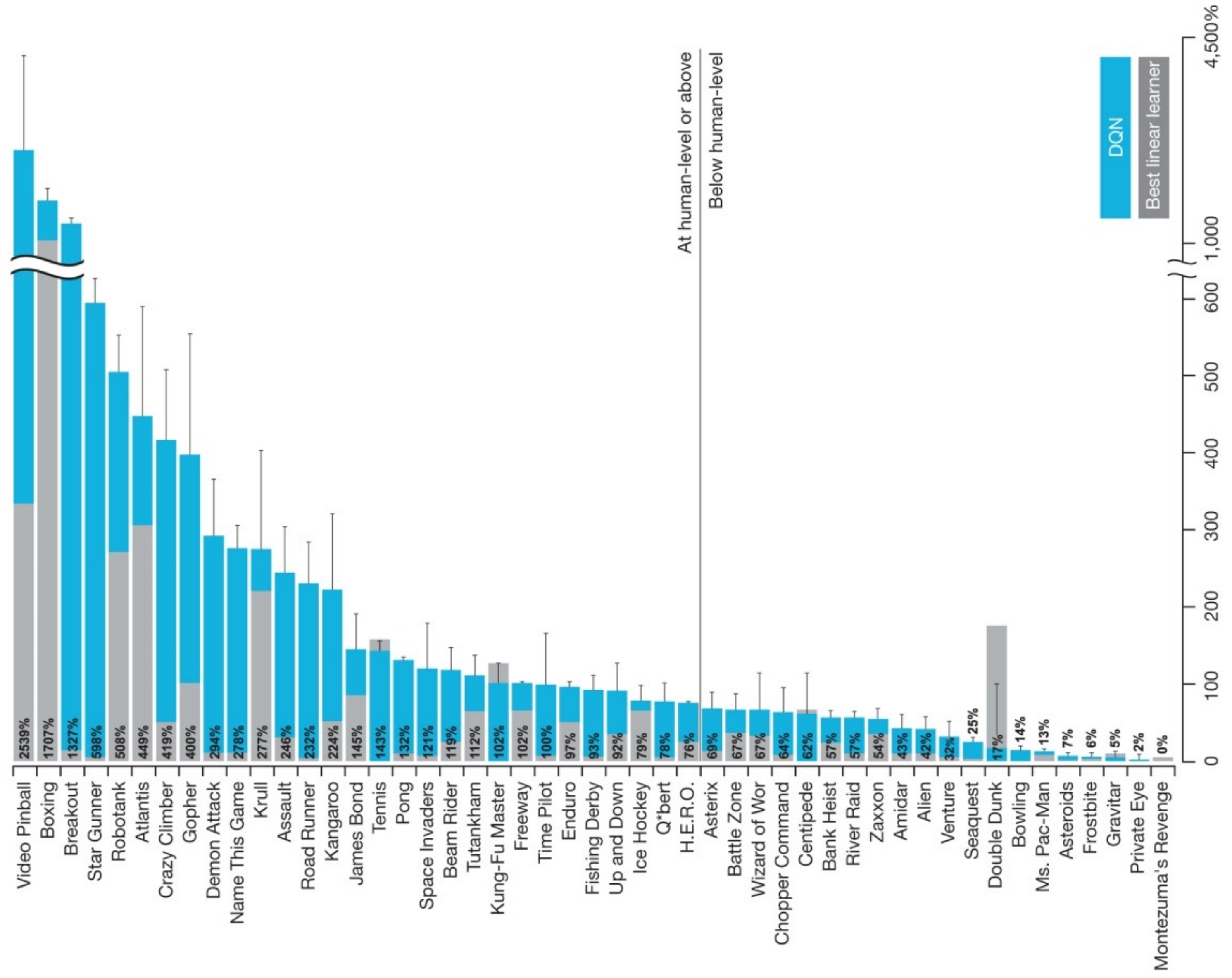
- **Solution: Fix Q-targets**
  - To address the non-stationary target issue, DQN uses a separate target network ($\boldsymbol{Q}^{\sim}$) to estimate the target. The target network has the same architecture as the function approximator but with frozen parameters. The parameters from the prediction network are copied to the target network at every C iterations.

$$L = \frac{1}{2}\left[\underbrace{R_{t+1} + \gamma \max_{a} Q^{\sim}(S_{t+1}, a; w)}_{\textbf{target}} - \underbrace{Q(S_t, A_t; w)}_{\textbf{prediction}}\right]^2$$

DQN: other elements

- **State**
  - A human playing any Atari game sees 210x160 pixel image frames with 128 colors at 60Hz. To reduce memory and processing requirements, the authors preprocessed each frame to produce an 84x84 array of luminance values.
  - The authors “stacked” the four most recent frames so that the input dimension is 84x84x4. This did not eliminate partial observability for all of the games, but it was helpful in making many of them more Markovian.
- **Reward function**
  - The reward indicated how a game’s score changed from one time step to the next: +1 whenever it increased, 1 whenever it decreased, and 0 otherwise. This standardized the reward signal across the games.
- **Error clipping**
  - The authors clipped the TD error in [-1,+1].

DQN: Results



DQN
Best linear learner
At human-level or above
Below human-level
0
100
200
300
400
500
600
1,000
4,500%
Video Pinball 2539%
Boxing 1707%
Breakout 1327%
Star Gunner 598%
Robotank 508%
Atlantis 449%
Crazy Climber 419%
Gopher 400%
Demon Attack 294%
Name This Game 278%
Krull 277%
Assault 246%
Road Runner 232%
Kangaroo 224%
James Bond 145%
Tennis 143%
Pong 132%
Space Invaders 121%
Beam Rider 119%
Tutankham 112%
Kung-Fu Master 102%
Freeway 102%
Time Pilot 100%
Enduro 97%
Fishing Derby 93%
Up and Down 92%
Ice Hockey 79%
Q*bert 78%
H.E.R.O. 76%
Asterix 69%
Battle Zone 67%
Wizard of Wor 67%
Chopper Command 64%
Centipede 62%
Bank Heist 57%
River Raid 57%
Zaxxon 54%
Amidar 43%
Alien 42%
Venture 32%
Seaquest 25%
Double Dunk 17%
Bowling 14%
Ms. Pac-Man 13%
Asteroids 7%
Frostbite 6%
Gravitar 5%
Private Eye 2%
Montezuma's Revenge 0%

Prioritised Experience Replay

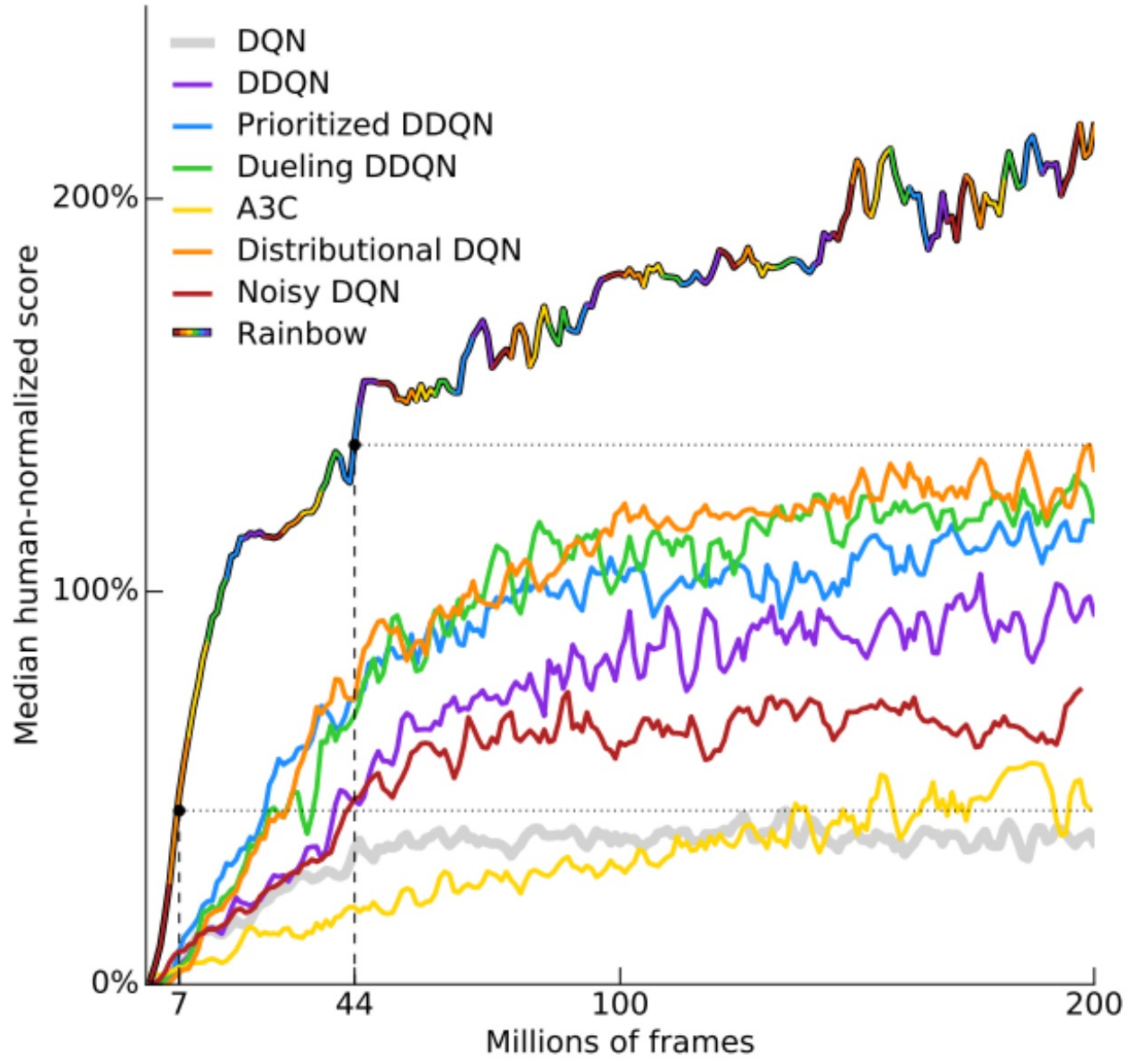
- Experience Replay does not consider the importance of an experience $e_j =< s_j, a_j, r_{j+1}, s_{j+1} >$.
- **Idea**: sample experiences based on a priority function.
- Priority is proportional to TD error.

$$p_j = |R_{j+1} + \gamma \max_a Q^{\sim}(S_{j+1}, a; w) - Q(S_j, A_j; w)|$$

- Priority function:
    - Parameter α determines "how much" prioritisation is used.

$$P(j) = \frac{p_j^\alpha}{\sum_k p_k^a} \qquad \boxed{\alpha = 0?}$$

Rainbow



Drawbacks

- Slow training
- Hard to tune
- Hard to implement / debug

MULTI-AGENT RL

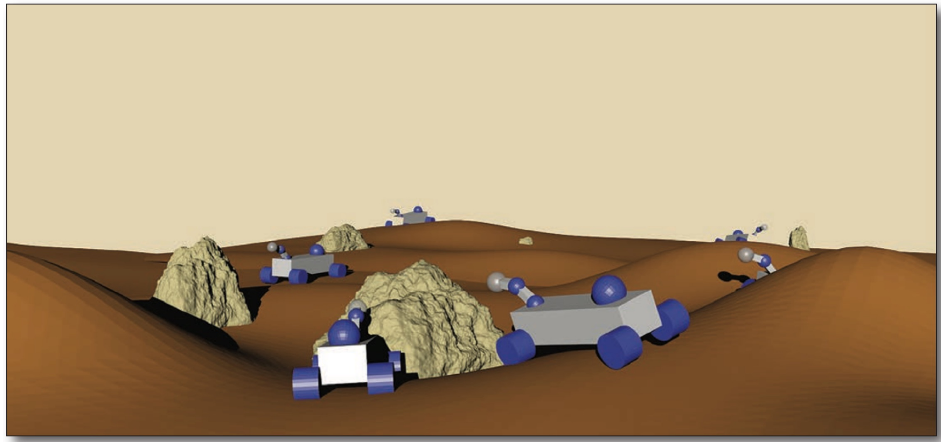
Multi-agent systems and RL

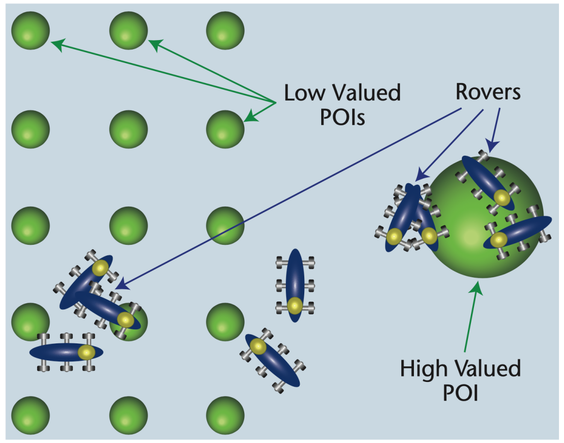
- *“A **multiagent system** deals with the construction of a system involving multiple autonomous and interacting agents, which are situated in a common environment that can perceive through sensors, and act upon it through actuators.”* (Busoniu et al., 2008)

- *“Multiagent systems often need to be very complex, and **multiagent reinforcement learning** is a promising candidate for dealing with this emerging complexity.”* (Stone & Veloso 2000).

Multi-agent RL: Applications

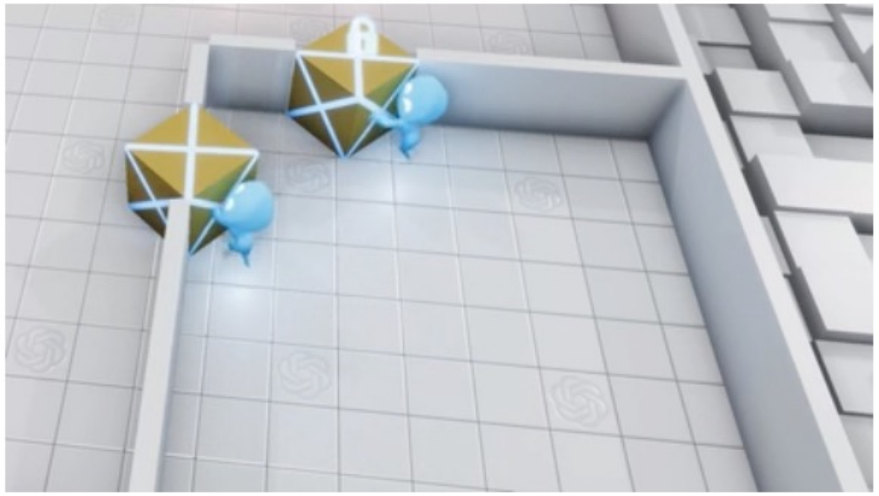
- **Space exploration**





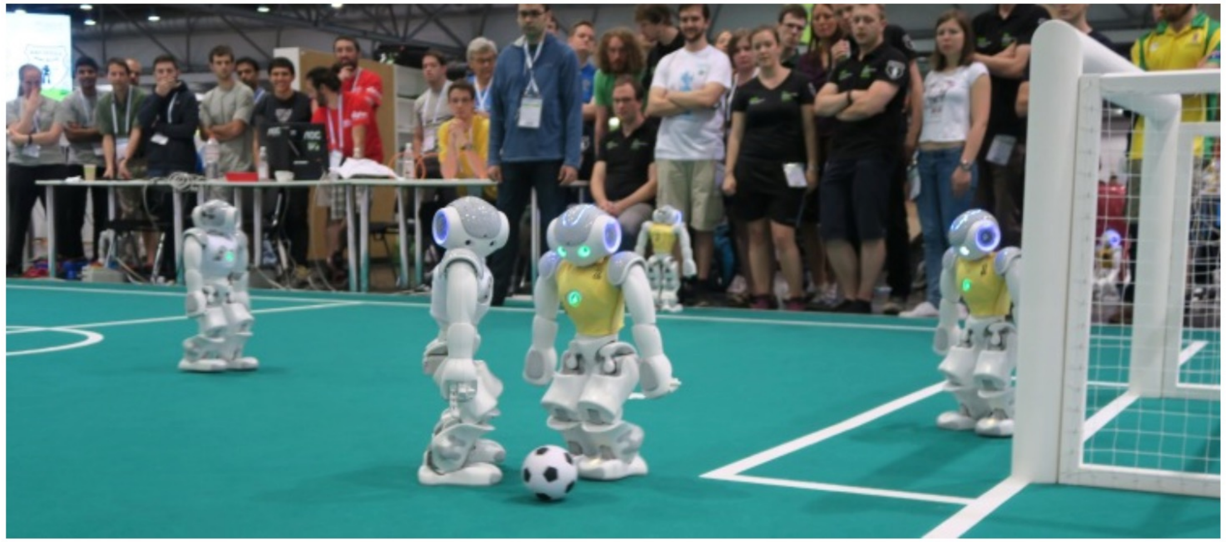
- **Hide-and-Seek**
  - Agents discover progressively more complex tool use while playing a simple game of hide-and-seek.

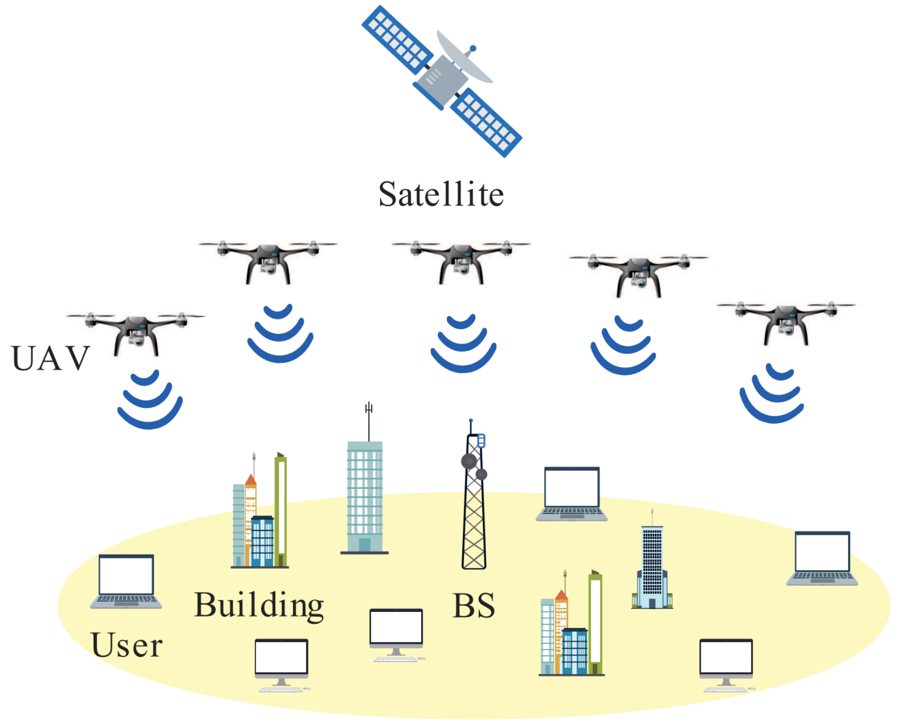
Video



Multi-agent RL: Applications (2)

- **Robotic soccer (Robocup)**
  - *“By the middle of the 21st century, a team of fully autonomous humanoid robot soccer players shall win a soccer game, complying with the official rules of FIFA, against the winner of the most recent World Cup.”* Website



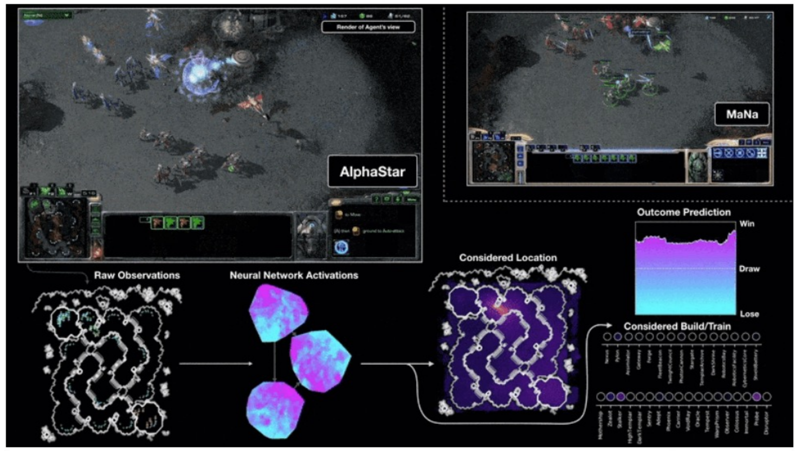


- **UAV-aided communication network**

Multi-agent RL: Applications (3)

- **Expert-level performance in Dota 2 (OpenAI)**
  - It learned by playing over 10,000 years of games against itself.
  - OpenAI receives a $1B investment from Microsoft!

- **Mastering StarCraft II (DeepMind)**
  - *"Despite the recent successes (Atari, Dota 2), AI techniques have struggled to cope with the complexity of StarCraft II.* ***AlphaStar*** *is the first AI to defeat a top professional player."*



Multi-agent RL: Challenges

- **Curse of dimensionality**
  - Table-based methods are also affected by this. In MARL cases, the problem is worse as the complexity is now exponential in the number of agents in the system.

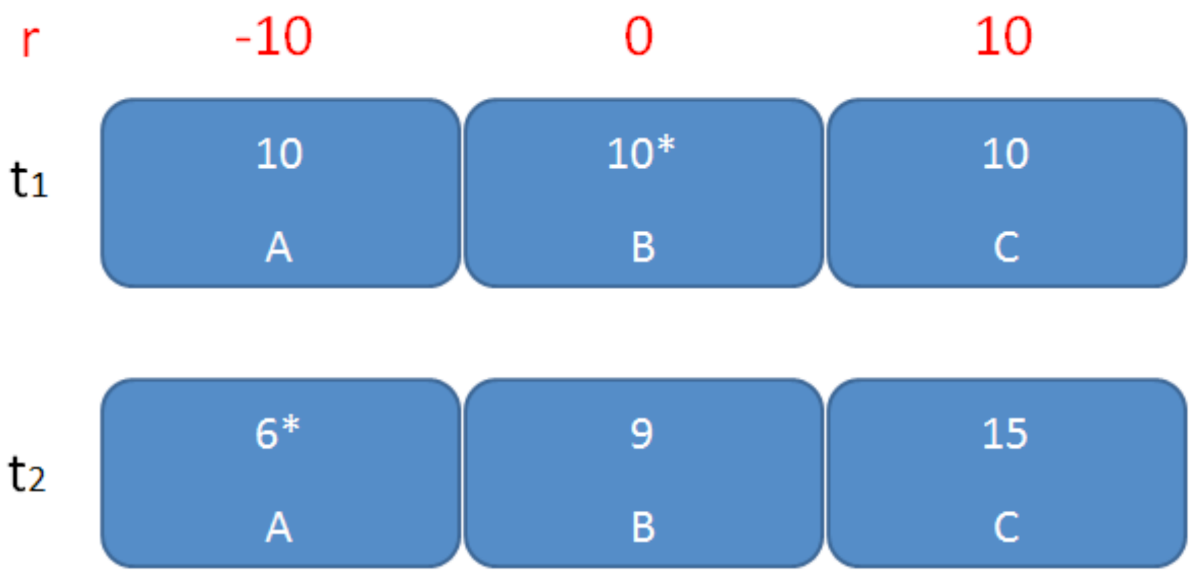
Function approximation, Agent organisation

- **Partial observability** and **Non-stationarity**
  - Each agent cannot observe the entire environment, but has only knowledge about its local environment.
  - The effects of an agent's action on the environment, also depend on the other agents' actions. Therefore, the Markov property does not hold if an agent cannot observe the joint state / action.

Agent communication

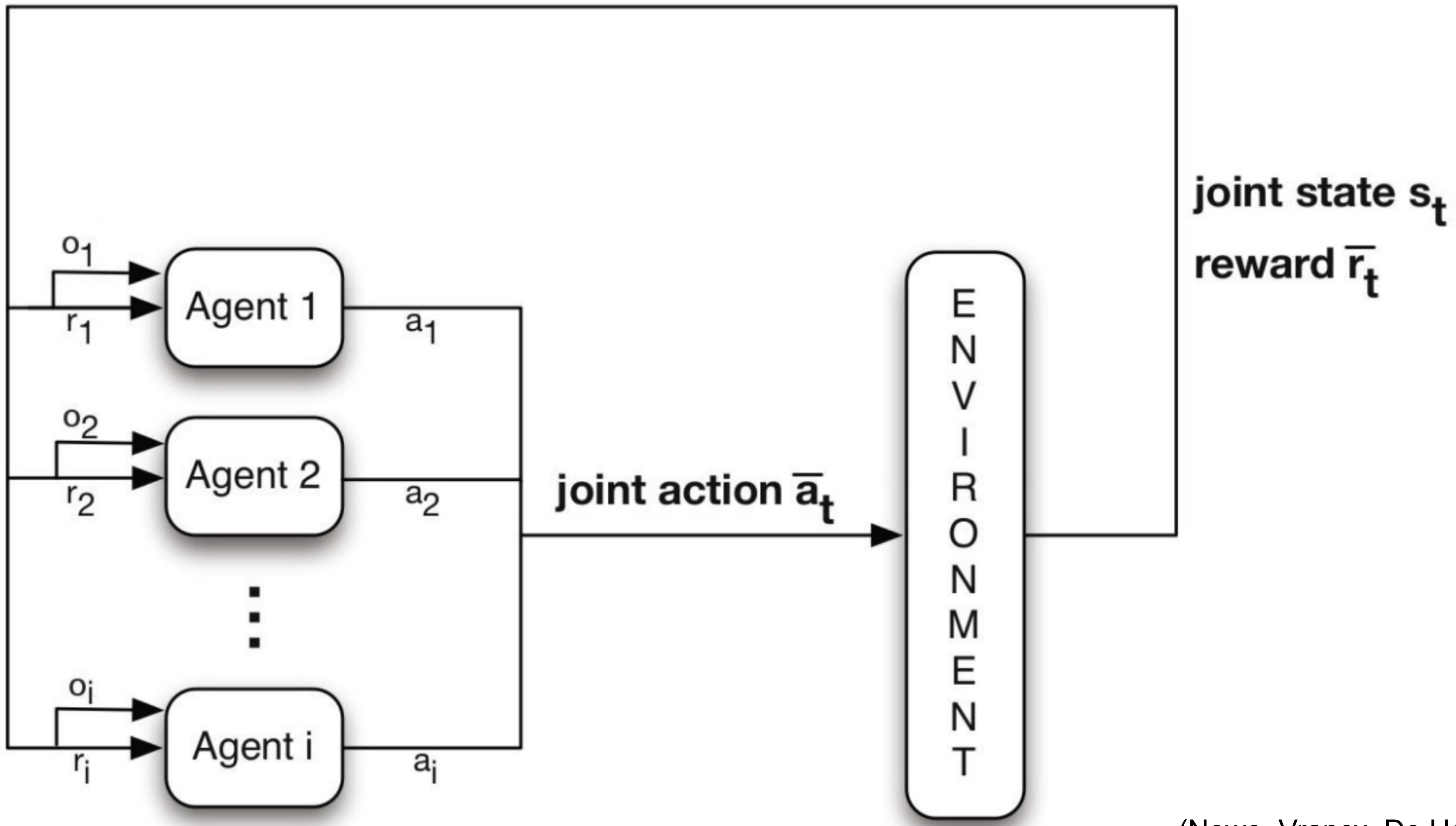
Multi-agent RL: Challenges (2)

- **Multiagent credit assignment**
  - In a **cooperative** multiagent system an agent is, typically, provided with a reward at the global / system level. Given that other agents also interact with the environment, an agent may be rewarded for taking a bad action, or punished for taking a good action.



Reward shaping

Multi-agent RL



(Nowe, Vrancx, De Hauwere, 2012)

Independent Learners (ILs)

- This involves the deployment of multiple agents each using a single-agent RL algorithm.

- Each IL **assumes any other agents to be a part of the environment** and so, as the others simultaneously learn, the environment appears to be dynamic as the probability of transition when taking action *a* in state *s* changes over time.

- If an IL i uses SARSA, then its update rule is:

$$\boldsymbol{Q_i(O_t, A_t) = Q_i(O_t, A_t) + \alpha[R_{t+1} + \gamma\, Q_i(O_{t+1}, A_{t+1}) - Q_i(O_t, A_t)]}$$

Joint Action Learners (JALs)

- This approach considers the existence of other agents. Specifically, each agent **observes the actions of the other agents, or each agent communicates its action to the others.**

- To overcome the appearance of a dynamic environment, JALs extend their value function to consider for each state the value of each possible combination of actions by all agents.

- If a JAL i uses SARSA, then its update rule is:

$$\boldsymbol{Q_i}(\boldsymbol{O_t}, \boldsymbol{A_t^{\rightarrow}}) \leftarrow \boldsymbol{Q_i}(\boldsymbol{O_t}, \boldsymbol{A_t^{\rightarrow}}) + \boldsymbol{\alpha}[\boldsymbol{R_{t+1}} + \boldsymbol{Q_i}(\boldsymbol{O_{t+1}}, \boldsymbol{A_{t+1}^{\rightarrow}}) - \boldsymbol{Q_i}(\boldsymbol{O_t}, \boldsymbol{A_t^{\rightarrow}})]$$

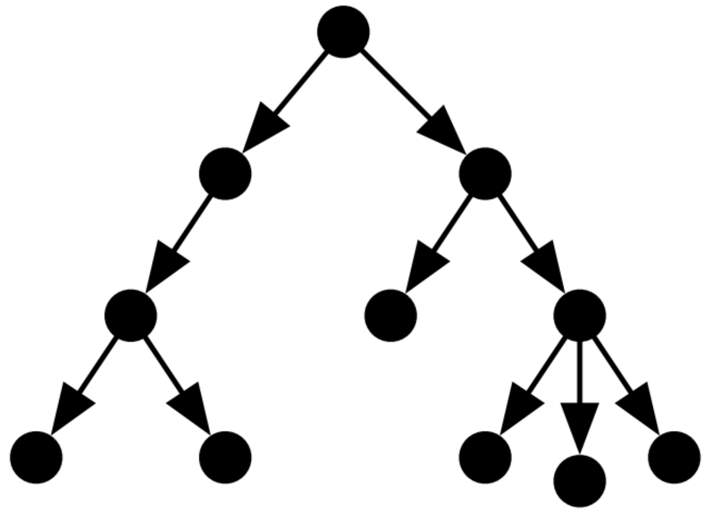
Agent communication

- Communication is practised by many species in nature, and has also been demonstrated to be beneficial in multiagent systems.
- Communication is considered to be the most common way of interaction between intelligent agents.
- An agent communicates or **shares its local information (observations, actions, rewards, or combination) with other agents** in the system.
- However, communication messages can be costly, noisy, corrupted, dropped or received out of order ➔ **communication should be minimal and simple.**

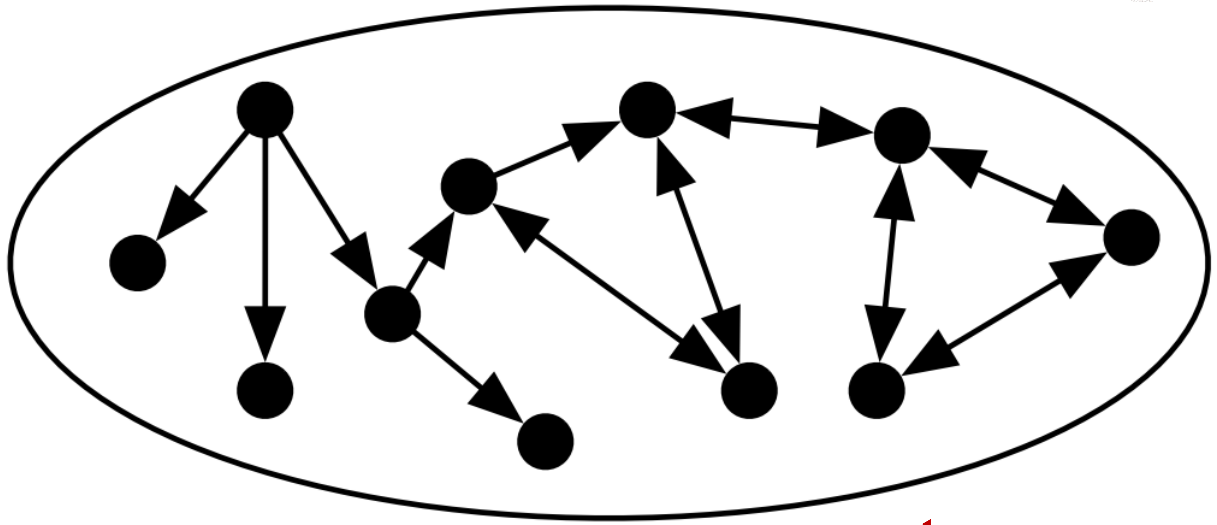
Agent organisation

- The organisation of a multi-agent system is the **collection of roles, relationships, and authority structures** which govern its behavior.
- Just as with human organisations, such agent organisations guide how the members of the population interact with one another.
- The organisational design employed by a multi-agent system can have a significant effect on its performance.

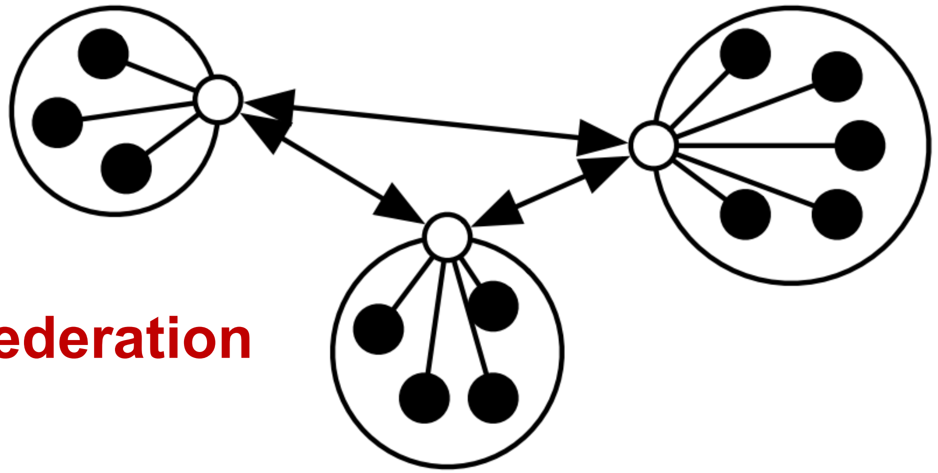
Agent organisation (cont'd)



hierarchy



team



federation

IMITATION LEARNING

Imitation learning

- **Challenges**
  - Determining an appropriate reward function.
  - Rewards can be sparse which makes the learning process.
- **Imitation learning**
  - A reward function is not assumed to be known a priori, but rather it is assumed to be described implicitly through **expert demonstrations**.

Video

Problem formulation

- The primary difference in formulation from the previous RL problem is that we **do not have access to the reward function**.

- Instead we **have access to a set of expert demonstrations** where each demonstration ξ consists of a sequence of state-control pairs:

$$\boldsymbol{\xi = \{(s_0, a_0), (s_1, a_1), \dots\}}$$

which are drawn from the expert policy π^*.

- The **imitation learning problem** is to leverage a set of demonstrations

$$\boldsymbol{\Xi = \{\xi_1, \xi_2, \dots, \xi_D\}}$$

from an expert policy π^* to find a policy $\hat{\pi}^*$ that imitates the expert policy.

Imitation learning methods

- Directly imitate the expert's policy
    - **Behaviour cloning**
    - …
- Indirectly imitate the expert's policy by learning the expert's reward function (Inverse RL)
    - Apprenticeship learning
    - …

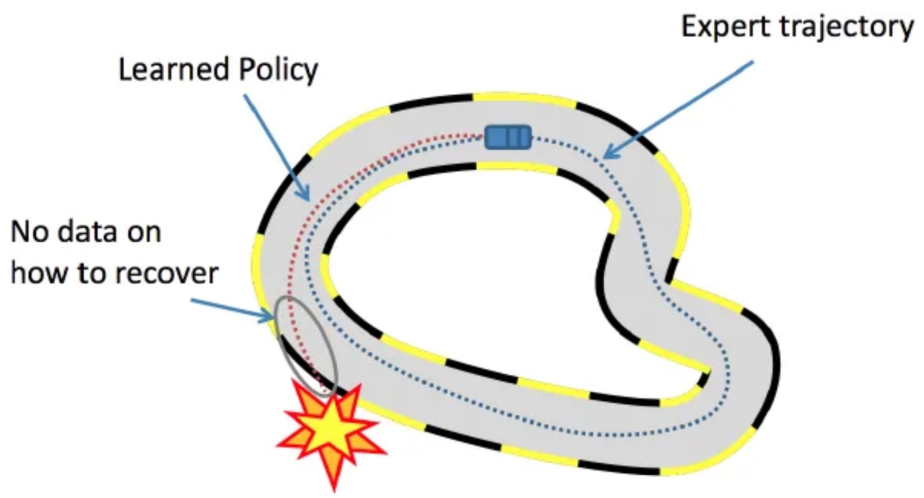
Behaviour cloning

- The difference between the learnt policy and expert demonstrations are minimized with respect to some metric through supervised learning techniques.

$$\hat{\pi} = \operatorname{argmin}_{\pi} \sum_{\xi \in \Xi} \sum_{s \in \xi} L(\pi(s) - \pi^{*}(s))$$

- It works well if the demonstrations are uniformly sampled across the entire state space.

- A mistake made by the agent can easily put it into a state that the expert has never visited and the agent has never trained on.


Expert trajectory
Learned Policy
No data on how to recover