An Overview of MultiAgent Systems & Reinforcement Learning

Alberto Guffanti
Dipartimento di Informatica - Università degli Studi di Torino

NNPDF Collaboration & N3PDF Meeting
Varenna, 27-30.07.2019

Five trends in History of Computing

- **Ubiquity**
  - **Cost reduction** allows us to introduce computing capabilities before was uneconomic.
- **Interconnection**
  - Computer systems are no more stand-alone but networked into **large distributed systems**.
- **Delegation**
  - Computers are performing **more and more tasks on our behalf**, even critical ones.
- **Intelligence**
  - **Growth in complexity** of problems that we are able to automate/delegate to computers.
- **Human-orientation**
  - Programming computer systems has evolved towards higher-level (more **human-oriented**) abstractions.

Agents - Definition

An (intelligent) **agent** is a computer system that is

- capable of **independent** (autonomous) **action** ,
- *on behalf of its user or owner*,
- in a given environment.

This involves figuring out what needs to be done in order to satisfy its own design objectives, rather than being constantly told.

MultiAgent Systems - Definition

"A **multiagent system** is a system that consists of a number of intelligent agents that interact with one-another."

- In the most general case, agents will be acting on behalf of users with different goals and motivations.
- To successfully interact they will require the ability to **cooperate**, **coordinate** and **negotiate** with each other, similar to what people do.

MultiAgent Systems - Micro and Macro Problems

Agent Design (Micro)
How do we build agents that are capable of independent, autonomous action in order to successfully carry out the tasks that we delegate to them?

Society Design (Macro)
How do we build agents that are capable of interacting (cooperate, coordinate and negotiate) with other agents in order to successfully carry out the tasks that we delegate to them? In particular, when other agents cannot be assumed to share the same interests/goals.

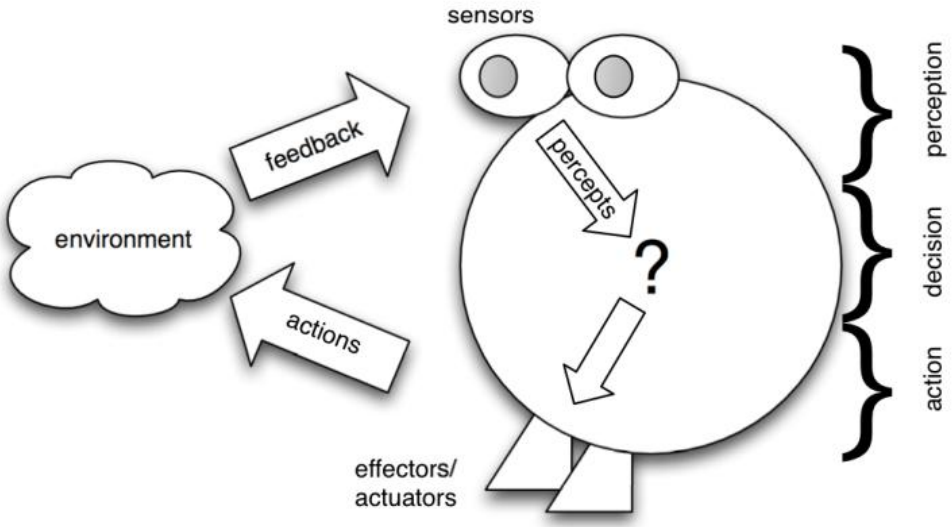
MultiAgent Systems - Society Design

Vast topic, that I will willfully ignore in the rest of the presentation it deals with problems of

- **Communication** between agents created by different owners
  - Definition of ontologies
  - Formal communication languages
- **Coordination** between agents that might have different goals
  - Distributed consensus algorithms
  - Cooperative vs. Non-cooperative Game Theory
  - Managing of Scarce Resources (Negotiation, Distributed Voting)

What is an Agent?

"An **agent** is a computer system **capable autonomous action in some environment**, in order to **achieve its delegated goals**."



Trivial Agents (uninteresting)

- **Thermostat**
  - Delegated goal: maintain a temperature in a room
  - Actions: heating ON/OFF
- **Unix biff daemon**
  - Delegated goal: monitor incoming email and flag it
  - Actions: GUI notifications

Trivial because the **decision making** they do is trivial

Intelligent Agents

Intelligent Agents exhibit three types of behaviour:

- **Reactive**
  - Maintain a constant interaction with the environment and reacts, in a timely fashion, to changes in the latter
- **Proactive**
  - Generate and attempt to achieve goals, not be driven solely by events, but by recongnising opportunities and taking initiative
- **Social**
  - Have the ability to interact with other agents via **cooperation**, **coordination** and **negotiation**

Intelligent Agents - Social Ability

- **Cooperation**
  - Working together to achieve a shared goal
  - Prompted by the fact that no agent can perform the whole task alone or that by cooperation a better result can be achieved
- **Coordination**
  - Managing the interdependencies between activities
  - For example, if there is a sharable resource that multiple agents want to use
- **Negotiation**
  - Ability to reach agreement on matters of common interest
  - Typically involves offers and counter-offers with compromises between participants

Intelligent Agents - Other properties (optional)

- **Mobility**
  - Ability of an agent to move around in a physical/digital environment
- **Veracity**
  - An agent will not wilfully communicate false information
- **Benevolence**
  - Agents do not have conflicting goals, and any agent will try to do what is asked for
- **Rationality**
  - An agent will always act in order to fulfill its goals
- **Capability of Learning/Adapting**
  - An agent improves/changes its performance over time

Properties of the Environment

- **Accessible vs. Inaccessible**
  - An environment is **accessible** if the agent can obtain complete, accurate and up-to-date information about the environment state
- **Deterministic vs. Non-deterministic**
  - An environment is **deterministic** if any action has a single guaranteed effect, *i.e.* there is no uncertainty about the state that will result from performing an action
- **Episodic vs. Non-episodic**
- **Static vs. Dynamic**
  - An environment is **static** if it can be assumed to remain unchanged except by the performance of action by the agent
- **Discrete vs. Continuous**
  - An environment is **discrete** if there is a finite number of actions and percepts in it

Intelligent Agents - Goal

We ultimately would like to design an **agent** that performs **complex tasks** and takes **autonomous action** to fulfill its **design goals**, in an **environment** that is: **partly inaccessible**, **non-deterministic**, **non-episodic**, **dynamic** and **continuous** (*i.e.* our World).

Machine Learning Paradigms

- **Supervised Learning**
  - **Dataset**: collection of labelled examples $\{(x_i, y_i)\}_{i=1..N}$
  - **Goal**: use the *dataset* to create a **model** that takes a feature vector as input and returns information that allows to deduce the corresponding label
- **Unsupervised Learning**
  - **Dataset**: collection of unlabelled examples $\{x_i\}_{i=1..N}$
  - **Goal**: create a model that takes a feature vector as input and returns another vector or a number that is the solution to a practical problem
    - **Clustering algorithm**: returns the id of the cluster the example belongs to
    - **Dimensional reduction**: returns a vector with fewer features
    - **Outlier detection**: returns a number that indicates how much *x* differs from a typical example

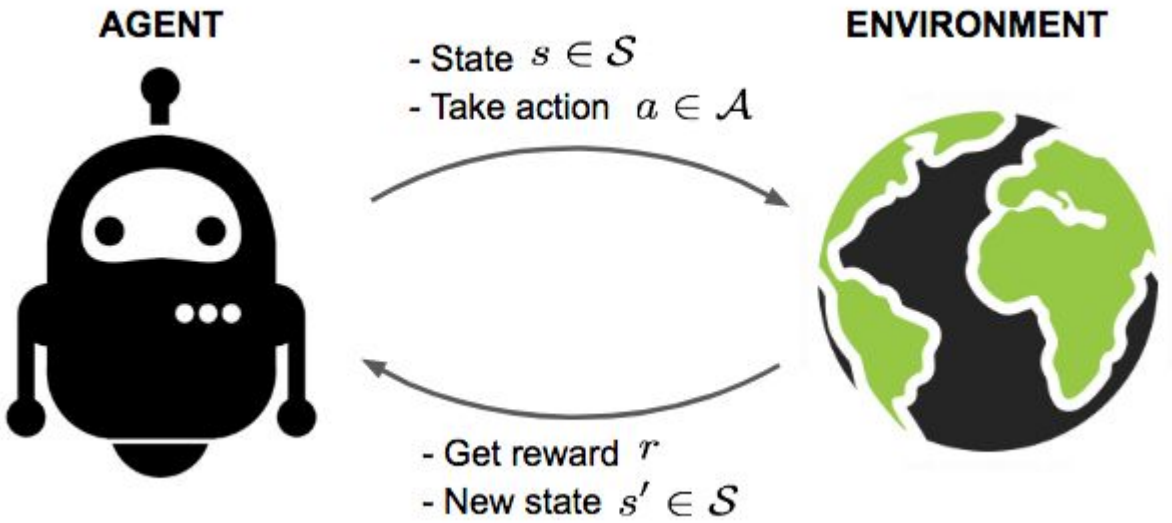
Machine Learning Paradigms

- **Reinforcement Learning**
  - The machine lives in an **environment** and **perceives the state** of the environment as a vector of features
  - The machine can **execute actions** in every state, with **different actions** bringing **different rewards**
  - **Goal**: learn a **policy**, *i.e.* a function that maps a features vector of a state to an **optimal** action to be taken in that stage
  - An action is **optimal** if it maximizes the **expected average reward**
  - **Reinforcement learning** solves a particular problem where **decision making** is **sequential** and the **goal** is **long-term** (*i.e.* game playing, robotics, resource management, ...)

Key Elements of a Reinforcement Learning system

- **Policy**
  - Defines the **agent's way of behaving** at a given time
  - **Mapping** between **perceived states** and **actions** to be taken (might be stochastic)
- **Reward Signal**
  - **Sent from the environment** to the agent after an action has been performed
  - The **agent's objective** is to **maximize the total reward** over the long run
- **Value Function**
  - **Value of a state**: total amount of **reward an agent can expect** to accumulate starting from it
  - **Values** indicate the **long term desirability** of a state given the states that will likely follow
- **Model of the environment** (optional)

Key Elements of a Reinforcement Learning system



Exploration vs. Exploitation Dilemma

- When an agent faces an unknown environment this is key to finding a good solution
- Without enough **exploration** we cannot learn the environment well enough
- Without enough **exploitation** we cannot maximize our reward in the long run
- On common solution is to adopt an **ε-greedy** algorithm, which takes the best action most of the time but does random exploration occasionally

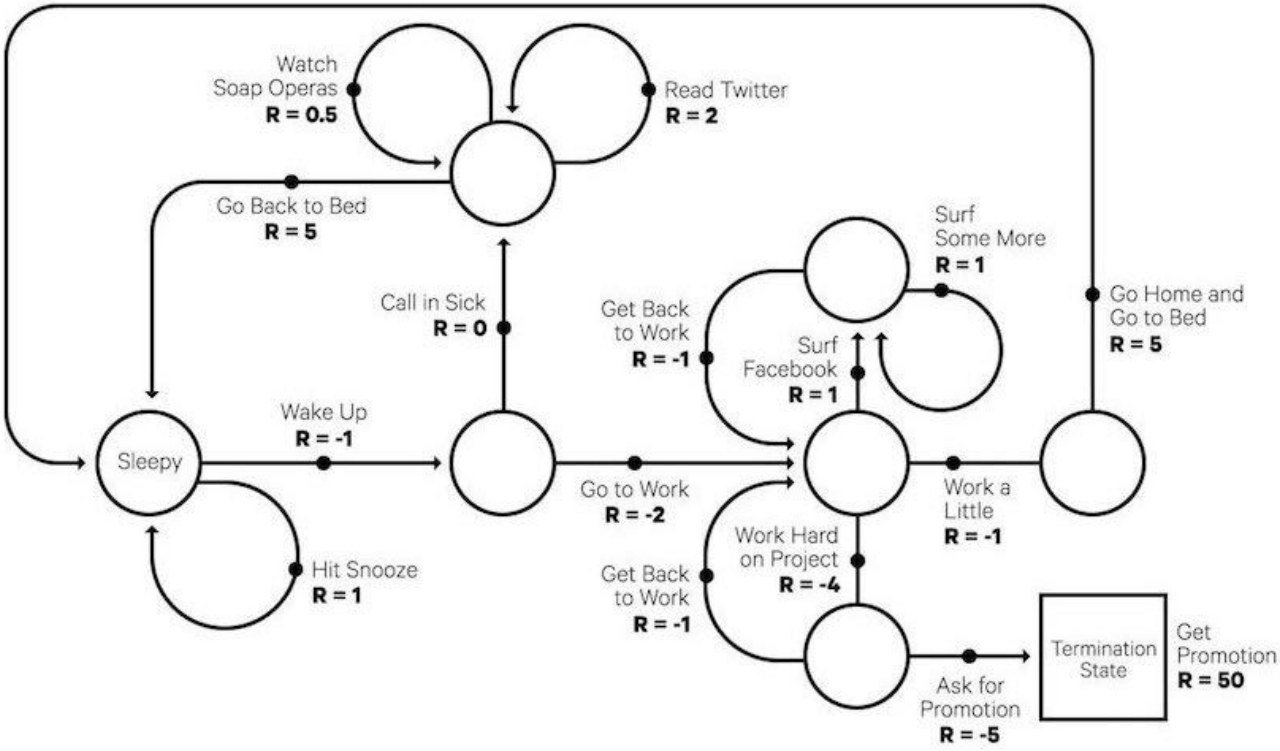
Reinforcement Learning system as MDP

- **Reinforcement Learning** problems can be framed in terms of **Markov Decision Processes** (MDP)
  - … the future only depends on the current state and not the history

$$\mathbb{P}[S_{t+1}|S_t] = \mathbb{P}[S_{t+1}|S_1, \ldots, S_t]$$

- A Markov Decision Process has 5 elements
  - A set of States (S)
  - A set of Actions (A)
  - A transition probability function (P)
  - A reward function (R)
  - A discounting factor for future rewards (γ)

Markov Decision Problem - A Typical Work Day



Reinforcement Learning - Common Approaches

- **Dynamic Programming**
  - When the model is fully known, following Bellman equations, we can use Dynamic Programming (DP) to iteratively evaluate value functions and improve policy.
- **Monte Carlo Methods**
  - Learns from episodes of raw experience without modeling the environmental dynamics and computes the observed mean return as an approximation of the expected return.
  - To compute the empirical return MC methods need to learn from **complete** episodes to compute, and all the episodes must eventually terminate.
- **Temporal Difference Learning**
  - Temporal-Difference (TD) Learning is model-free and learns from episodes of experience.
  - TD learning methods update targets with regard to existing estimates rather than exclusively relying on actual rewards and complete returns as in MC methods (**bootstrapping)**.

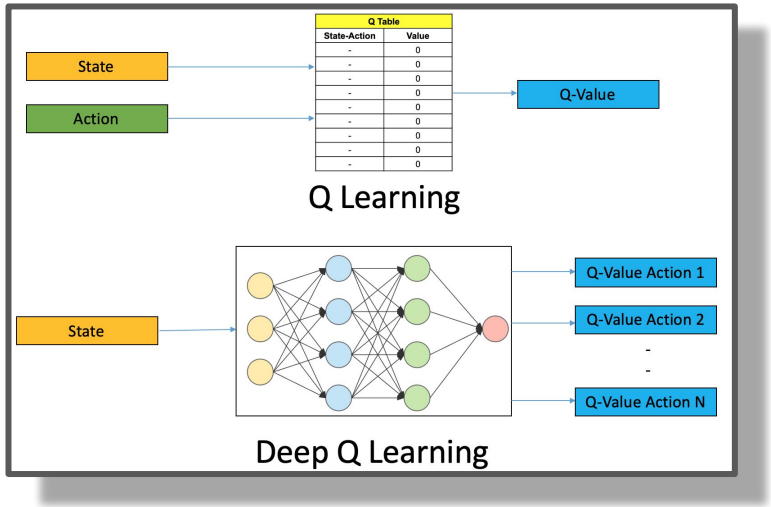
Reinforcement Learning - Q-Learning

- If we knew the **expected reward of each action at every step**, this would be a **cheat sheet** for the agent, which would know exactly which action to perform.
- The agent will perform a sequence of actions that will eventually generate the maximum total reward (**Q-value**), and we formalise our strategy as

1. At time step t, we start from state S_t and pick action according to Q values, $A_t = \arg\max_{a \in \mathcal{A}} Q(S_t, a)$; ε-greedy is commonly applied.
2. With action A_t, we observe reward R_{t+1} and get into the next state S_{t+1}.
3. Update the action-value function: $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \max_{a \in \mathcal{A}} Q(S_{t+1}, a) - Q(S_t, A_t))$.
4. t = t+1 and repeat from step 1.

Reinforcement Learning: from Q-Learning to DQN

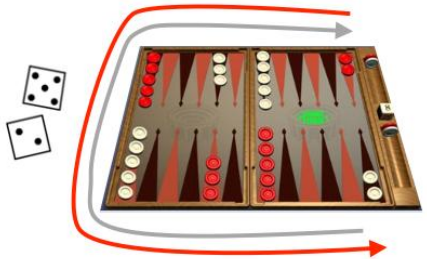
- **Q-learning** is a **simple yet powerful algorithm** to create a cheat sheet to help an agent to figure out which action to perform in a given state.
- Things quickly get out of control with the number of states and actions (an environment with 10k states and 1k actions per state, would create a table of 10 million cells).
- In **Deep Q-Network** (DQN) one replaces the **Q-values table** with a **Deep Neural Network**, with all its advantages and disadvantages

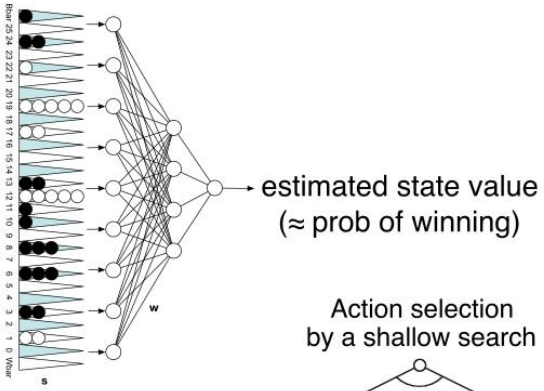


Reinforcement Learning - Examples

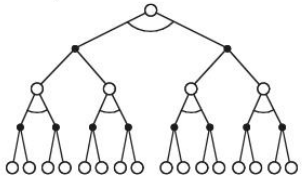
Example: TD-Gammon

Tesauro, 1992-1995





Action selection
by a shallow search



Start with a random Network

Play millions of games against itself

Learn a value function from this simulated experience

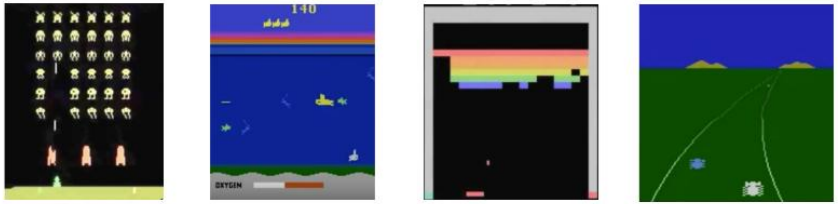
Six weeks later it's the best player of backgammon in the world

Originally used expert handcrafted features, later repeated with raw board positions

Deep Reinforcement Learning - Examples

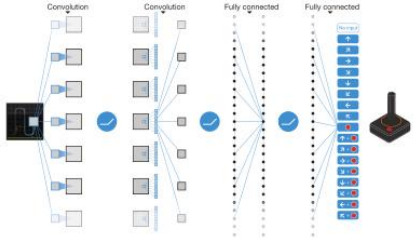
RL + Deep Learning, applied to Classic Atari Games

Google Deepmind 2015, Bowling et al. 2012



- Learned to play 49 games for the Atari 2600 game console, without labels or human input, from self-play and the score alone

mapping raw screen pixels



to predictions of final score for each of 18 joystick actions

- Learned to play better than all previous algorithms and at human level for more than half the games

Same learning algorithm applied to all 49 games! w/o human tuning

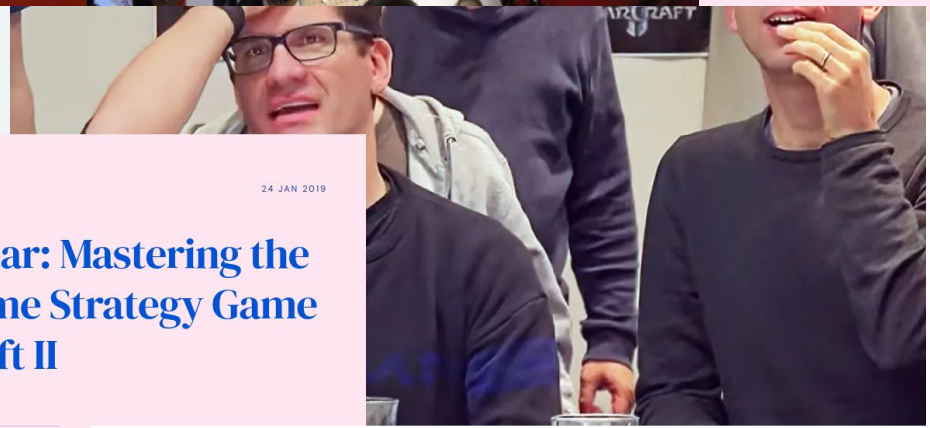
Deep Reinforcement Learning - Examples

AlphaGo: using machine learning to master the ancient game of Go

BLOG POST
RESEARCH

06 DEC 2018

AlphaZero: Shedding new light on chess, shogi, and Go



BLOG POST
RESEARCH

24 JAN 2019

AlphaStar: Mastering the Real-Time Strategy Game StarCraft II

In late 2017 we introduced AlphaZero, a single system that taught itself from scratch how to master the games of chess, shogi (Japanese chess), and Go, beating a world-champion program in each case. We were excited by the preliminary results and thrilled to see the response from members of the chess community, who saw in AlphaZero's games a ground-breaking, highly dynamic and "unconventional" style of play that differed from any chess playing engine that came before it.

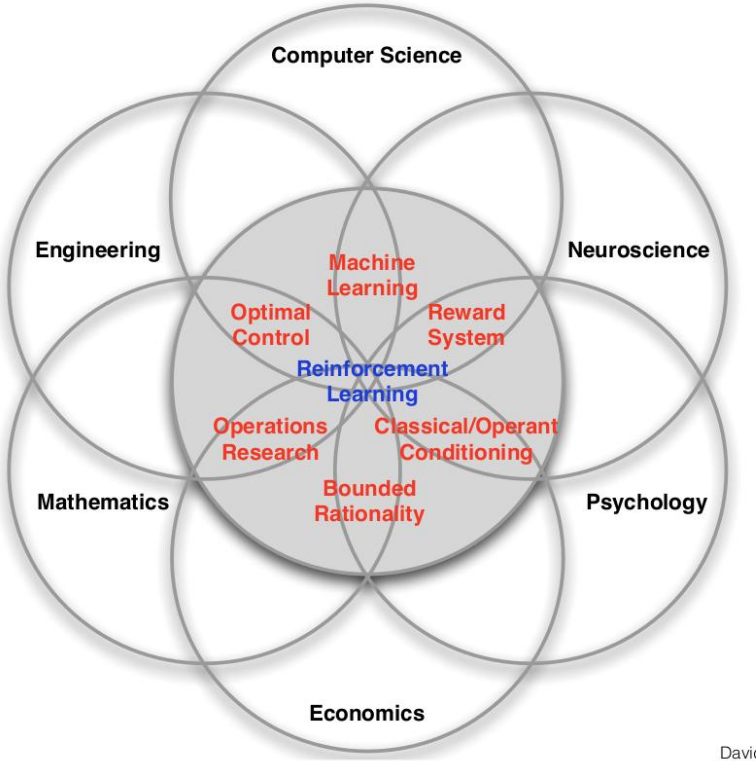
Critical appraisal of Deep Reinforcement Learning

- Deep Reinforcement Learning is the basis of the most striking results presented before but …
- ... it turns out to be a very specific technique and **DRL** trained **agents do not generalize** well, with **small changes** leading to **drastic drops in performance**
  - … moving the paddle up by a few pixels in *Breakout*
  - … changing map and “race” of the characters in *Starcraft*
- **DQN** seems to be a sort of **“super-memorization”**: systems that use it are capable of **amazing feats** but have a **shallow understanding of the context**
- **DQN** requires **huge amount of data** (*i.e.* millions of self-played games of Go), much more than a human would require for a similar task

Conclusions

- MultiAgent Systems are a natural development of trends that have ever been present in the history of computing (ubiquity, delegation, networking...)
- They are an interesting and multi-disciplinary field (distributed systems, design of intelligent agents, simulations of social interaction, game theory)
- Reinforcement Learning is one of the main techniques used in developing intelligent agents that act in complex environments
- Combining Reinforcement Learning (Q-learning) with Deep Neural Networks led to impressive results

Reinforcement Learning



David Silver 2015