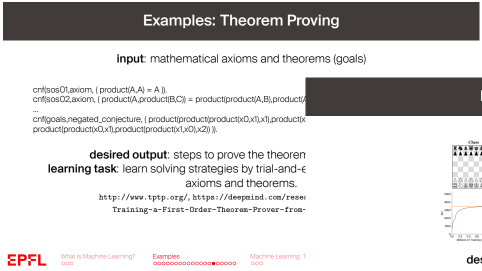
Reinforcement Learning

Johanni Brea

Introduction to Machine Learning

Examples of Reinforcement Learning



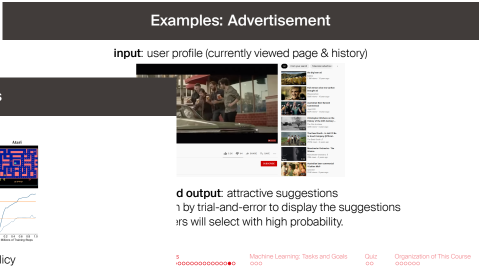
Examples: Theorem Proving

input: mathematical axioms and theorems (goals)

```
cnf(sos01,axiom, ( product(A,A) = A )).
cnf(sos02,axiom, ( product(A,product(B,C)) = product(product(A,B),product(
...
cnf(goals,negated_conjecture, ( product(product(product(x0,x1),x1),product(x
product(product(x0,x1),product(product(x1,x0),x2)) )).
```

desired output: steps to prove the theorem
learning task: learn solving strategies by trial-and-e
axioms and theorems.

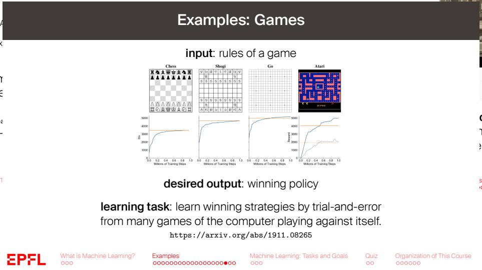
`http://www.tptp.org/`, `https://deepmind.com/resea`
`Training-a-First-Order-Theorem-Prover-from-`



Examples: Advertisement

input: user profile (currently viewed page & history)

d output: attractive suggestions
ı by trial-and-error to display the suggestions
ɪrs will select with high probability.



Examples: Games

input: rules of a game

Chess
Shogi
Go
Atari
Elo
Reward
Millions of Training Steps

desired output: winning policy

learning task: learn winning strategies by trial-and-error from many games of the computer playing against itself.
`https://arxiv.org/abs/1911.08265`

Learning by Trial-and-Error

Table of Contents

1. A Toy Problem: Chasse au Trésor

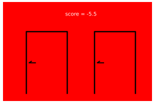
2. Q-Learning

3. Function Approximation and Deep Reinforcement Learning

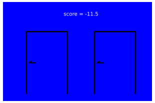
4. Reinforcement Learning and the Brain

5. Reinforcement Learning for Efficient Planning

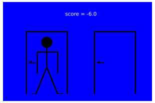
A Toy Problem: Chasse au Trésor



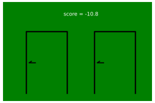
state 1



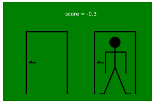
state 2



state 3



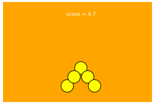
state 4



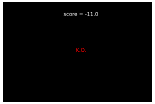
state 5

actions: (in each room)
open left door,
open right door

rewards: (depend on
state and action)
between -5 and 6

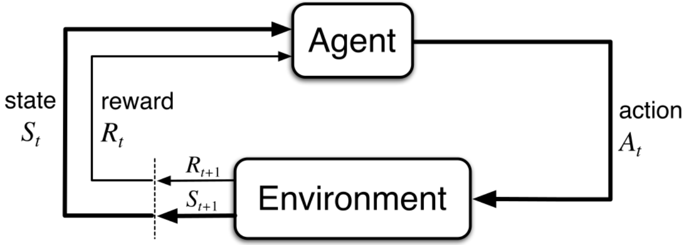


state 6



state 7

Agents and Environments: States, Actions, Rewards



The **agent** (a person, an animal, a computer program) observes at time t state S_t, reward R_t (for previous actions) and takes action A_t.

The **environment** (a game, a dynamical system, "the world") receives A_t and produces next state S_{t+1} and reward R_{t+1}.

- The dynamics of the environment can be stochastic, e.g. given by **transition probabilities** $P(S_{t+1}|S_t, A_t)$ and **reward probabilities** $P(R_{t+1}|S_t, A_t, S_{t+1})$.
- Usually, the agent starts with zero knowledge about the transition and reward probabilities.
- The agents goal is to maximize the expected cumulative reward.

Action Values

Action Values $Q(S_t, A_t)$ indicate the desirability of action A_t in state S_t by measuring the expected cumulative reward. They are also called **Q Values**.

Finite Horizon until T (episodic setting)

$$Q^{\text{exact}}(S_t, A_t) = \mathrm{E}\left[R_{t+1} + R_{t+2} + \cdots + R_T\right]$$

Infinite Horizon with Discount Factor $\gamma \in [0, 1)$ (continual setting)

$$Q^{\text{exact}}(S_t, A_t) = \mathrm{E}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots\right]$$

The expectation depends on the policy (action selection strategy) and the transition and reward probabilities. Because they are usually unknown to the RL agent, the agent cannot exactly compute these expectation but has to estimate them.

From now on we will use the symbol Q to denote an RL agent's estimate of Q^{exact}.

Learning Action Values with Monte Carlo Estimation

Compute the cumulative reward for every state-action visited in each episode (length T). Average the result over all episodes where the same state-action pair was visited.

```
1:  $Q(s, a)$: arbitrarily initialized
2:  CumulativeRewards($s, a$): an empty list
3:  for all episodes do
4:      for $t = 1, \ldots, T - 1$ do
5:          $G = \sum_{i=t+1}^{T} R_i$
6:          append $G$ to CumulativeRewards($S_t, A_t$)
7:          $Q(S_t, A_t)$ = average(CumulativeRewards($S_t, A_t$))
8:      end for
9:  end for
10: return $Q$
```

- The estimated Q-values depend on the actions we take!
- This algorithm can be optimized by using a recursive updates (see notebook).

Policies: Exploration and Exploitation

- A **policy** is a mapping from perceived states s to actions a to be taken when in those states.
- A policy can be a **deterministic** function $a = \pi(s)$.
- In general a policy is **stochastic** and described by conditional probabilities $\pi = P(a|s)$.
- The policy serves two goals:
  1. **Exploitation**: Choose the action that is currently assumed to be the one that maximizes the cumulative return.
  2. **Exploration**: Choose exploratory actions to learn more about the environment and know better which action is actually best.

  This is also called the **Exploration-Exploitation Dilemma** (or Trade-Off)

ϵ-Greedy Policies

- **ϵ-Greedy Policies** choose with probability $1-\epsilon$ the (supposedly) best action for the current state (according to e.g. the current estimate of the Q-values) and with probability ϵ a random action.
- The best action is also called **greedy**; the random action **exploratory**.
- With $\epsilon = 1$ the agent only explores.
- With $\epsilon = 0$ the agent only exploits. This is the **greedy policy**.
- ϵ is a critical hyper-parameter affecting speed of learning! Common choices for simple problems are $\epsilon = 0.1$ at the beginning of learning.
- Over the course of learning, the agent gets more and more certain about the best actions; therefore one can gradually decrease ϵ, i.e. exploit more.

Summary

Key Ingredients of Reinforcement Learning

- An **agent** performs **actions** A_t according to some **policy** in an **environment** and perceives **states** S_t and **rewards** R_t.
- The agent should choose a policy that trades off **exploitation** (acquire as much reward as possible) and **exploration** (learn more about potentially more rewarding parts of the environment).
- For exploitation the agent can rely on estimated **action values** $Q(s, a)$ that indicate the **expected cumulative reward** of action a in state s. The action values can be estimated with **Monte Carlo Estimation** (among many other methods not yet discussed).
- For exploitation the agent can occasionally take a random action. This is formalized in **epsilon-greedy** policies (among other exploration strategies not yet discussed).

Supervised vs. Unsupervised vs. Reinforcement Learning

	supervised	**unsupervised**	**reinforcement**
given	X, Y	X	an environment (the agent collects data)
goal	find $P(Y\|X)$	find structure in data	find optimal policy
evaluation	test error	?	cumulative reward
approach	fit training data	use training data	interact with environment

Table of Contents

1. A Toy Problem: Chasse au Trésor

2. Q-Learning

3. Function Approximation and Deep Reinforcement Learning

4. Reinforcement Learning and the Brain

5. Reinforcement Learning for Efficient Planning

Q-Learning

Remember that action values are defined as

$$Q_{\pi}^{\text{exact}}(S_t, A_t) = \mathrm{E}\left[R_{t+1} + R_{t+2} + \cdots + R_T\right]$$

The subscript π indicates that $Q_{\pi}^{\text{exact}}(S_t, A_t)$ depends on policy π used for future actions.

The equation above can also be written in recursive form as

$$Q_{\pi}^{\text{exact}}(S_t, A_t) = \mathrm{E}\left[R_{t+1} + Q_{\pi}^{\text{exact}}(S_{t+1}, A_{t+1})\right]$$

and for the optimal (greedy) policy π^* that takes in every state the action with maximal value

$$Q_{\pi^*}^{\text{exact}}(S_t, A_t) = \mathrm{E}\left[R_{t+1} + \max_a Q_{\pi^*}^{\text{exact}}(S_{t+1}, a)\right]$$

Therefore

$$\mathrm{E}\left[R_{t+1} + \max_a Q_{\pi^*}^{\text{exact}}(S_{t+1}, a) - Q_{\pi^*}^{\text{exact}}(S_t, A_t)\right] = 0$$

Q-Learning

$$\mathrm{E}\left[R_{t+1} + \max_a Q^{\text{exact}}_{\pi^*}(S_{t+1}, a) - Q^{\text{exact}}_{\pi^*}(S_t, A_t)\right] = 0$$

Question: Is there a way to start with an arbitrary $Q_0(S_t, A_t)$ and define an update rule $Q_k(S_t, A_t) \rightarrow Q_{k+1}(S_t, A_t)$ that depends on every observed transition $S_t, A_t \rightarrow R_{t+1}, S_{t+1}$ such that $\lim_{k\rightarrow\infty} Q_k(S_t, A_t) = Q^{\text{exact}}_{\pi^*}(S_t, A_t)$?

Idea: The update rule should be such that the action value moves always in direction of the **Temporal Difference Error (TD-Error)**

$$\delta_k = R_{t+1} + \max_a Q_k(S_{t+1}, a) - Q_k(S_t, A_t)$$

i.e.

$$Q_{k+1}(S_t, A_t) = Q_k(S_t, A_t) + \lambda\delta_k$$

where λ is a learning rate, e.g. $\lambda = 0.1$.

Properties of Q-Learning

At convergence:

$$0 = \mathrm{E}[\delta_k] = \mathrm{E}\left[R_{t+1} + \max_a Q_k(S_{t+1}, a) - Q_k(S_t, A_t)\right]$$

i.e. Q-Learning stops when the action values Q_k satisfy the same equation as $Q_{\pi^*}^{\text{exact}}$ of the optimal greedy policy π^*.

Q-Learning is an **off-policy** method, because it estimates the Q-values for the greedy policy no matter what exploration policy is used; the Q-values of e.g. Monte Carlo Estimation depend on the exploration policy (Monte Carlo Estimation is an **on-policy** method).

Table of Contents

1. A Toy Problem: Chasse au Trésor

2. Q-Learning

3. Function Approximation and Deep Reinforcement Learning

4. Reinforcement Learning and the Brain

5. Reinforcement Learning for Efficient Planning

Different State Representations

Enumerating all states is often not a good idea

- There can be a lot of states, e.g. chess has more than 10^{40} different positions.
- It is unclear how to generalize with enumerated states, e.g. we may learn quickly that it is not a good idea to open doors with a guard in front of it in our chase au trésor, but looking at the raw integers, state 3 is as different from state 5 as it is from state 4.

Alternatives: pixel images as input, or feature representation as input.
E.g. (in red room, in blue room, in green room, in treasure room, KO, guard present)

Function Approximation

- If the state s is a vector (instead of an integer) it does not make much sense to store the Q-values in a table.
- Instead, we can use a parametrized function family $q_\theta(s)$ to compute the Q-values.
- For a fixed θ the function q_θ takes the vector-valued state s as input and returns a vector of Q-values for each action; $q_\theta(s)_a$ is the Q-value of state s and action a.
- The parameters θ could be the weights of neural network.
- The correct Q-values can be learned by adjusting the parameters θ.

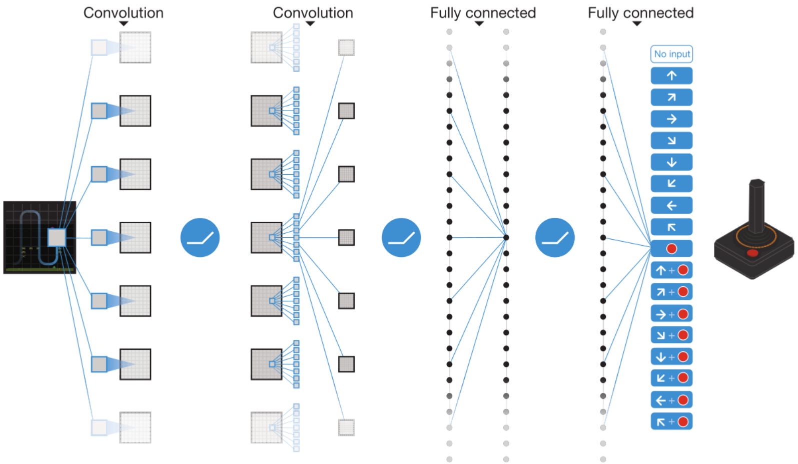
Deep Q Network

Main Idea: For observed transition S_t, A_t, S_{t+1}, adapt the parameters θ with gradient descent on the squared TD-Error loss function

$$\theta^{(k+1)} = \theta^{(k)} - \lambda \frac{\partial}{\partial \theta} \left(R_{t+1} + \max_a q_{\theta^{(k)}}(S_{t+1})_a - q_\theta(S_t)_{A_t} \right)^2$$

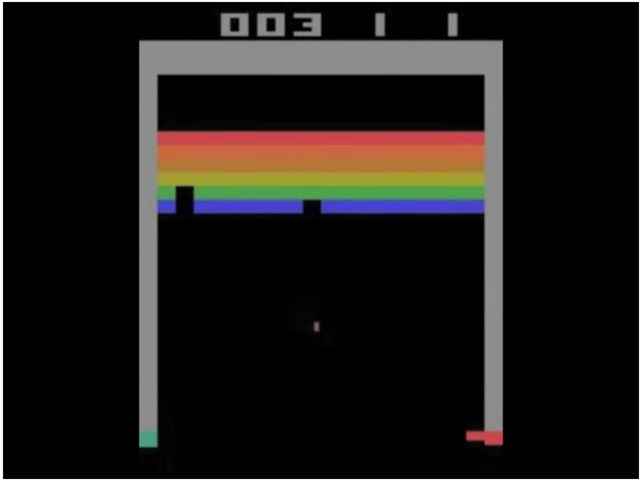
In practice many additional tricks are used...

Deep Q Network Learns to Play Atari Game



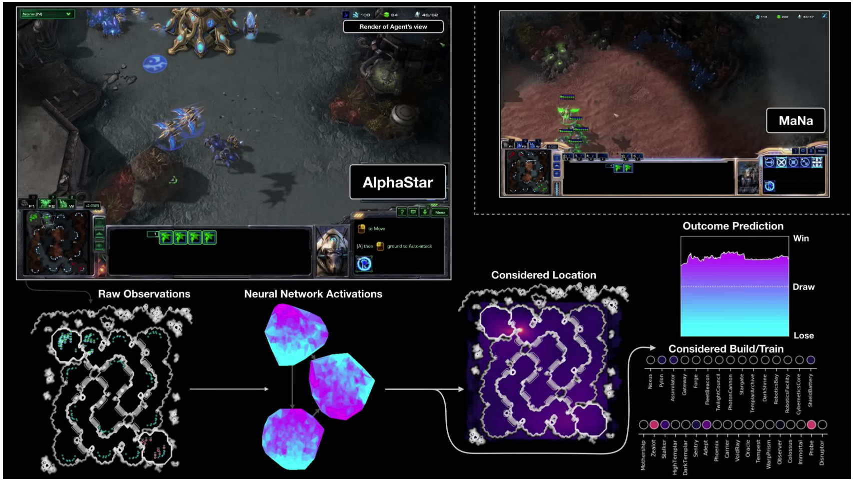
https://www.deepmind.com/publications/human-level-control-through-deep-reinforcement-learning

Deep Q Network Learns to Play Atari Game



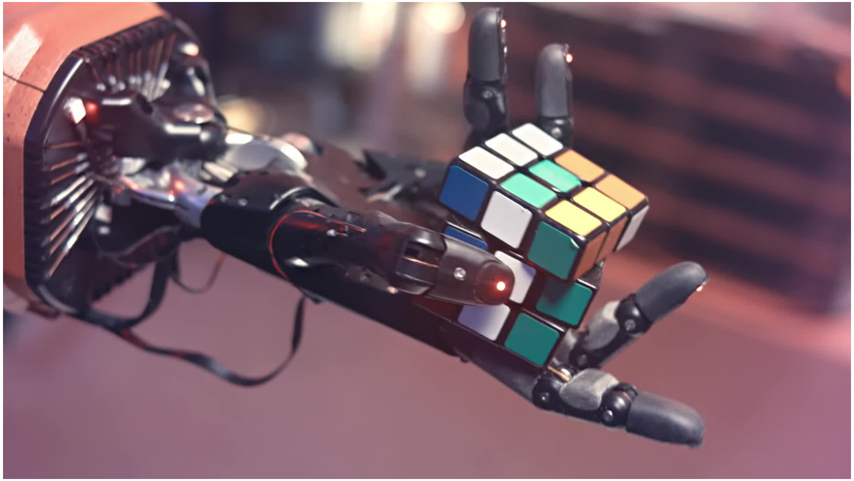
https://www.deepmind.com/publications/human-level-control-through-deep-reinforcement-learning

AlphaStar: Mastering StarCraft II



https://www.deepmind.com/blog/alphastar-mastering-the-real-time-strategy-game-starcraft-ii

Solving Rubik's Cube with a Robot Hand



https://openai.com/blog/solving-rubiks-cube/

Table of Contents

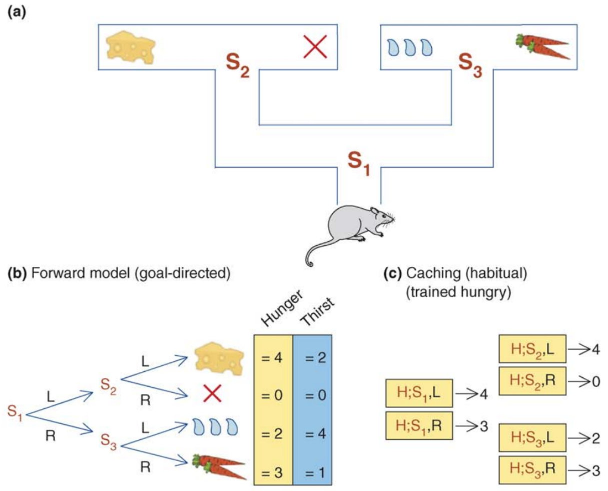
1. A Toy Problem: Chasse au Trésor
2. Q-Learning
3. Function Approximation and Deep Reinforcement Learning
4. **Reinforcement Learning and the Brain**
5. Reinforcement Learning for Efficient Planning

Does Dopamine Signal Reward Prediction Errors?

“When monkeys receive unexpected reward, dopamine neurons fire a burst of action potentials. If the monkeys learn to expect reward, that same reward no longer triggers a dopamine response. Finally, if an expected reward is omitted, dopamine neurons pause their firing at the exact moment reward is expected.”

`https://doi.org/10.1146/annurev-neuro-072116-031109`

Goal-Directed versus Habitual Behaviour



http://dx.doi.org/10.1016/j.tics.2006.06.010

It is hypothesized that animals and humans rely on a model-based reinforcement learning system that allows goal-directed planning and a model-free reinforcement learning system that forms habits.

The model-free system is fast and computationally cheap. Planning with the model-based system is slower and computationally more expensive but it allows to "try out things in the mind" and therefore propose creative solutions with potentially less trial-and-error.

Table of Contents

1. A Toy Problem: Chasse au Trésor
2. Q-Learning
3. Function Approximation and Deep Reinforcement Learning
4. Reinforcement Learning and the Brain
5. **Reinforcement Learning for Efficient Planning**

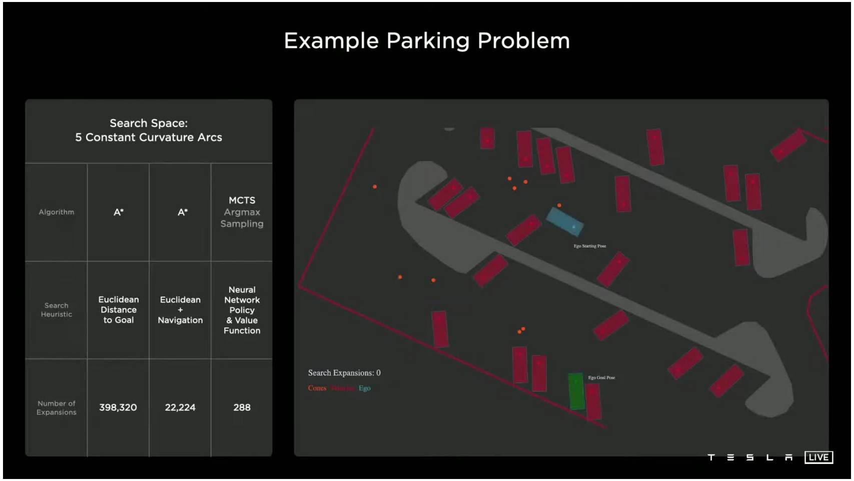
Reinforcement Learning for Efficient Planning

- Tic-Tac-Toe could also be solved with exhaustive search (planning), e.g. with policy iteration, because the transitions and the rewards are known.
- Many problems have too many states to be fully solved by exhaustive planning, e.g. Chess or Go (approximately 10^{170} states).
- Also model-free reinforcement learning would be inefficient in these cases, because it takes very long to learn an accurate policy for so many states.
- Instead one can use an inaccurate policy to guide local planning, and focus on the most promising actions starting from the current position.

AlphaZero: Mastering Two-Player Games

https://www.deepmind.com/research/highlighted-research/alphago

Application: Planning to Avoid Obstacles with a Tesla Car



Example Parking Problem

Search Space:
5 Constant Curvature Arcs

Algorithm	A*	A*	MCTS Argmax Sampling
Search Heuristic	Euclidean Distance to Goal	Euclidean + Navigation	Neural Network Policy & Value Function
Number of Expansions	398,320	22,224	288

https://youtu.be/j0z4FweCy4M

Summary

- Reinforcement Learning agents learn by **trial-and-error**.
- **Exploration** is achieved by taking actions that differ from the currently assumed best one.
- Even without learning an explicit transition and reward model, agents can learn to adapt to anticipated future needs (**model-free RL**), e.g. with Monte Carlo Estimation or Q-Learning.
- When the state space is too large for tabular RL, function approximation is useful for generalization (**deep RL**).
- When the transition dynamics and the reward structure is known or well estimated (**model-based RL**), planning (e.g. policy iteration) can be used.
- When the search space is too large for classic planning methods, RL can be used to learn a good **search heuristic**.

Resources

This lecture in inspired by the excellent text book

Reinforcement Learning: An Introduction
Richard S. Sutton and Andrew G. Barto
Second Edition
`http://incompleteideas.net/book/the-book-2nd.html`

You may also like the great online educational resource
`https://spinningup.openai.com`