Reinforcement Learning

Johanni Brea

Introduction to Machine Learning

EPFL BIO322 2021

Examples of Reinforcement Learning

Examples: Theorem Proving

input: mathematical axioms and theorems (goals)

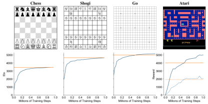
cnf(sos01,axiom, (product(A,A) = A)).
cnf(sos02,axiom, (product(A,product(B,C)) = product(product(A,B),product(
...
cnf(goals,negated_conjecture, (product(product(product(x0,x1),x1),product(x
product(product(x0,x1),product(product(x1,x0),x2)))).

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: Games

input: rules of a game



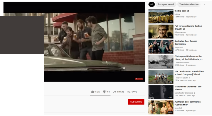
desired output: winning policy

learning task: learn winning strategies by trial-and-error from many games of the computer playing against itself.

`https://arxiv.org/abs/1911.08265`

Examples: Advertisement

input: user profile (currently viewed page & history)

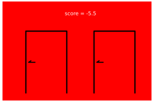


d **output**: attractive suggestions
ı by trial-and-error to display the suggestions
ɪrs will select with high probability.

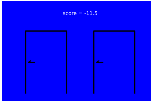
Learning by Trial-and-Error

Table of Contents

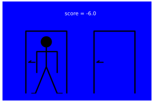
A Toy Problem: Chase au Trésor



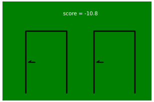
state 1



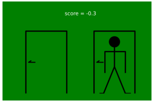
state 2



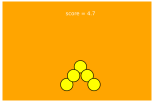
state 3



state 4



state 5



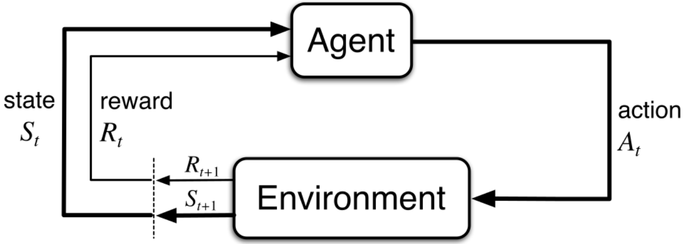
state 6

state 7

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

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

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

Finite Horizon until T (episodic setting)

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

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

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

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.

Learning Action Values with Monte Carlo Estimation

For every state-action pair visited in each episode compute the cumulative reward for that episode. Average the result over all episodes where the same state-action pair was visited.

```
Q(s, a): arbitrarily initialized
CumulativeRewards(s, a): an empty list
for all episodes do
    G ← 0
    for t = T − 1, T − 2, ..., 1 do
        G ← G + R_{t+1}
        append G to CumulativeRewards(S_t, A_t)
        Q(S_t, A_t) = average(CumulativeRewards(S_t, A_t))
    end for
end for
return Q
```

- The estimated Q-values depend on the actions we take!
- In practice we rather use a recursive update of the average.

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 (supposedly) best action to maximize 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.

Quiz

1. Suppose an agent has experienced the two episodes $((S_1 = s_1, A_1 = a_2, R_1 = 0), (S_2 = s_5, A_2 = a_3, R_2 = 2))$ and $((S_1 = s_1, A_1 = a_2, R_1 = 1), (S_2 = s_4, A_2 = a_2, R_2 = 1))$. Action values are learnt with Monte Carlo Estimation. Which of the following Q-values is correct?
   A $Q(s_1, a_2) = 2$ **B** $Q(s_4, a_2) = \frac{1}{2}$ **C** $Q(s_5, a_3) = 2$
2. Suppose in a certain state s an agent can take actions a_1 or a_2. The Q-values are $Q(s, a_1) = 1$, $Q(s, a_2) = 4$ The agent uses an epsilon-greedy policy with $\epsilon = 0.5$. With which probability does the agent take action a_1?
   A 0 **B** 0.25 **C** 0.5 **D** 0.75 **E** 1.0
3. With which probability would a greedy agent take action a_1 in the setting above?
   A 0 **B** 0.25 **C** 0.5 **D** 0.75 **E** 1.0

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).

Table of Contents

Markov Decision Processes (MDP)

A **Markov Decision Process** (MDP) is characterized by

1. a state space $\mathcal{S}$ (of all possible states)
2. an action space $\mathcal{A}$ (of all possible actions)
3. transition probabilities $P(S_{t+1}|S_t, A_t)$
4. reward probabilities $P(R_{t+1}|S_t, A_t)$ (sometimes $P(R_{t+1}|S_t, A_t, S_{t+1})$)

The **solution on an MDP** is a policy π that maximizes the expected cumulative reward, i.e.

$$\pi(s) = \arg\max_a Q(s, a).$$

With known transition and reward probabilities (and sufficient compute power) one can compute the expected cumulative reward $Q(s, a)$ and does not need exploration to estimate $Q(s, a)$.

Bellman Equation

The **Bellman Equations**: a recursive way to write the action values

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

We use the subindex π here to indicate that the action values depend on the policy.

$$\begin{aligned}
Q_\pi(S_t, A_t) &= \mathrm{E}\left[R_{t+1} + R_{t+2} + R_{t+3} + \cdots\right] \\
&= \sum_{S_{t+1}, A_{t+1}} \underbrace{\sum_{R_{t+1}} R_{t+1} P(R_{t+1}|S_t, A_t)}_{\bar{r}(S_t, A_t)} + \pi(A_{t+1}|S_{t+1}) P(S_{t+1}|S_t, A_t) \left(\bar{r}(S_{t+1}, A_{t+1}) + \sum_{S_{t+2}, A_{t+2}} \cdots\right) \\
&= \bar{r}(S_t, A_t) + \sum_{S_{t+1}, A_{t+1}} P(S_{t+1}|S_t, A_t) \pi(A_{t+1}|S_{t+1}) Q_\pi(S_{t+1}, A_{t+1}) \\
&= \bar{r}(S_t, A_t) + \sum_{S_{t+1}} P(S_{t+1}|S_t, A_t) Q_\pi\left(S_{t+1}, \pi(S_{t+1})\right) \qquad \text{for deterministic } \pi
\end{aligned}$$

Iterative Policy Evaluation

Some equations of the form $x = f(x)$ can be solved with a fixed point iteration:

Start with $x^{(0)}$ and compute

$$x^{(k)} = f(x^{(k-1)})$$

until $x^{(k)} \approx x^{(k-1)}$.

Example: Babylonian method for computing the square root of a

$$x = \frac{1}{2}\left(\frac{a}{x} + x\right) = f_a(x)$$

The Bellman Equations can be solved with fixed point iteration:

1. Start with random values for $Q_\pi^{(0)}(s, a)$.
2. Iterate
$$Q_\pi^{(k)}(s, a) = \bar{r}(s, a) + \sum_{s'} P(s'|s, a) Q_\pi^{(k-1)}(s', \pi(s'))$$
until $Q_\pi^{(k)}(s, a) \approx Q_\pi^{(k-1)}(s, a)$ for all s, a.

Policy Iteration

Policy Improvement Theorem

Let π and π' be any pair of deterministic policies such that for all s

$$Q_\pi(s, \pi'(s)) \geq Q_\pi(s, \pi(s)) .$$

Then the policy π' must be as good as or better than π, i.e.

$$Q_{\pi'}(s, \pi'(s)) \geq Q_\pi(s, \pi(s)) .$$

Policy Iteration

1. Start with a random deterministic policy $\pi^{(0)}$.
2. Iterate $\pi^{(k)}(s) = \arg\max_a Q_{\pi^{(k-1)}}(s, a)$ (policy improvement)

   where $Q_{\pi^{(k-1)}}(s, a)$ is computed e.g. with iterative policy evaluation,
   until $\pi^{(k)}(s) = \pi^{(k-1)}(s)$ for all s.

Model-Based versus Model-Free Reinforcement Learning

If the transition and reward probabilities are unknown there are two general strategies:

Model-Based Reinforcement Learning	Model-Free Reinforcement Learning
Transition and reward probabilities (the model of the world) **are estimated** from experience	Transition and reward probabilities are **not** explicitly estimated and stored.
The policy π is found with e.g. Policy Iteration using the estimated model of the world	The policy π improves directly by using the experienced samples, e.g. Monte Carlo Estimation of the Q-values.

Table of Contents

Generalized Policy Iteration

It is not necessary to fully evaluate the policy after each policy improvement; as long as policy evaluation and policy improvement continue to update every state occasionally, incomplete evaluation steps can be interleaved with incomplete improvement steps.

Q-Learning

In Q-Learning, the evaluation step is based on a single, observed transition $S_t, A_t \to S_{t+1}$.
It moves the Q-value only slightly in direction of the **Temporal Difference Error (TD-Error)**

$$\delta^{(k)} = R_{t+1} + \max_A Q_\pi^{(k)}(S_{t+1}, A) - Q_\pi^{(k)}(S_t, A_t)$$

The actual update step uses the TD-error:

$$Q_\pi^{(k+1)}(S_t, A_t) = Q_\pi^{(k)}(S_t, A_t) + \lambda\delta^{(k)}$$

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

Properties of Q-Learning

Remember, the Q-values satisfy the Bellman equations:

$$Q_\pi(S_t, A_t) = \mathrm{E}\left[R_{t+1} + Q_\pi(S_{t+1}, \pi(S_{t+1}))\right] \Rightarrow 0 = \mathrm{E}\left[R_{t+1} + Q_\pi(S_{t+1}, \pi(S_{t+1})) - Q_\pi(S_t, A_t)\right]$$

With $\pi(s) = \arg\max_a Q_\pi(s, a)$ we find at convergence:

$$\mathrm{E}[\delta^{(k)}] = \mathrm{E}\left[R_{t+1} + \max_A Q_\pi^{(k)}(S_{t+1}, A) - Q_\pi^{(k)}(S_t, A_t)\right] = 0$$

i.e. Q-Learning stops when the Q-values correctly estimate the values of the greedy policy.

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

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

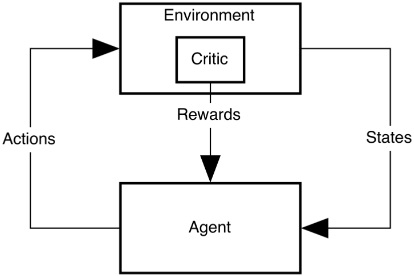
Main Idea: For observed transition S_t, A_t, S_{t+1}, adapt the parameters θ with gradient descent on the squared TD-Error loss function

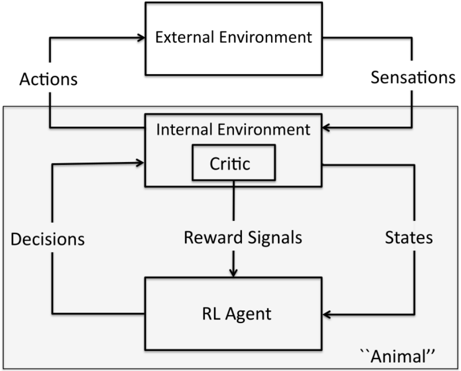
$$\theta^{(k+1)} = \theta^{(k)} - \lambda \frac{\partial}{\partial \theta} \left(R_{t+1} + \max_a q_{\theta^{(k)}}(S_{t+1})_a - q_\theta(S_t)_{A_t} \right)^2$$

In practice many additional tricks are used...

Table of Contents

An Evolutionary Perspective





Hypothesis: Evolution has equipped animals with a reasonable internal reward system.

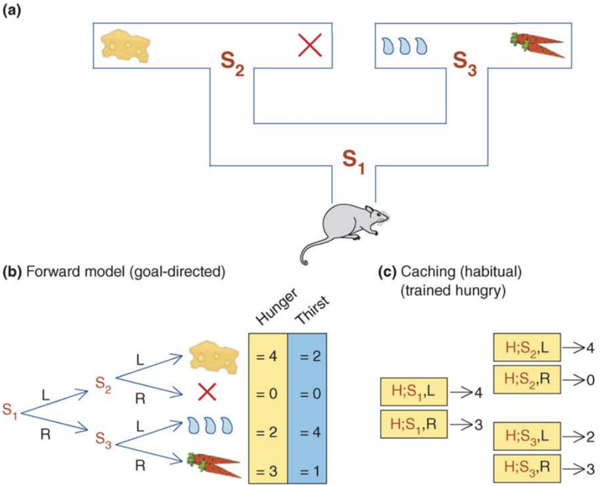
http://dx.doi.org/10.1109/tamd.2010.2051031

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



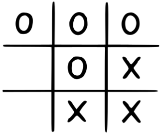
It is hypothesized that animals and humans rely on a model-based reinforcement learning system that allows goal-directed planning and a model-free reinforcement learning system that forms habits.

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.

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

Table of Contents

Application: Tic-Tac-Toe



The player who succeeds in placing three of their marks in a diagonal, horizontal, or vertical row is the winner.

There are 255'168 different games, 131'184 end in a win of the first player, 77'904 end in a win of the second player and 46'080 end in a draw.

There are 5'478 different board positions (765 without rotations and mirroring).

- We use an enumerated representation of the states, i.e. $s \in \{1, 2, 3, \ldots 5478\}$.
- We use a Monte Carlo Estimator to learn the Q-values.
- Two epsilon-greedy players (cross and nought) play against each other in **self-play**.
- Each player stores all states (encountered when it is the players turn) together with the selected action in its own episode.
- Each episode ends with reward +1 for the winner, reward -1 for the looser and reward 0 in case of a draw.

Book

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`