Introduction to Artificial Intelligence

13. Reinforcement learning

Bojana Dalbelo Bašić, Marko Čupić, Jan Šnajder

University of Zagrebu
Faculty of Electrical Engineering and Computing

Ak. god. 2021./2022.

v3.10

Types of learning - reminder

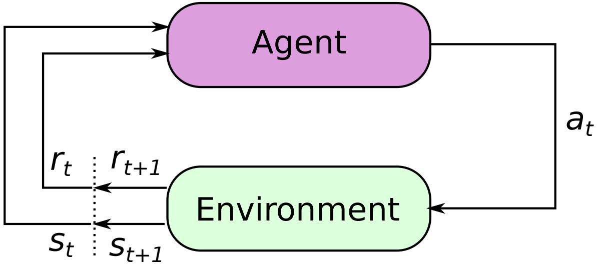
- *Unsupervised learning* - examples ar of form $(\vec{x})$ - clustering
- *Supervised learning* - examples ar of form $(\vec{x}) \rightarrow (\vec{y})$ - classification, function approximation
  - $\vec{x}$ the whole example; in case of game playing, all moves played during the game
  - $\vec{y}$ is class designation of function value; in case of game playing, total score earned during the game
  - In case of game, the problem is how to determine which of the played moves resulted in which portion of the score?
- People don't learn like that. E.g. little child learning to walk starts to run rather fast - reward is excitement. But soon after, it falls and feel spain, so it learns to be more careful. Learning is done based on the reward received during the action or with some delay.
- *Reinforcement learning* mimics this type of learning.

Content

Model

Model sutable for reinforcement learning is comprised of:

- *Agent* - a system which learns optimal behavior in environment based on observed environment states s and received rewards r
- *Environment* - in which agent undertakes action a_t causing the environment to change the state in s_{t+1} and receives a reward r_{t+1} from environment



Slika: Model

Environment

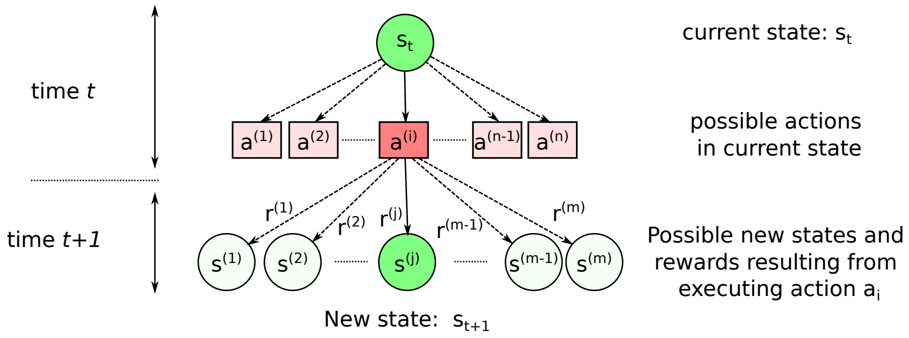
Environment in which agent operates can be deterministic or stohastic.

- When agent makes an action, the environment will change its state and will give some reward to the agent
- Both can be stohastic processes - it is possible that from same state and same action the enviroment will not transit into the same state every time, and to give different reward each time.

We are looking only at the environments satisfying the Markov Decision Process property, i.e. in which the conditional probability of environment transitioning into the next state can be determined just from the data available in the current state:

$$p(s_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}, s_{t-2}, a_{t-2}, \ldots, s_0, a_0) = p(s_{t+1}|s_t, a_t)$$

Environment



Slika: How action affects the environment

How agent operates

The way on which agent selects an action to perform is called policy and is denoted by π.

- Agent policy is a function which maps state s into selected action a. One possible task of reinforcement learning is to learn the optiomal policy for a problem that agent is facing.

Agent should always select actions in a way which maximizes the sum of the received rewards, i.e.:

$$R_t = r_{t+1} + \gamma \cdot r_{t+2} + \gamma^2 \cdot r_{t+3} + \gamma^3 \cdot r_{t+4} + \cdots + \gamma^{T-t-1} \cdot r_T = \sum_{k=0}^{T-t-1} \gamma^k \cdot r_{t+k+1} \tag{1}$$

where γ is a factor used to balance between an attempt to maximize immediate reward of the move (for $\gamma = 0$) and on the other hand the sum of all received rewards until the end of episode (for $\gamma = 1$).

How agent operates

It holds:

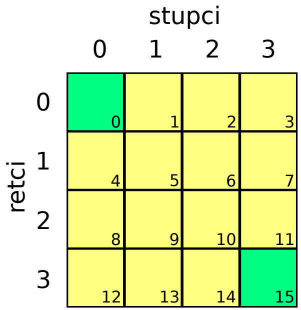
$$\begin{aligned} R_t &= r_{t+1} + \gamma \cdot r_{t+2} + \gamma^2 \cdot r_{t+3} + \gamma^3 \cdot r_{t+4} + \cdots + \gamma^{T-t-1} \cdot r_T \\ &= r_{t+1} + \gamma \cdot \left(r_{t+2} + \gamma^1 \cdot r_{t+3} + \gamma^2 \cdot r_{t+4} + \cdots + \gamma^{T-t-2} \cdot r_T \right) \\ &= r_{t+1} + \gamma \cdot R_{t+1} \end{aligned}$$

The problem which the agent is solving can be:

- *episodic* - if it is done in a finite number of steps, meaning also that the environment must contain terminal states
- *continuous* - if agent can perform action forever

Example: Grid world 1

Robot is in a grid world. Exits are green rectangles. Each step that robot makes costs it 1 unit of energy (reward is -1). Let $\gamma = 1$.



Slika: Grid world 1

Using given policy we want to determine total expected reward which the robot will get if it starts from each of the cells.

Playing a single episode

```
public static <S,A> double play(DiscreteEnvironment<S, A>
    world, Policy<S, A> policy, S startState, double gamma)
    {

  world.setCurrentState(startState);
  double totalReward = 0;
  double scaler = 1;

  while(!world.isFinished()) {
    S currentState = world.getCurrentState();
    A selectedAction = policy.pickAction(currentState);
    double reward = world.applyAction(selectedAction);
    totalReward += scaler * reward;
    scaler *= gamma;
  }

  return totalReward;
}
```

Example: Grid world 1

If the policy is "pick move direction randomly with uniform discribution":

```
Value function:
   0.000  -13.958  -20.076  -21.990
 -14.002  -17.986  -19.875  -20.009
 -20.136  -19.973  -17.992  -14.000
 -22.142  -19.971  -13.889    0.000
```

If the policy is "in 70% go up, in remaining 10%+10%+10% go left/right/down":

```
Value function:
   0.000  -29.911  -48.873  -58.008
  -5.658  -30.672  -48.666  -57.477
 -10.724  -31.219  -47.268  -51.167
 -14.545  -31.685  -41.595    0.000
```

Value function and action values

Value function under the policy π:

$$v_\pi(s) = \mathbb{E}_\pi[R_t|s_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k r_{k+t+1}|s_t = s\right] \tag{2}$$

Action value function under the policy π:

$$q_\pi(s,a) = \mathbb{E}_\pi[R_t|s_t = s, a_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k r_{k+t+1}|s_t = s, a_t = a\right] \tag{3}$$

It holds:

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s'} p(s'|s,a)\left[r(s,a,s') + \gamma \cdot v_\pi(s')\right] \tag{4}$$

and if we note that inner sum is actually q-value:

$$v_\pi(s) = \sum_a \pi(a|s) \cdot q_\pi(s,a) \tag{5}$$

Policy Evaluation

If whe have access to policy model and environment model, value function under the policy can be determined using more efficient procedure called policy evaluation.

The procedure is iterative.

1. Set values on some initial ones (e.g. 0) in an array v. Terminal states must be 0.
2. Repeat as long as the total change is larger the some threshold
   1. Using an array v and expression (4) calculate values for each state and put the result in $v^{'}$
   2. $v \leftarrow v^{'}$

The procedure can be shown to converge. It is also rather efficioent.

Example: Grid world 1 - policy evaluation

If the policy is "pick move direction randomly with uniform discribution":

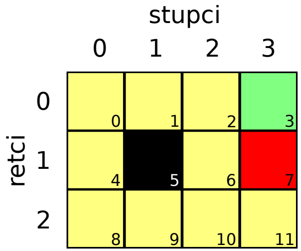
```
Value function:
   0.000  -14.000  -20.000  -22.000
 -14.000  -18.000  -20.000  -20.000
 -20.000  -20.000  -18.000  -14.000
 -22.000  -20.000  -14.000    0.000
```

If the policy is "in 70% go up, in remaining 10%+10%+10% go left/right/down":

```
Value function:
   0.000  -29.784  -48.928  -58.205
  -5.670  -30.423  -48.796  -57.481
 -10.607  -31.280  -47.557  -51.103
 -14.490  -31.675  -41.620    0.000
```

Example: Grid world 2

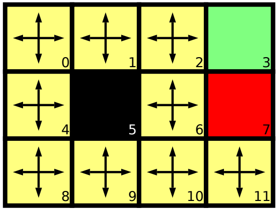
Environment is nondeterministic. Selected action is completed succesfully in 80% cases, in 10%+10% one of "non-opposite" action is performed. E.g. action "up" will in 80% cases move the robot up, in 10% cases to the left and in 10% cases to the right. Cell 5 is wall, cell 7 is fire pit which destroys the robot (reward -1), cell 3 is final (reward +1). All other transitions give reward r (so called *living reward*, usually negative).



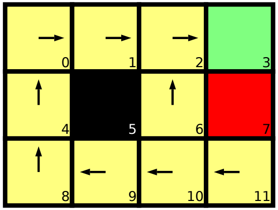
Slika: Grid world 2

Example: Grid world 2

Value function under different policies and with $r = 0$ and $\gamma = 1$ (left: policy; right: value function):



-0.038 0	0.089 1	0.215 2	0.000 3
-0.165 4	5	-0.443 6	0.000 7
-0.291 8	-0.418 9	-0.544 10	-0.772 11



0.973 0	0.973 1	0.973 2	0.000 3
0.973 4	5	0.753 6	0.000 7
0.973 8	0.973 9	0.948 10	0.732 11

Bellman's equations

If the agent is in state s, to play optimal, it needs to select the action which maximizes the sum of rewards it will receive by selecting that action and later again by playing optimal. But then for value function hold Bellman's equations:

$$v^*(s) = \max_a \left(\sum_{s'} p(s'|s,a) \left[r(s,a,s') + \gamma \cdot v^*(s') \right] \right) \tag{6}$$

$$v^*(s) = \max_a q^*(s,a) \tag{7}$$

where $v^*(s)$ denotes optimal value function, and $q^*(s,a)$ denoted optimal q-values.

Bellman's equations

Optimal value function and q-values are formally defined as follows:

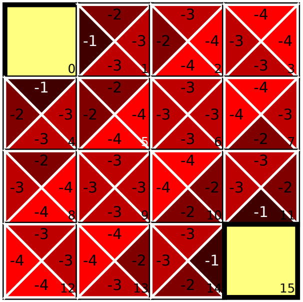
$$v^*(s) = \max_{\pi} v_\pi(s)$$

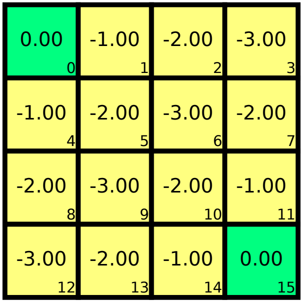
$$q^*(s,a) = \max_{\pi} q_\pi(s,a)$$

where maximum is looked over all possible policies. Policy π for which that maximum is achieved will be denoted π^* and called optimal policy.

Grid world 1: Value Iteration

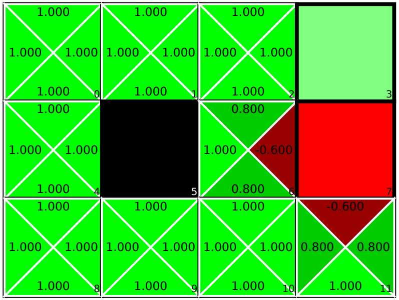
Value iteration is a proceure similar to policy evaluation, but the values are updated using Bellman's equation (6). Left: q-values; right: value function.

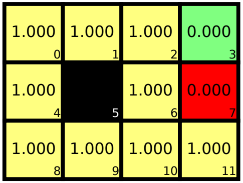




Grid world 2: Value Iteration

Value iteration is a proceure similar to policy evaluation, but the values are updated using Bellman's equation (6). Left: q-values; right: value function.





Reconstructing the optimal policy

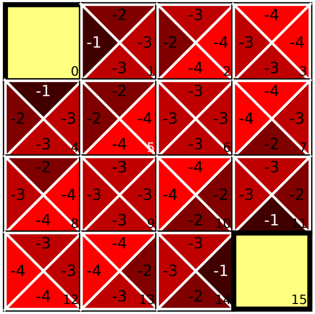
Once we have determined optimal value function, we can reconstruct the optimal policy:

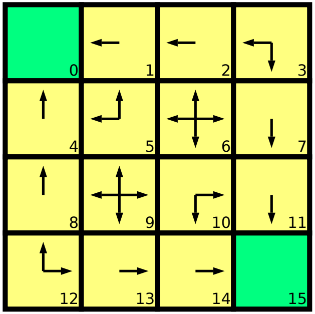
$$\pi^*(s) = \arg\max_a \left(\sum_{s'} p(s'|s,a) \left[r(s,a,s') + \gamma \cdot v^*(s') \right] \right) \tag{8}$$

$$\pi^*(s) = \arg\max_a \left(q^*(s,a) \right) \tag{9}$$

Grid world 1: Optimal policy

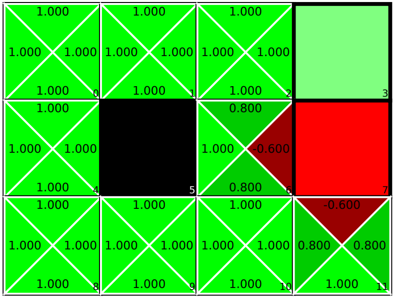
Optimal policy reconstructed from learned q-values using value iteration.

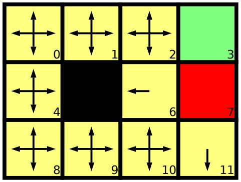




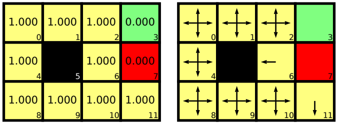
Grid world 2: Optimal policy

Optimal policy reconstructed from learned q-values using value iteration.





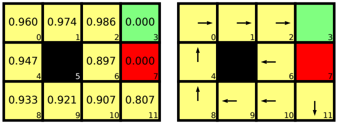
Grid world 2: influence of living reward



1.000 0	1.000 1	1.000 2	0.000 3
1.000 4	■ 5	1.000 6	0.000 7
1.000 8	1.000 9	1.000 10	1.000 11

↑↓←→ 0	↑↓←→ 1	↑↓←→ 2	🟩 3
↑↓←→ 4	■	← 6	🟥 7
↑↓←→ 8	↑↓←→ 9	↑↓←→ 10	↓ 11

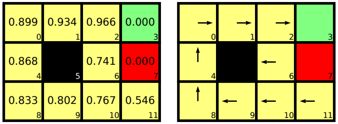
r=0, γ=1



0.960 0	0.974 1	0.986 2	0.000 3
0.947 4	■ 5	0.897 6	0.000 7
0.933 8	0.921 9	0.907 10	0.807 11

→ 0	→ 1	→ 2	🟩 3
↑ 4	■	← 6	🟥 7
↑ 8	← 9	← 10	↓ 11

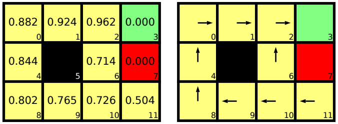
r=-0.01, γ=1



0.899 0	0.934 1	0.966 2	0.000 3
0.868 4	■ 5	0.741 6	0.000 7
0.833 8	0.802 9	0.767 10	0.546 11

→ 0	→ 1	→ 2	🟩 3
↑ 4	■	← 6	🟥 7
↑ 8	← 9	← 10	← 11

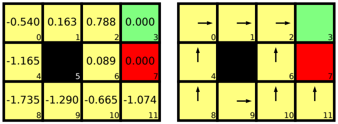
r=-0.025, γ=1



0.882 0	0.924 1	0.962 2	0.000 3
0.844 4	■ 5	0.714 6	0.000 7
0.802 8	0.765 9	0.726 10	0.504 11

→ 0	→ 1	→ 2	🟩 3
↑ 4	■	↑ 6	🟥 7
↑ 8	← 9	← 10	← 11

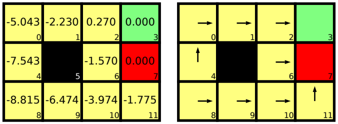
r=-0.03, γ=1



-0.540 0	0.163 1	0.788 2	0.000 3
-1.165 4	■ 5	0.089 6	0.000 7
-1.735 8	-1.290 9	-0.665 10	-1.074 11

→ 0	→ 1	→ 2	🟩 3
↑ 4	■	↑ 6	🟥 7
↑ 8	→ 9	↑ 10	↑ 11

r=-0.5, γ=1



-5.043 0	-2.230 1	0.270 2	0.000 3
-7.543 4	■ 5	-1.570 6	0.000 7
-8.815 8	-6.474 9	-3.974 10	-1.775 11

→ 0	→ 1	→ 2	🟩 3
↑ 4	■	→ 6	🟥 7
→ 8	→ 9	→ 10	↑ 11

r=-2, γ=1

Summary

1. If we have the environment model and the policy model, we can determine value function:
   - using simulation: not efficient
   - policy evaluation: efficient
2. If we have the environment model, we can determine optimal value function using Value Iteration and then reconstruct optimal policy.
   - There is an alternative procedure called *Policy Iteration*, but we wont cover that here.

But what can we do if we don't have the model of the environment?

Content

Learning during the single episode

Now we will look at the case where agent goes through episodes, and it learns directly during the interaction with the environment (whose model is not known).
We will cover Q-learning algorithm that learns q-values which can then be used to reconstruct the optimal policy.
This learning algorithm is iterative. It initializes q-values to some predetermined values (i.e. 0). Then agent goes through each episode, and in each step it uses ϵ-greedy policy which it learns at the same time.
ϵ-greedy means that in ϵ percent of cases agent picks and action randomly, and in $(1-\epsilon)$ percent of cases it picks the best action based on its current knowledge (by learned policy). Early on, ϵ should be big in order to allow agent to boldly explore; later, ϵ should be smaller and smaller so that agent more relies on learned policy.

Q-learning algorithm

Q-learning algorithm learns directly q-values which are defined as:

$$q(s,a) = r(s,a,s') + \gamma \cdot v_\pi(s')$$

It uses the fact:

$$v_\pi(s') = \max_{a'} q(s',a')$$

so it updates q-values using:

$$q(s,a) \leftarrow (1-\alpha) \cdot q(s,a) + \alpha \cdot \left(r(s,a,s') + \gamma \cdot \max_{a'} q(s',a') \right) \tag{10}$$

Here, optimal value function in state s' is calculated based on all q-values of actions in state s'.

Q-learning algorithm

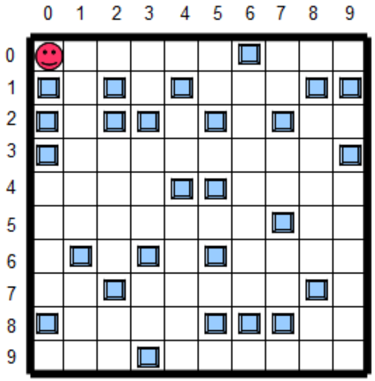
α is learning rate between 0 and 1. If it is 0, noting is updated; if it is 1, only the new value is carred over. Values between 0 and 1 perform linear interpolation, and we can show that:

$$(1-\alpha)\cdot q(s,a)+\alpha\cdot\left(r(s,a,s')+\gamma\cdot\max_{a'} q(s',a')\right)=$$
$$q(s,a)+\alpha\cdot\left\{\left(r(s,a,s')+\gamma\cdot\max_{a'} q(s',a')\right)-q(s,a)\right\}$$

Example: Robot Robby

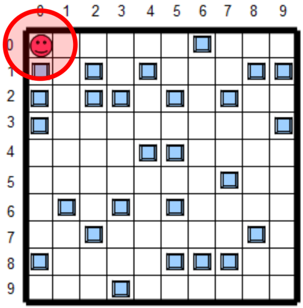
A short reminder on robot Robby:

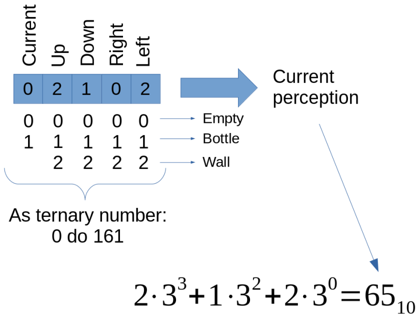
- it needs to collect as much as possible bottles in 150 steps
- based on perception of its enviroment, robot needs to select an action to perform



Example: Robot Robby

Robby's perception can be encoded using and integer - there are in total 162 different perceptions:





$$2 \cdot 3^3 + 1 \cdot 3^2 + 2 \cdot 3^0 = 65_{10}$$

Example: Robot Robby

There are 7 actions that Robby can perform:

- 0: do nothing
- 1: pick up bottle
- 2: go up
- 3: go down
- 4: go right
- 5: go left
- 6: go in random direction

In this example states are different perceptions. Therefore we have 162 different states, and in each of them 7 possible actions. So we use qTable[162][7].

Example: Robot Robby

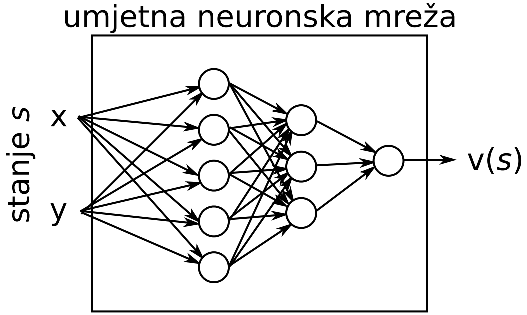
- Learning is done in 100 000 epochs.
- During each epoch we work on 20 different worlds.
- We regenerate worlds randomly after each 101 epochs (in order to prevent overfitting).
- Parameter α is linearly changed from 0.1 to 0.001 during first 60% epochs and then it remains on its minimal value.
- Parameter ϵ is linearly changed from 0.1 to 0.001 during first 60% epochs and then it remains on its minimal value.
- Parameter $\gamma = 0.9$
- World is 10×10 and it is filled up with 35% of bottles.
- Rewards: +10 for each bottle, -5 if picking up on empty cell, -10 if coliding with a wall, living reward is 0.

Example: Robot Robby

(DEMONSTRATION)

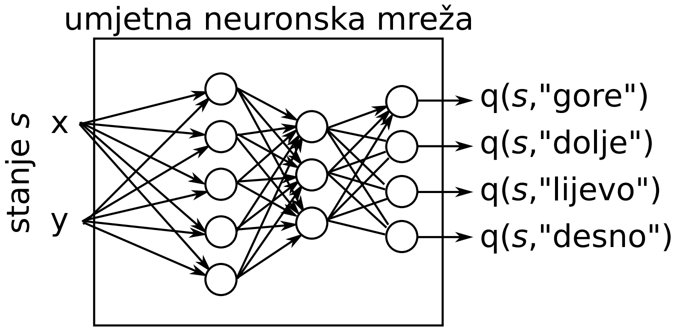
Large and/or continuous state spaces

If state space is large and/or continuous, then value function or q-values cannot be stored in tables (and keep in memory using arrays).



Large and/or continuous state spaces

If we work with q-values, we would use approximation model with multiple outputs - one for each action which generates q-value for that action. Image below depicts this on an example of a grid world where possible actions are "up", "down", "left" i "right".



Large and/or continuous state spaces

In both cases artificial neural network is used for approximation of value function or q-values.
Update is performed on a standard way using backward pass (and gradient methods).

Conclusion

- We gave the most basic introduction in reinforcement learning
- There are many more advanced algorithms invented to fix some shortcommings of the basic algorithms
- Today reinforcement learning is blending with deep learning and deep models are used to approximate the functions that are being learned