Could you train your vacuum cleaner like your dog?

Tomasz Tajmajer

Introduction

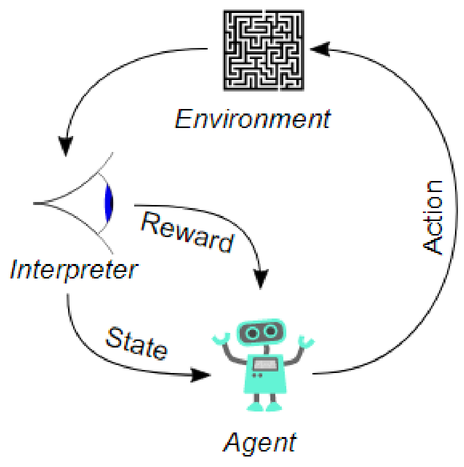
Artificial intelligence methods are often perceived as extremely complicated and understandable only by people that graduated computer science. The idea of an electronic device that, by itself, learns how to behave still seems to be very futuristic. In fact, we have many “smart” devices around us, but many of them are not smarter than a motion-based light switch. Yet, our world is full of intelligent agents: rats, cats, dogs, cockroaches and humans. The secret behind their behaviors lies in millions of years of evolution and something that we all know: trials and errors, rewards and punishments.

Let’s imagine a rat in a specially designed rat cage that has a button and a food supplier. When the button is pressed, a portion of food is supplied. A rat placed in such cage for the first time will use its **senses** to more or less randomly **explore** his new environment. At some point, the rat will press the button and receive a **reward** - some food will be supplied. Over time this situation will repeat and finally, the rat will **learn** that pressing the button releases the food. From now on, the rat will **exploit** his knowledge each time he feels hunger.

Now, consider a robotic vacuum cleaner - like one of those currently available on the market. Such a vacuum cleaner is able to autonomously move around the room and clean the carpet or the floor. It is smart enough to avoid collisions with various objects, it may also automatically return for recharging or be used by a cat as a taxi. Let us however assume that the vacuum cleaner comes with no predefined “intelligent” behavior at all - it just moves randomly hitting everything around. Would it be possible for the vacuum cleaner to learn by itself how to behave? Could it correct its behavior just by knowing that it *done something wrong*? In fact it could - using **reinforcement learning**.

Making mistakes

Reinforcement Learning is a method that, to some extent, mimics the learning of living creatures. In reinforcement learning we have some **environment** and an **agent**. The agent is able to perceive (sense) the **state** of the environment and perform **actions**. After performing an action, the state of the environment changes and a **reward** signal is provided to the agent.



In reinforcement learning, the reward is basically a number, which may have any real value. Often positive numbers are considered as positive rewards and negative numbers as negative rewards (punishments), however it is really the relation that matters.

Let us return to our cleaning robot. Let us assume that the robot have various sensors by which it senses the environment each 1 second. After that the robot's control algorithm may chose an action: move a bit forward, turn left or turn right. If the robot collides with anything, we provide it a negative reward signal (e.g -1). If it drives over some dirty place on the floor, we provide it with a positive reward signal (e.g. $+1$). In other cases we provide it with no reward (0).

Now, as the robot moves randomly, it collects rewards after each action (performed each 1 second). After some time, the robot will have a sequence of rewards such as : $[0,0,0,-1,0,0,-1,0,0,0,0,1,0,0,-1,0,0,\ldots]$. We may notice, that if the robot behaved as expected (avoided collisions and cleaned dirt), it would collect only $+1$ and 0 rewards. On the other hand, a reward sequence full of -1 would indicate that the robot constantly collides with something. As a matter of fact, we can calculate the **sum of rewards** and use it as an indicator of *how well* the robot is behaving.

Maximizing your needs

Let us say, that there are some **terminal states** - states that indicate the finish of a game, or accomplishment of some task. For example, a check-mate is a terminal state in chess. Running out of batteries may be a terminal state in case of a vacuum cleaner (as well as falling from the stairs and crushing into little pieces). When an agent achieves a terminal state, we get a sequence of rewards from some initial state till the terminal state. We can symbolically denote it as $[r_0,r_1,r_2,...,r_N]$. Now we can calculate the sum of rewards, that we will refer to as the **return**:

$$R = \sum_{t=0}^{N} r_t = r_0 + r_1 + r_2 + \cdots + r_N$$

It is clear that the better an agent behave, the greater the return value is. In fact, we can say that the **goal** of the agent is to **maximize** the return value.

For a general case, we can define a **discounted return value**:

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + \ldots$$

Such defined return allows us to say what will be the return if we start at timestep t and collect rewards from now on till infinity. The parameter γ is the **discount factor** (usually near 0.99), which regulates the impact of rewards received further in the future. Of course, the agent does not know what rewards it will receive in the future, but based on its **past experience** it can **expect** a return if it follows some strategy of action selection. We will refer to this strategy as to the **policy** of an agent and denote it by π. The policy π basically says what is the probability of choosing some action in a given state. For some types of problems, there is an optimal policy $\pi_\star$, following which results in getting the biggest return possible.

Using your experience

We can now more precisely say what our vacuum cleaner should do: it should learn some optimal policy, following which will leads to the maximum return value and thus, to the most desired behavior (with respect to the defined rewards - one may define rewards such that the vacuum cleaner will learn to hunt for the cat). The hard question is: how to learn the policy?

The general solution is to collect many examples of experiences (states, actions and rewards) and based on them derive the relations between selected actions and obtained rewards. It will not be surprising that there are actually many different reinforcement learning methods that use different approaches for tackling with this problem.

One of the issues in reinforcement learning is so called *credit assignment problem*. When an agent receives a reward, how can it know which past action actually was the one that had the most impact? Let us return to the rat example: if the food was supplied to the rat 12 hours after it pressed the button, would the rat discover the relation between pressing the button and receiving food? We see that if the rewards are generated very rarely, then the learning process may take a long time or even be impossible.

Let us describe now one, quite popular, reinforcement learning method - **Q-learning**. First, we will define a function Q(s,a) that for a given state s and action a will say how good it is to select this action in this state. For example, let us consider that our vacuum cleaner is located in a corner of a room in such way that moving forward or right will result in collision. The vacuum cleaner may choose to move forward, turn left or turn right. If we calculate the value of Q(s,a) for each possible action, then the returned values should be such that Q(s,left)>Q(s,forward) and Q(s,left)>(s,right). Thus using the Q-function, we may conclude that action a=left is the best one to select (as it has the greatest Q-value).

Q-learning is based on iterative updating of the Q-function. We start with Q-function that does not give any meaningful suggestions about the actions. Then while the agent explores the environment, received rewards are used to update the Q-function using the following equation:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(s_t, a_t) \right]$$

Let us analyze it step by step:

- α is so called *learning rate* and it determines the speed of learning. It has usually some small value e.g. $\alpha=0.1$
- γ is the discount factor described earlier, e.g $\gamma=0.99$
- To calculate the new $Q(s_t,a_t)$ value, we take the old value and increase it with a small portion (α) of $[r_{t+1}+\gamma \max_a Q(S_{t+1},a)-Q(s_t,a_t)]$, where
  - r_{t+1} is the reward received for performing action a_t in state s_t
  - $\max_a Q(S_{t+1},a)$ is the Q-value for the best action that we can select in the new state s_{t+1}
  - $Q(s_t,a_t)$ is the old value

In theory, if we repeat the Q-function update step many times using many experience examples, the Q-function should converge to an optimal solution (in many real life scenarios it will be a sub-optimal solution). Then we can derive a policy by selecting the best action suggested by Q(s,a) for each state s that the agent experiences.

Is it good to be greedy?

In reinforcement learning there is, so called, exploration vs exploitation problem. Basically, the problem is how the agent should gather knowledge about states that it did not yet experienced and when it should choose to exploit its knowledge rather than seek for new solutions.

Assume that our vacuum cleaner approached a wall and decided to turn left. This decision lead to positive rewards in the future, so on the next time when this happened, the robot also turned left. Eventually, the vacuum cleaner learned to always turn left when in front of a wall. We know however, that there is a one more good solution - to turn right. The vacuum cleaner did not learned that, because it never tried to perform such action as it started to **exploit** its knowledge immediately. The robot followed a greedy strategy - it selected the first suitable solution, without checking if there are any better ones.

We see that it would be better if the agent explored possible states and actions first (even at the cost of making many mistakes), and only then, after it gathered enough knowledge about the environment, it should start to exploit its knowledge.

A simple and common method for solving this issue is so called ϵ-greedy strategy: the first actions chosen by the agent are completely random, then as the agent gathers more experience, the probability of selecting a random action decreases. The agent then choses some actions randomly and some according to its knowledge. Finally, at the end of the learning process, the agent no longer performs any moves randomly.

The curse of dimensionality

The classical approach to Q-learning has a certain flaw - we need to store Q-function values for each observable state. If our problem is a bit more complex than tic-tac-toe, then the number of states may be **extremely large**. This makes Q-learning not suitable for real-life problems. But there is a solution - we can *approximate* the Q-function using neural networks.

Neural networks are very powerful when it comes to many tasks such as classification or regression. They are also universal approximators - we can train them to approximate any function (with certain accuracy). In 2013 a Deep Q-learning method was proposed. This method uses deep neural networks, to approximate the Q-function. What is the most astonishing, is the fact that the states in this method may be represented by images.

Deep Q-learning may use high-level visual input - like screenshots from video games, and rewards so simple as the score from a video game, to effectively learn how to play those video games. In 2015 a publication in Nature presented the results of using Deep Q-learning with 49 Atari games. For most of them, the algorithm played better than humans.

Conclusions

Going back to our initial question - could we train our vacuum cleaners just in the same way that we train our household pets? According to the current state of the art - why not. Applying methods such as Deep Q-learning to robots, should give very interesting results. For example, we can imagine a vacuum cleaner that can recognize our voice and thus recognize when we are shouting at it. When the vacuum cleaner does something wrong and we give it a vocal rebuke, it may dose itself a negative reward and then improve a bit its performance. Sounds reasonable? But what if the cat learns to meow at the vacuum cleaner as well…