DIGITAL CULTURE
ITMO UNIVERSITY

Reinforcement Learning

High School of Digital Culture
ITMO University
dc@itmo.ru

Contents

1 The Concept of Reinforcement Learning

1.1 Basic Concepts and Formulation of the Problem

Hello everyone! In this module, we are going to discuss a branch or a different type of ML, which is known as reinforcement learning (or RL). We will begin with the justification and focus on the differences between RL and other ML types we considered.

So, let's recall what we do to solve a supervised learning problem. We have a set of predictors (or input variables) $X_1, X_2, ..., X_p$ and a set of responses Y (or a set of correct answers). Our task here is to generalize the experience (the experience gained from training data) to the entire space (random test data). When solving a regression problem when $p = 1$, we have a set of n data pairs $(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$ only. We aim to construct a relationship of the type $Y = f(X)$ to predict a correct answer for any X from the set of all possible values of the predictor X_1.

The same also applies to classification problems. Predefined classes of a finite number of training objects allow the classification algorithm to color the entire feature space (with an infinite number of objects) in colors corresponding to different classes. This is how the entire space is generalized. These statements are also true for unsupervised learning.

However, real-life learning doesn't always correspond to the flowchart where the correct answers are known. For example, the development of a new drug and the derivation of a new complicated formula are the processes with no responses except a final one (whether the developed drug is effective or whether the formula is correct). The process of creation (including that of a formula or drug) can be described as follows. When we do something, something happens. We estimate the result, think it over, and then continue doing something to obtain the desired outcome. No doubt that the assertions in this logic describe many different processes in everyday life.

1. For example, let's consider how babies learn to walk after they learn to crawl (note that our description can be inaccurate). First, babies bend their knees and bounce up, but fall without support. Soon they realize that it's easier to get up while holding mom's hand. Then babies start trying to stand up with support. A chair, coffee table, wall... Everything will do! Just climb a little bit, and here you go! That's how babies learn to get up.

2. What about a football match? Each team has tactics to make the most of its strengths and to hide weaknesses, as well as to tackle those of a competitor. However, it's difficult to predict the outcome. There are too many factors to consider. Anyway, today the right wing opens many opportunities for an

attack. Probably, the defenders are not in their best shape. Well, it makes sense to attack the right wing.

There are more examples, but you probably get the idea. Let's formally describe the problem and introduce definitions for future use.

Roughly speaking, reinforcement learning is a subset of machine learning based on the following paradigm: a trained model has no detailed information about the system surrounding it, but the model can take actions in this system and analyze the reaction of the latter. To put it differently, it can analyze the reinforcement. Since the model affects the system and the system affects the model, we cannot unpair them. Thus, the next definition is rather conditional.

Definition 1.1.1 *In reinforcement learning, a trained model is called an agent. The system surrounding the agent is known as an environment.*

Now we can formally describe the interface between the agent and the environment.

1. Let the agent and environment interact at discrete time steps $t \in \{0, 1, 2, ..., n\}$, $n \in \mathbb{N} \cup \{0, \infty\}$: the number of interactions can be finite (if n is finite) or infinite.

2. At each step, the agent is in some state $s_t \in S$, where S is a set of all possible states.

3. Based on the state s_t, the agent selects an action $a_t \in A(s_t)$, where $A(s_t)$ is a set of actions available to the agent in the state s_t.

4. After an action a_t, the environment generates a reward r_{t+1} and transitions the agent to a state s_{t+1}.

5. Steps 3-4 are repeated until a stopping criterion is met.

Later, we will explain what we mean when we say 'generate' and 'transition' and describe the process mathematically correct. For now, to fix this in mind, let's imagine the dialog between the agent and the environment:

1. **Environment:** Agent, you're in the state s_0. The actions available to you are $3, 4, 5$ ($A(s_0) = \{3, 4, 5\}$).
   Agent: I take the action 4 ($a_0 = 4$).

2. **Environment:** You receive the reward of 3 units ($r_1 = 3$) and transition to the state s_1. Now, the available actions are $1, 4, 5$ ($A(s_1) = \{1, 4, 5\}$).
   Agent: I take the action 5 ($a_1 = 5$).

3. **Environment:** You receive the reward of -1 unit ($r_2 = -1$) and transition to the state s_2. The available actions are... And so on.

As you can see, rewards (reinforcement) can be positive or negative. The agent objective of maximizing the cumulative reward is based on the reward hypothesis. The hypothesis says that all goals can be described by the maximization of the expected cumulative reward. We will explain what it means a little bit later.

Well, the interface seems to be clear. However, here's another problem we are going to focus on. We need to find a way to select the action a_t at the step t from the set of possible actions $A(s_t)$. This leads us to the concept of a policy.

Definition 1.1.2 *A set of probabilities $\pi_t(a|s_t)$ of selecting the available action a at the time step t (when the agent is in the state $s_t \in S$) is called the agent's policy.*

In other words, when at least one action of the set $A(s_t)$ is available in the state s_t at the time step t, the choice of the action is probabilistic under the policy $\pi_t(a|s_t)$. How to set the policy that maximizes the expected cumulative reward? And it is the million-dollar question that we will answer.

1.2 Trade-off between Exploration and Exploitation

Before we consider concrete examples and tasks, let's look at the RL issue called the exploration-exploitation dilemma or the trade-off between exploration and exploitation. The exploitation principle assumes that, at each step, the best (or the most rewarding) choice should be made based on the available information. The exploration principle changes the concept: it requires to collect more information for (possibly) more successful exploitation in the future. The idea is illustrated in Fig. 1.

Assume that a mouse (our agent) makes one move (action). The game ends when the mouse eats the cheese, when it is trapped, or when it has no more moves (say, there are 10 of them). When a finite number of games is played (the conducted experiments), the total amount of the eaten cheese is calculated. The greater the amount of the eaten cheese, the better. Moreover, the case when no cheese is eaten (no moves left), and the mouse is not trapped (the agent is not killed) is better than that when the mouse is trapped (which means the agent is killed).

The figure shows that the mouse can reach a small piece of cheese easier, safer, and faster because it is nearby. This policy yields a reward (a small one but still) by the end of each game. This approach is like the proverb 'A bird in the hand is worth two in the bush', and, according to it, the pile of cheese near the traps is worthless. Well, we've just described the exploitation principle.

On the other side, instead of fast results, the mouse can explore the environment and try to find a way to a greater reward to be counted toward the total

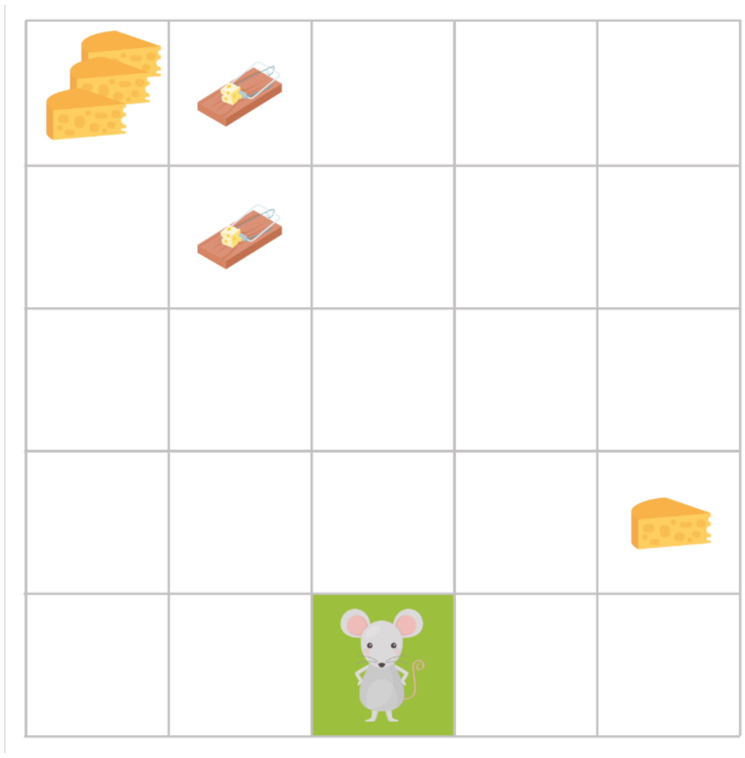


Figure 1: The trade-off between exploration and exploitation.

amount of the cheese, although such a path may be dangerous, difficult, and even deadly.

The main theses of the trade-off between exploration and exploitation are these:

1. It is necessary to collect sufficient information about the environment so that the solutions (and the results) are best on average.

2. The best long-term policy tolerates (and even includes) short-term losses.

The exploration-exploitation dilemma doesn't come out of anywhere. It's a part of everyday life. For example, you decided to have dinner at a restaurant. Exploitation is about choosing a previously visited restaurant that offers great food and good value. Exploration is about trying a new restaurant. What if it is a better choice?

So, a reinforcement learning problem and its formulation should be clear now. Next, we will consider one of the well-known and easy-to-understand problem: a k-armed bandit problem.

1.3 Formulation of the k-Armed Bandit Problem

A simple example that illustrates the idea of reinforcement learning is a k-armed bandit problem. It is named by analogy to a slot machine known as one-armed bandit, however, in our case, it has k levers instead of one.

Before we start, let's decide upon the agent, environment, policy, and so on. An agent is a person who (at each time step t in any state $s \in S$) can choose any of k actions of the same kind, in other words, pull one of the k levers. The choice made each time depends on the policy π_t. Having chosen the action a_t at the time step t, the agent receives a previously unknown reward r_{t+1} from the environment (the k-armed bandit) and transitions to the next state.

Remark 1.3.1 *Note that, since the options (or a set of possible actions $A(s)$) of the agent in each state $s \in S$ are the same, the choice of one of the k levers, as well as of the agent state s are the same at each step (or there is only one state). This assumption is not correct because the agent in each state $s_t \in S$ at the time step t follows a policy π_t that can differ.*

Now let's think which levers to pull. If we had known the value of each action, the logical greedy policy would have been to choose such an action (lever) at the time step t that maximizes the reward r_{t+1}. But we don't know that and can only estimate the result of the previous actions: pulled this lever, gained that much, pulled that one, gained that, and so on. It's reasonable to introduce the following definition.

Definition 1.3.1 *Let $q_0(a) \in \mathbb{R}$ be an initial estimated value of an action a (set by a researcher). The estimated value of an action a at the time step $t \geq 1$ is the value*

$$q_t(a) = \begin{cases} q_0(a), & \text{an action } a \text{ hasn't been selected} \\ \dfrac{\sum\limits_{i=0}^{t-1} r_{i+1} \mathsf{I}(a_i = a)}{\sum\limits_{i=0}^{t-1} \mathsf{I}(a_i = a)}, & \text{otherwise} \end{cases},$$

where $\mathsf{I}(a_i = a)$ is an indicator of the event $a_i = a$: it equals one if, at the ith time step, an action a is selected, or zero otherwise.

Remark 1.3.2 *This expression (if the action a has been selected at least once) is an arithmetic mean of rewards obtained when the action a is selected before the time step t.*

Let's look at an example.

Example 1.3.1 *Let the agent take one of two possible actions at each step in the game, a or b, under a policy and obtain a reward r_{t+1} at each step t. The data (in the introduced notation) is given in the table.*

Time step	***Action***	***Reward***
$t = 0$	$a_0 = b$	$r_1 = 3$
$t = 1$	$a_1 = a$	$r_2 = 0$
$t = 2$	$a_2 = a$	$r_3 = 8$
$t = 3$	$a_3 = b$	$r_4 = 1$
$t = 4$	$a_4 = b$	$r_5 = 5$

Let's find the estimated values of actions a and b at step 5. The action a was chosen 2 *times, and the rewards* 0 *and* 8 *were obtained. Thus, the estimated value of an action a at step 5 can be found as follows (as an arithmetic mean of the obtained rewards):*

$$q_5(a) = \frac{0+8}{2} = 4.$$

We obtain the same result if we take the original equality from the definition:

$$q_5(a) = \frac{\sum\limits_{i=0}^{5-1} r_{i+1}\mathbb{I}(a_i = a)}{\sum\limits_{i=0}^{5-1} \mathbb{I}(a_i = a)} = \frac{3\cdot 0 + 0\cdot 1 + 8\cdot 1 + 1\cdot 0 + 5\cdot 0}{0+1+1+0+0} = 4.$$

We can similarly find the estimated value of an action b at step 5:

$$q_5(b) = \frac{3+1+5}{3} = 3.$$

Note that the estimated value of an action a calculated at step 5 is greater than that of b. However, things may change if more information about rewards is collected.

The described approach for estimating the value has higher storage and computational demands. The reason is simple. To estimate the value, it is necessary to store the information about all the rewards obtained. At this point, it is reasonable to ask whether there is a less demanding approach. It turns out, there is. It is given by the lemma.

Lemma 1.3.1 (Recurrence formula for estimating the value) *Let the action a at $(t-1)$ steps be selected exactly $n \le (t-1)$ times and $q_t(a)$ be its estimated value at the step t. If it is selected at the step t, and the reward r_{t+1} is received,*

$$q_{t+1}(a) = q_t(a) + \frac{1}{n+1}\left(r_{t+1} - q_t(a)\right),$$

$q_{t+1}(a) = q_t(a)$ otherwise, and $q_0(a) \in \mathbb{R}$ is its initial estimated value.

Remark 1.3.3 *The introduced recurrence formula can store only two parameters: the current value of the action and the number of times this action is taken.*

Proof. When no action a is selected at the step t, the action-value estimate doesn't change. The case when the action a is selected at the step t is more informative, so we are going to consider it instead. By the definition,

$$q_t(a) = \frac{\sum\limits_{i=0}^{t-1} r_{i+1}\mathsf{I}(a_i = a)}{\sum\limits_{i=0}^{t-1} \mathsf{I}(a_i = a)} = \frac{\sum\limits_{i=0}^{t-1} r_{i+1}\mathsf{I}(a_i = a)}{n},$$

where the last equality is true due to the condition that the action a is selected n times before the step t. But then, $nq_t(a)$ is a sum of rewards for selecting the action a before the step $(t-1)$ inclusive. Since the action is also selected at the step t and since the reward r_{t+1} is obtained (and it is selected $(n+1)$ times), we can estimate its value as follows:

$$q_{t+1}(a) = \frac{1}{n+1}\left(r_{t+1} + nq_t(a)\right) = \frac{1}{n+1}\left(r_{t+1} + (n+1)q_t(a) - q_t(a)\right) =$$

$$= q_t(a) + \frac{1}{n+1}\left(r_{t+1} - q_t(a)\right).$$

□

Remark 1.3.4 *In addition to an arithmetic mean of the obtained rewards (in particular, to avoid changing the policy), the following rule of obtaining $q_{t+1}(a)$ is often used:*

$$q_{t+1}(a) = q_t(a) + \alpha_t\left(r_{t+1} - q_t(a)\right).$$

When $\alpha_t = \alpha$ is a constant, we get an exponential moving average:

$$q_{t+1}(a) = \alpha r_{t+1} + (1-\alpha)q_t(a).$$

The relation shows that when $\alpha \in (0, 1)$, the most valuable are fast rewards (the bird-in-the-hand principle), and when $\alpha > 1$, otherwise. As for now, we will not focus on this matter.

1.4 Greedy Policy

It's time to review different policies of selecting actions at the step t. Probably the first and most naive way of maximizing the reward is to select an action (or actions) at the step t with the highest estimated value. Let A_t be a set of actions in the state s_t with the highest (which also means equal) estimated value. As mathematicians, we say that the set A_t is a set

$$A_t = \operatorname*{Arg\,max}_{a \in A(s_t)} q_t(a).$$

Since we make the selection based on the action-value estimate and since all the actions of set A_t at the step t are equally valuable (others are not considered), any $a \in A_t$ is chosen equiprobably. Hence,

$$\pi_t(a|s_t) = \begin{cases} \frac{1}{|A_t|}, & a \in A_t \\ 0, & a \notin A_t \end{cases},$$

where $|A_t|$ is a number of elements in the set A_t. Thus, the actions with the highest estimated value are chosen with the probability $\frac{1}{|A_t|}$. Other actions are ignored.

Definition 1.4.1 *The introduced policy is called a greedy policy.*

We can also consider the opposite of a greedy policy. It's a so-called completely exploratory policy.

Definition 1.4.2 *If, as a result of the training, at each step t, an action of the set $A(s_t)$ of available actions in the state s_t is selected with equal probability, the policy is called completely exploratory.*

In terms of the trade-off between exploration and exploitation, the greedy policy is pure exploitation without an opportunity to explore. The completely exploratory policy is rather responsible for constant exploration. Let's consider an example of the greedy policy application.

Example 1.4.1 *The owner of a small shop wants to earn more money. The owner knows that changes in the store layout can increase impulse purchases. For example, if cookies are near tea, the probability of buying cookies increases, which is more profitable. The owner decides to try different configurations and observe the results.*

Well, let's see what is what. Let the bandit's lever be one of three configurations (a, b, c) and t be a number of the day when the experiment is conducted. Let the reward be the store's profit for 1 day. Let each game consist of 1000 *iterations (steps). Basically, it is the number of days when the experiment is conducted. When the configuration a, b, or c is chosen, the rewards are samples from statistical populations ξ_1, ξ_2, ξ_3 with the distributions $\mathsf{N}_{2,1}$, $\mathsf{N}_{2.5,1}$, and $\mathsf{N}_{3,1}$ respectively. Formally, the rewards in the considered example can be negative and interpreted as errors in the data or as business losses (lost or damaged goods).*

What does the agent do when the greedy policy is followed? Let $q_0(a) = q_0(b) = q_0(c) = 0$*. Since, at the time step* $t = 0$*, the estimated values of all actions (configurations) are the same, the choice of an action is made at random. Hence, any of three configurations is chosen equiprobably. The policy at step 0 is as follows:*

$$\pi_0(a|s_0) = \pi_0(b|s_0) = \pi_0(c|s_0) = \frac{1}{3}.$$

Let the configuration c be chosen ($a_0 = c$*). Thus, the estimated value of the configuration will become positive (it is assumed that the profit value is positive on the first day). According to the algorithm, the configuration c (*$a_1 = c$*) will be chosen in the first iteration, and so on. Therefore, the policy is as follows:*

$$\pi_t(c|s_t) = 1, \quad t \geq 1.$$

The example shows that (according to the greedy policy) the only configuration that will be (randomly) chosen is the one chosen at step 0.

Fig. 2 shows the results of three games. The blue curve corresponds to the case when the first selected action was $a_0 = c$*. As has been noted, it was also chosen at all subsequent steps (*$a_t = c$*,* $t \in \{0, 1, ..., 999\}$*). The graph shows the average reward or (what's the same) the estimated value of the action c at each step.*

In the second game, the first choice is the configuration a, which is $a_0 = a$ *(the red curve). The graph also shows the average reward when this configuration is chosen (*$a_t = a$*,* $t \in \{0, 1, ..., 999\}$*).*

In the third game, the first chosen action (as well as subsequent ones) was b. The values of the average reward are shown in orange. The highest average reward is reached only in one of three cases. Moreover, the minimum average reward is obtained in one of three cases.

The games in this example can be implemented in practice. For example, there are three chain stores in the same conditions. Each of them adheres to one configuration for 1000 *days and then estimates the average profit. Then, the estimates are compared and the conclusion is drawn to define the best configuration (if there is any).*

The choice of the initial configuration is a matter of chance in our case. That's why it's so interesting to find out how the algorithm behaves on average in terms of a policy. A series of games will put everything in its place. Fig. 2 illustrates in black the mean of the average reward in three games at each step. Fig. 3 shows the results of applying the greedy policy when conducting 1000 games, each having 1000 iterations. The black line indicates the mean of the average reward in all games. As to the layout example, the described approach can be implemented in the following way. 1000 chain stores are in the same conditions. Each store

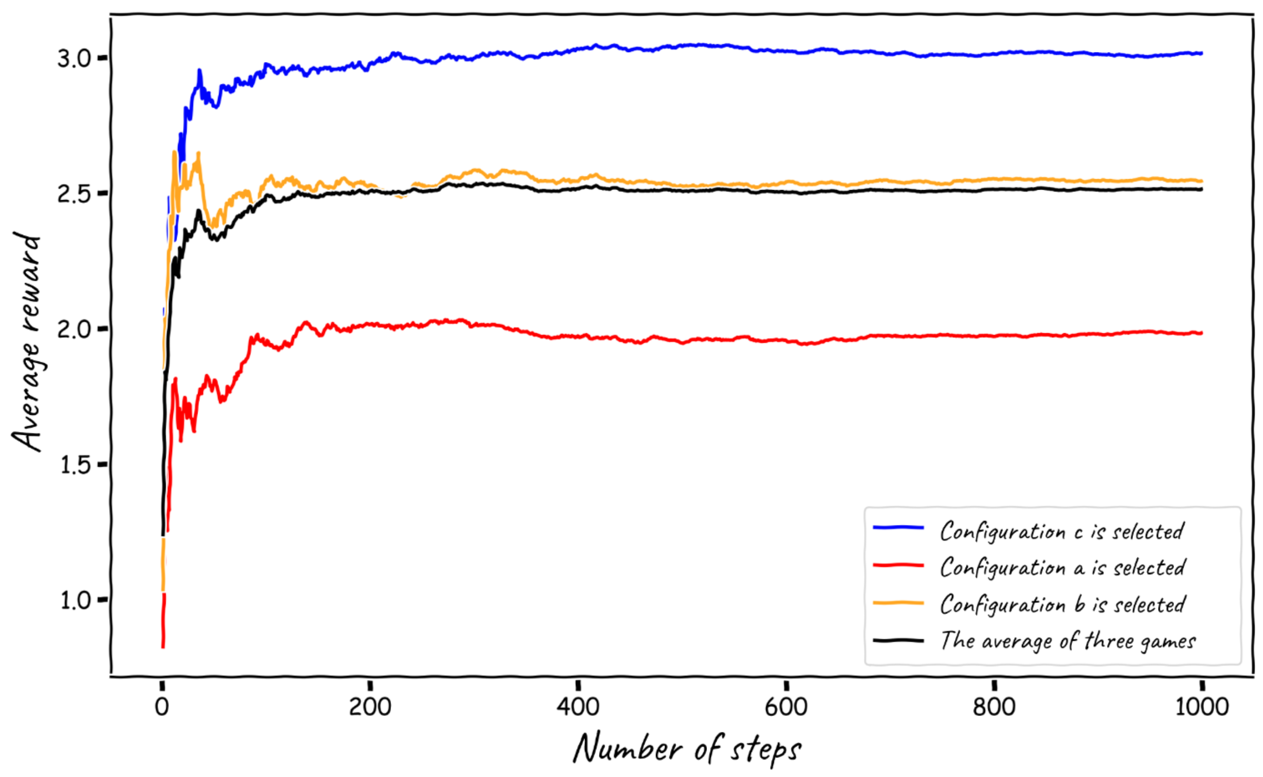


Figure 2: The average profit with respect to the chosen configuration.

is estimating one of three configurations for 1000 days. Under the greedy policy, each store generates a daily profit amounting to 2.5 units on average.

Remark 1.4.1 *Later, we will conduct* 1000 *games, each having* 1000 *iterations, to compare the policies. Since the obtained results are not due to the policy exploitation, but rather due to the averaging of 1-thousand policy exploitations, we will use the term algorithm (for example, a greedy algorithm).*

A major disadvantage of the greedy algorithm in the example is that the environment exploration stops at the very first step, which significantly affects the results. As has been well noted, Fig. 2 shows that the greedy principle in the store example maximizes average profit in one of three cases. Moreover, it also leads to the minimum average profit in one of three cases. The reason is a total ban on any exploration. A so-called ε-greedy policy can help to overcome this problem.

1.5 ε-Greedy Policy

By now you've learned that it is necessary to use a mix of completely exploratory and greedy policies to achieve a satisfactory result. The more the agent knows about the environment, the better it acts. At the same time, exploration for exploration is the problem of little interest. Thus, we cannot completely disable greed. The principle 'take what is given' is still relevant.

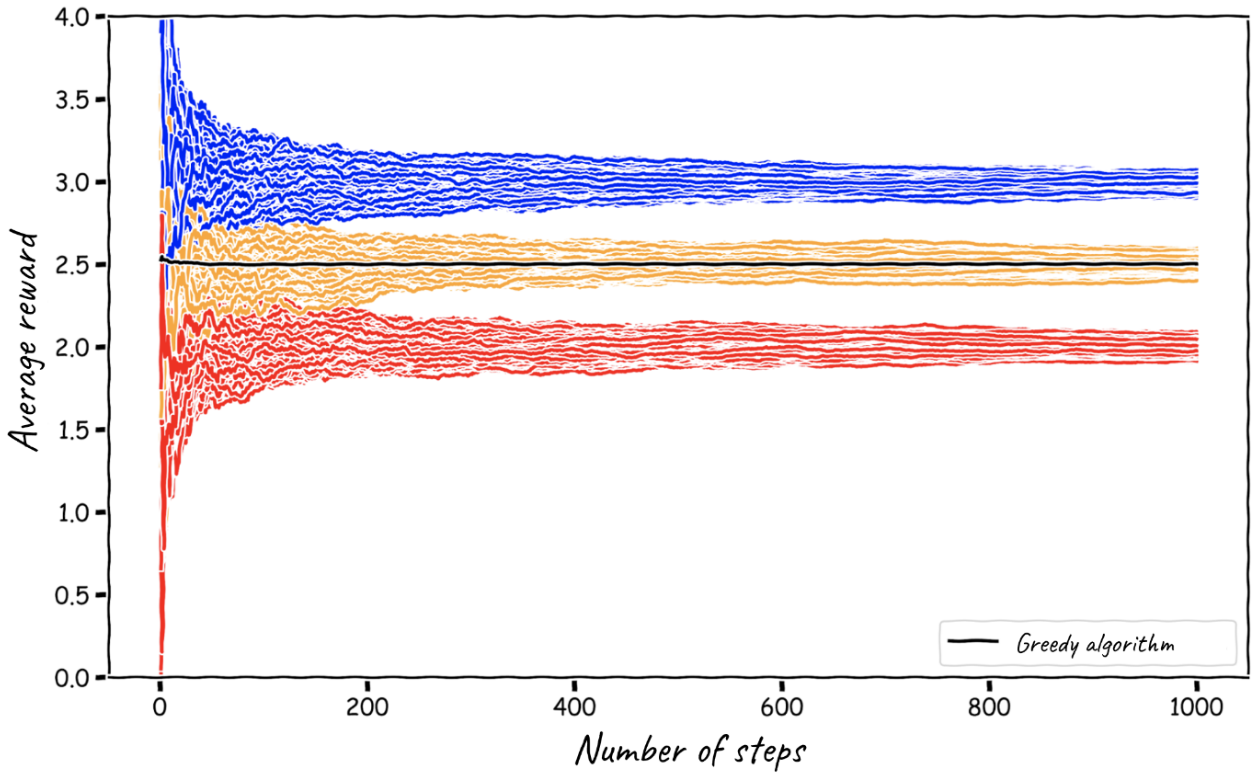


Figure 3: The illustration of the greedy algorithm.

Definition 1.5.1 *Let $0 < \varepsilon < 1$. The policy, $\pi_t(a|s_t)$, which, at the time step t, behaves as a greedy one with the probability $(1-\varepsilon)$ and as a completely exploratory with the probability ε, is called ε-greedy.*

Hence, the introduced policy is an attempt to ensure the trade-off between exploration and exploitation. Note that, when $\varepsilon = 0$ or $\varepsilon = 1$ (even though it is prohibited by the definition), ε-greedy policy becomes greedy and completely exploratory respectively.

The analytical expression for ε-greedy policy useful for practicing is as follows:

$$\pi_t(a|s_t) = \begin{cases} \dfrac{1-\varepsilon}{|A_t|} + \dfrac{\varepsilon}{|A(s_t)|}, & a \in A_t \\ \\ \dfrac{\varepsilon}{|A(s_t)|}, & a \notin A_t \end{cases},$$

where

$$A_t = \operatorname*{Arg\,max}_{a \in A(S_t)} q_t(a).$$

Remark 1.5.1 *Let's explain why the written formulas correspond to the definition. The first task is to find the probability of choosing the action $a \in A_t$. This* *action is chosen with the probability $\frac{1}{|A_t|}$ under the greedy policy and with the probability $\frac{1}{|A(s_t)|}$ under the completely exploratory policy. Since the greedy policy is chosen with the probability $(1-\varepsilon)$, and the completely exploratory policy is chosen with the probability ε, we obtain the following probability of choosing the action $a \in A_t$:*

$$\frac{1-\varepsilon}{|A_t|} + \frac{\varepsilon}{|A(s_t)|}.$$

Similarly, the action $a \notin A_t$ is chosen only under the completely exploratory policy. In this case, it is chosen with the probability $\frac{1}{|A(s_t)|}$. Since the completely exploratory policy is chosen with the probability ε, the probability of choosing the action $a \notin A_t$ equals

$$\frac{\varepsilon}{|A(s_t)|}.$$

Let's prove that the introduced definition is correct, or, put differently, that the introduced policy is a policy in the sense of the introduced definition.

Lemma 1.5.1 *ε-greedy policy is a policy.*

Proof. It's sufficient to prove that

$$\sum_{a \in A(s_t)} \pi_t(a|s_t) = 1.$$

It is true because

$$\sum_{a \in A(s_t)} \pi_t(a|s_t) = |A_t| \left(\frac{1-\varepsilon}{|A_t|} + \frac{\varepsilon}{|A(s_t)|} \right) + (|A(s_t)| - |A_t|) \left(\frac{\varepsilon}{|A(s_t)|} \right) = 1.$$

□

The greedy and ε-greedy algorithm in the layout example (1000 games, each having 1000 iterations) are compared in Fig. 4. Note that, for clarity, this and subsequent graphs will be constructed starting from the step $t = 1$. Note that the results differ when ε values vary. For example, the greater average reward is attained faster when $\varepsilon = 0.2$ (when the completely exploratory policy is given more freedom) rather than when $\varepsilon = 0.1$ and $\varepsilon = 0.01$. When the action values are well-studied (starting from step 300) and $\varepsilon = 0.2$, the algorithm is worse than the case when $\varepsilon = 0.1$ because the rate of its exploration actions is higher, and the actual optimal action (which estimated value is already quite accurate) is chosen less often. When $\varepsilon = 0.01$, more time is needed to estimate the value of all actions. The potential of this algorithm is reached when $10,000$ steps are taken instead of 1000 (Fig. 5).

When $\varepsilon = 0.01$, the average reward turns out to be greater than that of the competitors starting from step $t = 8,000$.

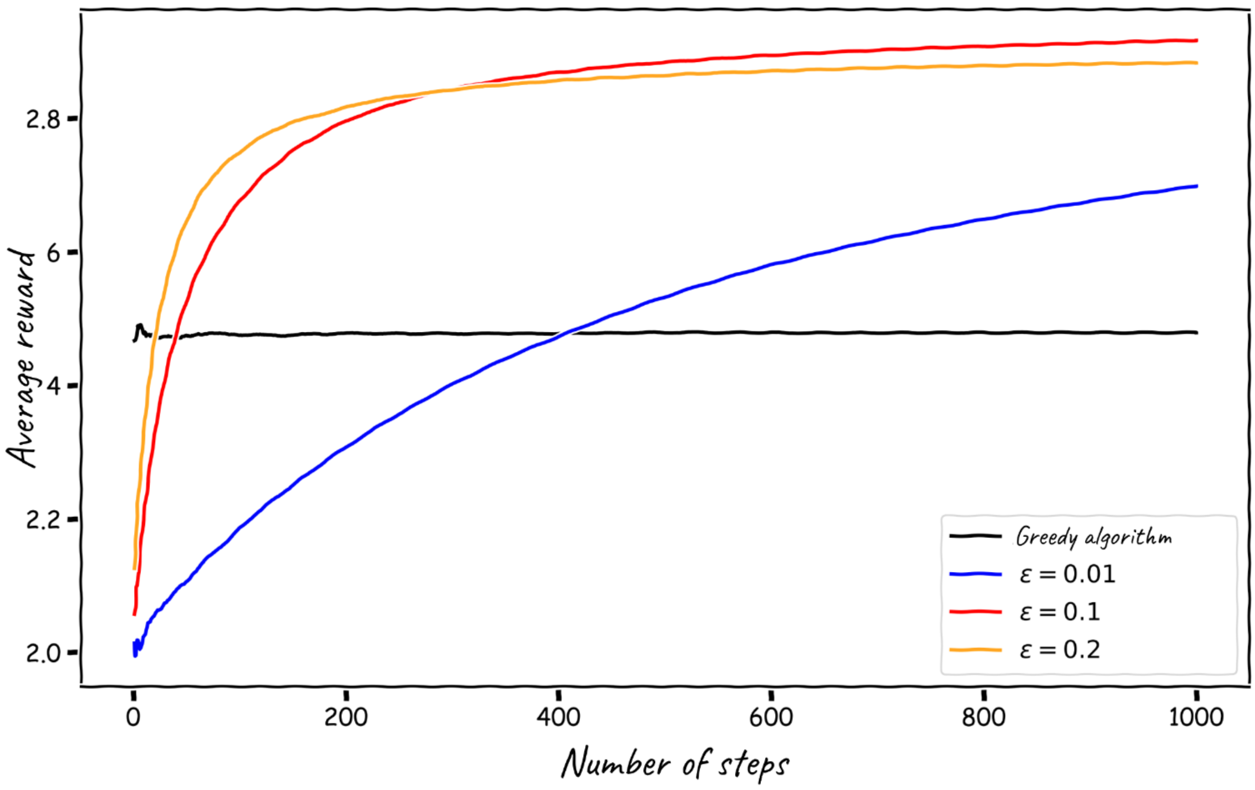


Figure 4: The comparison of the greedy algorithm and ε-greedy algorithms.

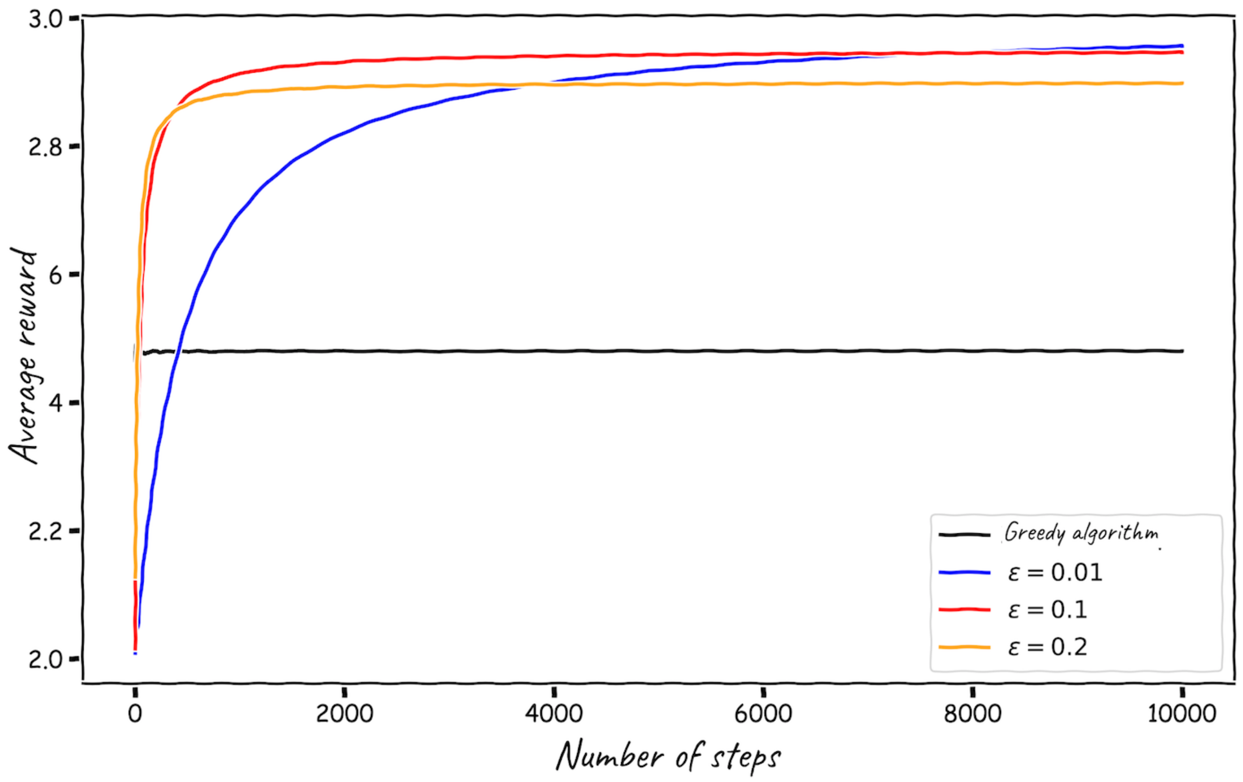


Figure 5: The comparison of the greedy algorithm and ε-greedy algorithms.

Remark 1.5.2 *Note that each of these ε-greedy algorithms is better than a greedy algorithm. Moreover, the values of the average reward in each considered case quickly converge to the average maximum, which is* 3*. When a number of steps is small, sudden changes in some graphs occur because of inaccurate data about the action-value estimate (not enough data is collected). To clarify it, actions are selected chaotically most of the time, which leads to the sudden changes in the average reward.*

Remark 1.5.3 *Finally, we would like to make a heuristic note. It makes sense to decrease the parameter ε over time (when t increases) because the action-value estimates become more accurate, and it's better to prioritize exploitation, rather than exploration. For example, we can consider the relationship between ε and time t as the function of the form:*

$$\varepsilon(t) = \frac{1}{1 + t\beta},$$

where t is a present moment, and $\beta \in (0, 1)$ is a coefficient of the rate of ε decrease. As β, we can take, for example, $\frac{1}{k}$, where k is a number of the bandit's arms. The choice of the function is up to a researcher.

1.6 Softmax Policy

A ε-greedy policy has a definite disadvantage. When exploration is ongoing, actions are chosen with equal probability, which negatively affects the goal of maximizing the sum of rewards (because actions with high and low estimated values have equal chances to be chosen). The principle behind a softmax policy is to reduce losses during the exploration in the iteration t by selecting actions a with a low estimated value $q_t(a)$ less often. Because of that a weight is calculated for each action. Then it is used to choose an action. More precisely, the policy is given by the following definition.

Definition 1.6.1 *The policy when the probability of choosing the action $a_t \in A(s_t)$ at the time step t equals*

$$\pi_t(a|s_t) = \frac{e^{q_t(a)/\tau}}{\sum\limits_{a \in A(s_t)} e^{q_t(a)/\tau}}, \quad \tau > 0,$$

is called a softmax policy.

The parameter $\tau > 0$, which is also called the temperature, is a model parameter. When values of τ are high, the policy becomes more exploratory, and when they are small, it approaches a greedy one. Fig. 6 (given $t \geq 1$) shows the comparison

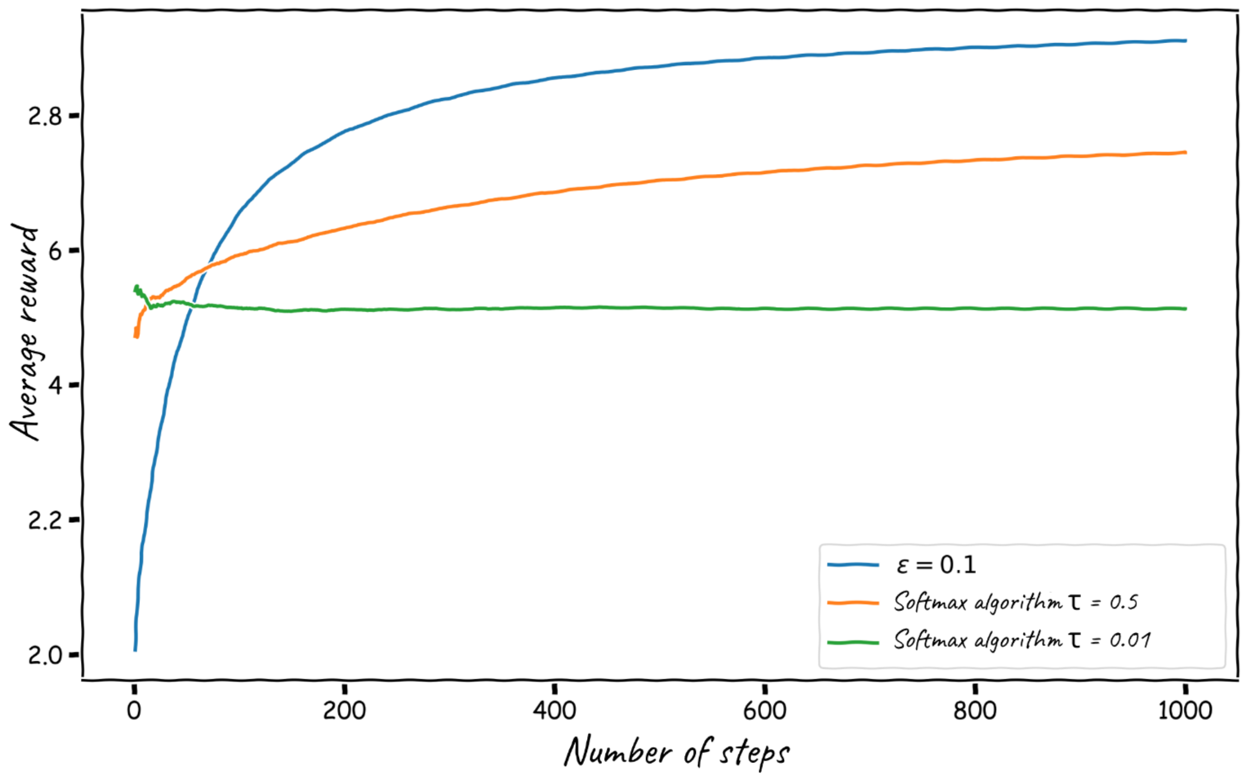


Figure 6: The comparison of the ε-greedy algorithm and softmax algorithm.

of a ε-greedy algorithm with parameter $\varepsilon = 0.1$ and a softmax algorithm, given different temperature τ. In our case, the ε-greedy algorithm shows the best result. As always, the choice of the method is made by a researcher. Note that when $\tau = 0.01$, the softmax algorithm is close to the greedy one.

Remark 1.6.1 *The introduced probability distribution is often called in physics a Gibbs distribution (or Boltzmann distribution). Note that as $\tau \to +\infty$, we get almost equal probabilities of selecting actions $a \in A(s_t)$. Thus, the policy approaches the completely exploratory policy. It approaches the greedy policy as $\tau \to 0+$ because the greatest weight among all exponents has that one for which $q_t(a)$ is greater.*

Remark 1.6.2 *As in the case of the ε-greedy policy, it makes sense to decrease the parameter τ over time. A researcher should decide how to do it (and whether it is necessary).*

1.7 Upper Confidence Bound (UCB)

Let's look at one more way of finding the trade-off between exploration and exploitation. It is called an Upper Confidence Bound (UCB) method. The idea is as follows. The less an action a is selected, the less accurate the estimate of its value (because the action is underexplored). To solve this problem, it's reasonable to consider a new estimated value of an action a as a sum of the current estimate of the value and its proximity to a true value (the right boundary of the confidence interval for the true value. Let's formally describe the method.

Let an agent be in a state s_t, $t \geq 0$. Let $N_0(a) = 0$, $N_t(a)$ be a number of times when an action a is selected before the step $(t-1)$ inclusive $(t \geq 1)$.

1. If there is $a \in A(s_t)$ so that $N_t(a) = 0$, then

$$A_t = \{a \in A(s_t) : \; N_t(a) = 0\}$$

and the policy of selecting $a_t \in A_t$ is as follows:

$$\pi_t(a|s_t) = \begin{cases} \frac{1}{|A_t|}, & a \in A_t \\ 0, & a \notin A_t \end{cases}.$$

2. Otherwise,

$$A_t = \operatorname*{Arg\,max}_{a \in A(S_t)} \left(q_t(a) + \delta \sqrt{\frac{\ln t}{N_t(a)}} \right), \quad \delta > 0,$$

and the policy of selecting $a_t \in A_t$ is as follows:

$$\pi_t(a|s_t) = \begin{cases} \frac{1}{|A_t|}, & a \in A_t \\ 0, & a \notin A_t \end{cases}.$$

To put it differently, the action that hasn't been chosen is selected next (when there are several such actions, one of them is selected at random). When there are no such actions, the choice is dependent on the estimated value of an action as well as the extent to which it is explored. The parameter $\delta > 0$ is responsible for the contribution of the latter. You can find detailed explanations in the additional materials.

Lemma 1.7.1 *Let $X_1, X_2, \ldots, X_n$ be a sample from the population ξ, given $X_i \in$* $[0, 1]$,

$$\overline{X} = \frac{1}{n} \sum_{i=1}^{n} X_i.$$

Therefore, the Hoeffding's inequality is true:

$$\mathsf{P}(\overline{X} - \mathsf{E}\xi \geq \varepsilon) \leq e^{-2n\varepsilon^2}.$$

Let $U \geq 0$. Having rewritten the inequality in the accepted notation and by the symmetry of the latter, we obtain

$$\mathsf{P}(\mathsf{E}\xi - q_t(a) \geq U) \leq e^{-2N_t(a)U^2}.$$

Or

$$\mathsf{P}(\mathsf{E}\xi \geq q_t(a) + U) \leq e^{-2N_t(a)U^2}.$$

Well, the probability that the true value is greater than the sum of the estimated value and U (the upper confidence bound (UCB)) is bounded from above. We designate by

$$\mathsf{p} = e^{-2N_t(a)U^2},$$

hence,

$$\ln \mathsf{p} = -2N_t(a)U^2 \Longrightarrow U = \sqrt{\frac{\ln \mathsf{p}}{-2N_t(a)}}.$$

p can be a function of t, and it makes sense that

$$\mathsf{p}(t) \xrightarrow[t\to+\infty]{} 0.$$

Thus,

$$\mathsf{p} = \frac{1}{t^{2\delta^2}},$$

we obtain

$$U = \delta\sqrt{\frac{\ln t}{N_t(a)}},$$

which is used in the proposed algorithm.

Remark 1.7.1 *According to the Hoeffding's inequality, all sample elements* $X_i \in [0, 1]$. *To apply the method correctly, it is recommended to perform normalization in advance.*

Fig. 7 shows the comparison of UCB with different parameters δ and ε-greedy algorithm. In the observed conditions, the greater δ, the earlier the algorithm will reach a performance plateau. When $\delta = 1.5$, the rate of exploration actions is lower than that when $\delta = 2$ and $\delta = 3$. That's why more time is needed for exploration, which, however, has a long-lasting positive impact.

Remark 1.7.2 *Note that the parameter* δ *needs to be decreased over time to shift from exploration to exploitation and increase the average reward.*

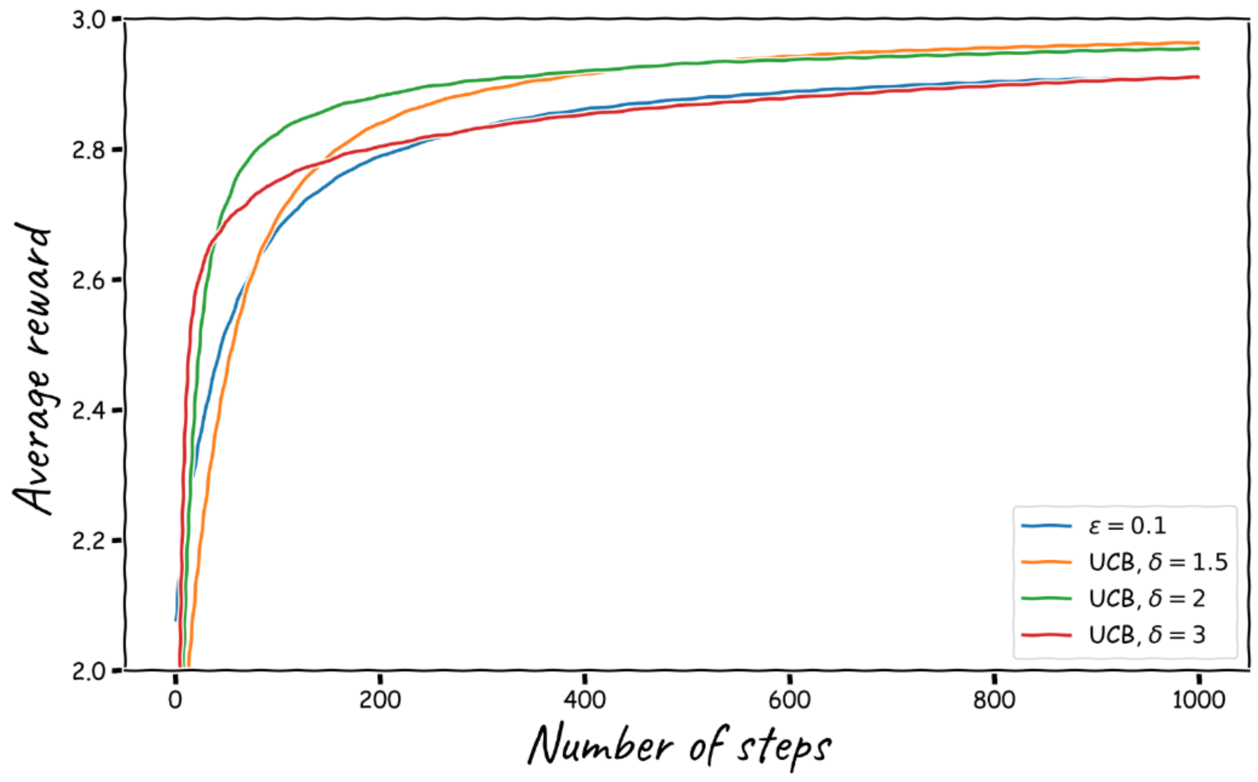


Figure 7: The comparison of the ε-greedy algorithm and UCB algorithms.

1.8 Optimistic Initial Estimates

The choice of actions was based on the estimate of action value in the considered cases. It was assumed that the initial estimates of all actions were equal because the agent had no information about the values or other actions in the beginning. When we use a ε-greedy algorithm, the action-value estimates become more accurate when actions are selected.

The method of optimistic initial estimates (or optimistic initial values) shares the same idea. The optimism comes from the fact that an action is set too high estimated value at the initialization. Let's explain it using the example.

Recall that reward values were generated from populations with the distributions $\mathsf{N}_{2,1}, \mathsf{N}_{2.5,1}, \mathsf{N}_{3,1}$. Hence, if we assume $q_0(a) = q_0(b) = q_0(c) = 10$ are initial values (which is definitely higher than 2, 2.5, and 3), we will consider these estimates overly optimistic because they will decrease during the model training. The choice of actions will be based on the greedy policy.

There are several steps. We will fill the estimated values in the table.

Step	$q_t(a)$	$q_t(b)$	$q_t(c)$
$t = 0$	10	10	10

Since the action-value estimates are the same, a random action (for example,

c) is selected. Let the reward be $r_1 = 2.8$, then

$$q_1(c) = q_0(c) + \frac{1}{1+1}\left(r_1 - q_0(c)\right) = 10 + \frac{(2.8 - 10)}{2} = 6.4.$$

We add the value into the table

Step	$q_t(a)$	$q_t(b)$	$q_t(c)$
$t = 0$	10	10	10
$t = 1$	10	10	6.4

According to the greedy algorithm, the agent will select one of the actions a or b, since their current estimated value is higher. Assume that an action a is chosen, and the reward $r_2 = 2.2$ is obtained. Then,

$$q_2(a) = q_1(a) + \frac{1}{1+1}\left(r_2 - q_1(a)\right) = 10 + \frac{(2.2 - 10)}{2} = 6.1.$$

We add the result to the table.

Step	$q_t(a)$	$q_t(b)$	$q_t(c)$
$t = 0$	10	10	10
$t = 1$	10	10	6.4
$t = 2$	6.1	10	6.4

It's probably obvious that the next will be the action b. Then the agent will switch between the actions, being disappointed with a gained reward. It will try all the actions several times to converge the estimates to the true values.

This optimism leads to exploration at first steps. When the estimated values are adjusted, it shifts to exploitation.

The greedy algorithm with optimistic initial estimates and previous algorithms (1000 games of 1000 steps given $t \geq 1$) are compared in Fig. 8.

You may notice that the optimistic algorithm cannot outperform ε-greedy algorithm with decreasing ε because it explores the environment. However, it performs better over time.

2 General Formulation of the Problem

2.1 Agent-Environment Interface

We described the interface between an agent and environment earlier when we were talking about k-armed bandits. The most important thing in the described example is a policy, according to which the agent selects an action at the time step t. We will take the time to explain the entire process in detail.

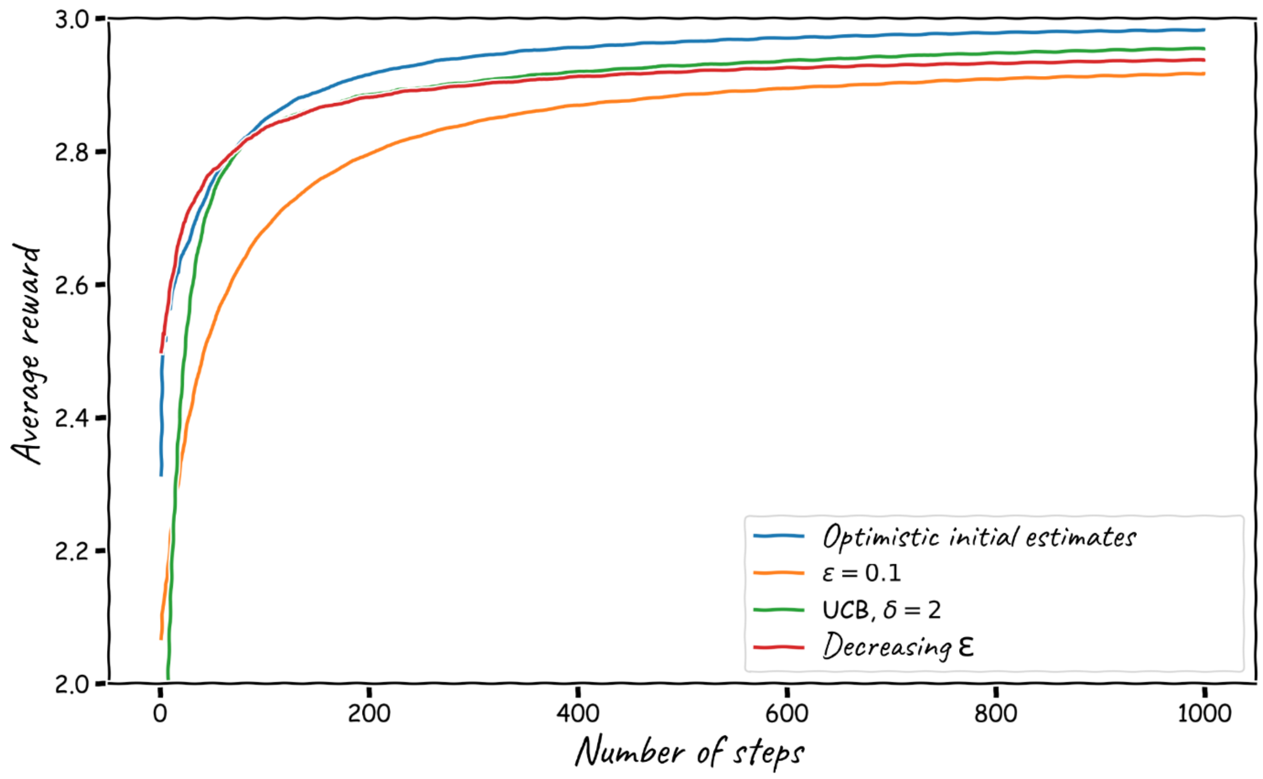


Figure 8: The comparison of different algorithms.

At every discrete time step $t \in \{0, 1, 2, ..., n\}$, $n \in \mathbb{N} \cup \{0, \infty\}$, the agent is in a state $s_t \in S$, where S is a set of all possible states. According to the policy $\pi_t(a|s_t)$, the agent selects one of possible actions $a_t \in A(s_t)$, where $A(s_t)$ is a set of actions available to the agent in a state s_t. If all actions are available to the agent at any step, we will write $a_t \in A$, where $A = \bigcup\limits_{s \in S} A(s)$ is a set of all possible actions. When an action is taken, the agent receives a reward $r_{t+1} \in \mathbb{R}$ and finds itself in a state s_{t+1}. We can consider the learning history as a sequence of three items (state, action, reward):

$$s_0, a_0, r_1,$$

$$s_1, a_1, r_2,$$

$$\dots$$

Remark 2.1.1 *Note that, since the actions are selected according to the policy (probabilistically rather than deterministically), the reward and subsequent agent state are also random in a sense. That's why we can say that the environment generates a reward r_{t+1} for the action a_t and the subsequent state s_{t+1} of the agent with respect to the sequence of previous states and chosen actions. Thus, s_t and r_t are random variables, and*

$$s_{t+1} \sim p(s|s_t, a_t, r_t, s_{t-1}, a_{t-1}, r_{t-1}, ..., s_0, a_0),$$

$$r_{t+1} \sim p(s|s_t, a_t, r_t, s_{t-1}, a_{t-1}, r_{t-1}, ..., s_0, a_0),$$

where p is a probabilistic distribution.

Hence, the three-tuple (an agent state, a policy of choosing the action, a reward for choosing the action) at each subsequent step depends on each previous step. Such a formulation is too complicated in terms of theory (multivariate joint distributions) and practice (storing large amounts of information). This time, we will keep it simple and think that each following step depends only on the current state, and the experience is not considered. Let's introduce the following definition.

Definition 2.1.1 *A Markov decision process is a four-tuple (S, A, R, P), where*

1. *S is a set of possible states of the agent.*
2. *A is a set of actions available to the agent.*
3. *$R : S \times A \to \mathbb{R}$ is a reward function (or an expected reward) when a transition from a state s to a state s' is made when the action a is selected.*
4. *$P : S \times A \to \Pi(S)$ is a state transition function, $\Pi(S)$ is a set of probability distributions over S,*

and the probabilities of subsequent transitions do not depend on the history of previous ones, that is,

$$s_{t+1} \sim \kappa(s_t, a_t) \in \Pi(S),$$

$$\mathsf{P}_\kappa(s_{t+1} = s'|s_t, a_t, r_t, s_{t-1}, a_{t-1}, r_{t-1}, ..., s_0, a_0) = \mathsf{P}_\kappa(s_{t+1} = s'|s_t, a_t).$$

So, $\kappa(s_t, a_t)$ is a probability distribution of transitioning from the state s_t to another state after taking the action a_t. The agent has no memory when the MDP is used because the probability of transitioning to the state s' at the step t depends only on the current state s_t and selected action a_t.

Remark 2.1.2 (!) *Here's an important point that we need to discuss before we explain the statements in the definition. Reinforcement learning is based on MDP, but only MDP cannot describe reinforcement learning. The main difference is that the agent in each state s selects an action $a \in A(s)$ with the probability $\pi(a|s)$ (acts under a policy). From now on, we will assume that the agent's policy is intrinsic to MDP without making a special note about it.*

Now we can move on to the statements in the definition. We are already familiar with the concepts of agent states and a set of available actions. R is a function that outputs the expected reward when the action $a \in A(s_t)$ is selected in the state s_t. In other words, when the input is a current state and selected action, the function R outputs a parameter, which is an expected reward for such a move. The function P outputs a probability distribution of state transition when the agent is in the state s_t and selects the action a_t.

Example 2.1.1 *Fig. 9 shows an example of MDP.*

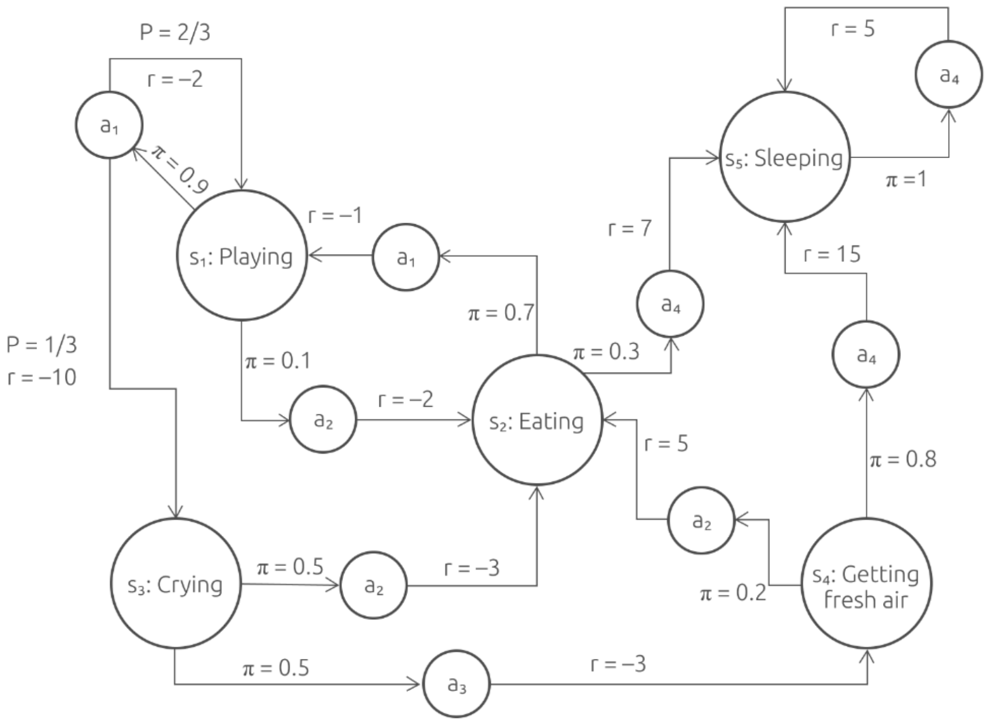


Figure 9: MDP.

In this case, the agent is a child who can be in one of five states (big circles):
s_1 : *Playing.*
s_2 : *Eating.*
s_3 : *Crying.*
s_4 : *Getting fresh air.*
s_5 : *Sleeping.*
Assume that the agent can select only from 4 *actions (small circles):*
a_1 : *Playing.*
a_2 : *Eating.*
a_3 : *Getting fresh air.*
a_4 : *Sleeping.*

How to read and understand the information in Fig. 9? The arrows coming out from the state circles point to the actions available in this state. (The values of π *near these arrows are dictated by the agent's policy. They show the probability of choosing the action to which the arrow points, being in the state from which the arrow originates.) Then, after selecting an action, we observe a probability* *distribution over S. The values of P near the arrows that originate from the action circles show the probability and state, to which the agent can transition, as well as the reward R that the agent can get. Thus, the agent transitions from the state s_1 by choosing the action a_1 (with the probability 0.9) to the state s_3 with the probability $1/3$ and gets the reward $r = -10$. The other variant is that the agent returns to the state s_1 with the probability $2/3$ and gets the reward -2. The probability distribution of transitioning from the state s_1 when choosing the action a_1 is a distribution $\kappa(s_1, a_1)$ from the definition of MDP. In our example, it is given by the table:*

	s_1	s_3
P	2/3	1/3

.

At the same time, the choice of the action a_2 (with the probability 0.1) in the same state leads to the transition (with the probability $P = 1$) to the state s_2 and the reward $r = -2$. Thus, the distribution $\kappa(s_1, a_2)$ is given by the table:

	s_2
P	1

.

And so on.

Sets of states S and actions A can be finite or infinite in a general case. However, both these sets are finite in this module. These processes even have a term.

Definition 2.1.2 *A Markov decision process, in which the sets A and S are finite, is called finite.*

Here's another important definition.

Definition 2.1.3 *If, at the step T, the environment transits to a state, when the interface between an agent and environment stops, this state is called terminal.*

For example, in tic-tac-toe (when the field is finite), the terminal state is reached when a player wins, or all cells are filled.

Example 2.1.2 *Let's slightly change the considered example. The state s_5 has changed. This state becomes terminal because it leads to no further actions. And now the parents can enjoy themselves.*

Now that the main definitions have been introduced, we need to decide what to estimate at the end of the game.

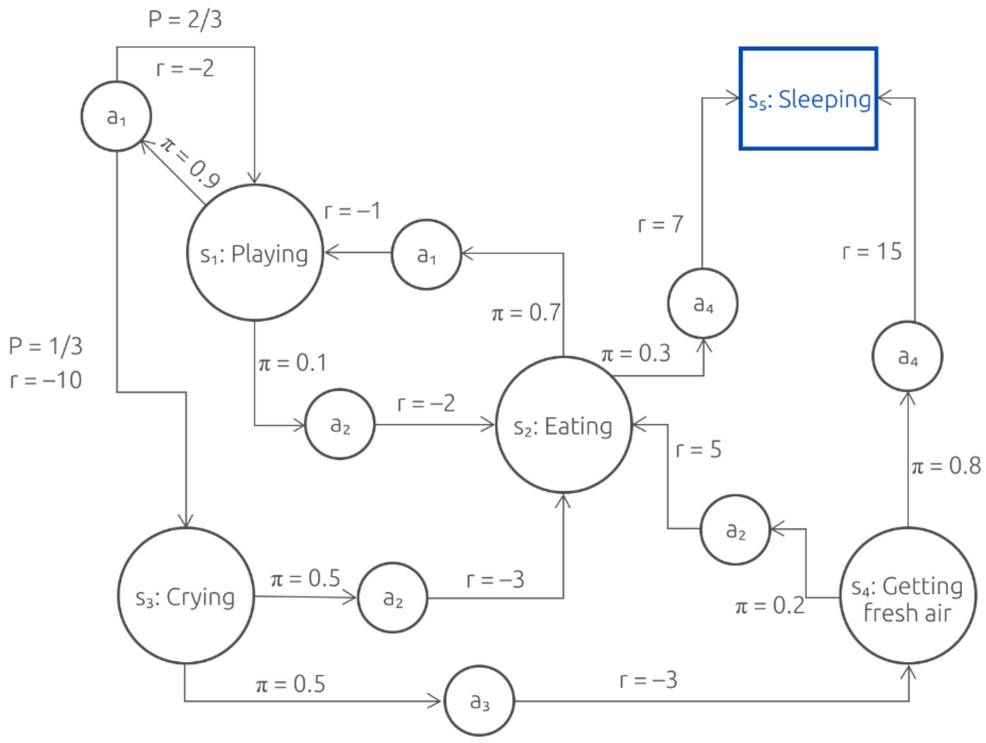


Figure 10: MDP with the terminal state.

Definition 2.1.4 *Suppose that we are at the step t, and the interface between an agent and environment assumes an infinite number of steps. Then, the expected return after the step t is a parameter:*

$$G_t = r_{t+1} + r_{t+2} + \ldots = \sum_{k=0}^{\infty} r_{t+k+1}$$

If we consider the interface that assumes reaching a terminal state in a finite number of steps $t \in \{0, 1, 2, ..., T\}$, then the expected return is a parameter:

$$G_t = r_{t+1} + r_{t+2} + \ldots + r_T.$$

Probably, the introduced definition is one of the most natural ones because the simplest thing is to sum up all the rewards that will be obtained (although we don't know which ones).

Remark 2.1.3 *Note that the infinite interface between the agent and environment ($T = +\infty$) is not an outlier, but a common case. Moreover, the so-called lifespan of the agent is not defined most of the time. For example, it's possible to find a way out of a maze in a finite number of moves (unknown (!) in advance), and it's also possible to keep moving (if the agent is lost).*

At the same time, the maximization of the cumulative reward is a naive thing. For example, a player may collect items by moving slowly on the way out of the maze, although it may seem obvious that the best thing to do is to get out as fast as possible based on the principle the sooner reward, the better. Hence, the following concept seems more logical.

Definition 2.1.5 *Suppose that we are at the step t, and the interface between an agent and environment assumes an infinite number of steps. Thus, the expected discounted return is as follows:*

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

where the parameter $\gamma \in [0, 1]$ is called the discount rate.

If we consider the interface that assumes reaching a terminal state in a finite number of steps $t \in \{0, 1, 2, ..., T\}$, then the expected discounted return is a parameter:

$$G_t = r_{t+1} + \gamma r_{t+2} + ... + \gamma^{T-t-1} r_T.$$

We can also call the expected discounted return the principle of reward depreciation. According to the described approach, the agent is trying to select actions in such a way that maximizes the sum of the obtained depreciated rewards. Thus, at each step, the agent is trying to select such an action a_t so that the largest possible reward doesn't depreciate too much (the latter happens due to slow reaction).

We can also describe it as follows. The discount rate γ determines how valuable future rewards are by adjusting the degree of the agent's short-sightedness. When $\gamma = 0$, the agent doesn't care about future rewards while striving to maximize the current one. In the long term, it can reduce the expected return. As γ approaches one, future rewards become more valuable.

Remark 2.1.4 *Note that the expected return in case of a finite or infinite number of steps is obtained from the discounted return when $\gamma = 1$.*

The following lemma is true.

Lemma 2.1.1 *If $\gamma \in (0, 1)$ and a sequence $r_{t+1}, r_{t+2}, r_{t+3}, \ldots$ is finite, then the discounted return*

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

has a finite value.

Proof. Since the sequence $\{r_{t+k+1}\}_{k=0}^{\infty}$ is finite, then $|r_{t+k+1}| \leq C$. Then,

$$\left|\gamma^k r_{t+k+1}\right| \leq C\gamma^k.$$

Since the series with the common element γ^k given $\gamma \in (0, 1)$ is convergent, the initial series is absolutely convergent, thus, it is convergent. $\square$

2.2 Action-Value Function and State-Value Function

To make the agent understand what to do while interacting with the environment, and what policy to apply, we need an apparatus that can describe the agent states. Reinforcement learning uses the approach when each state is assigned a numeric value that reflects the value of this state. The state value characterizes this state in terms of the discounted return.

Definition 2.2.1 *Let π be the agent's policy. The function of the value of a state s under a policy π is the function of the form*

$$v_\pi(s) = \mathsf{E}_\pi\left(G_t | s_t = s\right) = \mathsf{E}_\pi\left(\sum_{k=0}^{\infty}\gamma^k r_{t+k+1} | s_t = s\right), \quad s \in S.$$

The state-value function defines the average discounted return when starting from a state s and following a policy π. Based on the value of each state, it's logical to follow such a policy that leads to the state of the maximum value.

Remark 2.2.1 *Formally (in the definition of the value function), it's a conditional expected value, or the function of the state $s \in S$. When the state s changes, the policy $\pi(a|s)$ changes, which results in the change in $v_\pi(s)$.*

We can approach the estimated values more thoroughly by estimating the discounted return starting in a state s and selecting an action a. This leads us to the concept of the value of an action.

Definition 2.2.2 *Let π be the agent's policy. The action-value function of an action a in a state s under a policy π is a function*

$$q_\pi(s,a) = \mathsf{E}_\pi\left(G_t | s_t = s, a_t = a\right) = \mathsf{E}_\pi\left(\sum_{k=0}^{\infty}\gamma^k r_{t+k+1} | s_t = s, a_t = a\right).$$

Note that the action-value function is like the state-value function. The difference is that we define the discounted return when we start in a state s, select an action a, and follow the policy π.

It seems logical. Moreover, if we knew all about the environment and agent, it would be enough to select an action a that maximizes $q(s,a)$. Of course, it's impossible, but it turns out that value functions satisfy recursive relationships.

Lemma 2.2.1 *Let $v_\pi(s)$ and $q_\pi(s,a)$ be a state-value and action-value function respectively. All information about the environment is available, including the probabilities of all transitions from s to s' when taking an action a*

$$\mathcal{P}_{ss'}^a = \mathsf{P}(s_{t+1} = s' | s_t = s, a_t = a)$$

and all the expected rewards

$$\mathcal{R}^a_{ss'} = \mathsf{E}_\pi \left(r_{t+1} | s_t = s, a_t = a, s_{t+1} = s'\right).$$

Therefore, the following equalities are true:

$$v_\pi(s) = \sum_{a \in A(s)} \pi(a|s) \sum_{s' \in S} \mathcal{P}^a_{ss'} \left(\mathcal{R}^a_{ss'} + \gamma v_\pi\left(s'\right)\right)$$

and

$$q_\pi(s,a) = \sum_{s' \in S} \mathcal{P}^a_{ss'} \left(\mathcal{R}^a_{ss'} + \gamma \sum_{a' \in A(s')} \pi(a'|s') q_\pi(s', a')\right) =$$

$$= \sum_{s' \in S} \mathcal{P}^a_{ss'} \left(\mathcal{R}^a_{ss'} + \gamma v_{pi}(s')\right)$$

Proof. 1. Let's prove the first equality. By the definition, using the linearity property of the expected value, we obtain

$$v_\pi(s) = \mathsf{E}_\pi \left(G_t | s_t = s\right) = \mathsf{E}_\pi \left(\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s\right) =$$

$$= \mathsf{E}_\pi \left(r_{t+1} | s_t = s\right) + \gamma \mathsf{E}_\pi \left(\sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_t = s\right).$$

Let's consider a pair of obtained terms one by one and begin with the first one. In a state s, each action is selected with the probability $\pi(a|s)$, the transition to the state s' is made with the probability $\mathcal{P}^a_{ss'}$, and the expected reward $\mathcal{R}^a_{ss'}$ is gained. Then,

$$\mathsf{E}_\pi \left(r_{t+1} | s_t = s\right) = \sum_{a \in A(s)} \sum_{s' \in S} \pi(a|s) \mathcal{R}^a_{ss'} \mathcal{P}^a_{ss'}.$$

Similarly, to substitute $s_t = s$ for $s_{t+1} = s'$, we obtain

$$\mathsf{E}_\pi \left(\sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_t = s\right) = \sum_{a \in A(s)} \sum_{s' \in S} \pi(a|s) \mathsf{E}_\pi \left(\sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_{t+1} = s'\right) \mathcal{P}^a_{ss'}.$$

After summing up and regrouping, we obtain:

$$v_\pi(s) = \sum_{a \in A(s)} \pi(a|s) \sum_{s' \in S} \mathcal{P}^a_{ss'} \left(\mathcal{R}^a_{ss'} + \gamma \mathsf{E}_\pi \left(\sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_{t+1} = s'\right)\right).$$

Since

$$v_\pi(s') = \mathsf{E}_\pi \left(\sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_{t+1} = s' \right),$$

we obtain

$$v_\pi(s) = \sum_{a \in A(s)} \pi(a|s) \sum_{s' \in S} \mathcal{P}^a_{ss'} \left(\mathcal{R}^a_{ss'} + \gamma v_\pi(s') \right).$$

2. We can similarly prove the relation for $q_\pi(s, a)$. By the definition, using the linearity property of the expected value,

$$q_\pi(s,a) = \mathsf{E}_\pi(G_t | s_t = s, a_t = a) = \mathsf{E}_\pi \left(\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a \right) =$$

$$= \mathsf{E}_\pi (r_{t+1} | s_t = s, a_t = a) + \gamma \mathsf{E}_\pi \left(\sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_t = s, a_t = a \right).$$

Since the action a is selected, then, firstly,

$$\mathsf{E}_\pi (r_{t+1} | s_t = s, a_t = a) = \sum_{s' \in S} \mathcal{P}^a_{ss'} \mathcal{R}^a_{ss'},$$

and secondly,

$$\mathsf{E}_\pi \left(\sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_t = s, a_t = a \right) = \sum_{s' \in S} \mathcal{P}^a_{ss'} \mathsf{E}_\pi \left(\sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_{t+1} = s', a_t = a \right) =$$

$$= \sum_{s' \in S} \mathcal{P}^a_{ss'} q_\pi(s', a) = \sum_{s' \in S} \mathcal{P}^a_{ss'} \sum_{a' \in A(s')} \pi(a'|s') q_\pi(s', a').$$

Therefore,

$$q_\pi(s,a) = \sum_{s' \in S} \mathcal{P}^a_{ss'} \left(\mathcal{R}^a_{ss'} + \gamma \sum_{a' \in A(s')} \pi(a'|s') q_\pi(s', a') \right) = \sum_{s' \in S} \mathcal{P}^a_{ss'} \left(\mathcal{R}^a_{ss'} + \gamma v(s') \right).$$

□

To obtain the state-value and action-value functions, we need a system of $|S|$ equations with unknown $|S|$ in each case.

Remark 2.2.2 *The obtained equations are called Bellman equations for value functions under the policy π. For a state-value function, the Bellman equation expresses a relationship between the value of a state and the values of its subsequent states. For an action-value function, the Bellman equation expresses a relationship between the value of a pair (state, action) and the values of subsequent pairs.*

The idea (as well as the application) of the equations is shown in Fig. 11. Assume that the values of all states are known. Then, we consider the state s_1 as a current state in this case. Hence, using the relation

$$v_\pi(s) = \sum_{a \in A(s)} \pi(a|s) \sum_{s' \in S} \mathcal{P}_{ss'}^a \left(\mathcal{R}_{ss'}^a + \gamma v_\pi\left(s'\right)\right)$$

when $\gamma = 0.8$, we obtain

$$-2.23 = 0.9 \left(\frac{1}{3} (-10 + 0.8 \cdot 11.45) + \frac{2}{3} (-2 + 0.8 \cdot (-2.23)) \right) +$$

$$+0.1 \cdot 1 \cdot (-2 + 0.8 \cdot 6.15),$$

which is a true equality. Of course, the main question is how to obtain the state values. Let's look at an example.

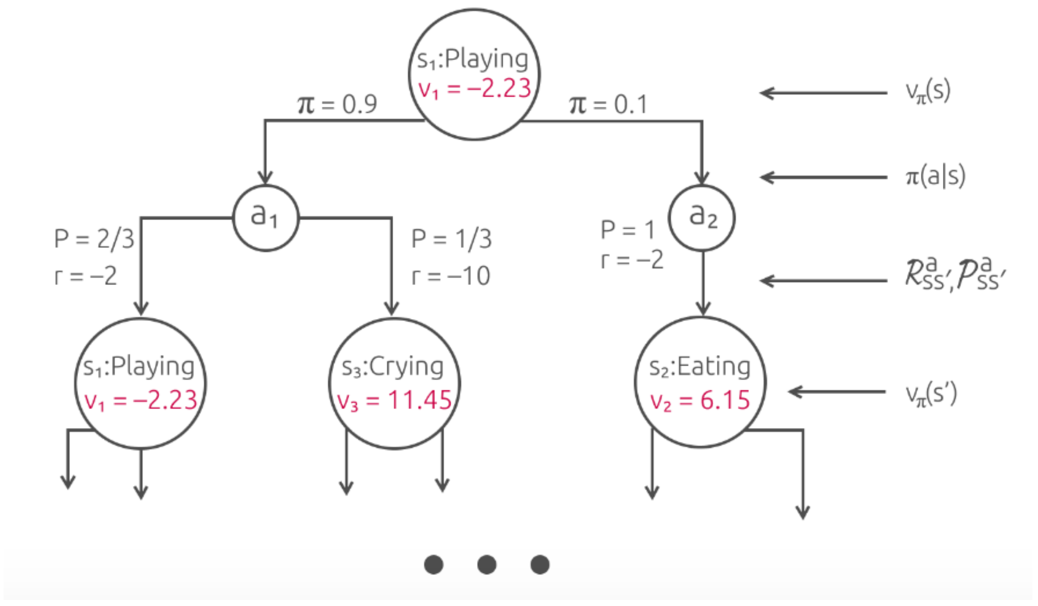


Figure 11: MDP.

2.3 Example

Earlier, we showed the relationship between the value of the current state s and subsequent states s' that the agent can transition. We can consider the example of getting all states using the Bellman equations and a concrete policy π. The input data from the example is shown in Fig. 12. Let $\gamma = 0.8$ be the discount rate. Considering that

$$v_\pi(s) = \sum_{a \in A(s)} \pi(a|s) \sum_{s' \in S} \mathcal{P}_{ss'}^a \left(\mathcal{R}_{ss'}^a + \gamma v_\pi\left(s'\right)\right),$$

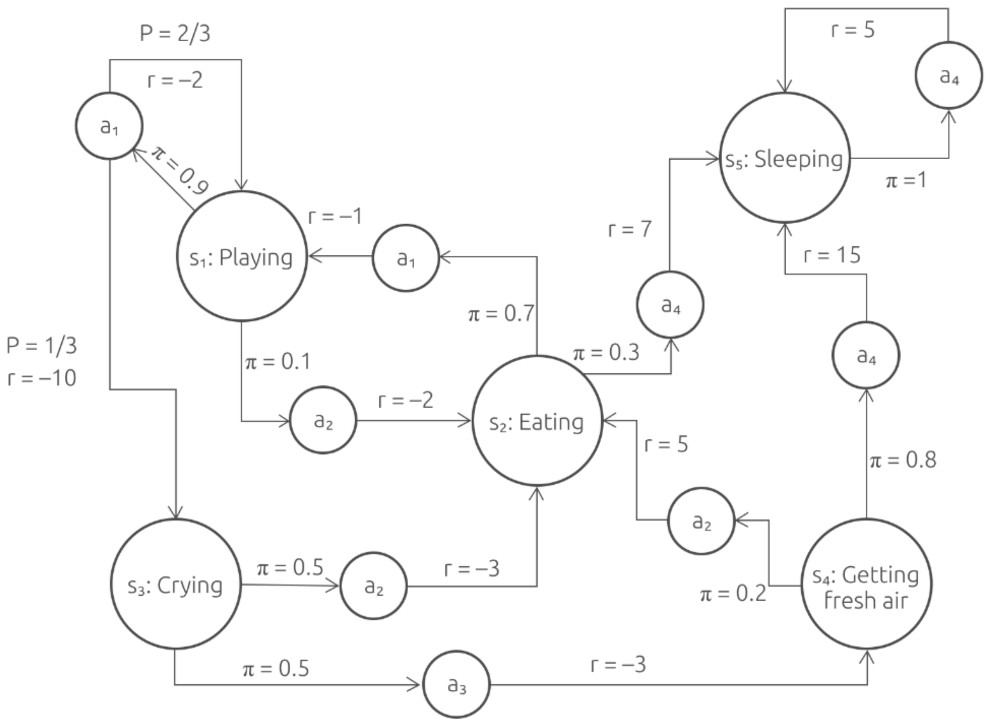


Figure 12: The input data.

we write the Bellman equation for the state Playing:

$$v_\pi(s_1) = 0.9\left(\frac{1}{3}\left(-10 + 0.8v_\pi(s_3)\right) + \frac{2}{3}\left(-2 + 0.8v_\pi(s_1)\right)\right) +$$

$$+0.1 \cdot 1 \cdot (-2 + 0.8v_\pi(s_2)).$$

Since the transition to the next state is deterministic (has the probability 1) in all states different from s_1, then the inner sum by s' consists of only one term (multiplied by 1). Based on this observation, we will write the remaining equations:

$$v_\pi(s_2) = 0.7(-1 + 0.8v_\pi(s_1)) + 0.3(7 + 0.8v_\pi(s_5)),$$

$$v_\pi(s_3) = 0.5(-3 + 0.8v_\pi(s_2)) + 0.5(-3 + 0.8v_\pi(s_4)),$$

$$v_\pi(s_4) = 0.2(5 + 0.8v_\pi(s_2)) + 0.8(15 + 0.8v_\pi(s_5)),$$

$$v_\pi(s_5) = 5 + 0.8v_\pi(s_5).$$

We solve the system and obtain the following state values (rounded to the nearest hundredth). The results are also shown in Fig. 13:

$$\begin{cases} v_\pi(s_1) = -2.23 \\ v_\pi(s_2) = 6.15 \\ v_\pi(s_3) = 11.45 \\ v_\pi(s_4) = 29.98 \\ v_\pi(s_5) = 25. \end{cases}$$

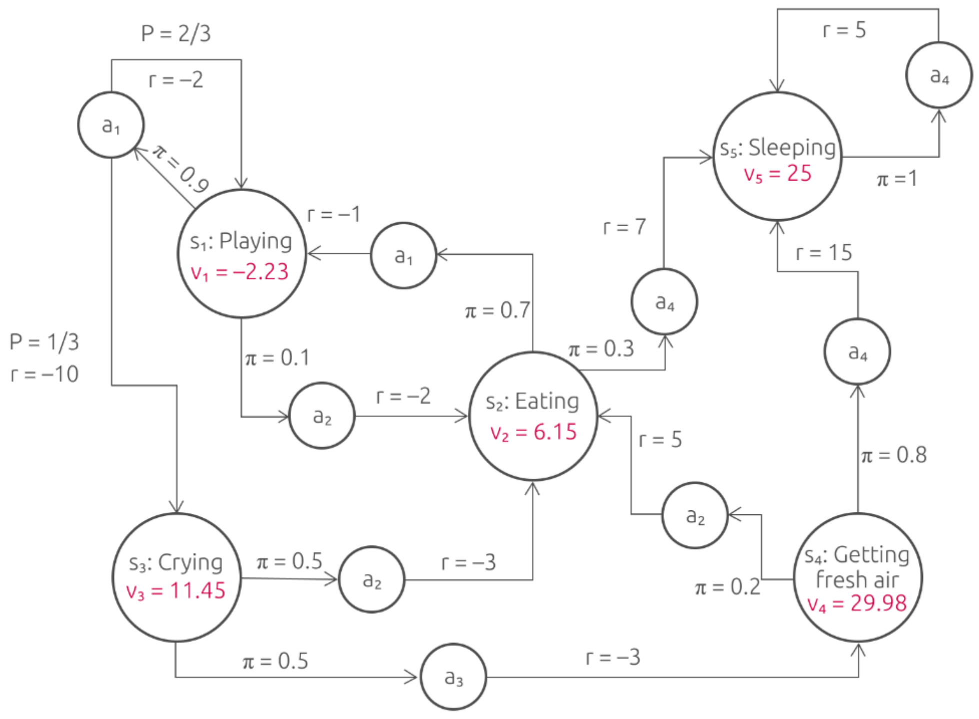


Figure 13: The state values under the selected policy.

Remark 2.3.1 *We observe that the state values directly depend on the policy selected by the agent. If the policy changes, the state values will change too. For example, if* $\pi(a_1|s_1) = 0.5, \pi(a_2|s_1) = 0.5$ *and the remaining are the same, we obtain the following state values (rounded to the nearest hundredth):*

$$\begin{cases} v_\pi(s_1) = 2.59 \\ v_\pi(s_2) = 8.85 \\ v_\pi(s_3) = 12.71 \\ v_\pi(s_4) = 30.42 \\ v_\pi(s_5) = 25. \end{cases}$$

Note that in this case, the values of all states are not less than those under the previous policy. We will discuss this case in detail a little bit later.

2.4 Policy Evaluation

In the considered example, we used the Bellman equations to find the state values. It's easy to make and solve a system of linear equations when there are five states. However, the number of possible states of the agent can be very high in practice. As has been well noted, when the number of possible states of the agent equals n, we need to solve a system of n equations with unknown n. Is there a way to simplify calculations? It turns out there is.

2.4.1 About Matrix Equations and Normalized Spaces

We can write the solution to the system of Bellman equations in an explicit form. Let's rewrite the system of equations

$$v_\pi(s) = \sum_{a\in A(s)} \pi(a|s) \sum_{s'\in S} \mathcal{P}^a_{ss'} \left(\mathcal{R}^a_{ss'} + \gamma v_\pi(s')\right), \quad s \in S,$$

in matrix notation:

1. Let v_π be a column vector of the height $|S|$, jth element of which equals $v_\pi(s_j)$, $j \in \{1, 2, ..., |S|\}$.
2. Let r_π be a column vector of the height $|S|$, jth element of which equals $\mathsf{E}_\pi(r_{t+1}|s_t = s_j)$, $j \in \{1, 2, ..., |S|\}$.
3. Let T_π be a matrix of size $|S| \times |S|$ with the elements

$$(T_\pi)_{ij} = \mathsf{P}_\pi(s_{t+1} = s_j | s_t = s_i) = \sum_{a\in A(s_i)} \pi(a|s_i)\mathcal{P}^a_{s_i s_j}.$$

In the accepted notation, the system of Bellman equations is rewritten as

$$v_\pi = r_\pi + \gamma T_\pi v_\pi.$$

Hence,

$$(I - \gamma T_\pi)v_\pi = r_\pi \;\Rightarrow\; v_\pi = (I - \gamma T_\pi)^{-1} r_\pi.$$

To study the expression, let's recall the concept of a normalized space.

Definition 2.4.1 (Normalized space) *Let X be a linear space. A norm is a function $\|\cdot\| : X \to \mathbb{R}$ that satisfies the following axioms:*

1. $\forall x \in X \Rightarrow \|x\| \geq 0$, and $\|x\| = 0 \Leftrightarrow x = 0$.

2. $\|\lambda x\| = |\lambda|\|x\|$, $\lambda \in \mathbb{R}$, $x \in X$.

3. $\|x+y\| \leq \|x\| + \|y\|$, $x, y \in X$.

Remark 2.4.1 *In a general case, a norm generalizes the concept of length. If $X = \mathbb{R}$, we can assume $\|x\| = |x|$. For example, if $X = \mathbb{R}^n$, we can consider the vector length as the norm $x = (x_1, x_2, ..., x_n) \in \mathbb{R}^n$:*

$$\|x\| = \sqrt{x_1^2 + x_2^2 + ... + x_n^2}.$$

Remark 2.4.2 *In $\mathbb{R}^n$, we can also consider other norms, for example:*

$$\|x\|_p = \sqrt[p]{|x_1|^p + |x_2|^p + ... + |x_n|^p}, \ p \in \mathbb{N}.$$

The vector length is a special case of the written norm when $p = 2$. Further on, we will use the norm when $p = +\infty$:

$$\|x\|_\infty = \max_{i \in \{1,...,n\}} |x_i|.$$

Now let's move on to the concept of the operator norm.

Definition 2.4.2 *Let X be a linear normalized space and $A : X \to X$ be a linear operator. The operator norm A is a value*

$$\|A\| = \sup_{x\colon \|x\|=1} \|Ax\|.$$

In a sense, the operator norm is the largest value by which the operator A stretches a unit vector. Note that the operator norm in a finite-dimensional space is finite.

Remark 2.4.3 *The next inequality directly follows from the definition of the norm:*

$$\|Ax\| \leq \|A\|\|x\|, \ \forall x \in X.$$

Remark 2.4.4 *It is well known that a linear operator $A : X \to X$ can be written as a matrix (a basis is fixed by X). The sum and the product of operators correspond to the sum and the product of the corresponding matrices, and the inverse operator to the inverse matrix. Thus, the operator inversion is equivalent to the inversion of the operator matrix in some basis.*

Example 2.4.1 *Let's find the operator norm $A : \mathbb{R}^n \to \mathbb{R}^n$ when the norm $\|x\|_\infty$. Let the matrix A with elements a_{ij}, $i, j \in \{1, 2, ..., n\}$, represent an operator in a fixed basis. Then,*

$$\|Ax\|_\infty = \max_{i \in \{1,...,n\}} \left| \sum_{j=1}^{n} a_{ij} x_j \right| \leq \max_{i \in \{1,...,n\}} \sum_{j=1}^{n} |a_{ij}| \, \|x\|_\infty.$$

Hence, when $x \neq 0$, we designate $y = \frac{x}{\|x\|_\infty}$

$$\|Ay\| \leq \max_{i\in\{1,\dots,n\}} \sum_{j=1}^{n} |a_{ij}|\,, \ \ \|y\|_\infty = 1.$$

If we prove that the written estimate is attained, we will also prove that

$$\|A\| = \max_{i\in\{1,\dots,n\}} \sum_{j=1}^{n} |a_{ij}|\,.$$

Let at least one matrix element not be equal to zero (otherwise, its norm equals 0, and the equality is attained). Then let

$$i_0 = \arg\max_{i\in\{1,\dots,n\}} \sum_{j=1}^{n} |a_{ij}|$$

be some i, at which the maximum is attained. Thus, we take a vector $x = (\operatorname{sign} a_{i_0 1}, \operatorname{sign} a_{i_0 2}, ..., \operatorname{sign} a_{i_0 n})$ as x. Clearly, $\|x\|_\infty = 1$ and

$$Ax = \sum_{j=1}^{n} a_{i_0 j} \operatorname{sign} a_{i_0 j} = \sum_{j=1}^{n} |a_{i_0 j}| = \max_{i\in\{1,\dots,n\}} \sum_{j=1}^{n} |a_{ij}|\,.$$

Lemma 2.4.1 (About the operator inversion) *A necessary and sufficient condition for a linear operator $A : X \to X$ (where X is a normalized space with the norm $\|\cdot\|$) to be invertible is that there exists such an $m > 0$ that*

$$\forall x \in X \ \Rightarrow \ \|Ax\| \geq m\|x\|.$$

Proof. Let's prove the sufficiency. The inequality means that when $x_1 \neq 0 \ \Rightarrow Ax_1 \neq 0$. In the case of a linear operator, it means the inversion.

We can also prove the necessity. Let the operator be inverse, then $y = Ax$, $x = A^{-1}y$ and

$$\|A^{-1}y\| \leq \|A^{-1}\|\|y\| \Rightarrow \|x\| \leq \|A^{-1}\|\|Ax\| \Rightarrow \|Ax\| \geq \|A^{-1}\|^{-1}\|x\|$$

and $m = \|A^{-1}\|^{-1}$. □

Lemma 2.4.2 (About the existence of a solution to the system) *Let $\gamma \in (0, 1)$. The matrix $(I - \gamma T_\pi)$ has an inverse. Thus, a solution to the system of Bellman equations exists and it's unique. Moreover,*

$$v_\pi = (I - \gamma T_\pi)^{-1} r_\pi.$$

Proof. The matrix inversion is equivalent to the inversion of the operator in a fixed basis of the matrix $(I - \gamma T_\pi)$. Since $\|I\| = 1$ and $\|T_\pi\| = 1$ (because we are dealing with the space $R^{|S|}$ with the norm $\|x\|_\infty$), then

$$\|(I - \gamma T_\pi)x\| \geq |\|Ix\| - \gamma\|T_\pi x\|| \geq (1-\gamma)\|x\|.$$

Thus, the operator has an inverse, and the lemma is proved. □
Now we can introduce the next important definition.

Definition 2.4.3 *Let $A : X \to X$ be a linear operator, and X be a normalized space. The operator A is called the contraction if $\exists \alpha \in (0,1)$ that*

$$\|Ax - Ay\| \leq \alpha\|x - y\|.$$

The contraction shortens the distance between mappings.

Definition 2.4.4 *Let $A : X \to X$ be a linear operator. Then the point $x^* \in X$, for which*

$$Ax^* = x^*,$$

is called a fixed point of the operator A.

The fixed points of the operator are those that are mapped to itself by the operator.

Theorem 2.4.1 (The simplest method of contraction mapping) *Let $A : \mathbb{R}^n \to \mathbb{R}^n$ be a linear operator, which is the contraction with the parameter α, and $\|\cdot\|$ be an arbitrary norm in $\mathbb{R}^n$. Thus, the operator A has one fixed point x^*, and the sequence*

$$x_{k+1} = Ax_k, \quad x_0 \in \mathbb{R}^n$$

converges to x^* as $k \to +\infty$. Moreover,

$$\|x^* - x_k\| \leq \frac{\alpha^k}{1-\alpha}\|x_1 - x_0\|.$$

Proof. We can show that the sequence x_k is fundamental. Its convergence will follow from it (despite the introduced norm). By the triangle inequality, we obtain

$$\|x_{k+p} - x_k\| \leq \|x_{k+p} - x_{k+p-1}\| + \|x_{k+p-1} - x_{k+p-2}\| + ... + \|x_{k+1} - x_k\|.$$

According to the definition of the sequence x_k,

$$\|x_{k+p} - x_k\| \leq \|Ax_{k+p-1} - Ax_{k+p-2}\| + ... + \|Ax_k - Ax_{k-1}\| \leq$$

$$\left(\alpha^{k+p-1} + ... + \alpha^k\right)\|x_1 - x_0\| = \frac{\alpha^k(1-\alpha^p)}{1-\alpha}\|x_1 - x_0\| \leq \frac{\alpha^k}{1-\alpha}\|x_1 - x_0\|,$$

which proves fundamentality. Hence, the sequence converges. Let $p \to +\infty$, then

$$\|x^* - x_k\| \le \frac{\alpha^k}{1-\alpha}\|x_1 - x_0\|.$$

Next, we are going to prove the uniqueness. Let x^*, y^* be two fixed points. Then,

$$\|x^* - y^*\| = \|Ax^* - Ay^*\| \le \alpha\|x^* - y^*\|.$$

Thus, $\|x^* - y^*\| = 0$ and $x^* = y^*$. □

2.4.2 Iterative Policy Evaluation

Now we are ready to formulate the theory about iterative policy evaluation.

Theorem 2.4.2 (Iterative Policy Evaluation) *Let $v_\pi(s)$ be a state-value function, $s \in S$. Let also $v_0(s) \in \mathbb{R}$ be a random number if the state s is not terminal, and $v_0(s) = 0$ otherwise. Then the sequence*

$$v_{k+1}(s) = \sum_{a \in A(s)} \pi(a|s) \sum_{s' \in S} \mathcal{P}^a_{ss'}\left(\mathcal{R}^a_{ss'} + \gamma v_k\left(s'\right)\right)$$

converges to $v_\pi(s)$ as $k \to +\infty$.
Proof. *Let's consider a so-called Bellman operator.*

$$Bv = r_\pi + \gamma T_\pi v.$$

Let's prove that this operator is the contraction of the considered norm $\|\cdot\|_\infty$. Since $\|T_\pi\| = 1$, then

$$\|Bv - Bv'\|_\infty = \|\gamma T_\pi(v - v')\| \le \gamma\|v - v'\|.$$

All we've got left is to apply the proven method of contraction mapping. □

Remark 2.4.5 *We can evaluate the error of the substitution of the true state value for its approximation.*

$$|v_\pi(s) - v_k(s)| \le \frac{\gamma^k}{1-\gamma}\max_{s \in S}|v_1(s) - v_0(s)|.$$

To evaluate the state value under a policy, we can use the following algorithm:

Algorithm 1. Iterative policy evaluation.

```
1: Select the policy for evaluation
2: Initialize $v_0(s) = 0$ for all $s \in S$
3: Initialize $k = 0$
4: do
5:     $\Delta \leftarrow 0$
6:     for $s \in S$ do
7:         $v^* \leftarrow v_k(s)$
8:         $v_{k+1}(s) \leftarrow \sum\limits_{a \in A(s)} \pi(a|s) \sum\limits_{s' \in S} \mathcal{P}^a_{ss'} \left(\mathcal{R}^a_{ss'} + \gamma v_k(s')\right)$
9:         $\Delta \leftarrow \max\left(\Delta, |v^* - v_{k+1}(s)|\right)$
10:    end for
11:    $k \leftarrow k + 1$
12: while $\Delta < \theta$ (a small positive number)
13: return $v_k$
```

Apply the described algorithm to the example. The MDP flowchart is shown in Fig. 16.

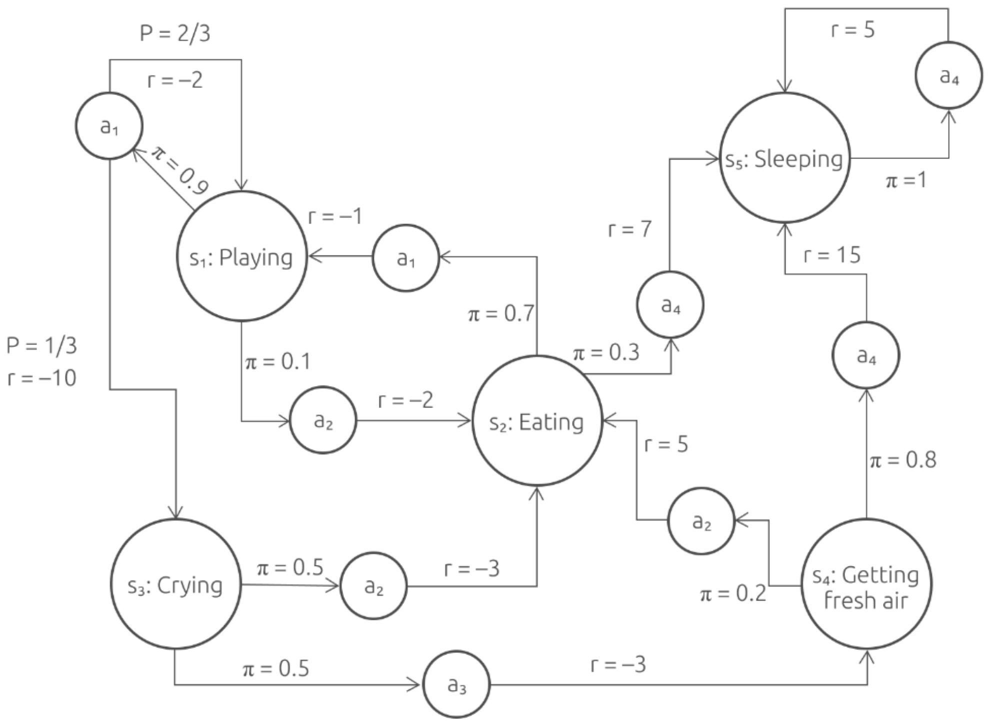


Figure 14: The input data.

At first, all state values were considered zero. Next, with each step, the state values were updated according to the obtained Bellman equations:

$$v_{k+1}(s_1) = 0.9\left(\frac{1}{3}\left(-10+0.8v_k(s_3)\right)+\frac{2}{3}\left(-2+0.8v_k(s_1)\right)\right)+$$
$$+0.1\cdot 1\cdot(-2+0.8v_k(s_2)).$$
$$v_{k+1}(s_2) = 0.7(-1+0.8v_k(s_1))+0.3(7+0.8v_k(s_5))$$
$$v_{k+1}(s_3) = 0.5(-3+0.8v_k(s_2))+0.5(-3+0.8v_k(s_4))$$
$$v_{k+1}(s_4) = 0.2(5+0.8v_k(s_2))+0.8(15+0.8v_k(s_5))$$
$$v_{k+1}(s_5) = 5+0.8v_k(s_5)$$

The results of the calculations (rounded to the nearest hundredth) are given in the table.

Step	$v_k(s_1)$	$v_k(s_2)$	$v_k(s_3)$	$v_k(s_4)$	$v_k(s_5)$
0	0	0	0	0	0
1	-4.4	1.4	-3	13	5
2	-7.12	0.14	2.76	16.42	9
3	-7.14	-0.43	3.62	18.78	12.2
...	...	...	...	...	...
36	-2.24	6.14	11.45	29.98	24.99
37	-2.23	6.15	11.45	29.98	24.99
38	-2.23	6.15	11.45	29.98	24.99
39	-2.23	6.15	11.45	29.98	25

Note that a large number of iterations was necessary to achieve the required accuracy (up to the hundredth). The error is less than

$$13\cdot\frac{0.8^{39}}{1-0.8}\approx 0.011.$$

2.5 Optimal Policies

In reinforcement learning, we are interested not only in states but also in policies that maximize a discounted return. By finding a solution to the Bellman equation for each policy, we can pick the best or so-called optimal policy. What does it mean? We will begin with the definitions.

Definition 2.5.1 *A function*

$$v^*(s) = \max_{\pi} v_\pi(s), \quad \forall s\in S,$$

is called an optimal state-value function.

Definition 2.5.2 *A function*

$$q^*(s,a) = \max_{\pi} q_\pi(s,a), \quad \forall s \in S,\ \forall a \in A(s),$$

is called an optimal action-value function.

The introduced functions show how large the value of a state and/or action can be.

Remark 2.5.1 *We can show that the maxima in the case of a finite Markov decision process exist. For example, one can wonder why $v^*(s)$ is a correctly defined function. According to the Bellman equation,*

$$v_\pi(s) = \sum_{a\in A(s)} \pi(a|s) \sum_{s'\in S} \mathcal{P}^a_{ss'} \left(\mathcal{R}^a_{ss'} + \gamma v_\pi(s')\right) \leq$$

$$\leq \sum_{a\in A(s)} \pi(a|s) \max_{a\in A(s)} \left(\sum_{s'\in S} \mathcal{P}^a_{ss'} \left(\mathcal{R}^a_{ss'} + \gamma v_\pi(s')\right)\right) =$$

$$= \max_{a\in A(s)} \left(\sum_{s'\in S} \mathcal{P}^a_{ss'} \left(\mathcal{R}^a_{ss'} + \gamma v_\pi(s')\right)\right) = \max_{a\in A(s)} q_\pi(s,a).$$

Since the set $A(s)$ is finite, the policy, under which the expression on the right is obtained, is as follows (when the maximum is attained with one action):

$$\pi(a|s) = \begin{cases} 1, & a = \arg\max\limits_{a\in A(s)} q_\pi(s,a) \\ 0, & \text{otherwise} \end{cases}.$$

When the maximum is attained with multiple actions, any of them can be chosen. Well, we've found that the function is bounded from above and found the policy, under which the estimate is obtained. Thus, it is the desired maximum.

It will be useful to discuss this using the policy terms.

Definition 2.5.3 *It is said that a policy π is not worse than a policy π', and it is written as*

$$\pi \geq \pi',$$

if for all $s \in S$

$$v_\pi(s) \geq v_{\pi'}(s).$$

Remark 2.5.2 *It is often said that the introduced relation $\geq$ induces a partial ordering on the set of all policies π. Note that sometimes it's not possible to compare two policies π_1 and π_2. This means there are $s_1, s_2 \in S$ so that*

$$v_{\pi_1}(s_1) \leq v_{\pi_2}(s_1),$$

however,

$$v_{\pi_1}(s_2) \geq v_{\pi_2}(s_2).$$

Example 2.5.1 *If, instead of the original policy, Fig. (15.a)), we use the policy π', where $\pi'(a_1|s_1) = 1, \pi'(a_2|s_1) = 0, \pi'(a_4|s_4) = 1, \pi'(a_2|s_4) = 0$ (Fig. (15.b)) in the considered example, we obtain the following state values:*

Policy	$v(s_1)$	$v(s_2)$	$v(s_3)$	$v(s_4)$	$v(s_5)$
π	-2.23	6.15	11.45	29.98	25
π'	-2.32	6.1	13.44	35	25

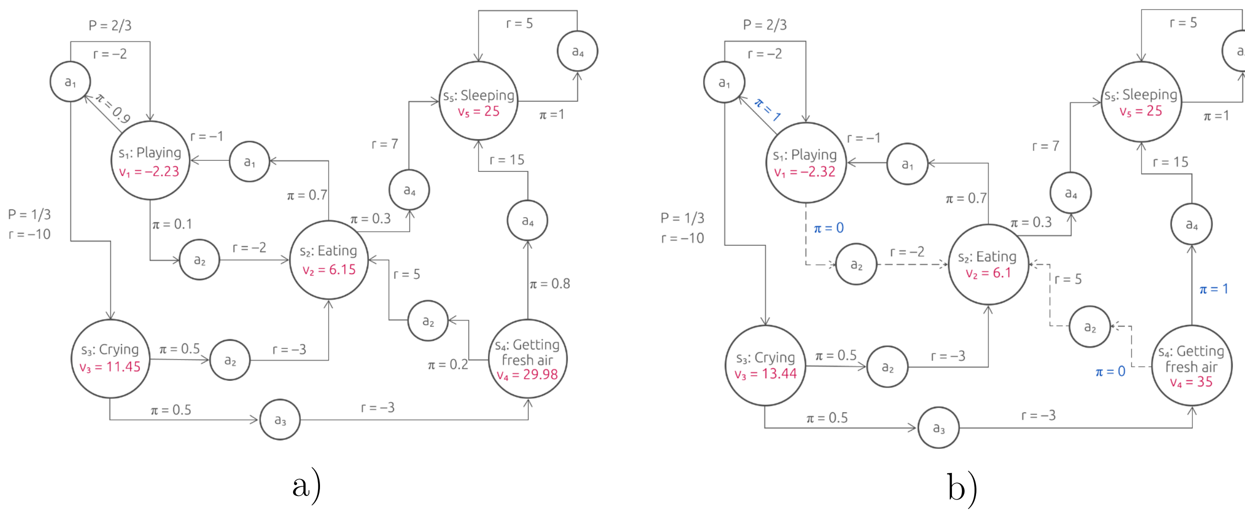


Figure 15: The example of incomparable policies.

Note that larger state values are obtained for the states s_1 and s_2 when the policy π is used, and vice versa for s_3 and s_4.

We aim to find a policy that is better than all other policies.

Definition 2.5.4 *A policy π^* is optimal if, for any policy π, the following inequality is satisfied:*

$$\pi^* \geq \pi.$$

Remark 2.5.3 *The introduced definition raises multiple questions. Does an optimal policy always exist? Will the optimal policy be the only one? In a general case, the answer to both questions is No. We will return to that a little bit later.*

At the same time, having the optimal policy π^*, it's logical to consider the functions $v_{\pi^*}(s)$ and $q_{\pi^*}(s, a)$ that can be an optimal state-value and action-value function respectively. Let's define the relationship between them. First, let's state that, when an optimal policy exists, the functions $v_{\pi^*}(s)$ and $q_{\pi^*}(s, a)$ are defined correctly (they output the same value under different optimal policies).

Lemma 2.5.1 *For any optimal policies π_1^* and π_2^**

$$v_{\pi_1^*} \equiv v_{\pi_2^*} \text{ and } q_{\pi_1^*} \equiv q_{\pi_2^*}.$$

Proof. Let's prove the first identity. Since π_1^* is an optimal policy, then, according to the definition,

$$v_{\pi_1^*}(s) \geq v_\pi(s), \quad \forall s \in S,$$

when $\pi = \pi_2^*$, thus, $v_{\pi_1^*}(s) \geq v_{\pi_2^*}(s)$. We switch π_1^* and π_2^* and obtain the inequality $v_{\pi_2^*}(s) \geq v_{\pi_1^*}(s)$, which completes the proof. □

So, does an optimal policy always exist? In the assumption that the Markov decision process is finite, an optimal policy always exists. Moreover, the following theorem is also true.

Theorem 2.5.1 *Let a Markov decision process be finite. Then,*

1. *There is an optimal policy π^*.*
2. *For any optimal policy $v_{\pi^*} \equiv v^*$.*
3. *For any optimal policy $q_{\pi^*} \equiv q^*$.*

Well, optimal state-value and action-value functions are the corresponding state-value and action-value functions when the optimal policy is followed.

Proof. The proof follows from the things discussed earlier. An optimal policy π^* can be the following one (when the maximum is attained with one action):

$$\pi(a|s) = \begin{cases} 1, & a = \arg\max\limits_{a \in A(s)} q_\pi(a) \\ 0, & \text{otherwise} \end{cases}.$$

When the maximum is attained with multiple actions, any of them can be chosen. Further statements are obvious consequences. □

As the Bellman equations for $v_\pi(s)$ and $q_\pi(s, a)$ have been found, let's find them for $v^*(s)$ and $q^*(s, a)$.

Lemma 2.5.2 *Let $v^*(s)$ and $q^*(s,a)$ be optimal state-value and action-value functions respectively. Then, the following relations are applicable:*

$$v^*(s) = \max_{a \in A(s)} \sum_{s' \in S} \mathcal{P}^a_{ss'} \left(\mathcal{R}^a_{ss'} + \gamma v^*(s') \right),$$

$$q^*(s,a) = \sum_{s' \in S} \mathcal{P}^a_{ss'} \left(\mathcal{R}^a_{ss'} + \gamma \max_{a' \in A(s')} q^*(s', a') \right).$$

Proof. Since optimal value functions are value functions under the optimal policy π^*, that is, if the maximum is attained with one action, under the policy:

$$\pi(a|s) = \begin{cases} 1, & a = \arg\max\limits_{a \in A(s)} q_\pi(a) \\ 0, & \text{otherwise} \end{cases}.$$

Since, in a general case,

$$v_\pi(s) = \sum_{a \in A(s)} \pi(a|s) \sum_{s' \in S} \mathcal{P}^a_{ss'}(\mathcal{P}^a_{ss'} + \gamma v_\pi(s')),$$

then, for the optimal policy, π^*

$$v^*(s) = \max_{a \in A(s)} \sum_{s' \in S} \mathcal{P}^a_{ss'} \left(\mathcal{R}^a_{ss'} + \gamma v^*(s') \right).$$

The proof of the second relation is similar. □

Remark 2.5.4 *Note that the knowledge of the optimal state-value function gives us an optimal policy: transition to a state with the greatest value.*

Remark 2.5.5 *Note that the obtained equations are called the Bellman optimality equations with respect to the value of states and actions respectively. Unlike the Bellman equations, the obtained system is not a system of linear algebraic equations, and solving it is another problem.*

It turns out that the discussed iterative approach can also be used in this case.

Theorem 2.5.2 *Let $v^*(s)$ be an optimal state-value function, $s \in S$. Let also $v_0(s) \in \mathbb{R}$ be a random number if the state s is not terminal, and $v_0(s) = 0$ otherwise. Then the sequence*

$$v_{k+1}(s) = \max_{a \in A(s)} \sum_{s' \in S} \mathcal{P}^a_{ss'} \left(\mathcal{R}^a_{ss'} + \gamma v_k(s') \right)$$

converges to $v^*(s)$ as $k \to +\infty$.

Proof. Let's consider an optimal Bellman operator B^* based on the rule

$$B^*v = \max_{a\in A} \sum_{s'\in S} \mathcal{P}^a_{ss'}(\mathcal{R}^a_{ss'} + \gamma v(s')),$$

where

$$v = \big(v(s_1), v(s_2), \dots, v(s_{|S|})\big)^T .$$

Let's consider B^*v and B^*v'. Since a Markov decision process is finite, there are such a and a' (not necessarily different) that

$$B^*v = \sum_{s'\in S} \mathcal{P}^a_{ss'}(R^a_{ss'} + \gamma v(s')),$$

$$B^*v' = \sum_{s'\in S} \mathcal{P}^{a'}_{ss'}(R^{a'}_{ss'} + \gamma v'(s')).$$

Moreover, $B^*v' \geq \sum\limits_{s'\in S} \mathcal{P}^a_{ss'}(R^a_{ss'} + \gamma v'(s')$, thus,

$$B^*v - B^*v' \leq \sum_{s'\in S} \mathcal{P}^a_{ss'}(R^a_{ss'} + \gamma v(s')) - \sum_{s'\in S} \mathcal{P}^a_{ss'}(R^a_{ss'} + \gamma v'(s')) =$$

$$= \gamma \sum_{s'\in S} \mathcal{P}^a_{ss'}(v(s') - v'(s')) \leq \gamma \|v - v'\|_\infty.$$

We switch v and v' and obtain that

$$|B^*v - B^*v'| \leq \gamma \|v - v'\|_\infty \ \Rightarrow \|B^*v - B^*v'\|_\infty \leq \gamma \|v - v'\|_\infty.$$

All we've got left is to use the simplest method of contraction mapping. □

As earlier, we can formulate the algorithm of the iterative search for optimal functions of the states.

Algorithm 2. Iterative evaluation of the optimal policy.

```
1:  Select the policy for evaluation
2:  Initialize $v_0(s) = 0$ for all $s \in S$
3:  Initialize $k = 0$
4:  do
5:      $\Delta \leftarrow 0$
6:      for $s \in S$ do
7:          $v^* \leftarrow v_k(s)$
8:          $v_{k+1}(s) \leftarrow \max_{a\in A(s)} \sum_{s'\in S} \mathcal{P}^a_{ss'} \left(\mathcal{R}^a_{ss'} + \gamma v_k\left(s'\right)\right)$
9:          $\Delta \leftarrow \max\left(\Delta, |v^* - v_{k+1}(s)|\right)$
10:     end for
11:     $k \leftarrow k + 1$
12: while $\Delta < \theta$ (a small positive number)
13: return $v_k$
```

Remark 2.5.6 *As has been done earlier, we can evaluate the error between the true value and the approximate value of the optimal policy at each step of the algorithm.*

Let's return to the example of the child's day (Fig. 16) and find out, which actions the agent should take to maximize the discounted return.

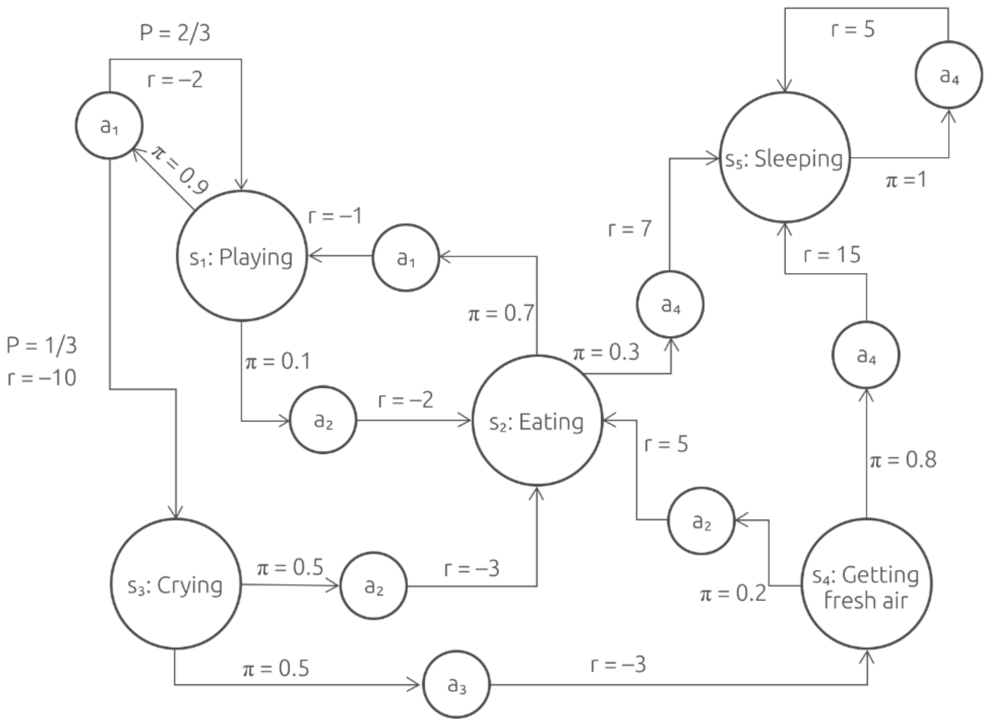


Figure 16: The input data.

The state values under the optimal policy are shown in Fig. 17.

We are going to make sure that the obtained values do follow the optimal policy. In other words, we will verify that, for each state, the following equality is satisfied:

$$v^*(s) = \max_{a \in A(s)} \sum_{s' \in S} \mathcal{P}^a_{ss'} \left(\mathcal{R}^a_{ss'} + \gamma v^*(s')\right).$$

For $v^*(s_1)$, we have

$$19.6 = \max \left\{ \left(\frac{1}{3}(-10 + 0.8 \cdot 25) + \frac{2}{3}(-2 + 0.8 \cdot 19.6) \right), \right.$$

$$\left. (-2 + 0.8 \cdot 27) \right\},$$

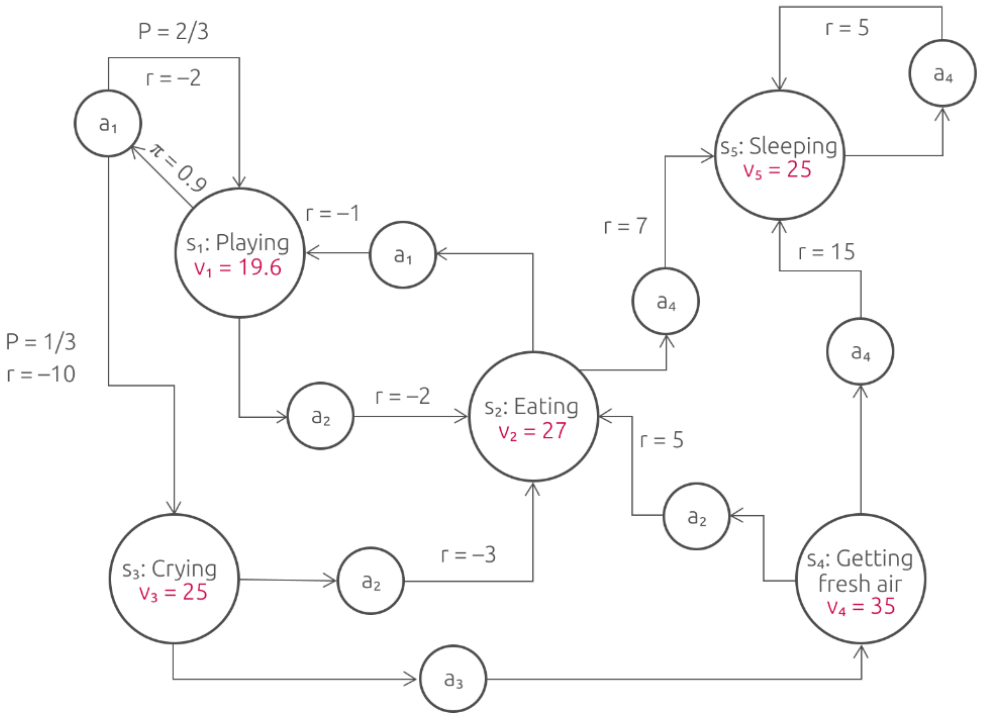


Figure 17: The state values under the optimal policy.

that is,

$$19.6 = \max\{12.45, 19.6\},$$

which is true. Similarly, for $v^*(s_2)$:

$$27 = \max\{(-1 + 0.8 \cdot 19.6), (7 + 0.8 \cdot 25)\} = \max\{14.68, 27\},$$

which is also true, and so on.

When the values of the optimal state-value function are known, the policy is obviously formed. It's necessary just to transition to the state with the maximum value (Fig. 18).

As you can see, our policy is now entirely deterministic because the agent knows which action to take depending on the moment. For example, when in the state Playing, it's necessary to eat and go to bed, and, when in the state Crying, it's time to go outside, then eat and fall asleep.

There remains the question of obtaining the required state values. To obtain accurate values, we can solve a system of Bellman optimality equations. In our example, it will be as follows:

$$v^*(s_1) = \max\left\{ \left(\frac{1}{3}(-10 + 0.8v^*(s_3)) + \frac{2}{3}(-2 + 0.8v^*(s_1))\right),\right.$$

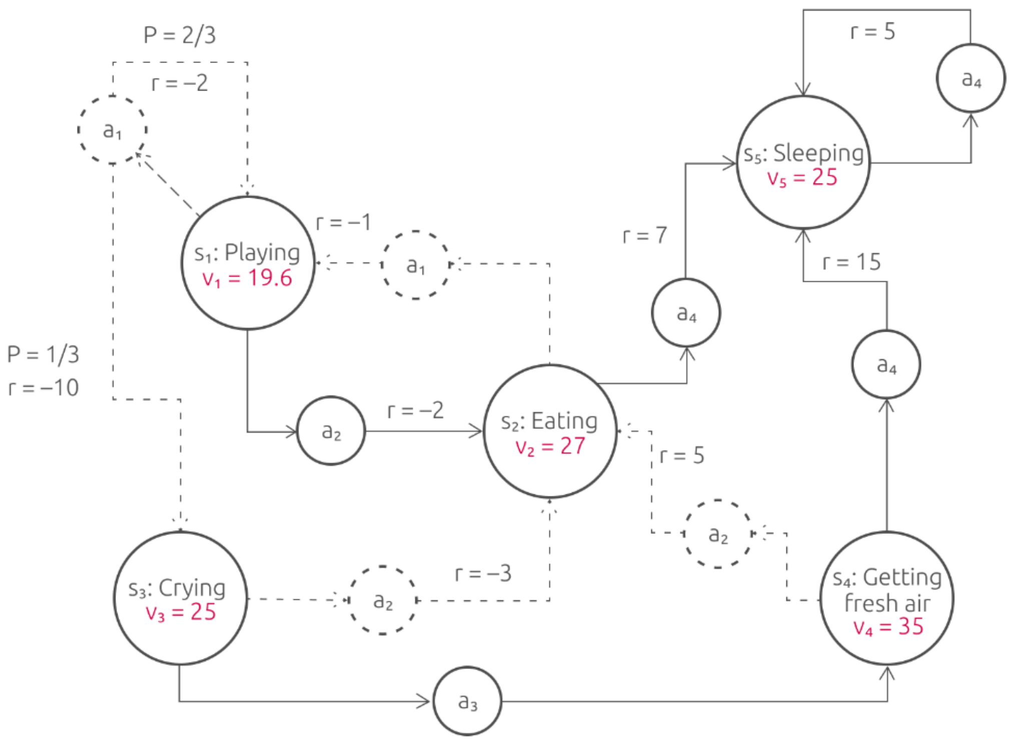


Figure 18: The optimal policy.

$$(-2 + 0.8v^*(s_2))\Big\},$$

$$v^*(s_2) = \max\{(-1 + 0.8v^*(s_1)),\ (7 + 0.8v^*(s_5))\}$$
$$v^*(s_3) = \max\{(-3 + 0.8v^*(s_2)),\ (-3 + 0.8v^*(s_4))\}$$
$$v^*(s_4) = \max\{(5 + 0.8v^*(s_2)),\ (15 + 0.8v^*(s_5))\}$$
$$v^*(s_5) = 5 + 0.8v^*(s_5)$$

This system is not linear, and it's not easy to solve. We can use the discussed iterative approach. In the first step, we assume that all values are zero. The results of the calculations (rounded to the nearest hundredth) are given in the table.

Step	$v_k(s_1)$	$v_k(s_2)$	$v_k(s_3)$	$v_k(s_4)$	$v_k(s_5)$
0	0	0	0	0	0
1	-2	7	-3	15	5
2	3.6	11	9	19	9
3	6.8	14.2	12.2	22.2	12.2
...	...	...	...	...	...
38	19.59	26.99	24.99	34.99	24.99
39	19.6	27	25	35	25

Note that, in our case, about forty iterations were performed to achieve the acceptable level of accuracy (up to the hundredth).

2.6 SARSA

The machine learning methods discussed earlier have one big drawback. They assume that the agent is aware of the environment configuration. To put it differently, in the Bellman equation,

$$v_\pi(s) = \sum_{a\in A(s)} \pi(a|s) \sum_{s'\in S} \mathcal{P}^a_{ss'} \left(\mathcal{R}^a_{ss'} + \gamma v_\pi(s')\right),$$

it is assumed that $\mathcal{P}^a_{ss'}$ (the probabilities of the transition from the state s to s' when the action a is taken) are known, and all the expected returns, or $\mathcal{R}^a_{ss'}$. However, it differs in practice. An obvious solution is to generate estimates based on the taken actions and obtained results. It's like the idea we discussed when talking about multi-armed bandits. Let's recall the recurrence formula for finding the value of an action based on the exponential moving average.

$$q_{t+1}(a) = q_t(a) + \alpha\left(r_{t+1} - q_t(a)\right),$$

where $\alpha \in (0, 1]$. This approach is also used to find the value of states. It's called temporal-difference (TD) learning. Considering that

$$v_\pi(s) = \mathsf{E}_\pi\left(G_t|s_t = s\right) = \mathsf{E}_\pi\left(\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}|s_t = s\right) = \mathsf{E}_\pi\left(r_{t+1} + \gamma v_\pi(s')\,|s_t = s\right),$$

we can find the value of the state s_t:

$$v(s_t) \leftarrow v(s_t) + \alpha(r_{t+1} + \gamma v(s_{t+1}) - v(s_t))$$

In this case, being in the state s_t, the agent following a policy takes the action a_t, receives the reward r_{t+1}, and transitions to the state s_{t+1}. The value $v(s_{t+1})$ of a new state s_{t+1} is also known. Thus, we can update the value of the state s_t. In the next step, the value estimate for the previous step is updated.

The state-value estimate is important, but it is not the goal because actions lead to rewards, and the actions are selected according to a policy. That's why a policy should also be optimized in learning. For that, we can use the idea of TD to estimate the action functions $q(s, a)$, that is

$$q(s_t, a_t) \leftarrow q(s_t, a_t) + \alpha(r_{t+1} + \gamma q(s_{t+1}, a_{t+1}) - q(s_t, a_t))$$

You may notice that the same sequence repeats: state, action, reward, state, and so on, or

$$s, a, r, s, a, r, s, a, r, \dots,$$

and that's why this method is called *SARSA*. When the estimated value of each action is known, we can use a greedy policy of selecting an action, which means the following policy:

$$A_t = \operatorname*{Arg\,max}_{a \in A(s_t)} q_t(a), \quad \pi(a|s_t) = \begin{cases} \frac{1}{|A_t|}, & a \in A_t \\ 0, & \text{otherwise} \end{cases},$$

and it's also important to focus on exploration. We can use the discussed ε-greedy policy:

$$\pi_t(a|s_t) = \begin{cases} \dfrac{1-\varepsilon}{|A_t|} + \dfrac{\varepsilon}{|A(s_t)|}, & a \in A_t \\ \\ \dfrac{\varepsilon}{|A(s_t)|}, & a \notin A_t \end{cases},$$

Here's a formal description of the *SARSA* algorithm.

Algorithm 3. The SARSA algorithm.

```
1: Let $S$ be a set of states, and $A(s)$, $s \in S$, be a set of actions available in the
   state $s$.
2: Initialize $q(s,a)$, $s \in S$, $s$ is not terminal, $a \in A(s)$ arbitrarily
3: Initialize $\alpha$ and $\gamma$
4: for each game do
5:     Initialize a nonterminal state $s_0$ at random
6:     Select $a_0$ under the policy $\pi_0(a|s_0)$
7:     $t \leftarrow 0$
8:     for each step of the game $t$ until a stopping criterion is reached or until $s_t$
   is a nonterminal state, do
9:         Take action $a_t$, find $r_{t+1}$, transition to $s_{t+1}$
10:        if $s_{t+1}$ is a terminal state, then
11:            $q(s_t,a_t) \leftarrow q(s_t,a_t) + \alpha(r_{t+1} - q(s_t,a_t))$
12:        else
13:            Select $a_{t+1}$ under the policy $\pi_{t+1}(a|s_{t+1})$
14:            $q(s_t,a_t) \leftarrow q(s_t,a_t) + \alpha(r_{t+1} + \gamma q(s_{t+1},a_{t+1}) - q(s_t,a_t))$
15:        end if
16:        $t \leftarrow (t+1)$
17:    end for
18: end for
```

According to the algorithm, the values of all available actions $a \in A(s)$ for each nonterminal state $s \in S$ are initialized at random in the first step. Then, a series of games is played. In each game, an initial nonterminal state of the agent is initialized at random. Then, in each game, at each step t, the agent in the state s_t selects an action a_t under the selected policy, transitions to the next state s_{t+1}, and receives a reward r_{t+1}. If the state s_{t+1} is not terminal, then the action a_{t+1} is selected under the policy, and the value $q(s_t, a_t)$ is updated according to the formula.

$$q(s_t, a_t) \leftarrow q(s_t, a_t) + \alpha(r_{t+1} + \gamma q(s_{t+1}, a_{t+1}) - q(s_t, a_t)).$$

Then, a transition to a new iteration of the game is made. If the state s_{t+1} is terminal, then $A(s_{t+1}) = \varnothing$ and

$$q(s_t, a_t) \leftarrow q(s_t, a_t) + \alpha(r_{t+1} - q(s_t, a_t)).$$

The game ends when a transition to a terminal state is made or a stopping criterion is reached. A stopping criterion is reached in the following cases:

- A level set for the sum of received rewards is reached.
- A level set for the average rewards is reached.
- A level set for the number of steps is reached.
- Etc.

Now we can apply *SARSA* to the discussed (but slightly modified) example. We will consider the state Sleeping as a terminal state (when the agent transitions to this state, the game ends). The flowchart is shown in Fig. 19.

Assume that $\gamma = 0.8$, $\alpha = 0.1$. Our policy will be the ε-greedy policy with the value $\varepsilon = 0.1$. Note that we don't know the probabilities of transitions from states s to states s' as a result of taking the action a and we don't know which rewards we will get in this case. We will estimate values during the game. For convenience, let's create a table and fill it out with the action values depending on the states. The next thing will be to perform initialization (set all possible values to zero). Note that the symbol $\times$ denotes that an action is not available in the current state.

	a_1	a_2	a_3	a_4
s_1	0	0	$\times$	$\times$
s_2	0	$\times$	$\times$	0
s_3	$\times$	0	0	$\times$
s_4	$\times$	0	$\times$	0
s_5	$\times$	$\times$	$\times$	$\times$

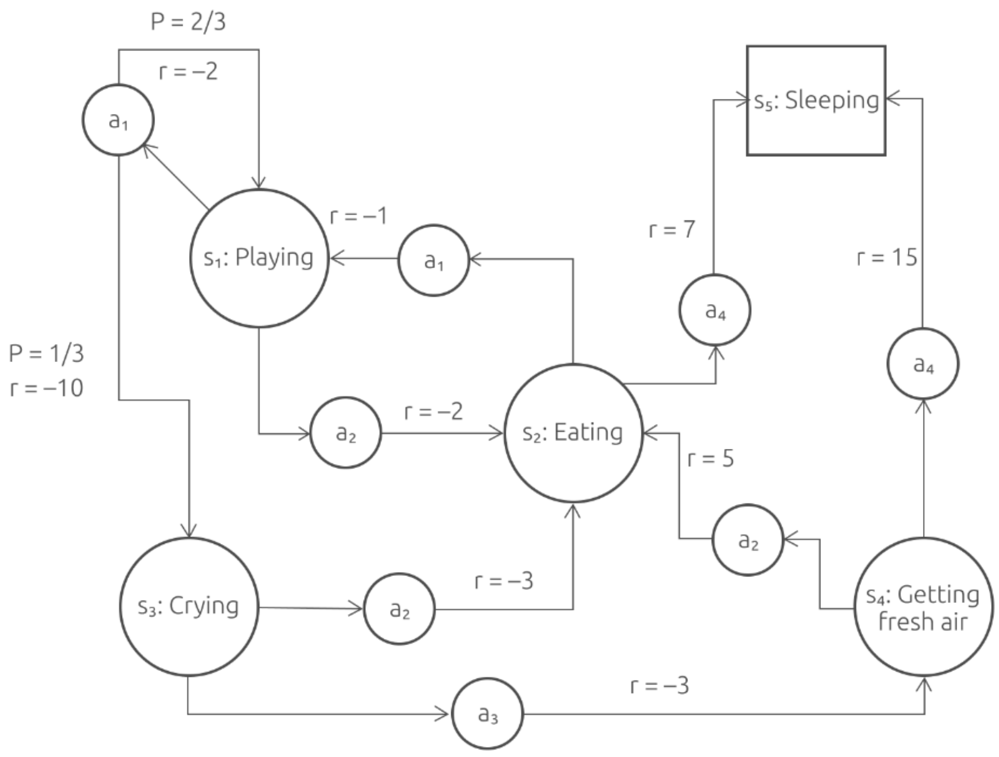


Figure 19: The environment description.

Assume that we start from the state s_1. The actions a_1 and a_2 are available, and $q(s_1, a_1) = q(s_1, a_2) = 0$. According to the policy, an available action is selected at random in this case. Let the action a_2 be selected. Then the agent transitions to the state s_2 and receives the reward $r = -2$. In the state s_2, the next action should be selected. The actions a_1 and a_4 are available. Since the values are the same (they are equal to zero), we select any, for example, a_4. Now we can update $q(s_1, a_2)$.

$$q(s_1, a_2) \leftarrow q(s_1, a_2) + \alpha(r + \gamma q(s_2, a_4) - q(s_1, a_2)) =$$
$$= 0 + 0.1 \cdot (-2 + 0.8 \cdot 0 - 0) = -0.2.$$

We add the obtained value to the table.

	a_1	a_2	a_3	a_4
s_1	0	−0.2	×	×
s_2	0	×	×	0
s_3	×	0	0	×
s_4	×	0	×	0
s_5	×	×	×	×

Let's continue with the algorithm.
Current state: s_2

Selected action: a_4
Next state: s_5
Reward: 7
Note that since s_5 is a terminal state, the game ends. Then,

$$q(s_2, a_4) \leftarrow 0 + 0.1 \cdot (7 - 0) = 0.7.$$

	a_1	a_2	a_3	a_4
s_1	0	-0.2	×	×
s_2	0	×	×	0.7
s_3	×	0	0	×
s_4	×	0	×	0
s_5	×	×	×	×

Several games later, we obtain the following table.

	a_1	a_2	a_3	a_4
s_1	-0.32	0.88	×	×
s_2	-0.02	×	×	6.5
s_3	×	-0.09	-0.19	×
s_4	×	0.99	×	6.14
s_5	×	×	×	×

To estimate a taken action, we look into the future and select the next action, which affects the estimate of the current action.

Remark 2.6.1 *We can use the last table to form an optimal policy (if actions are selected greedily) that differs from a previously found theoretical one because of the terminality of the state s_5 (Fig. 20). In the state s_1, the action a_2 should be selected because*

$$q(s_1, a_2) = 0.88 > q(s_1, a_1) = -0.32.$$

In the state s_2, the action a_4 should be selected because

$$q(s_2, a_4) = 6.5 > q(s_2, a_1) = -0.02$$

etc.

Remark 2.6.2 *In our example, the rewards received throughout the game have the same values, regardless of s, s', and a. Things can be different in a general case.*

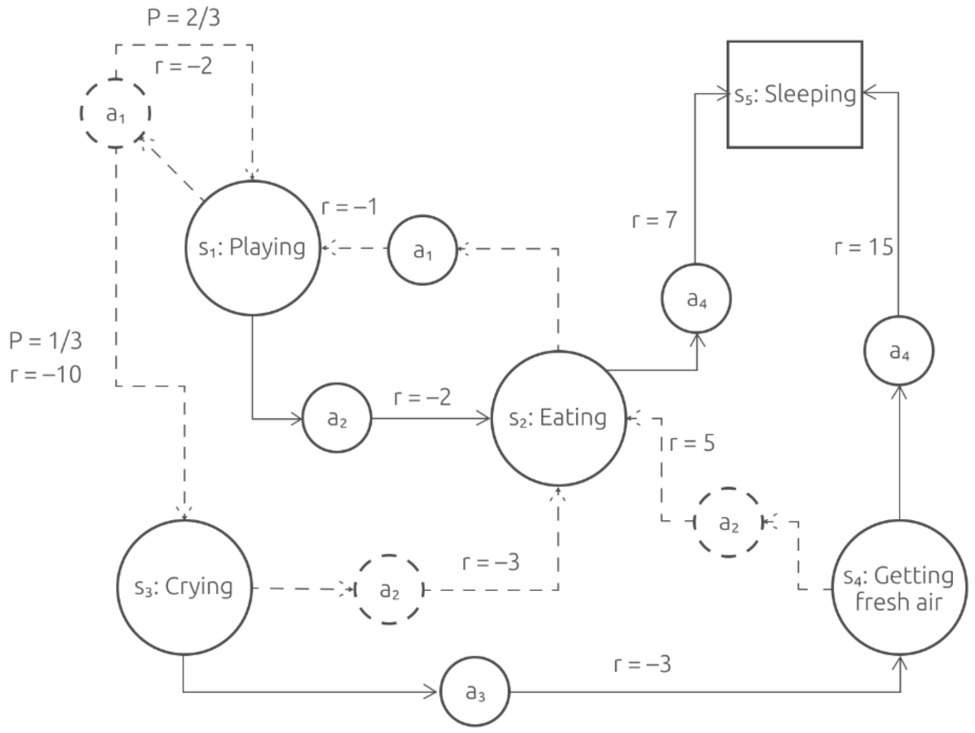


Figure 20: The optimal policy.

2.7 Q-Learning

Let's consider one more TD-based algorithm known as Q-learning. Earlier, we obtained Bellman optimality equations, in particular, an equation for the optimal action-value function.

$$q^*(s,a) = \sum_{s' \in S} \mathcal{P}^a_{ss'} \left(\mathcal{R}^a_{ss'} + \gamma \max_{a' \in A(s')} q^*(s',a') \right).$$

Q-learning approximates the optimal value function as follows:

$$q(s_t,a_t) \leftarrow q(s_t,a_t) + \left(r_{t+1} + \gamma \max_{a \in A(s_{t+1})} q(s_{t+1},a) - q(s_t,a_t) \right).$$

Let's stop for a second and think about the difference between $SARSA$ and Q-learning. $SARSA$ uses the value $q(s_{t+1},a_{t+1})$ to update $q(s_t,a_t)$, which means that the value of taking an action a_{t+1} selected in the state s_{t+1} according to a set policy $\pi_{t+1}(a|s_{t+1})$. Q-learning uses an optimal policy. An action is selected if $\max\limits_{a \in A(s_{t+1})} q(s_{t+1},a)$ is attained for it. Thus, to update the action-value function, we use not the selected policy (possibly, not optimal), but the optimal policy, although in the next step (!). If the current policy is optimal, the algorithms perform the same. Let's formally describe the Q-learning algorithm.

Algorithm 4. Q-learning algorithm

```
1: Let $S$ be a set of states, and $A(s)$, $s \in S$, be a set of actions available in the
   state $s$.
2: Initialize $q(s, a)$, $s \in S$, $s$ is not terminal, $a \in A(s)$ arbitrarily
3: Initialize $\alpha$ and $\gamma$
4: for each game do
5:     Initialize a nonterminal state $s_0$ at random
6:     $t \leftarrow 0$
7:     for each step of the game $t$ until a stopping criterion is reached or until $s_t$
   is a nonterminal state, do
8:         Select $a_t$ under the policy $\pi_t(a|s_t)$
9:         Take action $a_t$, find $r_{t+1}$, transition to $s_{t+1}$
10:        if $s_{t+1}$ is a terminal state, then
11:            $q(s_t, a_t) \leftarrow q(s_t, a_t) + \alpha(r_{t+1} - q(s_t, a_t))$
12:        else
13:            $q(s_t, a_t) \leftarrow q(s_t, a_t) + \alpha(r_{t+1} + \gamma \max_{a \in A(s_{t+1})} q(s_{t+1}, a) - q(s_t, a_t))$
14:        end if
15:        $t \leftarrow (t + 1)$
16:    end for
17: end for
```

Firstly, the main difference from $SARSA$ is a way to estimate the value of an action. Secondly, the choice of an action differs. In $SARSA$, a current and next action are in the current iteration. In Q-learning, only the current action is selected. If the state s_{t+1} is not terminal, $q(s_t, a_t)$ is updated according to the formula:

$$q(s_t, a_t) \leftarrow q(s_t, a_t) + \alpha(r_{t+1} + \gamma \max_{a \in A(s_{t+1})} q(s_{t+1}, a) - q(s_t, a_t)).$$

Then, a transition to a new iteration of the game is made. If the state s_{t+1} is terminal, then $A(s_{t+1}) = \varnothing$ and

$$q(s_t, a_t) \leftarrow q(s_t, a_t) + \alpha(r_{t+1} - q(s_t, a_t)).$$

The game ends when a stopping criterion is reached or a transition to a terminal state is made.

To clarify the difference between Q-learning and $SARSA$, we will use the same example. Let the agent be in a state s_1 in some step and select an action a_2. The estimate table will be as follows:

	a_1	a_2	a_3	a_4
s_1	-0.32	0.88	$\times$	$\times$
s_2	-0.02	$\times$	$\times$	6.5
s_3	$\times$	-0.09	-0.19	$\times$
s_4	$\times$	0.99	$\times$	6.14
s_5	$\times$	$\times$	$\times$	$\times$

Hence, the estimate $q(s_1, a_2)$ is adjusted (regardless of the applied policy) as follows:

$$q(s_1, a_2) \leftarrow 0.88 + 0.1 \cdot (-2 + 0.8 \cdot 6.5 - 0.88) \approx 1.12,$$

since

$$q(s_2, a_4) = \max(q(s_2, a_1), q(s_2, a_4)).$$

However, it's not always the case in $SARSA$. For example, if the action a_1 is selected instead of a_4 under a ε-greedy policy in the state s_2, $q(s_2, a_1)$ will be used to update the value $q(s_1, a_2)$. In Q-learning, a policy (for example, a ε-greedy policy) is used only to select the current action, and it doesn't contribute to updating the estimated value.

2.8 Example

Let's consider the examples of how the discussed algorithms can be applied. We are going to simulate a game, in which the agent needs to find a way out of the maze. An example of a solution is shown in Fig. 21.

In the game, the agent initially finds itself in a starting point. The agent can move in four directions (up, right, down, left). The agent's goal is to exit the maze. The agent receives a reward after each action. The rewards received in one game are accumulated. Each step is rewarded -0.05. When the agent moves to a visited cell, the reward is -0.25. When the agent encounters a wall, the reward is -0.75. When the agent finds a way out, it is granted the reward of $+10$. A terminal state is always reached in one game. A player either finds a way out and wins the game or the sum of the received rewards is less than the set threshold. In this case, the threshold is

$$-\frac{1}{2} \cdot (\text{the size of the maze}) = -32.$$

The ε-greedy algorithm with the value $\varepsilon = 0.1$ was used for learning. The discount rate γ equals 0.9, and $\alpha = 0.1$.

44 initial states are available to the agent (white cells, exclusive of the exit cell). An episode is a series of 44 games. In each series, each state (cell) is used once as an initial state. Learning ends when the agent can find a way out (not necessarily optimal), starting from any of 44 available states. In any state, the agent exactly knows what actions to take in order to find a way out of the maze.

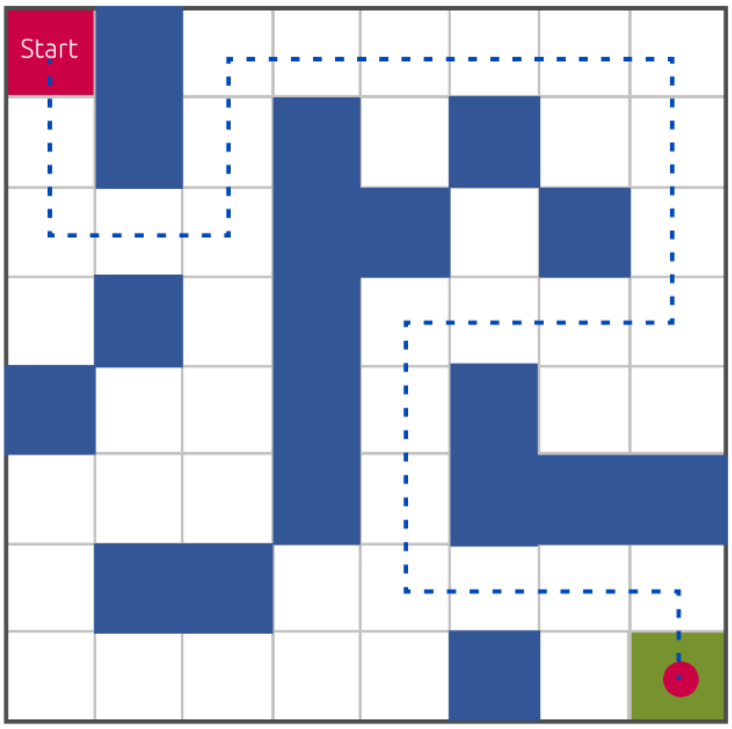


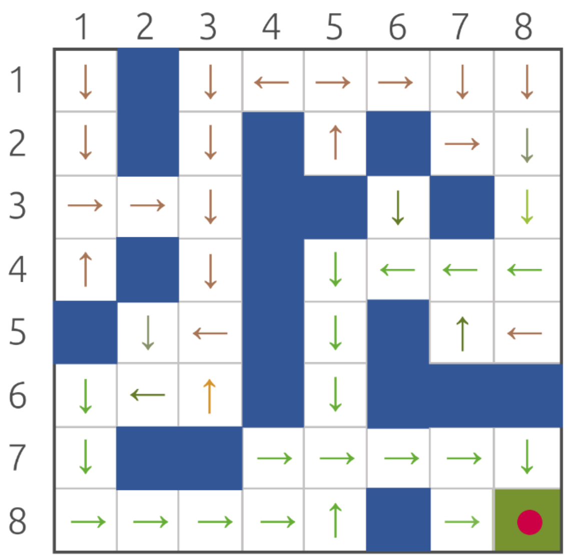
Figure 21: An example of a path through the maze.

Based on the obtained action values for all states, we can create an optimal policy. The policies obtained for $SARSA$ and Q-learning algorithms are shown in Fig. 22.a) and 22.b) respectively.

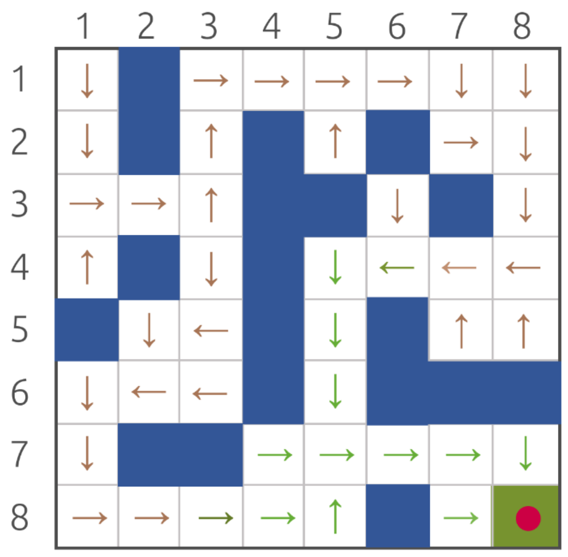
There are interesting differences related to ε-greedy policies. Look at the state $(3, 3)$ (at the intersection of the third row and third column). If it is a starting point, the $SARSA$ policy suggests moving down, and the exit will be found in 17 moves. The Q-learning policy says the opposite. It suggests moving up, but the exit will be found in 21 moves. $SARSA$ is more effective for this state. On the other side, $SARSA$ makes a strange move in the cell $(6, 3)$ (at the intersection of the sixth row and third column). The differences, for example, in the cell $(5, 8)$ don't matter.

2.9 Summary

In this summary, we would like to note that reinforcement learning is probably one of the most exciting and promising machine-learning branches. It encompasses supervised and unsupervised learning at the same time. On the one side, an agent learns by itself with no human help. On the other side, an environment is like a teacher. Another benefit is the ability to conduct learning in a dynamic environment. In this module, we've reviewed only some methods that reinforcement learning has to offer. New solutions and ideas in this field are a popular subject in academic publishing.



a)



b)

Figure 22: The *SARSA* and *Q*-learning algorithm policies.

2.10 Conclusion

Dear students, it was the last module of the Machine Learning Course. We hope that the knowledge you gained will be of great use throughout your career and life and that your experiences in this course will inspire you to take additional courses in artificial intelligence. Good luck and see you soon!