Reinforcement Learning

Simon Hellbrück
Faculty Computer Science
Augsburg University of Applied Sciences
simon.hellbrueck@hs-augsburg.de

Abstract

Reinforcement learning (RL) is a machine learning technique which is utilized to maximize the success of software agents when being in a specific (potentially unknown) environment. Previous achievements prove that RL approaches, which are solving various complex problems, have a high potential to enhance life quality when theoretical approaches are applied in a sophisticated manner.

1 Introduction

When a child runs around, sits on a swing or pulls a face, it has no explicit teacher giving it orders to do these things, rather more it learns by interacting with the current environment. After gathering information while taking specific actions in an environment, a child can then modify its behaviour in order to achieve specific goals. Furthermore, this phenomenon can be summarized as goal-directed learning from interactions in a specific environment [SB18c].

The same concept applies to Reinforcement Learning: An artificial agent learns by interacting with its environment. Just like the child, the artificial agent is able to optimize specific goals while getting constantly rewarded or penalized. Generally speaking, this concept applies to any kind of consecutive decision-making problem that is based on past experience [FLHI+18]. So-called decision-making problems can be modelled by Markov decision processes (MDPs), in principle these are the problems one is trying to solve by using RL [Gos19]. In recent years, applying RL to aforementioned problems has become more and more popular. Moreover, various achievements result from the combination of deep learning techniques with RL [LBH15, Sch15, GBC16]. Deep RL is primarily advantageous when applying it to problems with high dimensional state-space. Former approaches using regular RL techniques had difficulties regarding design in the choice of features [MM02, BNVB13]. By virtue of the ability to learn diverse levels of abstractions from data, applying deep RL has been successful even in complex tasks and with smaller prior knowledge. A deep RL agent can even learn from a large number of perceivable pixels and remains successful [MKS+15]. This ability enables a deep RL agent to imitate human solution approaches [FLHI+18]. In the past years noteworthy work in the field of deep RL can be highlighted: agents gained superhuman skills and convinced the general public by playing Atari games [MKS+15] or beating highly-skilled poker players [BSM17, MSB+17]. Moreover, the work of David Silver's team with RL agents can be considered as a breakthrough. The self-learning agent, AlphaGo, was practising by playing games against itself and was able to defeat Go world champion Lee Sedol [SHM+16], a few years later an RL agent (AlphaStar) was able to master the complex strategy video game StarCraft II, which had been declared as a remarkably difficult challenge for AI learning systems [RP20]. However, these systems are not limited to games, deep RL also has potential for real-world applications, for instance finance [DBK+17], self-driving cars [PYWL17] and robotics [LFDA16, GPG17, PAW+17]. Considering the wide range of applications, the following chapters will elaborate the underlying concept of RL in order to gain a general understanding. Also, exemplary applications will illustrate the principle of RL.

In: Märtin, C. (ed.): Current Trends in Computer Science and Business Information Systems, Proc. of the M.Sc. Seminar, Summer Semester 2020, Augsbug University of Applied Sciences, 2020

2 Overview

The main machine learning models fall into three parts [Hei19]:

- Supervised learning
- Unsupervised learning

- Reinforcement learning

The main difference between (un-)supervised learning and RL is that everything an RL agent perceives should not only be used for understanding, interpreting and prediction, but also for improving the agent's ability to optimize its behaviour in the future in order to be able to accomplish its goals [SB18c]. Moreover, unlike supervised learning, labelled input/output pairs are not needed in RL. The core lies in acquiring new knowledge by discovering unknown territory (exploration) on the one hand and making use of knowledge already obtained before (exploitation) on the other hand [FLHI+18].
Sequential decision problems are in general targeted by using RL [SB18c]. In the following chapters, the RL learning setup as well as the Markov decision process (which is applied in order to mathematically model sequential decision problems) will be explained. Also, examples will illustrate practical applications of the theory thereafter.

3 The Reinforcement Learning Setup

As mentioned before, an RL problem consists of an environment. In this environment an agent learns by the interaction between situation and action – or informally by trial-and-error interactions. Observations can be received by the use of technical devices such as a sensor or in the form of symbolic descriptions. Actions that can be taken do not have to be high level, it can be as simple as water pressure for water supply and heating for example. If the agent is able to thoroughly perceive all the information in the environment that potentially has an influence on the method of choice, the chosen actions are based on true states of the environment. This condition is considered as the best possible base for RL [HH96].

The RL problem is formally expressed as a discrete time stochastic control process. An agent interacts with its environment as follows [FLHI+18]:

- Start of the agent in a specific state and environment: $s_0 \in S$
- Obtaining an initial observation: $\omega_0 \in \Omega$
- Taking an action $a_t \in \mathcal{A}$ at each time frame t

As a result, the following effects appear [FLHI+18]:

- The agents earns a reward: $r_t \in \mathcal{R}$
- The state proceeds: $s_{t+1} \in S$
- The agent gains another observation: $\omega_{t+1} \in \Omega$

Figure 1 shows the fundamental principle of the RL learning setup. The framework was first proposed by Bellman in 1958 [Bel58] and illustrates the main components of reinforcement learning as well as corresponding interactions.

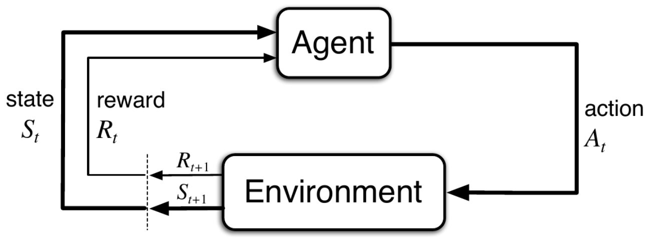

Figure 1: Interaction between agent and environment in RL [SB18c]

4 Markovian Decision Process

The control process mentioned in chapter 3 can be modelled as a Markovian discrete time stochastic control process if all actions that will be taken depend on the current state; hence the full history is not relevant. Formally this condition can be expressed as shown in the following equations and is well-known as the Markov property [Ash18].

$$P(w_{t+1}|w_t, a_t) = P(w_{t+1}|w_t, a_t, ..., , w_0, a_0). \qquad (1)$$

From equation 1 it can be inferred that the new observation an agent receives depends only on the current observation and action.

$$P(r_t|w_t, a_t) = P(r_t|w_t, a_t, ..., , w_0, a_0). \qquad (2)$$

From equation 2 it can be inferred that the new reward an agent receives also only depends on the current observation and action.

A Markov chain is a specific stochastic process. A Markov chain is characterized by the fact that even knowing rarely – or even nothing – about the past can lead to good predictions about future development and can be as helpful as knowing the entire history of the process [NN98]. It is worth pointing out that every action taken in an environment result in a state change, which then lets the agent gain a new observation and a reward. The following sequence illustrates an exemplary run:

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, S_3, A_3... \qquad (3)$$

The state of a system is considered as a parameter or an array of parameters which can be used to characterize the system [SB18c]. Geographical coordinates of a car can be considered as its state. When having state changes depending on the time, the system is called dynamic, which is the case in this example. In a concrete manner, the MDP is defined as the following finite sequence of terms: $\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma$ [FLHI+18]. Where

- $\mathcal{S}$ represents the state space,
- $\mathcal{A}$ the action space,
- $\mathcal{T} : \mathcal{S}$ x $\mathcal{A}$ x $\mathcal{S} \rightarrow [0,1]$ the transition function,
- $\mathcal{R} : \mathcal{S}$ x $\mathcal{A}$ x $\mathcal{S} \rightarrow R$ the reward function,
- $\gamma \in [0,1]$ the discount factor.

In an MDP, the system is completely visible which leads to the equation $w_t = s_t$, which expresses that the observation is identical to the state of the environment. The chance of moving from s_t to s_{t+1} at each time interval t is defined by the transition function $T(s_t, a_t, s_{t+1})$, while the reward function $R(s_t, a_t, s_{t+1})$ returns the reward [FLHI+18]. This is illustrated in Figure 2.

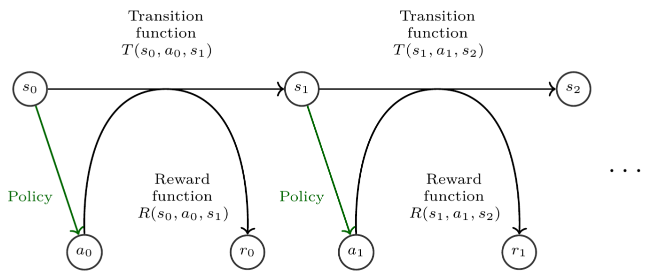
Transition function $T(s_0, a_0, s_1)$
Transition function $T(s_1, a_1, s_2)$
s_0
s_1
s_2
Policy
Reward function $R(s_0, a_0, s_1)$
Policy
Reward function $R(s_1, a_1, s_2)$
...
a_0
r_0
a_1
r_1

Figure 2: Overview MDP [FLHI+18]

The illustrated MDP shows that the agent takes an action at each step which then transitions the state from s_t to s_{t+1} and gets the agent rewarded.

4.1 Policies

Policies are used in order to define how an agent is modeled. More generally speaking, a policy specifies the way an agent chooses an action. Policies can be differentiated regarding their stationariness [SB18c]:

- **Stationary:** A policy is called stationary if it is not changing over time [BSGL09].
- **Non-stationary:** A policy is called non-stationary if it does depend on the time-step [BBBB95].

When using a policy that is stationary, the sequence of states $X_n : n = 0, 1, 2, 3, 4...$ is called a Markov chain with the transition probabilities $P_{ij} = P_{ij}(f(i))$ [BBBB95]. This suits the assumptions mentioned before that the way an agent chooses an action is completely based on current observations and not based on history. Furthermore, policies can be distinguished between stochastic and deterministic [SB18c]:

- **Deterministic:** A deterministic policy is described by $\pi(s) : \mathcal{S} \rightarrow \mathcal{A}$, which is considered as a mapping from set of states to a set of actions [HMLIR07].
- **Stochastic:** A policy is called stochastic if it is described by $\pi(s, a) : \mathcal{S}\ X\ \mathcal{A} \rightarrow [0,1]$ where the chance that in a specific state s action a may be chosen is denoted by $\pi(s, a)$ [FLHI+18].

Policies can also be differentiated in the way of choosing solutions [SB18c]:

- **Greedy policy:** With a greedy policy, the first best solution is chosen. Even if this solution is particularly bad, this solution is always taken. Always using the greedy policy means pure **exploitation**.
- **Optimal policy:** Using an optimal policy means trying to find the best way and means **exploration**. Even if a value would be maximized for an action / situation, another decision is preferred in order to find better ways.
- **ϵ-Greedy policy:** An ϵ-greedy policy represents a trade-off between the aforementioned policies. When using an ϵ-greedy policy, an ϵ is defined in the range of $[0,1]$. Before actually taking an action, a number in the same range ($[0,1]$) is chosen. If the selected number is higher than ϵ, the greedy action will be taken. Otherwise, an explorative action is selected. If ϵ equals 0, the policy is most likely the greedy policy (chances that the random number is zero as well are pretty low). If ϵ equals 1, the agent will always explore [Zyc19].

4.2 Principle of Optimality

It is important to find a suitable policy that optimizes the sum of rewards considering the time [SB18c]:

$$R_t = \sum_{t=0}^{\infty} \gamma^t r_{t+1} \tag{4}$$

Whereby $\gamma \in [0,1]$ is considered the discount factor and decides how powerful the weights of actual rewards are weighted as to upcoming rewards. Example 5 illustrates how rewards are multiplied with respect to the time step (numbers are rounded up to one decimal place). Where the exponent represents the time step from 1 to 4 and $\gamma = 0.8$. In other words: γ to the power of t.

$$0.8^1 = 0.8, 0.8^2 = 0.6, 0.8^3 = 0.5, 0.8^4 = 0,4 \;...\; 0.8^n \tag{5}$$

As γ decreases over time, the rewards will become smaller and smaller and will have consequently less meaning until reaching a state a convergence. Usually a $\gamma < 1$ is used in order to force problems to converge which would not converge otherwise. When using a $\gamma = 1$, the environment itself changes to a final state after a finite number of iterations (which would be the case in a chess game for example). Due to the fact that the transition from state s_t to s_{t+1} does not follow a fixed pattern (as it would when simulating it with a simple loop), it might be observable that a difference between the sum of rewards regarding time (see equation 4) and the expected return $\mathbb{E}[R_t|\pi, s_0]$ occurs [FLHI+18].

4.3 V-value Function

The expected return $\mathbb{E}[R_t|\pi, s_0]$ is considered as what the agent receives on average when applying policy π and starting from state s_0. The return is defined in equation 6 [SB18c].

$$V^\pi(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_{t+1} | \pi, s_0 = s\right] \tag{6}$$

Moreover, when considering all possible value functions, there exists an optimal value function that has a higher value than other functions given state s. The optimal V-value function can be selected by finding the policy π that returns the maximum for the v function for state s [FLHI+18]. At this point, it has to be noted that the symbol $*$ as an exponent represents the optimal/best function or rather policy.

$$V^*(s) = \underset{\pi \in \Pi}{max} V^\pi(s) \tag{7}$$

Figure 3 shows an example of the optimal value function for 16 states.

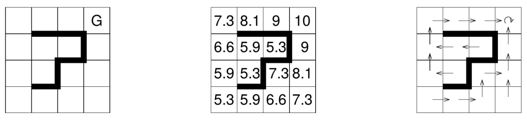

Figure 3: Example of an optimal value function for 16 states [HMLIR07]

The example in figure 3 shows a robot moving in a grid that consists of 16 fields. The robot is able to move vertically and horizontally. Moreover, it can also just stay. It cannot move diagonally. Whenever it tries to leave the grid, it will be punished by -1. For staying in the goal state (G) it will be rewarded by 1, if it goes anywhere else the reward is 0. The starting state s_0 is random. The grid in the middle represents the value function V^* with the corresponding discount factor $\gamma = 0.9$ while the third grid shows an optimal policy. The optimal policy obviously suggests to stay and to move away from positions returning low rewards, directing the agent to the goal state with the highest reward and to stay in the goal state. This example illustrates that an optimal policy simply defines the best behaviour of an agent depending on the current state [SB18c].

4.4 Q-value Function

Also, other functions exist and can be used. For example, the Q-value function $Q^\pi(s,a) : \mathcal{S} X \mathcal{A} \rightarrow \mathbb{R}$ that is defined in the following manner [SB18c]:

$$Q^\pi(s,a) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_{t+1} | \pi, s_0 = s, a_0 = a\right] \tag{8}$$

The main difference between V-value and Q-value function is that the Q function evaluates state-action pairs. This function returns the cumulative reward when starting in a specific state and applying action a (by using the policy π) and always choosing the optimal solution among the possible successive state-action pairs. It is focusing on a particular action at a particular state. The optimal Q-value function can be found by selecting the policy π that returns the maximum for the Q-value function for state s and action a [FLHI+18].

$$Q^*(s,a) = \underset{\pi \in \Pi}{max} Q^\pi(s,a) \tag{9}$$

4.5 Evaluating the Policy

It can be noticed that the Q-value function takes the two parameters (s, a) while the V-value function only takes one (s). Moreover, the best policy can be gained straight from the Q-value function $Q^*(s,a)$ as follows [SB18c]:

$$\pi^*(s) = \underset{a \in \mathcal{A}}{argmax} Q^*(s,a) \tag{10}$$

By finding an action a among all possible actions that leads to the highest return value when using the Q function in a given state s the best policy is extracted. Simply speaking, this means that when the agent find itself in a specific state, action a should be taken in order to gain a high reward. Taking an action in a state means behaving in a situation and – as explained in chapter 4.1 – policies define *how* an agent behaves.

While the ideal V-value function $V(s)$ represents the expected discounted reward for a given state s and following the policy π^*, the best Q-value function $Q^*(s,a)$ returns the expected discounted return for a given state s and a given action a when following the policy π^*. Moreover, an advantage function can be used in order to describe how well an action a performs in comparison to the expected return while following policy π [SB18c]. The advantage function is defined in equation 11.

$$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s) \tag{11}$$

The advantage function is basically the difference between Q-value and V-value, that means the difference between the reward of taking an action in a certain state (expressed by the Q-value function) and the reward of being in a specific state (expressed by the V-value function). It can be interpreted as what the agent would have received when taking a particular action. If the policy is *not* optimal, then there are state-action pairs with positive advantage (there exists an action with a higher reward). This suggests that the policy should be changed in order to find a better one that returns higher rewards. When the agent already follows the optimal policy, the function will return a negative advantage or 0, because no room for improvement is left [SB18c].

4.6 Components of an RL agent

A distinction can be drawn between the components an RL agent includes.

Component	Type
A representation of a value function returning a forecast of how good the respective state s or state/action pair (s,a) is.	model-free
Representing the policy directly by $\pi(s)$ or $\pi(s,a)$	model-free
Using a model of the corresponding environment as well as a planning algorithm.	model-based

Table 1: Components to learn a policy [FLHI+18]

On the one hand it is possible to exclusively use a model (*model learning*). On the other hand, an agent can make use of a value function or a policy in order to take actions in the environment (*model-free RL*). Figure 4 illustrates the different approaches for an agent to learn a policy.

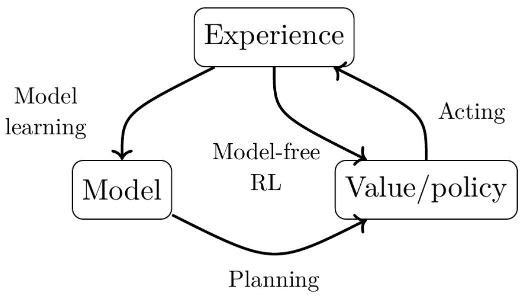

Figure 4: Different ways of learning a policy [FLHI+18]

4.7 General Learning Modes

The process of learning a consecutive decision-making task shows up in two cases [FLHI+18]:

- **Offline**: Just constrained information on a given environment is accessible.
- **Online**: The agent is able to regularly assemble new information in the environment.

In the two cases, the applied calculations are basically equivalent. The special characteristic of offline learning lies – beside the fact of relying on limited data – in the inability of interacting any further with the environment. If the agent is in an environment consisting of constrained information, the idea of generalization has the main priority. The idea behind generalization is that in an RL algorithm an agent is supposed to be able to provide a good performance even when just a low amount of data is available. For further details, the paper from Jonathan Bartlett and Eric Holloway on generalization in machine learning models is recommended [BH19].

5 Exemplary Applications

The following chapters will illustrate the practical use of a V-value and a Q-value function. Moreover, an example of an RL-based adaptive learning system will be demonstrated.

5.1 Q-value Function Example

In the following example (which is mainly based on [Tek05]), the Q-value function was used in order to train an agent to find the best way from an initial state to a goal state. More specifically, figure 5 shows existing states and connections between them. Numbers at edges indicate a reward the agent earns when taking a path. The whole map can be interpreted as a flat or house, where the agent tries to find a way to the goal state (for example going out of the house).

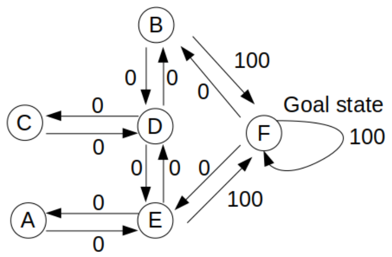

Figure 5: Map of existing paths and corresponding rewards

Moreover, a reward matrix shows the values an agent earns when taking an action in a specific state.

S/A	A	B	C	D	E	F
A	-	-	-	-	0	-
B	-	-	-	0	-	100
C	-	-	-	0	-	-
D	-	0	0	-	0	-
E	0	-	-	0	-	100
F	-	0	-	-	0	100

Figure 6: Reward matrix

Rows in the reward matrix are states, columns are actions. The agent now learns by going to each state and evaluating each action possible in that state. Each result is saved in a Q-table, which can be considered as the agent's *brain*.

S/A	A	B	C	D	E	F
A	-	-	-	-	0	-
B	-	-	-	0	-	0
C	-	-	-	0	-	-
D	-	0	0	-	0	-
E	0	-	-	0	-	0
F	-	0	-	-	0	0

Figure 7: Initial Q-table

Suppose the agent starts in state B and wants to go to F. The Q-value function looks as follows:

$$Q(B,F) = R(B,F) + \gamma \cdot Max[Q(F,B), Q(F,E), Q(F,F)] \quad (12)$$

Now the reward from the reward matrix can be inserted as well as the values from the Q-table. Also, γ is set to 0.8. This leads to the following result.

$$Q(B,F) = 100 + 0.8 \cdot 0 = 100 \quad (13)$$

The result is saved in the Q-table.

S/A	A	B	C	D	E	F
A	-	-	-	-	0	-
B	-	-	-	0	-	100
C	-	-	-	0	-	-
D	-	0	0	-	0	-
E	0	-	-	0	-	0
F	-	0	-	-	0	0

Figure 8: Q-table after one update

Now the agent goes from D to B. The reward is calculated as before:

$$Q(D,B) = 0 + 0.8 \cdot 100 = 80 \quad (14)$$

Again, the Q-table is updated:

S/A	A	B	C	D	E	F
A	-	-	-	-	0	-
B	-	-	-	0	-	100
C	-	-	-	0	-	-
D	-	80	0	-	0	-
E	0	-	-	0	-	0
F	-	0	-	-	0	0

Figure 9: Q-table after two updates

When applying this procedure several times, the values in the Q-table will be constantly updated. At some point (to be precise, it takes 23 iterations) the table gets to a state of convergence, thus the values are not changing any more. One iteration means that the Q values for all states and corresponding actions are calculated once.

S/A	A	B	C	D	E	F
A	-	-	-	-	400	-
B	-	-	-	320	-	500
C	-	-	-	320	-	-
D	-	400	256	-	400	-
E	320	-	-	320	-	500
F	-	400	-	-	400	500

Figure 10: Q-table after 23 iterations

Now the agent is able to navigate from a random state to the goal state since he can just always follow the highest reward. It has to be noted that the numbers in figure 11 are the normalized values of the Q-table from figure 10.

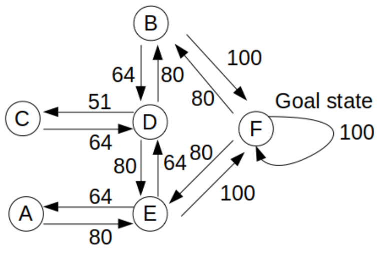

Figure 11: Map of existing paths and corresponding rewards

It can be seen in figure 11 that the rewards are decreasing the further away the agent moves from the goal state. While going towards the goal state lets the reward increase.

5.2 V-value Function Example

This example illustrates an RL problem with the aid of the game tic-tac-toe and is mainly adopted from the time-proven and well-known standard book *Reinforcement Learning* by Richard Sutton and Andrew Barto [SB18c]. Also, this example represents a significantly shortened version of the original and illustrates the use of a value function in RL.

The rules of this game are fairly simple: When having a three-by-three board, one player wins by having three crosses (or Os) in a row – either vertically, horizontally or diagonally. It is a draw if none of the players was able to achieve this and the board is full. Since a good player is able to play in a manner that he or she never loses, the assumption is that an agent is playing against an imperfect player, who therefore not always plays the right move. Another assumption is that draws are as bad as losses. When having these assumptions, the question is how to create a player that maximizes its odds by finding mistakes in its adversary. The goal is to estimate the chances a specific player will make certain moves. However, it is also assumed that these details are not known before the game, but rather become known by playing many games. The first step is to train a model based on the behaviour of the adversary. But the most important operation is to estimate the value function V for a specific policy π. The latter mentioned estimation is also named *prediction problem* [SB18a]. Furthermore, temporal difference methods correspond to the definition of online-policies explained in chapter 4.7. When approaching the tic-tac-toe problem by using a value function a set of numbers are defined representing the current state of the game. Furthermore, every number stands for the latest estimation of the odds for winning the game from a specific perspective.

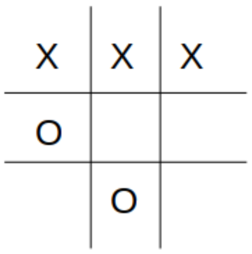

Figure 12: Playing in state s_4 with odds to win by 1

Figure 12 shows that obviously one player has won (the one drawing Xs) – the probability of winning the game as this player is 1.

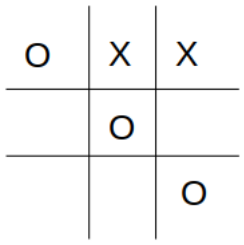

Figure 13: Playing in state s_4 with odds to win by 0

The player drawing the Xs in figure 13 has obviously lost, therefore his chance of winning is 0.

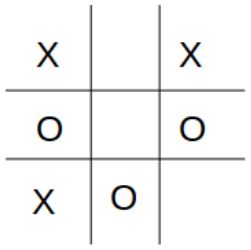

Figure 14: Playing in state s_5 with odds to win by 0.5

In all cases other than shown in the figures 12 and 13 the chances to win is 0.5 as shown in figure 14. The next step is to play several games against the opponent. The upcoming moves are calculated based on the estimation that would result from the potential move. It then obviously makes sense to play moves that result in high estimations of winning. From time to time moves are taken that do not follow the aforementioned pattern, these so-called *exploratory moves* are meant to take the agent to experience states, the agent would not have taken otherwise. Figure 15 shows a diagram of possible tic-tac-toe moves as well as explanatory moves taken instead of *better* moves.

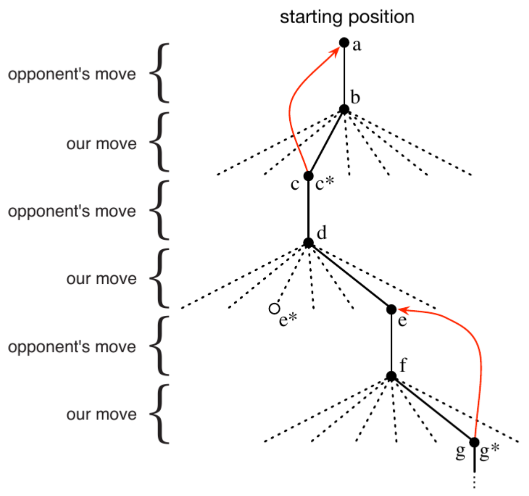

Figure 15: Diagrammed exemplary tic-tac-toe moves [SB18c]

Continuous black lines stand for moves that have been taken while dotted lines show potential moves which have not been taken but considered. The agent opens the game by setting his mark at a. After the opponent's move the RL agent then finds the best estimation when going to c. After the opponent's move from c to d it can be observed that actually e^* has been identified as the best option. In this case, an *exploratory move* was preferred. The agent learns by updating former values (red lines) by moving estimations from the last nodes to former nodes. However, this is only done when performing a *greedy move*. Strictly speaking, the value of the former node is adjusted to be closer to the value of the later node. This learning process can be formally expressed by defining S_t to be the earlier state (i.e. before the *greedy move*) and S_{t+1} is defined as the later move, the update can be formalized as shown in equation 15.

$$V(S_t) \leftarrow V(S_t) + \alpha\Big[V(S_{t+1}) - V(S_t)\Big] \qquad (15)$$

α represents the so-called *step-size parameter* and is used to adjust the learning rate. Due to the reason that this learning approach is based on differences between two consecutive nodes $(V(S_{t+1}) - V(S_t))$ this rule represents an example of *temporal-difference* learning. The explained method performs pretty well when applied in this scenario. If the step-size parameter is decreased periodically the method converges to true odds for winning given a perfect play by the agent. Moreover, if the step-size parameter remains the same, the agent is also able to play against players whose playing style slightly changes over time. For the sake of demonstration purposes this game is enough to illustrate the main aspects of reinforcement learning, which are in this case:

- Learning while interacting
- A specific environment (with an adversary)
- A concrete goal

Furthermore, there is no model or deep search over potential moves that the RL agent is using. It is planning and improving its playing style by a simple update method for former moves which are being followed by greedy moves. In conclusion, it can be said that this game should not lead to the assumption that RL is as limited as this simple game but rather provides a good starting point for gaining an initial understanding of V-value functions and an ϵ-greedy policy.

5.3 Adaptive Learning System

The following example shows how to implement a personalized learning system based on Reinforcement Learning as proposed by Doaa Shawky and Ashraf Badawi [SB18b].

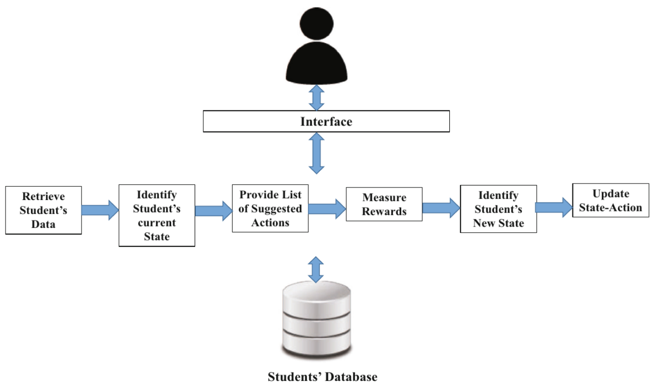
Interface
Retrieve Student's Data
Identify Student's current State
Provide List of Suggested Actions
Measure Rewards
Identify Student's New State
Update State-Action
Students' Database

Figure 16: Framework of a personalized student learning system based on RL [SB18b]

The system works as described in table 2.

Step	Title	Description
1	Retrieve student's data.	Loading the student's dynamic and static information.
2	Identify student's current state.	When a student enters the system the first time, a state that fits the student's static data is used.
3	Provide a list of suggested actions.	Recommending learning material or getting some simple advices from the teacher.
4	Measure rewards.	The reward represents a combination of the student's feedback, the level of interactivity with the system as well as results in exams and assignments.
5	Identify student's new state.	In this state, the student is able to choose among existing states as well as propose new states that should be added to the system (if none of the available states is suitable to describe the student's state).
6	Update state-action	Proposed states are reviews and analysed by an expert and possibly added to the system.

Table 2: Process of a personalized learning system based on RL

Static information as described in step 1 in table 2 can be the student's major, courses or the native language, based on this information the state in step 2 is chosen. If the student had already been logged in before, the last state is used. The reward the agent receives includes a combination of the feedback and the level of interactivity (which are both weighted by 5) as well as exam results. For example, if a student's grade deteriorate from 3.5 to 3 (in the American system), the negative reward is the difference between these grades, which is 0.5. The reward then would be -0.5. As a consequence, suggested actions in step 3 are punished equally by dividing the negative reward of 0.5 by the amount of actions suggested and punishing every action with the mean value (e.g. -0.1 for each action when having 5 action in total). This is due to the fact that the agent simply cannot know which action led to the decline in grades. The main focus is to maximize the reward by choosing the right set of actions for a student in order to increase the student's satisfaction and interactivity level with the system. In the last step, the student is able to choose between available states. If a student thinks that there is no suitable state, the student is able to propose a new one which is then reviewed by an expert. Dynamic information about a student such as the activity level and state-action-reward history are defined and updated every time a student works with the system. In order to choose a suitable state it is therefore inherently important to specify a student's dynamic attributes for the purpose of choosing appropriate set of actions that lead to an increase in the state-action-reward history. Figure 17 shows the RL framework applied to the use case "personalized learning system".

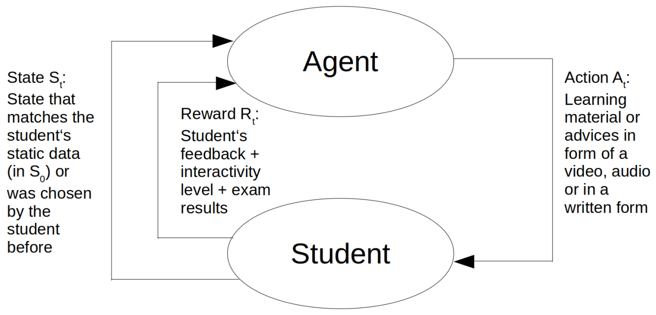
State S_t: State that matches the student's static data (in S_0) or was chosen by the student before
Agent
Reward R_t: Student's feedback + interactivity level + exam results
Action A_t: Learning material or advices in form of a video, audio or in a written form
Student

Figure 17: Diagrammed personalized learning system based on RL

Figure 18 shows simulations of 10 students by using a maximum of 100 iterations under the following conditions:

- Every state consists of 20 actions which leads to a 20 x 20 state-action matrix
- The mentioned matrix is filled with randomly created rewards which are distributed normally (mean = 0, standard deviation: $\sigma = 1$)
- An ϵ-greedy policy is used ($\epsilon = 0.1$)
- The generated rewards consist of 10 negative and 10 positive values
- The learning rate amounts to 0.1

The diagram in figure 18 illustrates that after a certain number of iterations rewards for taken actions increases. The proposed learning system is therefore – when applied often enough – able to increase the educational performance of a student.

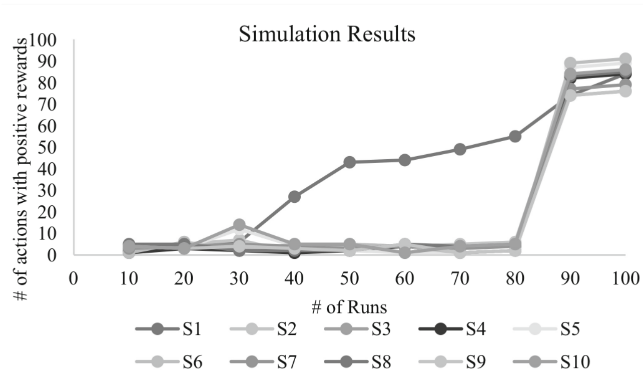

Figure 18: Simulations of an RL based personalized learning system. Iterations versus number of actions with positive reward [SB18a]

The proposed learning system is not necessarily suitable for every student (due to different learning techniques) and for every case/scenario (projects, group work). However, several cases exist in which the implementation of the proposed learning system could indeed record a success.

6 Previous achievements

Various problems in different fields have been tackled in the last 30 years by the use of reinforcement learning approaches and could be solved successfully:

- In 1995 an RL algorithm was able to beat other computer programs in Backgammon [Tes95].
- In 2015, an RL algorithm was able to reach superhuman-level performance by perceiving a large amount of pixels in playing Atari games [MKS+15].
- Davild Silver's team managed to create agents that mastered Go, Chess and Starcraft II [RP20].
- An RL based algorithm beat professional poker players [BSM17].

These achievements demonstrate the possibilities of RL techniques in various fields where high-dimensional vectors are required. Moreover, the high potential in real-world applications apart from games has to be emphasized. Also, successful working RL algorithms in various domains such as smart grids [FL17], autonomous driving [PYWL17] or finance [DBK+16] are already applied in live environments.

7 Future challenges

Applying stand-alone reinforcement learning applications are unlikely to result in breakthrough solutions for complex challenges in the future as the combination of various machine learning techniques proved to be successful in the past. In fact, all major breakthroughs and applications that brought recent success (DeepMind, AlphaGo) are based on stacks of several machine learning applications. For that reason, architectures are often multi-layered and complex and potentially difficult to apply to other applications. Another issue often arises is the agent's restriction in its freedom of movement in the respective environment. This is due to cost and time constrains as well as caused by safety reasons. [FLHI+18] emphasize two main reasons:

- No accurate environment: This problem occurs, when the real environment is not accessible and the agent therefore relies on an inaccurate simulation of the environment. This represents primarily a problem in robotics [ZMK+17, GHLL17]. When applying an environment-simulated trained agent in the real environment, the occurring differences are called reality-gap [JHH95].
- Lack of new observations: The agent simply does not obtain new observations due to a dynamic environment that for example depends on weather conditions (observations can only be acquired by daylight or darkness).

Of course, these problems can also occur at the same time [FLTEF16]. In order to address the aforementioned issues, more accurate simulations have to be developed. Also, algorithms should be designed in a manner to improve generalisation.

8 Conclusion

In this paper, the main concept of reinforcement learning has been introduced. Furthermore, RL techniques were illustrated by the use of specific practical applications. Also, successful work in the field of autonomous driving, finance and smart grids has been made. Recently, complex problems in the game world such as mastering chess, Go and Starcraft II have been successfully addressed by RL agents. Sophisticated simulations of environments along with well-developed combinations of several reinforcement learning tools could solve a great amount of diverse problems in the future or simply lead to more efficient designs. If the aforementioned points are taken on board, it can be anticipated that RL-combined machine learning solutions can be of great help in the upcoming years. Especially the combination of RL and deep learning with smart structures could offer new opportunities to tackle issues on a very abstract level and therefore would make it possible to address even more potential applications.

References

[Ash18] Mohammad Ashraf. The complete reinforcement learning dictionary, 2018. Retrieved 17 May 2020. Accessed at: https://towardsdatascience.comreinforcement-learning-demystified-markov-decision-processes-part-1-bf00dda41690.

[BBBB95] Dimitri P Bertsekas, Dimitri P Bertsekas, Dimitri P Bertsekas, and Dimitri P Bertsekas. *Dynamic programming and optimal control*, volume 1. Athena scientific Belmont, MA, 1995.

[Bel58] Richard Bellman. Dynamic programming and stochastic control processes. *Information and Control*, 1(3):228 – 239, 1958.

[BH19] Jonathan Bartlett and Eric Holloway. Generalized information: A straightforward method for judging machine learning models. *Bartlett, Jonathan and Eric Holloway*, pages 13–21, 2019.

[BNVB13] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

[BSGL09] Shalabh Bhatnagar, Richard S Sutton, Mohammad Ghavamzadeh, and Mark Lee. Natural actor–critic algorithms. *Automatica*, 45(11):2471–2482, 2009.

[BSM17] Noam Brown, Tuomas Sandholm, and Strategic Machine. Libratus: The superhuman ai for no-limit poker. In *IJCAI*, pages 5226–5228, 2017.

[DBK$^+$16] Yue Deng, Feng Bao, Youyong Kong, Zhiquan Ren, and Qionghai Dai. Deep direct reinforcement learning for financial signal representation and trading. *IEEE transactions on neural networks and learning systems*, 28(3):653–664, 2016.

[DBK$^+$17] Y. Deng, F. Bao, Y. Kong, Z. Ren, and Q. Dai. Deep direct reinforcement learning for financial signal representation and trading. *IEEE Transactions on Neural Networks and Learning Systems*, 28(3):653–664, 2017.

[FL17] Vincent François-Lavet. *Contributions to deep reinforcement learning and its applications in smartgrids*. PhD thesis, Université de Liège, Liège, Belgique, 2017.

[FLHI$^+$18] Vincent Franois-Lavet, Peter Henderson, Riashat Islam, Marc G. Bellemare, and Joelle Pineau. An introduction to deep reinforcement learning. *Foundations and Trends in Machine Learning*, 11(3-4):219354, 2018.

[FLTEF16] Vincent François-Lavet, David Taralla, Damien Ernst, and Raphaël Fonteneau. Deep reinforcement learning solutions for energy microgrids management. In *European Workshop on Reinforcement Learning (EWRL 2016)*, 2016.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[GHLL17] Shixiang Gu, Ethan Holly, Timothy Lillicrap, and Sergey Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 3389–3396. IEEE, 2017.

[Gos19] Abhijit Gosavi. A tutorial for reinforcement learning. 2019.

[GPG17] D. Gandhi, L. Pinto, and A. Gupta. Learning to fly by crashing. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3948–3955, 2017.

[Hei19] Hunter Heidenreich. The future with reinforcement learning, 2019. Retrieved 17 May 2020. Accessed at: https://towardsdatascience.com/the-future-with-reinforcement-learning-877a17187d54.

[HH96] Mance E. Harmon and Stephanie S. Harmon. Reinforcement learning: A tutorial, 1996.

[HMLIR07] Verena Heidrich-Meisner, Martin Lauer, Christian Igel, and Martin A Riedmiller. Reinforcement learning in a nutshell. In *ESANN*, pages 277–288, 2007.

[JHH95] Nick Jakobi, Phil Husbands, and Inman Harvey. Noise and the reality gap: The use of simulation in evolutionary robotics. In *European Conference on Artificial Life*, pages 704–720. Springer, 1995.

[LBH15] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[LFDA16] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *J. Mach. Learn. Res.*, 17(1):13341373, January 2016.

[MKS$^+$15] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[MM02] Rémi Munos and Andrew Moore. Variable resolution discretization in optimal control. *Machine learning*, 49(2-3):291–323, 2002.

[MSB$^+$17] Matej Moravčík, Martin Schmid, Neil Burch, Viliam Lisỳ, Dustin Morrill, Nolan Bard, Trevor Davis, Kevin Waugh, Michael Johanson, and Michael Bowling. Deepstack: Expert-level artificial intelligence in heads-up no-limit poker. *Science*, 356(6337):508–513, 2017.

[NN98] James R Norris and James Robert Norris. *Markov chains*. Number 2. Cambridge university press, 1998.

[PAW$^+$17] Lerrel Pinto, Marcin Andrychowicz, Peter Welinder, Wojciech Zaremba, and Pieter Abbeel. Asymmetric actor critic for image-based robot learning, 2017.

[PYWL17] Xinlei Pan, Yurong You, Ziyan Wang, and Cewu Lu. Virtual to real reinforcement learning for autonomous driving, 2017.

[RP20] Sebastian Risi and Mike Preuss. Behind deepminds alphastar ai that reached grandmaster level in starcraft ii. *KI-Künstliche Intelligenz*, 34(1):85–86, 2020.

[SB18a] Doaa Shawky and Ashraf Badawi. A reinforcement learning-based adaptive learning system. In *International Conference on Advanced Machine Learning Technologies and Applications*, pages 221–231. Springer, 2018.

[SB18b] Doaa Shawky and Ashraf Badawi. *A Reinforcement Learning-Based Adaptive Learning System*, pages 221–231. 01 2018.

[SB18c] Richard Sutton and Andrew Barto. Reinforcement learning. In *Reinforcement Learning*, pages 1–400, 2018.

[Sch15] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

[SHM$^+$16] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

[Tek05] Kardi Teknomo. Q-learning numerical example, 2005. Retrieved 17 May 2020. Accessed at: https://people.revoledu.com/kardi-tutorial/ReinforcementLearning/Q-Learning-Example.htm.

[Tes95] Gerald Tesauro. Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68, 1995.

[ZMK$^+$17] Yuke Zhu, Roozbeh Mottaghi, Eric Kolve, Joseph J Lim, Abhinav Gupta, Li Fei-Fei, and Ali Farhadi. Target-driven visual navigation in indoor scenes using deep reinforcement learning. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 3357–3364. IEEE, 2017.

[Zyc19] Shaked Zychlinski. The complete reinforcement learning dictionary, 2019. Retrieved 17 May 2020. Accessed at: https://towardsdatascience.com/the-complete-reinforcement-learning-dictionary-e16230b7d24e.