APPENDIX A

Reinforcement Learning

This Appendix provides an overview and definition of Reinforcement Learning.

RL is located between supervised and unsupervised learning to learn what to do to maximize a numerical reward signal [1]. The learner does not know what actions to take to reach the goal of maximizing the reward signal and only can pick and try actions to detect those increasing the accumulative reward [2].

Reinforcement learning algorithms are defined in an iterative way not by characterizing learning methods, but by characterizing a learning problem. The agent and the environment are interacting continually: the agent selects actions and the environment responds to the actions, presents new states to the agent which gives rise to rewards. This cycle is repeated as part of a Markov Decision Process (MDP) [1], [3]. MDPs are used as stochastic extensions of finite automata or Markovian process to model the decision making process and solve optimizing problems. They are augmented by actions and rewards so that they consist of actions, transitions, labels, and states. In the following paragraphs, some definitions are introduced that are helpful in demonstrating our proposed approach in the next section.

Definition 1. *Markov decision process (MDP).*

MDPs provide a mathematical framework to model the decision making process in situations where outcomes are partly random and partly under the control of a decision maker. They are useful in addressing a wide range of optimization problems solved via dynamic programming and reinforcement learning [1]. MDPs are an extension of Markov chains; the difference is the addition of actions (allowing choice) and rewards (giving motivation). Conversely, if only one action exists for each state and all rewards are the same (e.g. "zero"), an MDP reduces to a Markov chain.

More precisely, an MDP is a discrete time stochastic control process. In other words, an MDP contains:

1. A set of possible states S.
2. A set of possible actions A.
3. Transition function, $T: S \times A \times S \rightarrow [0,1]$ giving for each state and action. It computes the probability of reaching state s' by performing action a in state s and is denoted as $T(s, a, s')$, $s, s' \in S$, $a \in A$.
4. Reward function, $R(s, a, s')$ specifies rewards for transitioning from state s to state s' due to action a.

Therefore, an MDP has a set of states. These states will play the role of outcomes in the decision making approach as well as providing information, which is necessary for choosing actions. For example, in the case of a robot navigating through a building, the state might be the room it is in, or the x and y coordinates. For a factory controller, it might be the temperature and pressure in the boiler.

An MDP also has a set of actions. At each time step, the process is in state s, and the decision maker may choose any action a that is available in state s. The process responds at the next time-step by randomly moving into a new state s', and giving the decision maker a corresponding reward $R(s, a, s')$.

In order to indicate the order of different states and actions during agent and environment interaction, they should be denoted according to the time at which they occur. So, $s_t \in S$ denotes the state at time t [4]; according to this definition of a Markovian process, we would have:

$$P(s_{t+1}|s_t, s_{t-1}, s_{t-2}, \dots) = P(s_{t+1}|s_t) = T(s_t, a_t, s_{t+1}) \quad (1)$$

So, it specifies the probability that the next state will be s_{t+1} for each state $s_t \in S$ and action $a_t \in A$. The probability that the process moves into its new state is influenced by the chosen action.

This process is called Markov, because it has what is known as the Markov property; that is, the next state s' depends on the current state s and the decision maker's action a. Therefore, the next state is independent of all the previous states and actions. The current state captures all that is relevant about the world in order to predict what the next state will be.

Finally, there is the Reward function. The agent's utility is defined by the reward function. The goal of the reward function is to specify the reward to each action that the agent performing. So, $R: S \times A \times S \rightarrow \mathbb{R}$ gives rewards for particular transitions between states. The reward function plays an important role in an MDP. At each discrete time, an agent observes state $s_t \in S$ and chooses action $a_t \in A$ and receives immediate reward $r_t \in R$. Then state changes to s_{t+1}. The immediate reward indicates the immediate feedback value of reaching a certain state. An agent tries to maximize its total reward in long term view. So an MDP can be denoted by the tuple $\langle S, A, T, R \rangle$ depicting it as a state transition graph [3].

Definition 2. *Policy*

The goal in an MDP is to find a function, called a policy, which determines which action to take in each state, so as to maximize the reward function. Policy π gives the probability of taking action a when in state s:

$$\pi: S \times A \rightarrow [0,1]$$

$$\pi(s, a) = P(a_t = a|s_t = s) \quad (2)$$

Definition 3. *Next-State Function*

At each discrete time, an agent receives an immediate reward r_t by choosing action a_t from state s_t and then the state changes to s_{t+1}. Markov assumes that $s_{t+1} = \delta(s_t, a_t)$ and $r_t = R(s_t, a_t, s_{t+1})$. r_t and s_{t+1} depend only on current state and action. The next-state function δ for state s, maps the state s at time t to the state at time t+1 that follows the action. The value of reaching state s through action a is computed using the action-value function which is defined in the following paragraph. The next-state function δ and reward function R may be nondeterministic and not necessarily known to the agent.

If next-state function is known, dynamic programming can be used to learn value functions like V(S), defined below, in order to obtaining the policy. If next-state function is unknown, Q-Learning can be used to learn action-value functions like Q(S, A). The value function is described in the following paragraphs.

Definition 4. *State-Value Function*

The state-value-function returns the value, i.e. the expected return for selecting a certain state s. Return means the overall reward. We can define the value of a state under a policy π, formally $V^{\pi}(s)$, as [3]:

$$V^{\pi}(s) = E_{\pi}\{ r_t + r_{t+1} + r_{t+2} + \cdots | s_t = s\} \quad (3)$$

The sum in equation (3) is typically infinite. In a more realistic setting, rewards in the future get discounted, and equation (3) becomes [2]:

$$V^{\pi}(s) = E_{\pi}\{r_t + \gamma\, r_{t+1} + \gamma^2\, r_{t+2} + \cdots | s_t = s\} = E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \middle| s_t = s\right\} ,\gamma \in [0,1] \quad (4)$$

Where E_{π} is the expected value by following policy π and discount factor γ, which determines the importance of future rewards. The state-value function $V^{\pi}(s)$ specifies the value of a state is equal to the total amount of rewards a learner can accumulate, starting from that state. $\gamma = 0$ indicates that it is only concerned about immediate rewards.

Definition 5. *State-Action Value Function*

The state-action value function returns the value, i.e. the expected return for using action a in a certain state s as:

$$Q^{\pi}(s,a) = E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \middle| s_t = s, a_t = a\right\} \ , \gamma \in [0,1] \quad (5)$$

Where E_{π} is the expected value by policy π.

The Bellman-Equation expresses the relationship between the value of a state and the value of a successor state.

$$V^{\pi}(s) = \sum_{a\in A} \pi(s,a) \sum_{s'\in S} T(s,a,s')\left(R(s,a,s') + \gamma V^{\pi}(s')\right) \quad (6)$$

Where R is the reward for passing from state s to successor state s'. $V^{\pi}(s')$ returns the state-value of the successor state s'. T is the probability to access state s' when choosing action a in state s. $\pi(s,a)$ is the probability of choosing action a in state s. The Bellman-Equation calculates the value of a state by considering all available options and assessing each by its likelihood of appearance.

The aim of an agent is to find the optimal policy. There is at least always one policy that is better or equal than all other policies. This policy is called the optimal policy π^*.

$$\pi^* = \arg\max_{\pi} V^{\pi}(s) \ ,\forall(s) \quad (7)$$

There is only one optimal-value-function, which is the base for an optimal policy. The optimal value-function chooses an action a in a state s not by following a policy but by choosing the maximum. We attempt to learn the value function of the optimal policy π^*, denoted by V^{π^*} (which we write as V^*). It can then do a look-ahead search to choose the best action from any state s. The optimal-value-function can be noted as follows:

$$V^*(s) = \max_{a\in A} \sum_{s'\in S} T(s,a,s') \left(R(s,a,s') + \gamma V^*(s')\right) \quad (8)$$

As aforementioned, this works well if agent knows next-state function $\delta: S \times A \to S$, and $R: S \times A \times S \to \mathbb{R}$. When the agent does not know δ, the Q-function which is very similar to V^* is defined as:

$$Q^{\pi}(s,a) = R(s,a,s') + \gamma V^*(s') \quad (9)$$

If the agent learns Q, it can choose an optimal action even without knowing δ.

$$\pi^*(s) = \arg\max_{a} [Q(s,a)] \quad (10)$$

Where Q is the action-value function the agent will learn. Q and V^* are closely related as:

$$V^*(s) = \max_{a} Q(s,a) \quad (11)$$

This allows us to write Q recursively as:

$$Q^{\pi}(s_t,a_t) = R(s_t,a_t,s_{t+1}) + \gamma V^*\left(\delta(s_t,a_t)\right) = R(s_t,a_t,s_{t+1}) + \gamma \max_{a} Q(s_{t+1},a) \quad (12)$$

All RL-based algorithms are based upon providing an approach for appropriately estimating state-action value functions. This has led to the exploration and production of several different estimating methods and techniques. One of the most popular of these is Q-Learning [5], which is an off-policy Temporal Difference control algorithm. In other words, Q-Learning is able to estimate Q-value functions (Q-learning based estimations of the state-action value function) without requiring an initial model of the environment.

Additionally, Q-learning can handle problems with stochastic transitions and rewards without requiring any adaptations. It has been proven that for any finite MDP, Q-learning eventually finds an optimal policy. This means that the expected value of the total reward that has been returned over all successive steps is the maximum achievable.

In this situation, because of the lack of known transition and reward models, the algorithm handles problems with stochastic transitions and rewards and uses exploration and sampling approaches to learn the required model. Therefore, Q-learning finds an optimal policy for any finite MDP and estimates the agent's Q-value function based on the Q-value estimation of an action. In other words, using the above definitions equation (13) is inferred. This process is incrementally evaluated as follows [3]:

$$Q_{k+1}(s_t,a_t) = Q_k(s_t,a_t) + \alpha\left(r_t + \gamma \max_{a} Q_k(s_{t+1},a) - Q_k(s_t,a_t)\right) \quad (13)$$

Where, α $(0 < \alpha \leq 1)$ is the learning rate, which determines the extent to which new information can override old information and how fast we modify our estimates [6].

REFRENCES

[1] C. Szepesvári, "Algorithms for Reinforcement Learning, Synthesis Lectures on Artificial Intelligence and Machine Learning" vol. 4, 2010. Morgan & Claypool Publishers

[2] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, Massachusetts: The MIT Press, 1998.

[3] M. Wiering and M. v. Otterlo, *Reinforcement Learning (State of the Art)* Springer, 2012.

[4] L. Long-Ji, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Machine Learning,* journal article vol. 8, no. 3, pp. 293-321, May 01 1992.

[5] C. J. Watkin, "Learning From Delayed Rewards," PhD, King's College, University of Cambridge, 1989.4

[6] P. Dayan and C. J. Watkin, "Technical Note Q,-Learning," *Machine Learning* vol. 8, no. 3, pp. 279-292 1992. Springer Netherlands