An In-Depth Analysis of Reinforcement Learning

William Connor Bean

December 6th, 2020

Abstract

This report will discuss the reinforcement learning model and defining it in terms of a Markov Decision Process, with an emphasis on model-based as well as model-free learning methods. We will then discuss how agents can learn about their environment by comparing an exploration vs. exploitation strategy as it relates to Q-Learning. Finally, we will discuss reward shaping and its role in allowing an agent to quickly learn how to make progress toward a goal.

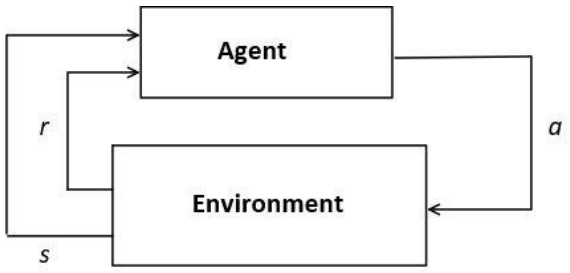
1. Introduction

Reinforcement learning defines problems in which agents explore an environment [Stone, 2017]. Agents make a series of sequential steps, receiving rewards at each step for positive or negative behaviors [Stone, 2017]. Positive and negative behaviors are usually defined in terms of progress made toward the goal state [Wiewiora, 2017].

The goal of any reinforcement learning algorithm is to determine a policy [Stone, 2017]. The policy defines what action to take in a given state within the environment such that the policy maximizes the long-term reward [Even-Dar, 2008]. This differs from similar machine learning techniques such as supervised learning in that the agent must receive a reward (or punishment) from the environment based on a chosen action [Qiang and Zhongli, 2011]. This means that an agent does not know the difference between a positive and negative behavior before performing an action. In supervised learning, an agent has access to examples of input and output values, providing it insight into behaviors that are more likely to lead to the goal state without first exploring its environment [Stone, 2017].

2. Reinforcement Learning Model

The reinforcement learning model is comprised of 5 components: the agent, the environment, an action *a*, a reward signal *r*, and a transition to state *s* [Pandey and Pandey, 2010].



[Pandey and Pandey, 2010]

In the above model, the agent selects an action *a* [Pandey and Pandey, 2010]. The action signal is sent to the environment, to which the environment outputs a new state *s* for the agent to transition into [Qiang and Zhongli, 2011]. Furthermore, based on the desired goal state, the environment will also output a reward/punishment signal *r* [Qiang and Zhongli, 2011]. The signal *r* allows the agent to learn about how effective their action choice was at reaching the goal state.

The goal of any reinforcement learning system is to determine an action sequence, known as a policy, that leads it to the goal state such that the agent maximizes the long-term reward received from the environment [Pandey and Pandey, 2010]. This can be defined with the following formula, where γ is the discount rate [Qiang and Zhongli, 2011]:

$$Reward = \sum_{i=0}^{\infty} \gamma^i r_i \ \ where\ y \in (0, 1]$$

The equation describes that if the agent receives a positive reward at step *i* for selecting an action *a* and transitioning from state *s* to state s', the agent will strengthen this trend [Qiang and Zhongli, 2011]. This makes it more likely for the agent to select the same action when presented with the same state in the future [Pandey and Pandey, 2010]. Likewise, negative rewards (or punishments) will decrease this trend, making it less likely for the agent to select action *a* at state *s* [Qiang and Zhongli, 2011].

3. Markov Decision Process

We can formalize our basic reinforcement learning model above by defining a Markov Decision Process (MDP). We can define an MDP as a 4-tuple *M = <S, A, P, R>,* where *S* is the set of all possible states in the environment, *A* is the set of all possible actions, $P_{s's}^{a}$, is the probably of transitioning from state *s* to state *s'* when applying action $a \in A$, and *R(s, a)* is the reward received from the environment when applying action *a* in state *s* [Even-Dar, 2008]. In our above definition, *M*, *P* and *R* are said to be Markovian as they do not rely on any previous states or actions, but rather only the current state and action [Uther, 2017].

Markov Decision Processes are defined in order to find the optimal set of actions to take given the environment, also known as a policy π [Uther, 2017]. A Markov policy is defined as $S \rightarrow A$, meaning that the policy π selects the optimal action to take for each state [Uther, 2017]. This is a local optimality, meaning that an action is not selected based on the global state set *S*, but rather whatever action is most optimal for the current individual state $s \in S$ [Uther, 2017]. This is where our reward equation from above comes into play. We want our policy π to maximize the long-term sum of rewards received at each state [Uther, 2017]. We apply the

discount rate γ in order to smooth the learning and not let any individual *R(s, a)* dictate the policy, but rather let the policy learn through trial-and-error [Qiang and Zhongli, 2011].

3.1 Model-Based MDP

Reinforcement learning methods that use an MDP model generally take two forms: Model-Based/On-Policy methods and Model-free/Off-Policy methods [Qiang and Zhongli, 2011]. Before we begin discussing the model-based approach, lets first define what we mean by "model". The "model" of the MDP refers to the environment a reinforcement learning agent is presented with. More concretely, the "model" is an understanding of the probability of transitioning from one state to another and the probability that a given action is taken [Sewak, 2019]. A reinforcement learning algorithm that first generates an MDP model and uses that model to determine the optimal policy is said to be a model-based reinforcement learning algorithm [Qiang and Zhongli, 2011]. Simply put, model-based methods rely on using the same mechanism for selecting an action as they do to update the policy [Sewak, 2019].

3.2 Model-Free MDP

On the other hand, model-free approaches do not require any knowledge of the MDP "model" in order to find the optimal policy [Sewak, 2019]. Relating this to our discussion above, model-free approaches will use a similar mechanism to model-based approaches for updating the policy itself, however, the mechanism used to determine agent behavior (i.e. choosing an action in a given state) will differ [Sewak, 2019]. One such mechanism, which is used by the Q-Learning algorithm, is known as an epsilon-greedy strategy [Sewak, 2019]. We will discuss this mechanism in detail in a later section of this paper.

4. Temporal Difference Learning

Before we begin discussing the SARSA and Q-Learning algorithm, we need to touch on a few other topics. Firstly, we need to discuss the difference between estimation and control problems in reinforcement learning, as well as a strategy for the estimation subproblem. The estimation and control subproblems are present in any situation in which an agent acts based on a value function that optimizes or maximizes [Sewak, 2019]. The control subproblem refers to the agent deciding what action to take in a given state, while the estimation subproblem refers to updating/determining the value function (i.e. the policy) [Sewak, 2019].

Temporal difference learning is a strategy that solves the estimation subproblem [Sewak, 2019]. We can formalize how temporal difference learning updates the value function with the following definition:

$$V(s) = V(s_t) + \alpha(r_t + \gamma V(s_{t+1}) - V(s_t))$$

[Uther, 2011]

The equation above represents updating an MDP with a finite number of states [Uther, 2011]. We define each state with an entry in the array V, updating our value function at each time step *t* [Uther, 2011]. Thus, our value function at state *s* gets updated with the previous state, s_t [Uther, 2011]. We then add the difference between the instantaneous reward received at time step *t,* r_t, plus the immediate next state at a discounted rate, $\gamma V(s_{t+1})$, and the previous state, s_t [Sewak, 2019].

We can denote the difference between $r_t + \gamma V(s_{t+1})$ and $V(s_t)$ as:

$$\Delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$$

[Uther, 2011]

You will notice that Δ_t is multiplied by a constant α in our equation for updating the value function. This constant is known as the learning rate, where $\alpha \in (0, 1]$ [Uther, 2011]. Due to variances in data, the value at the immediate next state $V(s_{t+1})$ can sometimes vary significantly from the value at the previous state $V(s_t)$ [Sewak, 2019]. The learning rate, α, provides a smoothing effect to the learning as to not let significant data variances disadvantageously affect the learning process [Sewak, 2019].

5. SARSA Algorithm

The State-Action-Reward-State-Action (SARSA) algorithm follows a similar pattern to updating the value function as we discussed in section 4 [Sewak, 2019]. In this case, the value function is defined as an action-value function, denoted as *Q* [Sewak, 2019]. The Q function is updated at each time step and can be formalized with the following formula [Qiang and Zhongli, 2011]:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha[r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

[Qiang and Zhongli, 2011]

Comparing the above equation to our value function equation from section 4, we can see that the only difference between the two is that our action-value function (i.e. Q function) acts on two properties; the state *s* and action *a,* at time step *t* [Sewak, 2019].

This covers the estimation subproblem we discussed in the section on temporal difference learning. So, the next question is, how does SARSA handle the control subproblem? More concretely, how does SARSA select what action to take in state *s* at time step *t*?

The SARSA algorithm is a model-based (i.e. on-policy) approach [Sewak, 2019]. Recall the discussion on model-based MDPs, we mentioned that a model-based MDP uses the same mechanism for updating the policy as it does for selecting an action [Sewak, 2019]. In terms of the SARSA algorithm, this refers to the Q function. Therefore, the actions selected are directly based on the values in Q [Sewak, 2019]. Often the largest action value in Q at state *s* is selected [Uther, 2017]. This method requires selecting good initial values of Q, as poor initial values could cause the agent to exploit these actions repeatedly, not allowing for the other Q values to be updated [Sewak, 2019]. This can result in poor learning performance. "Good" initial values are often extremely low values, that result in no initial bias by the agent [Sewak, 2019].

6. Q-Learning

Q-Learning is an off-policy or model-free method of reinforcement learning that is similar to SARSA, but has some significant differences [Pandey and Pandey, 2010]. Let us first look at how Q-Learning updates the Q function by analyzing the following equation:

$$Q(s,a) = Q(s,a) + \alpha[r + \gamma max_{a' \epsilon A(s')} Q(s',a') - Q(s,a)]$$

[Notsu et al., 2011]

The main difference between Q-Learning and SARSA in updating the Q function is this specific part: $max_{a' \epsilon A(s')} Q(s',a')$. This tells the update function to select the maximum next action from all the possible next actions from the next state [Sewak, 2019]. Q-Learning does

this because it employs an exploitation and exploration approach [Sewak, 2019]. Therefore, the maximum next action is selected as to concentrate on exploiting learned knowledge from the Q function when the algorithm decides to exploit rather than explore [Sewak, 2019]. In the next section we will discuss one approach to deciding between exploiting learned knowledge and exploring the environment.

Recall our discussion on off-policy approaches above, unlike SARSA, Q-Learning does not use the same mechanism to update the Q function as it does to guide behavior of an agent [Sewak, 2019]. This means that Q-Learning can calculate an optimal Q function without any knowledge of the environment [Notsu et al., 2011]. One benefit of Q-Learning over SARSA is that since the agent does not need any knowledge of its environment, the Q table can be initialized to 0's [Sewak, 2019]. This further allows such an agent to converge on an optimal policy regardless of the success of the action-value update function [Sewak, 2019]. This is because the mechanism to allow the agent to explore its environment is disjoint from the action-value update function [Sewak, 2019].

6.1 Epsilon-Greedy Strategy

Q-Learning uses an exploration and exploitation approach that we briefly discussed in the previous section. One such approach to balancing the agent between exploring its environment and exploiting the Q function is called the epsilon-greedy strategy [Beysolow II, 2019]. This approach can be implemented in various forms, but the general idea stays the same in all implementations.

A constant ε (epsilon) is chosen to represent the probability of an agent selecting a random action *a* to take in state *s* (exploration) vs. using the Q function to determine the action to take in state *s* (exploitation) [Sewak, 2019]. Epsilon is a decimal value such that $\varepsilon \, \epsilon \, (0, 1)$ [Beysolow II, 2019]. For example, if $\varepsilon = 0.7$, this would mean that there is a 70% probability that the agent would choose a random action *a* in state *s* and a 30% probability of exploiting learned knowledge by using the Q function to determine the next action [Sewak, 2019]. Therefore, the strategy becomes to explore when the agent does not know much about its environment, and to exploit when the agent has learned about its environment [Sewak, 2019].

The question then becomes, at what stage of the learning process do we begin exploiting over exploring? This paper is accompanied by a python implementation of Q-Learning using the OpenAPI Gym library with the FrozenLake-v0 environment. The strategy used for balancing exploration and exploitation is an exponential one, where the value of ε decays exponentially with respect to the episode of the training session. This means that in the initial episodes of training, the agent will mostly choose a random action to take in a given state. While in the later episodes of training it will almost always exploit the learned information, with only a 1% probability of selecting a random action. This approach is similar to the annealing epsilon approach discussed by Mohit Sewak in *Deep Reinforcement Learning* [Sewak, 2019].

7. Reward Shaping

Reward shaping refers to providing an agent with frequent feedback in order to strengthen positive behavior and discourage negative behavior [Wiewiora, 2017]. In small domains, providing the agent with reward signals for simply reaching the goal state is sufficient

[Wiewiora, 2017]. However, in larger domains, more frequent rewards are required so that the agent can understand promising behaviors that are likely to lead it to the goal state [Wiewiora, 2017]. Take a game of chess for example, rewarding the agent for simply winning a game of chess would result in ineffective learning as coming across a strategy that results in a win would take a significant amount of time [Wiewiora, 2017]. A frequent award strategy could positively reward the agent for each piece it takes from the opponent, and similarly apply a negative reward signal for each piece taken by the opponent [Wiewiora, 2017]. While this can result in distracting the agent, it is necessary to provide these frequent rewards in such large domains to prevent the agent from trying unsuccessful strategies over and over (also known as "thrashing") [Wiewiora, 2017].

In terms of the Q-Learning implementation provided alongside this report, a reward signal of +1 is supplied to the agent when it reaches the goal state. This is sufficient given the small domain that is the FrozenLake-v0 environment. The other scenarios include when the agent falls into a hole and if the agent does not fall into a hole or find the goal state in 100-time steps. In both cases, no reward signal is provided. This proves to be quite a successful strategy after averaging the rewards received per 1000 episodes of training, over a 10,000 episodes training session. The average reward received in the first 1000 episodes of training is approximately 0.053, meaning that the agent only finds the goal state about 5% of the time at the beginning of training. However, after 10,000 episodes of training the agent is receiving an average reward of approximately 0.695, meaning that it reaches the goal state about 70% of the time. This is a significant improvement given that the agent knew nothing about the environment at the start of its training.

8. Conclusion

To conclude, reinforcement learning defines problems in which agents explore an environment, making a series of sequential steps or actions, and receiving rewards at each step [Stone, 2017]. The goal being to converge on an optimal policy that defines the what actions to take in each state, maximizing the long-term reward [Even-Dar, 2008]. Reinforcement learning problems can be defined in terms of a Markov Decision Process (MDP), consisting of a set of possible states, a set of possible actions, the probability of transitioning from one state to another when applying an action, and a reward signal [Even-Dar, 2008]. MDPs can be broken down into model-based and model-free approaches, where model-based approaches rely on using the same mechanism for selecting an action as they to do to update the policy, and model-free approach rely on different mechanisms [Sewak, 2019]. SARSA is one such model-based algorithm, while Q-Learning is a model-free algorithm [Sewak, 2019].

In real world scenarios, the domain for reinforcement learning problems are often large. Therefore, to maximize learning, frequent rewards are provided to agent so that the agent can understand behavior that will likely lead it to its goal state [Wiewiora, 2017]. This has the added benefit of minimizing the potential of "thrashing" [Wiewiora, 2017]. This can be experimented with by modifying the reward signal in the Q-Learning implementation provided with this paper.

References

[Stone, 2017] Stone P. (2017) Reinforcement Learning. In: Sammut C., Webb G.I. (eds) *Encyclopedia of Machine Learning and Data Mining*. Springer, Boston, MA. https://doi-org.uml.idm.oclc.org/10.1007/978-1-4899-7687-1_720

[Even-Dar, 2008] Even-Dar E. (2008) Reinforcement Learning. In: Kao MY. (eds) *Encyclopedia of Algorithms*. Springer, Boston, MA. https://doi-org.uml.idm.oclc.org/10.1007/978-0-387-30162-4_341

[Qiang and Zhongli, 2011] W. Qiang and Z. Zhongli, "Reinforcement learning model, algorithms and its application," 2011 *International Conference on Mechatronic Science*, Electric Engineering and Computer (MEC), Jilin, 2011, pp. 1143-1146, doi: 10.1109/MEC.2011.6025669.

[Pandey and Pandey, 2010] D. Pandey and P. Pandey, "Approximate Q-Learning: An Introduction," 2010 Second *International Conference on Machine Learning and Computing*, Bangalore, 2010, pp. 317-320, doi: 10.1109/ICMLC.2010.38.

[Uther, 2017] Uther W. (2017) Markov Decision Processes. In: Sammut C., Webb G.I. (eds) *Encyclopedia of Machine Learning and Data Mining*. Springer, Boston, MA. https://doi-org.uml.idm.oclc.org/10.1007/978-1-4899-7687-1_512

[Uther, 2017] Uther . (2017) Temporal Difference Learning. In: Sammut C., Webb G.I. (eds) *Encyclopedia of Machine Learning and Data Mining*. Springer, Boston, MA. https://doi-org.uml.idm.oclc.org/10.1007/978-1-4899-7687-1_817

[Sewak, 2019] Sewak M. (2019) Temporal Difference Learning, SARSA, and Q-Learning. In: *Deep Reinforcement Learning*. Springer, Singapore. https://doi-org.uml.idm.oclc.org/10.1007/978-981-13-8285-7_4

[Uther, 2011] Uther W. (2011) Temporal Difference Learning. In: Sammut C., Webb G.I. (eds) *Encyclopedia of Machine Learning*. Springer, Boston, MA. https://doi-org.uml.idm.oclc.org/10.1007/978-0-387-30164-8_817

[Notsu et al., 2011] A. Notsu, K. Honda, H. Ichihashi and Y. Komori, "Simple Reinforcement Learning for Small-Memory Agent," 2011 10th *International Conference on Machine Learning and Applications and Workshops*, Honolulu, HI, 2011, pp. 458-461, doi: 10.1109/ICMLA.2011.127.

[Beysolow II, 2019] Beysolow II T. (2019) Reinforcement Learning Algorithms: Q Learning and Its Variants. In: *Applied Reinforcement Learning with Python*. Apress, Berkeley, CA. https://doi-org.uml.idm.oclc.org/10.1007/978-1-4842-5127-0_3

[Wiewiora, 2017] Wiewiora E. (2017) Reward Shaping. In: Sammut C., Webb G.I. (eds) *Encyclopedia of Machine Learning and Data Mining*. Springer, Boston, MA. https://doi-org.uml.idm.oclc.org/10.1007/978-1-4899-7687-1_966