Decision-Making: Q-learning to Select Effective Policies for Intelligent Agents

Ji Min Ryu 7939825

Comp 3190 Paper

Dec 4, 2023

Contents

1 Abstract

This paper introduces fundamental reinforcement learning concepts like policies and the Markov Decision Process. It also explores the extension of these ideas to Q-Learning and its application in selecting effective policies for intelligent agents. We compare the challenges in Q-Learning design and offer detailed insights into potential algorithm improvements. The paper also delves into policy formation using Q-Learning and addresses the potential issues in this reinforcement learning approach. Designing policies for intelligent agents in dynamic domains remains a persistent challenge that we aim to improve, this is a challenge because of the large number of states and actions imposed by dynamic environments. This article discusses Deep-Q networks as a potential solution. We will also highlight a coding program that I wrote, which implements Q-Learning in a static environment and prints out the average reward as the number of runs goes on, showing convergence to an effective policy.

2 Introduction

2.1 Reinforcement Learning

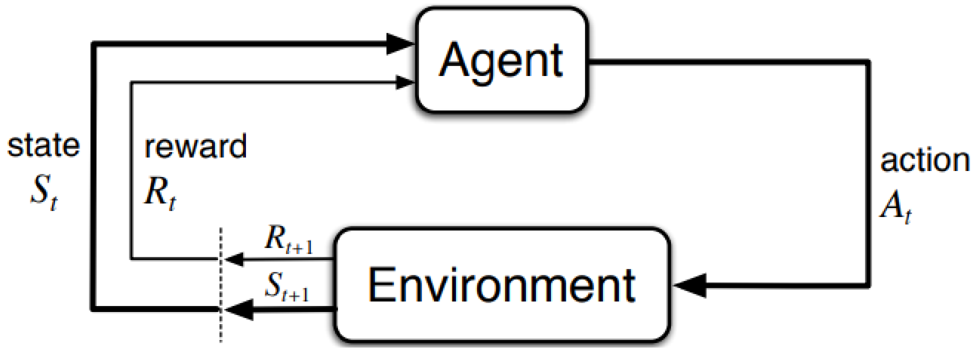
Reinforcement learning is defined as learning so as to maximize a numerical reward value [Sutton and Barto, 2020]. A set of actions is not given to the learner, so the learner must discover which actions reap the most rewards by attempting them [Sutton and Barto, 2020]. This aspect of search distinguishes reinforcement learning from supervised learning, which uses a static set of labeled examples to guide the learning process [Naeem et al., 2020]. Although search is a key part of reinforcement learning, it is not the primary goal. Maximization of a numerical reward signal is the desired outcome in reinforcement learning, not, finding a hidden structure or pattern through search [Sutton and Barto, 2020]. Therefore, reinforcement learning should not be confused with unsupervised learning, where those tasks are prioritized [Sutton and Barto, 2020]. Reinforcement learning is a crucial area of study in AI because it is the type of learning that most animals (including humans) perform, based off a reward system while balancing the trade-off of different actions. One area of reinforcement learning that we will explore is Q-learning, an off-policy learning algorithm that learns to find an optimal policy by observing different policies that are possibly sub-optimal compared to the current policy of the agent using Q-learning [Naeem et al., 2020].

2.2 Defining Intelligent Agents

The system that learns and makes decisions is known as the "agent" [Sutton and Barto, 2020]. As for what makes the agent intelligent, although a broad topic itself, can be viewed simply as the degree of decisions that the agent makes, whether that decision be found through search or be chosen from a lack of search. Intelligent agents in terms of artificial intelligence must have methods for learning and searching through a given or unknown environment. One theoretical framework used to guide intelligent agents for modeling decisions is the Markov Decision Process.

2.3 Markov Decision Process

Modeling decisions where outcomes are partially random and partially under the control of a decision maker requires a mathematical framework known as the Markov Decision Process, or MDP for short. The goal of the MDP is to give a mapping of optimal actions for each state of a domain [Naeem et al., 2020]. MDP is based off the Markovian property which disregards past information in lieu of the present information [Naeem et al., 2020]. MDP's consist of four elements: a set of states, a set of actions, a reward function, and a state transition function [Kaelbling et al., 1996]. State transition functions specifies a new state of the domain as a function of its current domain and the action that was taken by the agent [Kaelbling et al., 1996]. Models of decisions are considered Markovian if the state transitions are independent of previous states or actions [Kaelbling et al., 1996]. We can see the simple interaction process between agents and their domain if we define a set of states, rewards, and actions as S, R and A respectively where t is discrete time steps and follow the diagram below.



[Sutton and Barto, 2020].

We can see from the diagram above that the agent receives the current state S_t and a reward value R_t and selects an action A_t based on the understanding of the current state and reward given for its previous action. A new state S_{t+1} is found as a result of the agents' previous action A_t

and a corresponding numerical reward R_{t+1} is calculated. The new action taken may be influenced by the previous reward given but since each transitioned state is independent of the previous, this process is Markovian [Kaelbling et al., 1996].

2.4 Defining Policies

In the context of artificial intelligence, a policy is defined as a set of instructions with actions to choose in different situations, and the probability of selecting a possible action in a given situation [Sutton and Barto, 2020]. For a policy π, the probability that $A_t = a$ and $S_t = s$ is $\pi(a|s)$ [Sutton and Barto, 2020]. Or, $\pi(a|s)$ is the probability that action a is taken when state s is given. Since there can be a very large number of possible actions per given state, we need to find a method for selecting effective policies. Theoretically, policies can be a guide used to navigate through states, but practically, how can we select more efficient policies so that we maximize the expected cumulative reward E(X) over time?

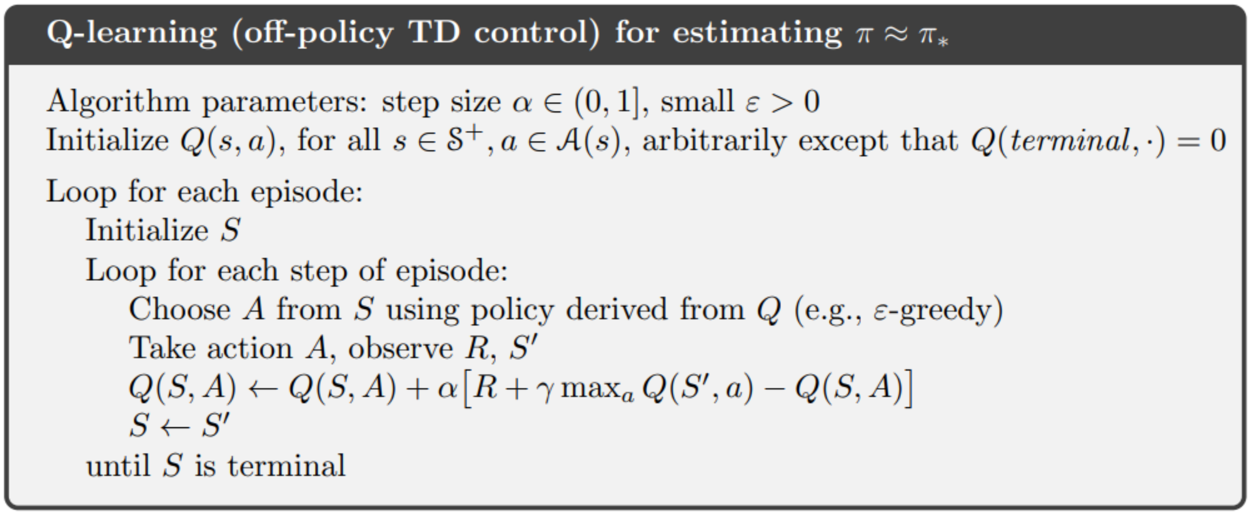
3 Q-Learning

3.1 Introduction

The issue of trivial or weak policies with low E(X) could be potentially rectified with Q-Learning, a style of reinforcement learning designed to work with the MDP to work towards converging to a policy with a maximized E(X) [Sutton and Barto, 2020]. Q-Learning is in the family of temporal difference (TD) methods where learning is achieved by bootstrapping from the approximation of the current function for values [Naeem et al., 2020]. Therefore, at every step, the utility function is updated based on the current function and theoretical reward values of potential actions from the current state [Sutton and Barto, 2020]. We will discuss how the use of this updated function at each step helps maximize the E(X) for policies.

3.2 Q-Learning Algorithm

Define $Q^{\pi}(s,a)$ as the action-value function in which E(X) is represented if an agent starts at the current state s, takes action a in state s, and follows policy π for subsequent actions [Naeem et al., 2020]. Our original problem was selecting effective policies hence we can expect the action-value function Q to change with a change in the policy in place [Naeem et al., 2020]. We also define ϵ as the probability of a, the action to take. α and γ are defined as the learning rate and discount factor respectively, where a larger learning rate allows the agent to adapt to more recent experiences but may lead to more oscillations in the process while a smaller learning rate allows the agent to be more conservative and less responsive to recent experiences, allowing for a steadier rate of learning with less fluctuations in the learning process [Kaelbling et al., 1996]. The discount factor allows the agent to weigh the importance of immediate rewards versus future rewards with a value closer to 0 prioritizing immediate rewards and a value closer to 1 prioritizing future rewards [Kaelbling et al., 1996]. Defining these variables, we can see the pseudo-code implementation of the Q-learning algorithm below.



Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

```
Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$
Loop for each episode:
    Initialize $S$
    Loop for each step of episode:
        Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
        Take action $A$, observe $R$, $S'$
        $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
        $S \leftarrow S'$
    until $S$ is terminal
```

[Sutton and Barto, 2020].

We can see in the Q-learning algorithm that in each repeat of the algorithm after initializing Q, we start off by

1. Initializing the current state.

   Then, we repeatedly

2. Choose the next action a from the state using a greedy strategy such as epsilon greedy (see sections 3.3 and 3.3.1)
3. Take action a and observe the received reward along with the next state
4. Update the Q function with state s and action a at time t to equal the temporal difference: representing the difference between the immediate reward and the expected cumulative future reward. This is known as the Q-Learning rule [Kaelbling et al., 1996].
5. Have the new state as the current state.
6. Keep repeating steps 2-6 until the current state is the final state in which we cannot extend to any other states.

7. Over multiple episodes (iterations) of this algorithm, the agent can select an effective policy by selecting actions in each state that will give us the highest E(X) from the knowledge that we have.

This process shows us how an effective policy can be formed without always using the current policy and exploring actions from different policies such as the epsilon-greedy strategy (see 3.3. and 3.3.1). We can conceptualize the MDP in this algorithm because we take an action while the environment gives us a reward and a new state based on our previous action, while also updating the Q-values based on the maximum estimated Q-value for the next state, regardless of the current action taken. This fulfills the Markovian state transition property discussed in section 2.3. Using greedy strategies, the Q-value function is updated in each step in hopes of potentially maximizing the E(X) for the updated policy π by adding the current greedy estimate and the temporal difference together.

3.3 Exploration vs. Exploitation

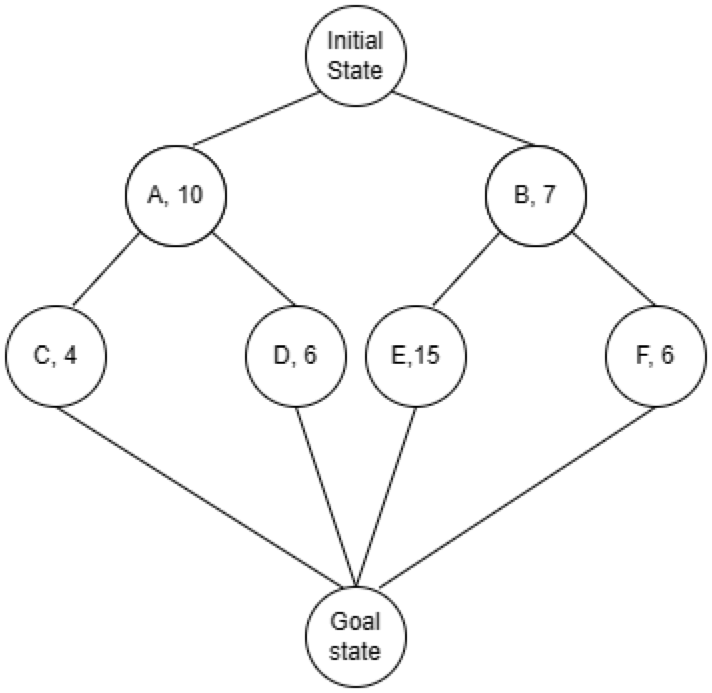
Although we use greedy strategies to search for the maximal value of a reward given a set of actions from a state, we must also consider the potential implications of not searching for an action at all. In practice, learning an optimal policy rarely happens, and when it does, there is extreme computational cost [Sutton and Barto, 2020]. Search is not cheap. Therefore, we must come up with mechanisms to balance the trade-off between exploration and exploitation. "If the agent explores too much, it cannot exploit its knowledge and without exploration, the agent will always have limited knowledge" [Asiain et al., 2019]. In the context of the Q-Learning algorithm, we can commonly see this problem when we are faced with a decision to select an action based on our current state. Do we select an action that we know has a high reward value from previous iterations of our algorithm or do we search for a new action, taking the liability of discovering an action with a reward value smaller than the values that were previously discovered? [Asiain et al., 2019].

3.3.1 Epsilon-Greedy

We propose a potential solution to this trade-off dilemma by using the epsilon-greedy algorithm. Epsilon-Greedy is a style of greedy algorithms that behave “greedily” most of the time [Asiain et al., 2019]. With a small probability ϵ, instead of exploiting our known Q-values for maximizing the immediate reward, we opt to arbitrarily select from all actions, independent of the Q-value estimates [Sutton and Barto, 2020]. With this algorithm, we can ensure that each iteration of the Q-Learning algorithm doesn’t explore for an action at every step, but also exploits pre-existing knowledge allowing for a greater balance of cost per iteration while also selecting effective and large Q-values. To clarify, at each step, the client has a small probability ϵ to randomly select an action based off the current state or use pre-existing knowledge from a previous action taken. Epsilon-greedy looks to balance between exploration and exploitation.

3.4 Search of Policies

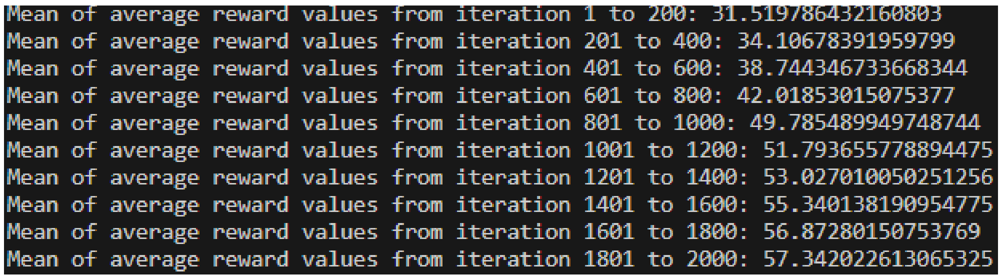
Exploration policies such as epsilon-greedy are not only used to balance search and exploits but also used to ensure that each action is tried at each state sufficiently enough [Asiain et al., 2019] for us to make a conclusion about which actions dictate the formulation of our “best” policy. The use of randomness in our exploration algorithms ensure that each action is sampled often enough in each state [Asiain et al., 2019]. Formation of an effective policy starts with attempting to maximize the average reward E(X) of all actions taken [Kaelbling et al., 1996], not necessarily by taking the highest reward of each step. The reasoning behind this can be seen in the diagram below.



We can see that greedily taking the highest value at each step may not result in the highest E(X) possible. Starting at the initial state, if we take A, then D, and finish at the goal state, we are left with 3 actions taken with an average reward of 10+6+0/3 = 5.333…, while taking B, E, then the goal state results in an average reward of 7+15+0/3 = 7.333…. Therefore all paths must be attempted to formulate the most effective policy possible. Eventually, through the Q-Learning algorithm, we will converge to an effective policy because of the implementation of strategies like epsilon-greedy and the Q-learning rule [Naeem et al., 2020].

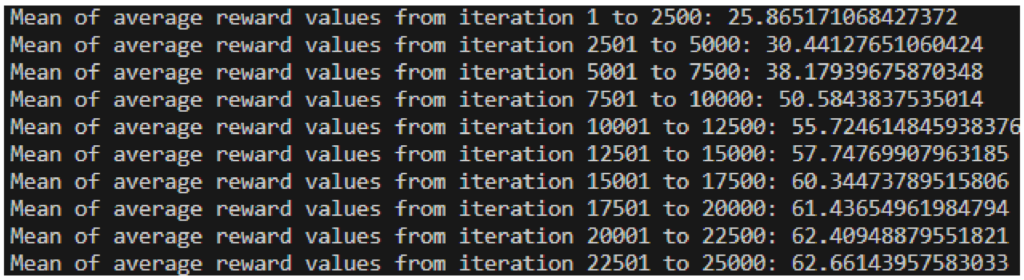
3.5 Implementation

Included with this paper is an implementation of the Q-Learning algorithm with Python, in a static environment. The algorithm has to step through 16 linear states, all with 4 actions. Regardless of which action was taken, we are automatically moved to the next state for simplicity. This was implemented using the Q-Learning pseudo algorithm provided by Sutton and Barto in 2020 (shown in 3.2). After running the program, we can see that E(X) increases as the number of iterations increase and therefore our policy improves after multiple runs.

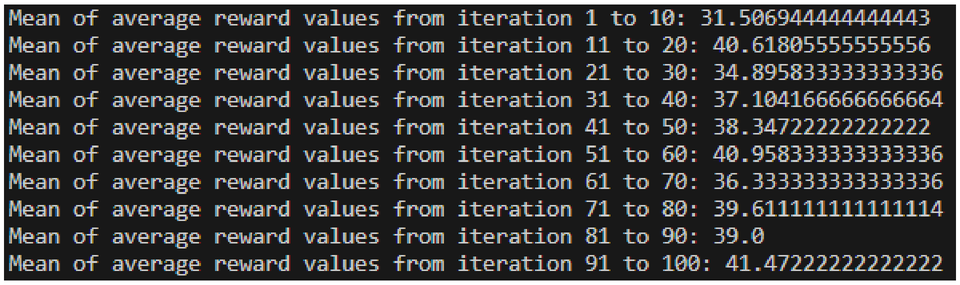


```
Mean of average reward values from iteration 1 to 200: 31.519786432160803
Mean of average reward values from iteration 201 to 400: 34.10678391959799
Mean of average reward values from iteration 401 to 600: 38.744346733668344
Mean of average reward values from iteration 601 to 800: 42.01853015075377
Mean of average reward values from iteration 801 to 1000: 49.785489949748744
Mean of average reward values from iteration 1001 to 1200: 51.793655778894475
Mean of average reward values from iteration 1201 to 1400: 53.027010050251256
Mean of average reward values from iteration 1401 to 1600: 55.340138190954775
Mean of average reward values from iteration 1601 to 1800: 56.87280150753769
Mean of average reward values from iteration 1801 to 2000: 57.342022613065325
```

This is a snippet of the output from my implementation of the Q-Learning algorithm, and we can see that our policy improves over time in terms of getting a larger E(X). Some considerations with this algorithm are changing the value of ϵ. Although the optimal policy would result in an E(X) of around ~63, we seem to converge at a number smaller that that. This is due to the fact that we are searching too much, (our ϵ value is too high) instead of using what we know. It seems that there is a trade-off between number of runs and speed of policy formation with altering ϵ in our implementation. ϵ is normally set to 0.1, but when changed to 0.01, the output is as follows.


```
Mean of average reward values from iteration 1 to 2500: 25.865171068427372
Mean of average reward values from iteration 2501 to 5000: 30.44127651060424
Mean of average reward values from iteration 5001 to 7500: 38.17939675870348
Mean of average reward values from iteration 7501 to 10000: 50.5843837535014
Mean of average reward values from iteration 10001 to 12500: 55.724614845938376
Mean of average reward values from iteration 12501 to 15000: 57.74769907963185
Mean of average reward values from iteration 15001 to 17500: 60.34473789515806
Mean of average reward values from iteration 17501 to 20000: 61.43654961984794
Mean of average reward values from iteration 20001 to 22500: 62.40948879551821
Mean of average reward values from iteration 22501 to 25000: 62.66143957583033
```

We can see that to receive a similar policy to when ϵ = 0.1, we need 12501 to 15000 iterations of the algorithm, when previously it was 1801 to 2000 iterations of the algorithm. However, we do seem to get closer to the optimal policy around ~63. Conversely, setting ϵ = 0.5 results in an output shown below.


```
Mean of average reward values from iteration 1 to 10: 31.506944444444443
Mean of average reward values from iteration 11 to 20: 40.61805555555556
Mean of average reward values from iteration 21 to 30: 34.895833333333336
Mean of average reward values from iteration 31 to 40: 37.104166666666664
Mean of average reward values from iteration 41 to 50: 38.34722222222222
Mean of average reward values from iteration 51 to 60: 40.958333333333336
Mean of average reward values from iteration 61 to 70: 36.333333333333336
Mean of average reward values from iteration 71 to 80: 39.611111111111114
Mean of average reward values from iteration 81 to 90: 39.0
Mean of average reward values from iteration 91 to 100: 41.47222222222222
```

It is clear that we converge to a policy relatively lower than the other two ϵ values. However, we need a substantially lower number of iterations to reach convergence to a policy. These examples show an apparent trade-off between exploration and exploitation. My implementation also shows that by using the Q-Learning rule and epsilon-greedy, we can form an effective policy through reinforcement learning.

3.6 Merits and Impediments

Although Q-Learning is particularly effective in environments with a clear exploration-exploitation trade-off through its simplicity in balancing exploration through strategies such as epsilon-greedy

[Kaelbling et al., 1996], Q-learning does not fare well in continuous environments due to it being designed for discrete spaces [Naeem et al., 2020]. Also, there are implications on domains that are too large, because Q-Learning requires a lot of memory to hold state-action pairs [Sutton and Barto, 2020]. Storing lots of information also means that there are lots of calculations to be done in terms of finding an effective policy, so it may take longer than other reinforcement learning methods to converge to an effective policy [Sutton and Barto, 2020].

4 Dynamic Environments

The challenges associated with learning are elevated when dynamic environments are prevalent. In dynamic environments, the goal state could potentially change at each time step [Bing et al., 2023]. So, creating a policy in such a domain requires an algorithm to adapt continuously [Bing et al., 2023]. We still need the domain to exhibit local consistency for at least some time steps because we need to be able to utilize data from prior timesteps [Bing et al., 2023].

4.1 Concept Drift

A non-static environment also comes with non-static reward values and goal states. If we decide to take an action as part of our most effective policy, but the environment changes, the reward value for the selected action may be influenced by external factors [Gupta et al., 2021]. Factors such as shifting conditions, evolving goals, or changes in the environment can potentially impact the effectiveness of not only the current action, but also every sub and super sequent action [Gupta et al., 2021].

4.2 Redefining “Effective” Policies

To deal with Q-Values that are not static, we must be also have a system to update our effective policy based off of the changing reward values from the dynamic environment. Therefore, if selecting an effective policy is based off the maximization of its E(X), an adjustment to policy as the current environment changes is necessary if we want to maintain the integrity of the most effective policy [Bing et al., 2023].

4.3 Q-Learning Faults when Applied to Dynamic Environments for Policy Formation

Q-Learning is naturally strong in dynamic environments due to its ability to update knowledge in each timestep. So, if we re-explore an action/state pair and receive a different Q-Value than what was previously encountered, a Q-Learning agent is able to adjust as necessary due to the ability to update the current policy in each step. However, when effective actions change, Q-Learning will need to explore new possibilities again [Gupta et al., 2021]. This will increase the need for memory since dynamic environments mean more actions since variables are changing [Gupta et al., 2021]. This also means that we will converge to an effective policy in a much slower manner due to the need of exploring the possible actions [Sutton and Barto, 2020].

4.4 Proposed Improvements

There are variants of Q-Learning to help with the large storage requirements of its base form [Gupta et al., 2021]. Deep-Q networks have the ability to store {s, a, r, s'} in a replay memory [Tan et al., 2017]. These stored experiences can be sampled from periodically instead of sampling from the consecutive experiences of normal Q-Learning [Tan et al., 2017]. This is more so compared to how a human thinks, where small bursts of memory give us information to take another action compared to only letting the current state and E(X) dictate our next action [Tan et al., 2017]. Deep-Q networks have a lower need for memory, because the necessity of storing and updating the entire set of Q-values is not required like it was previously in the base form of the Q-Learning algorithm [Tan et al., 2017]. This variant of the Q-Learning algorithm allows for more stable and efficient learning as we now have a variety of past experiences alongside an estimated E(X) to guide our search instead of a single experience [Tan et al., 2017]. Using Deep-Q networks, we can now handle the large number of states and actions that arise from selecting effective polices through a dynamic environment.

4.5 Adapting to the Domain

There are many possible variants of Q-Learning which all have their own benefits and faults. One variant of Q-Learning that we have discussed are Deep-Q networks, which attempts to model decisions off a structure based on replay memory, similar to how a human would select decisions. Modifying the algorithm shown in 3.2, during training we would in addition, initialize a memory replay at the start, update the memory replay after state transitions, and sample a random minibatch of transitions from the memory replay while also considering the Q-Learning rule [Tan et al., 2017]. This implicitly updates the current effective policy by way of updating the Q-Network (memory replay). This ensures that we are not just limited to the current off-policies' exploration of knowledge but also allows us to explore states while not being constraint by insufficient storage issues [Tan et al., 2017]. Using this approach, we are able to lessen the load on storage and rectify our issue with the normal Q-Learning algorithm where we will run into issues trying to store all the Q-values in dynamic environments.

5 Conclusion

In conclusion, our exploration of reinforcement learning has traversed fundamental concepts encompassing policies, Markov Decision Processes (MDP), and the nuances of Q-learning. We have touched over the intricate task of formulating effective policies within dynamic environments. Q-Learning, based on reward maximization, leverages the MDP framework to model decision-making and formulate effective policies. While Q-learning stands as a robust approach, its limitations become apparent in continuous and dynamic environments. The search for a new approach led us into discussing Deep-Q networks. Deep-Q networks allow us to model effective policies based on an approach similar to humans, by utilizing a memory replay that uses past experiences to help dictate our effective policy. Like humans, Deep-Q networks don't store every single Q-Value in their memory, allowing us to bypass the storage constraint imposed by finding effective policies in huge action/state spaces in dynamic environments.

6 References

[Asiain et al., 2019] Asiain, E., Clempner, J.B. and Poznyak, A.S., “Controller exploitation-exploration reinforcement learning architecture for computing near-optimal policies”. *Soft Computing*, 23(11):3591–3604, 2018.

[Bing et al., 2023] Bing, Z., Lerch, D., Knoll, A.*,* “Meta-reinforcement learning in non-stationary and Dynamic Environments”. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(3):1–17, 2022.

[Gupta et al., 2021] Gupta, A., Roy, P.P. and Dutt, V., “‘Evaluation of instance-based learning and Q-learning algorithms in Dynamic Environments”. *IEEE Access*, 9:138775–138790, 2021.

[Kaelbling et al., 1996] Kaelbling, L.P., Littman, M.L. and Moore, A.W., “Reinforcement learning: A survey”. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[Naeem et al., 2020] Naeem, M., Rizvi, S.T. and Coronato, A., “A gentle introduction to reinforcement learning and its application in different fields”. *IEEE Access*, 8:209320–209344, 2020.

[Sutton and Barto, 2020] Sutton, R.S. and Barto, A.G., *Reinforcement learning: An introduction*, The MIT Press, Cambridge, MA., 2020.

[Tan et al., 2017] Tan, F., Yan, P. and Guan, X., “Deep reinforcement learning: From Q-learning to Deep Q-Learning”. In Liu, D., Xie, S., Li, Y., Zhao, D., and El-Alfy, E.M. (Eds.), *Neural Information Processing*, pp. 475–483. Guangzhou, Guangdong, 2017.