A Comparative Study of Deterministic and Stochastic Policies for Q-learning

A Comparative Study of Deterministic and Stochastic Policies for Q-learning

Yaxin Bi
School of Computing
Ulster University
Belfast, United Kingdom
y.bi@ulster.ac.uk

Adam Thomas-Mitchell
School of Computing
Ulster University
United Kingdom
Thomas_Mitchell-A@ulster.ac.uk

Wei Zhai
Gansu Earthquake Agency
China Earthquake Administration
P. R. China
zhaiw@gsdzj.gov.cn

Naveed Khan
School of Computing
Ulster University
United Kingdom
n.khan@ulster.ac.uk

***Abstract*—Q-learning is a form of reinforcement learning that employs agents to perform actions in an environment under a policy to reach ultimate goals. Q-learning is also thought as a goal-directed learning to maximize the expected value of the cumulative rewards via optimizing policies. Deterministic and scholastic policies are commonly used in reinforcement learning. However, they perform quite different in Markov decision processes. In this study, we conduct a comparative study on these two policies in the context of a grid world problem with Q-learning and provide an insight into the superiority of the deterministic policy over the scholastic one.**

***Index Terms*—Reinforcement Learning, Q-Learning, Markov Decision Process, Deterministic and stochastic policies, GridWorld**

I. Introduction

Reinforcement learning (RL) is a branch of machine learning. This paradigm of learning is to employ an intelligent agent to perform actions in some environment and adapt its behaviour through trial-and-error to achieve the greatest cumulative reward from the environment, thereby controlling the agent to move toward desired goals and away from undesired goals [1]. Q-learning is a form of reinforcement learning, which is often referred to as a model-free reinforcement learning [2].

An agent has two essential components: a policy and a learning algorithm [3].

- The policy is to map from states to actions based on the observations from the environment.
- The learning algorithm updates the policy parameters based on the actions, observations, and reward. It aims to find an optimal policy that maximizes the cumulative reward received through interacting with the environment.

In a general sense, the interaction with an environment by an agent is formulated as a Markov Decision Process (MDP) that can be broken into sequences, called episodes [4]. The execution of the MDP can be described as a trajectory of occurrences (in terms of state, action, reward) over time-steps within episodes. Each episode ends in a terminal state, the agent could take varied time-steps to complete an episode as shown in Fig.1.

In fact, training an agent with reinforcement learning is an iterative process, each of which corresponds to an episode. The next episode begins independently of how the previous one ended, but all episodes can all considered to end in the same terminate state, i.e. goal, with different rewards for the different outcomes. Selections of actions and their corresponding reward in the later episode require to reflect on the convergence of a policy in the preceding episodes. If the training process does not converge to an optimal policy within a reasonable amount of time-steps, it is necessary to update relevant parameters of the policy and balance between exploration and exploitation when choosing actions.

Fig. 1. Time-steps and episodes over discrete states [5]

In past decades, reinforcement learning has achieved impressive advances in a variety of domains [8] and [9], but their applicability has been limited to domains in which useful features can be handcrafted, or to domains with fully observed, low-dimensional state spaces. A wide range of applications of RL had previously intractable issues, i.e. scaling RL to decision-making problems with high-dimensional state and action spaces. A recent development proposed a seminal method that has the ability to learn and play Atari 2600 video games at a superhuman level, directly from image pixels [6]. This work provided solutions for the instability of function approximation techniques for RL and demonstrates that an agent could be trained on raw, high-dimensional observations, solely based on a reward signal.

In practice the deployment of the RL solutions to real-world physical systems is often challenged by limited training data, formulation of complicated systems and costs of potential failures. A recent application of RL for regulating temperatures and airflow inside a large-scale data center (DC) was reported in [7], in which the authors developed a linear model of the DC dynamics based on model-predictive control, their approach uses safe and random exploration to start with little or no prior knowledge, and subsequently recommends control actions at

each time-step by optimizing the cost over model-predicted sequences.

In this study, we develop a Q-learning algorithm to study the competencies of the algorithm on the tasks of obtaining optimal policies. We developed a flexible agent, which is able to operate under deterministic and stochastic policies in the same environment, to investigate the effect of a varied range of policy parameters in maximising the cumulative reward. These results provide significant insights into real-world applications of reinforcement learning.

II. Reinforcement Learning

RL problems involve learning from interaction with the environment to achieve a goal. A model of this nature consists of two main components: the agent and the environment. The agent is the computational component that learns, makes decisions, and receives rewards. The environment is what the agent interacts with, and it is comprised of everything outside of the agent. Anything the agent cannot control is considered to be part of the environment [4].

The reward is a numerical value that may be positive or negative values, it gauges the effect of the actions taken by the agent. Each choice of action that the agent makes occurs on one time-step. The chosen action may or may not result in a change in state for the agent. This is dependent on the legality of the action from the current state within the environment. If the agent makes the choice of movement outside the environment, it will remain in its current state, but a time-step is counted. Thus when an agent in a state choses an action within the environment, the environment then sends an update to the agent of the new state and the reward gained from the chosen action. This process is formulated as a Markov Decision Process (MDP), consisting of four major concepts [4] as follows:

- **State** - A variable that communicates characteristic information about the environment to the agent, denoted by $S = \{s_1, s_2, ..., s_N\}$.
- **Action** - An interaction with or movement within the environment, denoted by $A = \{a_1, a_2, ..., a_K\}$.
- **Reward** - A numerical value received by the agent, denoted by R, where for any reward r, $r \in R$
- **Policy** - A policy π is a map from a state $s \in S$ to an action $a \in A$, where $\pi(s)$ represents an action taken in state s under *deterministic* policy, and $\pi(a|s)$ represents probability of taking action a in state s under *stochastic* policy π.

The dynamic trajectories of an MDP are described by the probability of transitioning between certain states. The probability distribution,

$$p(s', r|s, a) = Pr\{S_{t+1} = s', R_t = r|S_t = s, A_t = a\}, \quad (1)$$

gives the probability of transitioning to state s' and receiving reward r at time $t+1$ if at time t the environment was in state s and action a was taken. MDPs are defined by the *Markov Property*;

$$Pr\{S_{t+1}|S_t\} = Pr\{S_{t+1}|S_0, S_1, ..., S_t\}, \quad (2)$$

which asserts that the transition to the next state is dependent only on the current state; the history of the system is not relevant.

The agent aims to maximise the reward, which is the expected return in the long run, rather than the immediate reward. This can be achieved through either the *state-value function* or the *action-value function* [4].

The long term value of a given state under policy π is described by the *state-value function*;

$$v_\pi(s) = \mathbf{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s \right]. \quad (3)$$

The value of a state is the expected future reward with a discount rate, γ.

The long term value of an action given the state, under policy π, is described by the *action-value function*;

$$q_\pi(s, a) = \mathbf{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a \right]. \quad (4)$$

This is similar to Equation 3, however, this formulation considers state-action pairs rather than just the state.

The discount rate is a hyperparamter that determines the degree to which future rewards are considered at the current time step. As $\gamma \to 0$, the agent becomes more "myopic" and only considers immediate rewards. As $\gamma \to 1$, the agent becomes more "far-sighted" and considers future rewards to be as important as current reward.

For the model to learn to complete a task effectively, the agent must aim to find an optimal policy to maximise $v(s)$ or $q(s, a)$. To do this, one must consider the *Bellman Optimality Equations*. These equations highlight that if an agent uses an optimal policy then the value of a state is equal to the expected return of the action from that state that has the highest value. The *Bellman Optimality Equations* for the state-value function and action-value function, respectively, are as follows:

$$v_*(s) = \max_a \sum_{s',r} p(s', r|s, a)(r + \gamma v_*(s')), \quad (5)$$

$$q_*(s, a) = \max_{a'} \sum_{s',r} p(s', r|s, a)(r + \gamma \max_{a'} q_*(s', a')). \quad (6)$$

The main objective for an RL task is to solve one of these equations as that would represent an agent that has the optimal policy, such that it can perform the desired task to a high degree of competence.

III. The GridWorld Problem

A grid world is a two-dimensional cell-based environment, where the agent has five possible actions within the environment in terms of *up, down, left, right, jump*, and the state is defined as the agents position on the cells. The agent starts from one cell and moves toward the terminal cell while accumulating as much reward as possible. To train an agent, we can configure a grid world with different dimensions, starting

and ending cells, and obstacles over the cells to materialise an environment, Q-learning algorithms would discover optimal paths and policies on the grid to arrive at the terminal goal in the fewest moves with the maximal reward. Fig.2 illustrates a configuration of a grid world for this study.

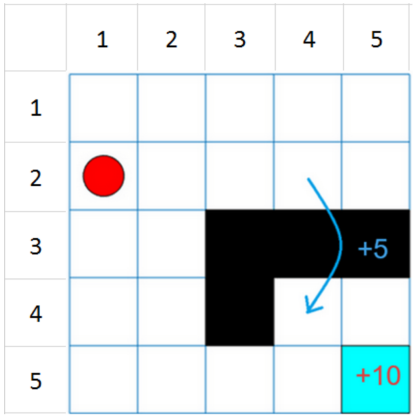

Fig. 2. Gridworld environment [3]

Specifically, the agent starts from cell [2,1] and receives a reward +10 if it reaches the goal of cell [5,5]. The possible actions are dependent on the position of the agent. There are two constraints for the movement of the agent. The first one is that the agent cannot move outside the grid world as the grid has borders in place. For instance, if the agent is in its initial state [2, 1], it only has three possible actions of emphup, down and emphright. The second is the black squares that denote obstacles, these black cells cannot be positioned by the agent, but the agent can take a *jump* action from cell [2,4] to [4,4] with a reward of +5, which would encourage the agent to take that shortcut. All other actions result in –1 reward.

IV. Methodology

A. Q-learning formulation

As previously stated, the Q-learning method is utilised to develop a solution to this grid world task. Q-learning is able to learn action-values over the trajectories of state-action pairs that the agent takes from the initial state to the terminal state. At each time-step t, the agent performs an action a_t to transition from one state s_t to another s_{t+1} and obtains a reward r_t as illustrated in Fig. 1. The agent aims to learn a policy that allows it to maximise the cumulative reward over the trajectories.

In practice Q-learning approximates the optimal action-value function, q_*, and accumulate r_t into Q-values for each of state-action pairs. The values are incrementally updated based on reward feedback r_t and the current Q-values. The Q-value will be calculated by the following equation,

$$Q(S_t, A_t) \longleftarrow Q(S_t, A_t) + \\ \alpha \left[R_{t+1} + \gamma max_a Q(S_{t+1}, A_t) - Q(S_t, A_t) \right], \quad (7)$$

where α and γ are the hyperparamters known as the learning rate and discount rate, respectively.

The learning rate, $\alpha \in [0, 1]$, represents how much importance is placed on new information. If $\alpha = 0$ the agent will learn nothing from new actions, whereas if $\alpha = 1$, the agent will entirely ignore previous knowledge and only value the most recent information. The discount rate is the same as was described earlier.

In addition, function $max()$ takes the maximum of the future Q-values, which is combined with the current state for next iteration. The function uses a possible future reward to impact the current action, as such it helps the agent select an action with the highest return at any given state.

As Q-learning is a model-free algorithm, the agent does not construct an internal model or MDP, instead learns from trial and error. The algorithm uses a ϵ greedy strategy to determine an action between exploration and exploitation within the environment. At the beginning the agent explores the environment and randomly chooses actions as the agent does not know anything about the environment. With more explorations the agent progressively obtains the knowledge about the environment and become more confident in estimating Q-values.

With decreasing of the epsilon rate ϵ, the agent starts to exploit the environment. However if purely exploration, the agent will develop a good knowledge of the environment but it will never exploit this knowledge to solve the task. If purely exploitation, the agent may only ever use the first policy it tries and it will not discover other methods that could provide a higher return. The means if setting $\epsilon = 0$, the agent never explores but always exploit the knowledge it already has. On the contrary, if setting $\epsilon = 1$, the greedy strategy forces the algorithm to always take random actions and never use past knowledge. Usually, ϵ is selected as a small number close to 0. The epsilon parameter introduces randomness into the algorithm, forcing the agent to try different actions. This helps not getting stuck in a local optimum (maxima).

As a result there is a need to make a good balance between exploration and exploitation. The algorithm randomly generate a value $z \in [0, 1]$ when selecting an action. If $z < \epsilon$ then the agent explores the environment, otherwise, the agent exploits the environment and selects the action based on the largest Q-value for the current state on the policy.

B. Deterministic and Stochastic Policies

The convergence of state-action values always depends on a policy. As stated previously the Q-learning algorithm implicitly uses the ϵ-greedy strategy to balance between exploration and exploitation when selecting actions. This strategy encourages the agent to explore as many states and actions as possible. The more iterations it performs and the more paths it explores, the more confident the agent becomes that it has tried all the options available to find better Q-values.

Apart from trading-off between exploration and exploitation, the value function given in Equations 3 and 5 determines how good it is for the agent to be in a particular state. To determine how good it will be in a particular state, that must depend on some actions that the agent will take, which is

determined by a policy π, i.e. *deterministic* and *stochastic*, which can be calculated by Equations 4 and 6.

Normally the deterministic policy is used in an deterministic environment. In such an environment, the actions taken determine the outcome. By contrast the stochastic policy is used when the environment is uncertain. This process is often called a Partially Observable Markov Decision Process (POMDP). But a deterministic policy can be interpreted as a stochastic policy that gives the probability distribution $\pi(a|s) = 1$.

Given two policies, denoted by π and π', there is always one policy that is better than or equal to another, that means $\pi \geq \pi'$ if and only $v_\pi(s) \geq v_{\pi'}(s)$ for all $s \in S$ or if and only $q_\pi(s, a) \geq q_{\pi'}(s, a)$ for all $s \in S, a \in A$. Although there may be more than one, we denote all the optimal policies by π_*. To compare the performance of two policies over the sequences of state-action pairs, we need to obtain π_* through comparing results produced by either Equation 5 or 6.

V. Evaluation and Results

To compare the effectiveness of the deterministic and stochastic policies, we have implemented a Q-learning algorithm for solving the grid-world problem and manually allocated probabilities over the five actions for studying the stochastic policy. The evaluation mainly focuses on assessing the effect of three hyperparameters ϵ, α and γ over a set of values $\{0.2, 0.5, 0.7\}$, and runs the algorithm 1000 episodes on the set of values for the three hyperparameters.

Fig. 3 shows the cumulative reward under the deterministic policy. From the figure it can be seen that around 200 episodes, the fluctuation of the cumulative reward appears to be stable. With the increase of ϵ values the cumulative reward increases, but that also brings more fluctuations. The final cumulative reward is 66.97 on average. This result reveals that the agent takes actions at a more random way and use less previous knowledge learnt from the environment. Fig. 4 presents the cumulative reward under the stochastic policy, which follows some similar patterns with the exception of the smaller final cumulative reward, i.e. 32.13 on average.

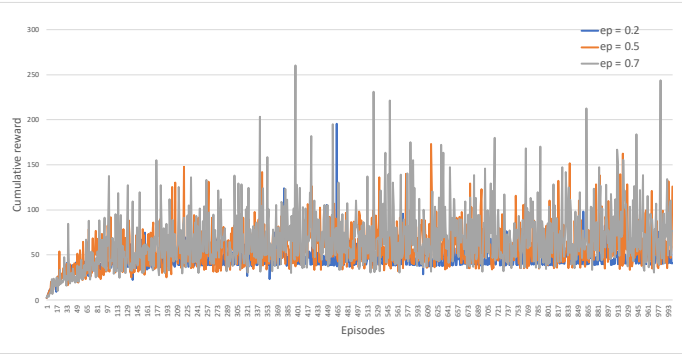

Fig. 3. Cumulative reward with varied ϵ values in Deterministic policy

Fig. 5 shows the cumulative reward under the deterministic policy, it can be observed that there are larger variations of the reward on $\alpha = 0.5, 0.7$, where for $\alpha = 0.5$, the variations

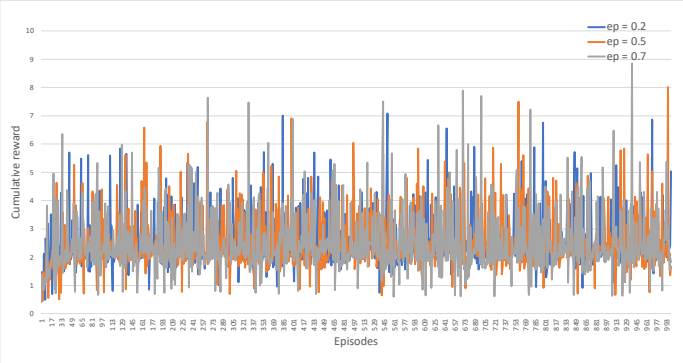

Fig. 4. Cumulative reward with varied ϵ values in Stochastic policy

occur before 500 episodes, whereas there are more variations after 500 episodes when $\alpha = 0.7$. This pattern indicates that the agent prefers to learn from most recent information in maximising the reward. While Fig. 6 presents the cumulative reward under the stochastic policy in the same settings, three different values of $\alpha = \{0.2, 0.5, 0.7\}$ have a similar effect with more fluctuations over the training process, and the cumulative reward is approximately 2.47 on average.

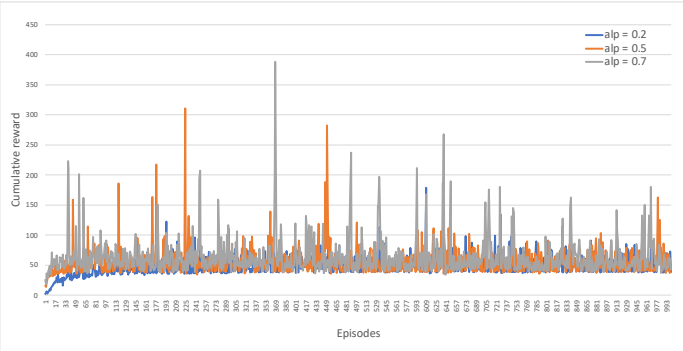

Fig. 5. Cumulative reward with varied α values in Deterministic policy

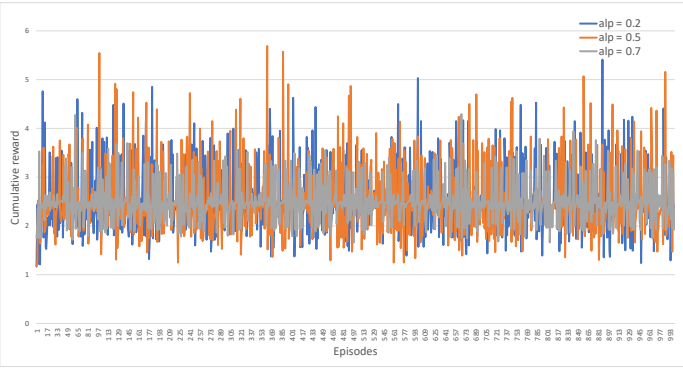

Fig. 6. Cumulative reward with varied α values in Stochastic policy

Fig. 7 shows the cumulative reward under the deterministic policy. The figure clearly demonstrates the different roles the discount rate plays, that is the larger the discount rate,

the agent takes more previous knowledge into consideration in maximising the reward. This result reflects what effect described in Equation 4. Fig. 8 shows the cumulative reward under the stochastic policy in the same setting, it can be seen that three different discount values have a similar effect, but not positive in maximising the reward.

The final training results over 1000 episodes under the deterministic policy is given in Fig. 9, as the calculation of reward is in a backward way with the setting with $\epsilon = 0.2$ $\alpha = 0.2$ and $\gamma = 0.7$, the reward in the terminal cell alway remains the same as the initial value 10. The agent has no prior knowledge of the grid-world environment, the Q-value for each state-action pair is zero initially. During the training, the agent takes a random route through the environment and obtains a relatively low reward. With more episodes, the agent travels through the cells, it retains the knowledge that it gains by updating the Q-value of that state-action pair, eventually exploits what learnt to find optimal paths with a larger reward.

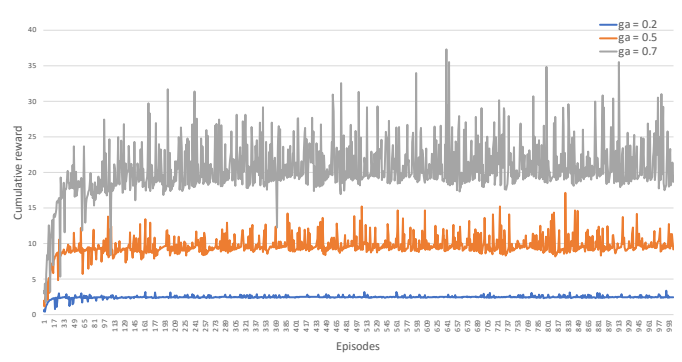

Fig. 7. Cumulative reward with varied γ values in Deterministic policy

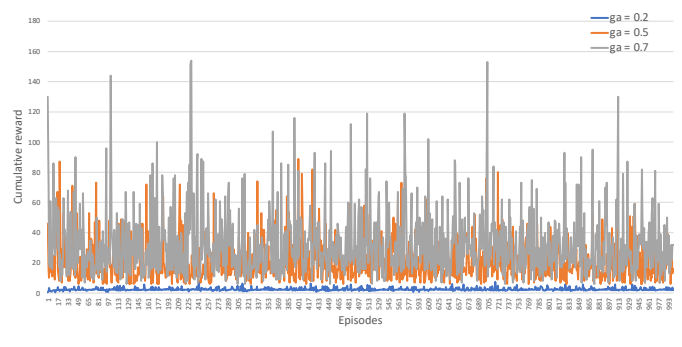

Fig. 8. Cumulative reward with varied γ values in Stochastic policy

VI. Conclusion

The study has implemented a Q-learning method for solving the grid world problem, where the agent could learn the optimal policy by interacting with the grid-world environment. The optimal policy involved the agent navigating from the initial state to the terminal state. The evaluation results reveal that the agent performs better under the deterministic policy in comparison with the stochastic policy. The evaluation results demonstrate that the effect of the learning rate and discount rate play smaller roles in maximising the cumulative reward, this could be because of a possible limitation of the grid-world environment.

	1	2	3	4	5
1	0.59	0.89	1.32	1.97	1.14
2	0.72	1.34	1.95	2.93	1.77
3	0.45	0.75	0.0	0.0	0.0
4	0.0	0.33	0.0	4.18	7.0
5	0.0	0.08	3.98	7.0	10.0

Fig. 9. Distribution of expected return on the grid cells in the deterministic policy

To ensure the basic Q-learning algorithm did not focus too much on exploring the environment without exploiting the learned knowledge, it was particularly investigated on the ϵ-greedy algorithm for action selection with the different values, it was found that a value of $\epsilon = 0.7$ provided a good balance between exploration and exploitation which allowed the agent's learned policy to converge to the optimal policy relatively quickly.

Additionally, the performance of the policies could be evaluated against different permutations of these hyperparameters to determine what values are useful for find an optimal policy with the least number of actions, which will remain in future.

References

[1] Christopher J. C. H. Watkins & Peter Dayan. Q-learning. Machine Learning volume 8, pages 279–292 (1992)

[2] Watldns, C.J.C.H. (1989). Learning from delayed rewards. PhD Thesis, University of Cambridge, England.

[3] Reinforcement Learning ToolboxTM, User's Guide, Matlab, R2021b.

[4] Sutton, R. S. & Barton, A. G. (2018), Reinforcement Learning: An Introduction , The MIT Press.

[5] Watt, J., Borhani, R., & Katsaggelos, A. (2020). Machine Learning Refined: Foundations, Algorithms, and Applications (2nd ed.). Cambridge: Cambridge University Press. doi:10.1017/9781108690935

[6] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, et al., "Human-level control through deep reinforcement learning," Nature, vol. 518, no. 7540, pp. 529–533, 2015.

[7] Nevena Lazic, Tyler Lu, Craig Boutilier, Moonkyung Ryu (2018). Data center cooling using model-predictive control. 32nd Conference on Neural Information Processing Systems (NeurIPS 2018), Montréal, Canada.

[8] Tesauro, G. Temporal difference learning and TD-Gammon. Commun. ACM 38, 58–68 (1995).

[9] Riedmiller, M., Gabel, T., Hafner, R. & Lange, S. Reinforcement learning for robot soccer. Auton. Robots 27, 55–73 (2009).