CSE 474: Project 4

Peter M. VanNostrand
Department of Computer Science
University at Buffalo
Buffalo, NY 14260
pmvannos@buffalo.edu

Abstract

In this project we implemented and trained a q-learning agent to solve a simple navigation task in basic grid-world environment. The agent begins by exploring the space by taking random actions, with each action the agent updates a Q-table which tracks the expected reward of moving in all directions. As training iterations advance the agent becomes more likely to exploit this learned information by taking actions that maximize the expected reward. This allows the agent to learn to navigate the environment and reach a navigational goal in an optimal number of steps.

1 Introduction

Reinforcement is a machine learning technique often used for performing real time actions. Unlike other forms of machine learning with are often concerned with data analysis and classification, reinforcement learning has a software agent which takes actions depending on its observations. Reinforcement learning is popular in the field of robotics where software agents can be trained by interacting with the real world through input sensors and output actuators. In reinforcement learning the agent is given a specific goal, such as navigating to a location or picking up an item, and is then given positive rewards for actions that bring it close to that goal and negatively rewarded for actions that take it farther from the goal. Q-learning is a type of reinforcement learning in which the agent attempts to learn the so-called Q function. This function $Q(s, a)$ represents the cumulative reward given to the agent for taking an action a while in the state s, the agent attempts to learn an approximation of Q such that it can find the action that maximizes the reward. Here we implement one possible algorithm to perform Q-learning, evaluate that algorithm in a simple environment, and present its results.

2 Environment

In this experiment we made use of simple grid-world environment provided by the course staff. This environment consists of a 5x5 grid consisting of a total of 25 spaces as shown in Figure 1

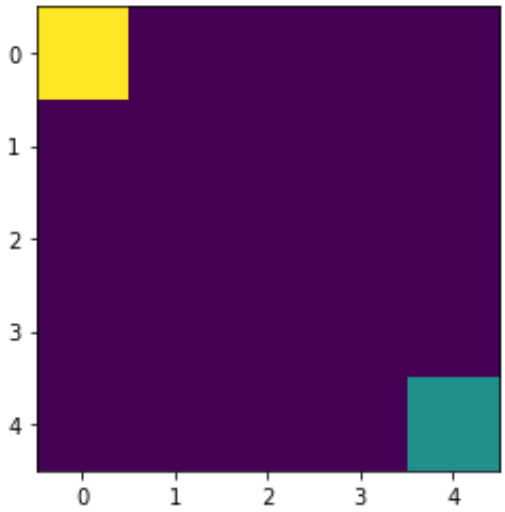

Figure 1: Environment

In the top left corner, we see the agent indicated by a yellow square. In the bottom right corner, we see the goal indicated by a teal square. Our goal is to create and train a q-learning agent that can navigate from its starting position to the goal. The agent can do this by moving in the four directions: up, down, left, and right. On each action the environment provides a reward of +1 for moving closer to the goal and -1 for moving away from the goal or staying the same distance from the goal. If the agent attempts to move off the edge of the world it will be prevented from doing so and will receive a reward of -1.

During execution the environment provides the agent with its current state. We represent these states as a pair of integers corresponding to the row and column location of the agent on the board, for example the agent starts in the state $(0, 0)$. Actions are represented as an integer in the range 0-3 with the following value action mapping

Table 1: Action Values

Value	Action
0	Down
1	Up
2	Right
3	Left

The environment then provides a step function which takes an action from the agent and returns a reward, the new state, and whether the agent has reached the goal or run out of time. The environment also provides a reset function, which moves the agent back to its starting location and resets the timer.

3 Q-Learning Algorithm

The goal of our agent is to learn an approximation of the Q function and then use that learned information to take actions that will maximize its cumulative reward. To do this our algorithm keeps a *Q-table*. This table maps all possible state action combinations to a Q value between zero and one, this value represents the expected cumulative reward resulting from taking the given action while in the given state. An example of such a Q-table is shown below

$Q(s,a)$ s γ = 0.95

a	000 100	000 010	000 001	100 000	010 000	001 000
↑	0.2	0.3	1.0	-0.22	-0.3	0.0
↓	-0.5	-0.4	-0.2	-0.04	-0.02	0.0
→	0.21	0.4	-0.3	0.5	1.0	0.0
←	-0.6	-0.1	-0.1	-0.31	-0.01	0.0

Figure 2: Sample Q-Table

When making selecting which action to perform the agent looks up the Q-values for its current state and performs the action that yields the highest Q-value. In our case we store the Q-table as a 5x5x4 matrix, with the first two dimensions representing the row and column of the agent in the world and then last dimension representing the Q-value for each of the four possible actions.

Unfortunately, the Q-table for a given world and goal is not known, therefore the agent must learn these values through its experience. To learn the Q-value for each state action pair the agent begins by exploring the world randomly. The Starting with a Q-table filled with zeros, every time the agent performs an action it updates the Q-value based upon the state and reward that results from taking that action. The new Q-value is determined based upon the following equation

$$Q^{new}(s_t, a_t) \leftarrow (1-\alpha) \cdot Q(s_t, a_t) + \alpha \cdot \left(r_t + \gamma \max_a Q(s_{t+1}, a)\right)$$

Where s_t is the current state, a_t is the action performed by the agent and r_t and s_{t+1} are the reward and new state resulting from performing that action. We also notice the two constants α and γ, these are hyperparameters which influence how the agent learns.

The constant α is a learning rate which scales the contribution of the new reward with respect to the existing Q-value, thereby controlling how quickly the agent learns. A value of $\alpha = 0$ would result in the new reward being entirely ignored, and a value of $\alpha = 1$ would result in the old Q-value being entirely replaced at every step. An appropriate value of the hyperparameter α should be found that allows the agent to retain information it has already learned while incorporating novel experiences.

The constant γ is known as the discount factor, this hyperparameter controls how much weight is given to the immediate reward of an action vs the long-term cumulative rewards possible after performing this action. This allows the agent to take actions which produce a small immediate reward, but move it in the direction of further rewards. A value of $\gamma = 0$ would create an instant gratification agent which learns only to get immediate rewards, so an appropriate value must also be found.

After many random actions the agent has probably learned a decent estimation of the Q function via its learned Q-table. We call this phase exploration, where the agent takes random actions to find all the possible state action Q-values. However, performing random actions indefinitely would not allow the agent to identify and perfect the optimal set actions. To combat this, we have the agent pick a random uniform number η, if that number is less than the value ϵ the agent performs a random action, otherwise it performs the optimal action determined by its learned Q-table. This makes agent randomly perform random actions. By decreasing the value of ϵ with each training iteration we decrease the likelihood of random actions over time. This causes the agent to pick the best possible action sequence and then perfect that sequence over time by learning all the sequences that differ by a few random decisions, we call this convergence to a specific solution exploitation. The staring value of ϵ and the rate at which epsilon decreases $\delta\epsilon$ are important hyperparameters for training a Q-learning agent. If the staring value of ϵ, denoted as ϵ_0, is too small or the value decreases too quickly the agent is likely to not be exposed to the globally optimal set of state action sequences, and these values should be chosen to balance exploitation and exploration. In our case ϵ decreases exponentially with each iteration as $\epsilon = \epsilon_0 e^{-\delta\epsilon \cdot t}$ where t is the iteration number.

4 Challenge

The goal of this project is to implement a software agent which using Q-learning to navigate to the goal location within the grid-world. Specifically, using the template provided by the instructors, we must implement the Q-Learning agent functions: policy and update.

The policy function takes the current state of the agent and determines which action to perform. First a random number is selected to determine if the agent should perform a random action, if that value is less than ϵ one of the four possible actions is randomly selected and returned. If the agent does not perform a random action, the function uses the learned Q-table to determine which action maximizes the expected cumulative reward. This decision is determined simply by looking up the Q-values for current state and then finding which action has the largest Q-value.

The update function takes the current state, the action performed, the next state, and the reward and updates the Q-value for its current state based upon the information it has just obtained. The function used to update the Q-value is described in Section 3.

Lastly, we implement an algorithm which connects the created agent to the environment and performs training. Using the step function of the agent we determine which action it would like to perform, then we use the environment's step function to obtain the resulting new state and reward. These values are passed to the agent's update function, this process is repeated for 1,000 iterations, with a decreasing value of ϵ at each iteration. We track the value of epsilon and the cumulative reward at each iteration, plots of these values are shown in Section 5.

5 Results

Using our completed code, we trained the agent for 1,000 iterations, below are the values of ϵ and the cumulative reward generated during the training of the agent.

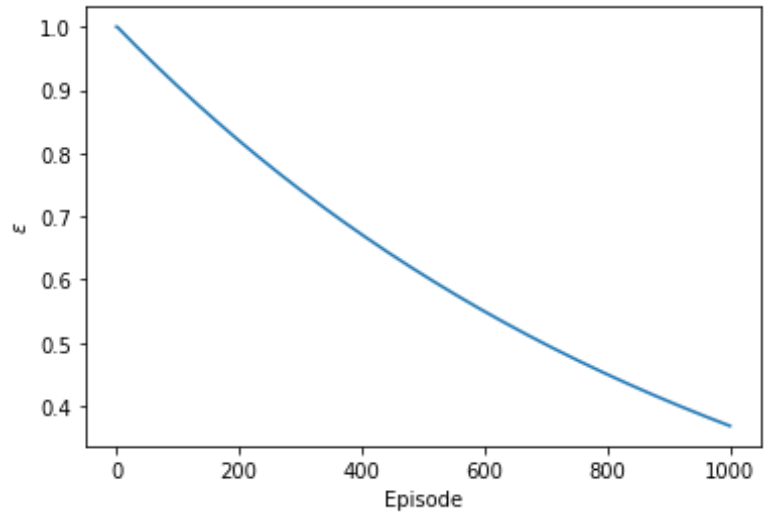

Figure 3: Value of Epsilon by Iteration

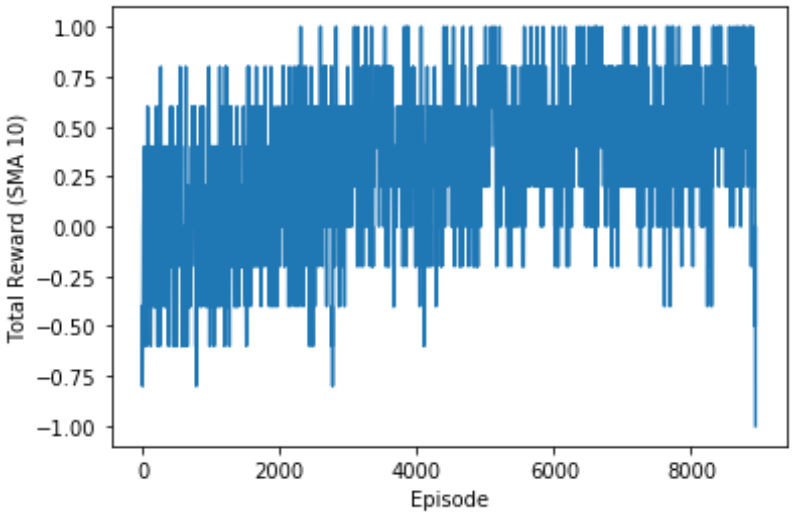

Figure 4: Cumulative Reward by Iteration

In Figure 3 we observe that the value of ϵ is decreasing exponentially as expected, the drives the agent to move from exploration at the start of training to exploitation towards the end of training. The initial value of $\epsilon_0 = 1$ with a decay rate of $\delta\epsilon = 0.001$. The learning rate used was $\alpha = 0.1$ with a discount factor of $\gamma = 0.9$

In Figure 4 we see the cumulative reward for each iteration. At the start of training we see that the rewards are randomly distributed around zero, with the agent gaining both large positive and negative rewards. Over time the average cumulative reward increases towards positive one, with the relative frequency of negative rewards decreasing. This is a result of the agent moving from exploration to exploitation. In the beginning the agent makes random decisions to explore the entire grid-world, then it picks the optimal path and begins to perfect that path over time resulting in the consistently increasing rewards.

Once training was complete the learned Q-table was used to have the agent navigate to the goal, the resulting set of actions is shown below in Figure 5

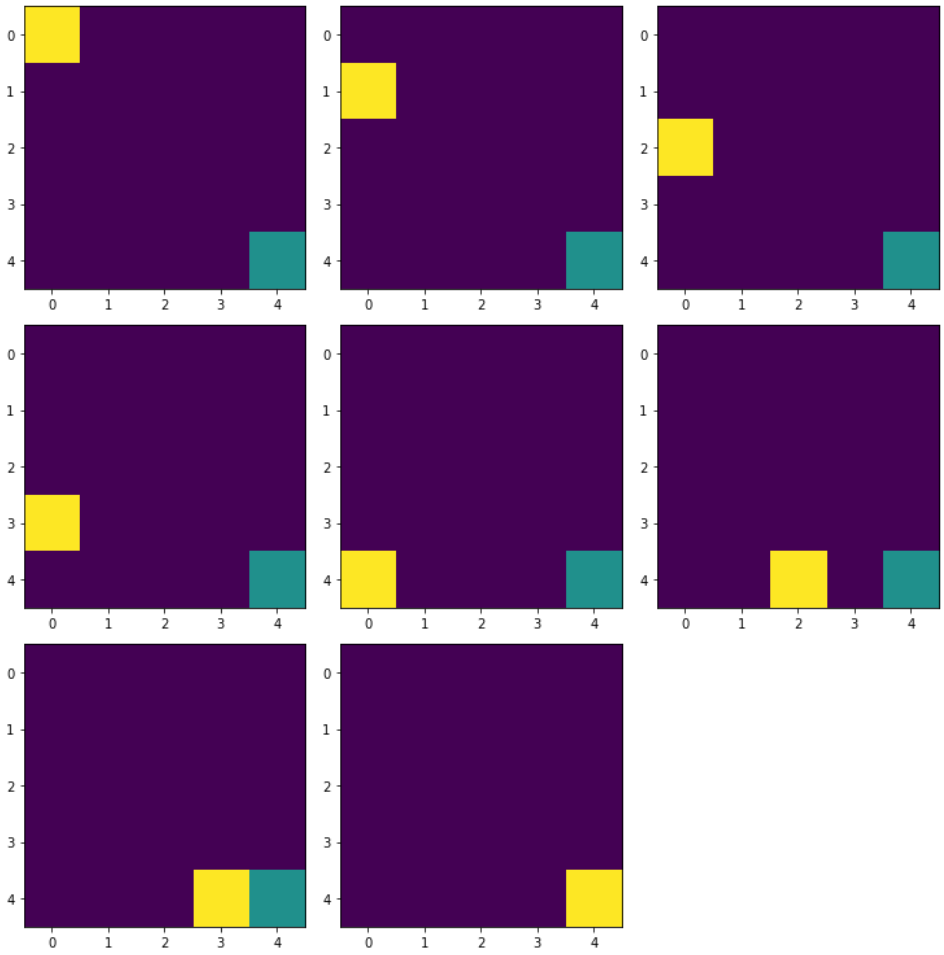

Figure 5: Trained Q-Learning Agent

We observe that the agent has learned to move down to the bottom of the world and then directly right towards the goal. This is one of many possible paths to the goal, but is completed in the optimal seven movements. The learned Q-table for performing these actions is shown below

Table 2: Q-Table

State		Action			
row	col	down	up	right	left
0	0	0.1235	-0.0988	0.1235	-0.0988
0	1	0.1235	-0.0988	0.1235	-0.0988
0	2	0.1235	-0.0988	0.1235	-0.0988
0	3	0.1235	-0.0988	0.1234	-0.0988
0	4	0.1233	-0.0988	-0.0988	-0.0987
1	0	0.1235	-0.0988	0.1235	-0.0988
1	1	0.1235	-0.0988	0.1235	-0.0988
1	2	0.1235	-0.0988	0.1235	-0.0988
1	3	0.1234	-0.0988	0.1233	-0.0988
1	4	0.1221	-0.0988	-0.0989	-0.0988
2	0	0.1235	-0.0988	0.1235	-0.0988
2	1	0.1235	-0.0988	0.1235	-0.0988
2	2	0.1234	-0.0988	0.1234	-0.0988
2	3	0.1233	-0.0988	0.1232	-0.0988
2	4	0.1220	-0.0989	0.0000	-0.0988
3	0	0.1235	-0.0988	0.1235	-0.0988
3	1	0.1234	-0.0988	0.1234	-0.0988
3	2	0.1233	-0.0988	0.1233	-0.0988
3	3	0.1222	-0.0988	0.1221	-0.0988
3	4	0.1100	-0.0982	-0.0910	-0.0890
4	0	-0.0988	-0.0988	0.1234	-0.0988
4	1	-0.0988	-0.0988	0.1233	-0.0988
4	2	-0.0989	-0.0988	0.1222	-0.0988
4	3	-0.1000	-0.0989	0.1111	-0.0989
4	4	0.0000	0.0000	0.0000	0.0000

Table 3: Best Decision by State

	Column					
Row		0	1	2	3	4
	0	D	D	D	D	D
	1	D	D	D	D	D
	2	D	D	D	D	D
	3	D	D	D	D	D
	4	R	R	R	R	D

6 Conclusion

In this project we explored the functioning of reinforcement learning by implementing a simple software agent. We saw how an agent can be trained using Q-Learning to determine an optimal series of actions, discussed how important hyperparameters such as ϵ, learning rate, and discount factor can affect the learning of a software agent, and then implemented a simple Q-learning algorithm. Next, we trained the agent we created in a simple grid-world with the goal of the agent navigating to a specific location by moving up, down, left, and right. During training we observed how the decreasing value of ϵ causes the agent to transition from exploration to exploitation, and how during this process it finds ones of many optimal action sequences resulting in an increasing cumulative reward. Lastly, we applied the trained agent to the navigational task and observed how it had learned to achieve its goal in an optimal amount of time. While this is a simple example of Q-Learning, similar algorithms are used in advanced robotics, automated decision making, and even to create video game playing artificial intelligence.