Reinforcement Learning

Soumita Das
University at Buffalo
UBIT Name: soumitad
UB Person Number: 50320170
soumitad@buffalo.edu

Abstract

Reinforcement Learning is a major paradigm in Machine Learning apart from Supervised and Unsupervised Learning. It is based on the notion of taking actions which maximizes the cumulative reward. This basically teaches automated agents to take actions based on its present state. In this project we are provided with an environment which consists of the agent and the goal. The three major tasks we are told to implement are the Q-Learning, policy determination and the training process. The upcoming sections contains the details of the algorithm and implementation.

1 Introduction

Machine learning has three main paradigms, *supervised*, *unsupervised* and *reinforcement learning*. Classification that depends on the attributes and target variables of the training set, and then go on to predict class labels for unknown test samples is a category of *supervised* learning. On the other hand clustering techniques which are *unsupervised* learning groups input samples based on the homogeneity among their features. Reinforcement Learning is based on teaching an automated agent to take actions that rewards it based on its current state.

While neural networks are responsible for recent breakthroughs in problems like computer vision, machine translation and time series prediction – they can also combine with reinforcement learning algorithms to create something astounding like AlphaGo.

Reinforcement learning refers to goal-oriented algorithms, which learn how to attain a complex objective (goal) or maximize along a particular dimension over many steps. They can start from a blank slate, and under the right conditions they achieve superhuman performance. This is done by penalizing the wrong decisions and rewarding the right ones.

Reinforcement algorithms that incorporate deep learning can beat world champions at the game of Go as well as human experts playing numerous Atari video games. While that may sound trivial, it's a vast improvement over their previous accomplishments, and the state of the art is progressing rapidly.

Reinforcement learning solves the difficult problem of correlating immediate actions with the delayed returns they produce with the use of hyper-parameters like epsilon discussed later. Like humans, reinforcement learning algorithms sometimes have to wait a while to see the fruit of their decisions. They operate in a delayed return environment, where it can be difficult to understand which action leads to which outcome over many time steps. With correct implementation we can expect RL agents to achieve success in real life applications.

2 Algorithm and Implementation

2.1 Markov Decision Process

Reinforcement Learning can be formally defined as a Markov Decision Process(MDP).
An MDP is a 4-tuple (S, A, P, R), where
• S is the set of all possible states for the environment. A state is a concrete and immediate situation in which the agent finds itself.
• A is the set of all possible actions the agent can take. An action is almost self-explanatory, but it should be noted that agents usually choose from a list of discrete, possible actions.
• $P = P(s_{t+1} = s' | s_t = s, a_t = a)$ is the state transition probability function
• R:S*A*S $->$ R is the reward function. From any given state, an agent sends output in the form of actions to the environment, and the environment returns the agent's new state (which resulted from acting on the previous state) as well as rewards, if there are any. Rewards can be immediate or delayed. They effectively evaluate the agent's action.

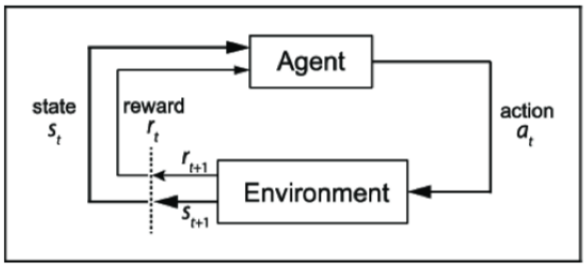

Figure 1: MDP

2.2 Q-Learning

Q-learning is a technique that evaluates which action to take based on an action-value function that determines the value of being in a certain state and taking a certain action at that state.

We have a function Q that takes as an input one state and one action and returns the expected reward of that action (and all subsequent actions) at that state. Before we explore the environment, Q gives the same (arbitrary) fixed value. But then, as we explore the environment more, Q gives us a better and better approximation of the value of an action a at a state s. We update our function Q as we go.

In our implementation we have formed a Q-Table. The dimension of the Q-Table is $|S| * |A|$ where $|S|$ is the number of states and $|A|$ is the number of actions. Each table entry $q_{i,j}$ is the Q value for the state i with the action j. Initially the table is initialized to zero values, as the agent learns, the values are updated.

The formula for Q-Learning update step is shown in the figure below:

$$Q^{new}(s_t, a_t) \leftarrow (1-\alpha) \cdot \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\left(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} \right)}^{\text{learned value}}$$

Figure 2:

The hyper-parameters like α and γ are discussed in detail in the next subsection. The estimate of optimal future value is the max of Q-values of all the states that can be reached from the state s_{t+1} after the present action a_t. The reward is the feedback by which we measure the success or failure of an agent's actions a_t in a given state.

2.3 Hyper-parameters

The main hyper-parameters used in the implementation are the following:

The Learning Rate α : this is how aggressive we want to be when updating our value. When alpha is close to 0, we're not updating very aggressively. When alpha is close to 1, we're simply replacing the old value with the updated value. In the implementation the learning rate was fixed to 0.1.

Discount Factor γ : The discount factor gives more weightage to immediate rewards than later rewards. In the implementation it is fixed to 0.9.

Epsilon in the ϵ-greedy method : This brings up the exploration/exploitation trade-off. One simple strategy for exploration would be for the agent to take the best known action most of the time, but occasionally explore a new, randomly selected direction even though it might be walking away from known reward.

This strategy is called the epsilon-greedy strategy, where epsilon is the percent of the time that the agent takes a randomly selected action rather than taking the action that is most likely to maximize reward given what it knows so far. We usually start with a lot of exploration (i.e. a higher value for epsilon say 1). Over time, as the agent learns more about the environment and which actions yield the most long-term reward, it would make sense to steadily reduce epsilon until it reached some predetermined minimal value (e.g., 0.1)

With every episode we set the new epsilon as 0.99*old epsilon value. However, when it reaches 0.1 we don't decrease it further. The decay is shown in Figure 3.

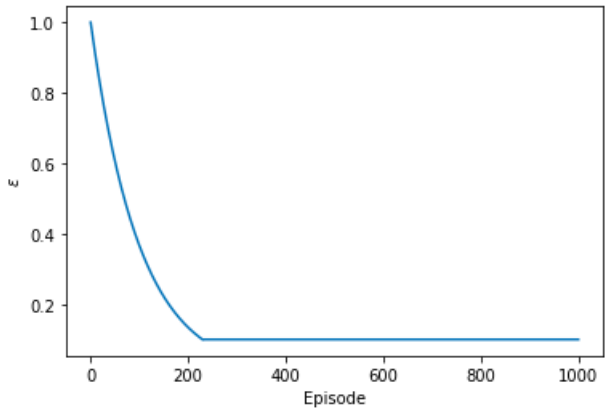

Figure 3: Decay of Epsilon Vs Episode

2.4 Policy Determination

Our agent will randomly select its action at first by a certain percentage, called *exploration rate* or *epsilon* discussed in the above section. This is because at first, it is better for the agent to try all kinds of routes before it starts to see the patterns. We select a random uniform number. If it's less than epsilon, we return the random choice action space.

When it is not deciding the action randomly, the agent will predict the reward value based on the current state and pick the action that will give the highest reward using the following formula:

$$\pi(s_t) = \underset{a \in A}{\operatorname{argmax}} Q_\theta(s_t, a)$$

3 Experimental Setup

3.1 Environment

Environment is the world through which the agent moves, and which responds to the agent. The environment takes the agent's current state and action as input, and returns as output the agent's reward and its next state.

In our project the environment is a basic deterministic n * n grid-world environment (the initial state for an 4 * 4 grid-world is shown in Figure 4) where the agent (shown as the green square) has to reach the goal (shown as the yellow square) in the least amount of time steps possible.

The agent will work within an action space consisting of four actions: up, down, left, right. At each time step, the agent will take one action and move in the direction described by the action. The agent will receive a reward of +1 for moving closer to the goal and -1 for moving away or remaining the same distance from the goal.

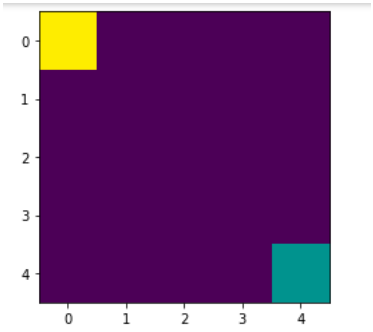

Figure 4: Environment for the project

3.2 Results and Discussions

We first ran experiments for random and heuristic agents. For an agent who takes actions randomly, it never reaches the goal state. The heuristic agent however, follows a predefined path without any randomness and reaches the goal. However this agent is just following instruction without applying its own intelligence and also does not explore the environment.

With these two agents we compare our Q-Learning agent. The Q-Learning agent learns with every episode of the game and finally reaches the goal within an optimum time. It both explores and exploits the environment. The path followed by this Q-Learning agent is shown in Figure 5.

Figure 6 shows how the rewards is increasing with every episode of the game played. It is explainable as the Q-Learning agent gains experience with the episode of each game playing, it goes on to update its Q-Table and thus make better decisions with every passing episode. This trend of increasing total rewards per episode is shown in Figure 6. The randomness in choice due to the epsilon results in slightly zig zag nature of the graph. Also, it is noted that the maximum rewards is close to 8 since the total number of moves of agent from initial state to final is 8.

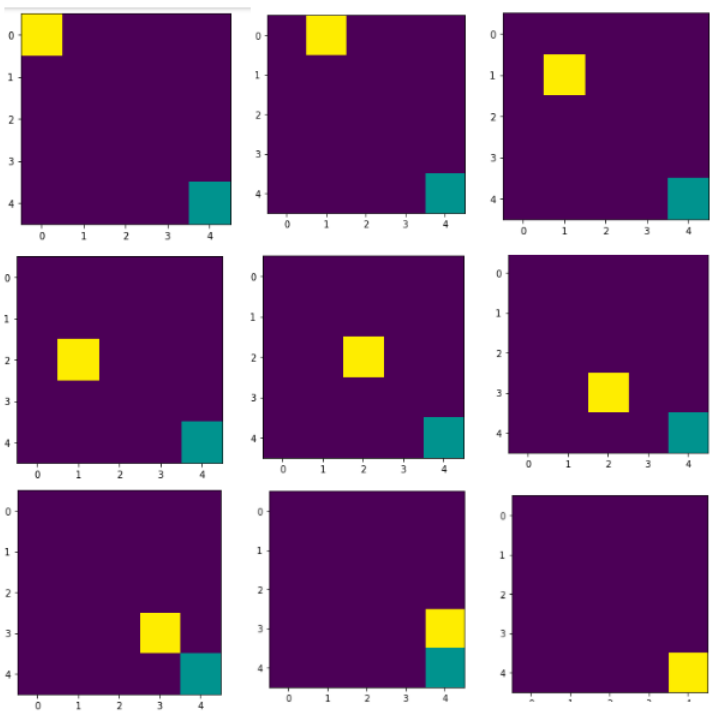

Figure 5: Q-Agent playing Game

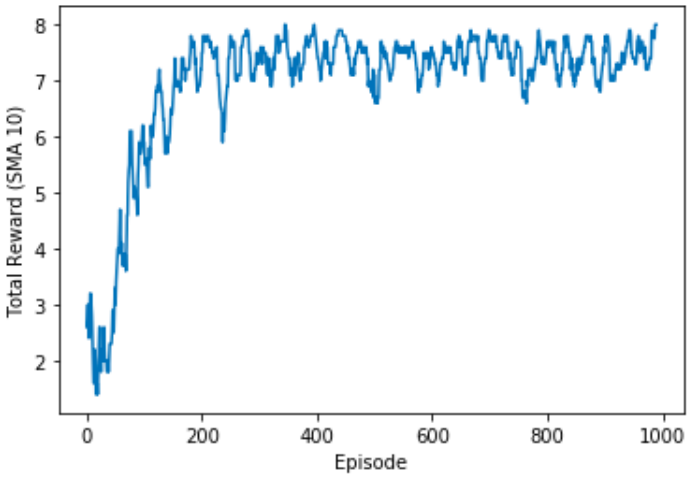

Figure 6: Rewards vs Episode

4 Conclusion

In this project we have implemented a Q-Learning agent. We observed how it is better than a random or heuristic agent. With every episode played the agent learned more about its environment and was able to maximize its rewards. After successful implementation the agent was able to intelligently explore and exploit the environment to reach the goal state.

5 References

[1] Pattern Recognition and Machine Learning by Christopher M. Bishop
[2] https://medium.com/machine-learning-for-humans/reinforcement-learning-6eacf258b265
[3] https://skymind.ai/wiki/deep-reinforcement-learning