

Date: 4th December 2018

Project 4
Reinforcement Learning

Instructor: Sargur Srihari

Name: Varun Nagaraj
UB Person Number: 50290761

Introduction to Reinforcement Learning

Reinforcement Learning is the science of making optimal decisions using experiences.

Breaking it down, the process of Reinforcement Learning involves these simple steps:

1. Observation of the environment
2. Deciding how to act using some strategy
3. Acting accordingly
4. Receiving a reward or penalty
5. Learning from the experiences and refining our strategy
6. Iterate until an optimal strategy is found

Reinforcement Learning lies between the spectrum of Supervised Learning and Unsupervised Learning, and there's a few important things to note:

Being greedy doesn't always work

There are things that are easy to do for instant gratification, and there's things that provide long term rewards. The goal is to not be greedy by looking for the quick immediate rewards, but instead to optimize for maximum rewards over the whole training.

Sequence matters in Reinforcement Learning

The reward agent does not just depend on the current state, but the entire history of states. Unlike supervised and unsupervised learning, time is important here.

Abstract:

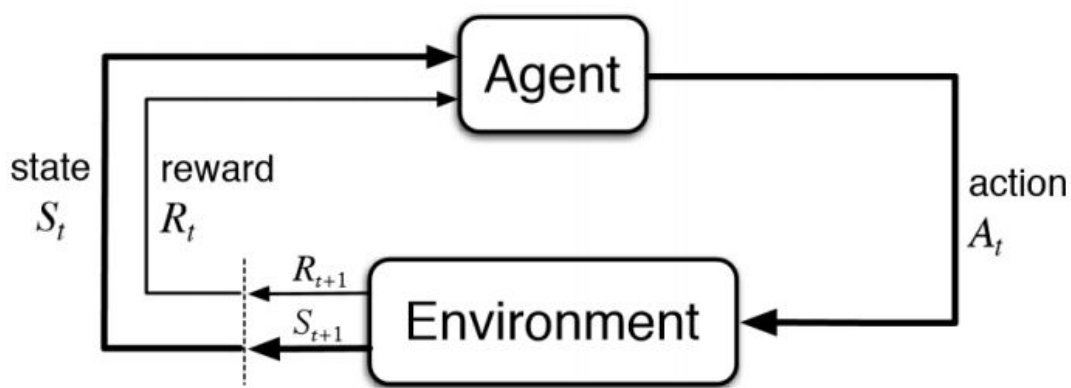
In this project we work on implementing Q-Learning method of Reinforcement Learning which utilizes a Q Matrix to learn from the previous experiences and tries to maximize the reward it can achieve and with highest accuracy. We implement Tom trying to catch Jerry in this program. We are given initial states and we randomly try to choose the next state to go to. Each transition to a new position is associated with a reward. This reward is either negative or positive depending on whether it is getting closer to the target state. If it is moving away from the target then the reward will be a negative value so that it will learn not to pick that route; if it is moving towards the target then a positive reward is associated with this transition. These two rewards reinforce the understanding over a period of time and eventually the program will learn the best route to take to reach the target in the shortest time with highest accuracy.

Markov Decision Process:

A Markov decision process (MDP) is a discrete time stochastic control process. It provides a mathematical framework for modeling decision making in situations where outcomes are partly random and partly under the control of a decision maker.

For this purpose it is useful to define a further function, which corresponds to taking the action 'a' and then continuing optimally (or according to whatever policy one currently has):

$$Q(s, a) = \sum_{s'} P_a(s, s') (R_a(s, s') + \gamma V(s')).$$



In application, we typically have an environment, which handles state and reward, and an agent, which decides which action to take given a particular state. An environment starts with some initial state s_0 , which is passed to the agent. The agent passes an action a_0 , based on the state s_0 , back to the environment. The environment reacts to the action, then passes the next state, s_1 , along with the resulting reward for taking the action, r_0 , back to the agent. This process continues until we reach a terminal state, indicating the end of an episode, at which point the process may start over again.

Discounting Factor:

The discount factor γ determines the importance of future rewards. A factor of 0 will make the agent "myopic" (or short-sighted) by only considering current rewards, i.e. r , while a factor approaching 1 will make it strive for a long-term high reward. This factor determines how fast or slow we can reach the terminal point.

What I implemented in this project?

I implemented a 3 layer Dense Neural Network with Linear output activated layer and 2 Relu activated layers for transfer of information between layers. The input layer will handle the information of coordinate positions of Tom and Jerry. This information is passed to the hidden layers where we try all the possible routes that can be chosen to maximize the reward by choosing the shortest path.

I also implemented the exponential decay formula for epsilon.

$$\epsilon = \epsilon_{min} + (\epsilon_{max} - \epsilon_{min}) * e^{-\lambda |S|}$$

where $\epsilon_{min}, \epsilon_{max} \in [0, 1]$

λ - hyperparameter for epsilon

$|S|$ - total number of steps

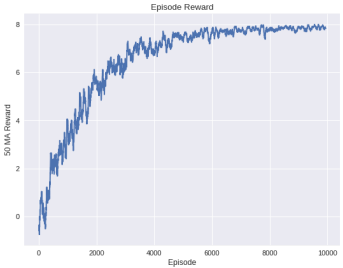
This formula result determines the exploration rate that has to be used to find the next random state to transition into.

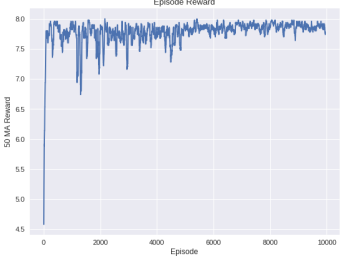
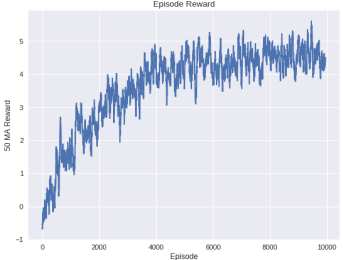
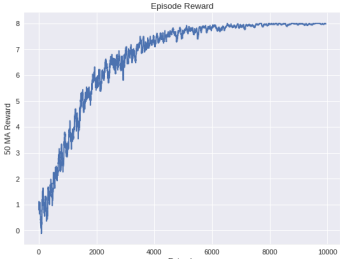
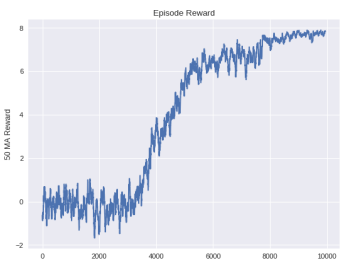
Next I implemented the Bellman equation or the Q-function which determines the new Q value which can be used to find the new position to travel to.

`t[act] = rew if st_next is None else rew + self.gamma * np.amax(q_vals_next[i])`

The above code does exactly the same. It tries to learn from the previous encounters and takes the max value and using that value it calculated the new Q value after applying the gamma function or discount factor on it.

Can these snippets be improved and how it will influence the training the agent?
And how quickly was the agent able to learn?

Max Epsilon	Min Epsilon	Lamda	Episodes	Result observation	Output image
1	0.05	0.00005	10000	Around 6000 episode onwards we were able to get highest reward of 8 continuously. Mean reward was 6.25	

1	0.05	0.005	10000	By the end of 500th episode I was able to see the terminal state being reached. But it was only after the 3500th episode that the accuracy of detection was continuously close to 8. Mean reward was 7.77	
1	0.5	0.00005	10000	Using this configuration, we were nowhere close to achieving the highest reward. The reward stagnated after the 4000th episode and the reward was between 4-5 and did not do good exploration. Mean reward was 3.603	
1	0.00005	0.00005	10000	Using this configuration we get almost perfect predictions after the 4000th episode. The learning is slow and after the 4000th episode we can see that most of the rewards are 8's. Mean reward is 6.645.	
5	0.02	0.00005	10000	Using this configuration we can see that the exploration in the starting is slow and after the 6000th episode we see the rewards reaching 8 many times. Mean rewards is 3.833	

Writing Task

1. Explain what happens in reinforcement learning if the agent always chooses the action that maximizes the Q-value. Suggest two ways to force the agent to explore.

The main issue that might occur is the agent always resorts to maximizing the Q-value is that the agent might get stuck performing non-optimal actions. Unless it has already found the optimal policy, exploring actions which do not have the highest Q-value might allow it to find a better policy.

**Shouldn't always be greedy (we won't explore much of the state space this way).
Shouldn't always be random (will take a long time to generate a good Q).**

Two ways to force exploration are as follows:

- The ϵ -greedy strategy is to select the greedy action (one that maximizes $Q[s,a]$) all but ϵ of the time and to select a random action ϵ of the time, where $0 \leq \epsilon \leq 1$.
- An alternative is "optimism in the face of uncertainty": initialize the Q -function to values that encourage exploration. If the Q -values are initialized to high values, the unexplored areas will look good, so that a greedy search will tend to explore.

2. Q Matrix Calculation

Q-Matrix Calculation

	U	D	L	R
S_0	2.9403	3.9404	2.9403	3.9404
S_1	1.9701	2.9701	0.9701	2.9701
S_2	-0.01	1.99	-0.01	1.99
S_3	-1	1	-1	0
S_4	0	0	0	0

$$Q(S_3, R) = 0 + 0.99(Q_{\max}(S_4, a)) \\ = 0.99(0) \\ = 0$$

$$Q(S_3, L) = -1 + 0.99(Q_{\max}(S_4, a)) \\ = -1$$

$$Q(S_3, U) = -1 + 0.99(0) \\ = -1$$

$$Q(S_3, D) = 1 + 0.99(0) \\ = 1$$

$$Q(S_2, R) = 1 + 0.99(Q(S_3, a)) \\ = 1 + 0.99(1) \\ = 1.99$$

$$Q(S_2, L) = -1 + 0.99(1) \\ = -0.01$$

$$Q(S_2, U) = -1 + 0.99(1) = -0.01$$

$$Q(S_2, D) = 1 + 0.99(1) = 1.99$$

$$Q(S_1, L) = -1 + 0.99(Q_{\max}(S_2, a)) \\ = -1 + 0.99(1.99) \\ = 0.9701$$

$$Q(S_1, R) = 1 + 0.99(1.99) = 2.9701$$

$$Q(S_1, U) = 0 + 0.99(1.99) = 1.9701$$

$$Q(S_1, D) = 1 + 0.99(1.99) = 2.9701$$

$$Q(S_0, L) = 0 + 0.99(2.9701) \\ = 2.9403$$

$$Q(S_0, R) = 1 + 0.99(2.9701) \\ = 3.9404$$

$$Q(S_0, U) = 0 + 0.99(2.9701) \\ = 2.9403$$

$$Q(S_0, D) = 1 + 0.99(2.9701) \\ = 3.9404$$

References:

1. https://www.utdallas.edu/~nrr150130/cs6375/2015fa/lects/Lecture_25_RL.pdf
2. <https://blog.goodaudience.com/reinforcement-learning-lane-change-algorithm-part-1-1451d00868ee>
3. https://ublearns.buffalo.edu/bbcswebdav/pid-4776954-dt-content-rid-21119813_1/course/s/2189_24904_COMB/project4_description.pdf