Reinforcement Learning

Kiran Prabhakar - kprabhak - 50287403

Abstract

The purpose of this project is to demonstrate the reinforcement learning task in which the agent(Tom) learns to reach its target (Jerry) in the shortest path possible on a 25(5X5) states environment.

1 Introduction

The current learning task is performed by implementing a Deep neural network, where the training data is obtained initially by exploring the environment using the discounting factor γ. Once the exploration is completed, the decisions for the next states are exploited using the predictions of the network.

2 Definitions

2.1 Reinforcement Learning

Reinforcement learning is a machine learning technique, where the agent learns in an interactive environment using trail and error approach.
For the action taken by the agent a positive or a negative reward will be given. Unlike supervised learning, the reinforcement learning will create the training data by exploring the environment and will exploit it in the future to attain rewards. The important components of reinforcement learning

- **Environment**: The physical entity where the agent takes the actions
- **State**: Current situation of the agent
- **Reward**: The positive or negative feedback given by the environment.
- **Policy**: The control action sequence taken by the agent to reach the target.

2.2 Reward process

As the environment on which the agent acts is stochastic, its ideal to value the short term reward more when compared to long term reward. By using the discounted reward process, instead of just summing up the individual reward of every future state, we calculate the discounted reward as below.

$$R_t = r_t + \gamma r_{t-1} + ... + \gamma^{n-t} r_n$$
$$R_t = r_t + \gamma R_{t-1}$$

γ value should be carefully chosen so that the reward attained at each step is maximum.

2.3 Value function

Value function states how good a state is. Irrespective of actions taken, if the agent is in state S, then it would get an reward (r). If the agent follows the policy π, then the value function for the policy is

Preprint. Work in progress.

defined as below.

$$V^{\pi}(s) = E(\sum_{t0} \gamma^t r_t)$$

The optimal value function is given by.

$$V^*(s) = maxV^{\pi}(s)$$

2.4 Q function

The agent can't control what state he ends up in, it depends on the actions taken in the current state. So as to determine this the Q function is used, it accepts the current state and the current action taken in the state and determines the reward of the by recursively computing the Q values of the future state. The Q function is calculated using the below formula.

$$Q_{t+1}(s_t, a_t) = Q_{t+1}(s_t, a_t) + \alpha(r_{t+1} + \gamma maxQ_t(s_{t+1}, a) - Q_t(s_t, a_t))$$

The above equation is called Bellman equation, which tells the maximum award the agent receives when it enters a state.
The learning rate (α) is decayed every episode and can also be considered as one. This is because, as the agent learns more and more, it believes there is not more left to learn in the environment. So the modified equation is as below

$$Q_{t+1}(s_t, a_t) = r_t + \gamma maxQ_t(s_{t+1}, a)$$

A Q table is constructed to keep track of the Q values calculated which is usually a **no. of states X no. of actions** array

2.5 Exploration Vs Exploitation

To maintain trade-off between exploration and exploitation, we use **epsilon** ϵ, in a greedy way. The value of ϵ is between 0 and 1.
The strategy of having high value of epsilon is to explore a lot initially and then use exponentially reduce the value of ϵ using the below formula. This will allow us to exploit more in future.

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

2.6 Markov Decision Process

Markov Decision Process is based on Markov property i.e. the future is independent of past given the present. This means the actions taken in future will completely depend on the present state and doesn't depend on the past history. This is because at every state the Markov property is followed. The mathematical reprsentation of Markov Decision Process is a below.

$$P_{ss'} = P[S_{t+1} = s' | S_t = s]$$

2.7 Markov Reward Process

Markov Reward Process is a value judgment strategy saying how much reward is accumulated by analyzing the particular sequence tat was sampled from the observation.

2.8 Experience reply and DQN

Using Deep Neural Network with Q function and Q table for reinforcement learning is called Deep Q Network. To solve the DQN experience relay strategy is used, where experiences including the state transitions, rewards and actions stored and then exploited in mini batches to update the neural networks.

3 Implementation details:

3.1 DQN in current project:

In the current project DQN is implemented using kears library. The network uses below setting to complete the reinforcement process for the game.
Below are the hyper parameters used in setting the network

Hidden layers	Activation	Nodes	Optimizer
2	Relu	128	RMS prop

3.2 Classes Used:

The current project uses various classes as below:

3.2.1 Environment

This class is responsible for setting up the environment for the agent to act up on. It has the below methods.

- **rest**: Will re-initialize the environment for and sets it up ready for next episode
- **step**: The method returns the reward for the action by accepting the action in the current step.
- **render**: Mostly used for debugging purpose to know the current state of the environment.

3.2.2 Memory

Used to store the Q-table for the learning process. The samples property holds the Q-table details.

- **add**: To add a exploration details to the Q-table. One record will be memory will be in the below format (current state, action taken, reward, next state)
- **sample**: Returns the random records from the memory that are to requested.

3.2.3 Agent

Holds the methods needed for the agent to take actions

- **act**: Takes the current state as input and based on the ϵ value a random action or a predicted action will be taken. This method will return the action that is taken in the current step.
- **observe**: Stores the observed results in the memory.
- **reply**: The batch data is sampled form the memory and the learning process is carried out by using the Q function.

3.3 Parameter Tuning:

3.3.1 Neural Network Parameters:

Hidden layers	Activation	Mean Reward	Training Time (sec)
2	Linear	3.90	849.76
2	Relu	6.21	834.80
4	Relu	5.16	980.96

Thus from the above table it is evident that the 2 layer neural network with relu activation function gives desired output.

3.3.2 Agent Parameters:

The current project has various parameters like λ, ϵ (max and min), λ and the number of episodes.

- **episodes**: For episodes the initial value provided was 10000. However, the agent could reach the target after completing around 7000 episodes, by taking approximately 0.24 sec. for each episode.
- **Exploration factor** ϵ: This parameter allows the environment to explore a lot in the agent. Irrespective of the min and max ϵ values, as the agent starts with '0' steps, the $|S| = 0$, which leads to maximum ϵ = 1 in the initial state. As the agent keeps learning the value exponentially reduces. The speed of decay of the ϵ is determined by λ parameter.

The various parameters that are tweaked in the current project and the corresponding results are as below.

Number of episodes	Lambda (λ)	Discounting factor(γ)	Mean Reward	Training Time (sec)
10000	0.00005	0.99	6.21	834.80
5000	0.05	0.5	7.88	371.59
3000	0.05	0.5	7.90	225.88

3.3.3 Decay function Parameters:

In addition to exponential decay, linear decay and quadratic decay are tried out in the current project.

For linear decay, the equation for epsilon decay used is

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

For simple product of static epsilon and also the decay factor gives.

$$\epsilon = \epsilon_{max} * -\lambda|S|$$

The below table are the various decay function results for 3000 episodes

Decay function	Final Epsilon(ϵ)	Mean Reward	Training Time (sec)
Exponential	0.05	7.98	334.80
Linear	-226.71	7.21	490.21
Simple Product	-5.80	6.12	420.98

3.4 Writing Task

3.4.1 Question 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 first part of question emphasizes on the problem of choosing the action by taking greedy approach for the Q-value.
- If the learning strategy chooses the action that maximizes the Q-value, then the agent might end up choosing the actions that are not optimal.
- Exploring the actions that do not have highest Q-value might allow the agent to take random actions and thus knows more about the environment.
- Until the agent finds the global **optimal policy** it should be allowed to explore more by **ignoring the local optimal policy** which are obtained by initially choosing the action with maximum Q-values.

The two approaches that are used to find the optimal policy are:

- **value Iteration**: Value iteration approach helps in finding the optimal value function and extracting the policy out of it. The value iteration will do the one optimal policy extraction and one optimal value function calculation.
- **Policy Iteration**: The policy iteration approach will redefine the policy at every step and take new policy until the policy converges to the optimum.

The second part of the question is about finding two ways to force the agent to explore more on the environment. This can be obtained by

1. **Increasing the epsilon value**, the exploration factor(ϵ). This will allow the agent to explore all possible patterns in the environment before it ends up with an optimal policy.
2. The other approach is to provide the **more probability** for selecting the random state than a predicted values. This is similar to giving **more weights** to the random events than to the predicted events

3.4.2 Question 2:

Given value of $\gamma = 0.99$. Action towards the target will give reward of **+1** Action away from the target will give reward of **-1** If no action is taken, then the reward is **0** The Q function used is:

$$Q_{t+1}(s_t, a_t) = r_t + \gamma max Q_t(s_{t+1}, a)$$

For convenience, we use **a_i** to represent all actions possible at a state

3.4.3 State S4:

Initially we start to fill the Q table from the S4 state. As it is the final state and the target is reached, we don't take any actions are taken and the Q values are taken as 0

State	UP	Down	Left	Right
0				
1				
2				
3				
4	0	0	0	0

3.4.4 State S3:

Now in S3 state, as per the given figure the optimal policy is to move down.
Moving Down:

$$Q(S_3, D) = 1 + \gamma max(Q(S_4, a_i))$$
$$Q(s_3, D) = 1 + 0 = 1$$

Moving right:
As there is no action is taken. The Q value is,

$$Q(S_3, R) = 0 + \gamma max(Q(currentstate))$$
$$Q(S_3, R) = 0 + 0.99 * 1 = 0.99$$

Moving Left:

$$Q(S_3, L) = -1 + \gamma max(Q(s2, a_i))$$

As we are not aware of **S2** will assume it as Q(S_2), which will the max value in the S2 row

Moving up:
This will lead to the state which will is symmetric to S2 state, so the Q value will be same as it is for moving right case (i.e - Q(S_2)

Thus the Q table for the **S3** is as below

State	UP	Down	Left	Right
0				
1				
2				
3	Q(S_2)	1	Q(S_2)	0.99
4	0	0	0	0

3.4.5 State S2:

Moving Right or Down
The optimal policy says the agent has to move right or down. So due the symmetry both right and left actions will lead to same Q value

$$Q(S_2, D) = Q(S_2, R) = 1 + \gamma max(Q(S_3, a_i))$$
$$Q(S_2, D) = Q(S_2, R) = 1 + 0.99 * 1$$
$$Q(S_2, D) = Q(S_2, R) = 1.99$$

Moving Left or Up
As the two movements leads to symmetric states, the Q values will be same.

$$Q(S_2, U) = Q(S_2, L) = -1 + \gamma max(Q(S_1, a_i))$$

As we are not sure of the Q value at **S1** let us assume it as Q(S1), which will the max value in the S1 row.
Now as the value of max S2 row is know, we can fill the values for Q(S_2) in the Q table.

$$Q(S_2) = Q(S_3, L) = -1 + 0.99 * max(Q(s2, a_i))$$
$$Q(S_2) = Q(S_3, L) = -1 + 0.99 * 1.99$$
$$Q(S_2) = Q(S_3, L) = -1 + 0.99 * 1.99$$
$$Q(S_2) = Q(S_3, L) = 0.97$$

So the Q table after **S2** state will be as below.

State	UP	Down	Left	Right
0				
1				
2	$Q(S_1)$	1.99	$Q(S_1)$	1.99
3	0.97	1	0.97	0.99
4	0	0	0	0

3.4.6 State S1:

Moving down or right:
Due to symmetry moving down or right is same.

$$Q(S_1, D) = Q(S_1, R) = 1 + \gamma max(Q(S_2, a_i))$$
$$Q(S_1, D) = Q(S_1, R) = 1 + 0.99 * 1.99$$
$$Q(S_1, D) = Q(S_1, R) = 1 + 1.9701$$
$$Q(S_1, D) = Q(S_1, R) = 2.9701$$

Moving Up:
As there is no state when moved up, the agent remains in the same state. The Q value is calculated as below

$$Q(S_1, U) = 0 + \gamma max(Q(currentstate))$$
$$Q(S_1, U) = 0 + 0.99 * 2.9701 = 2.94$$

Moving Left: As the Q value of the future state is not known, we shall consider it as $Q(S_0)$

From the current states optimal value, we can fill the Q table for S2 state. The values for the unknowns in S2 state is

$$Q(S_2, U) = Q(S_2, L) = -1 + \gamma max(Q(S_1, a_i))$$
$$Q(S_2, U) = Q(S_2, L) = -1 + 0.99 * 2.9701$$
$$Q(S_2, U) = Q(S_2, L) = -1 + 0.99 * 2.9701$$
$$Q(S_2, U) = Q(S_2, L) = 1.940$$

Therefore the Q table after S1 state is

State	UP	Down	Left	Right
0				
1	2.94	2.9701	$Q(S_0)$	2.9701
2	1.94	1.99	1.94	1.99
3	0.97	1	0.97	0.99
4	0	0	0	0

3.4.7 State S0:

Moving Right or Down:
The reward for the both actions is same as the states are symmetric.

$$Q(S_0, D) = Q(S_0, R) = 1 + \gamma max(Q(S_1, a_i))$$
$$Q(S_0, D) = Q(S_0, R) = 1 + 0.99 * 2.9701$$
$$Q(S_0, D) = Q(S_0, R) = 3.94$$

Moving Up or Left: These two actions will make the agent to remain in the same state.

$$Q(S_0, U) = Q(S_0, L) = -1 + \gamma max(Q(currentState))$$
$$Q(S_0, U) = Q(S_0, L) = -1 + 0.99 * 3.94$$
$$Q(S_0, U) = Q(S_0, L) = 2.90$$

Using the above value we can fill the $Q(S_0)$ for the S_1 state. So the final Q table is as below.

State	UP	Down	Left	Right
0	3.90	3.94	3.90	3.94
1	2.94	2.9701	2.90	2.9701
2	1.94	1.99	1.94	1.99
3	0.97	1	0.97	0.99
4	0	0	0	0

Thus the Q table is filled.

References

[1] https://towardsdatascience.com

[2] https://medium.com/data-science-group-iitr

[3] Lecture slides