Reinforcement learning

Reinforcement learning is a subset of machine learning that allows an AI driven system (an agent) to learn through trial and error using the results from previous actions made. The results can either be good or bad, either a reward or a punishment, where our aim is to get the optimum reward function. Reinforcement learning learns from the errors made by the agent, and it corrects itself to correct the outcome. The goal of reinforcement learning is to find the best action to maximise the total cumulative reward for the RL agent. No data is needed to be inputted for training. The agent solves the problem by understanding the environment and taking its own moves.

Agent

The agent in RL is the one who makes decisions based on the rewards and punishments.

Environment

The environment is where the agent moves around, it is also where the action and state are taken, and then next state is returned with an according reward depending on the outcome.

State

The certain position of an agent is known as its state. The state can either be the current position or any future positions of the agent.

Q-Learning

(This code was implemented from Vdiya Analytics)

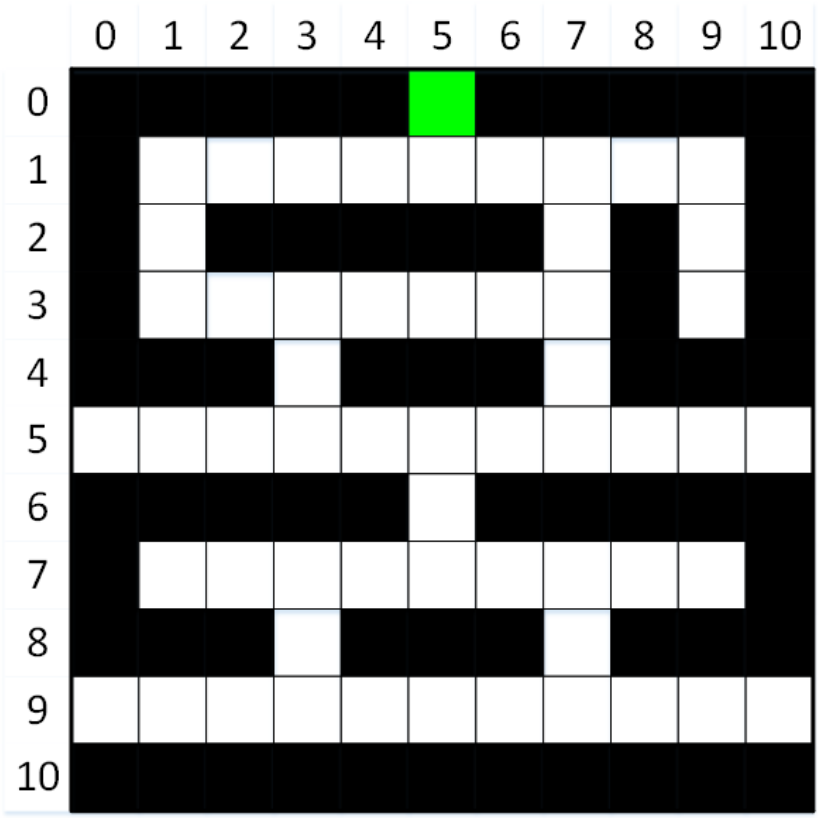
Q- learning is an algorithm of reinforcement learning which is used as it is a model-free, value-based, off-policy learning algorithm.

Using NumPy, the environment is a (11 x 11) grid which represents a warehouse. The aim for the agent is to travel around the grid to reach the goal. The agent can start on any white tile, and it should navigate itself to the green tile which is the end goal in the shortest number of moves.

In the warehouse the black tiles can store items and the white tiles are the path the agent can take. If the agent reaches the end goal or any of the black tiles, the algorithm will terminate and will begin another episode.

The optimal policy is determined by identifying the high rewards (the end goal) and the low rewards (going around the environment randomly) in the least number of steps.

Here is a visual representation of the grid:



The green tile represents the end and the optimal goal which has a reward of 100. The white tile represents a reward of -1 and the black tile represents a reward of -100.

The agent will be allowed to move in 4 different directions: Left, Right, Up and Down. Q-learning will be applied to this model. During learning, there can be random actions every few episodes however we need to try and reduce the probability of random actions as with deep learning we want the optimal outcome.

From the grid, there are 121 different possible positions that the agent could be at. At each state, the agent will be given 4 different directions to move in. This gives the agent a possibility of 484 (121 x 4) different direction changes in total. Of the 121 possible positions, 64 are not possible, indicated by black tiles, 57 are possible indicated by white tiles and a green tile. These movements will be our weights which will be updated according to the moves taken. The agent will learn to avoid the black tiles.

An agent observes an environment. At each time-step the agent gets some representation of the environments state. Given the representation the agent suggests an action to take. The environment is transitioned to a new state and the agent is given a reward. The goal of the agent is to maximise the total number of rewards, cumulatively.

State transition

From the grid, a state is just defined as a tile and the remaining tiles which the agent is not on is known as the available actions. The agent is randomly a start state, and the end goal is reaching the green tile at $S_{(0, 5)}$.

The state transition structure is given by $s_{t+1} = \delta(s_t, a_t)$ where the next state (s_{t+1}) is a function of is current state, s, and its corresponding action, a.

Action

The action is the set of possible moves the agent can make within a given environment. The action space $a \in A$, where A is made up of 4 actions that the agent can take, A = {Left, Right, Up and Down}.

Terminal state

The environment has a terminal state when the agent reaches the green tile.

Rewards

The final component of the environment that we need to define is the rewards. To assist in the learning of the agent, each state of the warehouse must be given a reward value. The agent can start at any white tile, but it has the same goal which is to maximise its cumulative reward. There are also negative rewards, punishments, which are at all the states apart from the end goal.

	0	1	2	3	4	5	6	7	8	9	10
0	-100	-100	-100	-100	-100	100	-100	-100	-100	-100	-100
1	-100	-1	-1	-1	-1	-1	-1	-1	-1	-1	-100
2	-100	-1	-100	-100	-100	-100	-100	-1	-100	-1	-100
3	-100	-1	-1	-1	-1	-1	-1	-1	-100	-1	-100
4	-100	-100	-100	-1	-100	-100	-100	-1	-100	-100	-100
5	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
6	-100	-100	-100	-100	-100	-1	-100	-100	-100	-100	-100
7	-100	-1	-1	-1	-1	-1	-1	-1	-1	-1	-100
8	-100	-100	-100	-1	-100	-100	-100	-1	-100	-100	-100
9	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
10	-100	-100	-100	-100	-100	-100	-100	-100	-100	-100	-100

To maximise the cumulative rewards, by minimising the cumulative punishments, the agent will need to identify the shortest path between the end goal and all other states in the warehouse where the agent can travel. As well as this, the agent will need to learn to prevent any collisions with the storage locations, the black tiles.

The reward is defined by $r_{(t+1)} = r(s_t, a_t)$, where r is the reward received depending on the given state and the given action.

Q-Learning parameters

The q-learning update rule has two parameters, that can be modified to impact the performance of the algorithm.

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

The main parameters for Q-Learning are learning rate (α), discount factor(γ), exploration rate(ε) and the learning policy(π)

1. The learning rate(α) where (α) is the element [0, 1] which shows the rate at which we want to update the current q-value with the new q-value. In other words, how quickly the agent abandons the previous q-value for the new q-value. The greater the value of alpha, the greater the learning rate.
2. The discount rate(γ) where (γ) is the element [0, 1] which shows the weight that the agent places on future rewards. The discount rate will be the rate for which we discount future rewards and will determine the present value for future rewards. The closer to 1, the more the agent has to be forced to work towards a long-term reward. The closer to 0, the less decisions have to be made to obtain future rewards.
3. Learning policy (pi) = epsilon greedy, this is a policy which is decided from the initial exploration and will become exploitation after the completion of each episode due to the epsilon decay factor.

Initially, we have

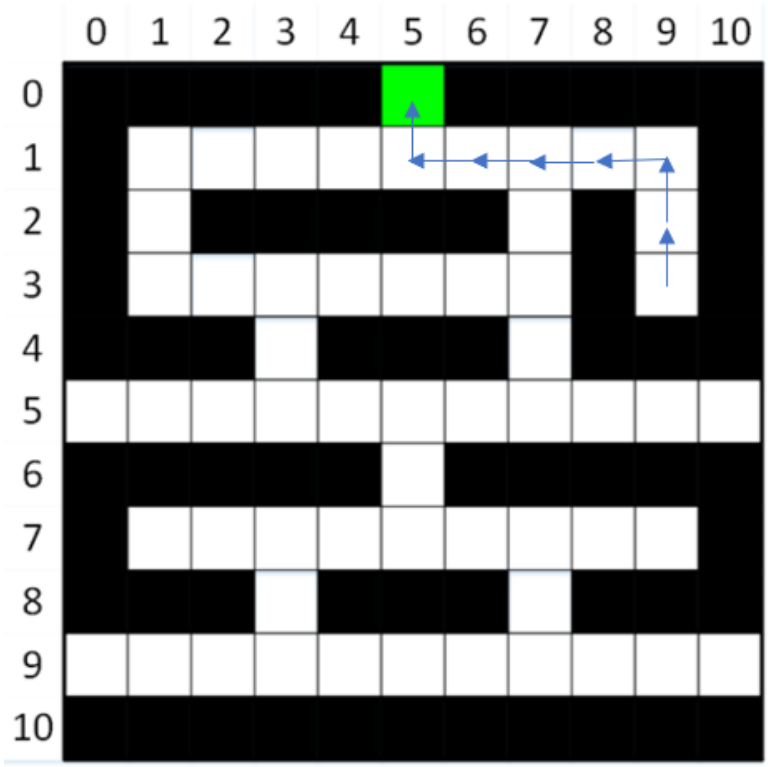
Alpha= 0.9

Gamma = 0.9

Epsilon = 0.9

Learning Episode

For a random episode beginning at [row 3, column 9], the route will be [2, 9] --> [1, 9] --> [1, 8] --> [1, 7] --> [1, 6] --> [1, 5] --> [0, 5] where our goal is finally reached as shown in the diagram below.



Advanced Task

Cartpole-v0 is used from the Open AI environment for the Advanced task. In Cartpole, the agent gets to decide between two options to take, moving either left or right, with the aim being to keep the pole balanced. There are four possible states (position, velocity, angle of the pole and the velocity of the base of the pole. The agent observes the current state of the environment and gets rewarded (+1) for every incremental timestep. If the pole is at an angle more than* degrees from the vertical axis,or if the cart moves more than 2.4 units away from the centre, this means the episode would have terminated.

Here we will look at applying Deep Q-Learning algorithm (DQN) and Double Q learning.

Replay memory

To train our DQN we will be using experience replay memory. Replay memory stores the transitions that the agent observes, allowing us to reuse this information at a later stage. As it is randomly sampled, the transitions which are built up as a batch are not correlated. This is used as for stability and makes training much better for the DQN.

Transition – is defined by the tuple, $e_t = (s_t, a_t, s_{t+1}, r_{t+1})$. What it does is, maps the state, action pair to the next state and reward.

Replay Memory – A cyclic buffer of bounded size that holds the transitions observed.

Choosing the policy (Exploitation Vs Exploration)

We define an exploration rate, epsilon, which is initially set to 1 (epsilon = 1)

As the agent learns more about the environment will begin to decay by a certain rate so the agent is less likely to explore the agent will become greedy and exploiting the environment once it has had that change to explore their environment.

To see if the agent will choose exploration or exploitation, we generate a random number (r) between zero and one.

If r > epsilon: The agent will choose exploitation.

Choose the action with the highest Q value from the Q-learning table.

If r < epsilon: The agent will choose exploration.

Randomly choosing the action and seeing what happens in the environment.

Initially as epsilon = 1, the agent will first explore the environment. The agent learns an optimal policy, in this case, the shortest route to the reward.

Deep Q Learning (DQN)

(This code was implemented from PyTorch)

DQN is a deep reinforcement learning algorithm which is a combination of Q-learning and neural networks that is used to prevent overestimation and give more realistic estimation error. The environment is deterministic.

Our aim is to train a policy that attempts to maximise the discounted cumulative reward, also known as the return (R_t). As mentioned earlier the discount, gamma, should be between 0 and 1 to ensure that the sum converges.

The main idea of Q-learning is if we had a function Q* = State x Action, that could give us the return if were to take an action given its state, then we could create an optimal policy which maximises our reward.

$$\pi^*(s)=\text{argmax}\ Q^*(s,a)$$

However, we don't know what Q* is but we can create and train it to be Q*.

For our training, we'll use the update rule based of Bellman equation:

$$Q_\pi(s,a)=r+\gamma Q_\pi(s',\pi(s'))$$

The difference between both sides is calculated to be the temporal difference error:

$$\delta=Q(s,a)-(r+\gamma\text{max}Q(s',a))$$

Double Deep Q-Learning

Double Deep Q-Learning is made up of two value functions, the estimator, and the target network. The estimator is trained, and the target network is worked out to find the target values. The weights that are produced from the estimator are then passed onto the target network to keep it updated. From here, the replay memory is used to train the neural network. They are randomly chosen in fixed intervals known as batches. Using this, the agent can learn from the experiences and can have better decision making.

We use experience replaying reinforcement learning to reduce the amount of exploration needed. This method requires a lot of memory but works out cheaper than the agents experience within the environment. A developed feature of experience replay is prioritised experience replay which has a goal of prioritising some transitions more than other which are more relevant this then helps to calculate the temporal difference error.

Reference

University of York. 2022. *What is reinforcement learning? - University of York*. [online] Available at: <https://online.york.ac.uk/what-is-reinforcement-learning/> [Accessed 24 April 2022].

Youtube.com. 2022. [online] Available at: <https://www.youtube.com/watch?v=nyjbcRQ-uQ8&list=PLZbbT5o_s2xoWNVdDudn51XM8lOuZ_Njv> [Accessed 24 April 2022].

VidyaAnalytics, 2022. Q – Learning Algorithm with Step by Step Implementation using Python. Available at: <https://www.analyticsvidhya.com/blog/2021/04/q-learning-algorithm-with-step-by-step-implementation-using-python/> [Accessed 24 April 2022].

Wikipedia, 2022. Q-Learning. Available at: <https://en.wikipedia.org/wiki/Q-learning> [Accessed 24 April 2022].

Pytorch.org. 2022. *Reinforcement Learning (DQN) Tutorial — PyTorch Tutorials 1.11.0+cu102 documentation*. [online] Available at: <https://pytorch.org/tutorials/intermediate/reinforcement_q_learning.html> [Accessed 24 April 2022].