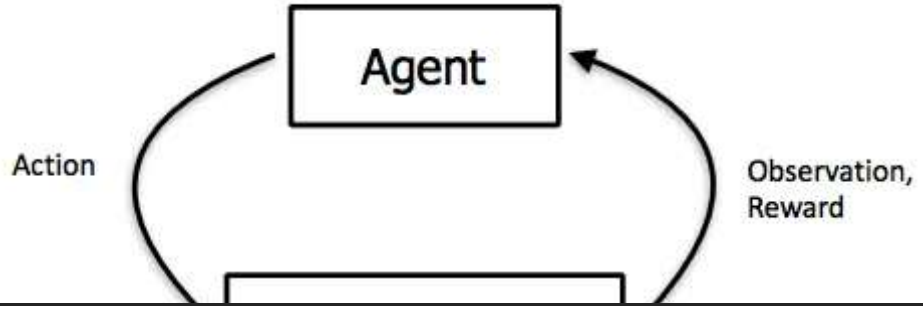
Reinforcement Learning with Q tables

Mohit Mayank · Follow

Published in ITNEXT · 8 min read · Mar 2, 2018



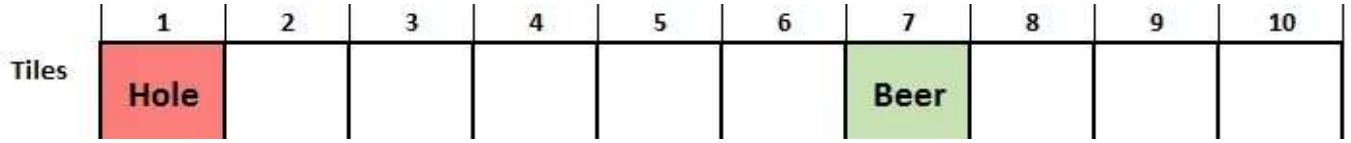
What is reinforcement learning

Reinforcement learning is an area of machine learning dealing with delayed reward.

What does this means? Well, simple, let me explain this with an example. For this I am assuming you have heard (better if you know) about neural networks or even a basic knowledge of regression or classification will do. So lets take an example of classification problem, you have been given a large chunk of images of dogs, and you have to design a system which will be able distinguish between an image by saying if it is of dog's or not. Anyone with a little knowledge of machine learning will advice you to use convolution neural network and train with the provided images, and yeah it will work. But how? Well, without going into details (maybe an article on this later?!) you train the neural network on sample images first. While training the neural network learns the little features and pattern unique to dog's image. During training you know the expected output, it is a dog images, so whenever the network predicts wrong we correct it. In a way, we know the reward for the provided images, if the prediction is right, we give positive

But what if we don't know the immediate rewards? Here, reinforcement learning comes into the picture.

To explain this, lets create a game. The game is simple, there are 10 tiles in a row. All tiles are not equal, some have hole where we do not want to go, whereas some have beer, where we definitely want to go. When the game start, you can spawn on any of tiles, and can either go left or right. The game will go on unless either we have won or its game over, lets call each such iteration an episode.



Go for beer!!

So, if you spawn on the 0th tile or somehow travels to 0th tile, its game over but if we travel to tile 6, we win.

Let's take one simple episode as example. Ok lets say we spawn on tile 2. Now suppose I haven't shown you the game map and you only have the

with hole and you lose. This is not what we want to happen, so lets assign a negative reward to our action of going left from 2 to 1 to 0. In next episode by some chance you spawn in tile 2 again, this time you keep on going right, until you reach tile 6. Here we got the beer, lets assign the actions with positive reward.

What we learned? for every step we take, until we hit the hole or beer, we don't know of the rewards. Delayed rewards guys. There is no one who tells you the right direction, after every step there is no reward, suggesting its the right or wrong direction. Now its even difficult for us to grasp the sense of right actions, what if we want the computer to learn this? Reinforcement learning to the rescue.

Markov Decision Process

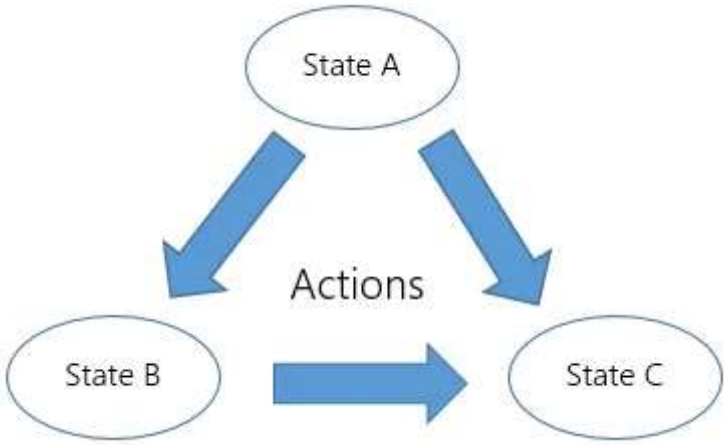
Now what is this Markov process and why do we need to learn it? Well I thought the same and to be clear, we don't need to deep dive into it, just a basic intuition would do.

So, markov decision process is used for modeling decision making in

the negative & positive reward we discussed can be modeled by markov process.

A markov decision process consist of,

1. **State (S)**: It is a set of states. Tiles in our example. So we have 10 states in our game.
2. **Action (A)**: It is a set of actions available form state . Left & right from our game. `s`
3. **Probability of transition** : It is the probability of transition to state at time if we took action in state at time . We were kinda sorted on this front, a left from tile 3 leads to tile 2, no question asked. `P(s'|s, a) s' t+1 a s t`
4. **Reward** : It is the reward we receive if we transition from state to state by taking action . `R(s'|s, a) s s' a`
5. **Discount (Y)**: It is the discount factor, which represents the difference in future and present rewards.



Change of state due to some action. As simple as that!

So markov process can be understood as a collection of states with some actions possible from every state with some probability . Each such action will lead to some reward . If the probability and rewards are unknown, the problem is of reinforcement learning. Here we are going to solve a simple such problem using Q Learning or better the most basic implementation of it, the Q table. S A P R

Q learning

Now taking all the above learned theory in consideration, we want to build an agent to traverse our game of beer and holes (looking for better name)

better our chances of winning the game, hence the name Q (quality) learning. The quality of our policy will improve upon training and will keep on improving. To learn, we are going to use the bellman equation, which goes as follows,

$$Q(s,a) = r + \gamma(max(Q(s',a'))$$

the bellman equation for discounted future rewards

where,

- `Q(s,a)` is the current policy of action from state `a s`
- `r` is the reward for the action
- `max(Q(s',a'))` defines the maximum future reward. Say we took action at state to reach state . From here we may have multiple actions, each corresponding to some rewards. The maximum of that reward is computed. `a s s'`
- `Y` is the discount factor. Now the value varies from 0 to 1, if value is near 0 immediate reward is given preference and if value goes near 1

Here, we are trying to formulate the delayed rewards into immediate rewards. For every action we take from a state, we update our policy table, lets call it Q table, to include a positive or negative reward. Say, we are in tile 4, and we are going to take a right, making tile 5 the next state, the immediate reward of tile 4 will include some factor (determined by discount) of the maximum reward of all the action possible from tile 5. And if you consult the game map, a right from tile 5 leads to tile 6 which is the ultimate goal in our game, so a right action from tile 4 is also assigned some positive reward.

Code

Ok too much theory for now, let code.

Let me define the states, actions and rewards as matrix. One way of doing it is having rows for all the state and columns for actions, so as we have 10 state and 2 actions, we will define a 10x2 matrix. For simplicity I am not using any library, just coding them with lists in python.

```
[0, 0],
[0, 100],
[0, 0],
[100, 0],
[0, 0],
[0, None]]
```

As you can see, taking right from tile 5 and taking left from tile 7 have high reward of 100 as it leads to tile 6. Also a left from tile 1 leads to hole, so it has a negative reward. Tile 0 and 9 have left and right reward as as there are no -1 or 10th tile. None

Now its time for our magic Q table, which will update as the agent learns on each episode.

```
q_matrix = [[0, 0],
      [0, 0],
      [0, 0],
      [0, 0],
      [0, 0],
      [0, 0],
      [0, 0],
      [0, 0],
      [0, 0],
```

For starters lets assign all to zero.

Defining some function which helps in game traversal.

```
win_loss_states = [0,6]

def getAllPossibleNextAction(cur_pos):
    step_matrix = [x != None for x in environment_matrix[cur_pos]]
    action = []
    if(step_matrix[0]):
        action.append(0)
    if(step_matrix[1]):
        action.append(1)
    return(action)

def isGoalStateReached(cur_pos):
    return (cur_pos in [6])

def getNextState(cur_pos, action):
    if (action == 0):
        return cur_pos - 1
    else:
        return cur_pos + 1

def isGameOver(cur_pos):
    return cur_pos in win_loss_states
```

Let's go though them one by one

- `getAllPossibleNextAction` pass your current state and it will return all the possible actions. Note for tile 0, only right action is there and same goes for tile 9 with only left action
- `isGoalStateReached` if the current tile is 6 it will return `True`
- `getNextState` pass current state and the action, and it will return the next state
- `isGameOver` if the state is 0 or 6, the game is over, this returns `True`

Now comes the traning part,

```
discount = 0.9
learning_rate = 0.1

for _ in range(1000):
    # get starting place
    cur_pos = random.choice([0,1,2,3,4,5,6,7,8,9])
    # while goal state is not reached
    while(not isGameOver(cur_pos)):
        # get all possible next states from cur_step
        possible_actions = getAllPossibleNextAction(cur_pos)
```

```
        q_matrix[cur_pos][action] = q_matrix[cur_pos][action] +
learning_rate * (environment_matrix[cur_pos][action] +
            discount * max(q_matrix[next_state]) - q_matrix[cur_pos]
[action])
        # go to next state
        cur_pos = next_state
    # print status
    print("Episode ", _ , " done")

print(q_matrix)
print("Training done...")
```

Let me clarify,

- First we defined the discount factor and learning rate
- We are going to train for 1000 episodes
- Spawning is completely random, it could be any of the tiles
- While the episode is not over, we keep on taking random actions and updating the Q table

After 1000 episodes, Q table some what looks like this,

```
99.99999999997391], [0, 0], [99.9999999999985, 80.99999999994624],
[89.99999999999515, 72.89999999997386], [80.99999999999046, 0]]
```

Let's beautify this a lil bit, here we go,

	Action	
State	Left	Right
0	0	0
1	-100	65.61
2	59.049	72.9
3	65.61	81
4	72.9	90
5	81	100
6	0	0
7	100	81
8	90	72.9
9	81	0

Now what does our policy says, if you find yourself at any of the state, choose the action with higher value (here darker shade of green) and you will reach the tile with beer. Not bad, right!

Next Step

Note: This note is from my personal website.

Machine Learning | Reinforcement Learning | Artificial Intelligence | Python | Q Learning

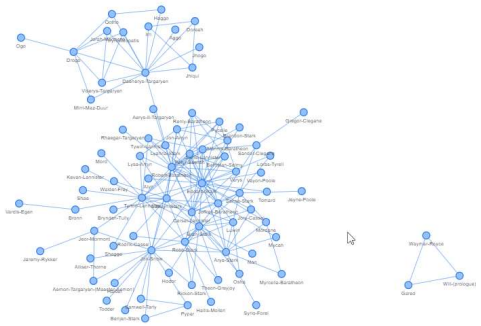
Written by Mohit Mayank

967 Followers · Writer for ITNEXT

Follow

Senior Data Scientist | AI/ML Researcher | Creator of “Jaal” | Author of “Lazy Data Science

More from Mohit Mayank and ITNEXT



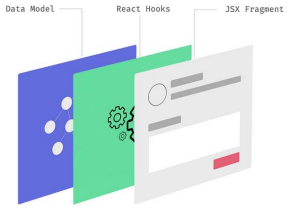
Mohit Mayank in Towards Data Science

Visualizing Networks in Python

A practical guide to tools which helps you “see” the network.

6 min read · Jan 26, 2021

1.3K 11



Juntao Qiu in ITNEXT

Decoupling UI and Logic in React: A Clean Code Approach with...

In the realm of front-end development, terms and paradigms can sometimes be mystifyin...

8 min read · Jul 6

777 10

Carlos Arguelles in ITNEXT

Amazon's Not So Secret Weapon

The magic of Working Backwards: a real-world case study

10 min read · Aug 7, 2022

2.3K 27

Mohit Mayank in Towards Data Science

Guide to fine-tuning Text Generation models: GPT-2, GPT-...

Going through the basics of massive language models, we learn about the differe...

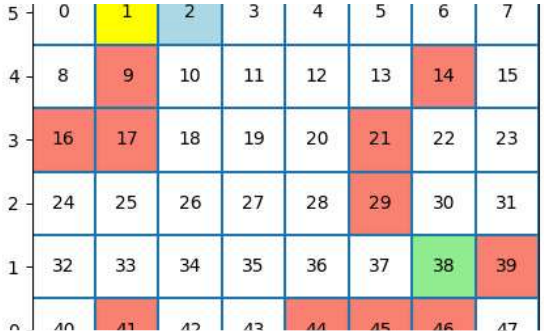
12 min read · Jul 11, 2021

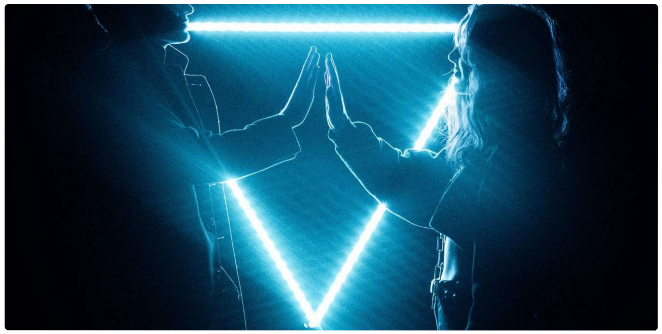
237 13

See all from Mohit Mayank

See all from ITNEXT

Recommended from Medium





Eligijus Bujokas in Towards Data Science

The Values of Actions in Reinforcement Learning using Q-...

The Q-learning algorithm implemented from scratch in Python

✦ · 10 min read · Feb 14

18

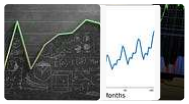
Wouter van Heeswijk, PhD in Towards Data Science

Proximal Policy Optimization (PPO) Explained

The journey from REINFORCE to the go-to algorithm in continuous control

✦ · 13 min read · Nov 29, 2022

188 2



Predictive Modeling w/ Python

18 stories228 saves ·

Practical Guides to Machine Learning

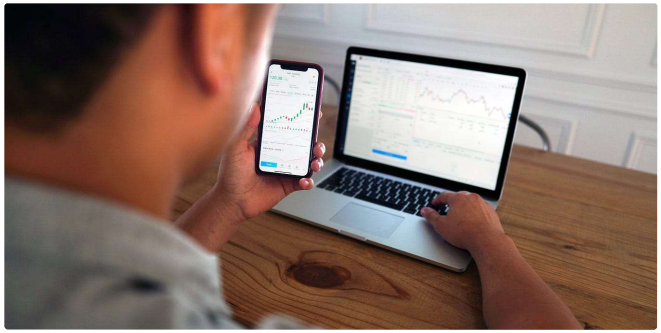
10 stories239 saves ·

Natural Language Processing

480 stories103 saves ·

ChatGPT

21 stories93 saves ·



Adrian J. Mayer in Python in Plain English

Using Python for Financial Portfolio Optimization

Includes a Complete Code Walkthrough

✦ · 11 min read · Feb 24

41 2

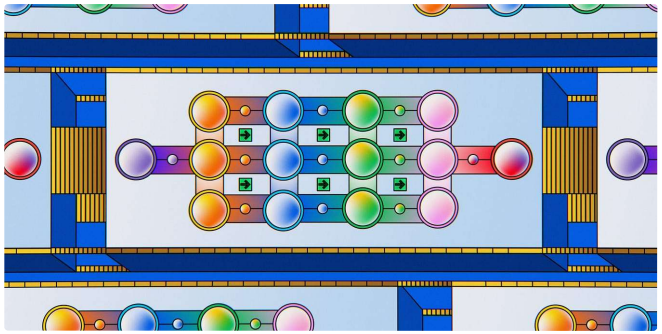
Hennie de Harder in Towards Data Science

Techniques to Improve the Performance of a DQN Agent

Reinforcement learning challenges and how to solve them.

✦ · 11 min read · Nov 30, 2022

152 1



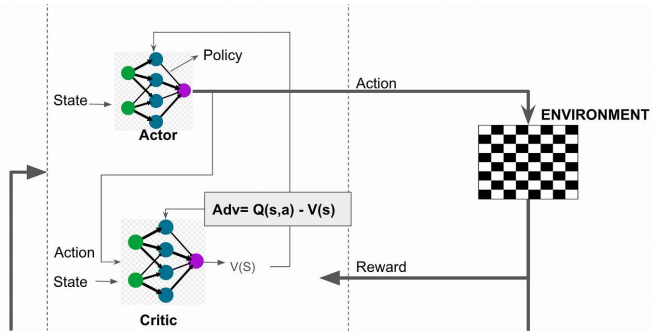
Javier Martínez Ojeda in Towards Data Science

Applied Reinforcement Learning IV: Implementation of DQN

Implementation of the DQN algorithm, and application to OpenAI Gym's CartPole-v1...

8 min read · Jan 10

107 3



Renu Khandelwal

Unlocking the Secrets of Actor-Critic Reinforcement Learning: A...

Understanding Actor-Critic Mechanisms, Different Flavors of Actor-Critic Algorithms,...

6 min read · Feb 20

56 1

See more recommendations