Q-Learning in Python

Last Updated: 19-04-2020

Pre-Requisite : Reinforcement Learning

Reinforcement Learning briefly is a paradigm of Learning Process in which a learning agent learns, overtime, to behave optimally in a certain environment by interacting continuously in the environment. The agent during its course of learning experience various different situations in the environment it is in. These are called *states*. The agent while being in that state may choose from a

1. **Q-Values or Action-Values:** Q-values are defined for states and actions. $Q(S, A)$ is an estimation of how good is it to take the action A at the state S. This estimation of $Q(S, A)$ will be iteratively computed using the **TD- Update rule** which we will see in the upcoming sections.
2. **Rewards and Episodes:** An agent over the course of its lifetime starts from a start state, makes a number of transitions from its current state to a next state based on its choice of action and also the environment the agent is interacting in. At every step of transition, the agent from a state takes an action, observes a reward from the environment, and then transits to another state. If at any point of time the agent ends up in one of the terminating states that means there are no further transition possible. This is said to be the completion of an episode.
3. **Temporal Difference or TD-Update:**
   The Temporal Difference or TD-Update rule can be represented as follows :

   $$Q(S,A) \leftarrow Q(S,A) + \alpha\left(R + \gamma Q(S',A') - Q(S,A)\right)$$

   This update rule to estimate the value of Q is applied at every time step of the agents interaction with the environment. The terms used are explained below. :

   - S : Current State of the agent.
   - A : Current Action Picked according to some policy.

- γ (>0 and <=1) : Discounting Factor for Future Rewards. Future rewars are less valuable than current rewards so they must be discounted. Since Q-value is an estimation of expected rewards from a state, discounting rule applies here as well.
- α : Step length taken to update the estimation of Q(S, A).

4. **Choosing the Action to take using ϵ-greedy policy:**
   ϵ-greedy policy of is a very simple policy of choosing actions using the current Q-value estimations. It goes as follows :
   - With probability $(1-\epsilon)$ choose the action which has the highest Q-value.
   - With probability (ϵ) choose any action at random.

Now with all the theory required in hand let us take an example. We will use OpenAI's gym environment to train our Q-Learning model.

Command to Install `gym` –

```
pip install gym
```

Before starting with example, you will need some helper code in order to visualize the working of the algorithms. There will be two helper files which need to be downloaded in the working directory. One can find the files here.

Step # 1 : Import required libraries.

```
import gym
import itertools
import matplotlib
import matplotlib.style
import numpy as np
import pandas as pd
import sys


from collections import defaultdict
from windy_gridworld import WindyGridworldEnv
import plotting

matplotlib.style.use('ggplot')
```

```
env = WindyGridworldEnv()
```

Step #3 : Make the ϵ-greedy policy.

```
def createEpsilonGreedyPolicy(Q, epsilon, num_actions):
    """
    Creates an epsilon-greedy policy based
    on a given Q-function and epsilon.

    Returns a function that takes the state
    as an input and returns the probabilities
    for each action in the form of a numpy array
    of length of the action space(set of possible actions).
    """
    def policyFunction(state):

        Action_probabilities = np.ones(num_actions,
                dtype = float) * epsilon / num_actions

        best_action = np.argmax(Q[state])
        Action_probabilities[best_action] += (1.0 - epsilon)
        return Action_probabilities

    return policyFunction
```

```
def qLearning(env, num_episodes, discount_factor = 1.0,
                            alpha = 0.6, epsilon = 0.1):
    """
    Q-Learning algorithm: Off-policy TD control.
    Finds the optimal greedy policy while improving
    following an epsilon-greedy policy"""

    # Action value function
    # A nested dictionary that maps
    # state -> (action -> action-value).
    Q = defaultdict(lambda: np.zeros(env.action_space.n))

    # Keeps track of useful statistics
    stats = plotting.EpisodeStats(
        episode_lengths = np.zeros(num_episodes),
        episode_rewards = np.zeros(num_episodes))

    # Create an epsilon greedy policy function
    # appropriately for environment action space
    policy = createEpsilonGreedyPolicy(Q, epsilon, env.action_space.n)

    # For every episode
    for ith_episode in range(num_episodes):

        # Reset the environment and pick the first action
        state = env.reset()

        for t in itertools.count():

            # get probabilities of all actions from current state
            action_probabilities = policy(state)

            # choose action according to
            # the probability distribution
            action = np.random.choice(np.arange(
                      len(action_probabilities)),
                       p = action_probabilities)

            # take action and get reward, transit to next state
            next_state, reward, done, _ = env.step(action)

            # Update statistics
            stats.episode_rewards[ith_episode] += reward
            stats.episode_lengths[ith_episode] = t

            # TD Update
            best_next_action = np.argmax(Q[next_state])
```

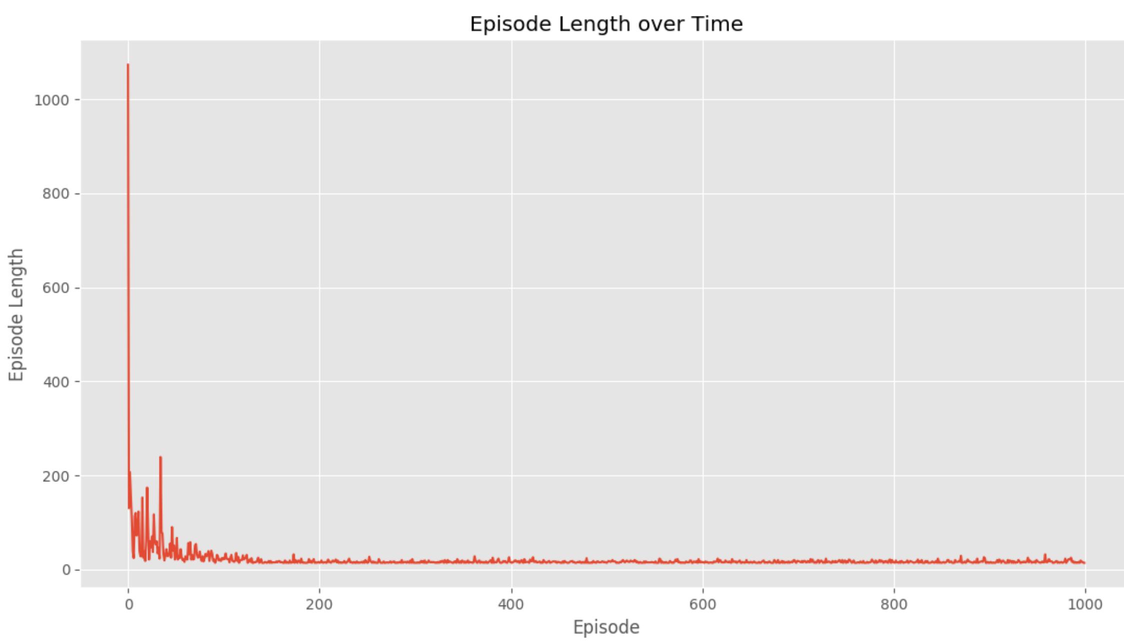
```
            state = next_state

    return Q, stats
```

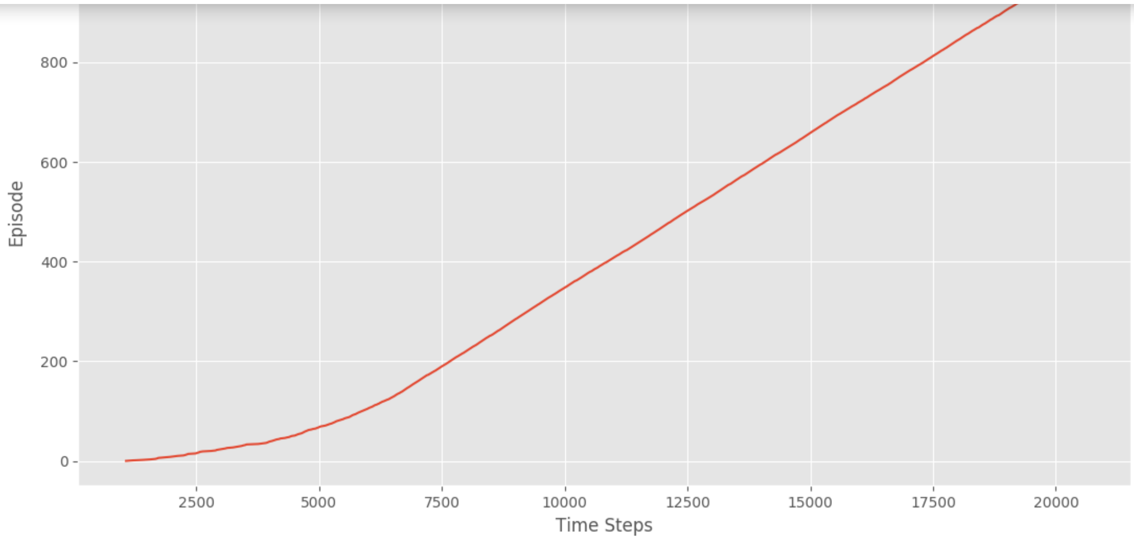
Step #5 : Train the model.

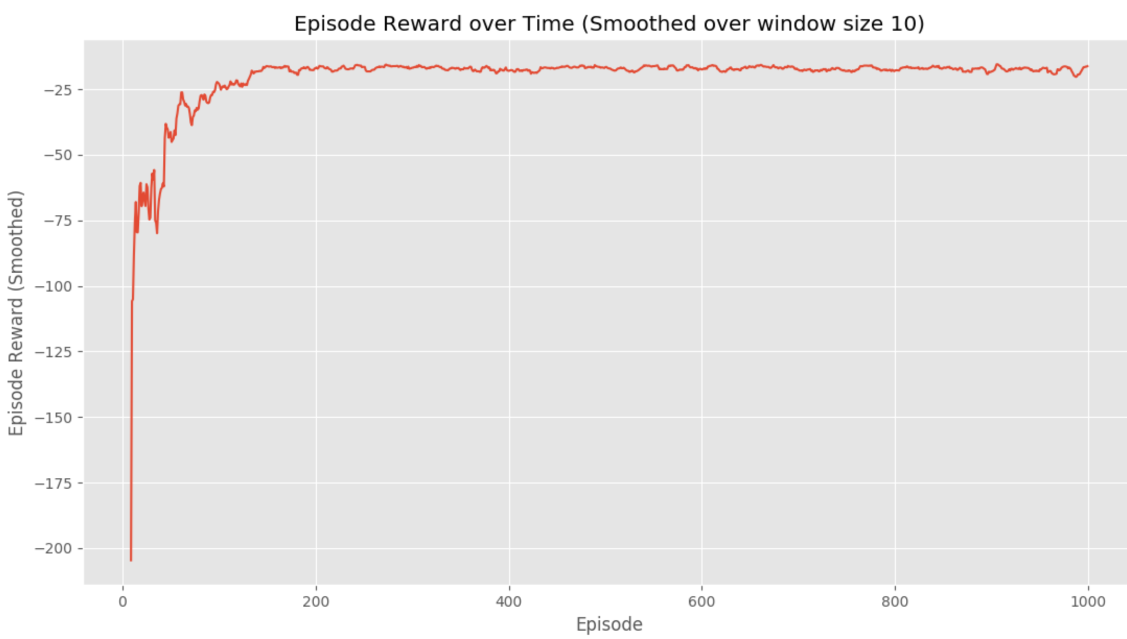
```
Q, stats = qLearning(env, 1000)
```

Step #6 : Plot important statistics.

```
plotting.plot_episode_stats(stats)
```







Conclusion:

We see that in the Episode Reward over time plot that the episode rewards *progressively increase* over time and ultimately *levels out* at a high reward per episode value which indicates that the agent has learnt to maximize its total reward earned in an episode by behaving optimally at every state.

Attention geek! Strengthen your foundations with the **Python Programming Foundation** Course

Recommended Posts:

Important differences between Python 2.x and Python 3.x with examples
Python | Set 4 (Dictionary, Keywords in Python)
Python | Sort Python Dictionaries by Key or Value
Python | Merge Python key values to list
Reading Python File-Like Objects from C | Python
Python | Add Logging to a Python Script
Python | Add Logging to Python Libraries
JavaScript vs Python : Can Python Overtop JavaScript by 2020?
Python | Visualizing O(n) using Python
Python | Index of Non-Zero elements in Python list
Python | Convert list to Python array
MySQL-Connector-Python module in Python
Python - Read blob object in python using wand library
Python | PRAW - Python Reddit API Wrapper
twitter-text-python (ttp) module - Python
Reusable piece of python functionality for wrapping arbitrary blocks of code : Python Context

Kaustav kumar Chanda

Check out this Author's contributed articles.

If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute.geeksforgeeks.org or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please Improve this article if you find anything incorrect by clicking on the "Improve Article" button below.

Improved By : VishvajeetRamanuj

Article Tags : Advanced Computer Subject Machine Learning Python

Practice Tags : Machine Learning

4

To-do Done

Based on **1** vote(s)

Improve Article

Please write to us at contribute@geeksforgeeks.org to report any issue with the above content.

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Load Comments

GeeksforGeeks

5th Floor, A-118,
Sector-136, Noida, Uttar Pradesh - 201305

feedback@geeksforgeeks.org

Company

About Us

Learn

Algorithms

Practice

Courses

Company-wise

Topic-wise

How to begin?

Contribute

Write an Article

Write Interview Experience

Internships

Videos