Homework 1

December 4, 2023

1 Reinforcement Learning: The Q-learning algorithm

Anna Vandi, Gabriele Cianni

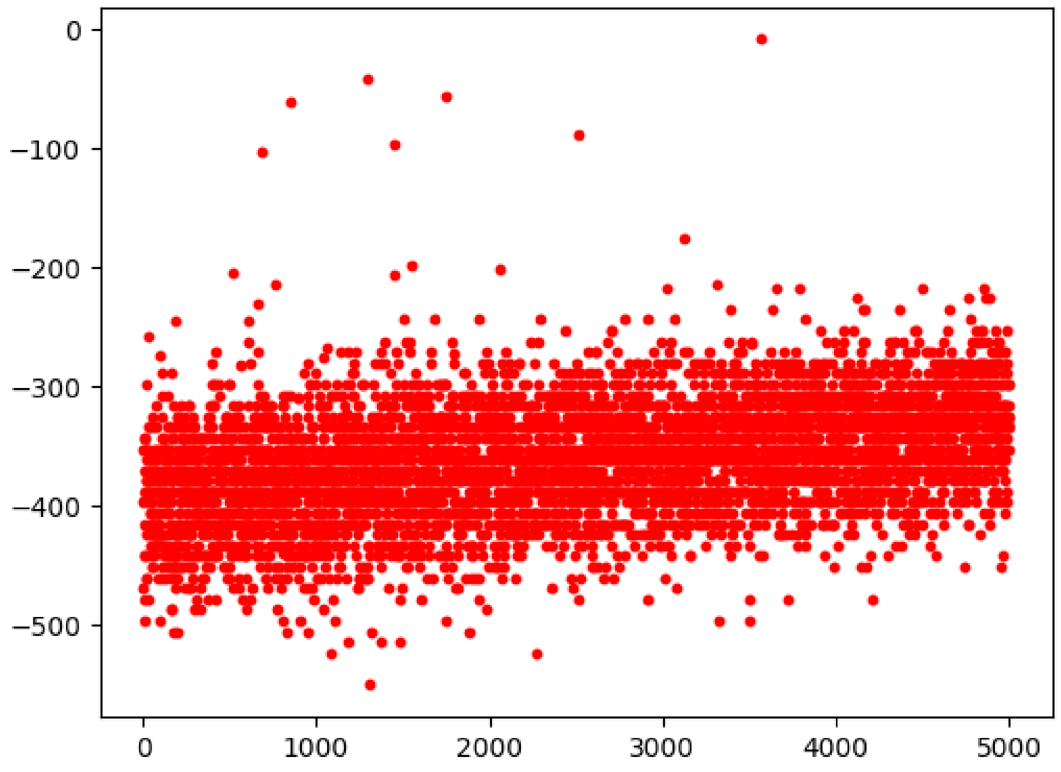
1.0.1 Exercise 2

[1]:
```
!pip install cmake 'gym[atari]' scipy
```

```
Requirement already satisfied: cmake in
/home/anna/anaconda3/lib/python3.10/site-packages (3.27.7)
Requirement already satisfied: gym[atari] in
/home/anna/anaconda3/lib/python3.10/site-packages (0.26.2)
Requirement already satisfied: scipy in /home/anna/.local/lib/python3.10/site-
packages (1.9.1)
Requirement already satisfied: numpy>=1.18.0 in
/home/anna/.local/lib/python3.10/site-packages (from gym[atari]) (1.23.3)
Requirement already satisfied: cloudpickle>=1.2.0 in
/home/anna/anaconda3/lib/python3.10/site-packages (from gym[atari]) (2.0.0)
Requirement already satisfied: gym-notices>=0.0.4 in
/home/anna/anaconda3/lib/python3.10/site-packages (from gym[atari]) (0.0.8)
Requirement already satisfied: ale-py~=0.8.0 in
/home/anna/anaconda3/lib/python3.10/site-packages (from gym[atari]) (0.8.1)
Requirement already satisfied: importlib-resources in
/home/anna/.local/lib/python3.10/site-packages (from ale-py~=0.8.0->gym[atari])
(5.10.1)
Requirement already satisfied: typing-extensions in
/home/anna/.local/lib/python3.10/site-packages (from ale-py~=0.8.0->gym[atari])
(4.4.0)

[notice] A new release of pip is
available: 23.2.1 -> 23.3.1
[notice] To update, run:
pip install --upgrade pip
```

[8]:
```
import numpy as np
import gym.spaces
import pandas as pd
from matplotlib import pyplot as plt
```

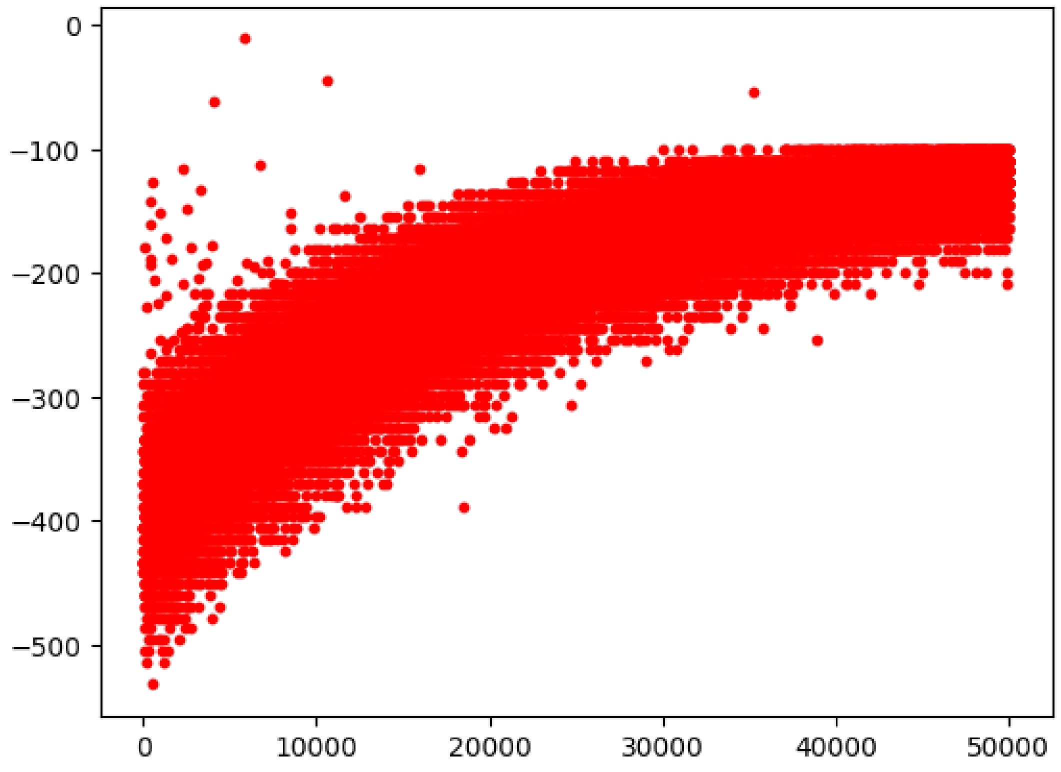
```
env = gym.make('Taxi-v3')
state_space = env.observation_space.n
action_space = env.action_space.n
qtable = np.zeros((state_space, action_space))# starting with a qtable of zeros
# Hyperparameters
epsilon = 1.0  # Greed 100%
epsilon_min = 0.005  # Greed 0.05%
epsilon_decay = 0.99995  # Decay multiplied by epsilon after each episode
episodes = 5000  # Number of episodes (amount of games)
max_steps = 100  # Max steps per episode
learning_rate = 0.65  # Learning rate
gamma = 0.65  # Discount rate
rews = []

# Smart agent
for episode in range(episodes):
    # reset the game state, done and score before every episode/game
    state, _ = env.reset()
    # print(state)
    done = False
    score = 0

    for _ in range(max_steps):
        # with the probability of (1-epsilon) take the best action  in our␣
 ↪Q-table
        if np.random.uniform(0, 1) > epsilon:
            action = np.argmax(qtable[state, :])
        else:
            # else take a random action
            action = env.action_space.sample()

        # step the game forward
        # print(env.step(action))
        state, reward, done, _, _ = env.step(action)

        #update penalties
        if reward == -10:
            penalties += 1

        # add up the score
        score += reward

        # update Q-table with the q-function
        qtable[state, action] = (1 - learning_rate) * qtable[state, action] +␣
 ↪learning_rate * (
                    reward + gamma * np.max(qtable[state, :]))
        if done:
            break

    rews.append(score)

    # reducing epsilon each episode (Exploration vs Exploitation trade-off  )
    if epsilon >= epsilon_min:
        epsilon *= epsilon_decay

#plt.plot(range(episodes), rews, "o")
plt.scatter(range(episodes), rews, 10, color="red")
plt.show()

print("Penalties: {}".format(penalties))
```



```
Penalties: 307907
```

We notice that, running the algorithm for inly 5000 episodes, the agent still behave like the random agent, This is why we need to train him for more episods (50000)

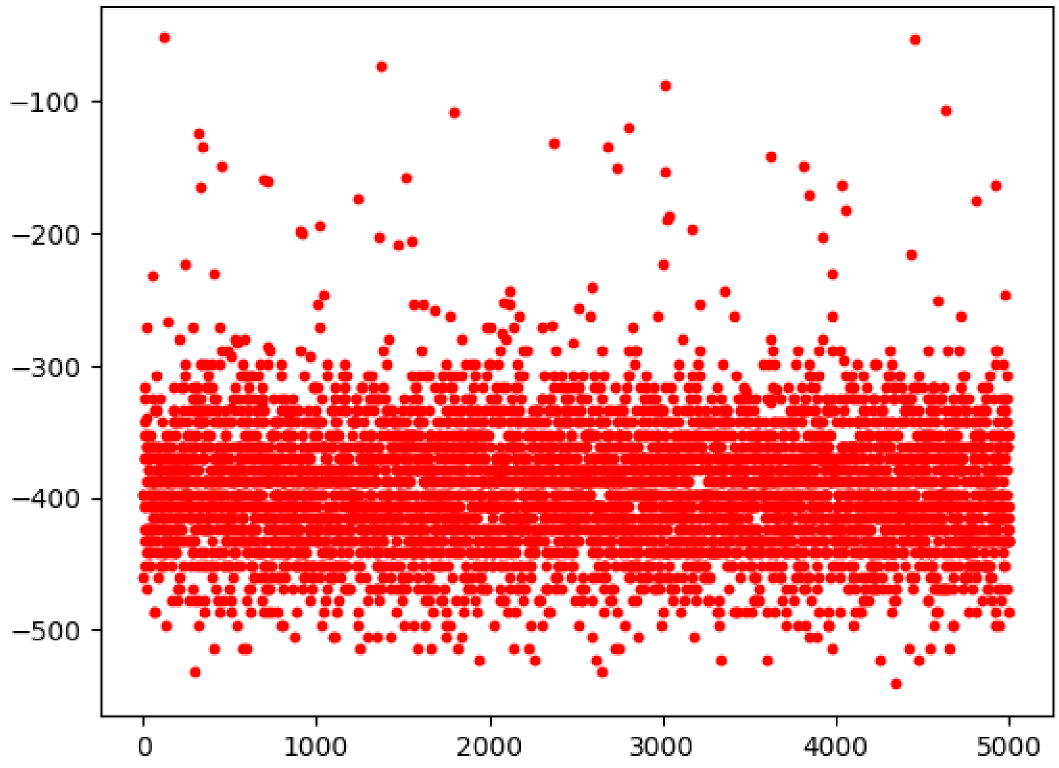
[9]:
```
import numpy as np
import gym.spaces
import pandas as pd
from matplotlib import pyplot as plt

env = gym.make('Taxi-v3')
state_space = env.observation_space.n
action_space = env.action_space.n
qtable = np.zeros((state_space, action_space))# starting with a qtable of zeros
# Hyperparameters
epsilon = 1.0  # Greed 100%
epsilon_min = 0.005  # Greed 0.05%
epsilon_decay = 0.99995  # Decay multiplied by epsilon after each episode
episodes = 50000  # Number of episodes (amount of games)
max_steps = 100  # Max steps per episode
learning_rate = 0.65  # Learning rate
gamma = 0.65  # Discount rate
penalties=0
rews = []

# Smart agent
for episode in range(episodes):
    # reset the game state, done and score before every episode/game
    state, _ = env.reset()
    # print(state)
    done = False
    score = 0

    for _ in range(max_steps):
        # with the probability of (1-epsilon) take the best action  in our␣
↪Q-table
        if np.random.uniform(0, 1) > epsilon:
            action = np.argmax(qtable[state, :])
        else:
            # else take a random action
            action = env.action_space.sample()

        # step the game forward
        # print(env.step(action))
        state, reward, done, _, _ = env.step(action)

        #update penalties
        if reward == -10:
            penalties += 1

        # add up the score
        score += reward
        # update Q-table with the q-function
        qtable[state, action] = (1 - learning_rate) * qtable[state, action] +␣
 ↪learning_rate * (
                        reward + gamma * np.max(qtable[state, :]))

        if done:
            break

    rews.append(score)

    # reducing epsilon each episode (Exploration vs Exploitation trade-off  )
    if epsilon >= epsilon_min:
        epsilon *= epsilon_decay

#plt.plot(range(episodes), rews, "o")
plt.scatter(range(episodes), rews, 10, color="red")
plt.show()

print("Penalties: {}".format(penalties))
```



```
Penalties: 607021
```

Now, random agent is implemented: only 5000 episodes in order to compare it with 5000 episodes run by smart agent.

[10]:
```python
import numpy as np
import gym.spaces
import pandas as pd
from matplotlib import pyplot as plt

env = gym.make('Taxi-v3')
state_space = env.observation_space.n
action_space = env.action_space.n
qtable = np.zeros((state_space, action_space))# starting with a qtable of zeros
# Hyperparameters
epsilon = 1.0  # Greed 100%
epsilon_min = 0.005  # Greed 0.05%
epsilon_decay = 0.99995  # Decay multiplied by epsilon after each episode
episodes = 5000  # Number of episodes (amount of games)
max_steps = 100  # Max steps per episode
learning_rate = 0.65  # Learning rate
gamma = 0.65  # Discount rate
penalties=0
rews = []

# Random agent
for episode in range(episodes):
    # reset the game state, done and score before every episode/game
    state, _ = env.reset()
    # print(state)
    done = False
    score = 0

    for _ in range(max_steps):
        action = env.action_space.sample()
        state, reward, done, _, _ = env.step(action)

        #update penalties
        if reward == -10:
            penalties += 1

        # add up the score
        score += reward

        if done:
            break

    rews.append(score)
    # reducing epsilon each episode (Exploration vs Exploitation trade-off  )
    if epsilon >= epsilon_min:
        epsilon *= epsilon_decay

plt.scatter(range(episodes), rews, 10, color="red")
plt.show()

print("Penalties: {}".format(penalties))
```

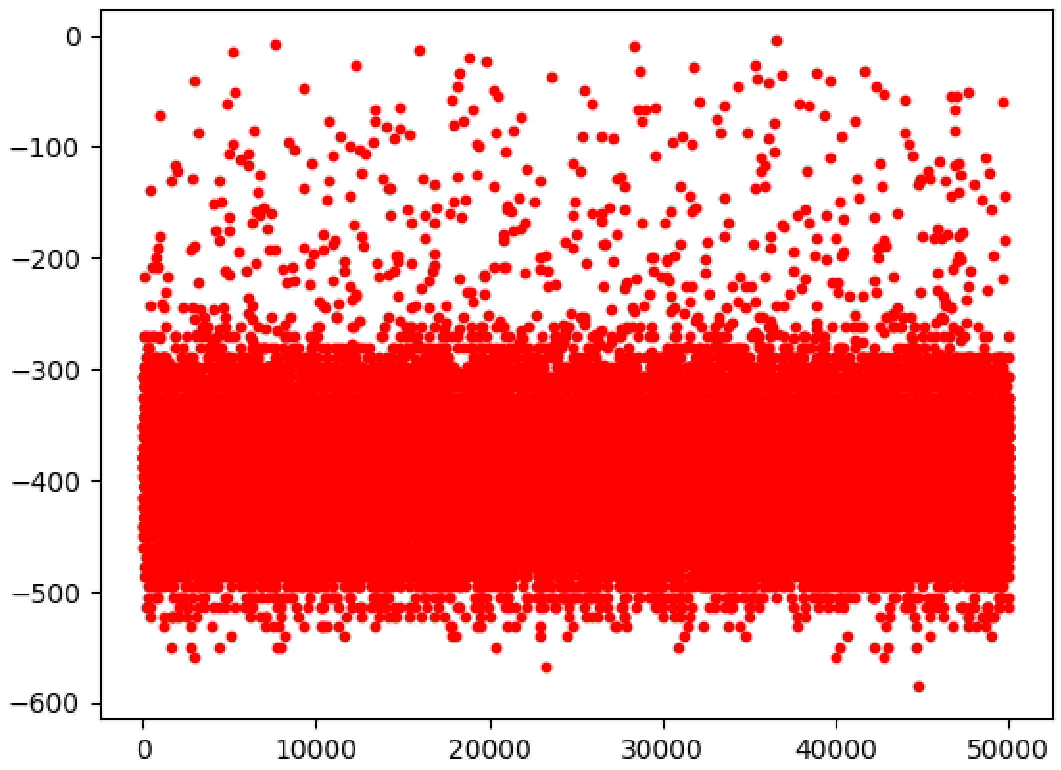


```
Penalties: 162725
```

We notice that the smart agent and the random agent have the same behaviour if run for only 5000 episodes (still random behaviour). Now we run the random agent for 50000 episodes:

[11]:
```
import numpy as np
import gym.spaces
import pandas as pd
from matplotlib import pyplot as plt

env = gym.make('Taxi-v3')
state_space = env.observation_space.n
action_space = env.action_space.n
qtable = np.zeros((state_space, action_space))# starting with a qtable of zeros
```

```python
# Hyperparameters
epsilon = 1.0  # Greed 100%
epsilon_min = 0.005  # Greed 0.05%
epsilon_decay = 0.99995  # Decay multiplied by epsilon after each episode
episodes = 50000  # Number of episodes (amount of games)
max_steps = 100  # Max steps per episode
learning_rate = 0.65  # Learning rate
gamma = 0.65  # Discount rate
penalties=0
rews = []

# Random agent
for episode in range(episodes):
    # reset the game state, done and score before every episode/game
    state, _ = env.reset()
    # print(state)
    done = False
    score = 0

    for _ in range(max_steps):
        action = env.action_space.sample()
        state, reward, done, _, _ = env.step(action)

        #update penalties
        if reward == -10:
            penalties += 1

        # add up the score
        score += reward

        if done:
            break

    rews.append(score)

    # reducing epsilon each episode (Exploration vs Exploitation trade-off  )
    if epsilon >= epsilon_min:
        epsilon *= epsilon_decay

plt.scatter(range(episodes), rews, 10, color="red")
plt.show()

print("Penalties: {}".format(penalties))
```



```
Penalties: 1623750
```

Comparing smart and random agent we notice that: if we run them only for 5000 episodes, they behave almost the same (smart agent behave still randomly): smart agent penalties are 307907 and random agent penalties are 162725. If we run them only for 50000, smart agent performs much better: smart agent penalties are 607021 and random agent penalties are 1623750.