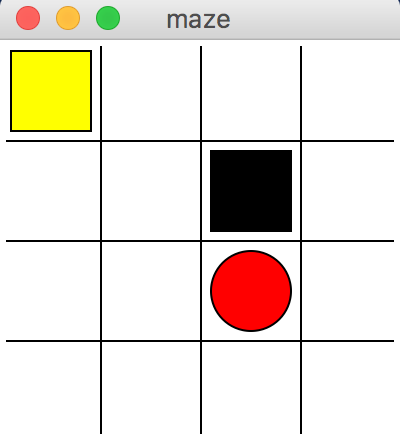
# Q

import Reinforcement\_learning

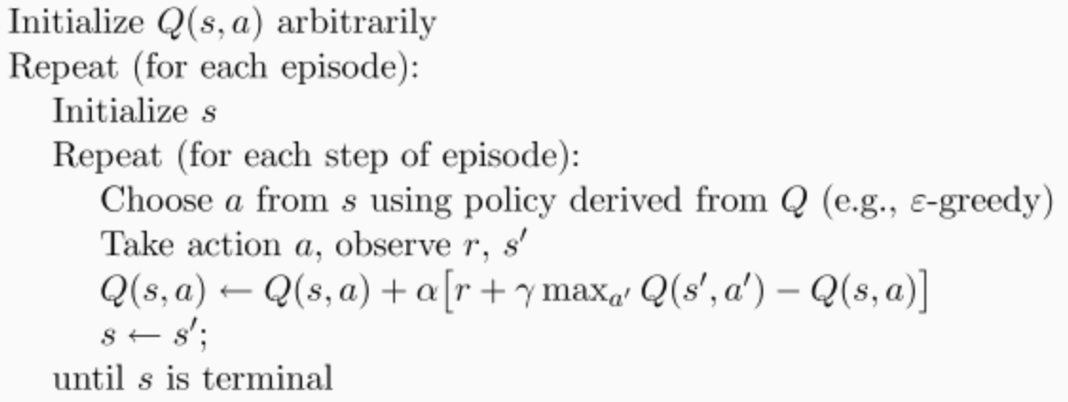
|  |  |  |
| --- | --- | --- |
| Algorithm | Average number of explorations(all time/  Post half time) | Frequency of exploration |
| Q-Learning | 5.04/ |  |
| Sarsa | 12.55/ |  |
| DQN | 21.25/6.6 |  |
| Doubel DQN | 36.38/ 8.82 |  |
| PERDQN | 50.17/27 |  |
| DuelingDQN | 64.82/29 |  |

All result base on Maze



## Q-Learing

### Base



### Object Initialization

from Reinforcement\_learning import Q\_Learning

Object Initialization

RL = Q\_Learning(actions=list(range(n\_actions)))

Input: The size of Actions set

### choose\_action()

action = RL.choose\_action(str(observation))

Input: state observation

Ouput: action

### learn()

RL.learn(str(observation), action, reward, str(observation\_))

Input: state observation, action, reward, next state observation

Ouput: null

### Example

import sys

import os

from Reinforcement\_learning import Q\_Learning

from Reinforcement\_learning import Maze

def update():

for episode in range(100):

# initial observation

observation = env.reset()

while True:

# fresh env

env.render()

# RL choose action based on observation

action = RL.choose\_action(str(observation))

# RL take action and get next observation and reward

observation\_, reward, done = env.step(action)

# RL learn from this transition

RL.learn(str(observation), action, reward, str(observation\_))

# swap observation

observation = observation\_

# break while loop when end of this episode

if done:

break

# end of game

print('game over')

env.destroy()

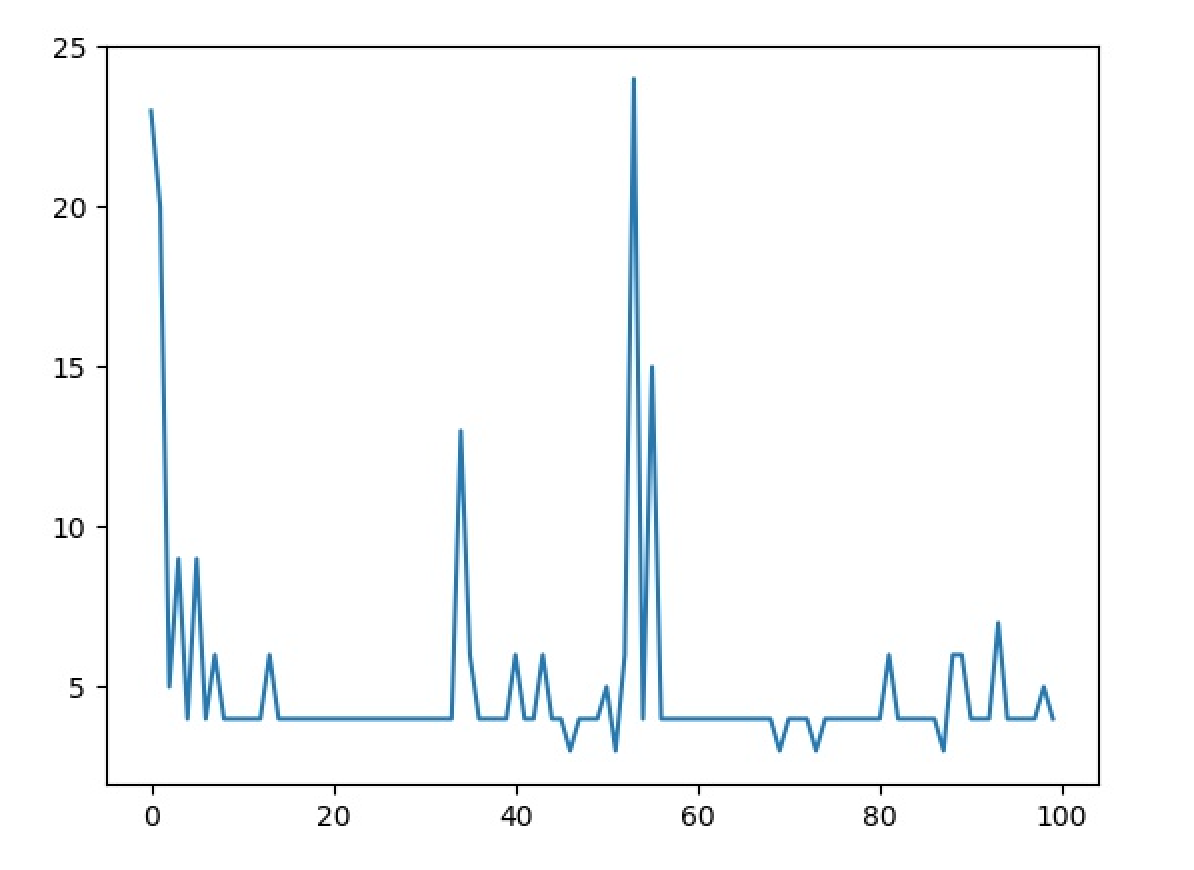
if \_\_name\_\_ == "\_\_main\_\_":

env = Maze()

RL = Q\_Learning(actions=list(range(env.n\_actions)))

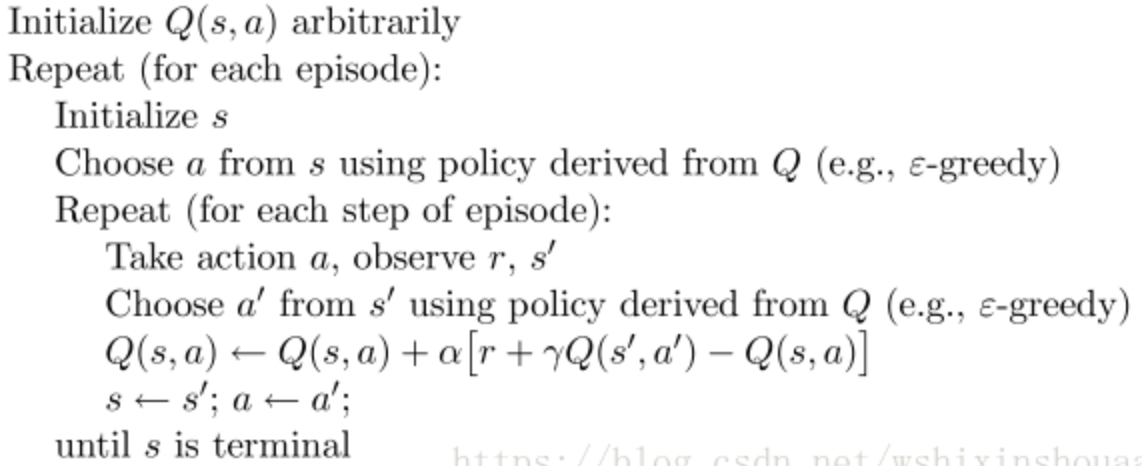
env.after(100, update)

env.mainloop()



## SARSA

### Base



### Object Initialization

RL = SARSA(actions=list(range(env.n\_actions)))

### choose\_action()

action = RL.choose\_action(str(observation))

Input: state observation

Ouput: action

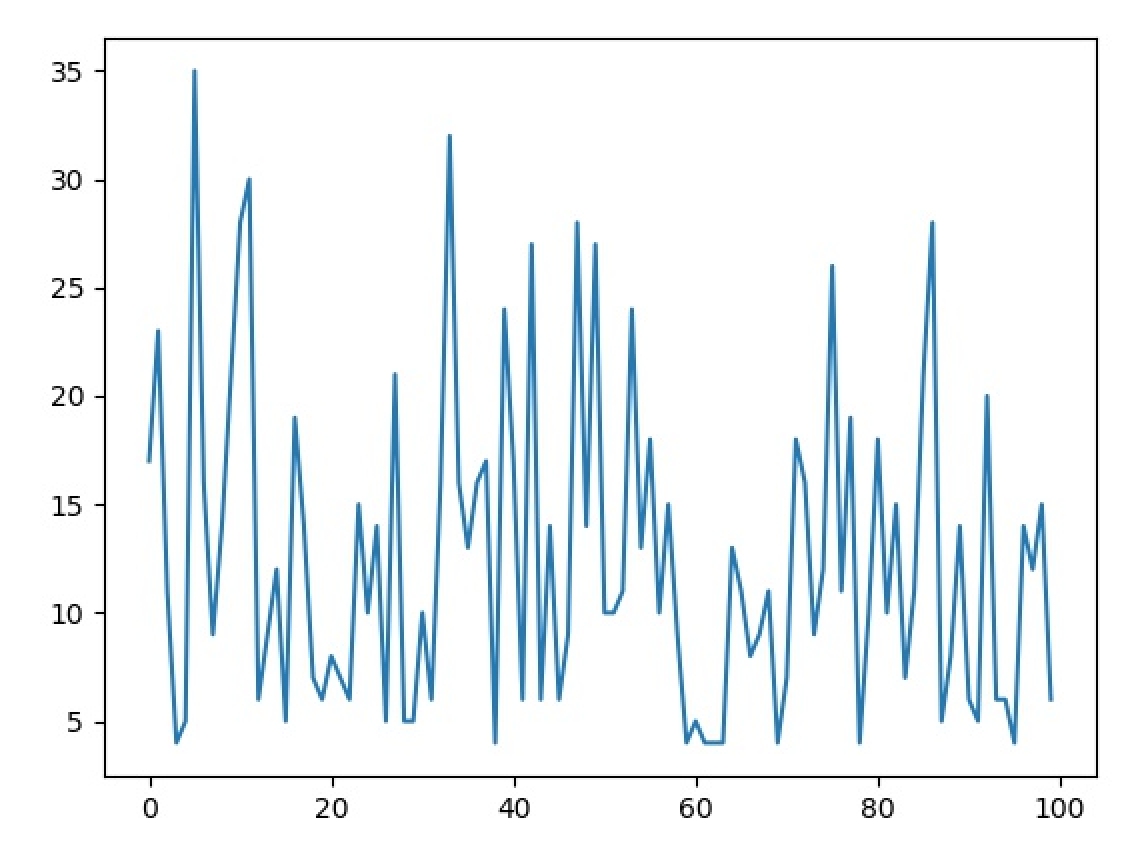
### Learn

RL.learn(str(observation), action, reward, str(observation\_))

Input: state observation, action, reward, next state observation

Ouput: null

### Example



## DQN

### Base

### Object Initialization

RL = DQN(env.n\_actions, env.n\_features,

learning\_rate=0.01,

reward\_decay=0.9,

e\_greedy=0.9,

replace\_target\_iter=200,

memory\_size=2000,

# output\_graph=True

)

### choose\_action()

action = RL.choose\_action(observation)

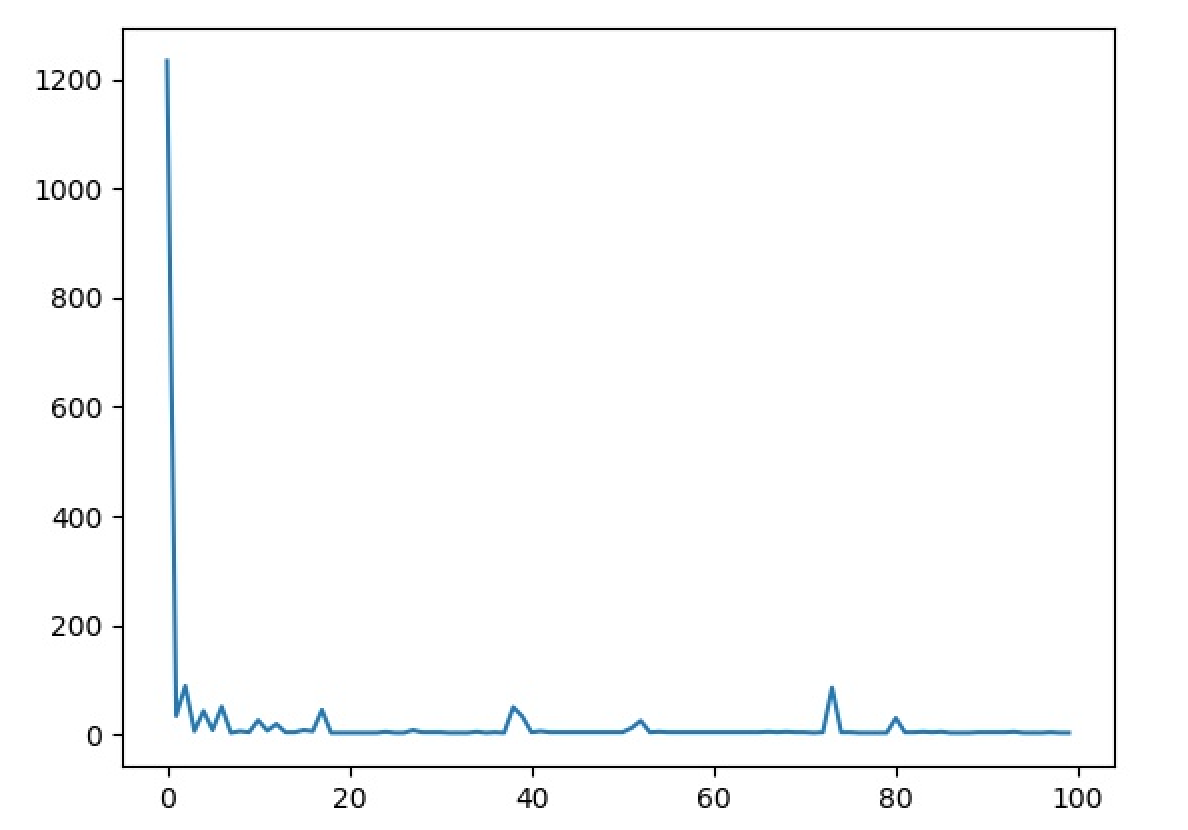
### store\_transition

RL.store\_transition(observation, action, reward, observation\_)

### Learn

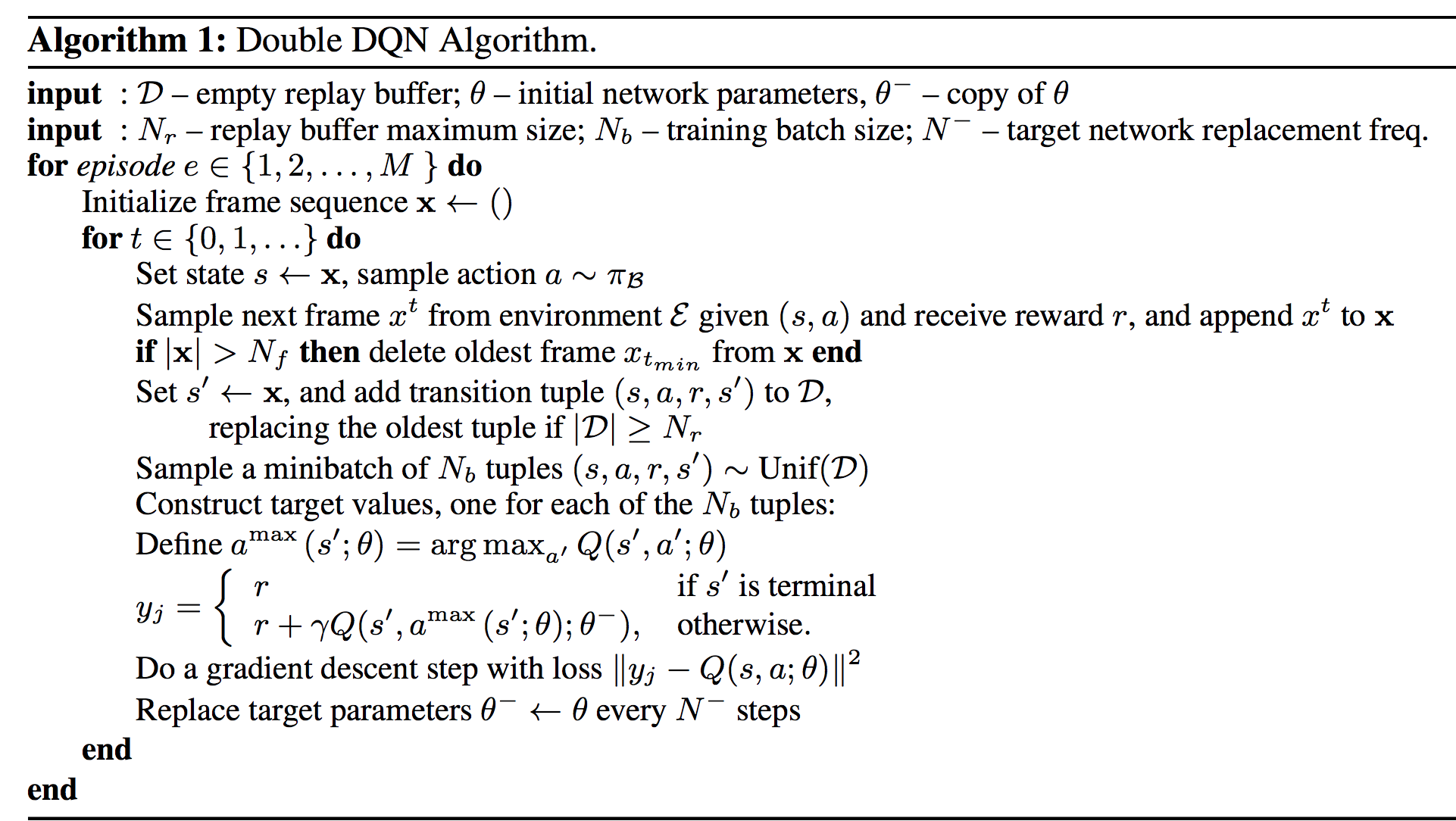
RL.learn()

### Example



## DDQN

### Base



### Object Initialization

RL = DQN(env.n\_actions, env.n\_features,

learning\_rate=0.01,

reward\_decay=0.9,

e\_greedy=0.9,

replace\_target\_iter=200,

memory\_size=2000,

# output\_graph=True

)

### choose\_action()

action = RL.choose\_action(observation)

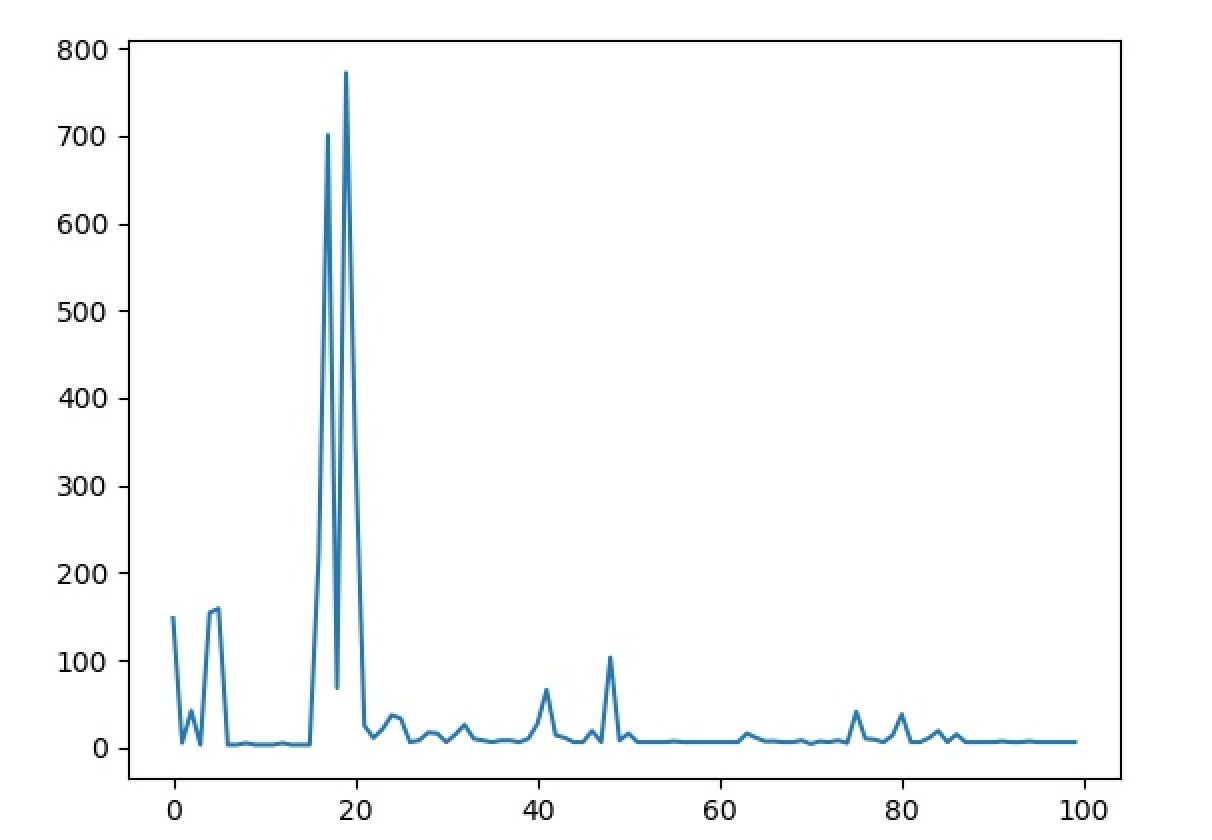
### store\_transition

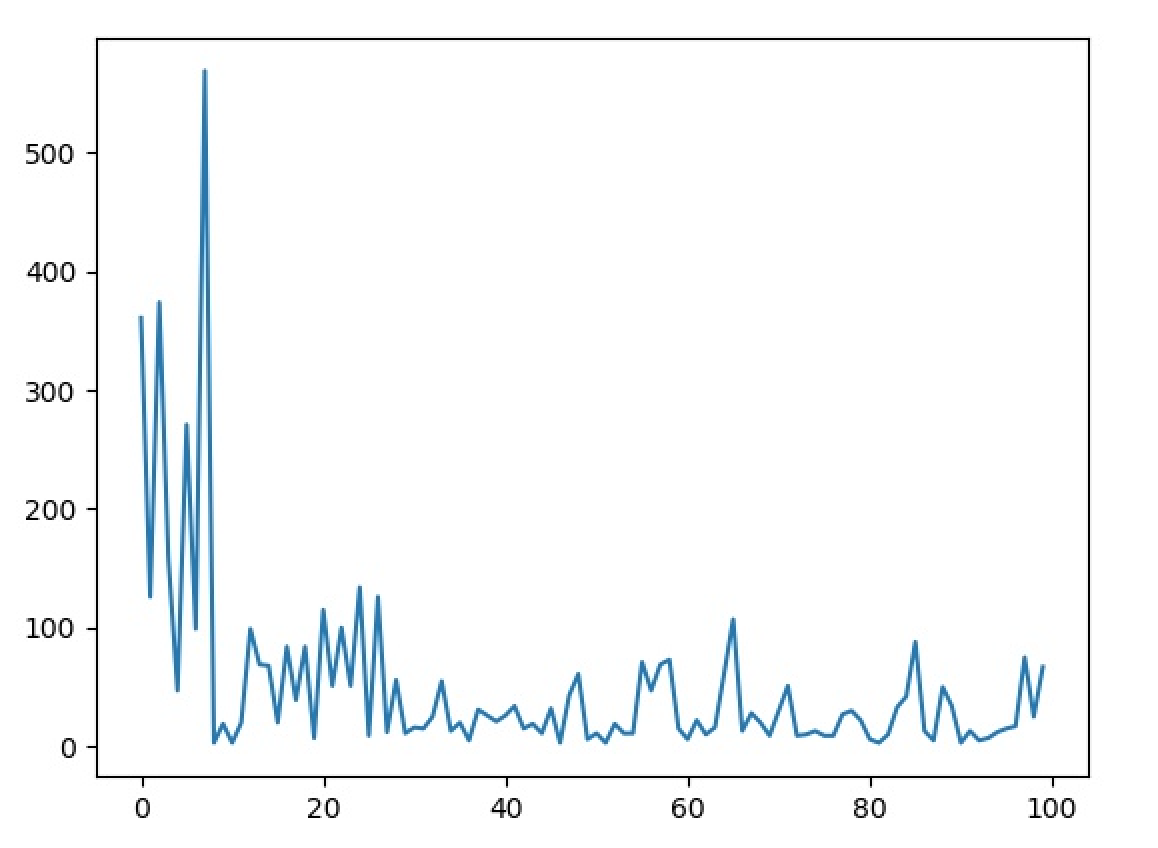
RL.store\_transition(observation, action, reward, observation\_)

### Learn

RL.learn()

### Example



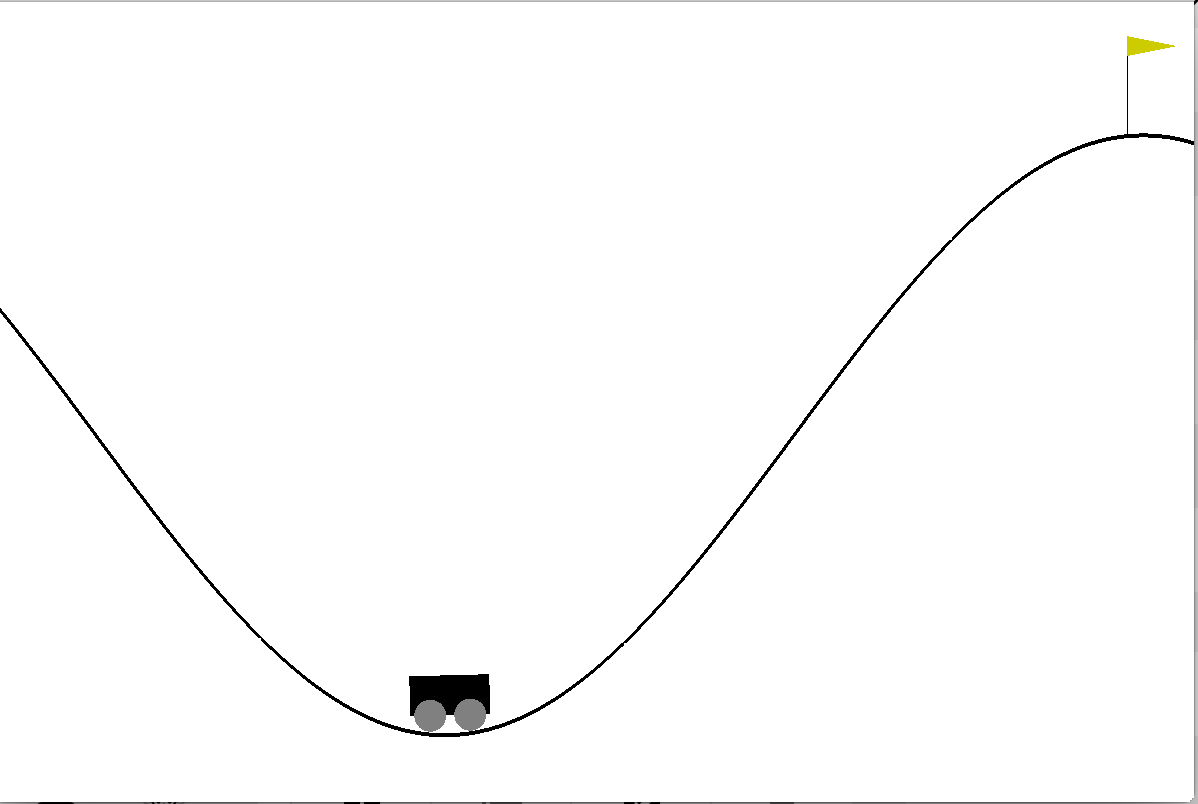


## Dueling DQN

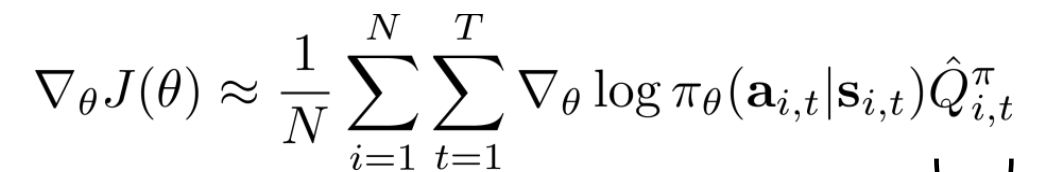
## PERDQN.py

# P

import Reinforcement\_learning



## PolicyGradient



## ActorCritic