

Playing with Reinforcement Learning in Python

The Q-Learning Algorithm

Geraint Palmer

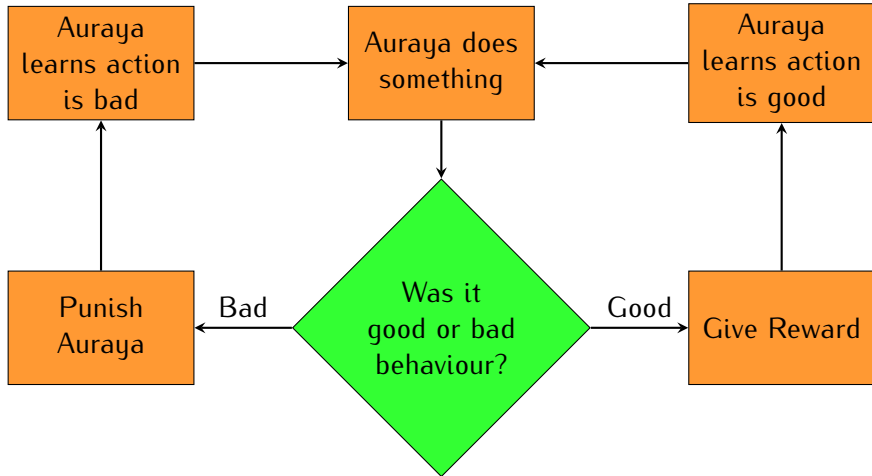
Python Namibia, 2015

<http://python-namibia.org>

Auraya



How to train a dog?



The Q-learning Algorithm

Set all Q and V values to 0

repeat

 Observe the current state s_t

 Select and perform an action a_t

 Observe the reward $r(s_t, a_t)$

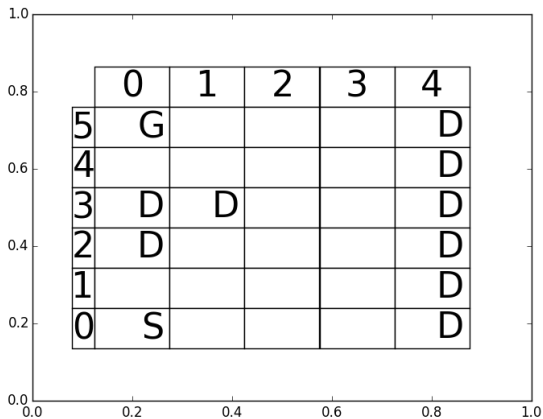
 Perform the following updates:

$$Q_{t+1} \leftarrow (1 - \alpha)Q_t(s_t, a_t) + \alpha[r(s_t, a_t) + \gamma V_t(s_{t+1})]$$

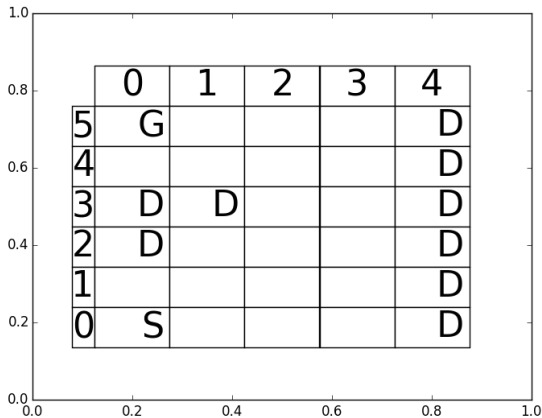
$$V_{t+1}(s) \leftarrow \max_a Q_t(s, a)$$

until *convergence*

Rory The Robot



Rory The Robot



only 85%
successful

Code Structure

```
class Board():
```

grid_height

grid_width

number_of_episodes

robot

squares

```
class Squares():
```

coords

identifier

reward

```
class Robot():
```

playing_board

actions

action_selection_parameter

learning_rate

discount_rate

moves

episode

coords

Vs

Qs

movement_dictionary

Code Structure

```
class Board():
```

grid_height

grid_width

number_of_episodes

robot

squares

```
class Squares():
```

coords

identifier

reward

```
class Robot():
```

playing_board

actions

action_selection_parameter

learning_rate

discount_rate

moves

episode

coords

Vs

Qs

movement_dictionary

Action Selection – Exploration vs Exploitation

ϵ -Soft Policy

```
def Q_Learning(self, action, reward, sqr, new_sqr):
    """
    Selects which action to take using the epsilon-soft action selection policy
    """
    rnd_num = random.random()
    if rnd_num < 1 - self.action_selection_parameter:
        return str(max(self.Qs[sqr], key=lambda x: self.Qs[sqr][x]))
    return random.choice(self.actions)
```

Movement

```
def find_destination(self, sqr, action):  
    """  
    Chooses the new coordinates after taking an action, according to the faultiness  
    """  
    rnd_num = random.random()  
    sum_p, indx = 0, 0  
    while rnd_num > sum_p:  
        direction = self.actions[indx]  
        sum_p += self.transitions[action][indx]  
        indx += 1  
    return self.movement_dict[direction](sqr)
```

Learning

```
def Q_Learning(self, action, reward, sqr, new_sqr):  
    """  
    Updates Rory's Q and V values  
    """  
    self.Qs[sqr][action] = (  
        1-self.learning_rate)*self.Qs[sqr][action] + self.learning_rate*(  
            reward + self.discount_rate*self.Vs[new_sqr]  
        )  
    self.Vs[sqr] = max(self.Qs[sqr].values())
```

Simulation

```
def simulate(self):
    """
    Simulates many episodes of the game while the robots learns the best policies
    """
    plt.ion()
    self.show_board()
    wait = raw_input('Press enter to continue.')
    print 'Simulating .....'
    while self.robot.episode < self.number_of_episodes:
        action = self.robot.select_action(self.robot.coords)
        new_coords = self.robot.find_destination(self.robot.coords, action)
        self.robot.moves += 1

        reward = self.squares[new_coords[1]][new_coords[0]].reward + (
            self.squares[new_coords[1]][new_coords[0]].move_cost * self.robot.moves)
        self.robot.Q_Learning(action, reward, self.robot.coords, new_coords)

        self.robot.coords = new_coords

        if (self.squares[new_coords[1]][new_coords[0]].identifier == 'Death' or
            self.squares[new_coords[1]][new_coords[0]].identifier == 'Goal'):
            self.robot.moves = 0
            self.robot.coords = tuple(self.starting_coords)
            self.robot.episode += 1
    self.update_results()
    wait = raw_input('Simulated. Press enter to exit.')
```

Simulation

```
def simulate(self):
    """
    Simulates many episodes of the game while the robots learns the best policies
    """
    plt.ion()
    self.show_board()
    wait = raw_input('Press enter to continue.')
    print 'Simulating ....'
    while self.robot.episode < self.number_of_episodes:
        action = self.robot.select_action(self.robot.coords)
        new_coords = self.robot.find_destination(self.robot.coords, action)
        self.robot.moves += 1

        reward = self.squares[new_coords[1]][new_coords[0]].reward + (
            self.squares[new_coords[1]][new_coords[0]].move_cost * self.robot.moves)
        self.robot.Q_Learning(action, reward, self.robot.coords, new_coords)

        self.robot.coords = new_coords

        if (self.squares[new_coords[1]][new_coords[0]].identifier == 'Death' or
            self.squares[new_coords[1]][new_coords[0]].identifier == 'Goal'):
            self.robot.moves = 0
            self.robot.coords = tuple(self.starting_coords)
            self.robot.episode += 1
    self.update_results()
    wait = raw_input('Simulated. Press enter to exit.')
```

Simulation

```
def simulate(self):
    """
    Simulates many episodes of the game while the robots learns the best policies
    """
    plt.ion()
    self.show_board()
    wait = raw_input('Press enter to continue.')
    print 'Simulating .....'
    while self.robot.episode < self.number_of_episodes:
        action = self.robot.select_action(self.robot.coords)
        new_coords = self.robot.find_destination(self.robot.coords, action)
        self.robot.moves += 1

        reward = self.squares[new_coords[1]][new_coords[0]].reward + (
            self.squares[new_coords[1]][new_coords[0]].move_cost * self.robot.moves)
        self.robot.Q_Learning(action, reward, self.robot.coords, new_coords)

        self.robot.coords = new_coords

        if (self.squares[new_coords[1]][new_coords[0]].identifier == 'Death' or
            self.squares[new_coords[1]][new_coords[0]].identifier == 'Goal'):
            self.robot.moves = 0
            self.robot.coords = tuple(self.starting_coords)
            self.robot.episode += 1
    self.update_results()
    wait = raw_input('Simulated. Press enter to exit.')
```

Simulation

```
def simulate(self):
    """
    Simulates many episodes of the game while the robots learns the best policies
    """
    plt.ion()
    self.show_board()
    wait = raw_input('Press enter to continue.')
    print 'Simulating .....'
    while self.robot.episode < self.number_of_episodes:
        action = self.robot.select_action(self.robot.coords)
        new_coords = self.robot.find_destination(self.robot.coords, action)
        self.robot.moves += 1

        reward = self.squares[new_coords[1]][new_coords[0]].reward + (
            self.squares[new_coords[1]][new_coords[0]].move_cost * self.robot.moves)
        self.robot.Q_Learning(action, reward, self.robot.coords, new_coords)

        self.robot.coords = new_coords

        if (self.squares[new_coords[1]][new_coords[0]].identifier == 'Death' or
            self.squares[new_coords[1]][new_coords[0]].identifier == 'Goal'):
            self.robot.moves = 0
            self.robot.coords = tuple(self.starting_coords)
            self.robot.episode += 1
    self.update_results()
    wait = raw_input('Simulated. Press enter to exit.')
```

Simulation

```
def simulate(self):
    """
    Simulates many episodes of the game while the robots learns the best policies
    """
    plt.ion()
    self.show_board()
    wait = raw_input('Press enter to continue.')
    print 'Simulating .....'
    while self.robot.episode < self.number_of_episodes:
        action = self.robot.select_action(self.robot.coords)
        new_coords = self.robot.find_destination(self.robot.coords, action)
        self.robot.moves += 1

        reward = self.squares[new_coords[1]][new_coords[0]].reward + (
            self.squares[new_coords[1]][new_coords[0]].move_cost * self.robot.moves)
        self.robot.Q_Learning(action, reward, self.robot.coords, new_coords)

        self.robot.coords = new_coords

        if (self.squares[new_coords[1]][new_coords[0]].identifier == 'Death' or
            self.squares[new_coords[1]][new_coords[0]].identifier == 'Goal'):
            self.robot.moves = 0
            self.robot.coords = tuple(self.starting_coords)
            self.robot.episode += 1
    self.update_results()
    wait = raw_input('Simulated. Press enter to exit.')
```

Demo

Learning Rate, α

Small $\alpha \implies$ historical rewards more important

Large $\alpha \implies$ the latest reward more important

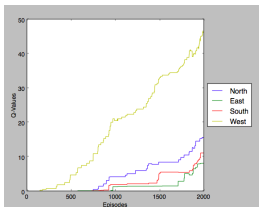


Figure: $\alpha = 0.03$

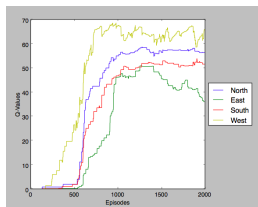


Figure: $\alpha = 0.1$

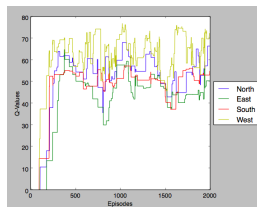


Figure: $\alpha = 0.5$

Discount Rate, γ

Small $\gamma \implies$ look for immediate rewards

Large $\gamma \implies$ strive for long term rewards

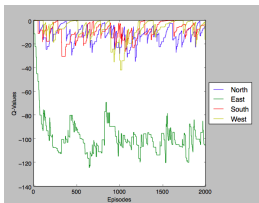


Figure: $\gamma = 0.1$

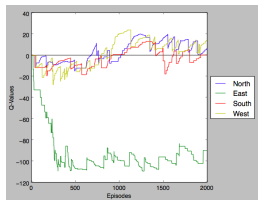


Figure: $\gamma = 0.9$

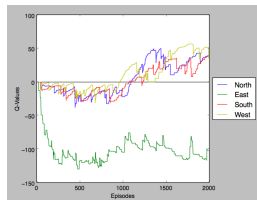
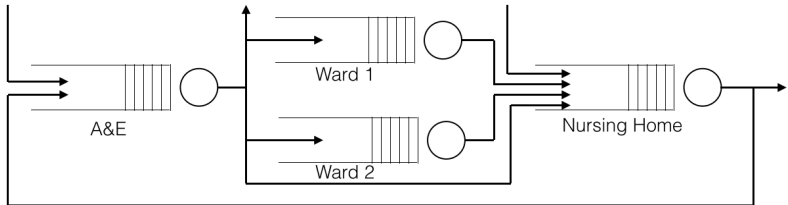


Figure: $\gamma = 0.99$

Using RL in a Healthcare System



Links

A robot learning to walk

A computer learning to play 'snake'

A virtual car learning not to crash

An agent learning the shortest route through a maze

Github repository for the code

This talk and code also here