

# Data Analysis and Machine Learning: Reinforcement Learning

Morten Hjorth-Jensen<sup>1,2</sup>

<sup>1</sup>Department of Physics, University of Oslo

<sup>2</sup>Department of Physics and Astronomy and National Superconducting Cyclotron Laboratory, Michigan State University

Dec 26, 2018

## Reinforcement Learning: Overarching view

Reinforcement Learning (RL) is one of the most exciting fields of Machine Learning today, and also one of the oldest. It has been around since the 1950s, producing many interesting applications over the years.

## Code example

```
"""
A simple example for Reinforcement Learning using table lookup Q-learning method.
An agent "o" is on the left of a 1 dimensional world, the treasure is on the rightmost location.
Run this program and to see how the agent will improve its strategy of finding the treasure.
View more on my tutorial page: https://morvanzhou.github.io/tutorials/
"""

import numpy as np
import pandas as pd
import time

np.random.seed(2) # reproducible

N_STATES = 6 # the length of the 1 dimensional world
ACTIONS = ['left', 'right'] # available actions
EPSILON = 0.9 # greedy police
ALPHA = 0.1 # learning rate
GAMMA = 0.9 # discount factor
MAX_EPISODES = 13 # maximum episodes
FRESH_TIME = 0.3 # fresh time for one move

def build_q_table(n_states, actions):
    table = pd.DataFrame(
        np.zeros((n_states, len(actions))), # q_table initial values
        columns=actions, # actions's name
    )
```

```

    # print(table)      # show table
    return table

def choose_action(state, q_table):
    # This is how to choose an action
    state_actions = q_table.iloc[state, :]
    if (np.random.uniform() > EPSILON) or ((state_actions == 0).all()): # act non-greedy or state
        action_name = np.random.choice(ACTIONS)
    else: # act greedy
        action_name = state_actions.idxmax() # replace argmax to idxmax as argmax means a diff
    return action_name

def get_env_feedback(S, A):
    # This is how agent will interact with the environment
    if A == 'right': # move right
        if S == N_STATES - 2: # terminate
            S_ = 'terminal'
            R = 1
        else:
            S_ = S + 1
            R = 0
    else: # move left
        R = 0
        if S == 0:
            S_ = S # reach the wall
        else:
            S_ = S - 1
    return S_, R

def update_env(S, episode, step_counter):
    # This is how environment be updated
    env_list = ['-']*(N_STATES-1) + ['T'] # '-----T' our environment
    if S == 'terminal':
        interaction = 'Episode %s: total_steps = %s' % (episode+1, step_counter)
        print('\n\r{}'.format(interaction), end='')
        time.sleep(2)
        print('\n\r', end='')
    else:
        env_list[S] = 'o'
        interaction = ''.join(env_list)
        print('\n\r{}'.format(interaction), end='')
        time.sleep(FRESH_TIME)

def rl():
    # main part of RL loop
    q_table = build_q_table(N_STATES, ACTIONS)
    for episode in range(MAX_EPISODES):
        step_counter = 0
        S = 0
        is_terminated = False
        update_env(S, episode, step_counter)
        while not is_terminated:

            A = choose_action(S, q_table)
            S_, R = get_env_feedback(S, A) # take action & get next state and reward
            q_predict = q_table.loc[S, A]
            if S_ != 'terminal':

```

```

        q_target = R + GAMMA * q_table.iloc[S_, :].max()    # next state is not terminal
    else:
        q_target = R    # next state is terminal
        is_terminated = True    # terminate this episode

        q_table.loc[S, A] += ALPHA * (q_target - q_predict) # update
        S = S_    # move to next state

        update_env(S, episode, step_counter+1)
        step_counter += 1
    return q_table

if __name__ == "__main__":
    q_table = rl()
    print('\r\nQ-table:\n')
    print(q_table)

```