# Unsupervised Learning , Recommenders, Reinforcement learning-II

Saturday, April 29, 2023 1:06 PM

## Reinforcement Learning

position of helicopter ⟶ how to move control sticks

State $s$ ⟶ action $a$

reward function

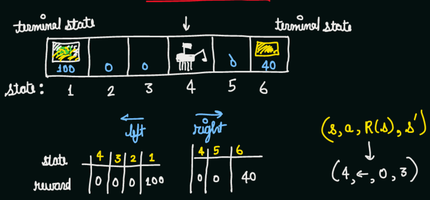
positive reward : helicopter flying well +1

negative reward : helicopter flying poorly −1000

## Applications

1. controlling robots
2. Factory optimization
3. Financial (stock) trading
4. Playing games (including video games)

## Mars Rover Example




$(s, a, R(s), s')$

$(4, \leftarrow, 0, 3)$

Return = Σ all rewards (with discount)

**Return** $= R_1 + \gamma R_2 + \gamma^2 R_3 + \ldots\ldots + (\text{until terminal state})$

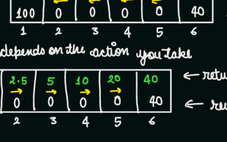
$\gamma \rightarrow$ discount factor

Let $\gamma = 0.9$

Return $= 0 + 0.9(0) + (0.9)^2(0) + (0.9)^3(100) = 72.9$

(state 4 to 1)

## Example of Return




Return depends on the action you take




## Policy ($\pi$)

state $s$ → policy $\pi$ → action $a$

→ mapping from states to action, telling what actions to take in a given state $s$.

The goal of RL is to find a policy $\pi$, $(a = \pi(s))$ that tells you what action to take in every step to maximize the return.

$\pi(2) = \leftarrow$
$\pi(3) = \leftarrow$
$\pi(4) = \leftarrow$
$\pi(5) = \rightarrow$
$\pi(6) = a$

| | Mars Rover | Helicopter | Chess |
|---|---|---|---|
| states | 6 states | position of hc. | pieces on board |
| action | ←, → | move control | set of legal moves |
| rewards | 100, 0, 40 | +1, −1000 | +1, 0, −1 |
| discount factor $\gamma$ | 0.5 | 0.99 | 0.995 |
| return | $R_2 + \gamma R_3 + \gamma^2 R_4 + \ldots$ | $R_1 + R_2(1) + R_3\gamma^2 + \ldots$ | $R_1 + R_2\gamma + R_3\gamma^2 + \ldots$ |
| policy $\pi$ | 100 ← ← → 40 | $\pi(s) = a$ | $\pi(s) = a$ |

## Markov Decision Process (MDP)

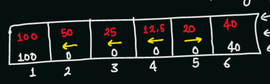

## State Action Value function (Q function)

$Q(s,a)$ = Return if you (Best Return)

→ start in state $s$

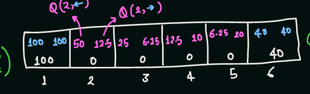
→ take action $a$ (once)

→ then behave optimally after that




$Q(2, \rightarrow) = (0) + (0.9)(0) + (0.9^2)(0) + (0.9^3)(100) = 12.5$

State : 2 3 2 1




$\max_a Q(s,a)$ → best possible return from state $s$

↖ Best possible action in state $s$ which give this maximum Q-value

$\max_a Q(s,a)$

$\pi(s) = a$

Q*/Optimal Q-function

## Bellman Equation

$Q(s,a)$ = Return if you

- start in state $s$
- take action $a$ (once)
- then behave optimally after that

$R(s)$ = reward of current state

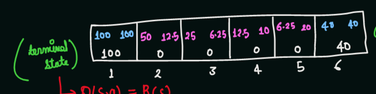
$s$ : current state

$a$ : current action

$s'$ : state you get to after taking action $a$

$a'$ : action you take in state $s'$

$$Q(s,a) = R(s) + \gamma \max_{a'} Q(s',a')$$




$Q(2, \rightarrow) \Rightarrow R(2) + 0.5 \max_{a'}(3, a')$ $\quad s=2, a=\rightarrow, s'=3$

$\Rightarrow 0 + 0.5(25) \Rightarrow 12.5$

$Q(4, \leftarrow) = R(4) + 0.5 \max_{a'}(3, a')$ $\quad s=4, a=\leftarrow, s'=3$

$= 0 + 0.5(25) = 12.5$

$$Q(s,a) = \underbrace{R(s)}_{\text{Reward you get right away (Immediate Reward)}} + \gamma \max_{a'} Q(s',a')$$

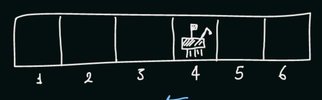
Return from behaving optimally starting from state $s'$

$$Q(s,a) = R_1 + \gamma R_2 + \gamma^2 R_3 + \gamma^3 R_4 + \ldots$$
$$= R_1 + \gamma\left[R_2 + \gamma R_3 + \gamma^2 R_4 + \ldots\right]$$

$R_1$ → immediate reward of state $s$.

↑ optimal return if you start from state $s'$

## Stochastic Environment



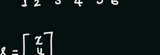

Expected Return = Average$\left(R_1 + \gamma R_2 + \gamma^2 R_3 + \gamma^3 R_4 + \ldots\right)$

$= E\left[R_1 + \gamma R_2 + \gamma^2 R_3 + \gamma^3 R_4 + \ldots\right]$

## Discrete V/s Continuous State

$s = \{1, 2, 3, 4, 5, 6\}$

Discrete State : 6-state Lunar lander




Continuous State :


$s = \begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \\ \theta \\ \dot{\theta} \\ l \\ r \end{bmatrix}$


$s = \begin{bmatrix} x \\ y \\ z \\ \dot{x} \\ \dot{y} \\ \dot{z} \\ \theta \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix}$ position (3D) → roll, pitch, yaw

continuous space MDP

## Lunar Lander

Actions :

→ do nothing

→ Left thruster

→ main thruster

→ right thruster

$s = \begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \\ \theta \\ \dot{\theta} \\ l \\ r \end{bmatrix}$ 0 or 1

Learn a policy $\pi$ for given state $s$, pick action, $a = \pi(s)$ so as to maximize the return $\gamma = 0.985$

## Reward function

→ Getting to landing pad : 100-140

→ Additional reward for moving toward/away from pad

→ Crash : −100

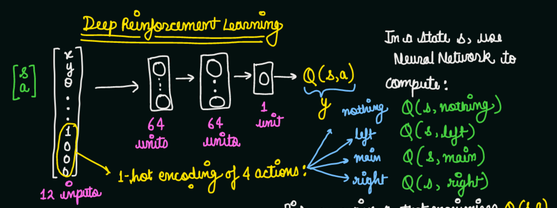
→ Soft landing : +100

→ Leg grounded : +10

→ Fire main engine : −0.3

→ Fire side thruster : −0.03

## Deep Reinforcement Learning




In a state $s$, use Neural Network, to compute :

$Q(s, \text{nothing})$

$Q(s, \text{left})$

$Q(s, \text{main})$

$Q(s, \text{right})$

Pick an action $a$, that maximizes $Q(s,a)$

## Bellman Equation

$$Q(s,a) = R(s) + \gamma \max_{a'} Q(s',a')$$

$x$ $\quad$ $y$

Random guess

$y^{(1)} = R(s^{(1)}) + \gamma \max_{a'} Q(s'^{(1)}, a')$

$f_{w,b}(x) \approx y$

$(s, a, R(s), s')$

$(s^{(1)}, a^{(1)}, R(s^{(1)}), s'^{(1)})$

$(s^{(2)}, a^{(2)}, R(s^{(2)}), s'^{(2)})$

$(s^{(3)}, a^{(3)}, R(s^{(3)}), s'^{(3)})$

$\left(s^{(m)}, a^{(m)}, R(s^{(m)}), s'^{(m)}\right)$

| $x$ | $y$ |
|---|---|
| $x^{(1)} = (s^{(1)}, a^{(1)})$ | $y^{(1)}$ |
| $x^{(2)} = (s^{(2)}, a^{(2)})$ | $y^{(2)}$ |
| ⋮ | ⋮ |
| $x^{(10,000)}$ | $y^{(10,000)}$ |

## Learning Algorithm (Deep Q-Network)

Initialize neural network randomly as guess $Q(s,a)$

Repeat {

Take actions in the Lunar Lander. Get $(s, a, R(s), s')$.

Store 10,000 most recent $(s, a, R(s), s')$ tuples. ← Replay Buffer

Train neural Network :

→ use minibatch of 1,000 examples (noisy, but speed up performance)

→ create training set of 10,000 examples using $(x, y)$

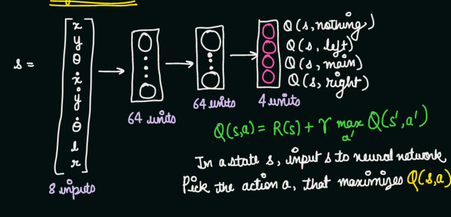
$x = (s, a)$ & $y = R(s) + \gamma \max_{a'} Q(s', a')$

→ Train $Q_{new}$ such that $Q_{new}(s,a) \approx y$

→ Set $Q = Q_{new}$ ← Soft update

}

## Refinements



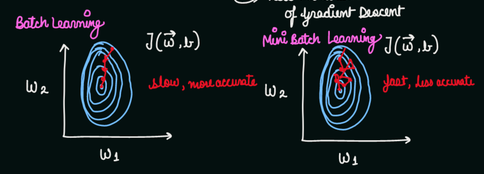

$Q(s,a) = R(s) + \gamma \max_{a'} Q(s', a')$

In a state $s$, input $s$ to neural network. Pick the action $a$, that maximizes $Q(s,a)$

## Mini Batch

$m = 100,000,000$ (very large, slow GD ☹)

$m' = 5,000$ { subset } → new minibatch on each iteration of gradient descent




## Limitations of Reinforcement Learning

- Much easier to get to work in a simulation than a real robot!
- Far fewer applications than Supervised & unsupervised learning.
- But... exciting research direction with potential for future applications.

## ε-greedy policy

How to choose actions while still learning?

In some state $s$

Option 1

Pick the action that maximizes $Q(s,a)$

Option 2 ($\varepsilon = 0.05$)

With probability 0.95, pick the action that maximizes $Q(s,a)$ "Exploitation" greedy

With probability 0.05, pick an action randomly ← $\varepsilon$ "Exploration" with $\varepsilon$

Start $\varepsilon$ high # More exploration

1.0 ⟶ 0.01

gradually decrease # More greedy

## Soft update

Let $Q = Q_{new}$

$w, b$ ← $w_{new}, b_{new}$ ← Converge more reliably

$w = 0.01 w_{new} + 0.99 w$

$b = 0.01 b_{new} + 0.99 b$