INTRODUCTION TO DEEP Q-LEARNING

- ABHINAND B

Plan for the Session

AGENDA

- WHAT IS REINFORCEMENT LEARNING?
- MARKOV DECISION PROCESSES
- BASICS OF Q-LEARNING
- HOW DEEP Q LEARNING WORKS?
- IMPLEMENTING A DQN AGENT
- EXAMINE THE RESULTS

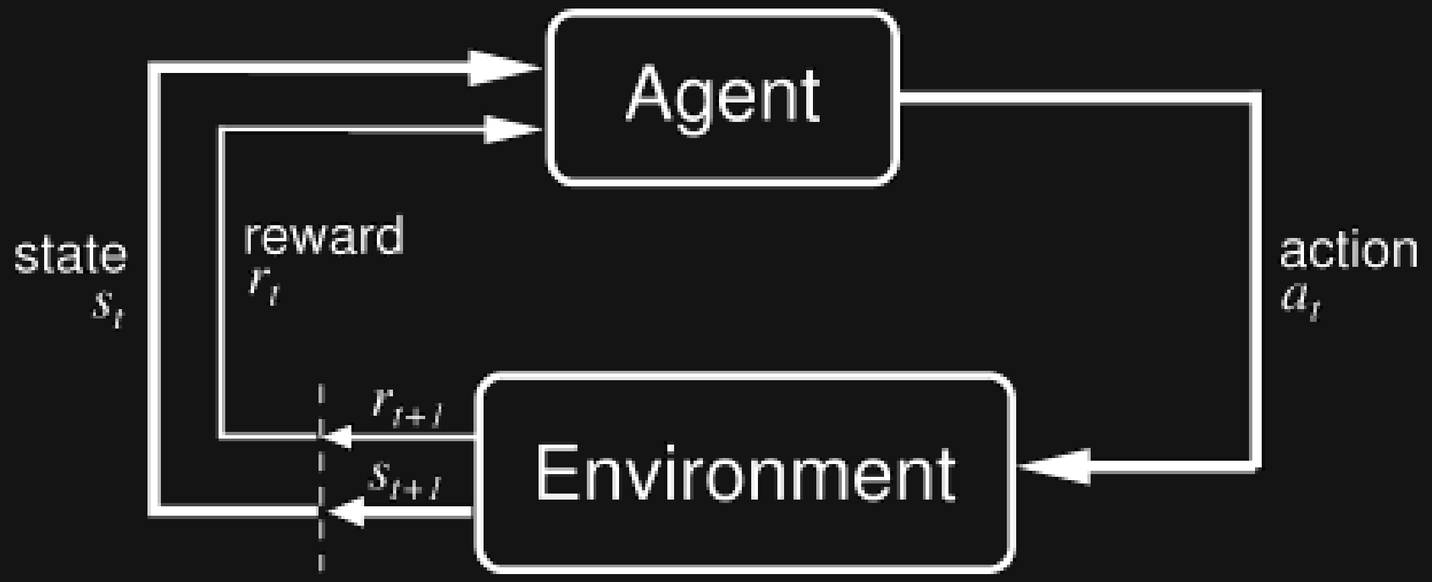
WHAT IS REINFORCEMENT LEARNING?

A GENERAL INTRODUCTION

Reinforcement learning (RL) is *learning by interacting with an environment*. An RL agent *learns from the consequences of its actions, rather than from being explicitly taught* and it selects its actions on basis of its past experiences (exploitation) and also by new choices (exploration), which is essentially trial and error learning.

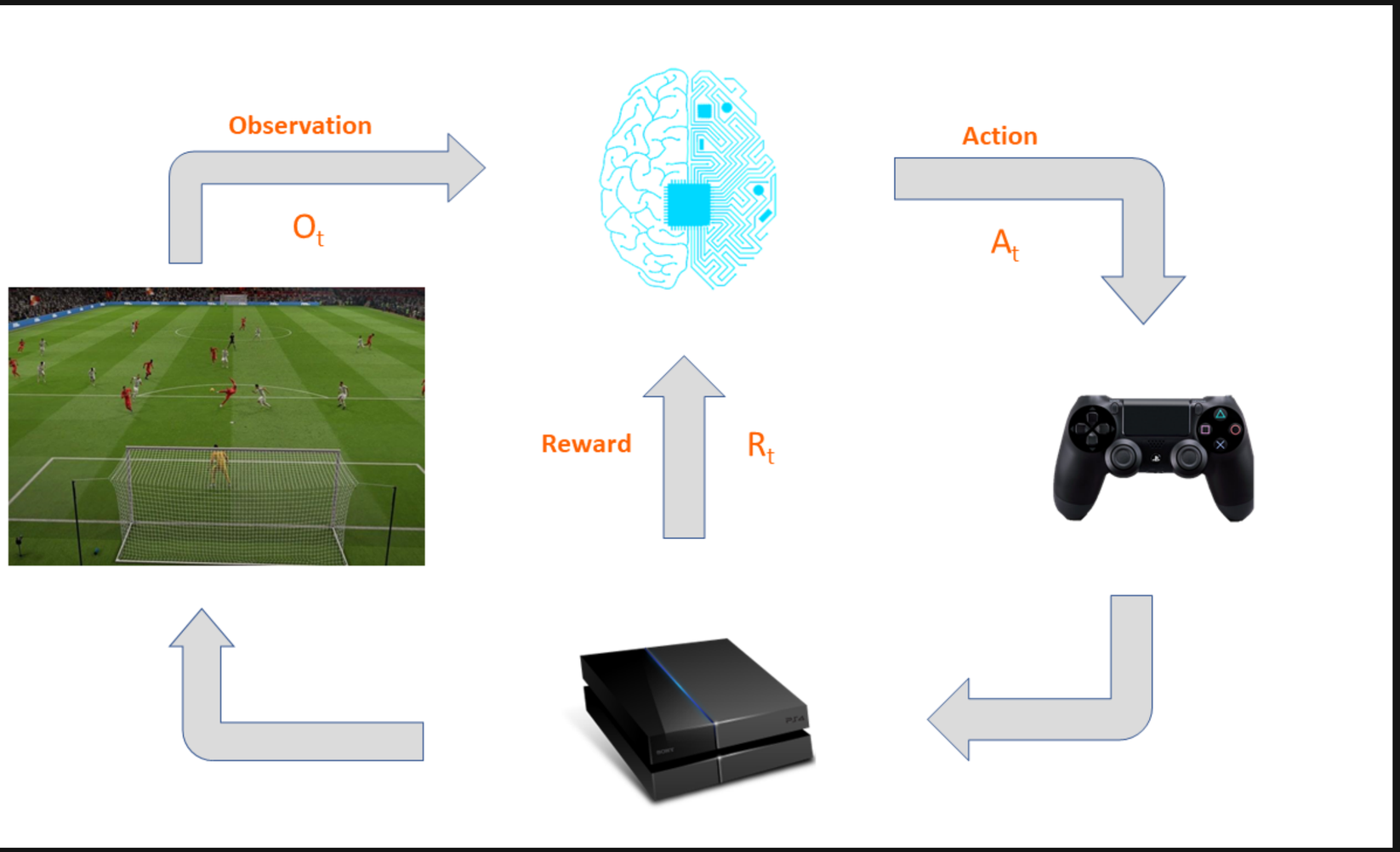
Source: Wiki

REPRESENTATION OF RL



Agent: The learner in an RL environment which we can design.
Environment: Everything our Agent interacts with.
State: State describes the current situation in an environment.
Action: Everything an Agent can do in an environment.
Reward: A scalar feedback signal for taking an action.

EXAMPLE



HOW IS RL DIFFERENT FROM OTHER ML PARADIGMS?

1. Based on ***Reward Signals***.
    - There are no labels. No supervision required.
    - Very differrent from un-supervised either.
    - Reward Signals are specific to RL.
2. There is **no instantaneous feedback** for taking any action.
    - Unlike other Machine Learning techniques in RL we do not get an immediate feedback of how good a certain action was or can be.
3. Data is **not i. i. d (Independent and Identically Distributed)**
    - Unlike other Machine Learning techniques in RL we do not get an immediate feedback of how good a certain action was or can be.

Markov Decision Processes - MDP

Markov State

Definition

A state S_t is *Markov* if and only if

$$\mathbb{P}[S_{t+1} \mid S_t] = \mathbb{P}[S_{t+1} \mid S_1, ..., S_t]$$

- Fundamental building block for defining an RL Environment.
- What it means is "The current state is a sufficient statistic of the future"
- It can be interpreted in different ways, but anyway it tries to say the same thing.

MARKOV PROCESS

A *Markov Process* (or *Markov Chain*) is a tuple $\langle \mathcal{S}, \mathcal{P} \rangle$

- $\mathcal{S}$ is a (finite) set of states
- $\mathcal{P}$ is a state transition probability matrix,
  $\mathcal{P}_{ss'} = \mathbb{P}[S_{t+1} = s' \mid S_t = s]$

- State *s* is memory-less as it is a Markov State
- *S* - state space, *P* - Transition Probability
- With Markov Property we can create a Transition Probability matrix which maps the transitions between any given state pair giving the Markov Property a whole new structure.

MARKOV REWARD PROCESS

RETURN : (Discounted Return)

Before jumping into rewards we need to understand what a *return* is.

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1},$$

- Gamma is the *Discount Factor*
- *Low* Gamma value leads to myopic evaluation
- High Gamma value leads to far-sighted evaluation

The equation says that the *returns* at time *t* is the sum of all subsequent (discounted) future rewards.

MARKOV REWARD PROCESS

Value Function

The value function *v(s)* describes the value of being in a state.

$$v(s) = \mathbb{E}[G_t \mid S_t = s]$$

The point the above equation is trying to make is,

Value of being in a state is the expected return from the current state based on the Markov Property.

MARKOV REWARD PROCESS

Bellman Equation:

The value function *v(s) can be decomposed* into two parts. (Cool)

$$
\begin{aligned}
v(s) &= \mathbb{E}\left[G_t \mid S_t = s\right] \\
&= \mathbb{E}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots \mid S_t = s\right] \\
&= \mathbb{E}\left[R_{t+1} + \gamma\left(R_{t+2} + \gamma R_{t+3} + \ldots\right) \mid S_t = s\right] \\
&= \mathbb{E}\left[R_{t+1} + \gamma G_{t+1} \mid S_t = s\right] \\
&= \mathbb{E}\left[R_{t+1} + \gamma v(S_{t+1}) \mid S_t = s\right]
\end{aligned}
$$

Now the new value function means,

The value of being in a state is the expected return of the sum of reward obtained from the next time step and the discounted value of the next state.

MARKOV DECISION PROCESS

In a tuple of (S, A, P, R, γ) where:

- S is a set of states
- A is the set of actions agent can choose to take
- P is the transition Probability Matrix
- R is the Reward accumulated by the actions of the agent
- γ is the discount factor.

Policy (π)

The behavior of the Agent. (Literally Policy)

$$\pi(a|s) = \mathbb{P}\left[A_t = a \mid S_t = s\right]$$

MARKOV DECISION PROCESS

State-value function: (Vπ)

$$v_\pi(s) = \mathbb{E}_\pi \left[G_t \mid S_t = s \right]$$

Expected return for state *s* is the value for that state for following a policy π

Action-value function: (Qπ)

$$q_\pi(s, a) = \mathbb{E}_\pi \left[G_t \mid S_t = s, A_t = a \right]$$

Expected return for **state *s* *and taking an action a*** is the value for that state for following a policy π.

Note: The Bellman Equation can also be used here to decompose the value functions which I'm leaving for you to do.

OPTIMAL VALUE FUNCTIONS: MDP

Optimal *state-value* function:

$$v_*(s) = \max_\pi v_\pi(s)$$

Denotes the maximum value function for all policies for every state.

Optimal *action-value* function:)

$$q_*(s,a) = \max_\pi q_\pi(s,a)$$

Denotes the maximum value for all actions for every state for all policies.

After applying Bellman Equation: (The Q we care about)

$$q_*(s,a) = E\left[R_{t+1} + \gamma \max_{a'} q_*(s',a')\right]$$

Q-Learning

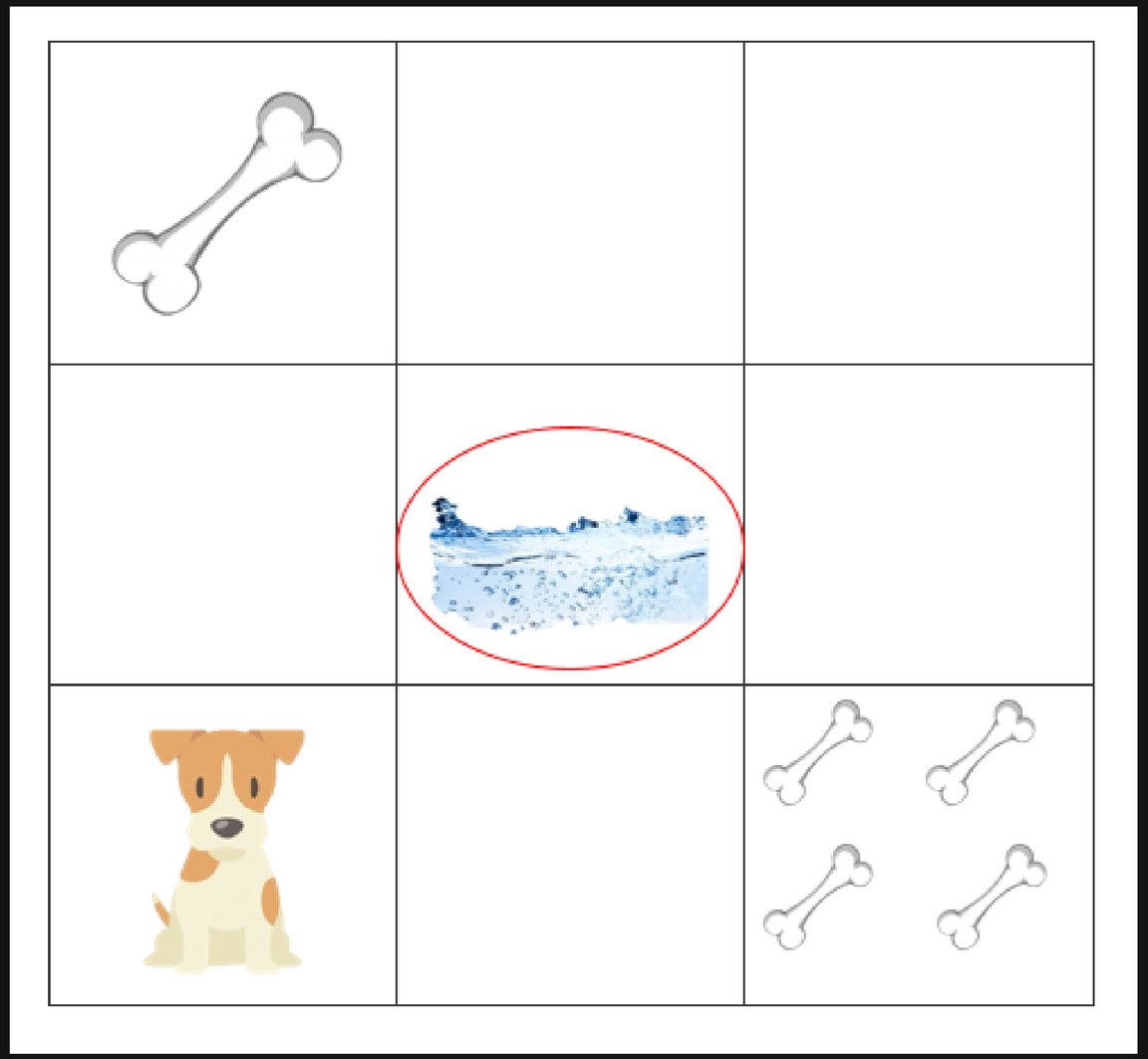
- Q-Learning essentially means learning the optimal q values as we have seen in the last slide.
- Q-Learning is an iterative solution

How it works?

By iteratively updating the Q-values for each state-action pair using Bellman equation until the Q-function converges to optimal Q function.

- We store the optimal Q-values for every single state-action pair in a **Q-Table**
- We use the **ε-Greedy** strategy to better balance the rate of exploration and exploitation while making subsequent decisions.
- Q-Table is the representation of optimal actions to take given any state, thus solving the environment.

EXAMPLE:



+1 Reward for 1 bone

- 1 Reward for empty

+10 Reward for 4 bones (Game Over)

-10 Reward for falling inside the pit (Game Over)

EXPLORATION-EXPLOITATION TRADE OFF

Exploration is the act of exploring the environment to find out information about it.

Exploitation is the act of exploiting the information that is already known about the environment in order to maximize the return.

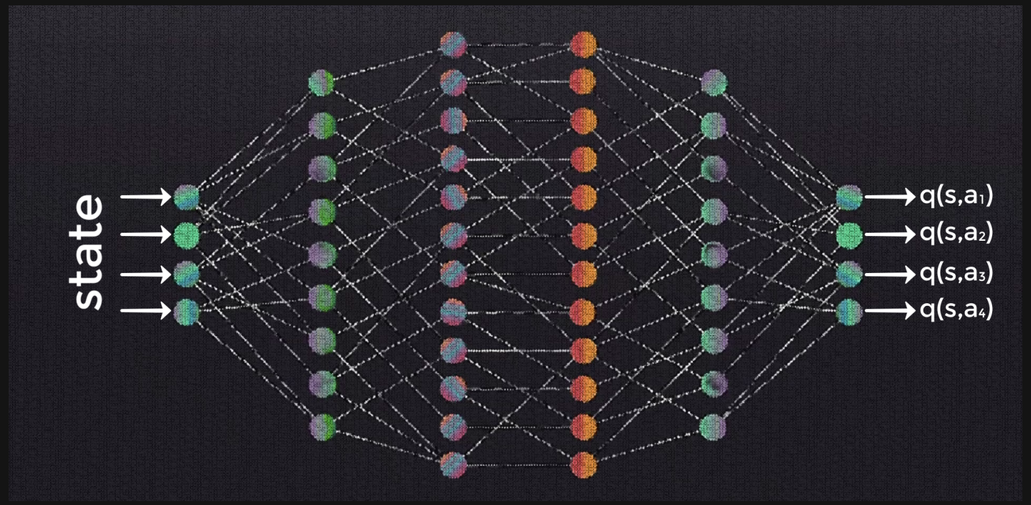
> The trade-off is that, since the goal of the agent is to maximize the cumulative reward, exploiting alone can be a good option but if keeps exploiting things it already knows about the environment there wouldn't be any progress. Some degree of exploration is necessary to hit the right balance.

Unique to Reinforcement Learning

Deep Q-Learning

- The Q-Table is good for small action-observation spaces.
- Imagine using an agent like that to play Dota-2. (famous video game)
- It then becomes computationally inefficient and infeasible to achieve.
- So what can work like a magic approximator for us then? Deep NNs!!!

For each given state input, the network outputs estimated Q-values for each action that can be taken from that state.



Experience Replay or Replay Memory

- We store the agent's experiences at each time step in a data set called the replay memory of finite size N. Experience is represented as,

$$e_t = (s_t, a_t, r_{t+1}, s_{t+1})$$

Why is this needed?

Why training on random samples of this Replay Memory rather than sequentially experiencing the environment?

> A key reason for using replay memory is to break the correlation between consecutive samples as the experiences are sequential in nature, thus avoiding unnecessary bias in the network.

TARGET NETWORK

- The Bellman equation which is used to compute the loss,

  loss = $q*$ (s, a) - q (s, a)
- For the Bellman equation there is a need to calculate the right side of the equation which needs optimal values from s' and a'
- This requires a second pass through the network each time.

If *s* and *s′* have only 1 step in between and they share the same network, chances are that the optimizer updates the Q-values, along with the target Qs as well, which makes the algorithms unstable.

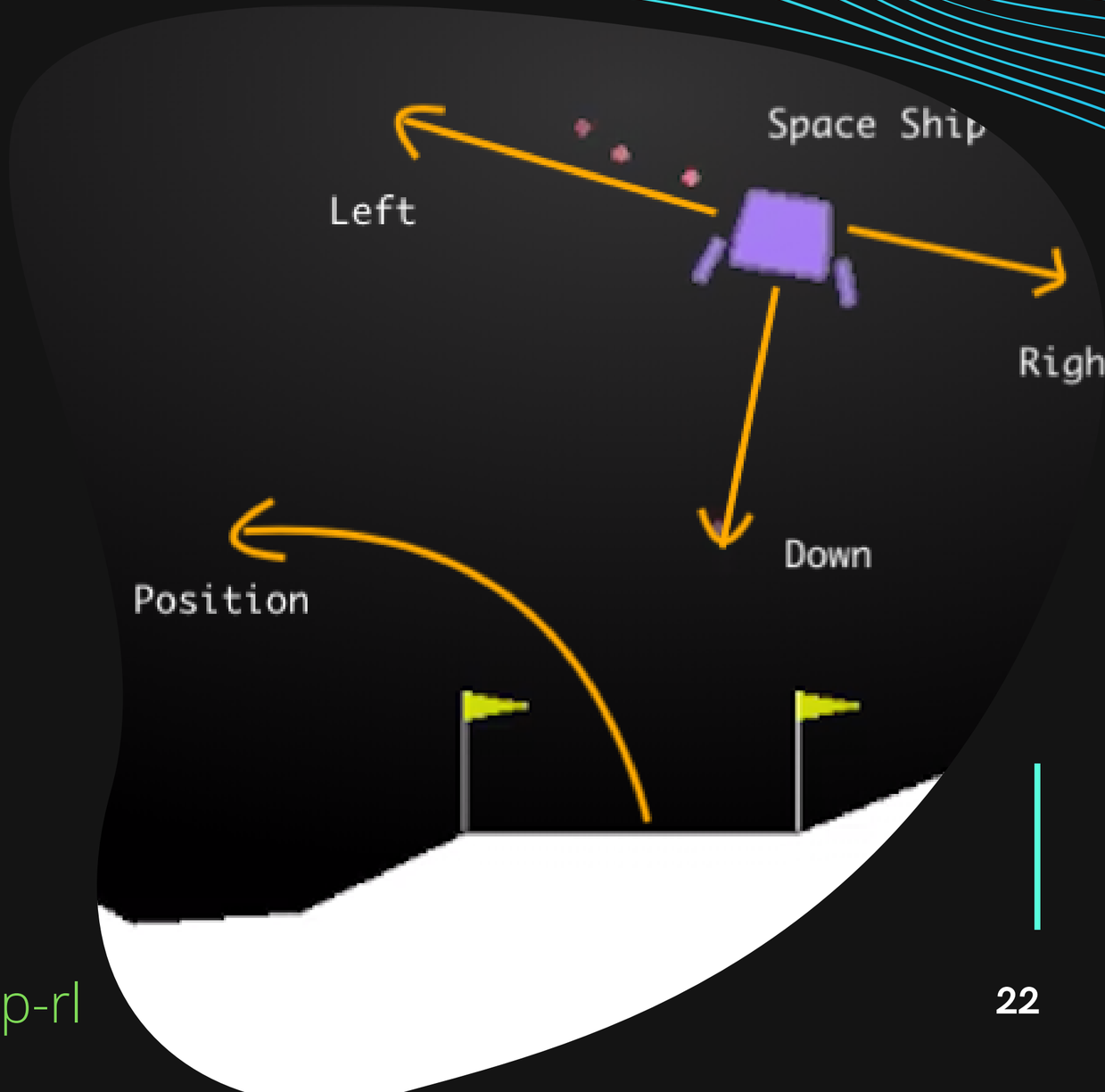
Rather than doing a second pass to the policy network to calculate the target Q-values, we instead obtain the target Q-values from a completely separate network, appropriately called the target network.

Deep Q-Learning Algorithm

1. Initialize replay memory capacity.
2. Initialize the policy network with random weights.
3. Clone the policy network, and call it the target network.
4. For each episode:
    a. 1. Initialize the starting state.
    b. 2. For each time step:
        i. 1. Select an action.
            1. Via exploration or exploitation
        ii. 2. Execute selected action in an emulator.
        iii. 3. Observe reward and next state.
        iv. 4. Store experience in replay memory.
        v. 5. Sample random batch from replay memory.
        vi. 6. Preprocess states from batch.
        vii. 7. Pass batch of preprocessed states to policy network.
        viii. 8. Calculate loss between output Q-values and target Q-values.
            1. Requires a pass to the target network for the next state
        ix. 9. Gradient descent updates weights in the policy network to minimize loss.
            1. After x time steps, weights in the target network are updated to the weights in the policy network.

Hands-On: Lunar Lander

- The goal here is to solve the **OpenAI Gym** environment called **LunarLander-v2**
- Isn't that a very interesting problem to solve?



Action Space:

0- Do nothing
1- Fire left engine
2- Fire down engine
3- Fire right engine

GitHub Repo: (All the code)

https://github.com/abhinand5/lunar-lander-deep-rl

HOPE YOU LEARNT SOMETHING NEW

Thanks a lot for joining.

EMAIL

abhinandalfio@gmail.com
abhinand.b@qpiai.tech

MOBILE

938-420-7320

LINKS

www.linkedin.com/in/abhinand-05/
www.github.com/abhinand5
www.kaggle.com/abhinand05