Deep Reinforcement learning

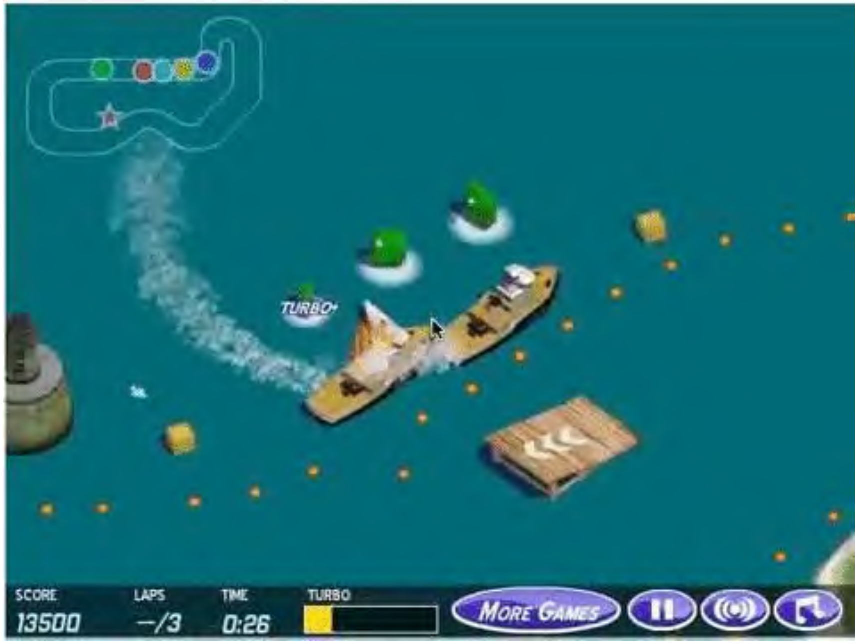
ML course – fall 1401

Dr Rezazadegan

Kiana Asgari

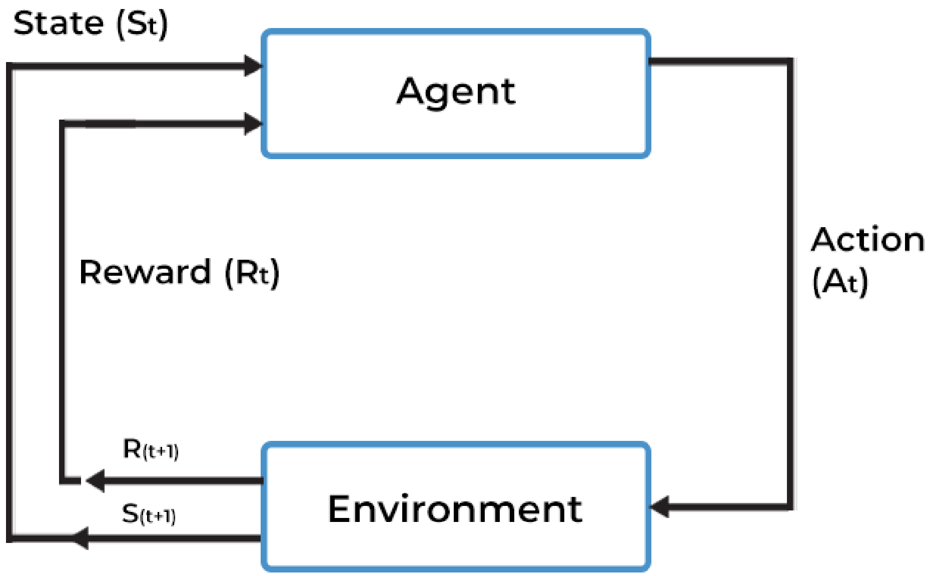
Zahra Janalizadeh

The Wonder of Reinforcement Learning



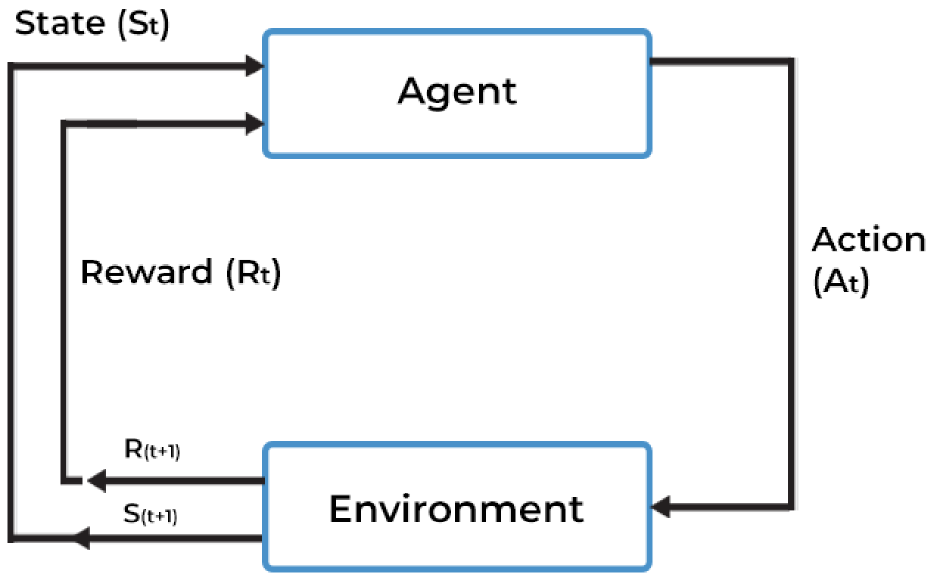
Coast Runners Game, played by RL agent.

Reinforcement Learning (RL): Key Concepts



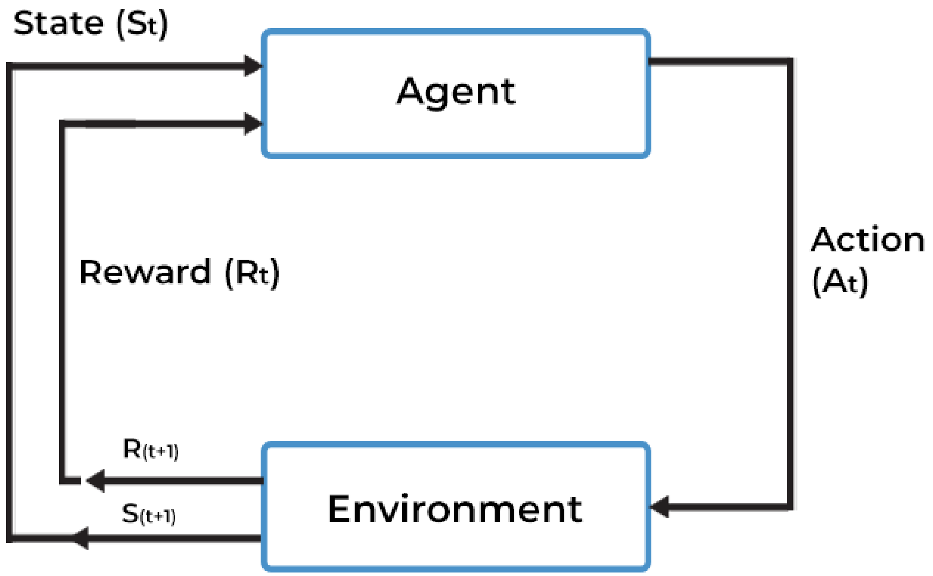
Agent: takes actions, for example, a drone. The algorithm is the agent, and in real life we are the agents.

Reinforcement Learning (RL): Key Concepts



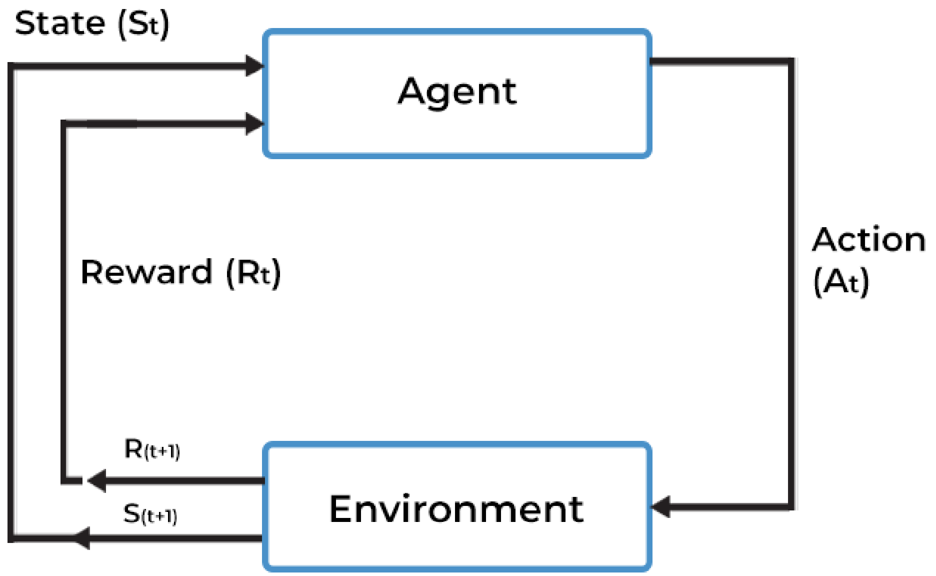
Environment: the world in which the agent exists and operates.

Reinforcement Learning (RL): Key Concepts



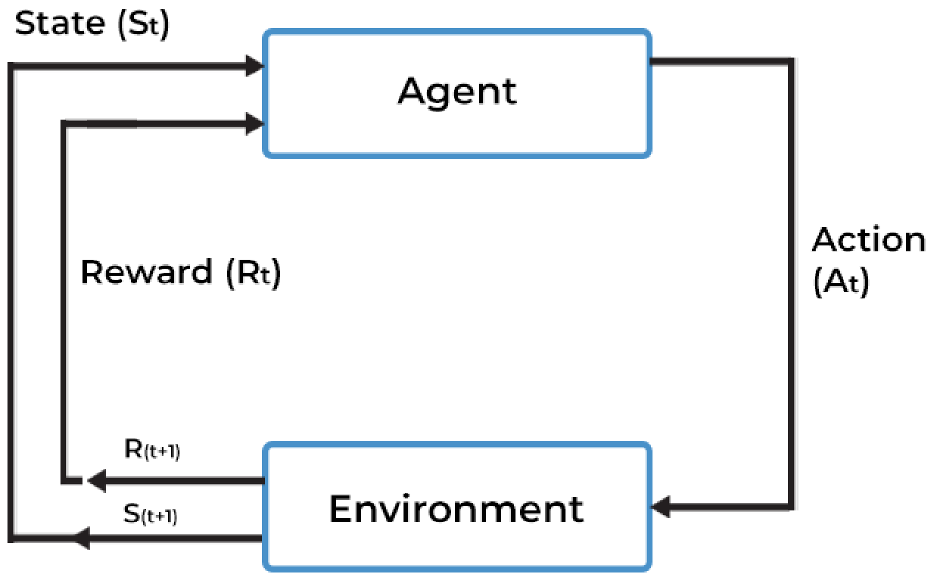
Action: a move the agent can make in the environment.

Reinforcement Learning (RL): Key Concepts



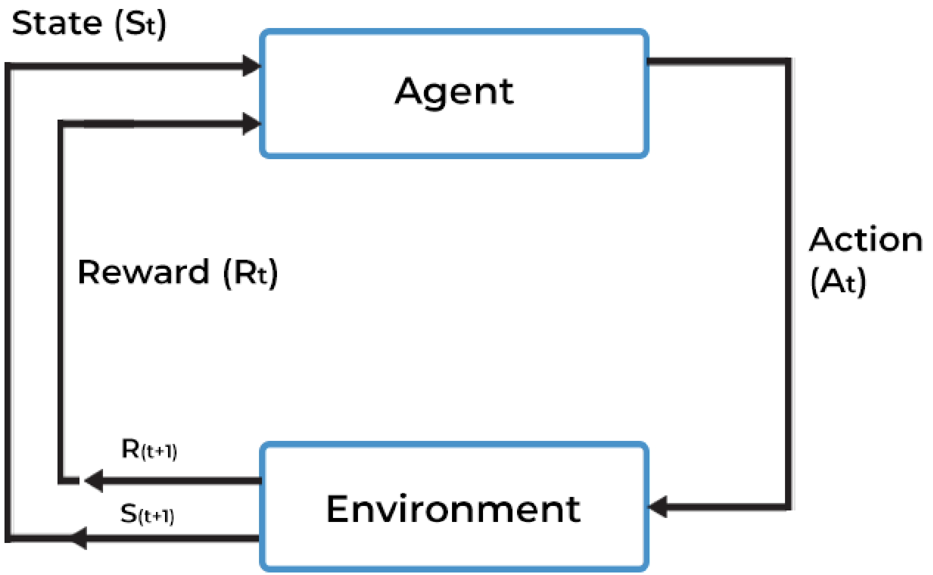
State: a situation which the agent perceives.

Reinforcement Learning (RL): Key Concepts



Reward: feedback that measures the success or failure of the agent’s action.

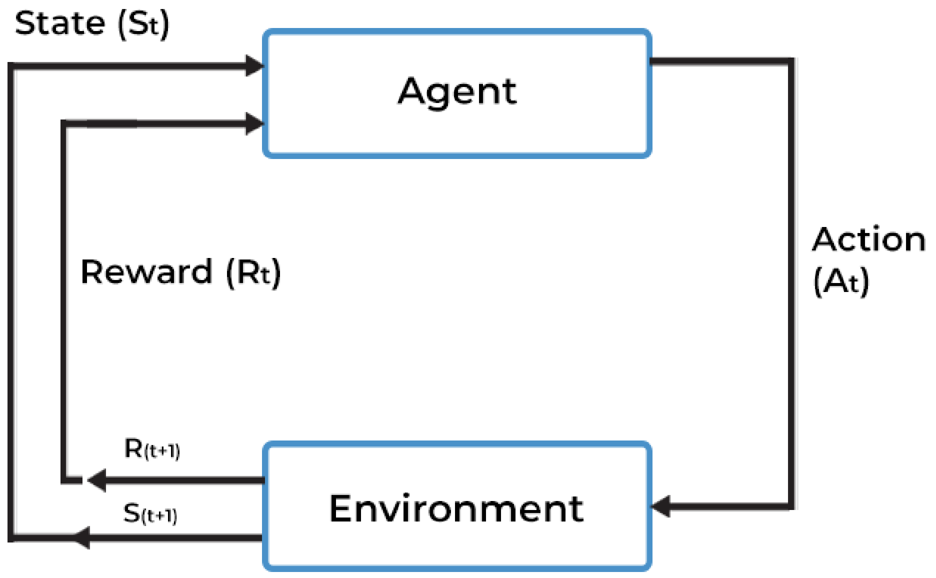
Reinforcement Learning (RL): Key Concepts



Total Reward

$$R_t = \sum_{i=t}^{\infty} r_i = r_t + r_{t+1} + \cdots + r_{t+n} + \cdots$$

Reinforcement Learning (RL): Key Concepts



Discounted Total Reward

$$R_t = \sum_{i=t}^{\infty} \gamma^i r_i = \gamma^t r_t + \gamma^{t+1} r_{t+1} + \cdots + \gamma^{t+n} r_{t+n} + \cdots$$

Defining the Q-function

$$R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots$$

Total Reward, $\boldsymbol{R_t}$, is the discounted sum of all rewards obtained from time $\boldsymbol{t}$

$$Q(s, a) = \mathbb{E}[R_t]$$

The Q-function captures the **Expected Total Future Reward** an agent in state, ***s***, can receive by executing a certain action, a

How to take actions given a Q-function?

$$Q(\underbrace{s}_{\text{state}},\underbrace{a}_{\text{action}}) = \mathbb{E}[R_t]$$

Ultimately, the agent needs a **policy *π*(s)*,** to infer the best action to take at its state, s

Strategy: the policy should choose an action that maximizes the future reward

$$\pi^*(s) = \underset{a}{\operatorname{argmax}}\ Q(s,a)$$

Deep Reinforcement Learning Breakthrough

Deep Reinforcement Learning Breakthrough

- DRL algorithms are able to handle high-dimensional state space!

Deep Reinforcement Learning Breakthrough

- ➔ DRL algorithms are able to handle high-dimensional state space!
- ➔ Beginning around 2013, DeepMind showed impressive learning results using deep RL to play **Atari** video games.

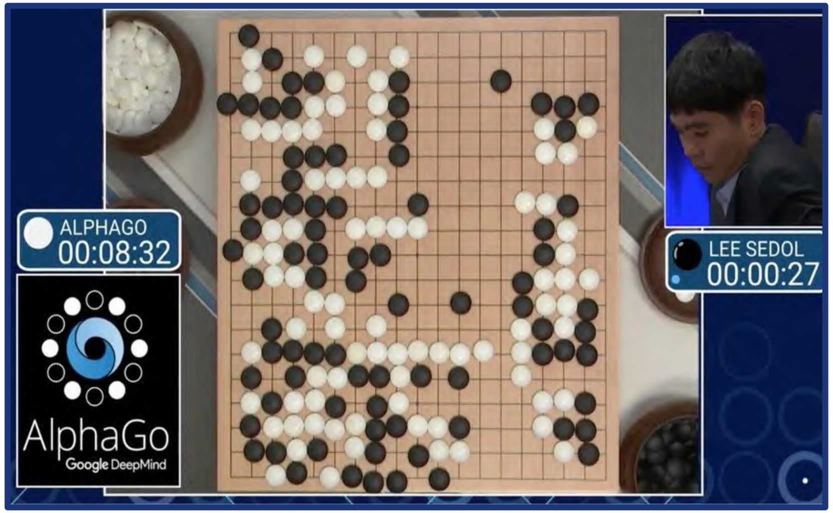
Deep Reinforcement Learning Breakthrough

- ➔ DRL algorithms are able to handle high-dimensional state space!
- ➔ Beginning around 2013, DeepMind showed impressive learning results using deep RL to play **Atari** video games.
- ➔ In 2016, DeepMind was able to solve **Go**, the most challenging classical problem for AI.

AlphaGo

AlphaGo! Making History

AlphaGo is the first computer program to defeat a professional human Go player, the first to defeat a Go world champion, and is arguably the strongest Go player in history.



Deep Reinforcement Learning Algorithms

Value Based

Find $Q(s,a)$

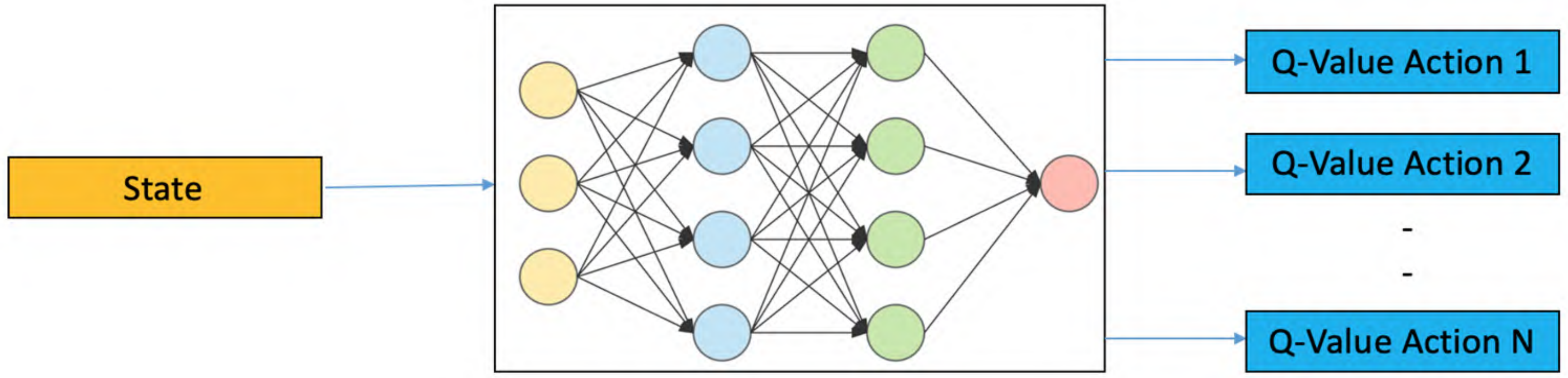
$$a = \underset{a}{argmax}\,Q(s,a)$$

Policy Based

Find $\pi(s)$

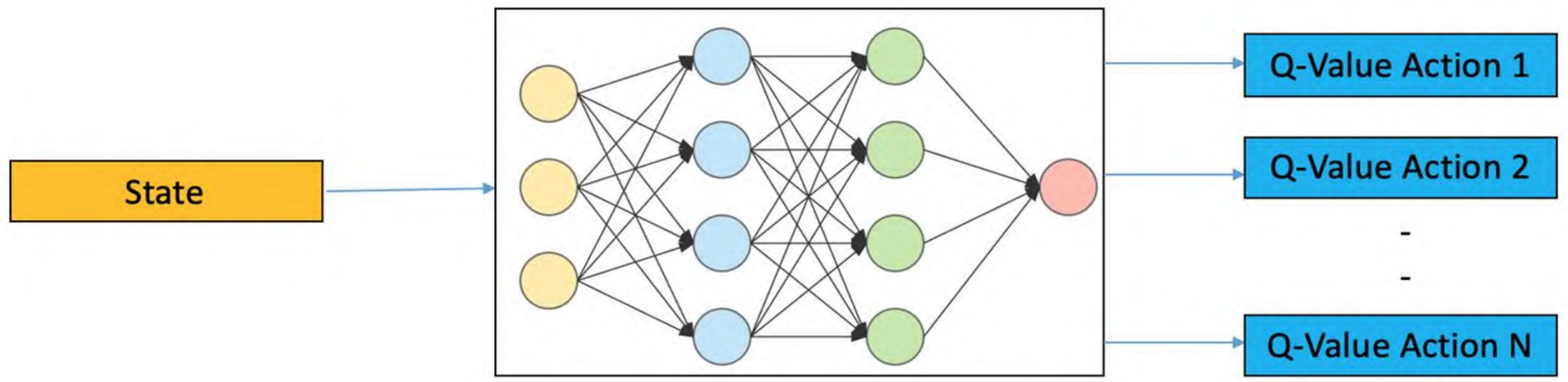
Sample a ⍰π(s)

Deep Q Network (DQN)



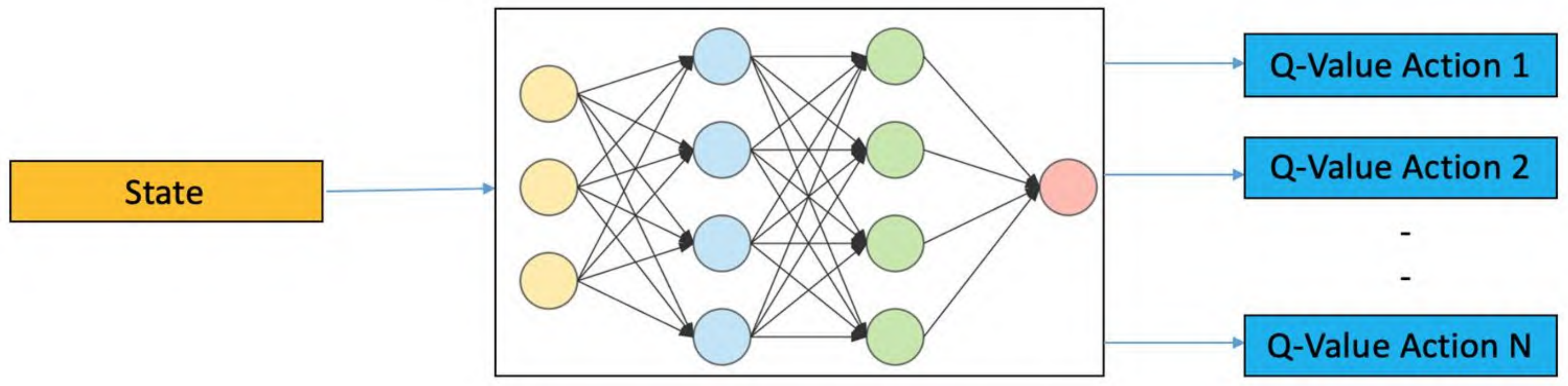
Deep Q Learning

Deep Q Network (DQN)



Step 1: Choose your network structure and initial the DQN.

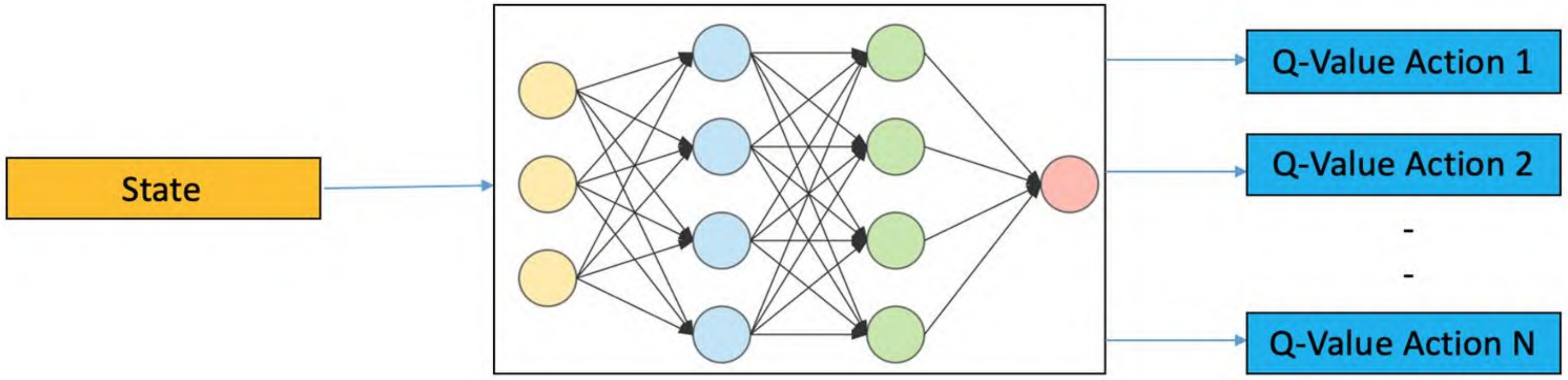
Deep Q Network (DQN)



Step 2: Choose an action from epsilon-greedy algorithm

$$\begin{cases} a = \underset{a}{\operatorname{argmax}}\, Q(s,a) & \text{with probability } 1-\varepsilon \\ a = \text{random action} & \text{with probability } \varepsilon \end{cases}$$

Deep Q Network (DQN)



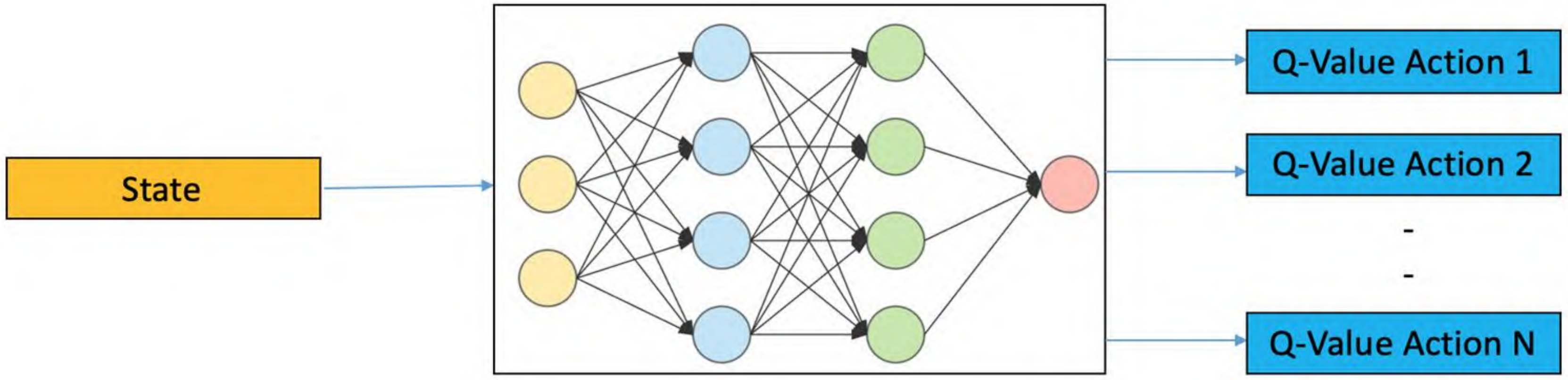
Step 3: Define your loss.

TD-target is UNKNOWN!

Current Q-value

$$LossFunction = \boxed{Q_{best}(s_t\ , a_t)} - \boxed{Q(s_t\ ,\ a_t)}$$

Deep Q Network (DQN)

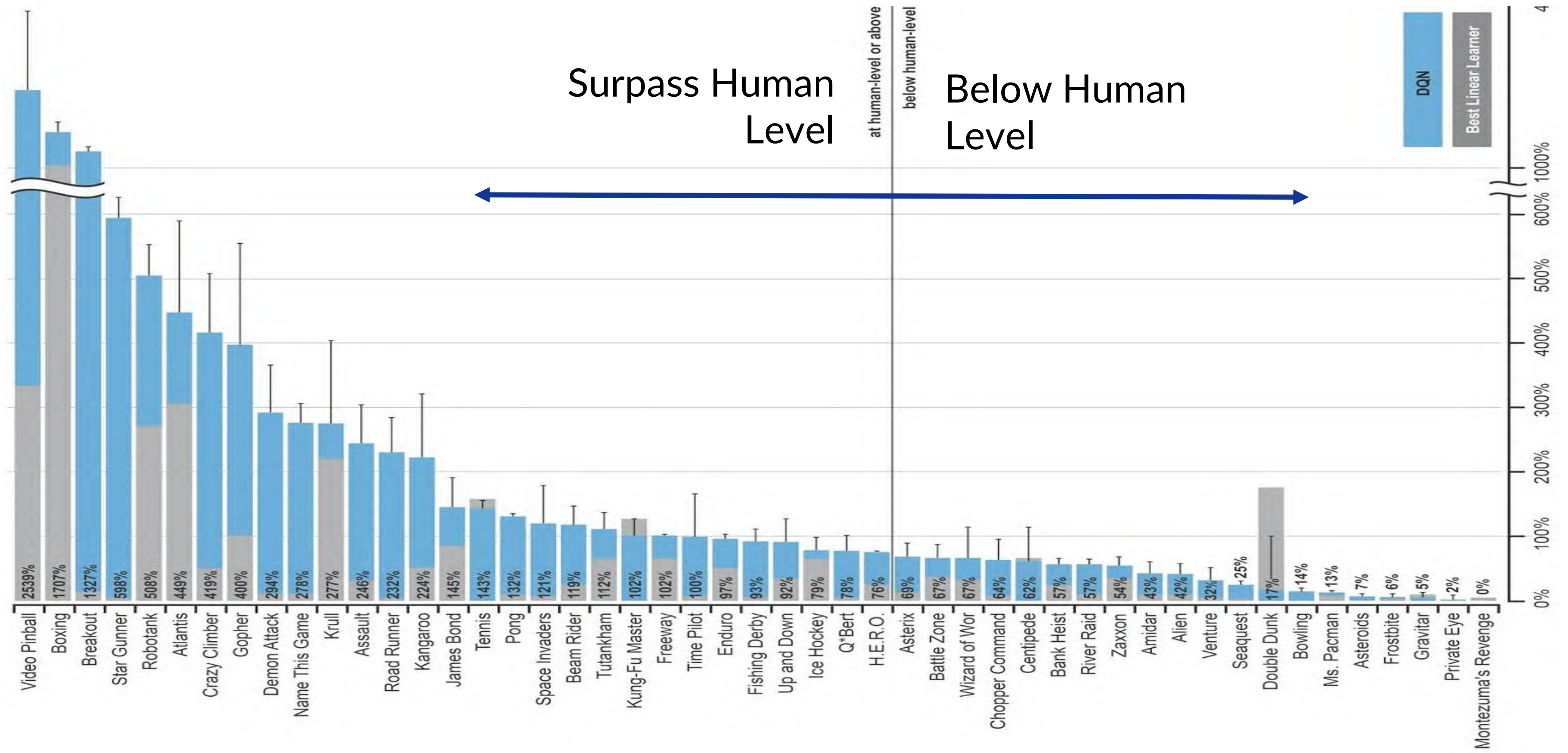


Step 3.5: Update your network weights using the Bellman Equation.

$$LossFunction = \underbrace{\overbrace{\underbrace{\boxed{R_{t+1}}}_{\text{Reward}} + \overset{\text{Discount factor}}{\boxed{\gamma}} \underbrace{\boxed{maxQ(s_{t+1}\ ,\ a)}}_{\text{Maximum next-state Q-value}}}^{\text{Estimated TD-target}}} - \underbrace{\boxed{Q(s_t\ ,\ a_t)}}_{\text{Current Q-value}}$$

Surpass Human Level

Below Human Level



Downsides of Q-learning

Complexity:

- Can model scenarios where the action scape is discrete and small
- <u>Cannot</u> handle continuous action spaces

Downsides of Q-learning

Complexity:

- Can model scenarios where the action scape is discrete and small
- Cannot handle continuous action spaces

Imagine you want to predict steering wheel angle of a car!

Downsides of Q-learning

Complexity:

- Can model scenarios where the action scape is discrete and small
- Cannot handle continuous action spaces

Imagine you want to predict steering wheel angle of a car!

Flexibility:

- Cannot learn stochastic policies, since policy is deterministically computed from the Q function

Downsides of Q-learning

Complexity:

- Can model scenarios where the action scape is discrete and small
- Cannot handle continuous action spaces

Imagine you want to predict steering wheel angle of a car!

Flexibility:

- Cannot learn stochastic policies, since policy is deterministically computed from the Q function

What can we do to overcome this?

Downsides of Q-learning

Complexity:

- Can model scenarios where the action scape in discrete and small
- Cannot handle continuous action spaces

Imagine you want to predict steering wheel angle of a car!

Flexibility:

- Cannot learn stochastic policies, since policy is deterministically computed from the Q function

What can we do to overcome this?

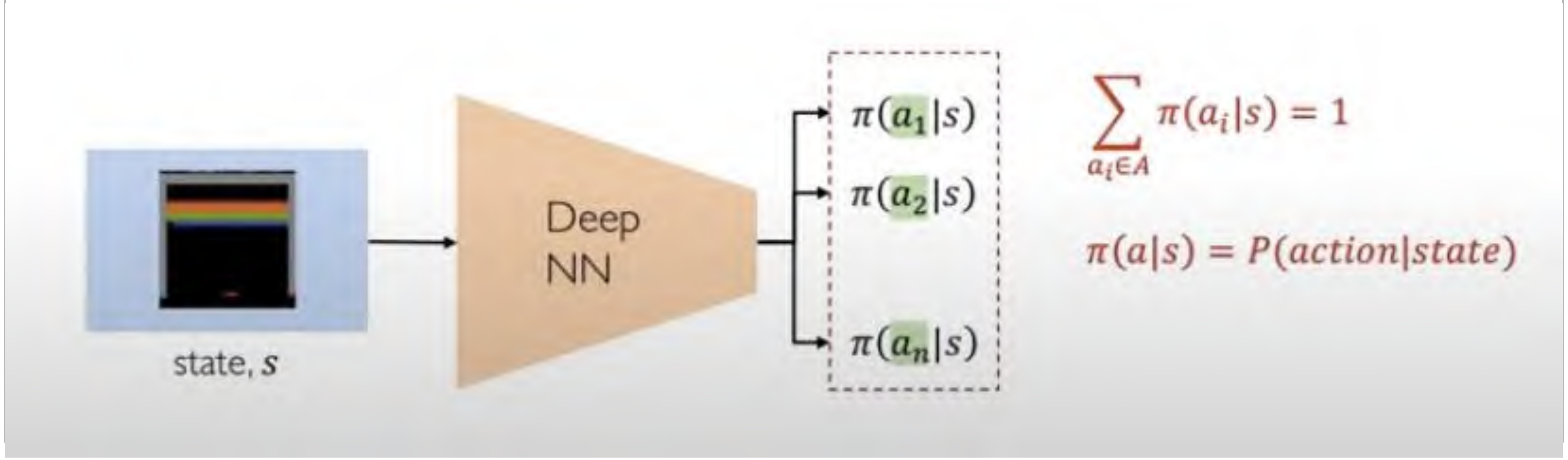
consider a new class of RL training algorithms: **Policy Gradient methods**

Policy Gradient (PG)

DQN (before)	Policy Gradient
Approximating Q and inferring the optimal policy.	Directly optimize the policy!

Policy Gradient (PG)

DQN (before)	Policy Gradient
Approximating Q and inferring the optimal policy.	Directly optimize the policy!

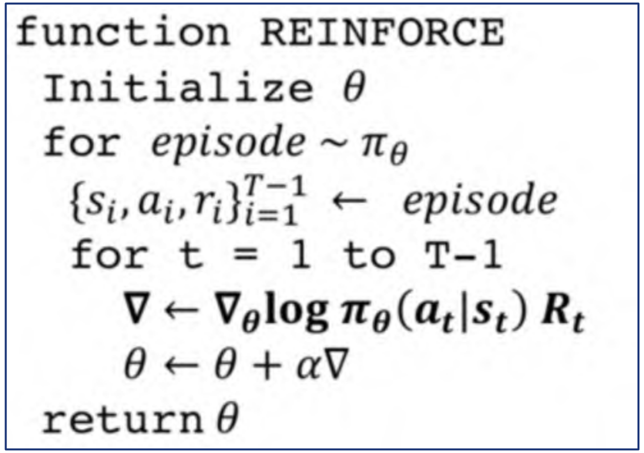

state, s
Deep NN
$\pi(a_1|s)$
$\pi(a_2|s)$
$\pi(a_n|s)$
$\sum_{a_i \in A} \pi(a_i|s) = 1$
$\pi(a|s) = P(action|state)$

Policy Gradient (PG): Training

1. Run a policy for a while
2. Increase probability of actions that lead to high rewards
3. Decrease probability of actions that lead to low/no rewards.

Policy Gradient (PG): Training

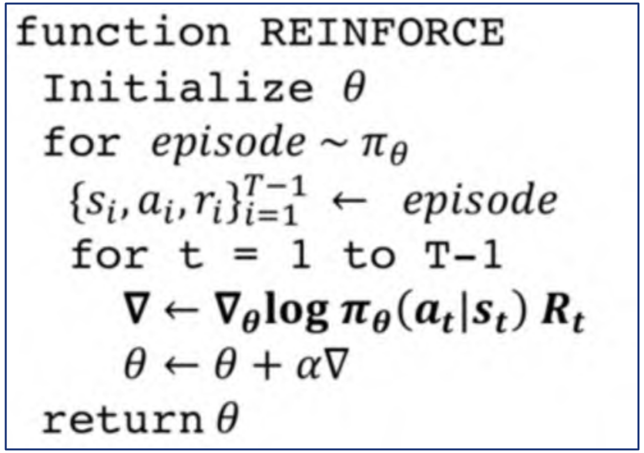
1. Run a policy for a while
2. Increase probability of actions that lead to high rewards
3. Decrease probability of actions that lead to low/no rewards.



```
function REINFORCE
 Initialize $\theta$
 for $episode \sim \pi_\theta$
  $\{s_i, a_i, r_i\}_{i=1}^{T-1} \leftarrow episode$
  for t = 1 to T-1
    $\boldsymbol{\nabla} \leftarrow \boldsymbol{\nabla}_\theta \log \boldsymbol{\pi}_\theta(\boldsymbol{a}_t|\boldsymbol{s}_t)\, \boldsymbol{R}_t$
    $\theta \leftarrow \theta + \alpha\nabla$
 return $\theta$
```

Policy Gradient (PG): Training

1. Run a policy for a while
2. Increase probability of actions that lead to high rewards
3. Decrease probability of actions that lead to low/no rewards.



```
function REINFORCE
 Initialize $\theta$
 for $episode \sim \pi_\theta$
  $\{s_i, a_i, r_i\}_{i=1}^{T-1} \leftarrow episode$
  for t = 1 to T-1
     $\boldsymbol{\nabla \leftarrow \nabla_\theta \log \pi_\theta(a_t|s_t)\, R_t}$
     $\theta \leftarrow \theta + \alpha\nabla$
 return $\theta$
```

$$\nabla_\theta \overbrace{\boxed{\log\pi_\theta(a_t|s_t)}}^{\text{Log-likelihood of action}} \underbrace{\boxed{R_t}}_{\text{Reward}}$$

Stable Baseline3:

A reliable user-friendly implementation of DRL!

Thank you for your Attention! ^.^