Deep Reinforcement Learning For Beginners

Ashab Uddin , PhD Student, ECE, University of Windsor

Ashab Uddin, PhD Candidates, Electrical and Computer Engineering

- **Research Interest:**

- ❑ **Connected Autonomous Vehicle Application(Edge computing, Computational offloading, Caching , etc.)**
- ❑ **Communication Resource Allocation for Vehicular Network**
- ❑ **Deep Reinforcement Learning and Game Theory Application**

Objectives of the session

- Fundamentals of Reinforcement Learning (RL)
- Differentiate between RL and DRL
- Experience a hands-on project
- Q & A

Fundamentals of Reinforcement Learning (RL)

Introduction

- Before digging inside of the basics , let's visit the real world application:
- The first wave of urban robots is here | Challengers

Reinforcement Learning

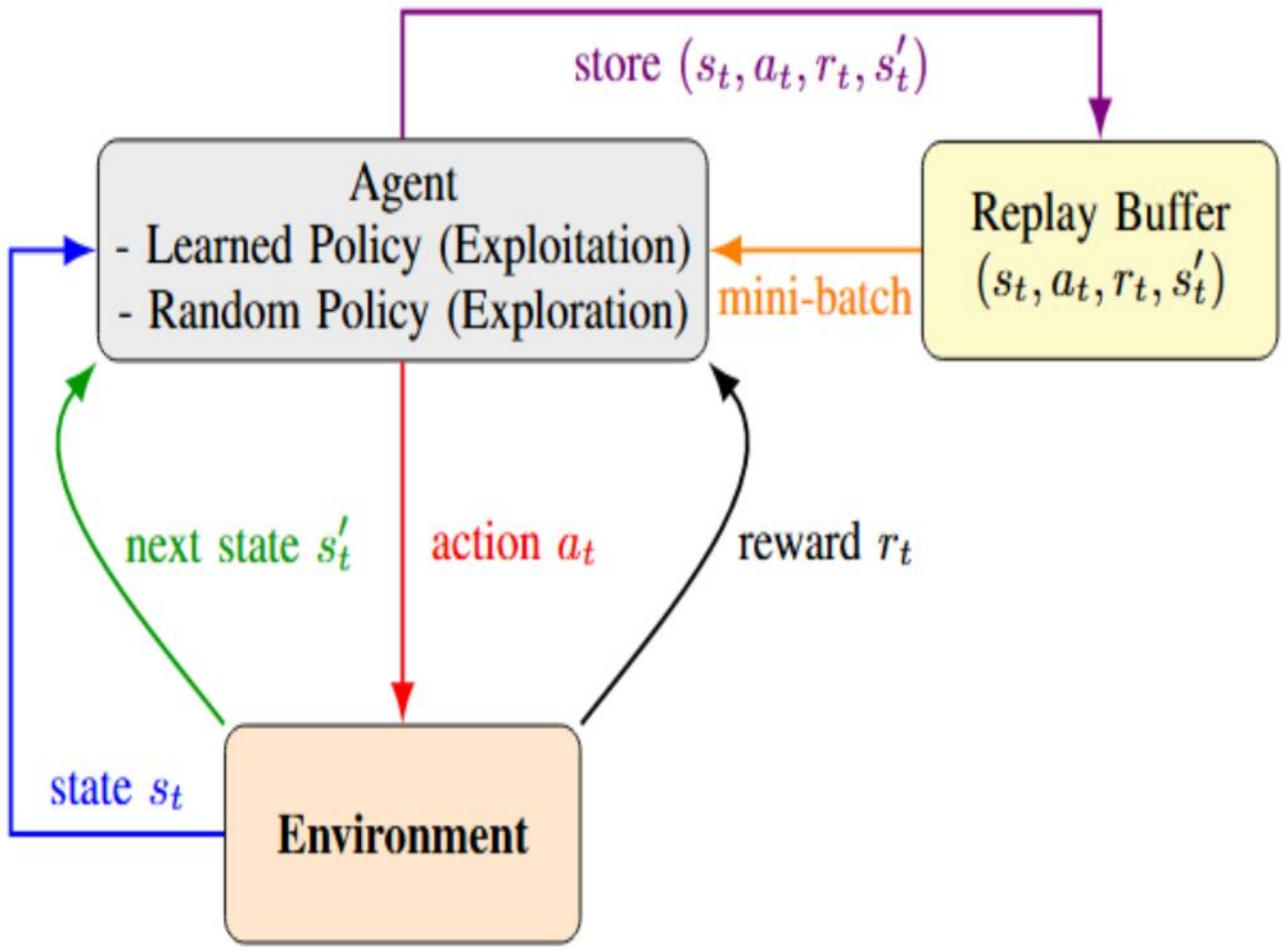
- ❖ Learns from interactions with the environment to achieve a long-term goal
- ❖ The goal is defined by a reward signal, which must be maximized
- ❖ The agent must be able to partially or fully sense the environment state
- ❖ The state is typically represented as a feature vector.
- ❖ State transitions must be related with actions

Exploration and Exploitation

- ❖ Agent learn through exploring action randomly
- ❖ Agent exploit the explored knowledge / learning during training in a controlled way that called exploitation.
- ❖ Example: Initially, a robot take steps (left, right, up, down) randomly with 50% probability and 50% from learned knowledge.

Reinforcement Learning Elements

- **Environment:** The whole system without the agent itself.
- **Agent:** A controller or decision maker.
- **State:** A snapshot of environment / Information Vector
- **Action:** Decision of Agent.
- **Policy:** A policy maps (a total map) states of the environment to actions



Policy Type

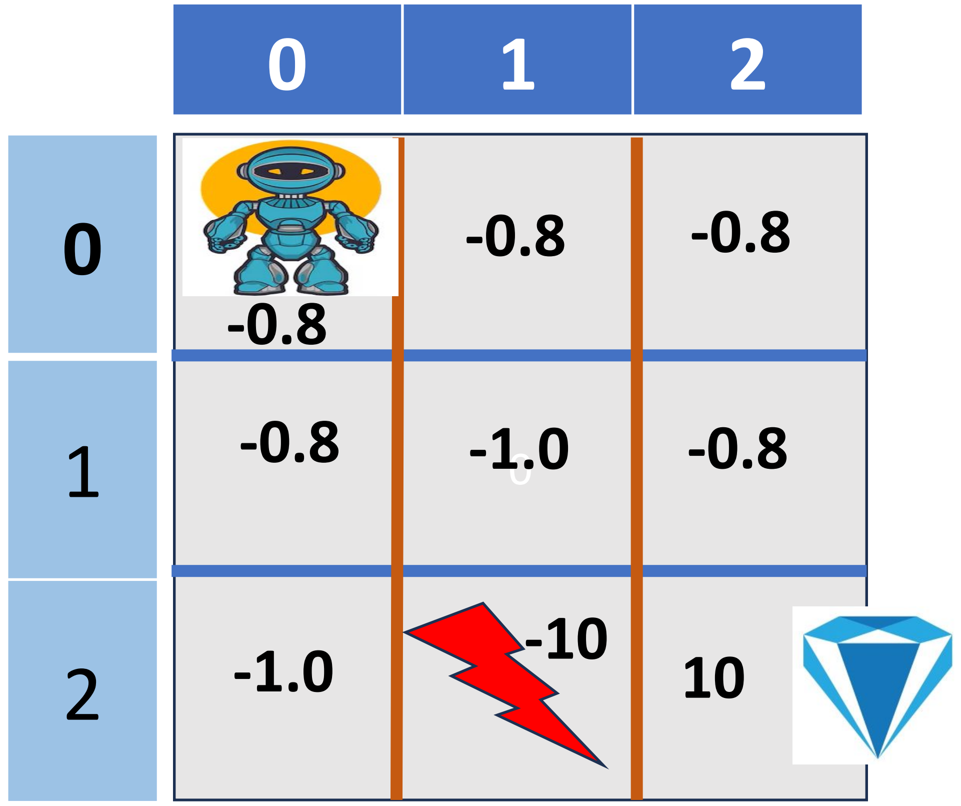
- **Stochastic Policy :**
  - Policy outputs a probability distribution over actions (e.g., SoftMax).
  - Defined as $\boldsymbol{\pi(a|s)}$
  - **Example:** A robot agent policy for current state, **s** is:

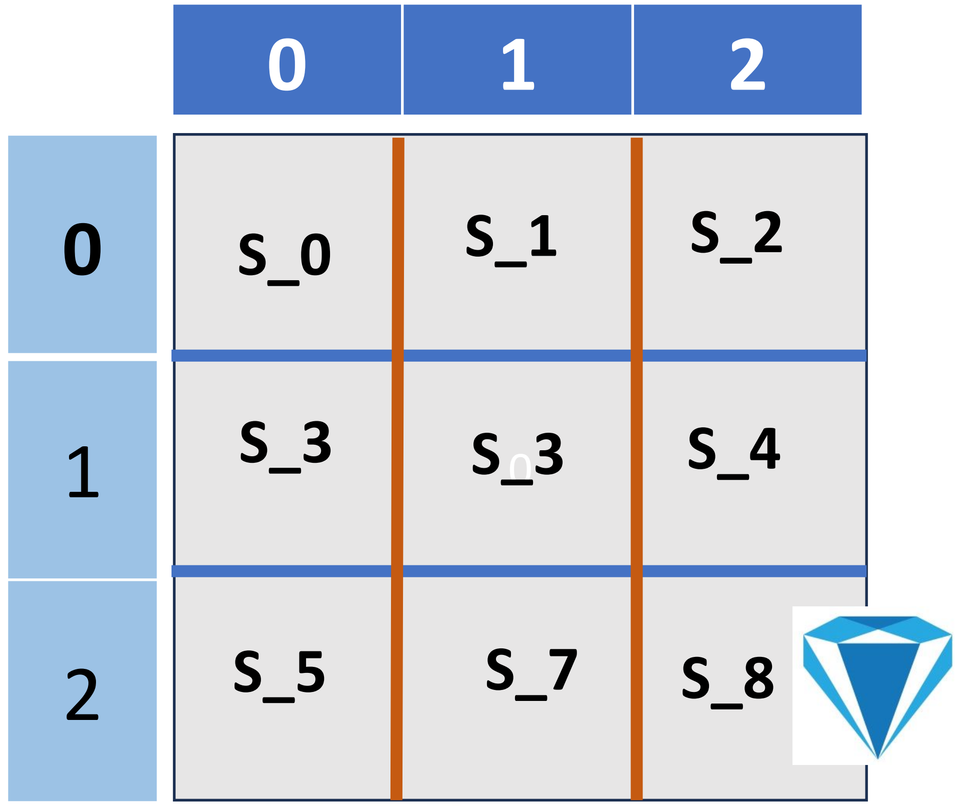
[Left, Right, Up, Down]==[0.3,0.20,0.35,0.15]==Up

- **Deterministic Policy:**
  - Policy outputs a specific action value.
  - Defined as $\boldsymbol{\pi(s) = a}$
  - Example: a car agent decide its velocity **a%** of max velocity.

Reinforcement Learning System Example

- ❑**Environment: ?**
- ❑**Agent: ?**
- ❑**State:** $\boldsymbol{s} =$
  $[s_{00}, s_{01}, s_{02}, s_{10}, s_{11}, s_{12}, s_{20}, s_{21}, s_{22}]$
  s_0=[1,0,0,0,0,0,0,0,0]
- ❑**Action: a=[P(left),P(right),p(up),p(down)]**
- ❑**Policy: will be developed by learning**

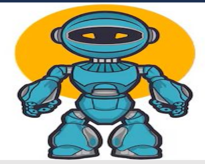
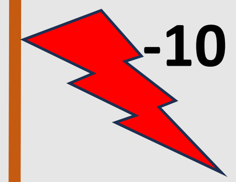




Markov Process

❑Markov Property:

$$\mathbf{p}(s'|s) = Pr\{S_{t+1} = s'|S_t = s\}$$

	0	1	2
0	 -0.8	-0.8	-0.8
1	-0.8	-1.0	-0.8
2	-1.0	 -10	10 

	0.5	0.25	0	0.25	0	0	0	0	0
	0.25	0.25	0.25	0.25	0	0	0	0	0
…… ….									
…… ….									

	0	1	2
0	S_0	S_1	S_2
1	S_3	S_3	S_4
2	S_5	S_7	S_8 

Non-Markovian

- **Non-Markovian Example**
- Depends on History:: How:

❑ $\boldsymbol{Reward}$ $= \color{green}{\boldsymbol{Cell\ Reward}} - \color{red}{\boldsymbol{f(historic\ energy\ drain\ so\ far)}}$

❑ **Objective: Maximize reward with minimizing energy consumption**

	0	1	2
0	-0.8	-0.8	-0.8
1	-0.8	10	-0.8
2	-1.0	-10	10

Non-Markovian to Markovian

- **Easily Solvable problem is Markovian**
- Need to convert it to Markovian
- $\boldsymbol{Reward}$
$$\boldsymbol{= Cell\ Reward - f(\,energy\ drain\ in\ \ current\ step)}$$

Markov Decision Process

Transition based on current state and action

$$\mathbf{p}(s', r|s, a) = Pr\{S_{t+1} = s', R_t = r|S_t = s, a_t = a\}$$

And MDP is a tuple presentation of (S, A, P, R, γ)

$S = set\ of\ states$

$A = set\ of\ action$

$P = Probability\ \ Transition\ \ Matrix$

$R = set\ of\ rewards$

$\gamma = discounted\ factor$

Probability Transition

$$\boldsymbol{Probability\ \ Transition\ \ Matrix} : \mathbf{p}(\boldsymbol{s}', \boldsymbol{r} | \boldsymbol{s}, \boldsymbol{a})$$

$$\sum_{s'} P(s', r | s, a) = 1$$

Left	1	0	0	0	0	0	0	0	0
	0	1	0	0	0	0	0	0	0
Up	1	0	0	0	0	0	0	0	0
Down	0	0	0	1	0	0	0	0	0

	0	**1**	**2**
0	S_0	S_1	S_2
1	S_3	S_3	S_4
2	S_5	S_7	S_8

Expected Return and Discounted Factor

Expected Return, $G(s_t) = r_{t+1} + \gamma * r_{t+2} + \gamma^2 * r_{t+3} + \cdots.$

First Step Reward$= r_{t+1}$

Future Reward= $\gamma * r_{t+2} + \gamma^2 * r_{t+3} + \cdots. = \gamma * (r_{t+2} + \gamma^1 * r_{t+3} + \cdots.)$

Put together, $G(s_t) = r_{t+1} + \gamma * G(s_{t+1})$

- The discount factor is like a "how fast it gains" maximization :
- Lower $\gamma \rightarrow$ **favors quick wins**(agent is short sighted)
- Higher $\gamma \rightarrow$ **favors total sum**, even if it takes longer(agent is far sighted)

State Value and Action Value Function

❖ State value function [1] :

how good a state is, following a policy π:

❖ $V_\pi(s) = \mathbb{E}_\pi[G_t \mid s_t = s] = \mathbb{E}_\pi[r_t + \gamma G_{t+1} \mid s_t = s]$

$$= \sum_a \pi(a|s) \sum_{s',r} p(s'|s,a)(r + \gamma\, V_\pi(s'))$$

$$= \sum_a \pi(a|s) Q_\pi(s,a) \quad \ldots .(1)$$

❖ Action value function [1]:

how good a (state , action) fair is, following a policy π:

❖ $Q_\pi(s,a) = \sum_{s',r} p(s'|s,a)(r + \gamma\, V_\pi(s'))$

$$= \sum_{s',r} p(s'|s,a)[r + \gamma \sum_{a'} \pi(a'|s') Q_\pi(s',a')] \ldots\ldots(2)$$

Bellman Equation

❖ **Bellman Optimality[1]:**

$$V_{\pi*}(s) = \max_a [r(s,a) + \gamma \sum_{s'} p(s'|s,a)\, V_\pi^*(s')] \text{ ....(3)}$$

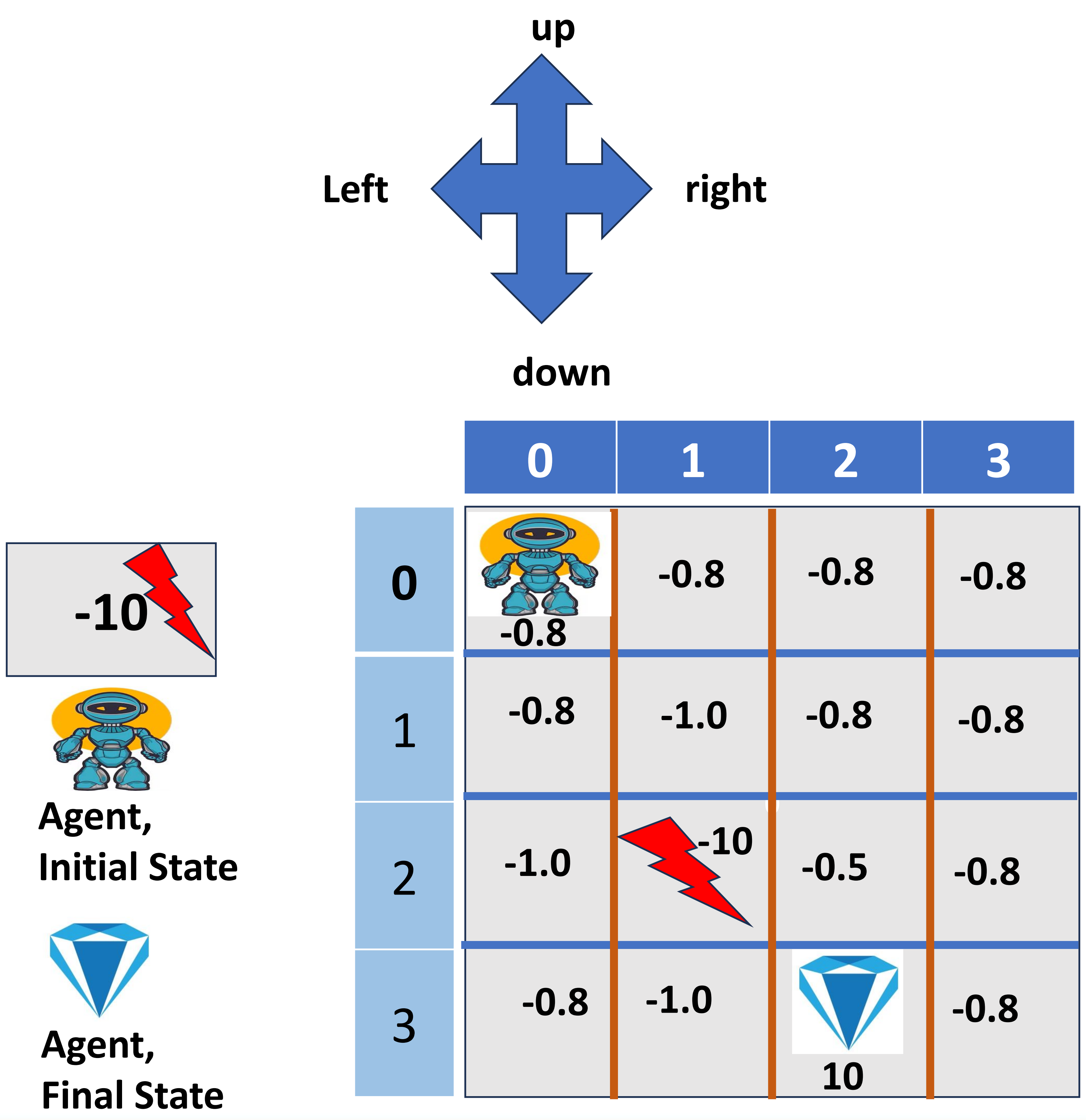
max_a denotes that we are taking the maximum over all possible actions.

$$Q_{\pi*}(s,a) = r(s,a) + \sum_{s',r} p(s'|s,a) * \gamma * \max_{a'} Q^*{}_\pi(s',a') \text{ ....(4)}$$

$max_{a'}$ denotes that we are taking the maximum over all possible actions.

❖ **π^* called the optimal policy among all policies π**

How RL Converge ?



- Objective is to reach final state with maximum reward.
- Optimal path:

(0,0)>(0,1)>(0,2)>(1,2)>(2,2)>(3,2):Return=7.1

- $\boldsymbol{Q\big((0,0),R\big) = (-0.8) + \max_{a}(Q\big((1,0),a\big) = (-0.8) + (-2.1|s = (1,0), a = R))}$

- Suboptimal Path: **Return=-3.1**
- Suboptimal Path: **Return=-3.6**

Differentiate between RL and DRL

RL:: Value Iteration Method

Bellman optimality update:

$$\mathbf{V}_k(\mathbf{s}) = \max_{a}\left[r(s,a) + \gamma \sum_{s'} p(s'|s,a)\, V_{k-1}(s') \right]$$

Policy Extraction: when $\mathrm{V}^*(\mathbf{s}) = \mathbf{V}_k(\mathbf{s})$

$$\pi^*(s) = \operatorname{argmax}_{a}\left[r(s,a) + \gamma \sum_{s'} p(s'|s,a)\, \mathrm{V}^*(\mathbf{s'}) \right]$$

```
Initialize V(S) ← 0 for all s ∈ S
theta ← small positive number
repeat
   Δ ← 0
   V_old ← V (S)
   for each s ∈ S do
      if s is terminal:
         V(s) ← 0       // or the terminal value
         continue
      // compute Q(s,a) for this state
      best_q ← -∞
      for each a ∈ A(s) do
         q ← Σ'_s P(s'|s,a) [ R(s,a,s') + γ * V_old(s') ]
         if q > best_q: best_q ← q
      Δ ← max(Δ, |best_q - V(s)|)
      V(s) ← best_q
   end for
until Δ ≤ theta
Next:Policy Evaluation
```

RL:: Policy Iteration Method

Bellman update with current policy:

$$\mathbf{V}_k(\,\mathbf{s})$$
$$= \sum_a \pi(a|s) \sum_{s'} p(s'|s,a)\,(r(s,a,s') + \gamma V^{\pi}_{k-1}(s'))$$

Check Policy Improvement:

$$\pi^*(s) =$$
$$\operatorname*{argmax}_a \left[r(s,a) + \gamma \sum_{s'} p(s'|s,a)\, V^*_k(s') \right]$$

Initialize: policy π(s) ∈ A(s), V(s) ← 0 for all s, initialize θ

repeat

Δ ← 0, V_old ← V (S), θ ←theta ← small positive number

for each s ∈ S do

if s is terminal:

V(s) ← 0

continue

$V(s) \leftarrow \sum_a \pi(a|s)\ \sum_s' P(s'|s,a)\ [\ R(s,a,s')\ +\ \gamma\ V\_old(s')\ ]$

Δ ← max(Δ, |V(s) - V_old(s)|)

end for

until Δ < θ

policy_stable ← true

for each s ∈ S do

if s is terminal

continue

a_old← π(s)

$a* \leftarrow argmax_a\ \sum_s' P(s'|s,a)\ [\ R(s,a,s')\ +\ \gamma\ V(s')\ ]$

π(s) ← $a*$

if a* ≠ a_old: policy_stable ← false

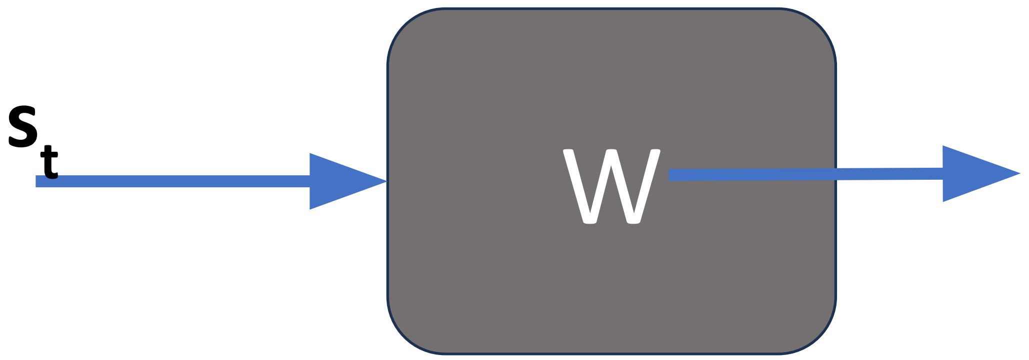
end for

until policy_stable

Output: π (optimal), V (= V^π)

DRL: Value Iteration by Value Function Approximation

- ❑ VFA used supervised learning building a parametrized model to map state to value function or action value function

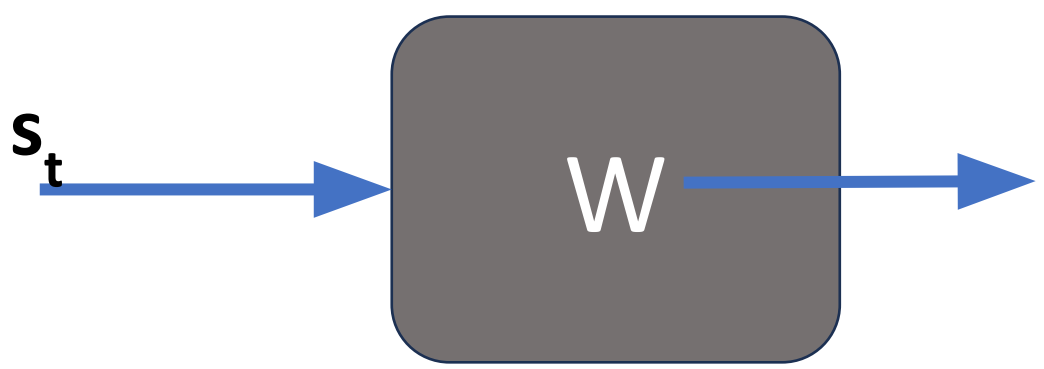


$\hat{V}(s_t; W) \in R^{d_A}$

- ❑ **Value Iteration through VFA[2]**: minimize loss between estimated and expected value of V or Q without doing gradient on policy itself.
- ❑ $J(W) = E_\pi[\left(r + \gamma \hat{Q}_{max,\pi}(s'_t, a'_t, W^-) - \hat{Q}_\pi(s_t, a_t, W)\right)^2]$

DRL: Policy Iteration by Value Function Approximation

- ❑ VFA used supervised learning building a parametrized model to map state to value function or action value function



$\hat{A}(s_t; W) \in R^{d_A}$

- ❑ **Policy Gradient through VFA[2]**: minimize loss function with doing gradient on policy itself.
- ❑ $J(W) = -E_{\pi}[log\, \pi(a_t|s_t, W) * \hat{Q}_{\pi}(s_t, a_t, W')$

Pseudocode for Deep Q Learning Network (DQN)

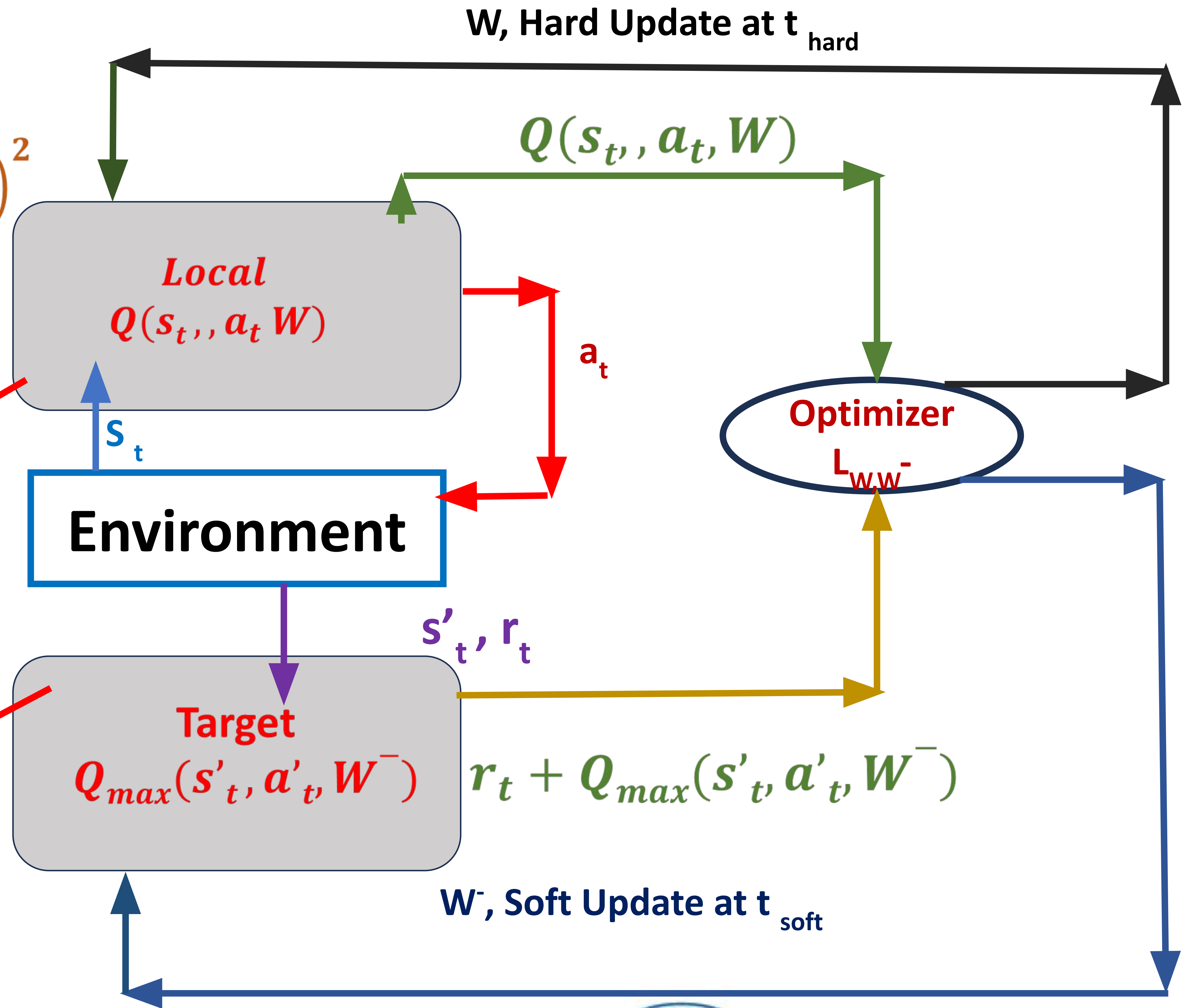
Input: $epoch\_no, \epsilon_{start}, \epsilon_{end}$
Output: *loss, gradients*
//1. Initialization:
Initialize replay memory D with capacity N;
Initialize action-value Q with random weights w;
Initialize target action-value $\hat{Q}$ weights $w^{-} \leftarrow w$;
Initialize scores with window size;
$\epsilon \leftarrow \epsilon_{start}$;
for *episode* $\leftarrow$ *1 to M* **do**
    Initialize input raw data x_1;
    Prepossess initial state: $S \leftarrow \phi(< x_1 >)$;
    for *time step:* $\tau \leftarrow 1$ *to* T_{max} **do**
        // 2. Generate training data:
        Select action A from state S using:
        $\pi \leftarrow \epsilon - Greedy(\hat{Q}(S, A, w))$;
        Take action A, Observe reward R and get next input $s_{\tau+1}$;
        Prepossessing next state: $S' \leftarrow \phi(s_{\tau+1})$;
        Store experience tuple (S, A, R, S') in replay memory D;
        $S' \leftarrow S$;
        // 3. Learning:
        Obtain random mini-batch of (s_j, a_j, r_j, s_{j+1}) from D;
        if *episode terminate at step j + 1* **then**
            Set target $\bar{F}_j \leftarrow r_j$;
        else
            Set target $\bar{F}_j \leftarrow r_j + \gamma \max_{a'} \hat{Q}(s_j, a, w^{-})$;
        Update: $w \leftarrow w + \alpha \nabla_{w_j} L(w_j)$ with *Adam*;
        Every $\mathcal{N}$ steps, update: $w^{-} \leftarrow w$;
    $\epsilon \leftarrow max(\epsilon_{end}, \epsilon * decay)$;
    Store score for current episode;

Loss Function & Update for Model Action Predictor: DQN[13,14]

- $L(W) = \left(r + \gamma \hat{Q}_{max,\pi}(s'_t, a'_t, W^-) - \hat{Q}_{\pi}(s_t, a_t, W)\right)^2$
- $\Rightarrow W = W - \alpha \frac{dL(W)}{dW}$.......(12)

Note: gradient is calculated on W,

W^- **is updated softly followed by W**



Local Output(s)= $\begin{vmatrix} Q_{left} \\ Q_{right} \\ Q_{up} \\ Q_{down} \end{vmatrix}$

Target Output(s')= $\begin{vmatrix} Q'_{left} \\ Q'_{right} \\ Q'_{up} \\ Q'_{down} \end{vmatrix}$

Why two Networks

- With one network
  - target estimation shifts every update
  - unstable feedback loop.

- With two networks
  - target estimation held constant for a while
  - smoother, more stable convergence of Q-values.

Tunable Parameters based on MDP

- Network architecture
- Learning Rate
- Batch size
- Target Network Update Parameters
- Discount Factor

Experience a hands-on project

How to setup

- 1. Go to Google Account Signup (https://accounts.google.com/signup)
- 2. Go to the Goggle Colab

  https://colab.research.google.com/
- 3. On the Colab homepage, click File → New Notebook.
- 4. In a cell code : print('Hello, DRL'), it will create new Folder as Colab Notebook.
- 5. Upload the shared folder to that folder.
- 6. Open script as Colab Laboratory

Key Recap and Takeaways

- RL = learning from experience
- DRL = Machines learning from experience
- Basics: agent, state, action, reward → learning through trial & error
- Key of the Design: Markov Decision Process

References

- [11] R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction,2nd ed. Cambridge, MA: MIT Press, 2018.
- [12] M. Riedmiller, ”Neural Fitted Q Iteration – First Experiences with aData Efficient Neural Reinforcement Learning Method,” in Proc. 16thEuropean Conf. Machine Learning (ECML 2005), pp. 317-328, 2005.

Thank You

Q & A

Reinforcement Learning

- ❖ learns from interactions with the environment to achieve a long-term goal
- ❖ The goal is defined by a reward signal, which must be maximized
- ❖ The agent must be able to partially or fully sense the environment state
- ❖ The state is typically represented as a feature vector.
- ❖ State transitions must be related with actions