Reinforcement Learning
Q-Learning

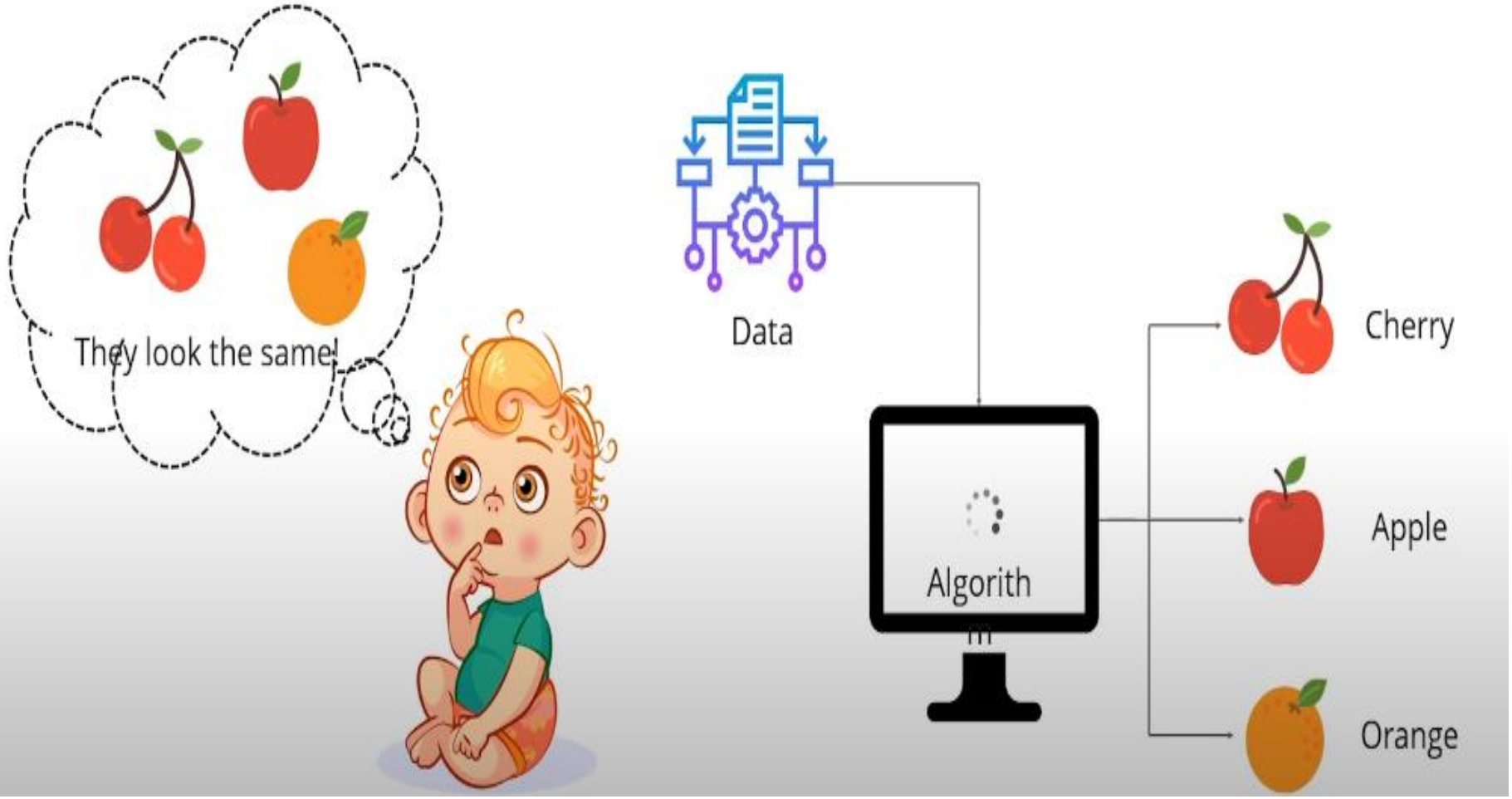
Outline

- Introduction
- Reinforcement Leaning
- RL model /MDP Model
- Learning Task
- Q-Learning Basic
- Q-Learning Algorithm
- Key word
- Reference

1. Introduction To Machine Learning
2. What is Reinforcement Learning?
3. Reinforcement Learning with an Analogy
4. Reinforcement Learning Process
5. Reinforcement Learning – Counter strike example
6. Reinforcement Learning Definitions
7. Reinforcement Learning Concepts
8. Markov's Decision Process
9. Understanding Q-Learning
10. [redacted]

What Is Machine Learning?

Machine learning is a subset of artificial intelligence (AI) which provides machines the ability to learn automatically & improve from experience without being explicitly programmed.



Types Of Machine Learning

Supervised Learning

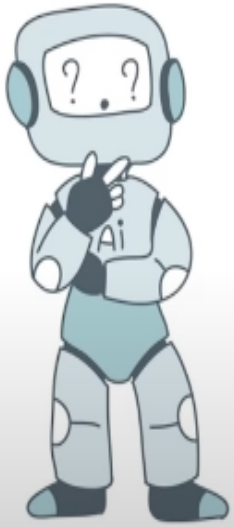
Unsupervised Learning

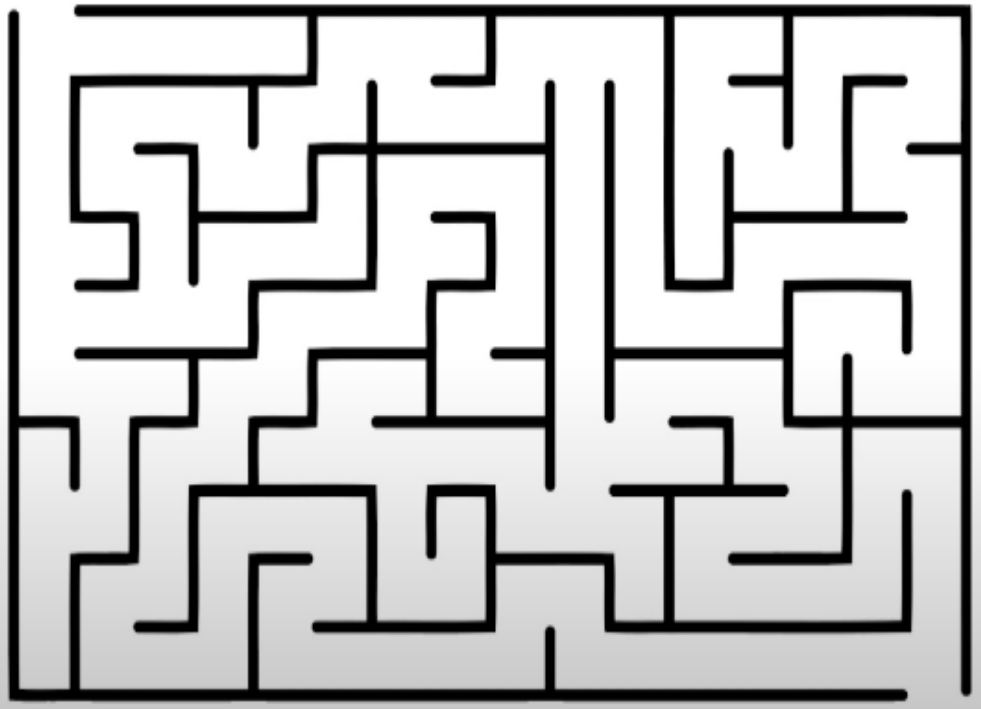
Reinforcement Learning

RL- Encourage or establish a pattern or behavior

What Is Reinforcement Learning?

Reinforcement learning is a type of Machine Learning where an agent learns to behave in a environment by performing actions and seeing the results

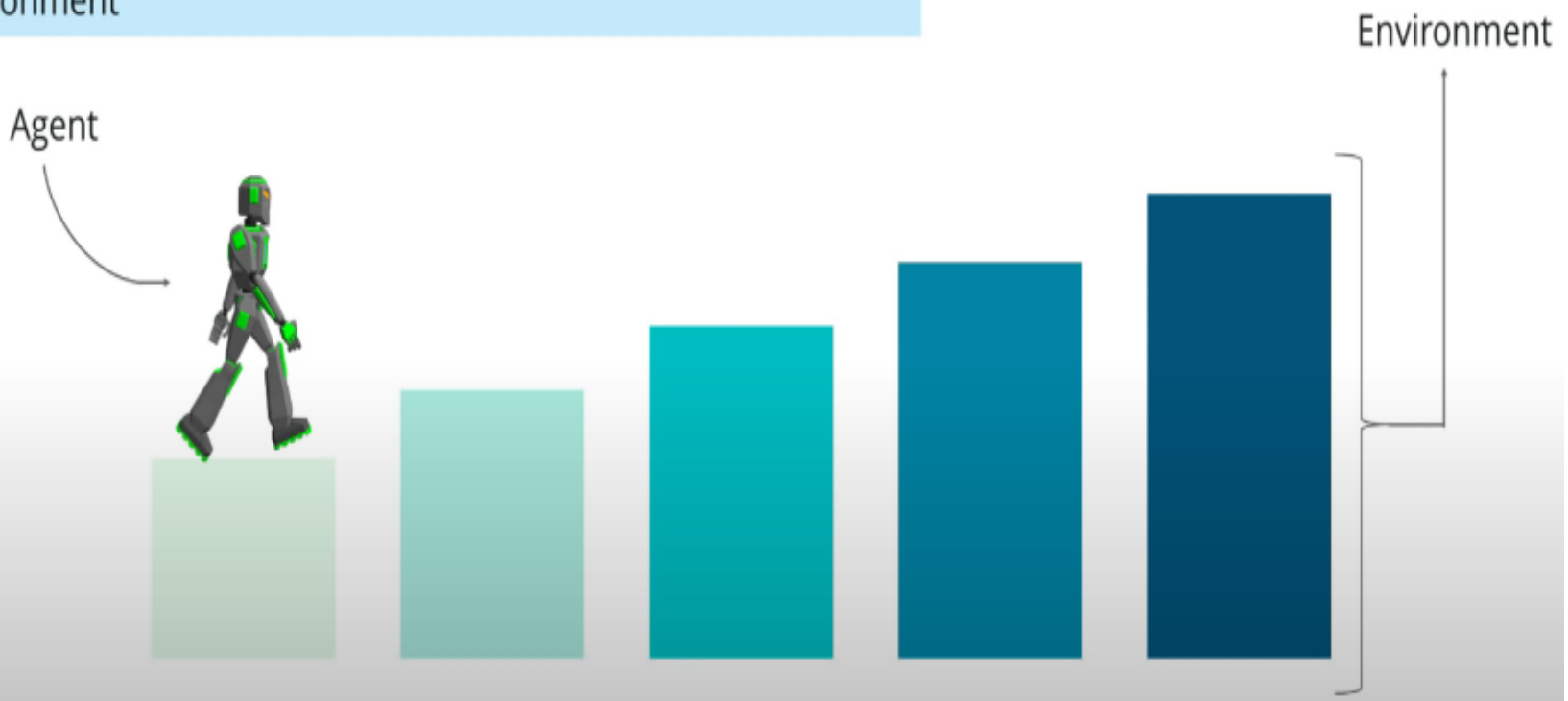




Reinforcement Learning Process

Reinforcement Learning system is comprised of two main components:

- Agent
- Environment



Reinforcement Learning Definitions

Agent: The RL algorithm that learns from trial and error

Environment: The world through which the agent moves

Action (A): All the possible steps that the agent can take

State (S): Current condition returned by the environment

Reinforcement Learning Definitions

Reward (R): An instant return from the environment to appraise the last action

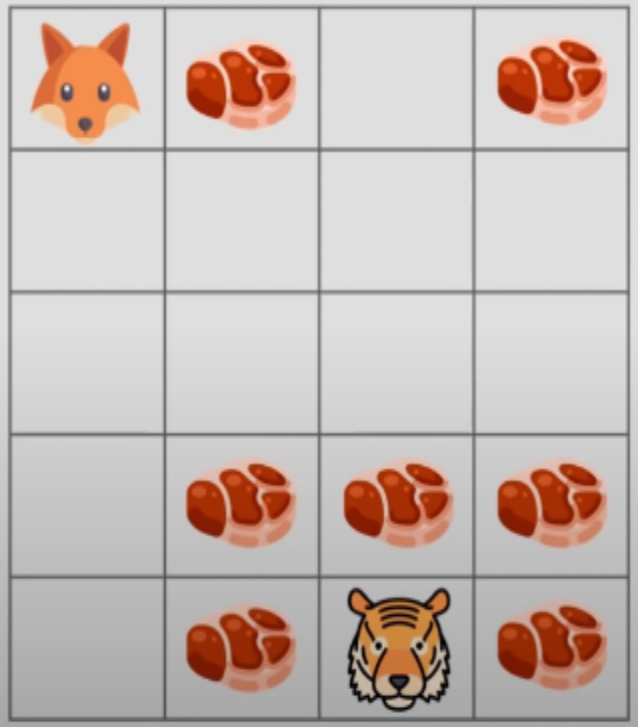
Policy (π): The approach that the agent uses to determine the next action based on the current state

Value (V): The expected long-term return with discount, as opposed to the short-term reward R

Action-value (Q): This similar to Value, except, it takes an extra parameter, the current action (A)

Reward Maximization

Reward maximization theory states that, *a RL agent must be trained in such a way that, he takes the best action so that the reward is maximum.*



Agent

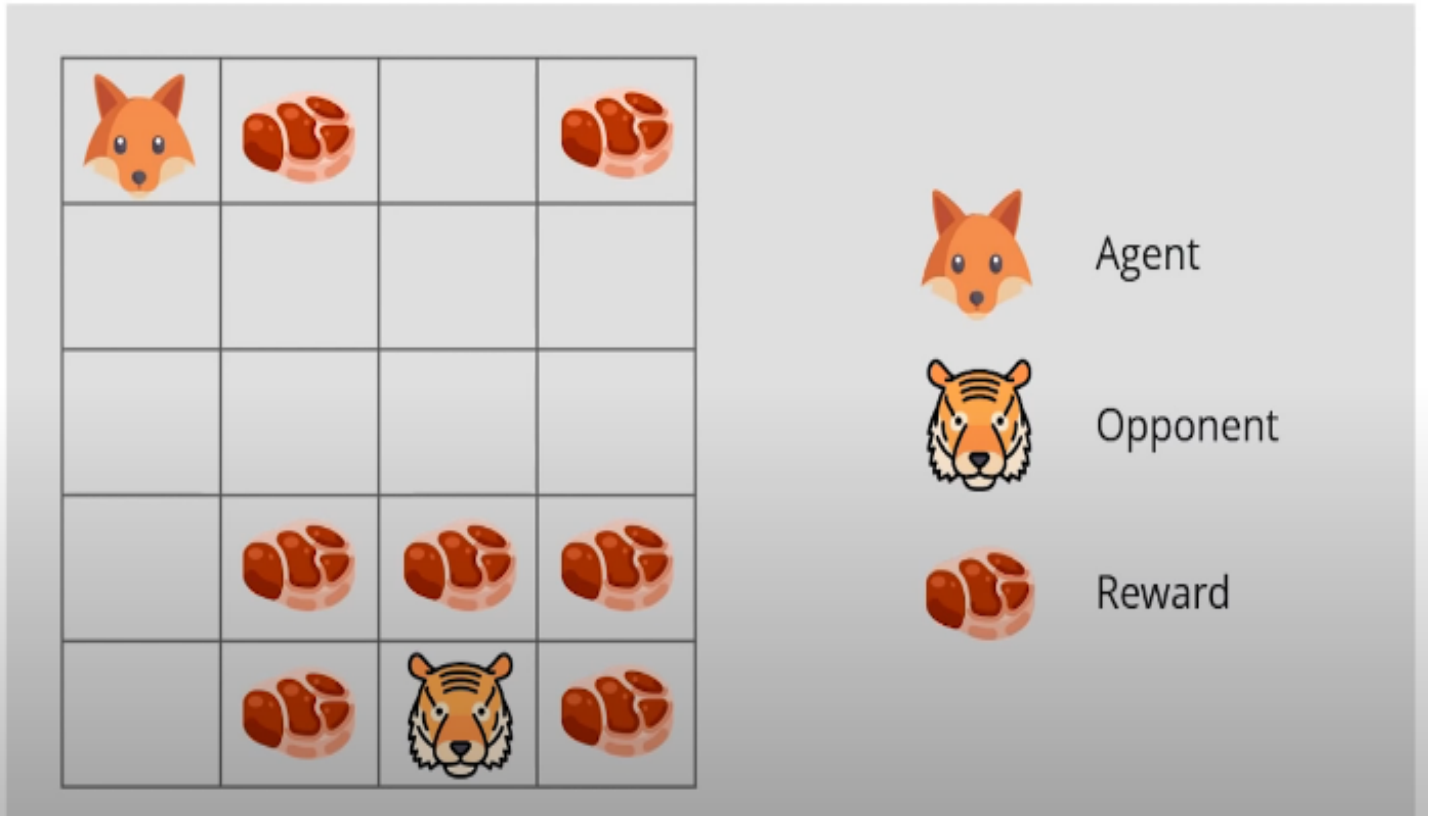
Opponent

Reward

Exploration & Exploitation

Exploitation is about using the already known exploited information to heighten the rewards

Exploration is about exploring and capturing more information about an environment

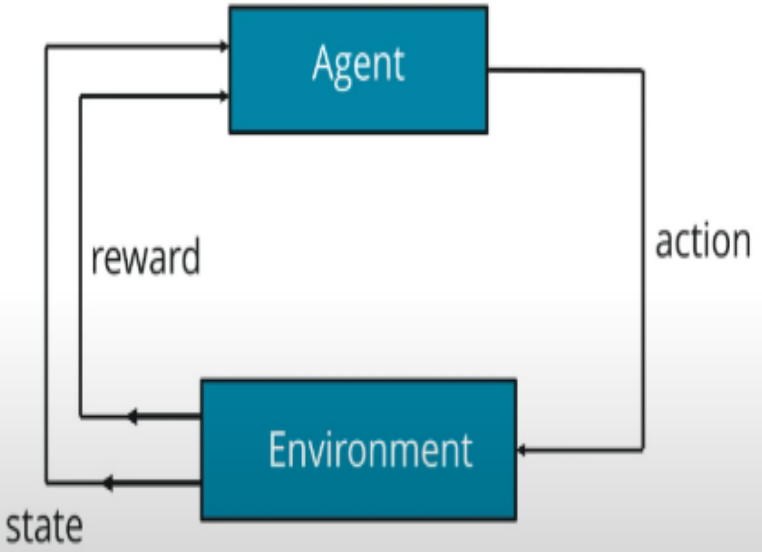


Markov Decision Process

The mathematical approach for mapping a solution in reinforcement learning is called *Markov Decision Process* (MDP)

The following parameters are used to attain a solution:

- Set of actions, A
- Set of states, S
- Reward, R
- Policy, π
- Value, V

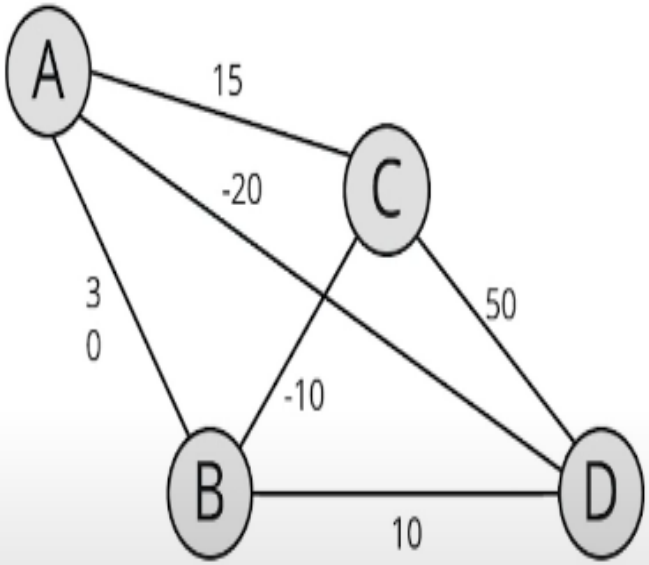


Markov Decision Process – Shortest Path Problem

Goal: Find the shortest path between A and D with minimum possible cost

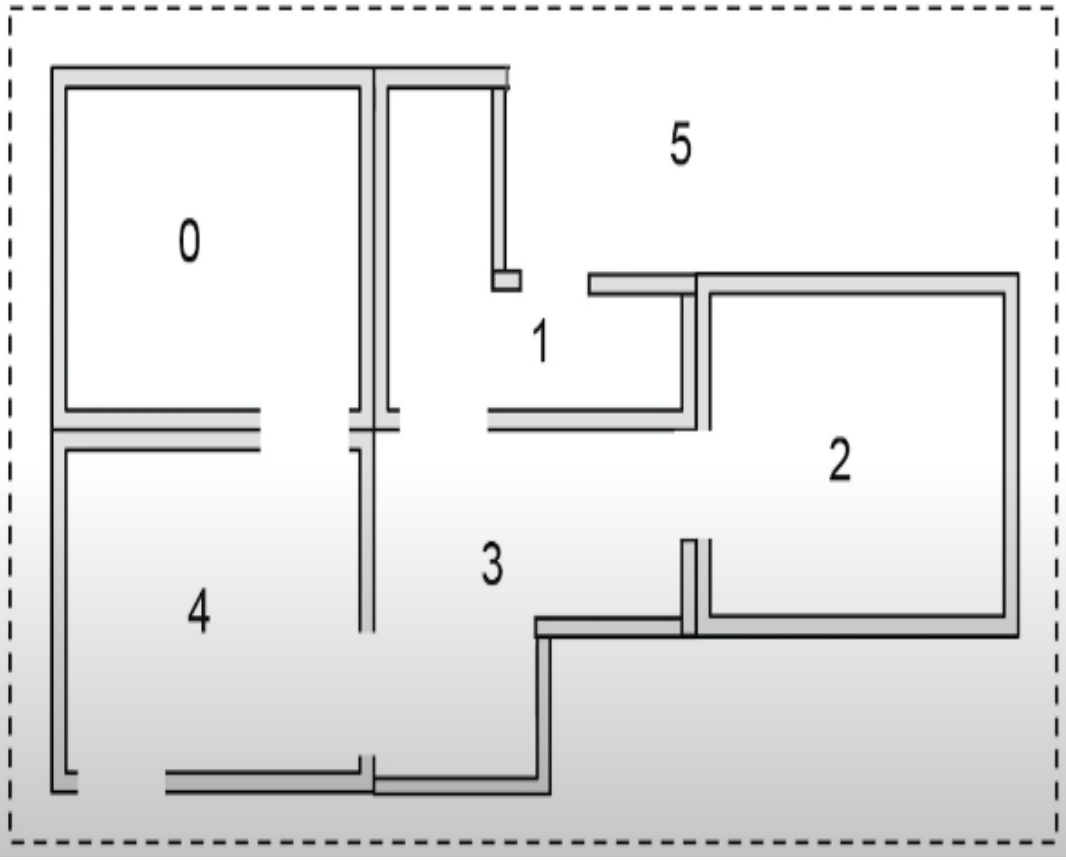
In this problem,

- Set of states are denoted by nodes i.e. {A, B, C, D}
- Action is to traverse from one node to another {A -> B, C -> D}
- Reward is the cost represented by each edge
- Policy is the path taken to reach the destination {A -> C -> D}



Understanding Q-Learning With An Example

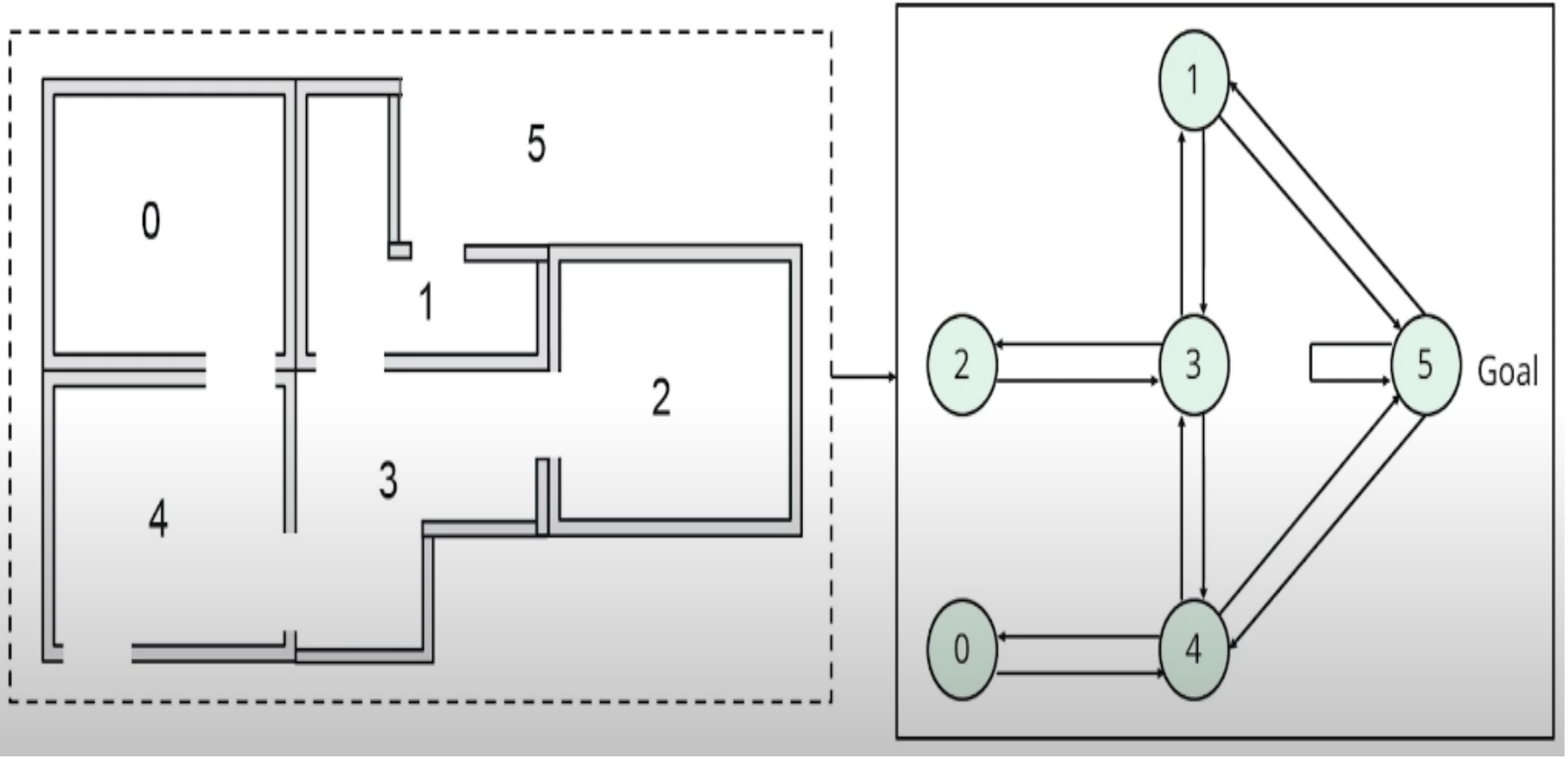
Place an agent in any one of the rooms (0,1,2,3,4) and the goal is to reach outside the building (room 5)



- 5 rooms in a building connected by doors
- each room is numbered 0 through 4
- The outside of the building can be thought of as one big room (5)
- Doors 1 and 4 lead into the building from room 5 (outside)

Understanding Q-Learning With An Example

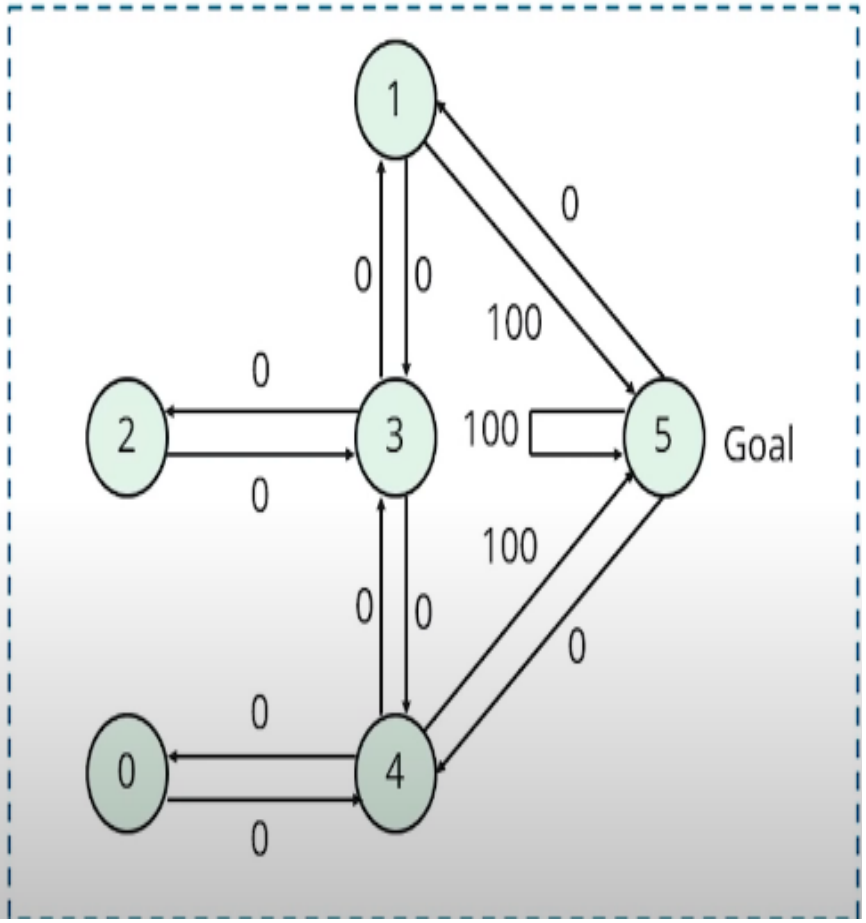
Let's represent the rooms on a graph, each room as a node, and each door as a link



Understanding Q-Learning With An Example

Next step is to associate a reward value to each door:

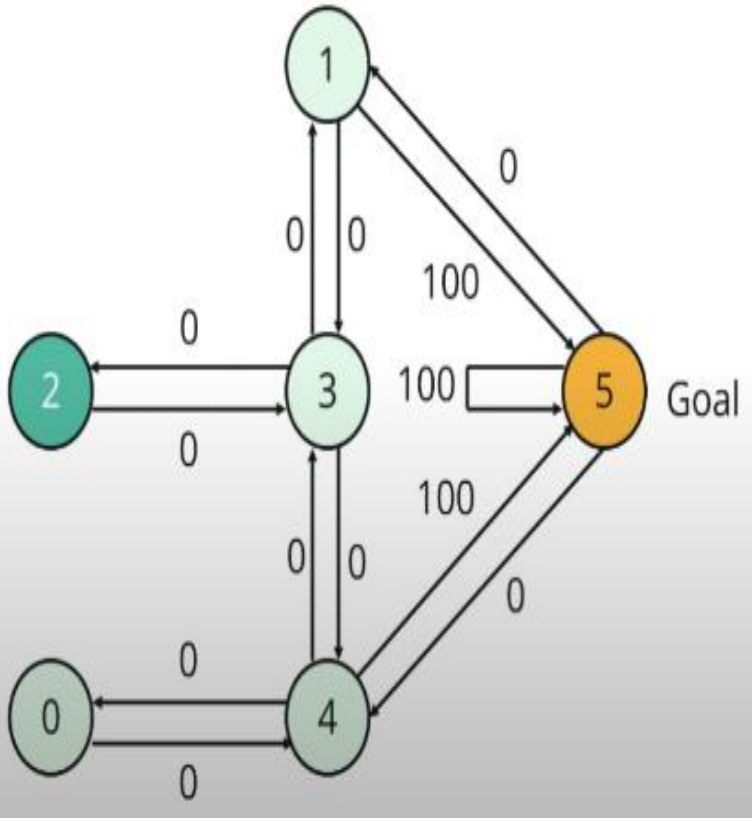
- doors that lead directly to the goal have a reward of 100
- Doors not directly connected to the target room have zero reward
- Because doors are two-way, two arrows are assigned to each room
- Each arrow contains an instant reward value



Understanding Q-Learning With An Example

The terminology in Q-Learning includes the terms state and action:

- Room (including room 5) represents a state
- agent's movement from one room to another represents an action
- In the figure, a state is depicted as a node, while "action" is represented by the arrows

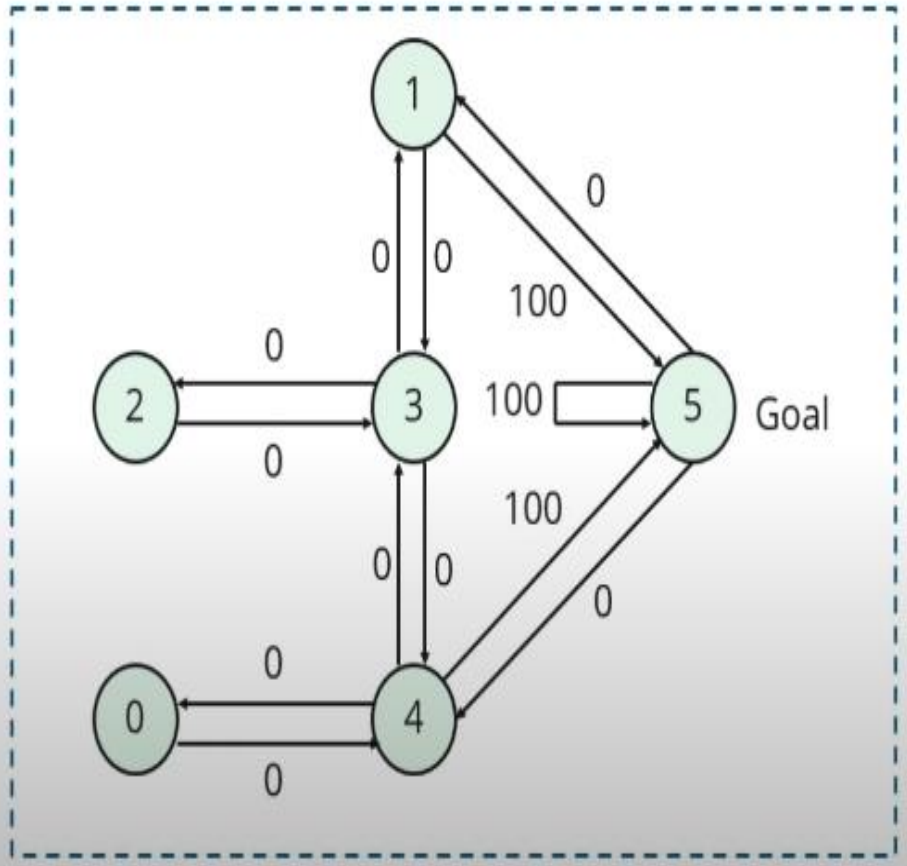


Example (Agent traverse from room 2 to room5):

1. Initial state = state 2
2. State 2 -> state 3
3. State 3 -> state (2, 1, 4)
4. State 4 -> state 5

Understanding Q-Learning With An Example

We can put the state diagram and the instant reward values into a reward table, matrix R.



$R =$

State \ Action	0	1	2	3	4	5
0	-1	-1	-1	-1	0	-1
1	-1	-1	-1	0	-1	100
2	-1	-1	-1	0	-1	-1
3	-1	0	0	-1	0	-1
4	0	-1	-1	0	-1	100
5	-1	0	-1	-1	0	100

The -1's in the table represent null values

Understanding Q-Learning With An Example

Add another matrix Q, representing the memory of what the agent has learned through experience.

- The rows of matrix Q represent the current state of the agent
- columns represent the possible actions leading to the next state
- Formula to calculate the Q matrix:

$$Q(state, action) = R(state, action) + Gamma * Max\ [Q(next\ state, all\ actions)]$$

Note

The Gamma parameter has a range of 0 to 1 (0 <= Gamma > 1).

- If Gamma is closer to zero, the agent will tend to consider only immediate rewards.
- If Gamma is closer to one, the agent will consider future rewards with greater weight

Q – Learning Example

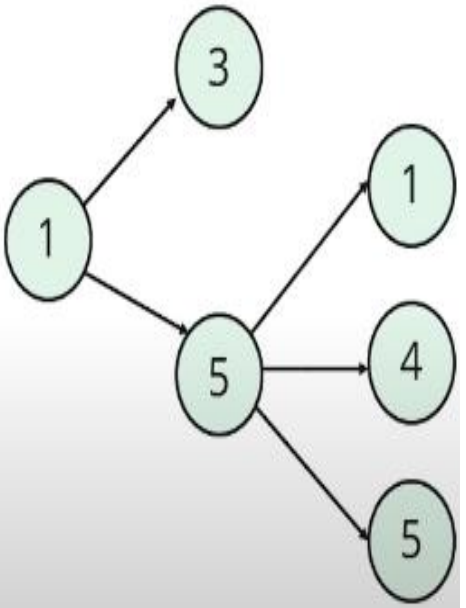
First step is to set the value of the learning parameter Gamma = 0.8, and the initial state as Room 1.

Next, initialize matrix Q as a zero matrix:

- From room 1 you can either go to room 3 or 5, let's select room 5.
- From room 5, calculate maximum Q value for this next state based on all possible actions:

*Q(state, action) = R(state, action) + Gamma * Max[Q(next state, all actions)]*

Q(1,5) = R(1,5) + 0.8 * Max[Q(5,1), Q(5,4), Q(5,5)] = 100 + 0.8 * 0 = 100



$Q =$

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	0	0	0	0	0	0
5	0	0	0	0	0	0

$R =$

State \ Action	0	1	2	3	4	5
0	-1	-1	-1	-1	0	-1
1	-1	-1	-1	0	-1	100
2	-1	-1	-1	0	-1	-1
3	-1	0	0	-1	0	-1
4	0	-1	-1	0	-1	100
5	-1	0	-1	-1	0	100

Q – Learning Algorithm

1. Set the gamma parameter, and environment rewards in matrix R
2. Initialize matrix Q to zero
3. Select a random initial state
4. Set initial state = current state
5. Select one among all possible actions for the current state
6. Using this possible action, consider going to the next state
7. Get maximum Q value for this next state based on all possible actions
8. Compute: Q(state, action) = R(state, action) + Gamma * Max[Q(next state, all actions)]
9. Repeat above steps until current state = goal state

Q – Learning Example

First step is to set the value of the learning parameter Gamma = 0.8, and the initial state as Room 1.

Next, initialize matrix Q as a zero matrix:

- From room 1 you can either go to room 3 or 5, let's select room 5.
- From room 5, calculate maximum Q value for this next state based on all possible actions:

*Q(state, action) = R(state, action) + Gamma * Max[Q(next state, all actions)]*

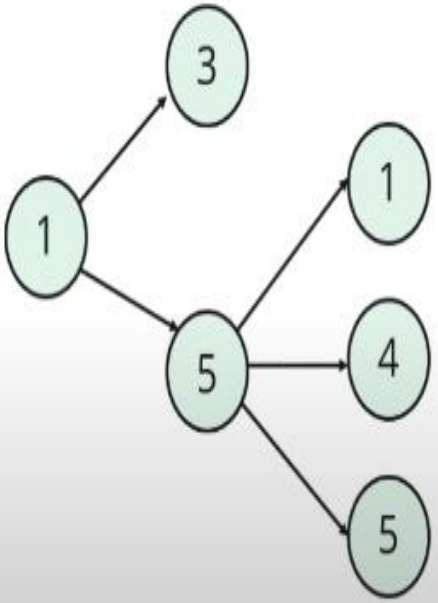
Q(1,5) = R(1,5) + 0.8 * Max[Q(5,1), Q(5,4), Q(5,5)] = 100 + 0.8 * 0 = 100

$Q =$

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	0	0	0	0	0	0
5	0	0	0	0	0	0

$R =$

State \ Action	0	1	2	3	4	5
0	-1	-1	-1	-1	0	-1
1	-1	-1	-1	0	-1	100
2	-1	-1	-1	0	-1	-1
3	-1	0	0	-1	0	-1
4	0	-1	-1	0	-1	100
5	-1	0	-1	-1	0	100



Q – Learning Example

For the next episode, we start with a randomly chosen initial state, i.e. state 3

- From room 3 you can either go to room 1,2 or 4, let's select room 1.
- From room 1, calculate maximum Q value for this next state based on all possible actions:

*Q(state, action) = R(state, action) + Gamma * Max[Q(next state, all actions)]*

Q(3,1) = R(3,1) + 0.8 * Max[Q(1,3), Q(1,5)]= 0 + 0.8 * [0, 100] = 80

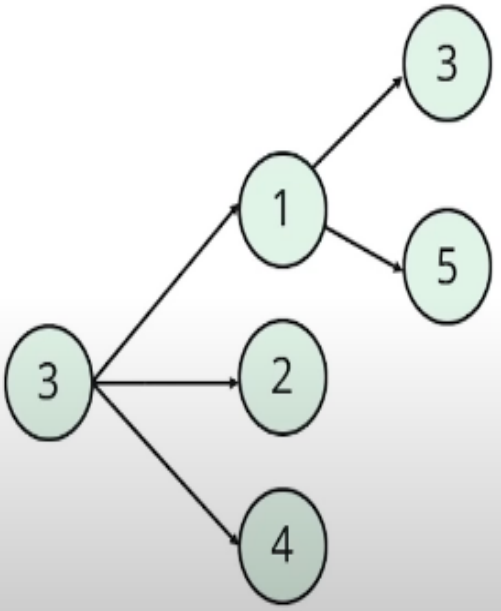
The matrix Q get's updated

$Q =$

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	0	100
2	0	0	0	0	0	0
3	0	80	0	0	0	0
4	0	0	0	0	0	0
5	0	0	0	0	0	0

$R =$

State \ Action	0	1	2	3	4	5
0	-1	-1	-1	-1	0	-1
1	-1	-1	-1	0	-1	100
2	-1	-1	-1	0	-1	-1
3	-1	0	0	-1	0	-1
4	0	-1	-1	0	-1	100
5	-1	0	-1	-1	0	100



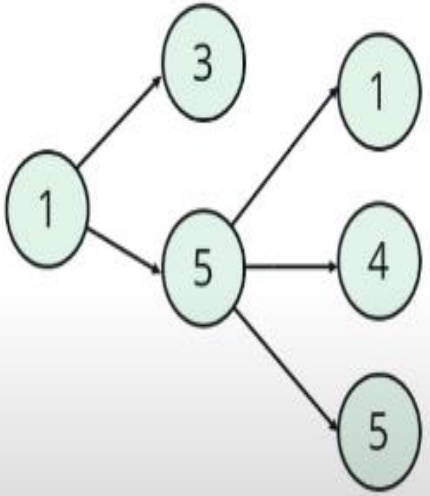
Q – Learning Example

For the next episode, the next state, 1, now becomes the current state. We repeat the inner loop of the Q learning algorithm because state 1 is not the goal state.

- From room 1 you can either go to room 3 or 5, let's select room 5.
- From room 5, calculate maximum Q value for this next state based on all possible actions:

*Q(state, action) = R(state, action) + Gamma * Max[Q(next state, all actions)]*

Q(1,5) = R(1,5) + 0.8 * Max[Q(5,1), Q(5,4), Q(5,5)] = 100 + 0.8 * 0 = 100
The matrix Q remains the same since, Q(1,5) is already fed to the agent



$Q =$

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	0	100
2	0	0	0	0	0	0
3	0	80	0	0	0	0
4	0	0	0	0	0	0
5	0	0	0	0	0	0

$R =$

State \ Action	0	1	2	3	4	5
0	-1	-1	-1	-1	0	-1
1	-1	-1	-1	0	-1	100
2	-1	-1	-1	0	-1	-1
3	-1	0	0	-1	0	-1
4	0	-1	-1	0	-1	100
5	-1	0	-1	-1	0	100

Q-Learning Basics

- At each step s, choose the action a which maximizes the function Q(s, a)
  - Q is the estimated utility function – it tells us how good an action is given a certain state
- Q(s, a) = immediate reward for making an action + best utility (Q) for the resulting state
- Note: recursive definition
- More formally ...

Formal Definition

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

$r(s,a)$ = Immediate reward

γ = relative value of delayed vs. immediate rewards (0 to 1)

s' = the new state after action a

a, a' : actions in states s and s', respectively

Selected action:

$$\pi(s) = \text{argmax}_a Q(s,a)$$

Q Learning Algorithm

For each state-action pair (s, a), initialize the table entry $\hat{Q}(s,a)$ to zero

Observe the current state s

Do forever:

---Select an action a and execute it

---Receive immediate reward r

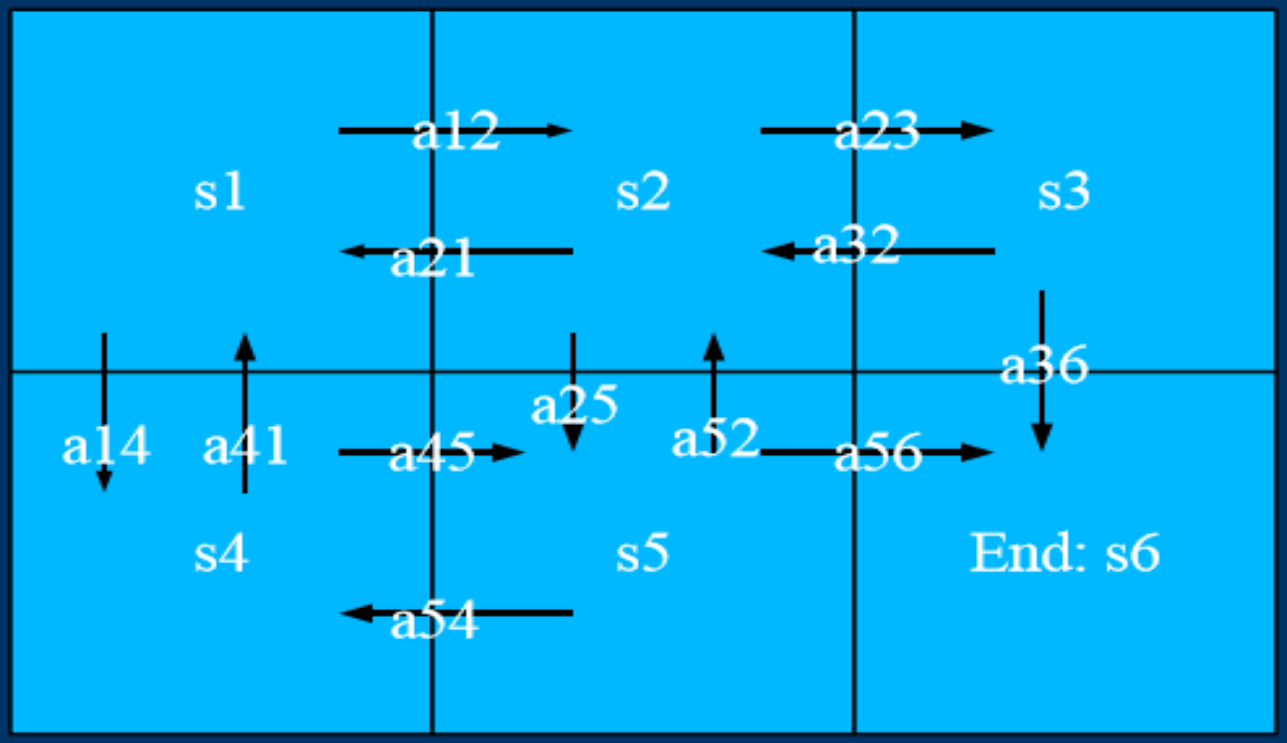
---Observe the new state s'

---Update the table entry for $\hat{Q}(s,a)$ as follows:

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

--- $s = s'$

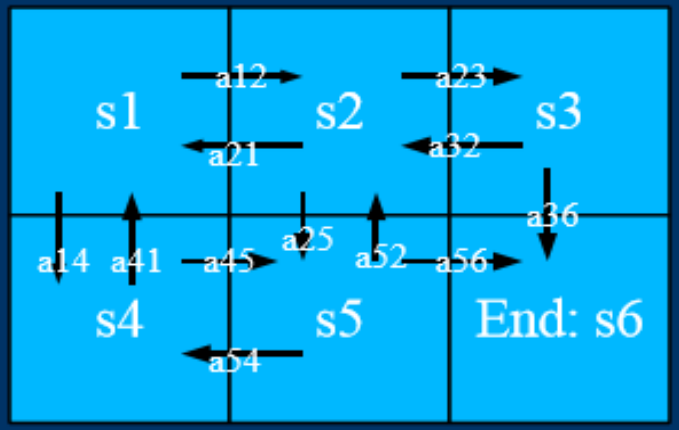
Example Problem



$\gamma = .5$, $r = 100$ if moving into state s6, 0 otherwise

Initial State

s1, a12	0
s1, a14	0
s2, a21	0
s2, a23	0
s2, a25	0
s3, a32	0
s3, a36	0
s4, a41	0
s4, a45	0
s5, a54	0
s5, a52	0
s5, a56	0

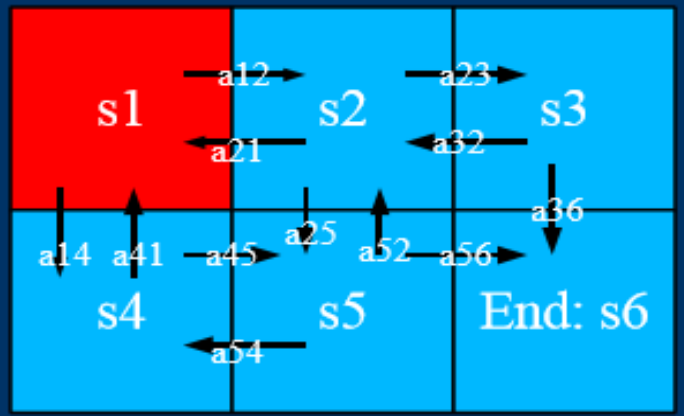


The Algorithm

s1, a12	0
s1, a14	0
s2, a21	0
s2, a23	0
s2, a25	0
s3, a32	0
s3, a36	0
s4, a41	0
s4, a45	0
s5, a54	0
s5, a52	0

Current Position: Red

Available actions: a12, a14
Chose a12



Update Q(s1, a12)

s1, a12	0
s1, a14	0
s2, a21	0
s2, a23	0
s2, a25	0
s3, a32	0
s3, a36	0
s4, a41	0
s4, a45	0
s5, a54	0
s5, a52	0

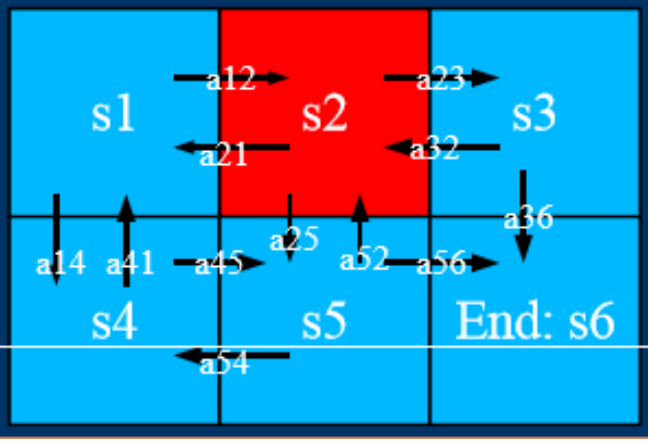
Current Position: Red

Available actions: a21, a25, a23

Update Q(s1, a12):

$Q(s1, a12) = r + .5 * \max(Q(s2,a21), Q(s2,a25), Q(s2,a23))$

$= 0$

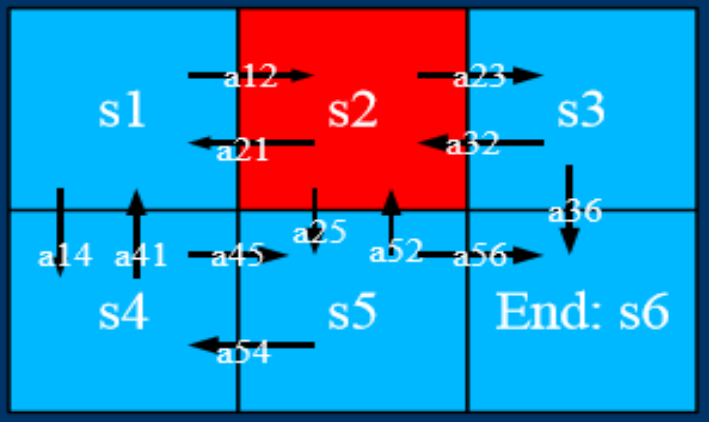


Next Move

s1, a12	0
s1, a14	0
s2, a21	0
s2, a23	0
s2, a25	0
s3, a32	0
s3, a36	0
s4, a41	0
s4, a45	0
s5, a54	0
s5, a52	0

Current Position: Red

Available actions: a21, a25, a23
Chose a23



Update Q(s2, a23)

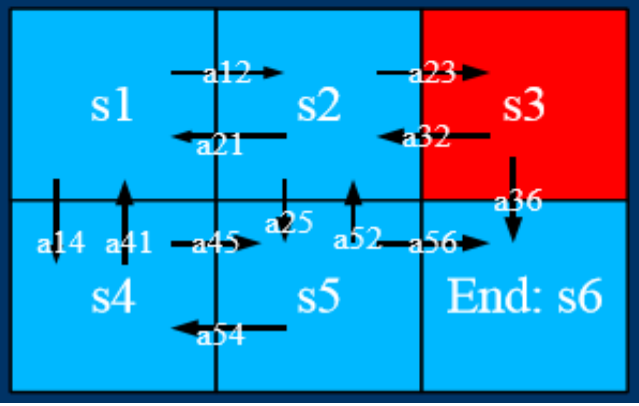
s1, a12	0
s1, a14	0
s2, a21	0
s2, a23	0
s2, a25	0
s3, a32	0
s3, a36	0
s4, a41	0
s4, a45	0
s5, a54	0
s5, a52	0

Current Position: Red

Available actions: a32, a36

Update Q(s1, a12):
Q(s2, a23) = r + .5 * max(Q(s3,a32), Q(s3,a36))

= 0

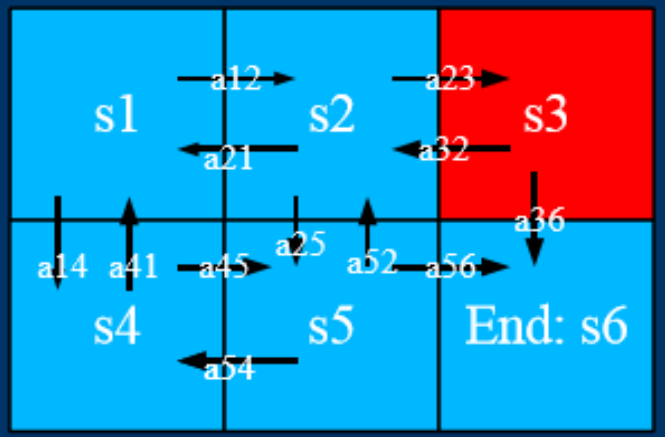


Next Move

s1, a12	0
s1, a14	0
s2, a21	0
s2, a23	0
s2, a25	0
s3, a32	0
s3, a36	0
s4, a41	0
s4, a45	0
s5, a54	0
s5, a52	0

Current Position: Red

Available actions: a32, a36
Chose a36



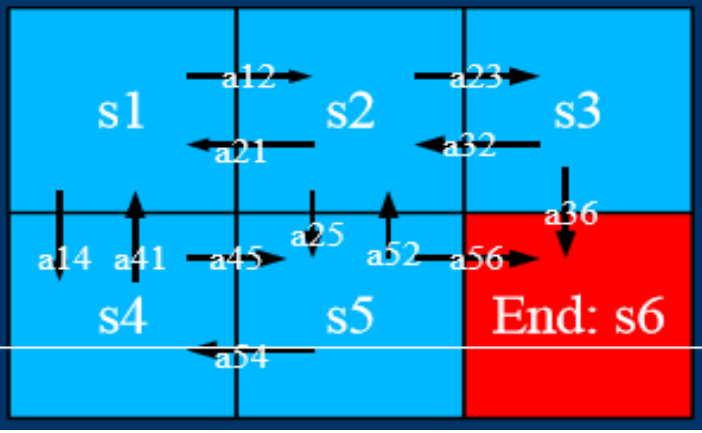
Update Q(s3,a36)

s1, a12	0
s1, a14	0
s2, a21	0
s2, a23	0
s2, a25	0
s3, a32	0
s3, a36	100
s4, a41	0
s4, a45	0
s5, a54	0
s5, a52	0

Current Position: Red

FINAL STATE!

Update Q(s1, a12):
Q(s2, a23) = r = 100

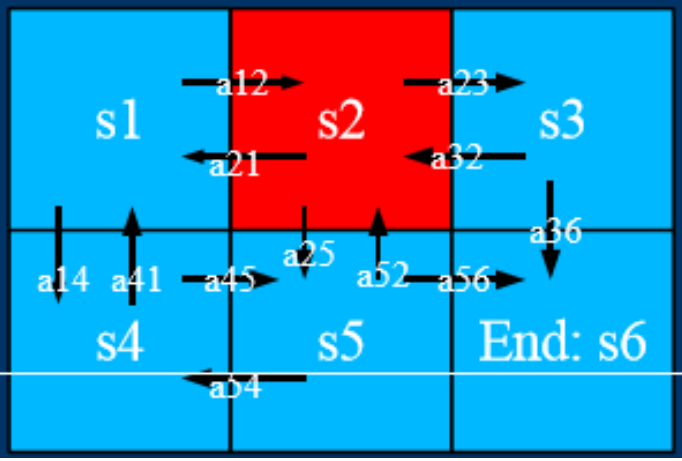


New Game

s1, a12	0
s1, a14	0
s2, a21	0
s2, a23	0
s2, a25	0
s3, a32	0
s3, a36	100
s4, a41	0
s4, a45	0
s5, a54	0
s5, a52	0

Current Position: Red

Available actions: a21, a25, a23
Chose a23



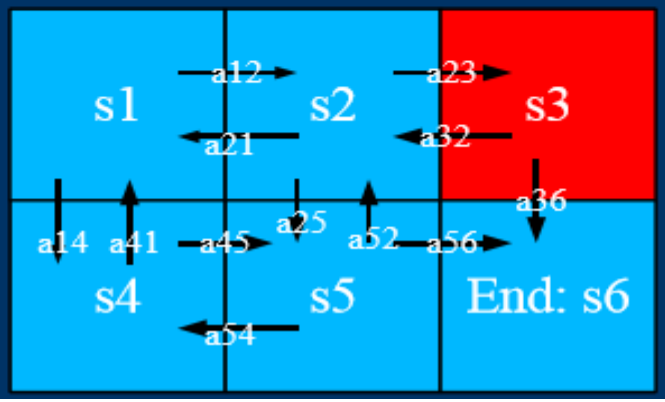
Update Q(s2, a23)

s1, a12	0
s1, a14	0
s2, a21	0
s2, a23	50
s2, a25	0
s3, a32	0
s3, a36	100
s4, a41	0
s4, a45	0
s5, a54	0
s5, a52	0

Current Position: Red

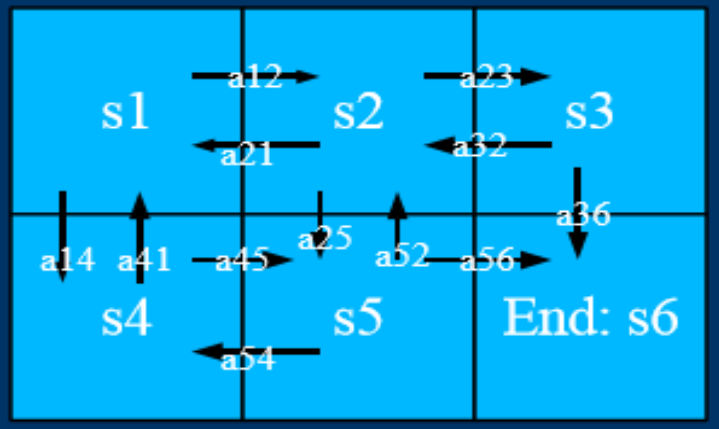
Available actions: a32, a36

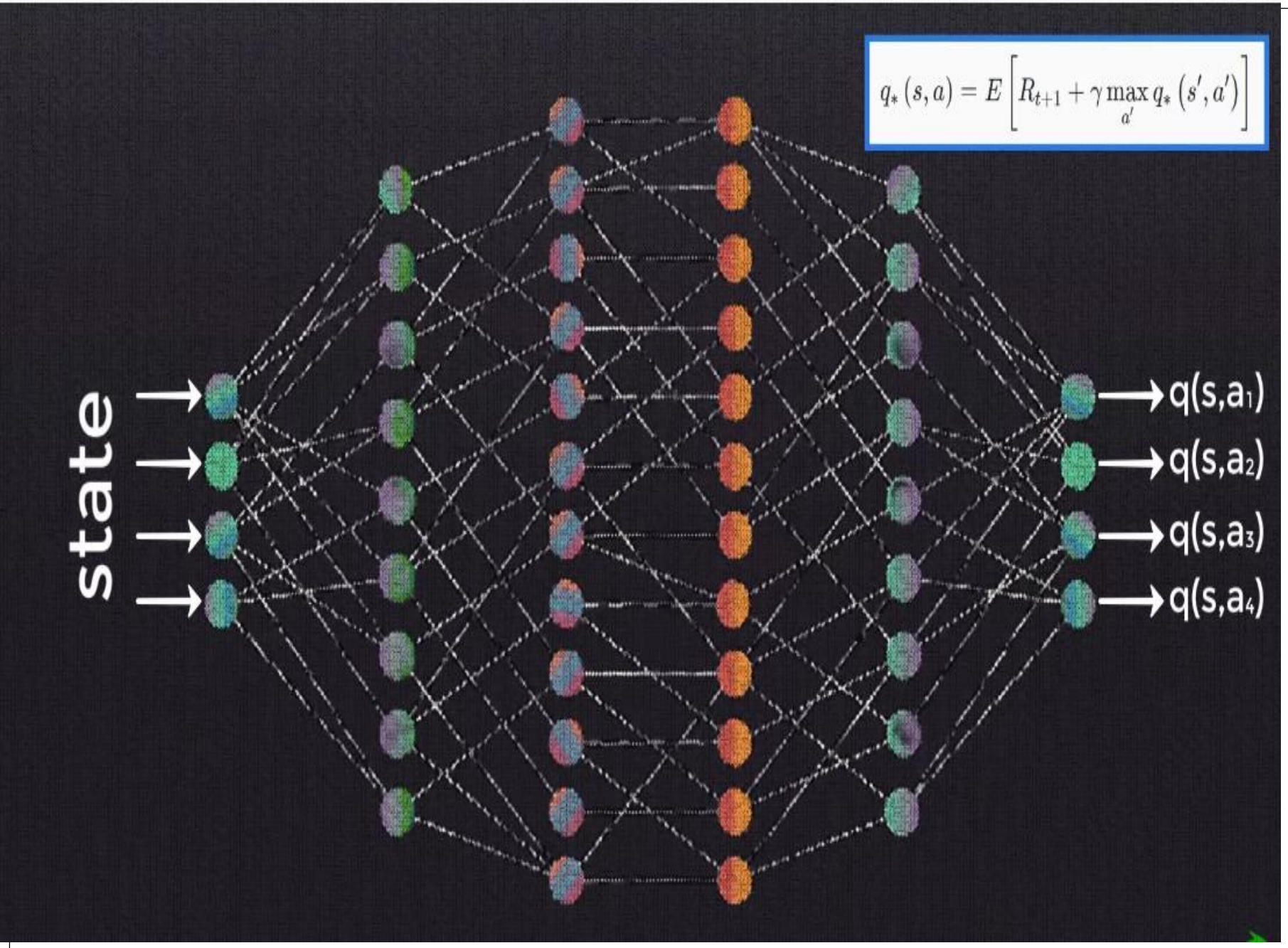
Update Q(s1, a12):
Q(s2, a23) = r + .5 * max(Q(s3,a32), Q(s3,a36))
= 0 + .5 * 100 = 50

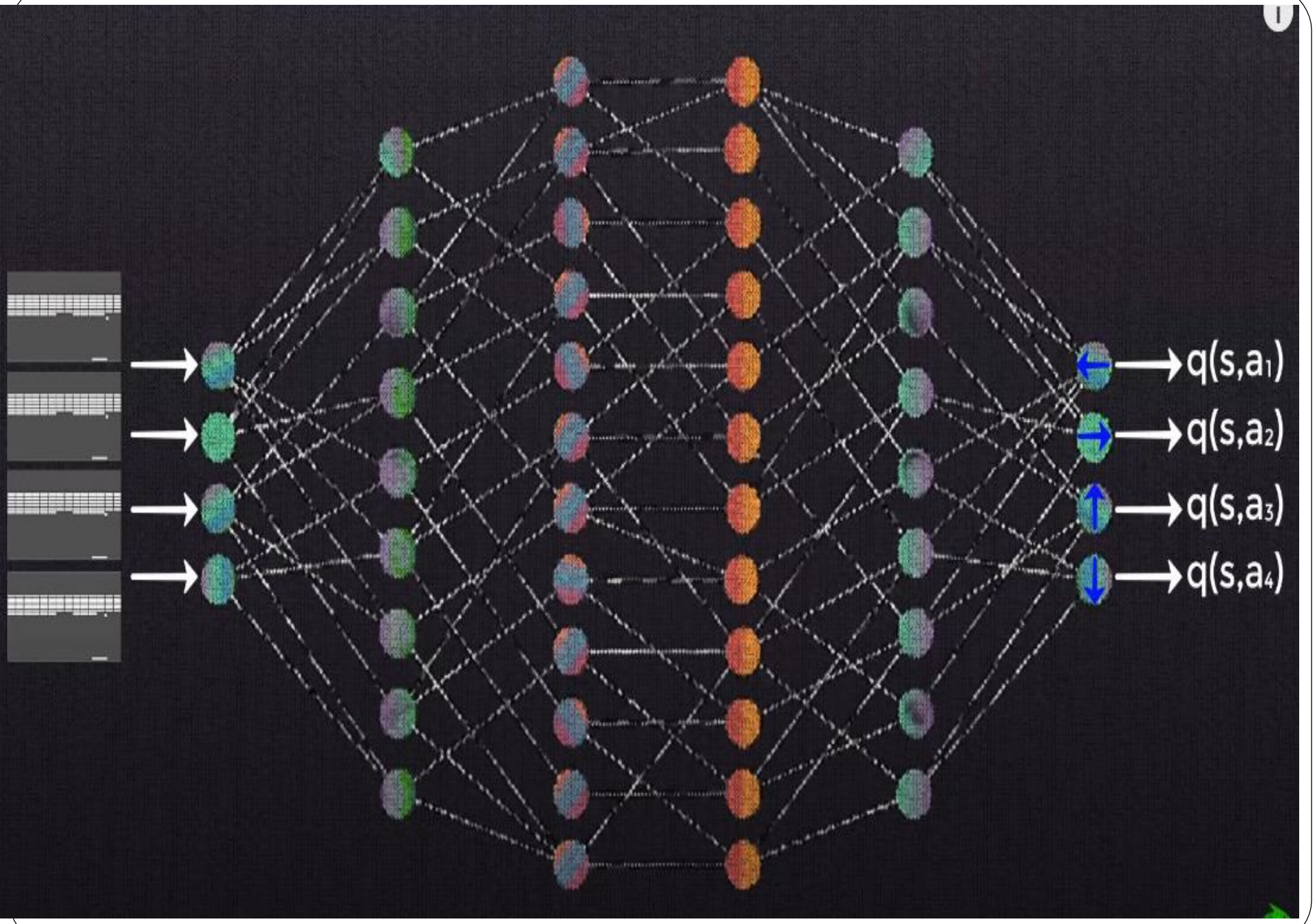


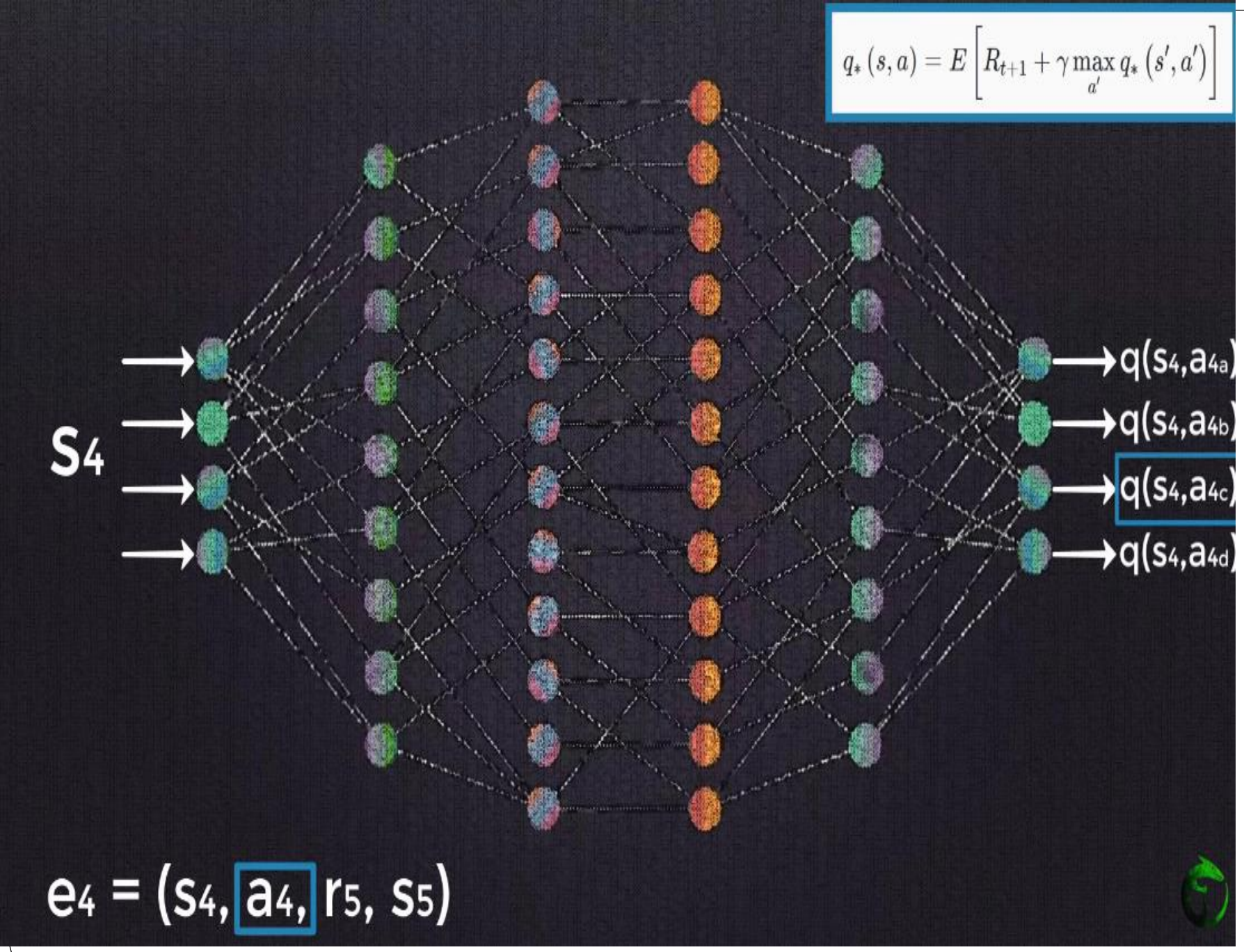
Final State (after many iterations)

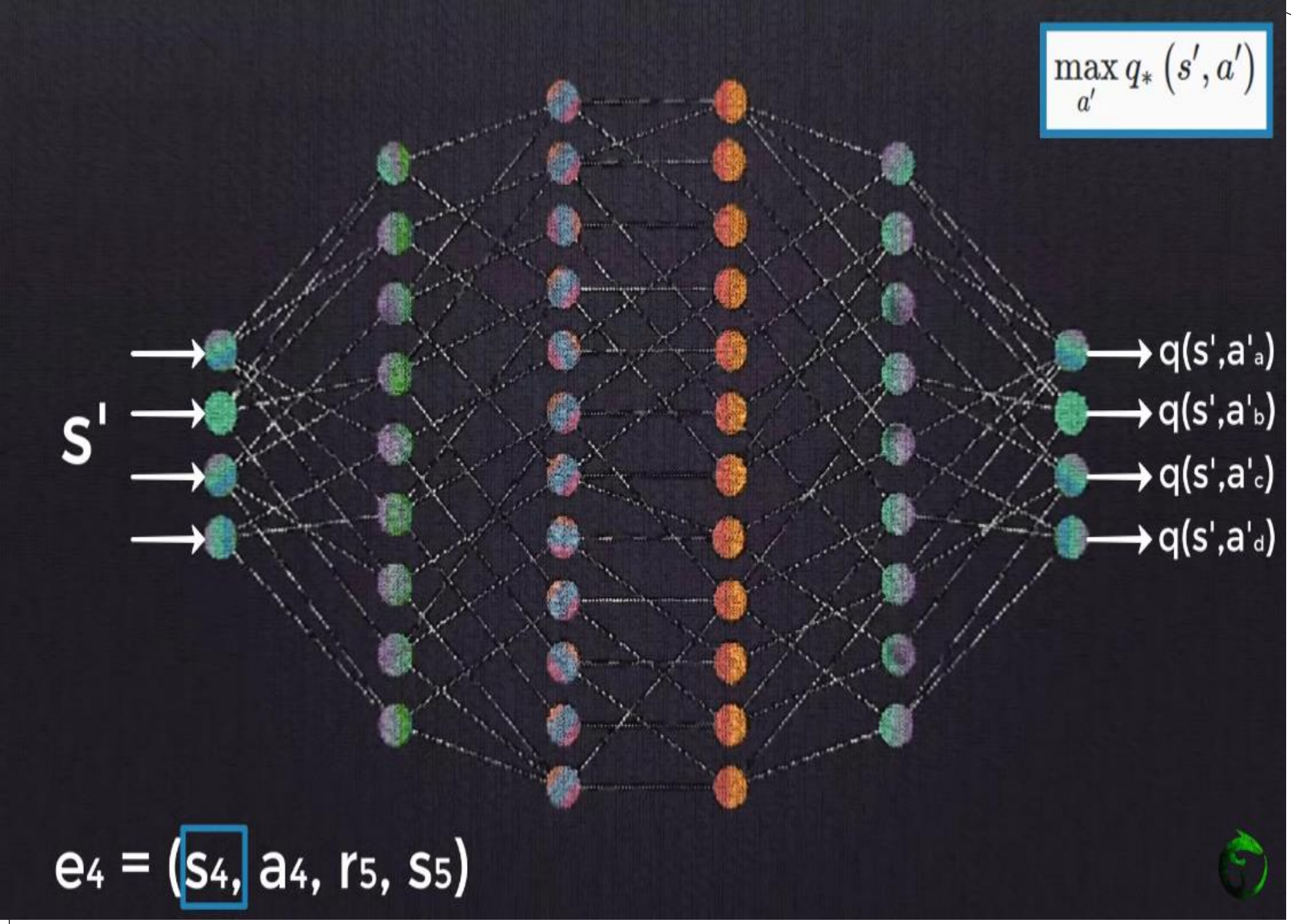
s1, a12	25
s1, a14	25
s2, a21	12.5
s2, a23	50
s2, a25	25
s3, a32	25
s3, a36	100
s4, a41	12.5
s4, a45	50
s5, a54	25
s5, a52	25
s5, a56	100




$q_*(s,a) = E\left[R_{t+1} + \gamma \max_{a'} q_*(s',a')\right]$
state
q(s,a1)
q(s,a2)
q(s,a3)
q(s,a4)


$q(s,a_1)$
$q(s,a_2)$
$q(s,a_3)$
$q(s,a_4)$


$q_*(s,a) = E\left[R_{t+1} + \gamma \max_{a'} q_*(s',a')\right]$
s_4
$q(s_4,a_{4a})$
$q(s_4,a_{4b})$
$q(s_4,a_{4c})$
$q(s_4,a_{4d})$
$e_4 = (s_4, a_4, r_5, s_5)$


$\max_{a'} q_*(s', a')$
s'
q(s',a'a)
q(s',a'b)
q(s',a'c)
q(s',a'd)
e4 = (s4, a4, r5, s5)

1. Initialize replay memory capacity.
2. Initialize the network with random weights.
3. *For each episode:*
    1. Initialize the starting state.
    2. *For each time step:*
        1. Select an action.
            - *Via exploration or exploitation*
        2. Execute selected action in an emulator.
        3. Observe reward and next state.
        4. Store experience in replay memory.
        5. Sample random batch from replay memory.
        6. Preprocess states from batch.
        7. Pass batch of preprocessed states to policy network.
        8. Calculate loss between output Q-values and target Q-values.
            - Requires a second pass to the network for the next state
        9. Gradient descent updates weights in the policy network to minimize loss.

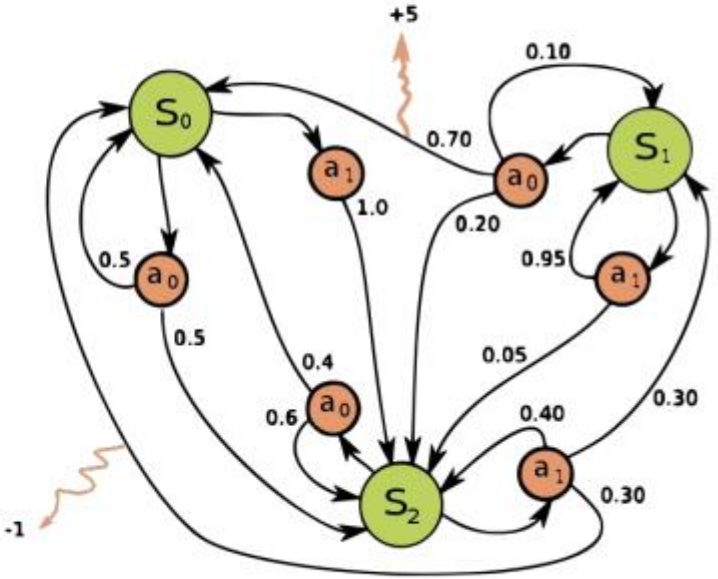
Outline

1. Motivation
2. Architecture
3. **Markov Decision Process (MDP)**
   - Policy
   - Optimal Policy
   - Value Function
   - Q-value function
   - Optimal Q-value function
   - Bellman equation
   - Value iteration algorithm
4. Deep Q-learning
5. RL Frameworks
6. Learn more

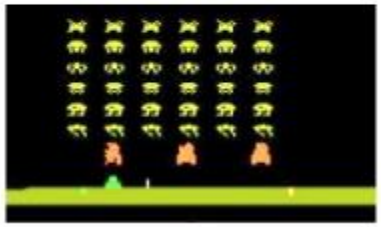
Markov Decisin Process (MDP)

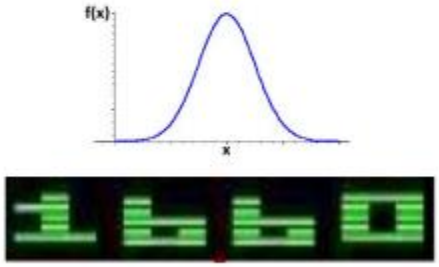
Markov Decision Processes provide a formalism for reinforcement learning problems.

Markov property:
Current state completely characterises the state of the world.



Markov Decision Process (MDP)



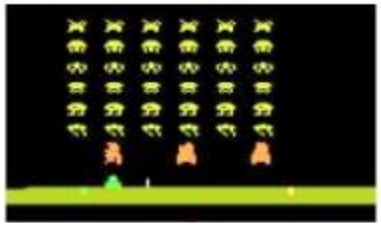


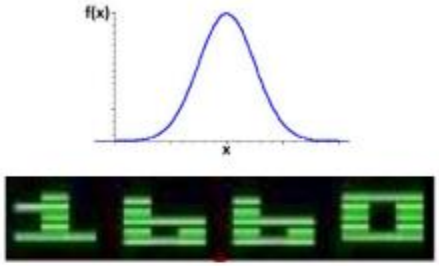
S A R P ɣ

Defined by: $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathbb{P}, \gamma)$

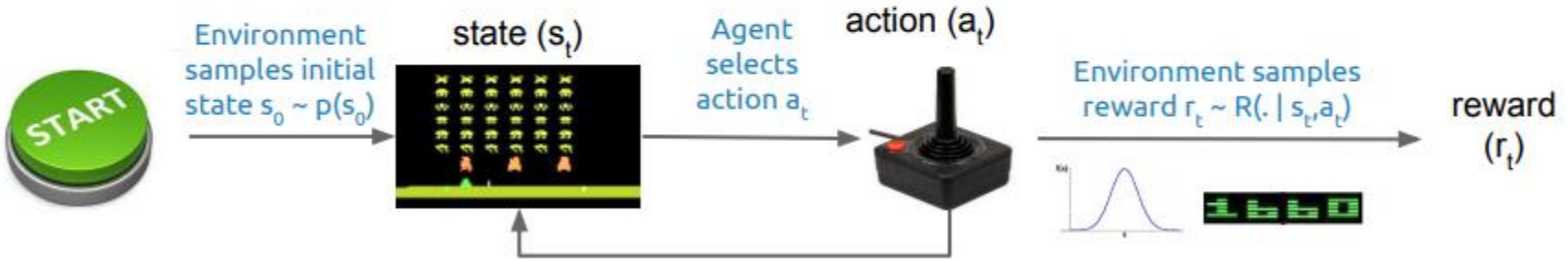
- $\mathcal{S}$: set of possible states
- $\mathcal{A}$: set of possible actions
- $\mathcal{R}$: distribution of reward given (state, action) pair
- $\mathbb{P}$: transition probability i.e. distribution over next state given (state, action) pair
- γ : discount factor

Markov Decision Process (MDP)





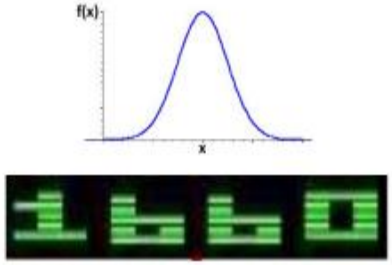
S A R P ɣ



MDP: Policy

S

A



R

P

γ

A **Policy π** is a function S → A that specifies which action to take in each state.

MDP: Policy

Agent

GOAL: Learn how to take **actions** to maximize **reward**

MDP

GOAL: Find **policy π*** that maximizes the cumulative discounted reward:

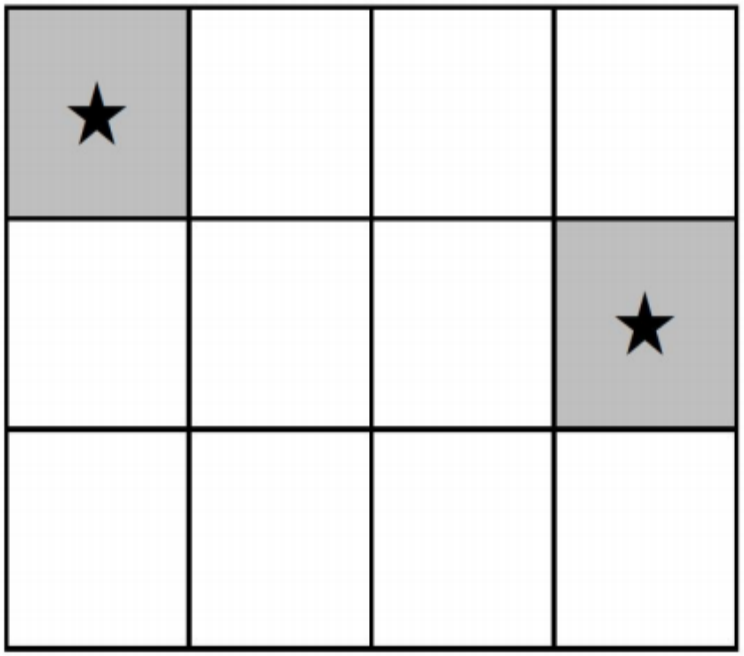
$$\sum_{t \geq 0} \gamma^t r_t$$

A **Policy π** is a function S → A that specifies which action to take in each state.

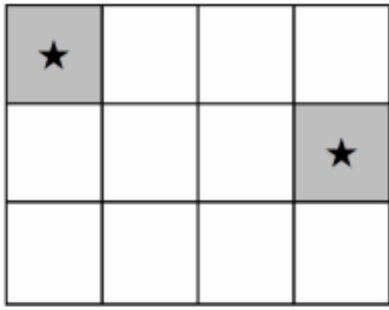
MDP: Policy

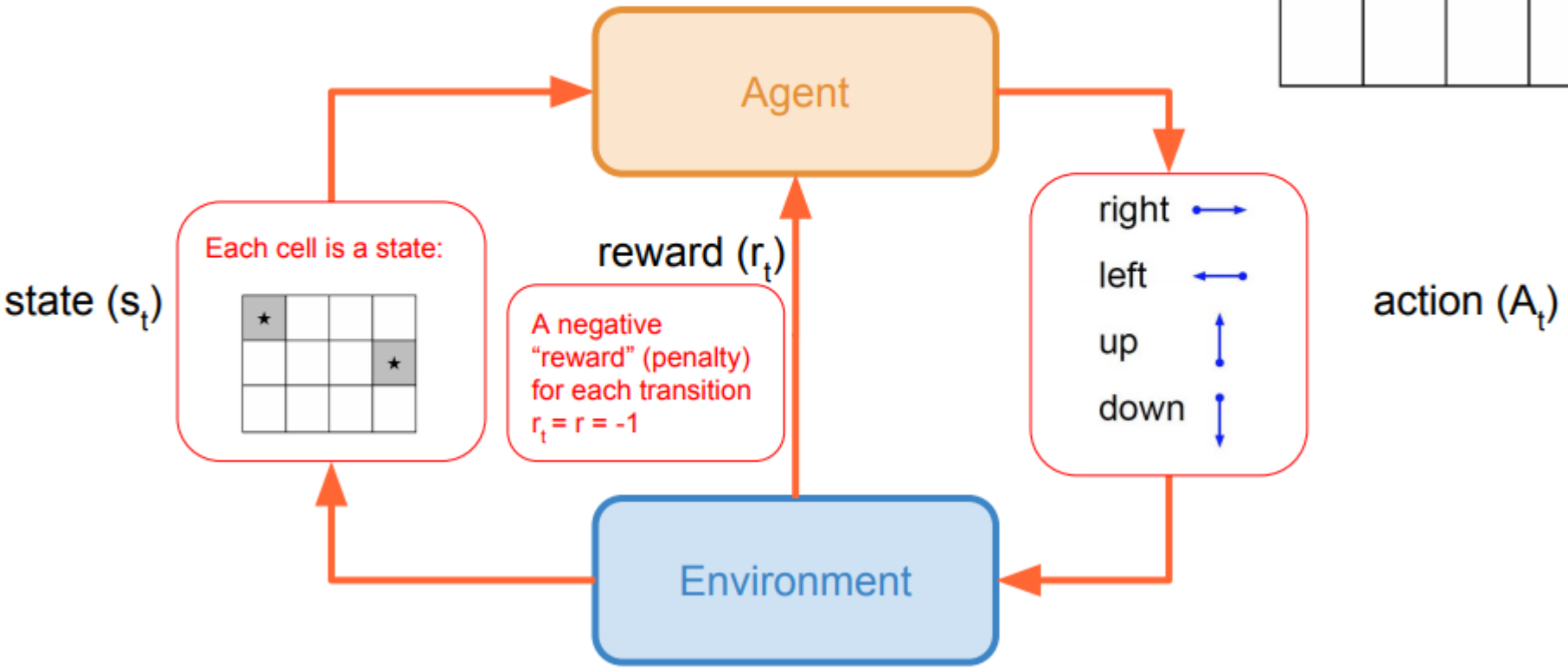
Multiple problems that can be formulated with a RL architecture.

Grid World (a simple MDP)
Objective: reach one of the terminal states (greyed out) in least number of actions.



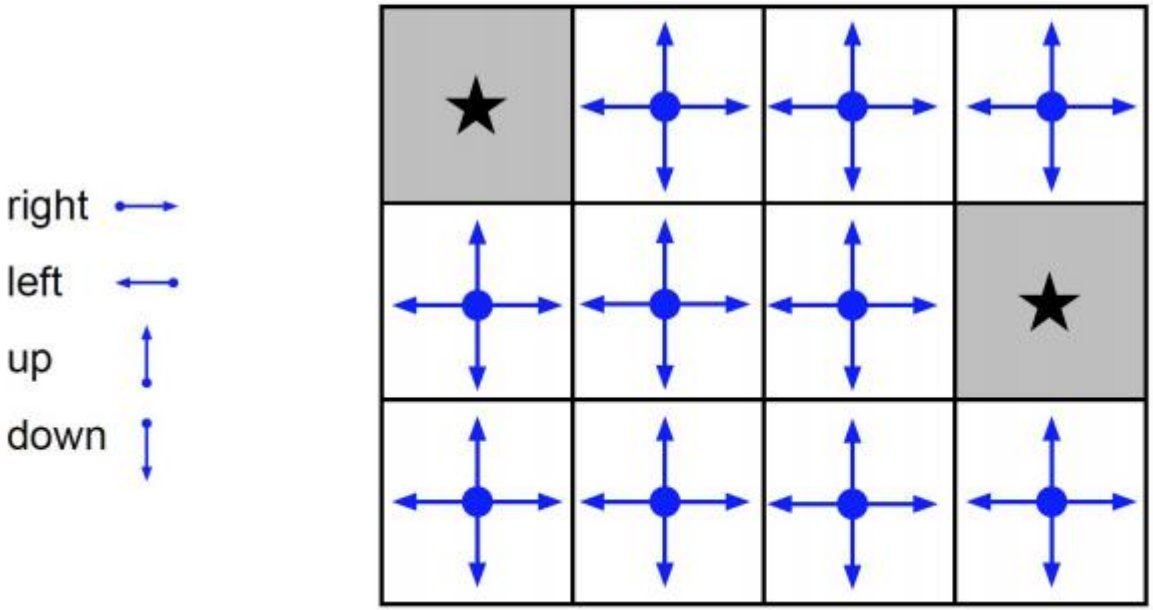
MDP: Policy




Agent
Each cell is a state:
state (s_t)
reward (r_t)
A negative "reward" (penalty) for each transition $r_t = r = -1$
right
left
up
down
action (A_t)
Environment

MDP: Policy: Random

Example: Actions resulting from applying a random policy on this Grid World problem.



MDP: Policy: Optimal Policy π*

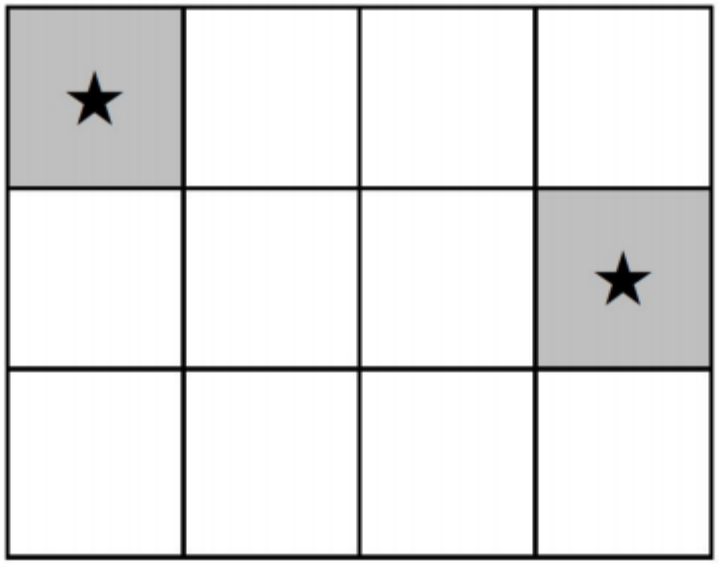
Exercise: Draw the actions resulting from applying an **optimal policy π*** in this Grid World problem.

right ⟷

left ⟷

up ↕

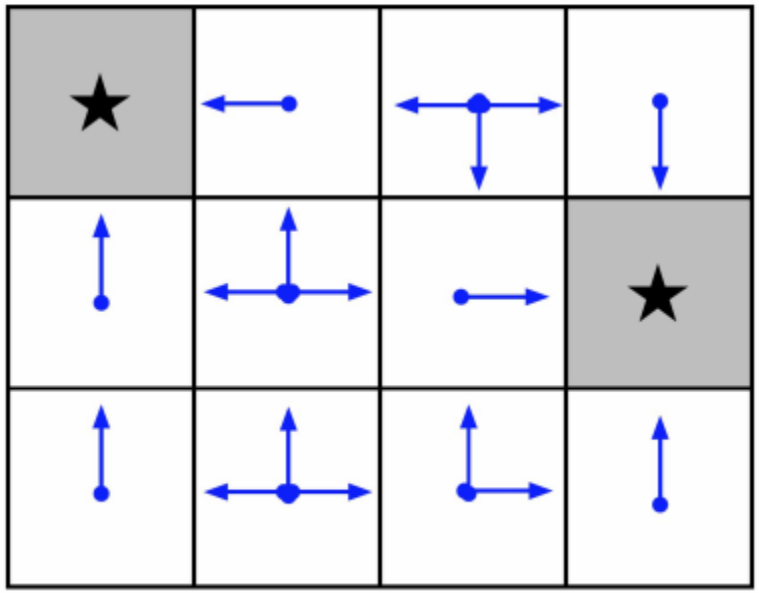
down ↕



MDP: Policy: Optimal Policy π*

Solution: Draw the actions resulting from applying an **optimal policy π*** in this Grid World problem.

right
left
up
down

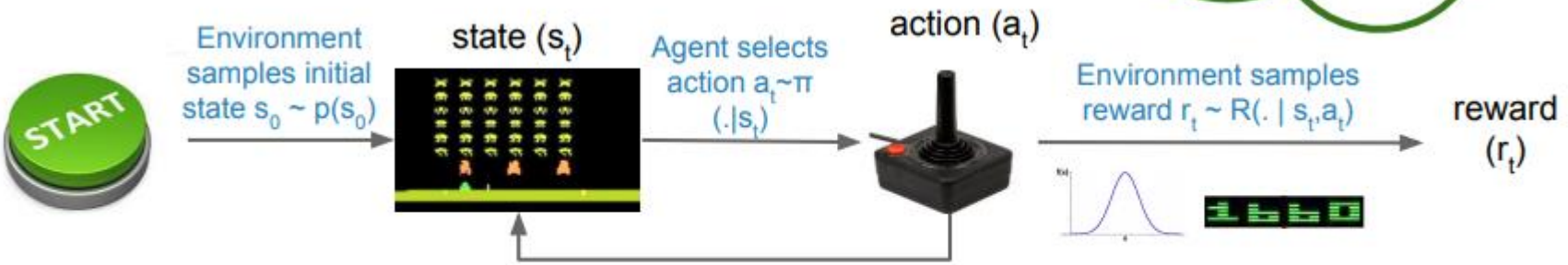


MDP: Policy: Optimal Policy π*

How do we handle the randomness (initial state s_0, transition probabilities, action...) ?

GOAL: Find **policy π*** that maximizes the cumulative discounted reward:

$$\sum_{t \geq 0} \gamma^t r_t$$



MDP: Policy: Optimal Policy π*

How do we handle the randomness (initial state s_0, transition probabilities, action) ?

GOAL: Find **policy π*** that maximizes the cumulative discounted reward:

$$\sum_{t \geq 0} \gamma^t r_t$$

The optimal policy π* will maximize the expected sum of rewards:

$$\pi^* = \arg\max_{\pi} \mathbb{E}\left[\sum_{t \geq 0} \gamma^t r_t | \pi\right] \quad \text{with} \quad s_0 \sim p(s_0), a_t \sim \pi(\cdot|s_t), s_{t+1} \sim p(\cdot|s_t, a_t)$$

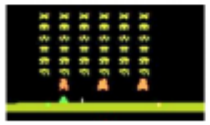
- expected cumulative discounted reward
- initial state
- selected action at t
- sampled state for t+1

MDP: Value Function $V^{\pi}(s)$

How to estimate how good state *s* is for a given policy π ?

With the **value function at state s, $V^{\pi}(s)$,** the expected cumulative reward from following policy π from state s.

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{t \geq 0} \gamma^t r_t | s_0 = s, \pi\right]$$



"Expected cumulative reward…""

"...from following policy π from state s."

MDP: Q-Value Function $Q^{\pi}(s,a)$

How to estimate how good a state-action pair (s,a) is for a given policy π?

With the **Q-value function at state *s* and action *a*, $Q^{\pi}(s,a)$,** the expected cumulative reward from taking action *a* in state *s*, and then following policy π.

$$Q^{\pi}(s,a) = \mathbb{E}\left[\sum_{t \geq 0} \gamma^t r_t | s_0 = s, a_0 = a, \pi\right]$$

"Expected cumulative reward…""

"...from taking action a in state s and then following policy π."

MDP: Optimal Q-Value Function Q*(s,a)

(From the previous page)
Q-value function

$$Q^{\pi}(s,a) = \mathbb{E}\left[\sum_{t\geq 0}\gamma^t r_t | s_0 = s, a_0 = a, \pi\right]$$

The optimal Q-value function at state s and action, $Q^*(s,a)$, is the **maximum expected cumulative reward** achievable from a given (state, action) pair:

$$Q^*(s,a) = \max_{\pi}\mathbb{E}\left[\sum_{t\geq 0}\gamma^t r_t | s_0 = s, a_0 = a, \pi\right]$$

choose the **policy** that maximizes the expected cumulative reward

MDP: Optimal Q-Value Function $Q^*(s,a)$

(From the previous page)
Optimal Q-value function

$$Q^*(s,a) = \max_{\pi} \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | s_0 = s, a_0 = a, \pi\right]$$

$Q^*(s,a)$ satisfies the following **Bellman equation:**

$$Q^*(s,a) = \mathbb{E}_{s'\sim\mathcal{E}}\left[r + \gamma \max_{a'} Q^*(s',a') | s,a\right]$$

- $Q^*(s,a)$: Maximum expected cumulative reward for considered pair (s,a)
- $\mathbb{E}_{s'\sim\mathcal{E}}$: Expectation across possible future states s' (randomness)
- r: reward for considered pair (s,a)
- γ: discount factor
- $\max_{a'} Q^*(s',a')$: Maximum expected cumulative reward for future pair (s',a') — FUTURE REWARD

MDP: Bellman Equation

GOAL: Find **policy π*** that maximizes the cumulative discounted reward:

$$\sum_{t \geq 0} \gamma^t r_t$$

$Q^*(s,a)$ satisfies the following **Bellman equation:**

$$Q^*(s,a) = \mathbb{E}_{s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q^*(s',a') | s,a \right]$$

select action a' that maximizes expected cumulative reward

The **optimal policy π*** corresponds to taking the best action in any state according to Q*.

MDP: Solving the Optimal Policy

(From the previous page)
Bellman Equation

$$Q^*(s,a) = \mathbb{E}_{s' \sim \mathcal{E}}\left[r + \gamma \max_{a'} Q^*(s',a') | s,a\right]$$

Value iteration algorithm: Estimate the Bellman equation with an iterative update.

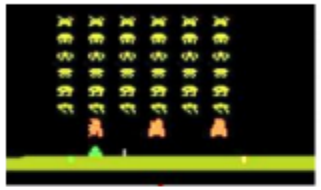
$$\underbrace{Q_{i+1}(s,a)}_{\text{Updated Q-value function}} = \mathbb{E}\left[r + \gamma \max_{a'} \underbrace{Q_i(s',a')}_{\text{Current Q-value for future pair (s',a')}} | s,a\right]$$

The iterative estimation $Q_i(s,a)$ will converge to the optimal $Q^*(s,a)$ as $i \rightarrow \infty$.

MDP: Solving the Optimal Policy

Exploring all positive states and action is **not scalable** by itself. Let alone iteratively.

Eg. If video game, it would require generating all possible pixels and actions… as many times as necessary iterations to estimate Q*(s,a).



$$\underbrace{Q_{i+1}(s,a)}_{\text{Updated Q-value function}} = \mathbb{E}\left[r + \gamma \max_{a'} \underbrace{Q_i(s',a')}_{\text{Current Q-value for future pair (s',a')}} | s,a\right]$$

The iterative estimation $Q_i(s,a)$ will converge to the optimal Q*(s,a) as $i \rightarrow \infty$.

MDP: Solving the Optimal Policy

Exploring all positive states and action is **not scalable**

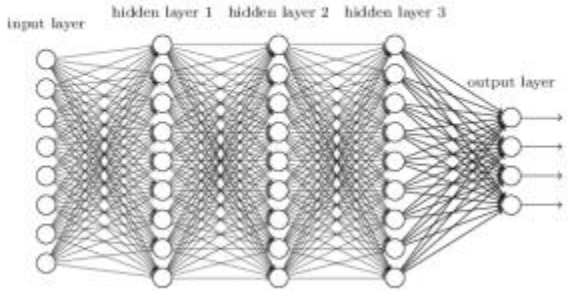
Eg. If video game, it would require generating all possible pixels and actions.

$$Q_{i+1}(s,a) = \mathbb{E}\left[r + \gamma \max_{a'} Q_i(s',a') | s,a\right]$$

Solution: Use a **deep neural network** as an function approximator of Q*(s,a).

$$Q(s,a,\Theta) \approx Q^*(s,a)$$

Neural Network parameters

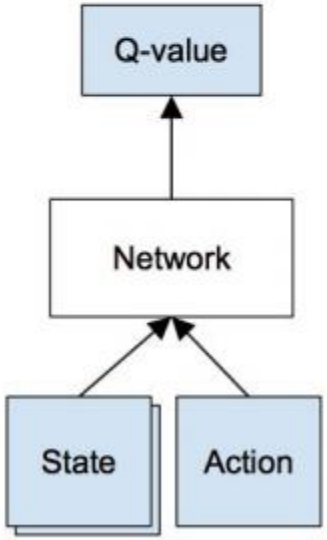


Deep Q-learning

The function to approximate is a Q-function that satisfies the Bellman equation:

$Q(s,a,\Theta) \approx Q^*(s,a)$

$$Q^*(s,a) = \mathbb{E}_{s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q^*(s',a') | s,a \right]$$



Deep Q-learning

The function to approximate is a Q-function that satisfies the Bellman equation:

$Q(s,a,\Theta) \approx Q^*(s,a)$

$$Q^*(s,a) = \mathbb{E}_{s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q^*(s',a') | s,a \right]$$

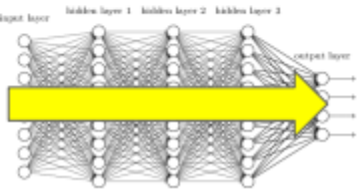
Forward Pass

Sample a (s,a) pair

Predicted Q-value with Θ_i

Loss function:

$$L_i(\theta_i) = \mathbb{E}_{s,a \sim \rho(\cdot)} \left[(y_i - Q(s,a;\theta_i))^2 \right]$$



$$y_i = \mathbb{E}_{s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q(s',a';\theta_{i-1}) | s,a \right]$$

Sample a future state s'

Predict Q-value with Θ_{i-1}

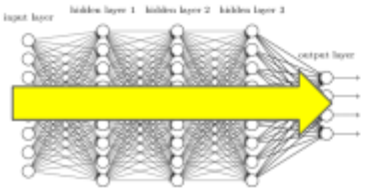
Deep Q-learning

Train the DNN to approximate a Q-value function that satisfies the Bellman equation

$$Q^*(s,a) = \mathbb{E}_{s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q^*(s',a') | s,a \right]$$

$$L_i(\theta_i) = \mathbb{E}_{s,a \sim \rho(\cdot)} \left[(y_i - Q(s,a;\theta_i))^2 \right]$$

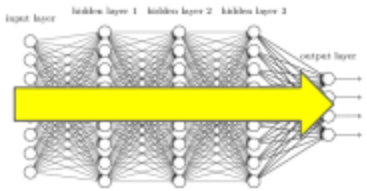
$$y_i = \mathbb{E}_{s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q(s',a';\theta_{i-1}) | s,a \right]$$



Deep Q-learning

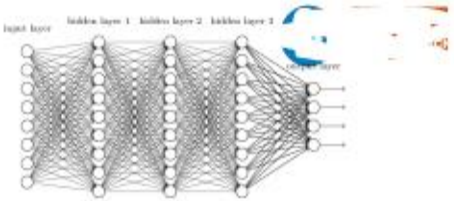
Must compute reward during training

$$L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(\cdot)}\left[(y_i - Q(s,a;\theta_i))^2\right]$$



$$y_i = \mathbb{E}_{s'\sim\mathcal{E}}\left[r + \gamma \max_{a'} Q(s',a';\theta_{i-1})|s,a\right]$$

Deep Q-learning



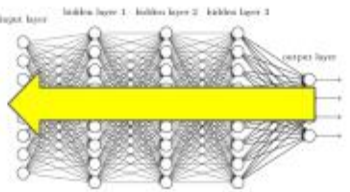
Forward Pass

Loss function: $$L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(\cdot)}\left[(y_i - Q(s,a;\theta_i))^2\right]$$

$$y_i = \mathbb{E}_{s'\sim\mathcal{E}}\left[r + \gamma \max_{a'} Q(s',a';\theta_{i-1})|s,a\right]$$

Backward Pass

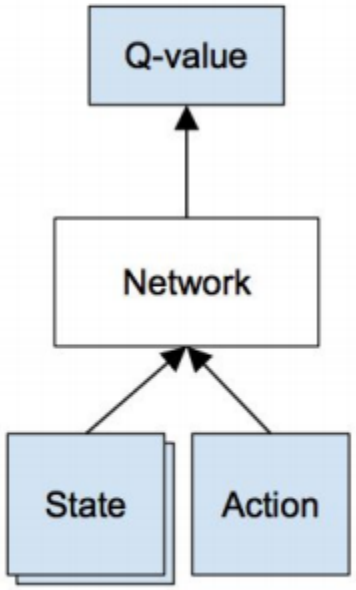
Gradient update (with respect to Q-function parameters Θ):



$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(\cdot);s'\sim\mathcal{E}}\left[r + \gamma \max_{a'} Q(s',a';\theta_{i-1}) - Q(s,a;\theta_i))\nabla_{\theta_i} Q(s,a;\theta_i)\right]$$

Deep Q-learning: Deep Q-Network DQN

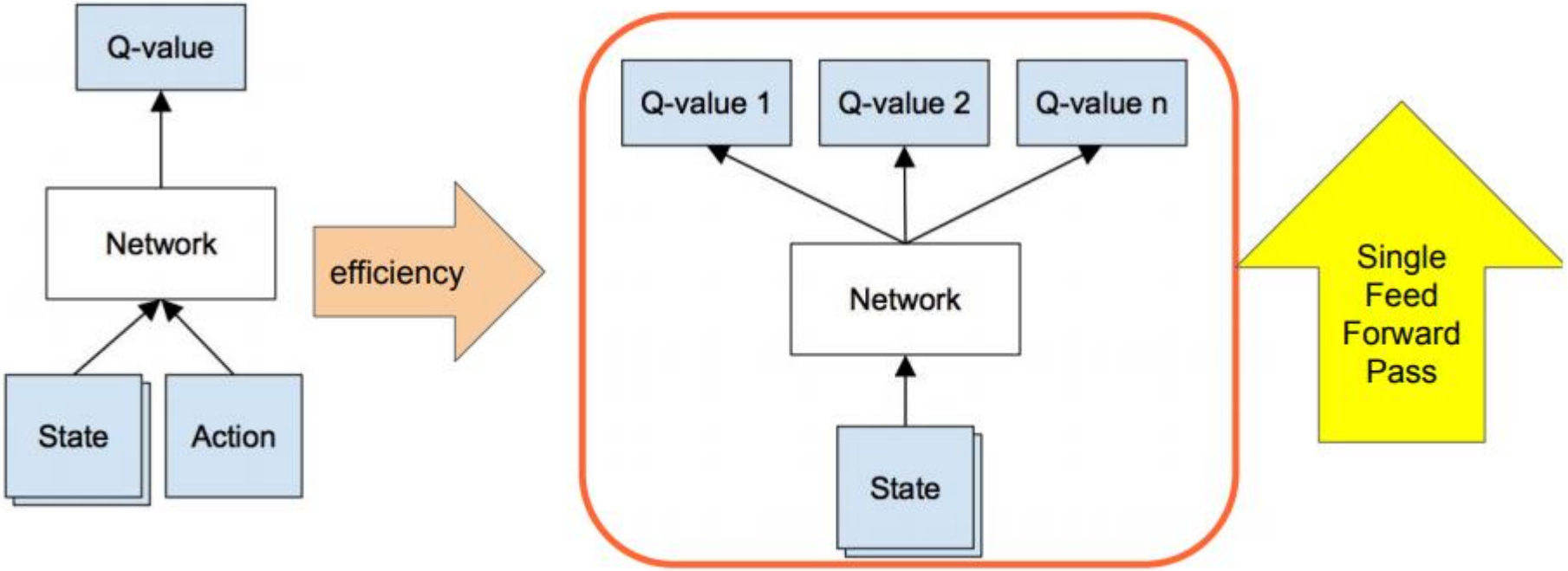
$$Q(s,a,\Theta) \approx Q^*(s,a)$$



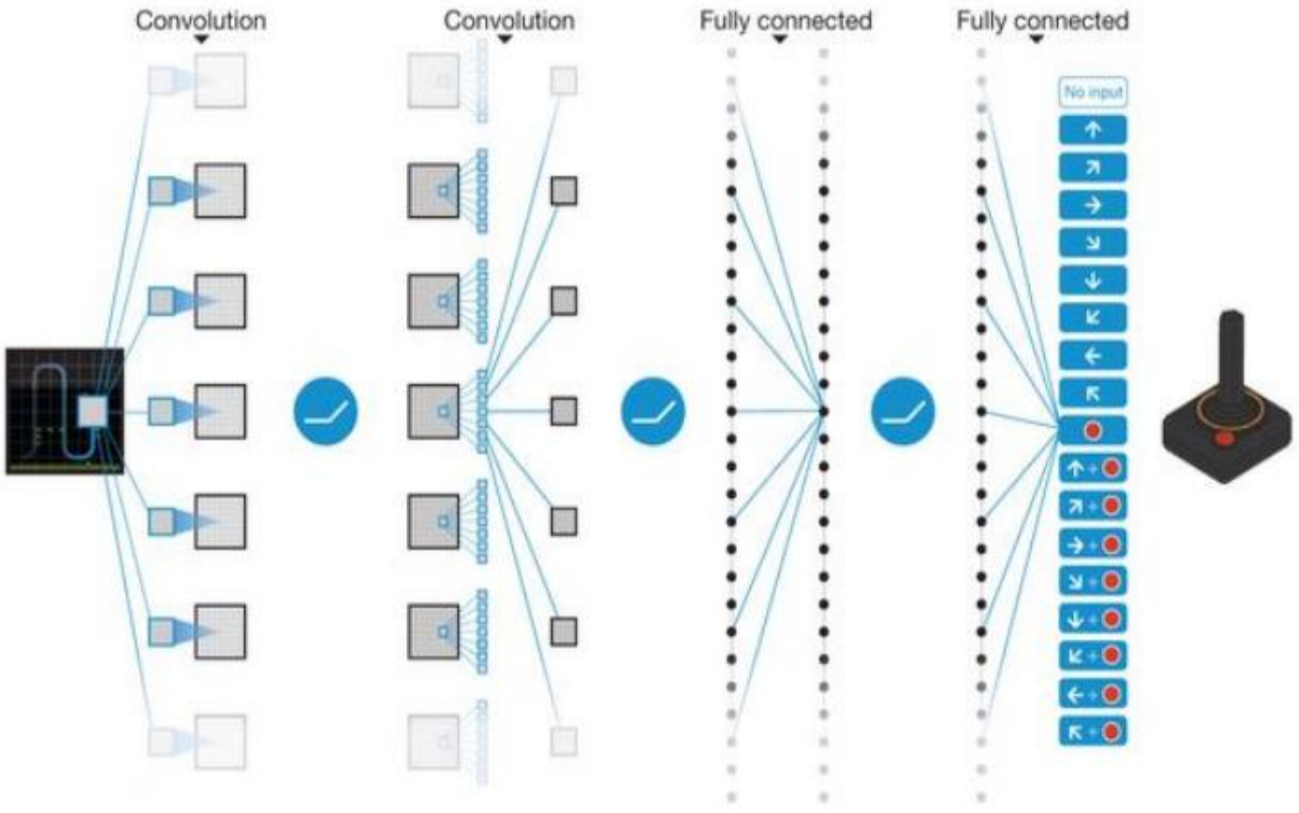
Deep Q-learning: Deep Q-Network DQN

$Q(s,a,\Theta) \approx Q^*(s,a)$

A **single feedforward** pass to compute the Q-values for all actions from the current state (efficient)



Deep Q-learning: Deep Q-Network DQN



Number of actions between 4-18, depending on the Atari game

Deep Q-learning: Deep Q-Network DQN

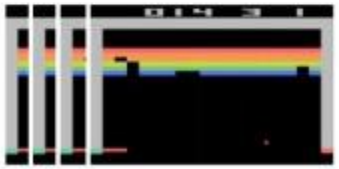
$Q(s_t, \leftarrow), Q(s_t, \rightarrow), Q(s_t, \uparrow), Q(s_t, \downarrow)$

FC-4 (Q-values)

FC-256

32 4x4 conv, stride 2

16 8x8 conv, stride 4



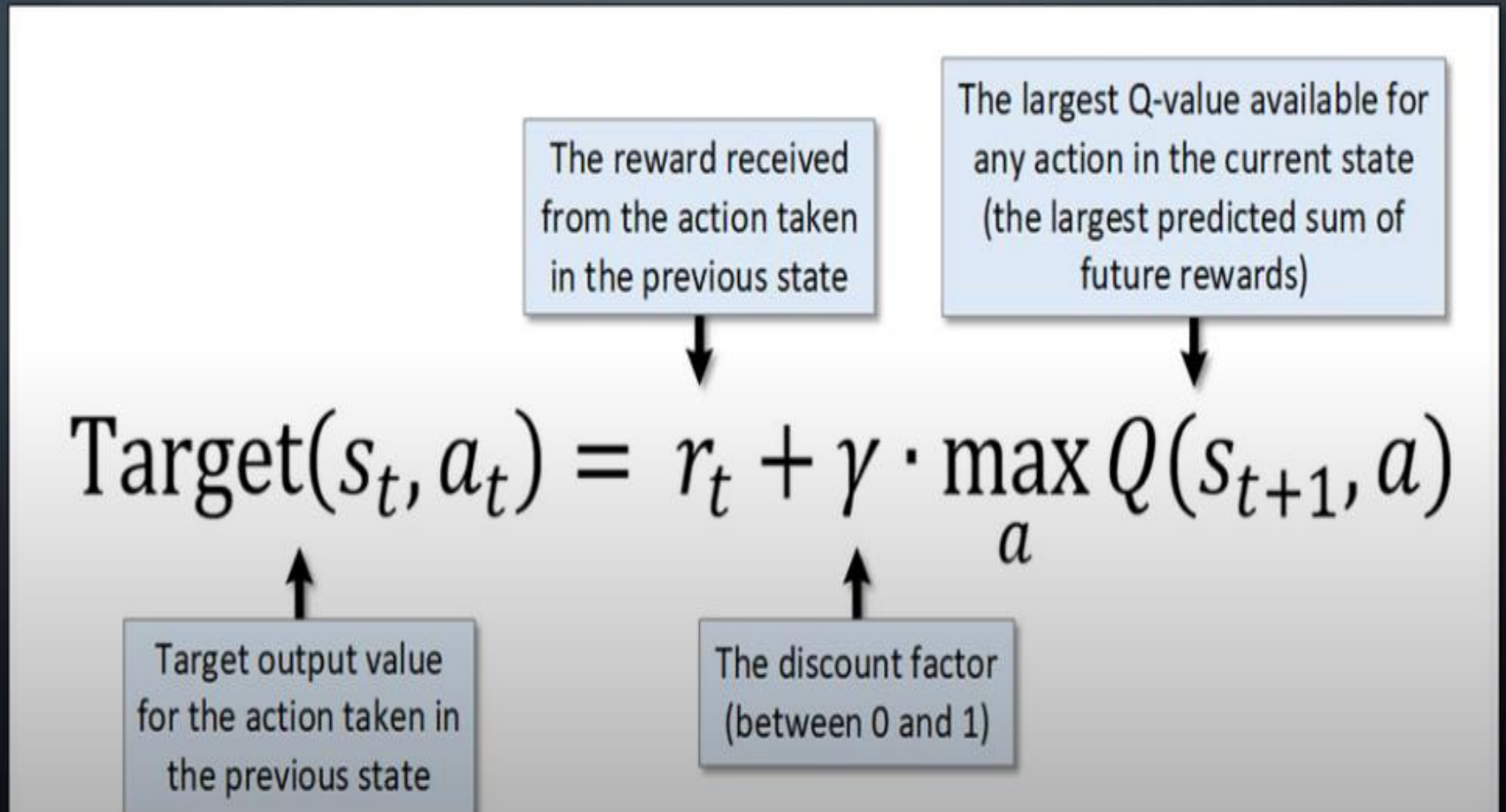
Current state s_t: 84x84x4 stack of last 4 frames
(after RGB->grayscale conversion, downsampling, and cropping)

WHAT IS DEEP Q-LEARNING?

- Deep Q-learning is a combination of Q-learning and *deep learning*
  - A deep learning model is simply a neural network with multiple hidden layers
- In deep Q-learning, the inputs into the deep neural network are the *states* of the environment, and the outputs are the set of *Q-values* for each action associated with the input state
  - We're thus using the neural network to predict the Q-values, rather than using the Bellman equation!
- Just as in regular Q-learning, the AI agent will usually take the action that has the largest Q-value
  - "Usually" rather than "always" because we need a mechanism that encourages the AI agent to explore its environment!

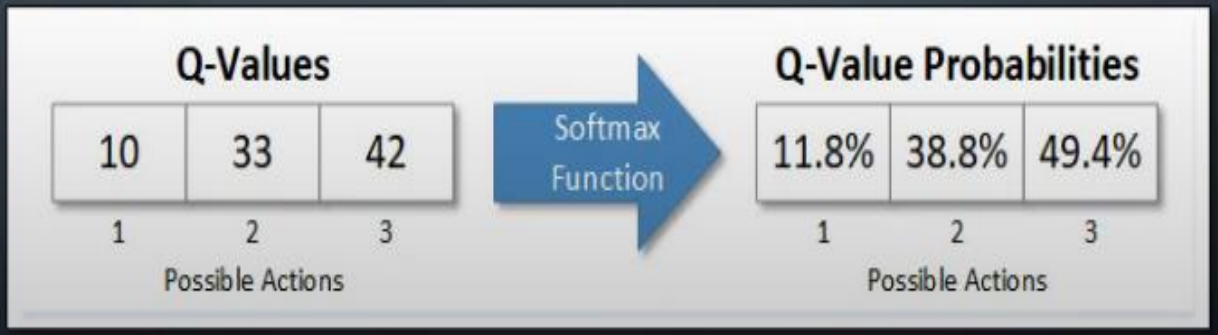
HOW DOES DEEP Q-LEARNING WORK?

- In deep Q-learning, the targets of an input state (i.e., the outputs of the neural network) are computed using part of the temporal difference formula:



HOW DOES DEEP Q-LEARNING WORK?

- In deep Q-learning, *exploration* is achieved by using the *softmax function*
  - Remember the exploration-exploitation dilemma – the AI agent needs to explore its options in order to identify how each option might influence the agent's goal of maximizing its total rewards
- The softmax function converts the set of Q-values for a state into a set of probabilities for each possible action that can be taken in that state
  - The softmax function thus yields a probability distribution for our Q-values!



- The action that the AI agent takes is determined by taking a random draw from the probability distribution returned by the softmax function!

EnD

Introduction

- In some applications, the output of the system is a sequence of *actions*. In such a case, a single action is not important

- *game playing where a single move by itself is not* that important.

Reinforcement learning

- in the case of the agent acts on its environment, it receives some evaluation of its action (reinforcement),

- but is not told of which action is the correct one to achieve its goal

Reinforcement learning

- Task
  - Learn how to behave successfully to achieve a goal while interacting with an external environment
  - *Learn via experiences!*
- Examples
  - Game playing: player knows whether it win or lose, but not know how to move at each step
  - Control: a traffic system can measure the delay of cars, but not know how to decrease it.

- At the heart of RL is the following analogy:
- consider teaching a dog a new trick: cannot tell it what to do, but you can reward/punish
- Here we train a computer as if we train a dog

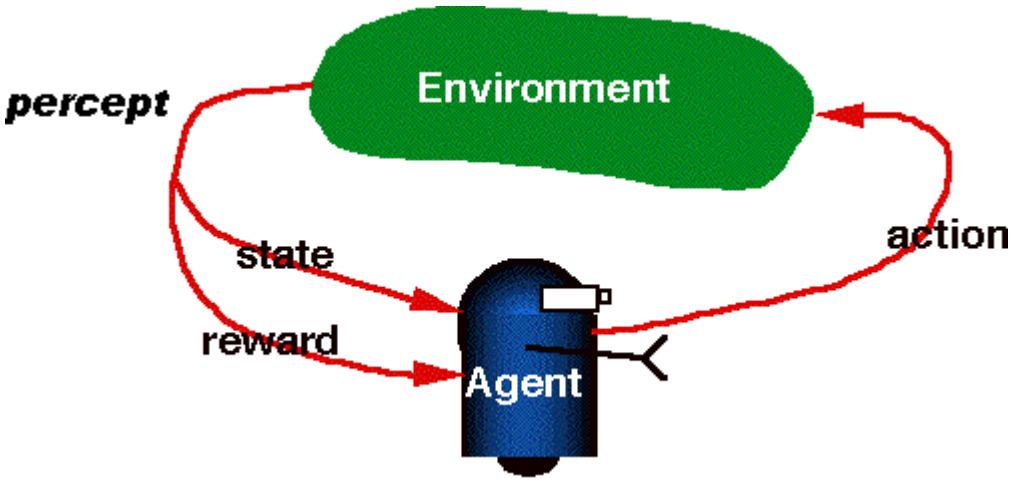
- If the dog obeys and acts according to our instructions we encourage it by giving biscuits or we punish it by beating or by scolding.

- Similarly, if the system works well then the teacher gives positive value (i.e. reward) or the teacher gives negative value (i.e. punishment).

- The learning system which gets the punishment has to improve itself.

- The *reinforcement learning algorithms* selectively retain the outputs that maximize the received reward over time.

RL is learning from interaction

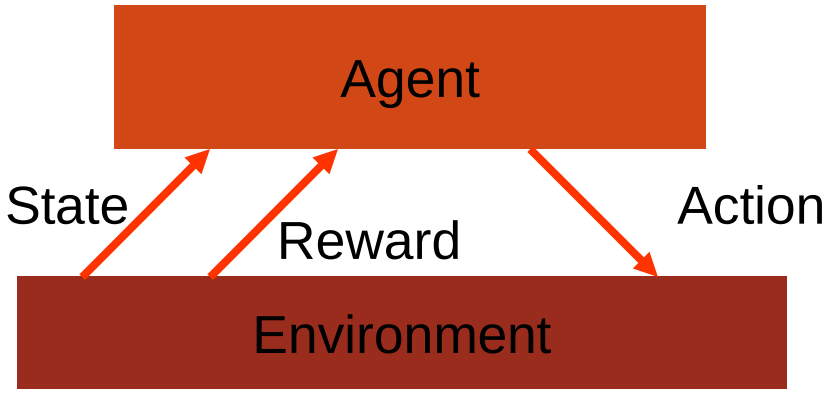


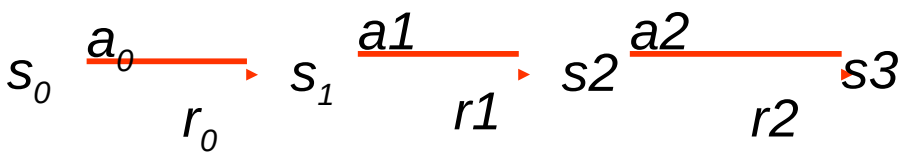
RL model

- Each percept(*e*) is enough to determine the State(the state is accessible)
- The agent can decompose the Reward component from a percept.
- The agent task: to find a optimal policy, mapping states to actions, that maximize long-run measure of the reinforcement
- Think of reinforcement as reward
- Can be modeled as MDP model!

Review of MDP model

- MDP model <S,T,A,R>





- S– set of states
- A– set of actions
- T(s,a,s') = P(s'|s,a)– the probability of transition from *s* to *s'* given action *a*
- R(s,a)– the expected reward for taking action *a* in state *s*

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

$$R(s,a) = \sum_{s'} T(s,a,s') r(s,a,s')$$

A Markov decision process is a 5-tuple $(S, A, P_{\cdot}(\cdot,\cdot), R_{\cdot}(\cdot,\cdot), \gamma)$, where

- S is a finite set of states,
- A is a finite set of actions (alternatively, A_s is the finite set of actions available from state s),
- $P_a(s, s') = \Pr(s_{t+1} = s' \mid s_t = s, a_t = a)$ is the probability that action a in state s at time t will lead to state s' at time $t+1$,
- $R_a(s, s')$ is the immediate reward (or expected immediate reward) received after transitioning from state s to state s', due to action a
- $\gamma \in [0, 1]$ is the discount factor, which represents the difference in importance between future rewards and present rewards.

(Note: The theory of Markov decision processes does not state that S or A are finite, but the basic algorithms below assume that they are finite.)

- The target function to be learned is in this case is a control policy. $\Phi: S \rightarrow A$.

- ***reinforcement learning problem differs from other function*** approximation tasks in several important respects.

- Delayed reward
- Exploration
- Partially observable states: sensors provide only partial information
- Life-long learning

THE LEARNING TASK

- formulate the problem of learning sequential control strategies more precisely
- **we might assume the agent's:**

1. actions are deterministic or that they are nondeterministic
2. can predict the next state that will result from each action
3. trained by an expert
4. train itself

Key word

- Reinforcement Learning
- Task
- Agent
- Environment
- Q-Learning
- MDP Model
- State
- Reward
- Target Fuction

References

- Our Text Book (Tom M.Techeel)
- https://www.quora.com/
- https://en.wikipedia.org/wiki/Markov_decision_process
- https://www-s.acm.illinois.edu/sigart/docs/QLearning.pdf
- http://www.cse.unsw.edu.au/~cs9417ml/RL1/algorithms.html
- http://cse-wiki.unl.edu/wiki/index.php/Q_learning
- http://mnemstudio.org/path-finding-q-learning-tutorial.htm
- http://artint.info/html/ArtInt_265.html
- https://en.wikipedia.org/wiki/Q-learning
- http://people.revoledu.com/kardi/tutorial/ReinforcementLearning/