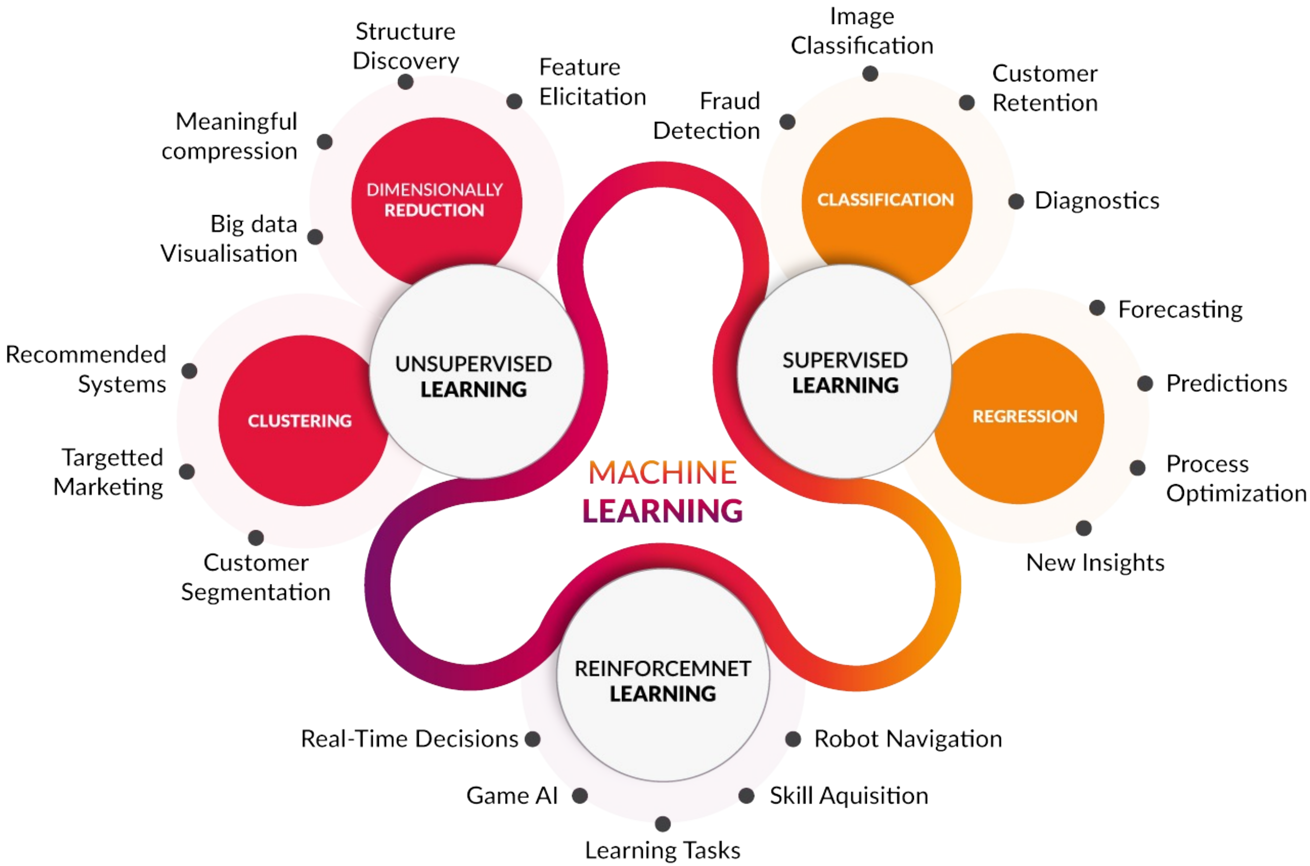
Reinforcement Learning

Overview

- What is Reinforcement Learning (RL)?
- Markov Decision Processes
- Q-Learning

Types of Learning



Source: cognub.com

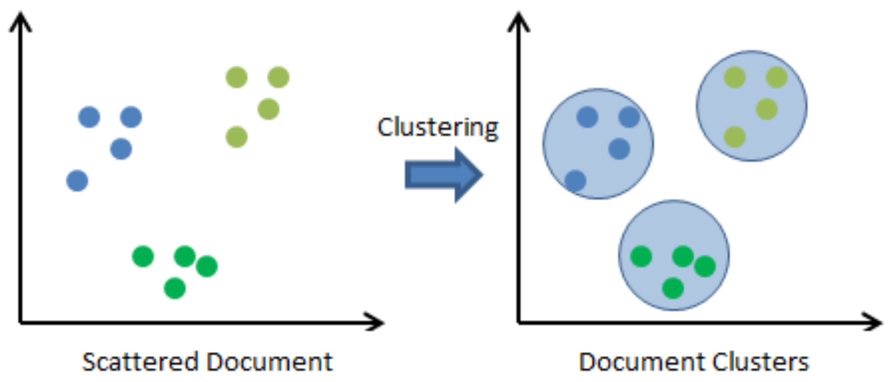
- **Data**: (x,y)
  - x is data
  - y is label
- **Goal**: Learn a function to map x -> y
- **Example**: Classification, regression, object detection, semantic segmentation, image captioning, etc.

CLASSIFICATION

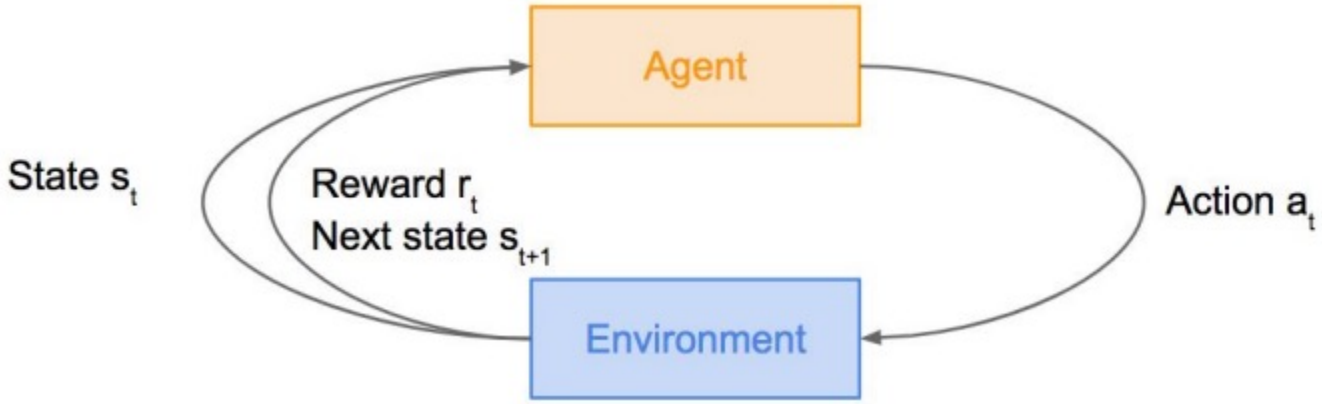
→ cat

Unsupervised Learning

- **Data**: x
  - Just data, NO label!
- **Goal**: Learn some underlying *hidden structure* of the data
- **Example**: Clustering, dimensionality reduction, feature learning, density estimation, etc.

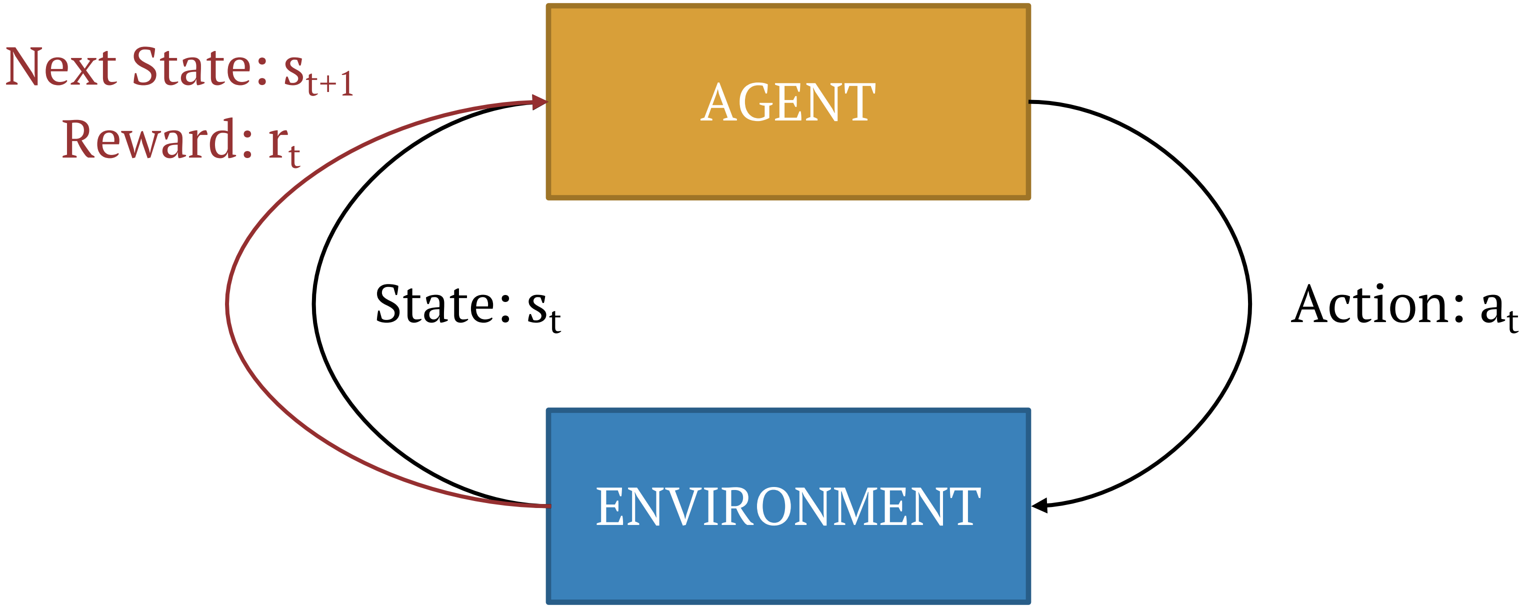


- Problems involving an **AGENT** interacting with an **ENVIRONMENT**, which provides numeric **REWARD** signals
- Goal: Learn how to take actions in order to maximize reward



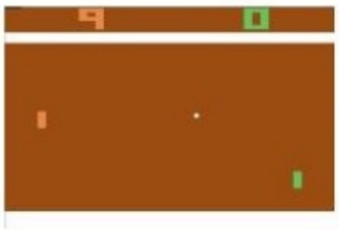
Atari's Arcade Game

Reinforcement Learning



Atari Games

- **Objective**: Complete the game with the highest score





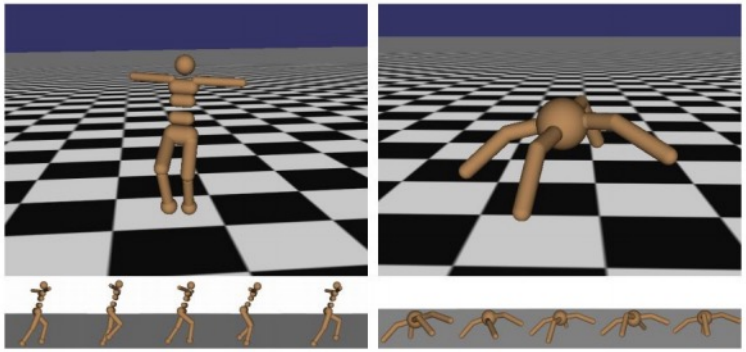
- **State**: Raw pixel inputs of the game state
- **Action**: Game controls e.g. Left, Right, Up, Down
- **Reward**: Score increase/decrease at each time step

Volodymyr Mnih et al., 2013

Robot Locomotion

- **Objective**: Make the robot move forward

- **State**:
- **Action**:
- **Reward**:

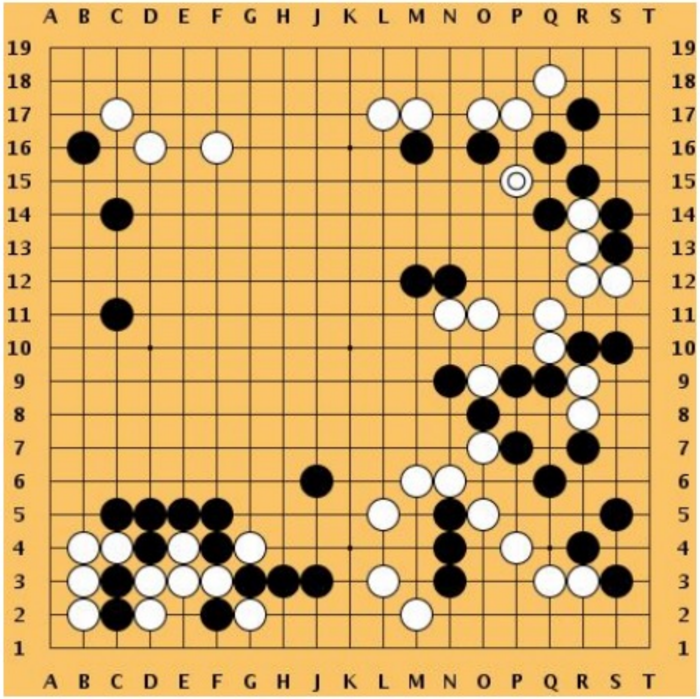


John Schulman et al., 2016

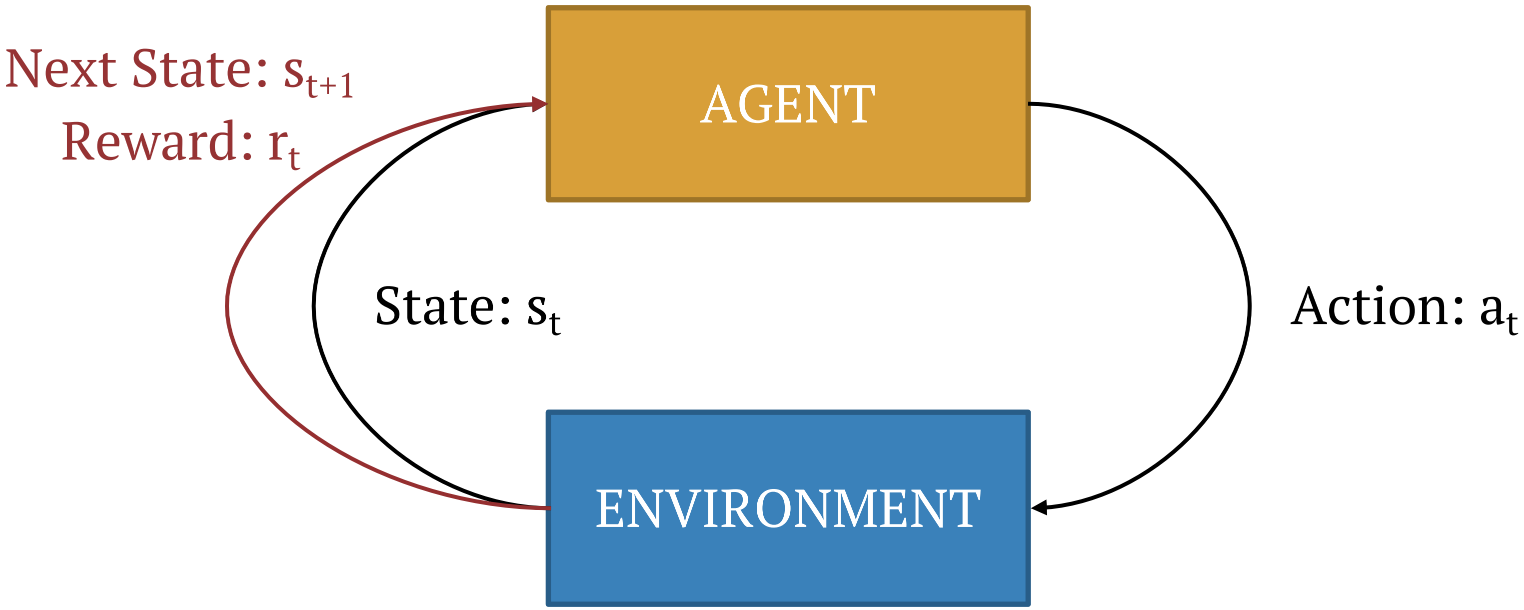
Go

- **Objective**: Win the game!

- **State**:
- **Action**:
- **Reward**:



Reinforcement Learning



How can we mathematically formalize the RL problem?

- Mathematical formulation of the RL problem –
- **Markov property**: Current state completely characterizes the state of the world

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

$\boldsymbol{\mathcal{S}}$: set of possible states

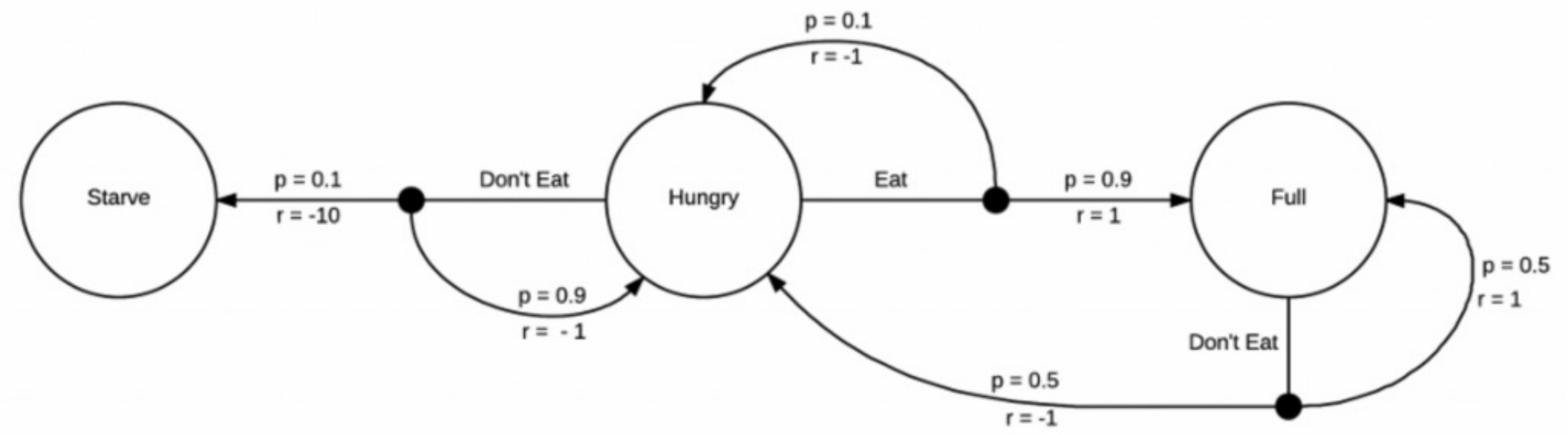
$\boldsymbol{\mathcal{A}}$: set of possible actions

$\boldsymbol{\mathcal{R}}$: 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: Example



https://bigdata.go.th/

Markov Decision Process

- At time step t=0, environment samples initial state $s_0 \sim p(s_0)$
- Then, for t=0 until done:
  - Agent selects action at a_t
  - Environment samples reward

$$r_t \sim R(.|s_t, a_t)$$

  - Environment samples next state

$$s_{t+1} \sim P(.|s_t, a_t)$$

  - Agent receives reward r_t and next state s_{t+1}

- A policy $\boldsymbol{\pi}$ is a function from S to A that specifies what action to take in each state
- **Objective**: find policy $\boldsymbol{\pi^*}$ that maximizes cumulative discounted reward:

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

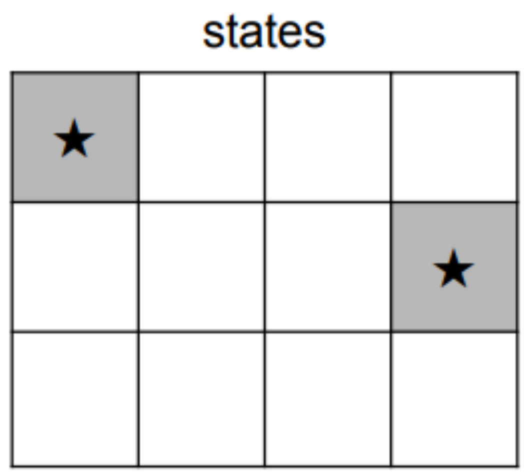
$$= r_t + \gamma^t r_{t+1} + \gamma^2 r_{t+2} + \cdots$$

A Simple MDP: Grid World

actions = {

1. right
2. left
3. up
4. down

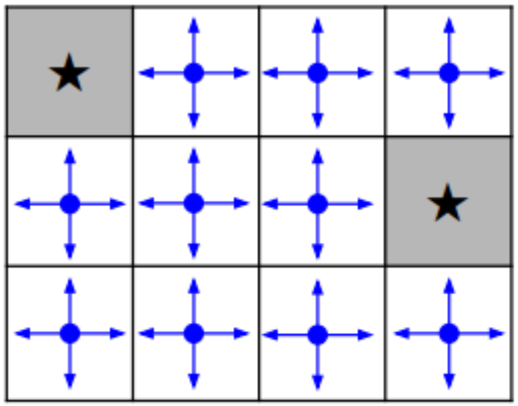
}



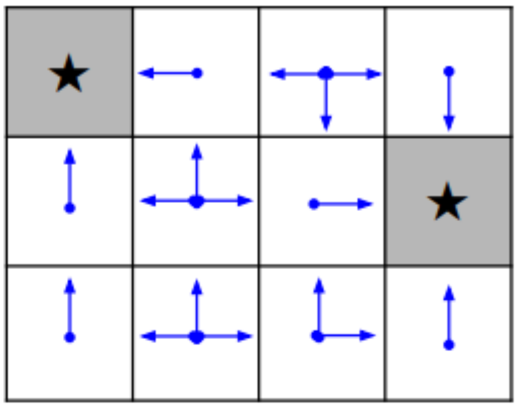
Set a negative "reward" for each transition (e.g. r = -1)

Objective: reach one of terminal states (greyed out) in least number of actions

A Simple MDP: Grid World



Random Policy



Optimal Policy

Objective: reach one of terminal states (greyed out) in least number of actions

The Optimal Policy ***π****

- **Objective**: find optimal policy ***π**** that maximizes the sum of rewards
- To handle the randomness (initial state, transition probability, etc.), we need to maximize the **expected sum of rewards**!

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

- Following a policy produces sample trajectories (or paths): $(s_0, a_0, r_0), (s_1, a_1, r_1), \ldots$

- **How good is a state?**
  - The **value function** at state s, is the expected cumulative reward from following the policy from state s:

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

Q-Value Function

- Following a policy produces sample trajectories (or paths): $(s_0, a_0, r_0), (s_1, a_1, r_1), \ldots$

- **How good is a state-action pair?**
  - The **Q-value function** at state s and action a, is the expected cumulative reward from taking action a in state s and then following the policy:

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

Bellman Equation

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

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

- Q* satisfies the following Bellman equation:

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

Solving for the Optimal Policy

- **Value iteration** algorithm: Use Bellman equation as an iterative update

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

Q_i will converge to Q^* as i -> infinity

- Problem: Not scalable.
  - Must compute Q(s,a) for every state-action pair.
  - If state, e.g. current game state pixels,is computationally infeasible to compute for entire state space!

Solving for the Optimal Policy: Q-Learning

- **Solution:** use a function approximator to estimate Q(s,a) e.g. a neural network!

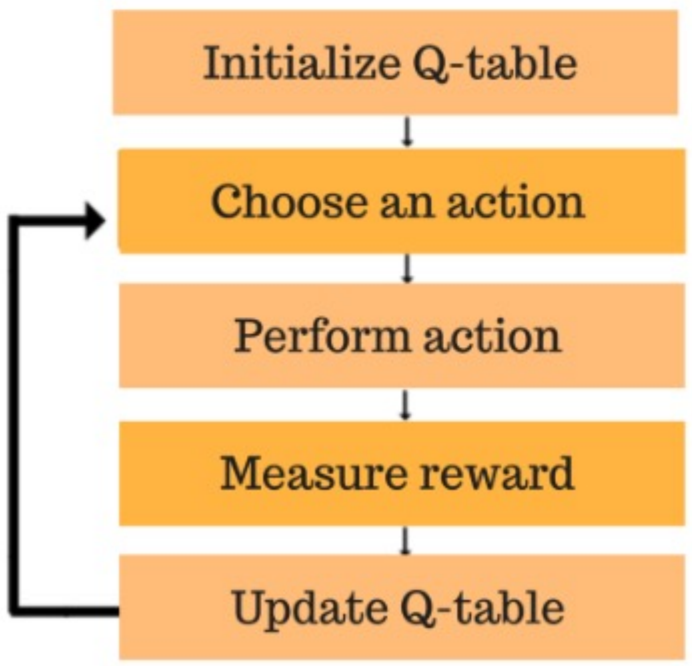
- **Q-learning** uses a function approximator to estimate the action-value function

$$Q(s, a; \theta) \approx Q^*(s, a)$$

function parameters (weights)

- If the function approximator is a deep neural network => **Deep Q-learning!**

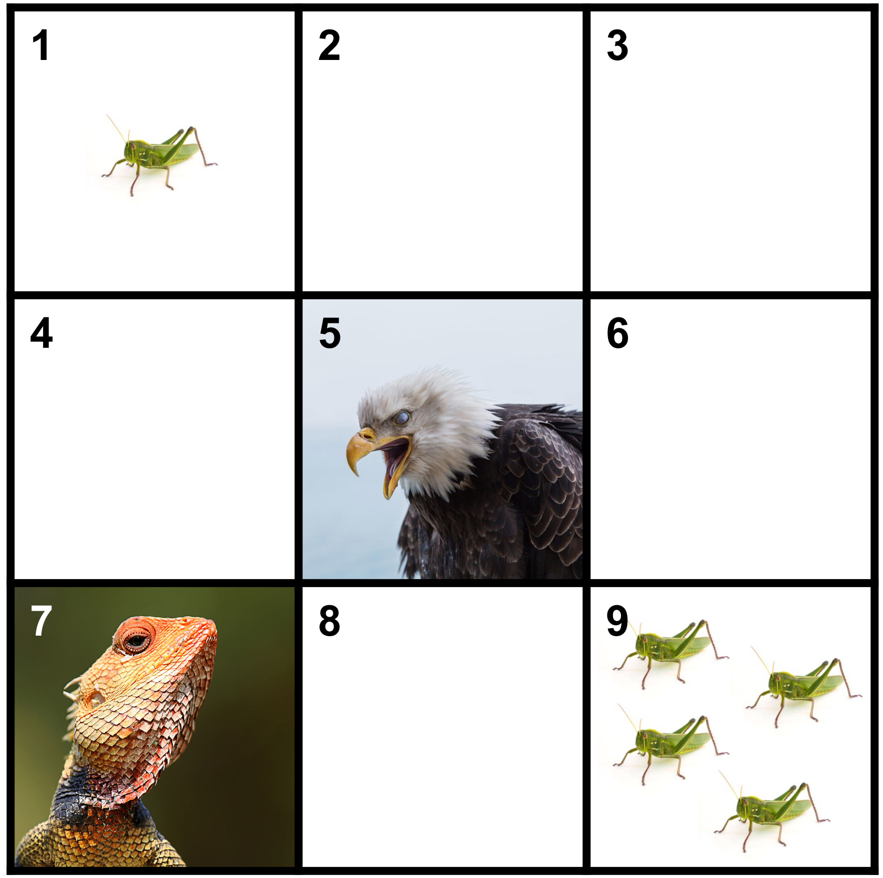
Simple Q-learning Algorithm Process

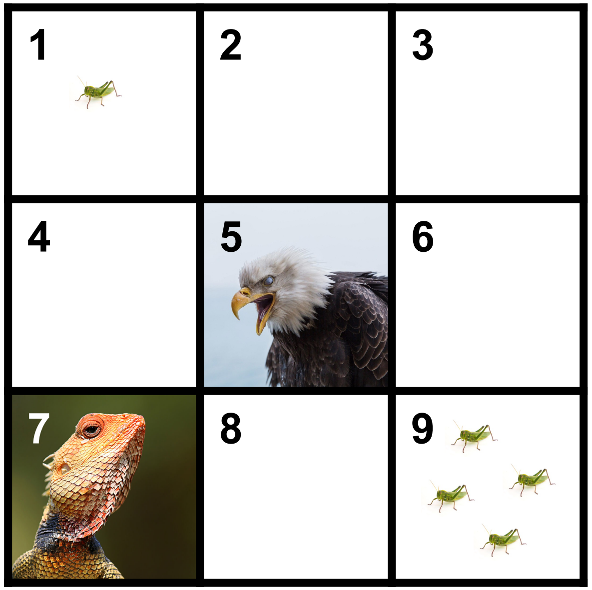


actions = {

1. right
2. left
3. up
4. down

}





Empty cell = -1
(1) Cricket = +1
(5) Eagle = -10 [End Game]
(9) Crickets = +10 [End Game]

Q-Table		**Action**			
		Left	**Right**	**Up**	**Down**
States	**1 cricket**	0	0	0	0
	2	0	0	0	0
	3	0	0	0	0
	4	0	0	0	0
	5 eagle	0	0	0	0
	6	0	0	0	0
	7 lizard	0	0	0	0
	8	0	0	0	0
	9 crickets	0	0	0	0

- Goal: want to find a Q-function that satisfies the Bellman Equation:

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

- **Forward Feeding**
  - Loss Function:

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

  - where

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

Q-Learning

- Goal: want to find a Q-function that satisfies the Bellman Equation:

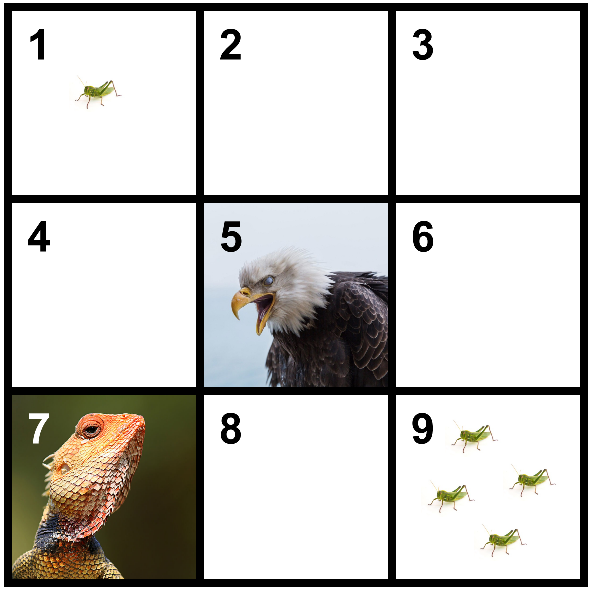
$$Q^*(s,a) = \mathbb{E}_{s' \sim \mathcal{E}}\left[r + \gamma \max_{a'} Q^*(s',a') \middle| s,a\right]$$

- **Backpropagation**
  - Gradient update (with respect to Q-function parameters θ):

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

Update Q-Table

$$\underbrace{NewQ(s,a)}_{\text{New Q value for that state and that action}} = \underbrace{Q(s,a)}_{\text{Current Q value}} + \underset{\text{Learning Rate}}{\alpha}\left[\underbrace{R(s,a)}_{\text{Reward for taking that action at that state}} + \underset{\text{Discount rate}}{\gamma}\, \underbrace{max\, Q'(s',a')}_{\text{Maximum expected future reward } \textbf{given the new s' and all possible actions at that new state}} - Q(s,a)\right]$$



Empty cell = -1
(1) Cricket = +1
(5) Eagle = -10 [End Game]
(9) Crickets = +10 [End Game]

Q-Table		Action			
		Left	Right	Up	Down
States	1 cricket	*UPDATE THE TABLE!!*			
	2				
	3				
	4				
	5 eagle				
	6				
	7 lizard				
	8				
	9 crickets				