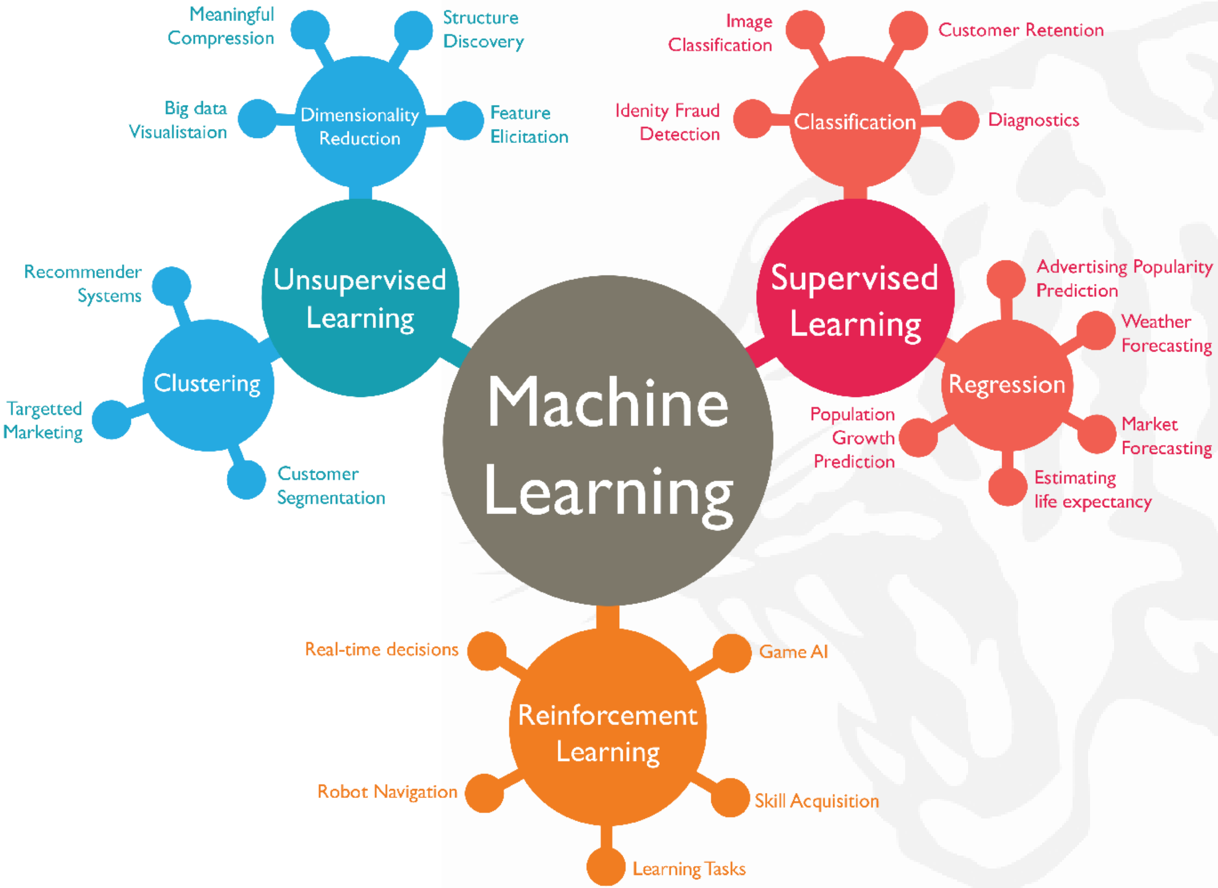
Reinforcement Learning

Deep Learning Basics

Prof. Joongheon Kim
Korea University, School of Electrical Engineering
Artificial Intelligence and Mobility Laboratory
https://joongheon.github.io
joongheon@korea.ac.kr

Machine Learning Overview

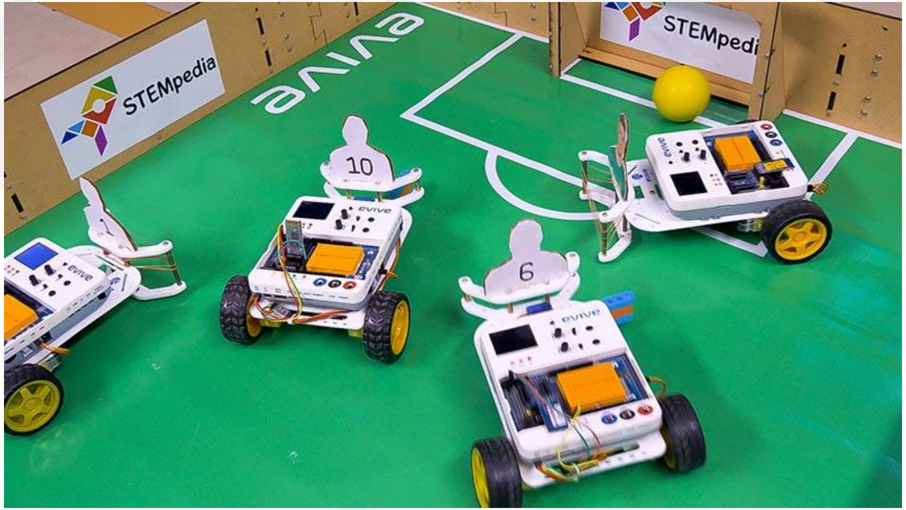


- Brief History and Successes
  - Minsky's PhD thesis (1954): Stochastic Neural-Analog **Reinforcement** Computer
  - Analogies with animal learning and psychology
  - Job-shop scheduling for NASA space missions (Zhang and Dietterich, 1997)
  - Robotic soccer (Stone and Veloso, 1998) – part of the world-champion approach

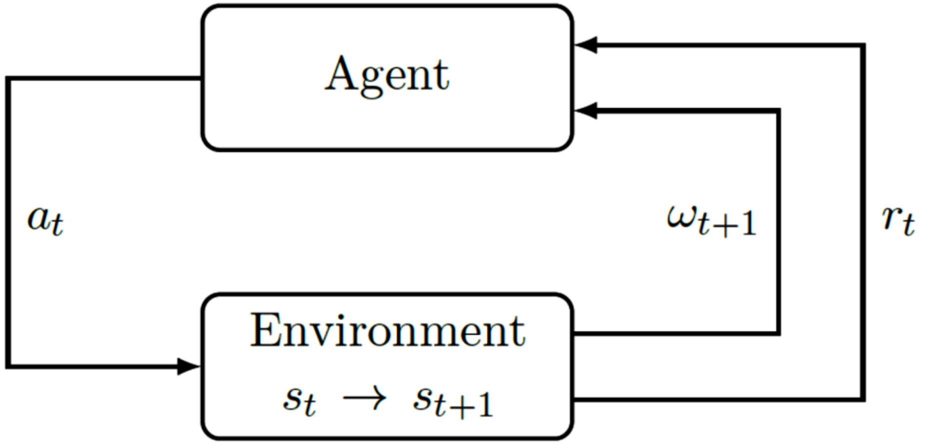
- When RL can be used?
  - Find the (approximated) **optimal action sequence** for **expected reward maximization (not for single optimal solution)**
  - Define **actions** and **rewards**. These are all we need to do.

Introduction to RL

- Action Sequence (also called **Policy**, later in this presentation)!



- **RL Setting**
  - The general RL problem is formalized as a **discrete time stochastic control process** where **an agent interacts with its environment** as follows:
    1. The agent starts
       in a given state within its environment $s_0 \in S$
       by gathering an initial observation $\omega_0 \in \Omega$.
    2. At each time step t,
       The agent has to take an action $a_t \in A$.
       It follows three consequences:
       1) Obtains a reward $r_t \in R$
       2) State transitions to $s_{t+1} \in S$
       3) Obtains an observation $\omega_{t+1} \in \Omega$



Course Outline

Reinforcement Learning

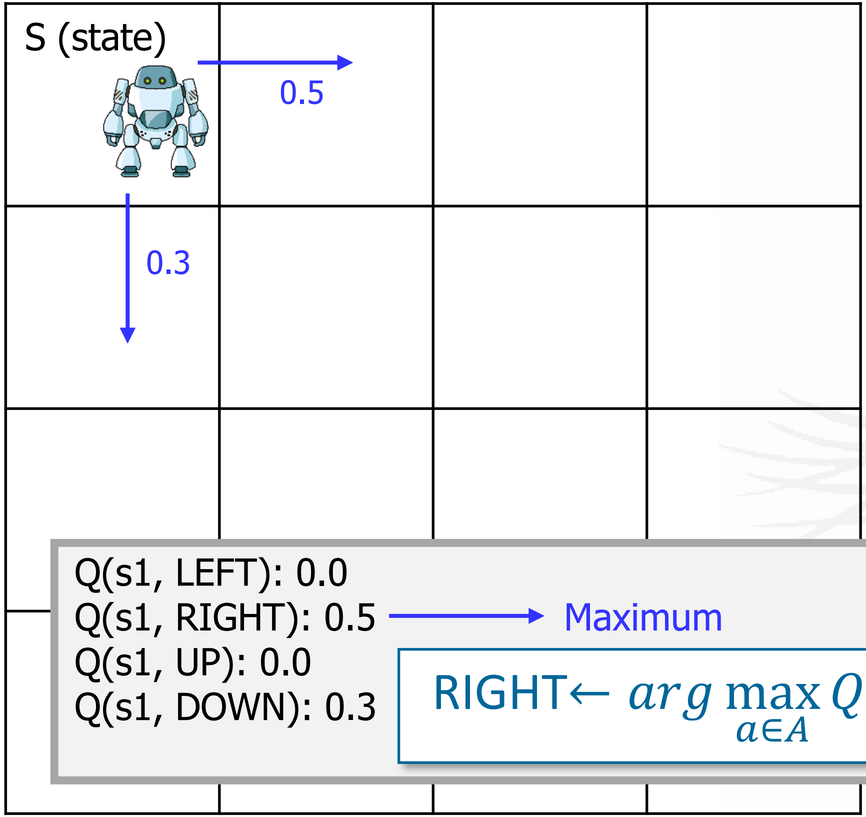
<u>Q-Learning</u>

MDP

Deep Reinforcement Learning

DQN

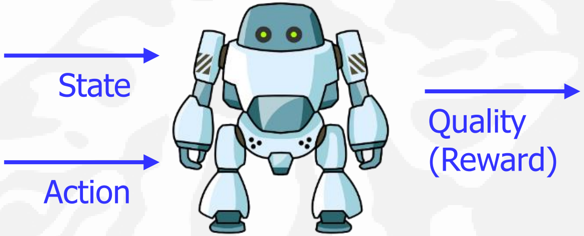
Imitation Learning



Q(s1, LEFT): 0.0
Q(s1, RIGHT): 0.5 → Maximum
Q(s1, UP): 0.0
Q(s1, DOWN): 0.3

$$\text{RIGHT} \leftarrow arg \max_{a \in A} Q(s_1, a)$$

- Q-Function (State-action value)

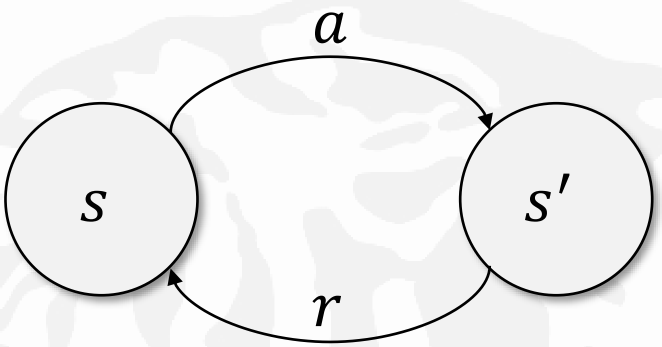


Q (state, action)

Optimal Policy π and Max Q

- Max Q $= \max_{a'} Q(s, a')$
- $\pi^*(s) = arg \max_{a} Q(s, a)$

- My condition
  - I am now in state s
  - When I do action a, I will go to s'.
  - When I do action a, I will get reward r
  - Q in s', it means $Q(s', a')$ exists.



- How can we express $Q(s, a)$ using $Q(s', a')$?

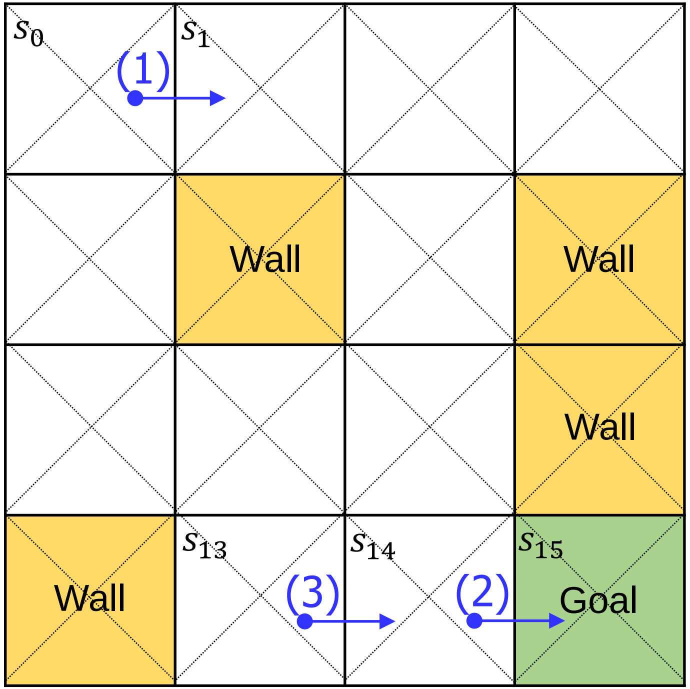
$$Q(s,a) = r + \max_{a'} Q(s', a')$$

Recurrence (e.g., factorial)

```
F(x){
        if (x != 1){ x * F(x-1) }
        if (x == 1){ F(x) = 1 }
        }
}
```

```
3! = F(3) = 3 * F(2)
          = 3 * 2 * F(1)
          = 3 * 2* 1 = 6
```

- 16 states and 4 actions (U, D, L, R)



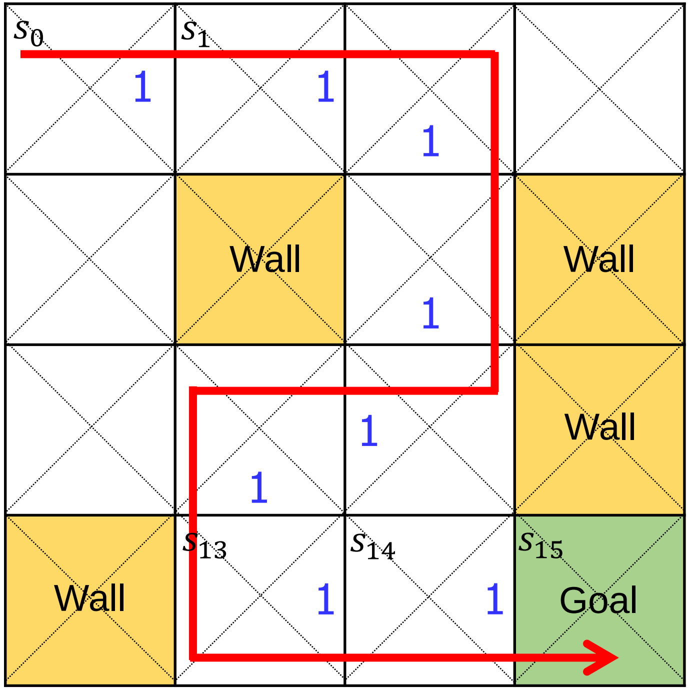
- Initial Status
  - All 64 Q values are 0,
  - Reward are all zero except $r_{s_{15},L} = 1$

- For (1), from s_0 to s_1
  - $Q(s_0, a_R) = r + \max_a Q(s_1, a) = 0 + \max\{0,0,0,0\} = 0$

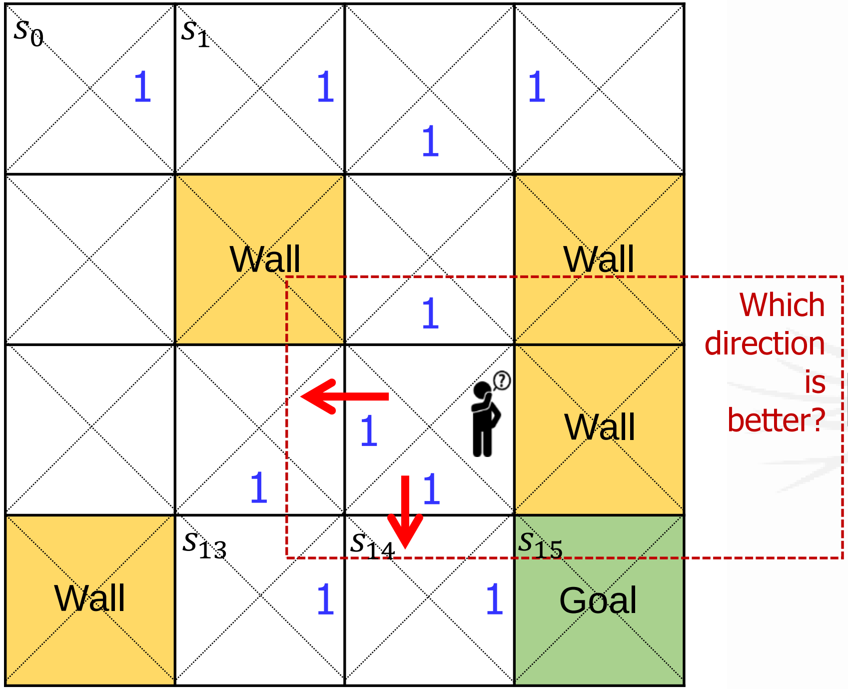
- For (2), from s_{14} to s_{15} (goal)
  - $Q(s_{14}, a_R) = r + \max_a Q(s_{15}, a) = 1 + \max\{0,0,0,0\} = 1$

- For (3), from s_{13} to s_{14}
  - $Q(s_{13}, a_R) = r + \max_a Q(s_{14}, a) = 0 + \max\{0,0,1,0\} = 1$

- 16 states and 4 actions (U, D, L, R)



- 16 states and 4 actions (U, D, L, R)



Learning $Q(s,a)$ with Discounted Reward

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

$$0 < \gamma \leq 1$$

Course Outline

Reinforcement Learning

Q-Learning

MDP

Deep Reinforcement Learning

DQN

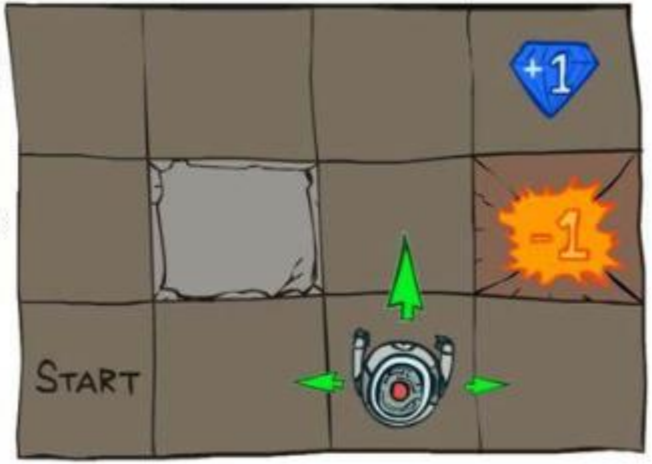
Imitation Learning

- Markov Decision Process (MDP) Components: <S, A, R, T, γ>
  - S: Set of states
  - A: Set of actions
  - R: Reward function
  - T: Transition function
  - γ: Discount factor

How can we use MDP to model agent in a maze?

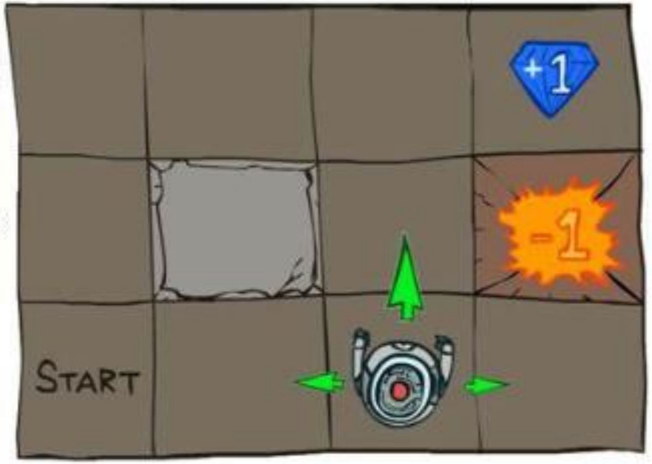
Markov Decision Process (MDP)

- Markov Decision Process (MDP) Components: $<S, A, R, T, \gamma>$
  - **S: Set of states**
  - A: Set of actions
  - R: Reward function
  - T: Transition function
  - γ: Discount factor



S: location (x, y) if the maze is a 2D grid
- s_0: starting state
- s: current state
- s': next state
- s_t: state at time t

- Markov Decision Process (MDP) Components: <S, A, R, T, γ>
  - S: Set of states
  - **A: Set of actions**
  - R: Reward function
  - T: Transition function
  - γ: Discount factor



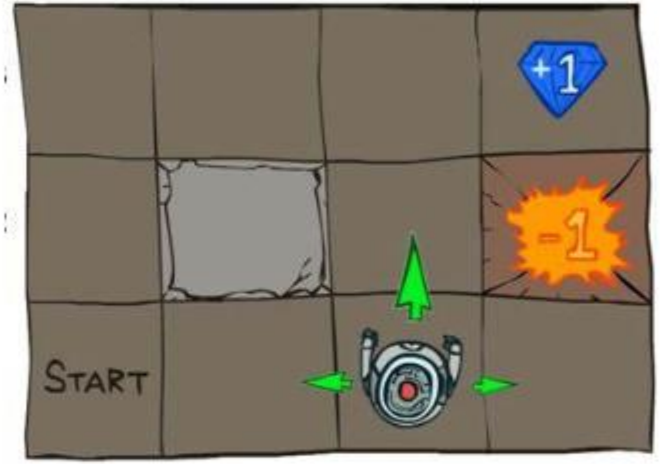
S: location (x, y) if the maze is a 2D grid

A: move up, down, left, or right

- $s \rightarrow s'$

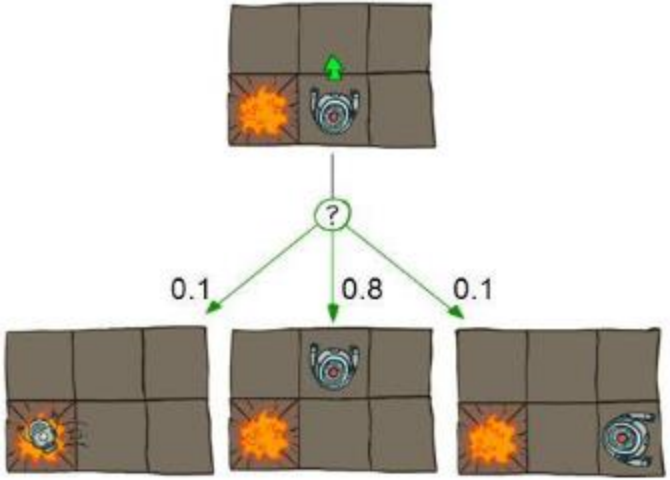
Markov Decision Process (MDP)

- Markov Decision Process (MDP) Components: $<S, A, R, T, \gamma>$
  - S: Set of states
  - A: Set of actions
  - **R: Reward function**
  - T: Transition function
  - γ: Discount factor



S: location (x, y) if the maze is a 2D grid
A: move up, down, left, or right
R: how good was the chosen action?
- $r = R(s, a, s')$
- -1 for moving (battery used)
- +1 for jewel? +100 for exit?

- Markov Decision Process (MDP) Components: <S, A, R, T, γ>
  - S: Set of states
  - A: Set of actions
  - R: Reward function
  - **T: Transition function**
  - γ: Discount factor



Stochastic Transition

S: location (x, y) if the maze is a 2D grid
A: move up, down, left, or right
R: how good was the chosen action?
T: where is the robot's new location?
- $T = P(s'|s, a)$

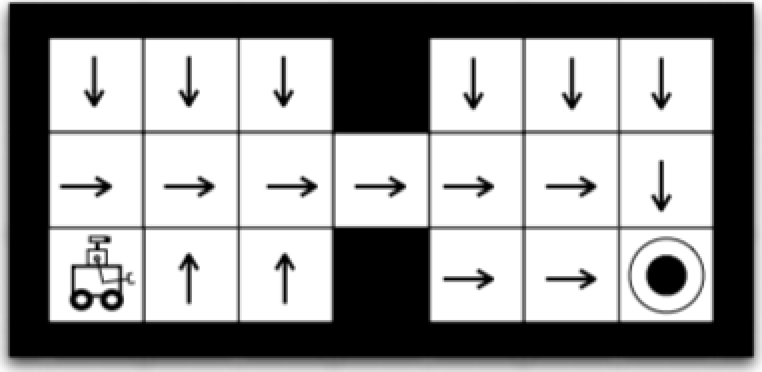
- Markov Decision Process (MDP) Components: $<S, A, R, T, \gamma>$
  - S: Set of states
  - A: Set of actions
  - R: Reward function
  - T: Transition function
  - **γ: Discount factor**

S: location (x, y) if the maze is a 2D grid
A: move up, down, left, or right
R: how good was the chosen action?
T: where is the robot's new location?
γ: how much does future reward worth?

- $0 \leq \gamma \leq 1$, [$\gamma \approx 0$: future reward is near 0 (immediate action is preferred)]

Markov Decision Process (MDP)

- Policy
  - $\pi: S \to A$
  - Maps states to actions
  - Gives an action for every state
- Return
  - $$\boxed{R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}}$$



Our goal:
Find π that maximizes expected return!

- State Value Function (V)

$$V^{\pi}(s) = E_{\pi}(R_t|s_t = s) = E_{\pi}\left(\sum_{k=0}^{\infty} \gamma^k r_{t+k} \middle| s_t = s\right)$$

  - Expected return of starting at state s and following policy π
  - How much return do I expect starting from state s?

- Action Value Function (Q)

$$Q^{\pi}(s,a) = E_{\pi}(R_t|s_t = s, a_t = a) = E_{\pi}\left(\sum_{k=0}^{\infty} \gamma^k r_{t+k} \middle| s_t = s, a_t = a\right)$$

  - Expected return of starting at state s, taking action a, and then following policy π
  - How much return do I expect starting from state s and taking action a?

- Our goal is to find the optimal policy

$$\boxed{\pi^*(s) = \max_{\pi} R^{\pi}(s)}$$

  - If $T(s'|s,a)$ and $R(s,a,s')$ are known, this is a **planning** problem.
  - We can use **dynamic programming** to find the optimal policy.

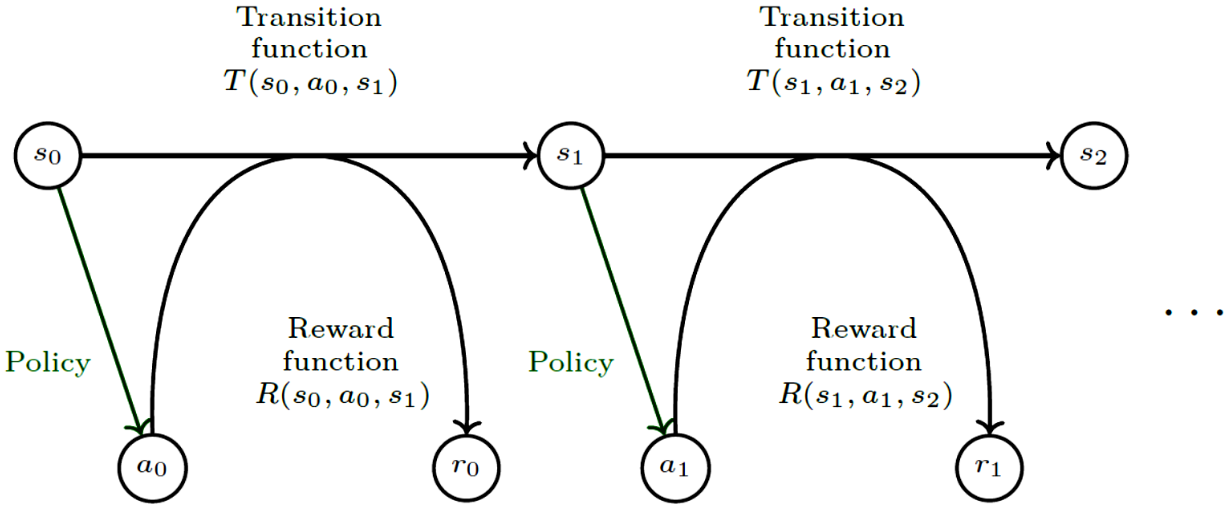
- Notes
  - Bellman Equation (Value Iteration)

$$\boxed{\forall s \in S:\ V^*(s) = \max_{a} \sum_{s'} \{R(s,a,s') \cdot T(s,a,s') + \gamma V^*(s')\}}$$

- **Markov Property**
  - [Definition (Markovian)] A discrete time stochastic control process is Markovian (i.e., it has the Markov property) if
    - $P(\omega_{t+1}|\omega_t, a_t) = P(\omega_{t+1}|\omega_t, a_t, \cdots, \omega_0, a_0)$, and
    - $P(r_t|\omega_t, a_t) = P(r_t|\omega_t, a_t, \cdots, \omega_0, a_0)$
  - The Markov property means that the future of the process only depends on the current observation, and the agent has no interest in looking at the full history.

- **Markov Property**
  - [Definition (MDP)] A Markov Decision Process (MDP) is a discrete time stochastic control process defined as follows. An MDP is a 5-tuple (S, A, T, R, γ) where:
    - S is the state space,
    - A is the action space,
    - $T: S \times A \times S \rightarrow [0,1]$ is the transition function (set of conditional transition probabilities between states),
    - $R: S \times A \times S \rightarrow R$ is the reward function, where R is a continuous set of possible rewards in a range $R_{\max} \in R^{+}$ (e.g., $[0, R_{\max}]$),
    - $\gamma \in [0,1)$ is the discount factor.

- **Markov Property**
  - The system in [Definition (MDP)] is fully observable in an MDP, which means that the observation is the same as the state of the environment: $\omega_t = s_t$.
  - At each time step t,
    - The probability of moving to s_{t+1} is given by the state transition function $T(s_t, a_t, s_{t+1})$ and the reward is given by a bounded reward function $R(s_t, a_t, s_{t+1}) \in R$.



- **Expected Return**
  - Q-value function $Q^{\pi}(s,a): S \times A \rightarrow R$ is defined as follows:

$$Q^{\pi}(s,a) = E\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \,|s_t = s, a_t = a, \pi\right]$$

  - This can be rewritten recursively in the case of an MDP using Bellman's equation:

$$Q^{\pi}(s,a) = \sum_{s\prime \in S} T(s,a,s')\{R(s,a,s') + \gamma Q^{\pi}(s', a = \pi(s'))\}$$

  - Similar to the V-value function, the optimal Q-value function $Q^{*}(s,a)$ is as:

$$Q^{*}(s,a) = \max_{\pi \in \Pi} Q^{\pi}(s,a)$$

- **Expected Return**
  - The **optimal policy** can be obtained directly from $Q^*(s,a) = \max_{\pi \in \Pi} Q^{\pi}(s,a)$:

$$\boxed{\pi^*(s) = arg \max_{a \in A} Q^*(s,a)}$$

Course Outline

Reinforcement Learning

Q-Learning
MDP

Deep Reinforcement Learning

DQN

Imitation Learning

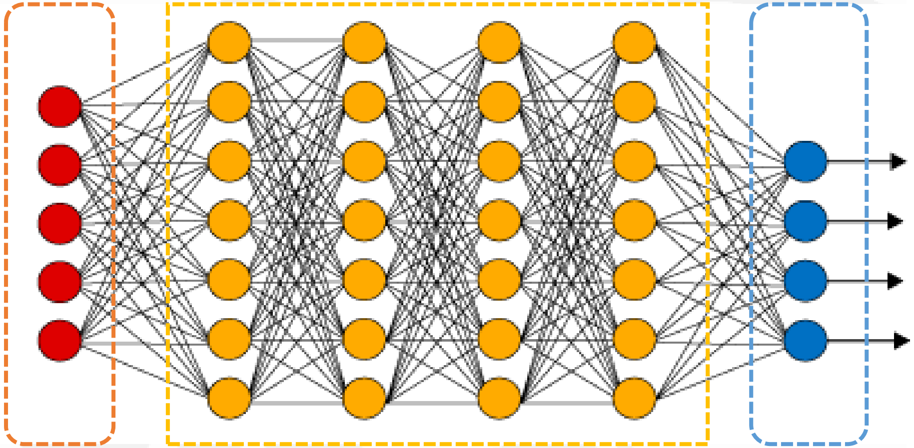
- How Deep Learning Works?
  - Deep Learning Computation Procedure

Deep Learning Model Setup
- MLP, CNN, RNN, GAN, or Customized
- # Hidden Layers, # Units, Input/Output, …
- Cost Function / Optimizer Selection

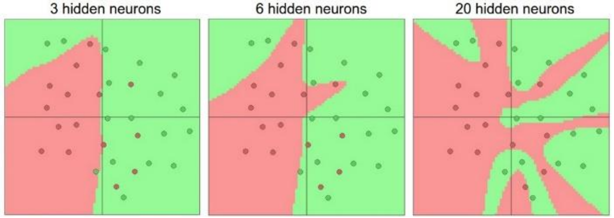
Training (with Large-Scale Dataset)
- Input: Data, Output: Labels
- Learning → Weights Updates for Cost Function Minimization

Inference / Testing (Real-Word Execution)
- Input: Real-World Input Data
- Output: Interference Results based on Updated Weights in Deep Neural Networks

Input **Multi-Layer Perceptron (MLP)** **Output**



Non-Linear Training (Weights Updates) for Cost Minimization: GD, SGD, Adam, etc.



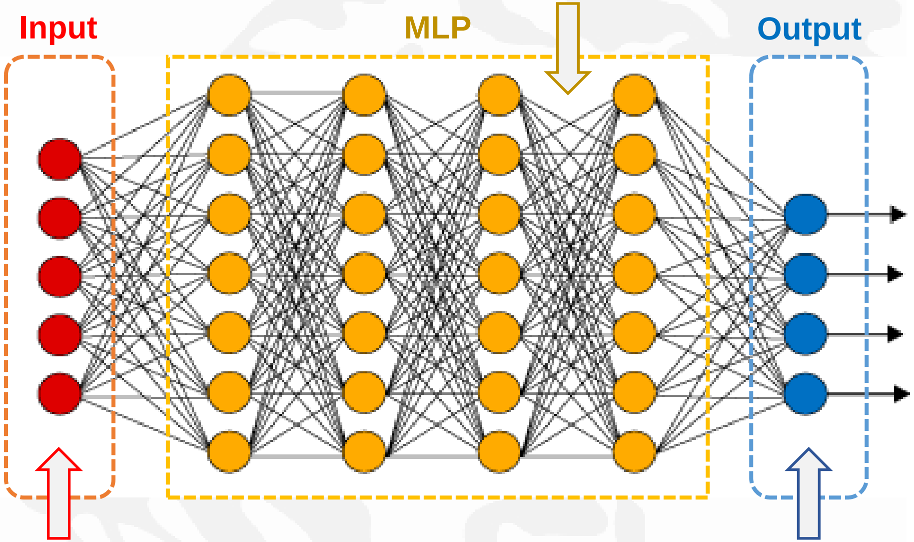
- How Deep Learning Works?
  - Deep Learning Computation Procedure

Deep Learning Model Setup
- MLP, CNN, RNN, GAN, or Customized
- # Hidden Layers, # Units, Input/Output, …
- Cost Function / Optimizer Selection

Training (with Large-Scale Dataset)
- Input: Data, Output: Labels
- Learning → Weights Updates for Cost Function Minimization

Inference / Testing (Real-Word Execution)
- Input: Real-World Input Data
- Output: Interference Results based on Updated Weights in Deep Neural Networks

All weights in units are trained/set (under cost minimization)



INPUT: Data
- One-Dimension Vector

OUTPUT: Labels
- One-Hot Encoding

We need a lot of training data for generality (otherwise, we will suffer from overfitting problem).

- How Deep Learning Works?
  - Deep Learning Computation Procedure

Deep Learning Model Setup
- MLP, CNN, RNN, GAN, or Customized
- # Hidden Layers, # Units, Input/Output, …
- Cost Function / Optimizer Selection

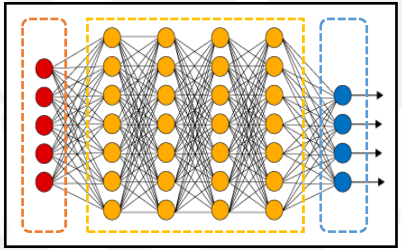
↓

Training (with Large-Scale Dataset)
- Input: Data, Output: Labels
- Learning → Weights Updates for Cost Function Minimization

↓

Inference / Testing (Real-Word Execution)
- Input: Real-World Input Data
- Output: Interference Results based on Updated Weights in Deep Neural Networks

Trained Model

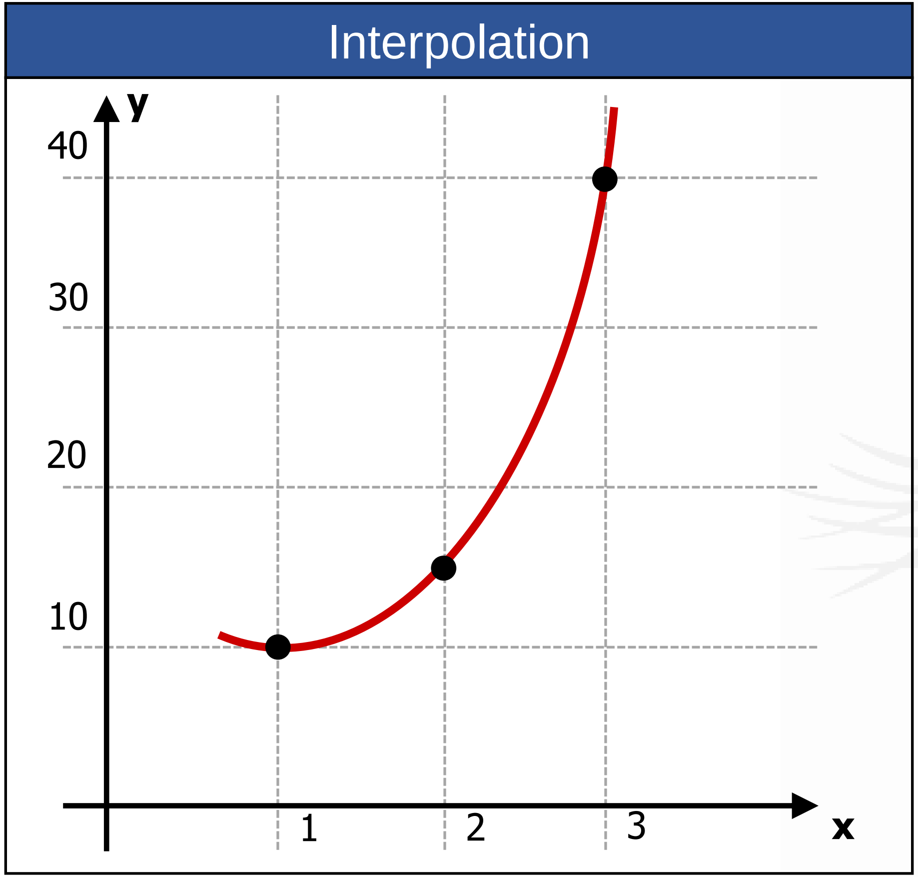


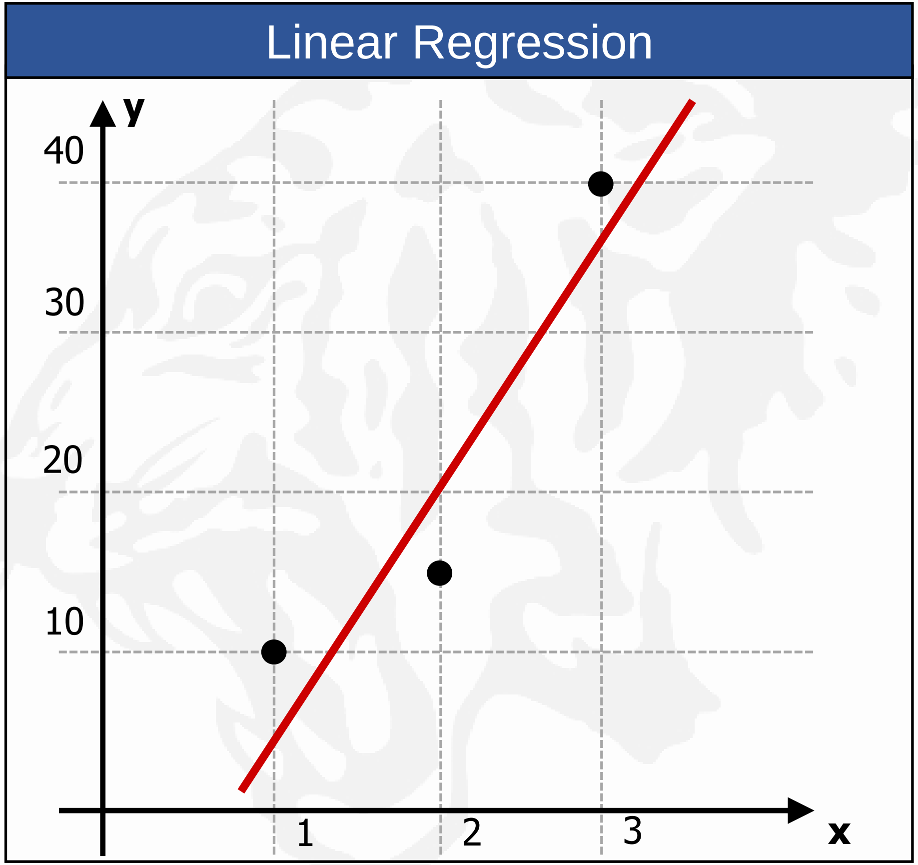
Intelligent Surveillance Platforms

INPUT: Real-Time Arrivals

OUTPUT: Inference
- Computation Results based on (i) INPUT and (ii) trained weights in units (trained model).

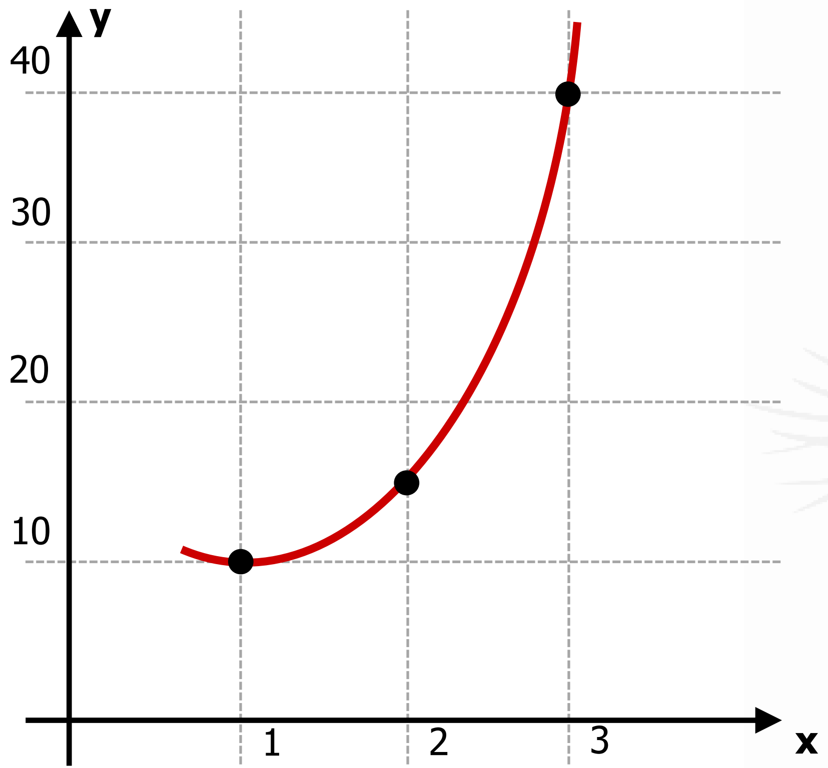
Interpolation vs. Linear Regression





Interpolation vs. Linear Regression

Interpolation



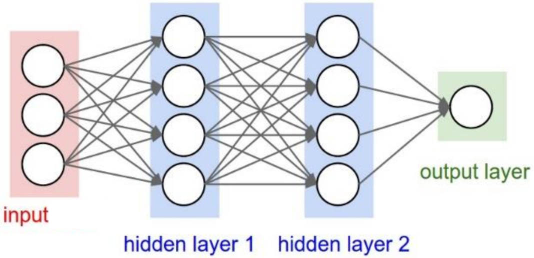
Interpolation with Polynomials

$$y = a_2x^2 + a_1x^1 + a_0$$

where three points are given.
→ Unique coefficients (a_0, a_1, a_2) can be calculated.

Is this related to **Neural Network Training**?

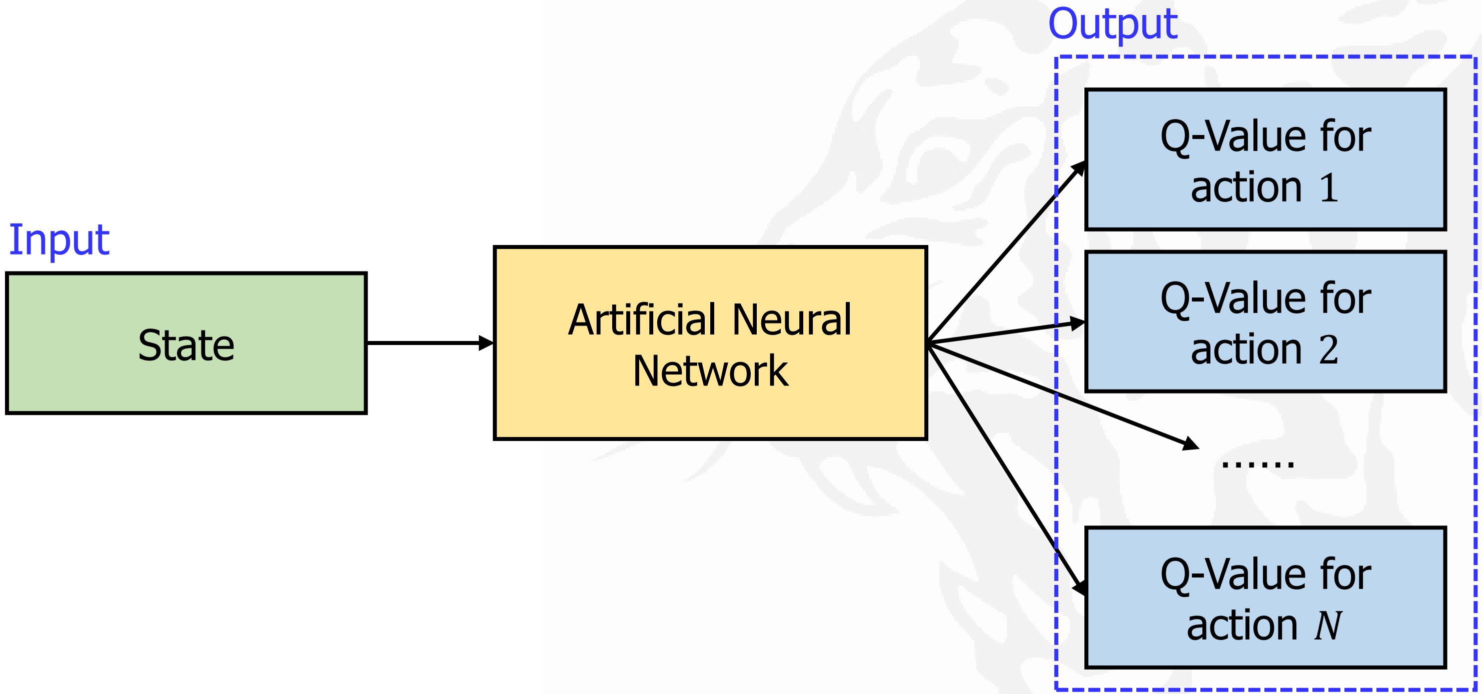
Interpolation and Neural Network Training



$$Y = a(a(a(X \cdot W_1 + b_1) \cdot W_2 + b_2) \cdot W_o + b_o)$$

where training data/labels (X: data, Y: labels) are given.
→ Find $W_1, b_1, W_2, b_2, W_o, b_o$
→ This is the mathematical meaning of neural network training.
→ **Function Approximation**
→ The most well-known function approximation with neural network: **Deep Reinforcement Learning**

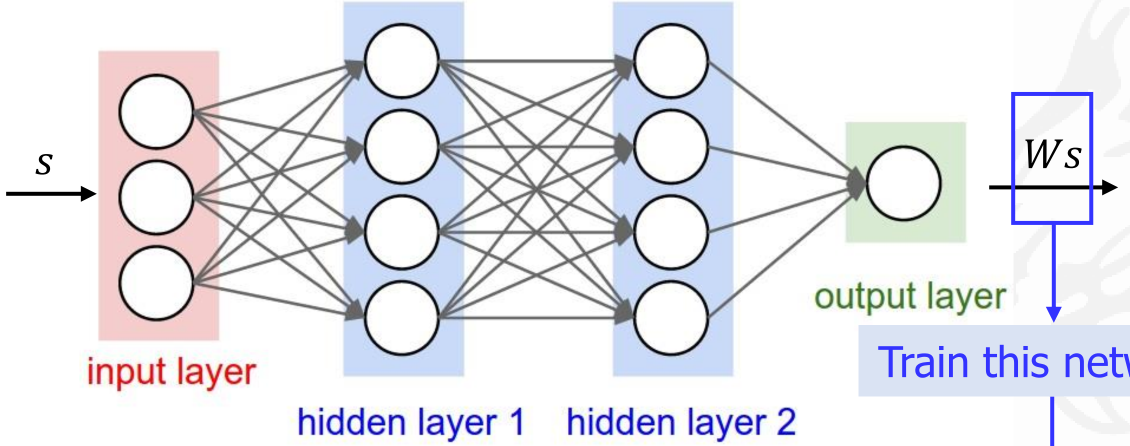
- It is inefficient to make the Q-table for each state-action pair.
  → ANN is used to **approximate the Q-function**.



- Q-Network Training (Linear Regression)

H(x)=Wx

Cost(W)= $\frac{1}{m}\sum_{i=1}^{m}\left(Wx^i - y^i\right)^2$



Train this network to approximate optimal Q, i.e., Q^*

$\boxed{Ws \approx Q^*}$

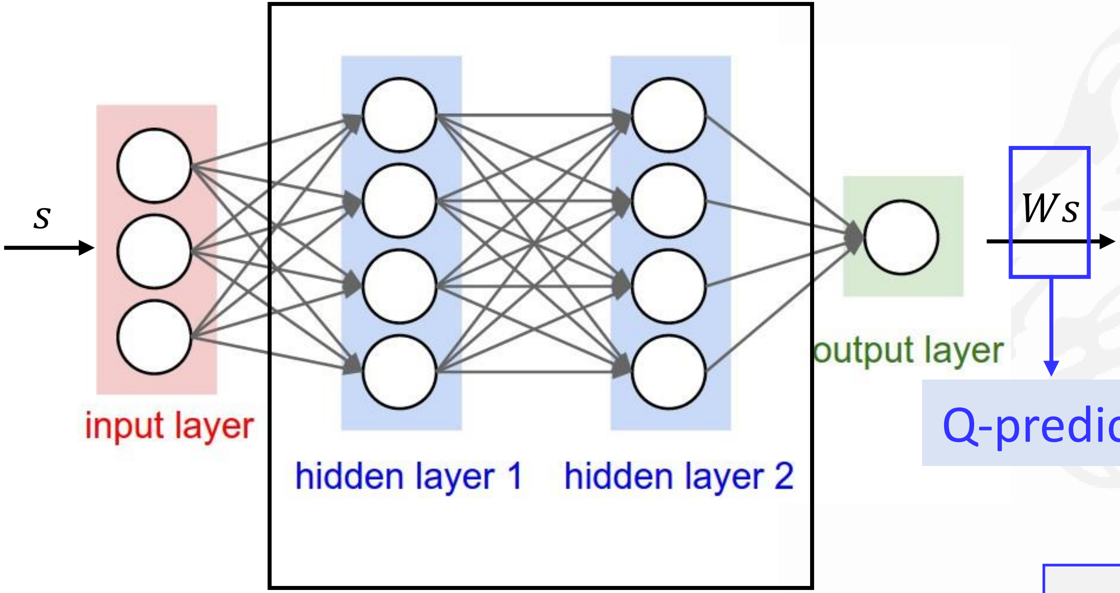
Q-Network

- Q-Network Training (Linear Regression)

$$\text{Cost}(W) = (Ws - y)^2$$

$$y = r + \gamma \max Q(s')$$

Q^*



Q-predict: $\hat{Q}$

Approximating Q^* using θ

Weights, θ

$$Ws = \hat{Q}(s, a|\theta) \approx Q^*(s, a)$$

- Q-Network Training (Linear Regression)

$$Ws = \hat{Q}(s,a|\theta) \approx Q^*(s,a)$$

Approximating Q^* using θ

$$\min_{\theta} \sum_{t=0}^{T} \Big[\underbrace{\hat{Q}(s_t,a_t|\theta)}_{\hat{Q}(s,a|\theta)} - \underbrace{\Big(r_t + \gamma \max_{a'} \hat{Q}(s_{t+1},a'|\theta) \Big)}_{Q^*} \Big]^2$$

Q-Network

Algorithm 1 Deep Q-learning

Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$

If preprocessing is not needed, $\emptyset(s) = s$

ε-greedy

Learning

$$\text{Set } y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$$

    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

Play Atari with Deep Reinforcement Learning

Q-Network

y-label

Set $\boxed{y_j} = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$

Perform a gradient descent step on $\boxed{\left(y_j - Q(\phi_j, a_j; \theta)\right)^2}$ according to equation 3

Cost function

Target/Label $\rightarrow$ Q^*

Ws $\rightarrow$ $\hat{Q}(s, a|\theta)$

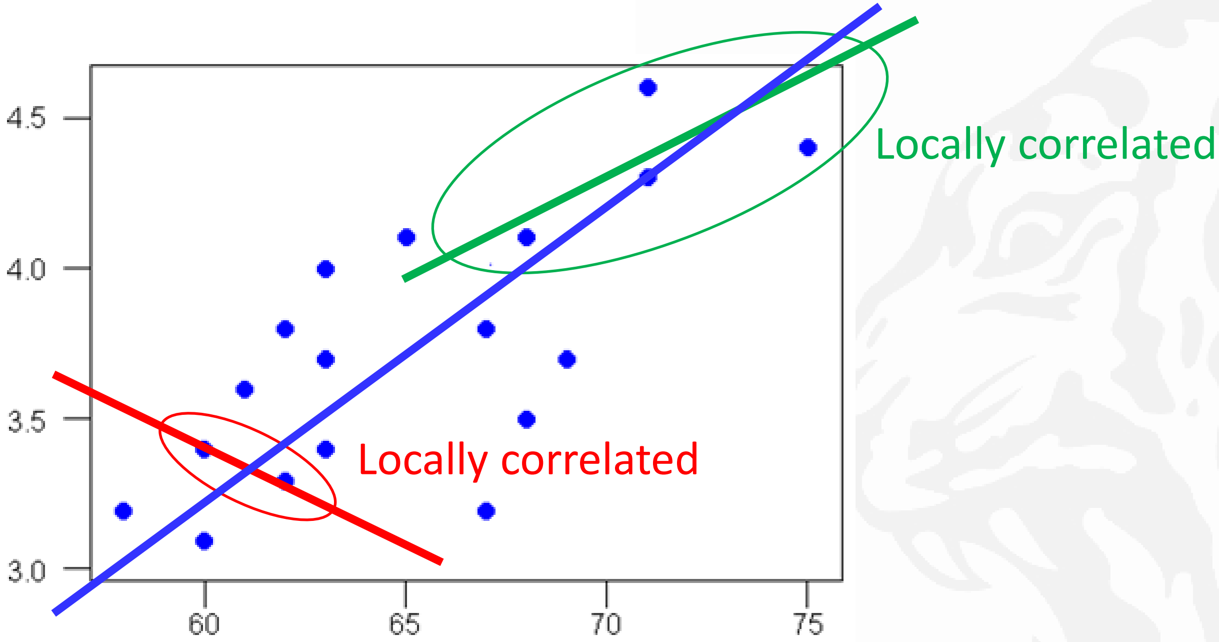
Deep Q-Network (DQN)

$$\min_{\theta} \sum_{t=0}^{T} \left[\hat{Q}(s_t, a_t|\theta) - \left(r_t + \gamma \max_{a'} \hat{Q}(s_{t+1}, a'|\theta) \right) \right]^2$$

- Converges to Q^* using table lookup representation
- However, **diverges** using neural networks due to
  - Correlations between samples → [Issue #1]
  - Non-stationary targets → [Issue #2]

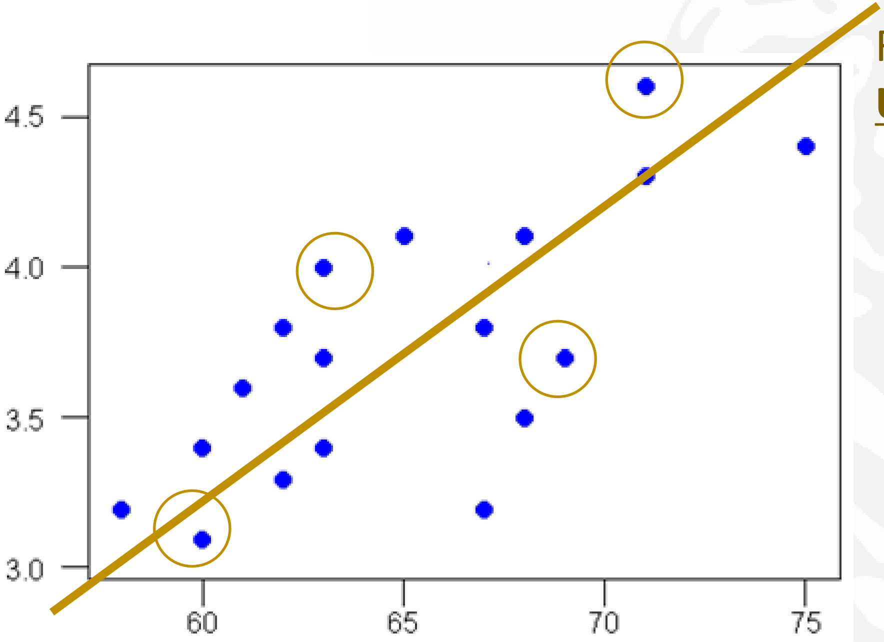
Tutorial by Google DeepMind: Deep Reinforcement Learning

- [Issue #1] Correlations between Samples



- **Solution) Capture and Replay**
  - Store learning states in buffers → random sampling and learning

- [Issue #1] Correlations between Samples
  - **Capture and Replay →** Experience Replay
    - Store learning states in buffers → random sampling and learning



Random Sampling Results are **Uniformed Distributed**.

- [Issue #2] Non-Stationary Targets

Target

$$\min_{\theta} \sum_{t=0}^{T} \left[\hat{Q}(s_t, a_t|\theta) - \underbrace{\left(r_t + \gamma \max_{a'} \hat{Q}(s_{t+1}, a'|\theta) \right)}_{\text{Target}} \right]^2$$

- Both sides uses same network θ.
  Thus, if our Q_predict is trained, our target is consequently updated.
  → **Non-stationary targets**.
- **Solution) Separate Networks** → create a target network

- [Issue #2] Non-Stationary Targets

Target

$$\min_{\theta} \sum_{t=0}^{T} \left[\hat{Q}(s_t, a_t | \theta) - \left(r_t + \gamma \max_{a'} \hat{Q}(s_{t+1}, a' | \theta) \right) \right]^2$$

$$\min_{\theta} \sum_{t=0}^{T} \left[\hat{Q}(s_t, a_t | \theta) - \left(r_t + \gamma \max_{a'} \hat{Q}(s_{t+1}, a' | \bar{\theta}) \right) \right]^2$$

And periodic update!

Course Outline

Reinforcement Learning

Q-Learning
MDP

Deep Reinforcement Learning

DQN

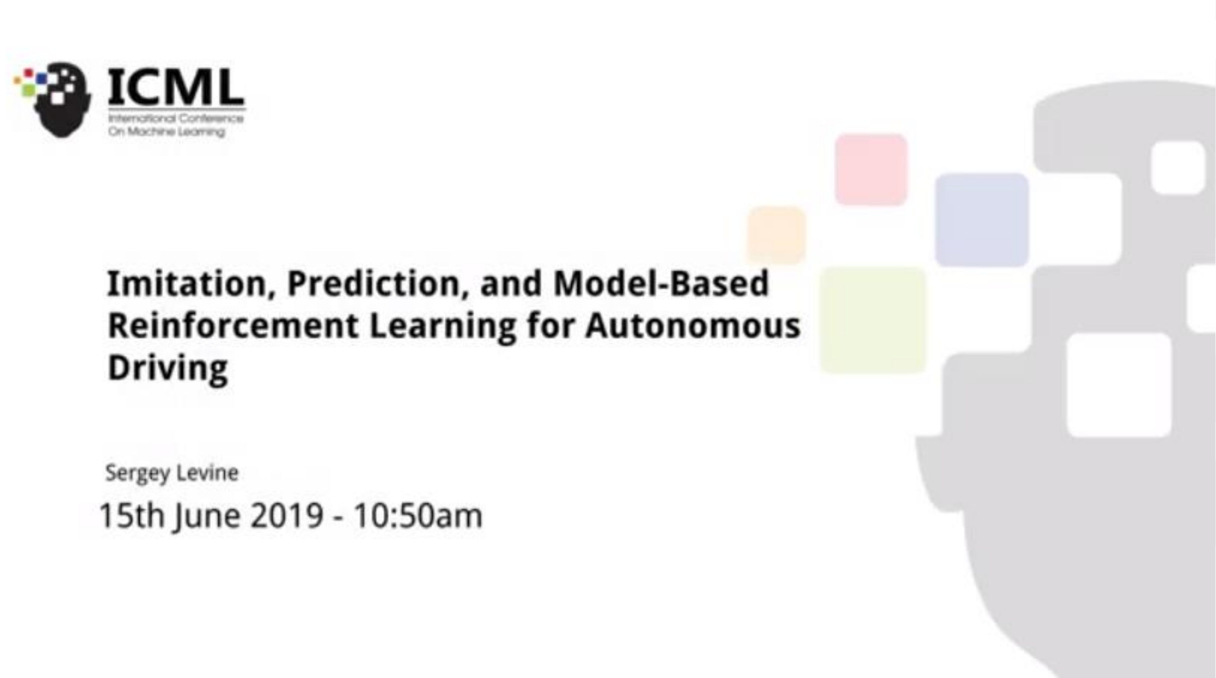
Imitation Learning

- ICML 2018 Tutorial
  - https://sites.google.com/view/icml2018-imitation-learning/

Imitation Learning Tutorial ICML 2018

Introduction

- ICML 2019 Tutorial
  - https://slideslive.com/38917941/imitation-prediction-and-modelbased-reinforcement-learning-for-autonomous-driving



- Gameplay

Pro-Gamer

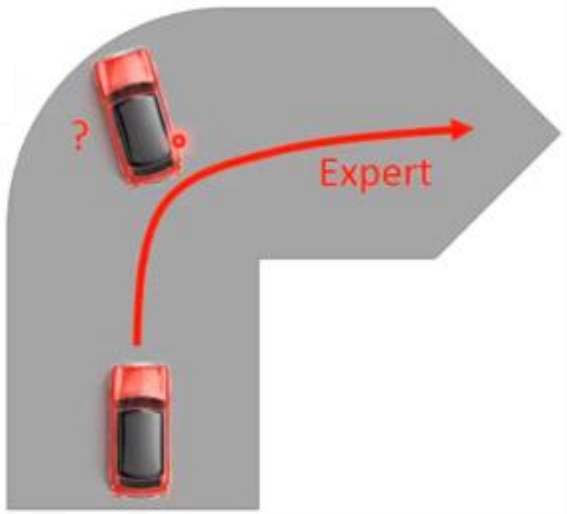
Trained Agent

The goal of Imitation Learning is to train a policy to mimic **the expert's demonstrations**

- **Behavior Cloning**
  - Define $P^* = P(s|\pi^*)$ (distribution of states visited by **expert**)
  - **Learning objective**

$$\operatorname{argmin}_{\theta} E_{(s,a_E)\sim P^*} L(a_E, \pi_{\theta}(s))$$
$$L\left(a_E, \pi_{\theta}(s)\right) = \left(a_E - \pi_{\theta}(s)\right)^2$$

  - **Discussion**
    - Works well when P^* close to the distribution of states visited by π_{θ}
    - **Minimize 1-step deviation error** along the expert trajectories



Imitation Learning Applications: Starcraft2

- Starcraft2

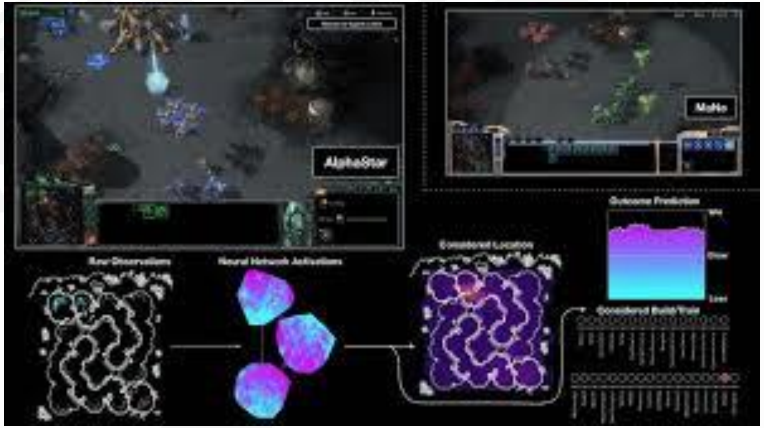
States: s = **minimap**, **screen**
Action: a = **select**, **drag**
Training set: $D = \{\tau \coloneqq (s, a)\}$ from expert
Goal: learn $\pi_\theta(s) \to$ a

States: s
Action: a
Policy: π_θ
- Policy maps states to actions : $\pi_\theta(s) \to \mathrm{a}$
- Distributions over actions : $\pi_\theta(s) \to P(\mathrm{a})$

State Dynamics: P(s′|s,a)
- Typically not known to policy
- Essentially the simulator/environment

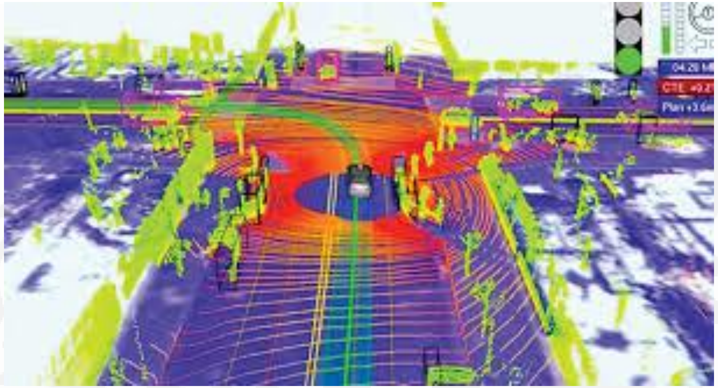
Rollout: sequentially execute $\pi_\theta(s_0)$ on initial state
- Produce trajectories $\boldsymbol{\tau}$

$\mathbf{P}(\boldsymbol{\tau}|\boldsymbol{\pi})$**:** distribution of trajectories induced by a policy
$\mathbf{P}(\mathbf{s}|\boldsymbol{\pi})$**:** distribution of states induced by a policy



- Autonomous Driving Control

States: s = **sensors**
Action: a = **steering wheel**, **brake**, …
Training set: $D = \{\tau \coloneqq (s, a)\}$ from expert
Goal: learn $\pi_\theta(s) \rightarrow$ a



- PPF/RFTN Injection Control in Medicine

States: s = **BIS**, **BP**, …
Action: a = **PPF**, **RFTN**, …
Training set: $D = \{\tau \coloneqq (s, a)\}$ from expert
Goal: learn $\pi_\theta(s) \rightarrow$ a

Thank you for your attention!

- More questions?
  - joongheon@korea.ac.kr
- More details?
  - https://joongheon.github.io/