Artificial Intelligence and Mobility Lab

Reinforcement Learning Algorithms and Applications

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

Introduction and Preliminaries

Deep Reinforcement Learning Theory

Deep Reinforcement Learning Implementation

Imitation Learning

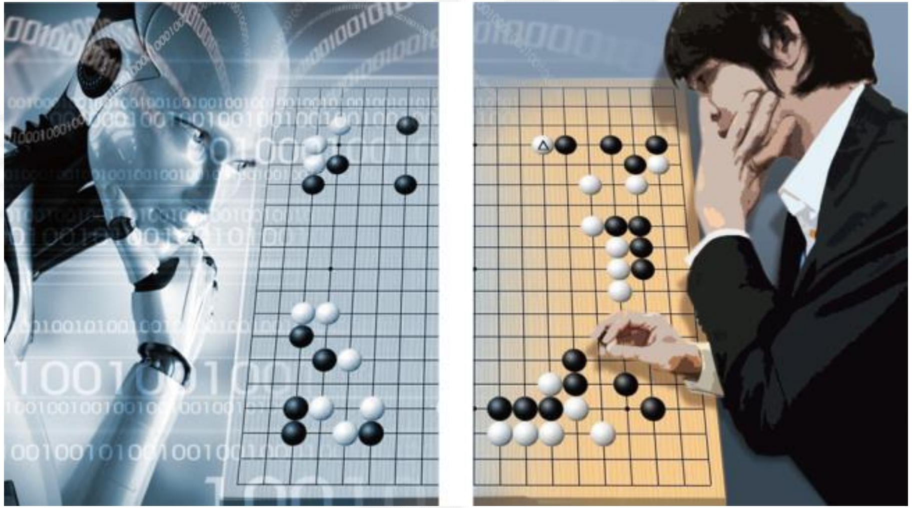
Autonomous Mobility Applications

- **Introduction and Applications**
- Q-Learning
- Dynamic Programming
- Markov Decision Process

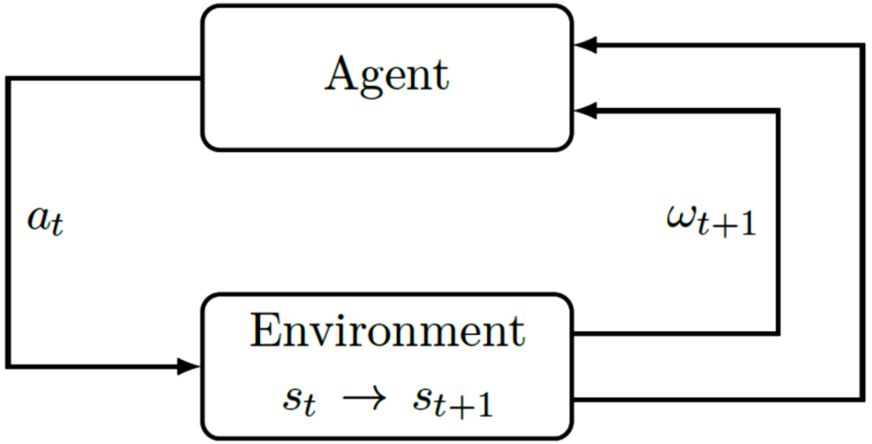
- Brief History and Successes
  - Minsky's PhD thesis (1954): Stochastic Neural-Analog **Reinforcement** Computer
  - Analogies with animal learning and psychology
  - Job-shop scheduling for NASA space missions (Zhang and Dietterich, 1997)
  - Robotic soccer (Stone and Veloso, 1998) – part of the world-champion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

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

- 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 $\omega_{t+1} \in \Omega$



Artificial Intelligence and Mobility Lab

Introduction and Preliminaries

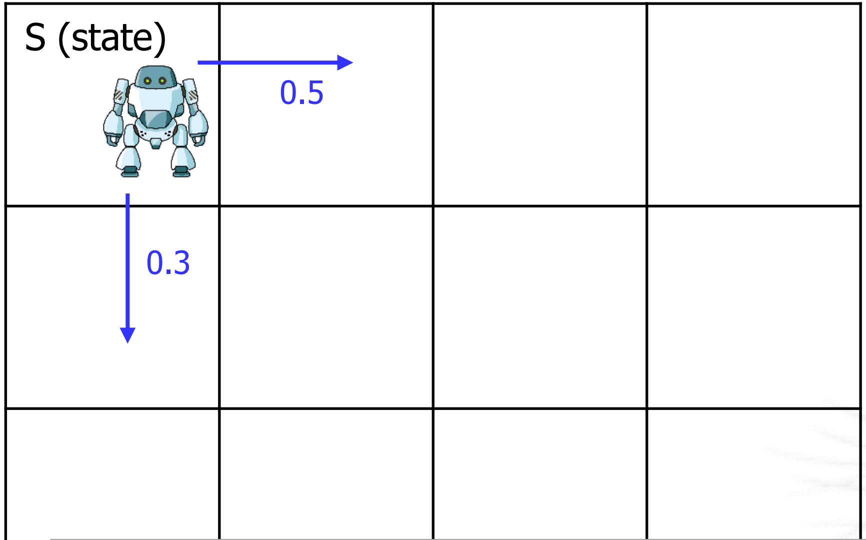
- Introduction and Applications
- **Q-Learning**
- Dynamic Programming
- Markov Decision Process

Deep Reinforcement Learning Theory

Deep Reinforcement Learning Implementation

Imitation Learning

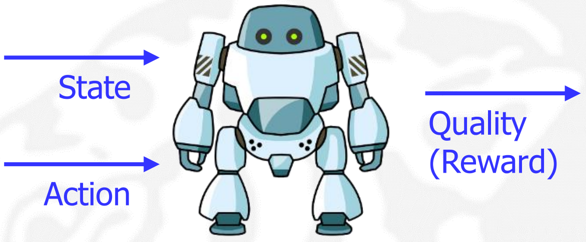
Autonomous Mobility Applications



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)

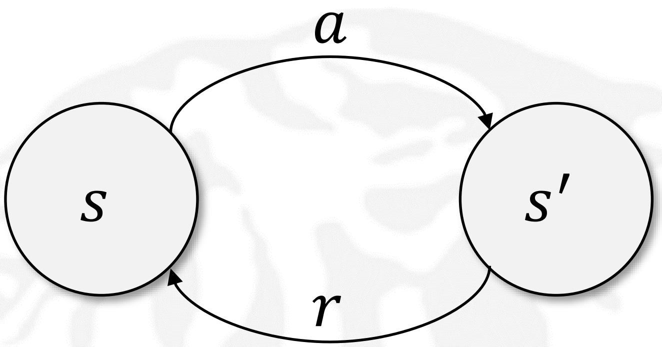


Q (state, action)

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 $Q(s', a')$?

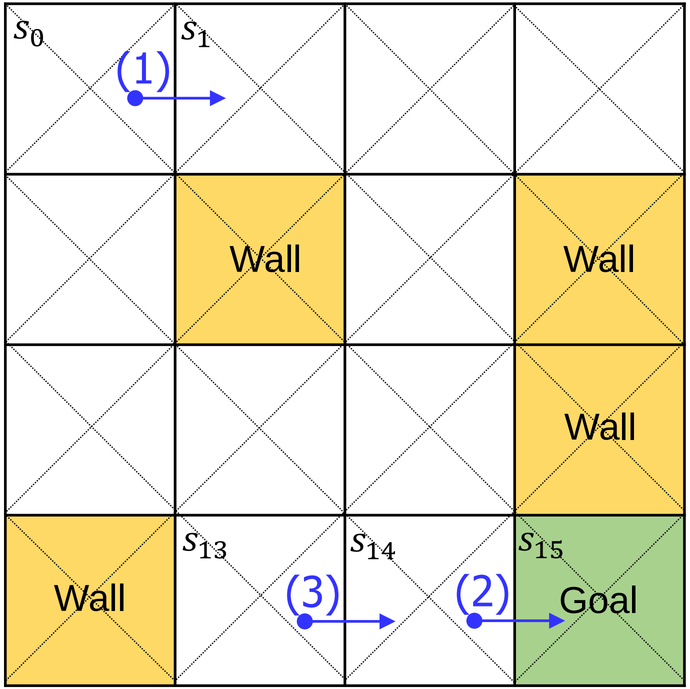
$$Q(s,a) = r + \max_{a'} Q(s', a')$$

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)



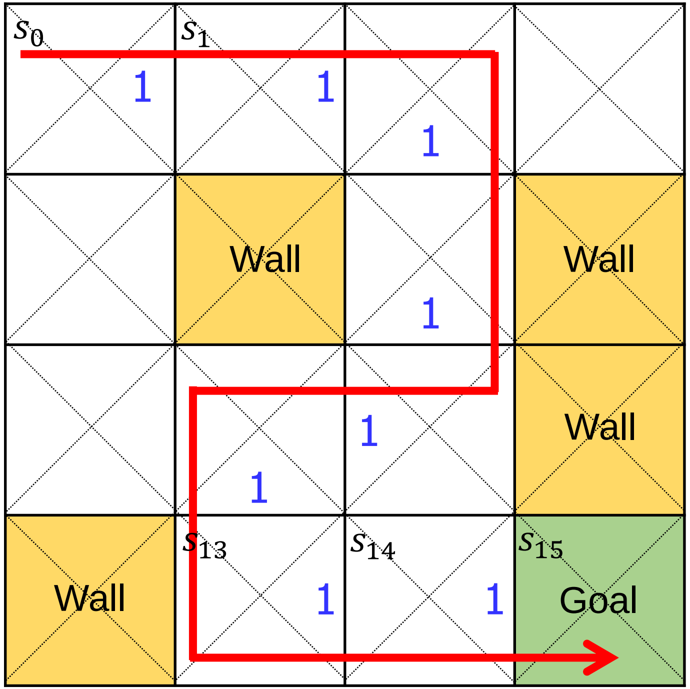
- Initial Status
  - All 64 Q values are 0,
  - Reward are all zero except $r_{s_{15},L} = 1$

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

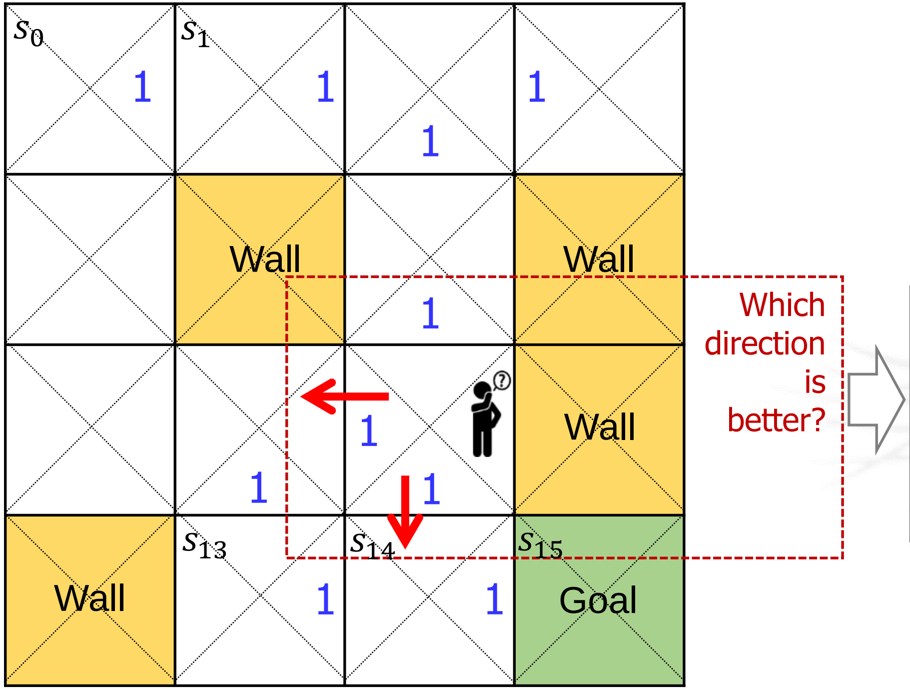
- For (2), from s_{14} to s_{15} (goal)
  - $Q(s_{14}, a_R) = r + \max_a Q(s_{15}, a) = 1 + \max\{0,0,0,0\} = 1$

- 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

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

$$0 < \gamma \leq 1$$

- For each s, a, initialize table entry $Q(s,a) \leftarrow 0$
- Observe 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 $Q(s,a)$ as follows:

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

  - $s \leftarrow s'$

Q-Learning with Exploit and Exploration: ε-Greedy

Finding the Best Restaurant
- Try the best one during weekdays.
- Try new ones during weekends.

ε-Greedy

```
e=0.1
IF (random < e)
        a = random;
ELSE
        a = argmax(Q(s,a));
```

Decaying ε-Greedy

```
for i in range (1000); e=0.1 / (i+1);
IF (random < e)
        a = random;
ELSE
        a = argmax(Q(s,a));
```

Q-Learning with Exploit and Exploration: Add Random Noise

Finding the Best Restaurant

McDonald's	KFC	Burger King	Lotteria
43	35	41	31

Add Random Noise

McDonald's	KFC	Burger King	Lotteria
52	**56**	53	32

Add Random Noise

a = argmax(Q(s,a) + **random_values**);

Add Decaying Random Noise

for i in range (1000);
a = argmax(Q(s,a) + random**/(i+1)**);

Artificial Intelligence and Mobility Lab

Introduction and Preliminaries

- Introduction and Applications
- Q-Learning
- **Dynamic Programming**
- Markov Decision Process

Deep Reinforcement Learning Theory

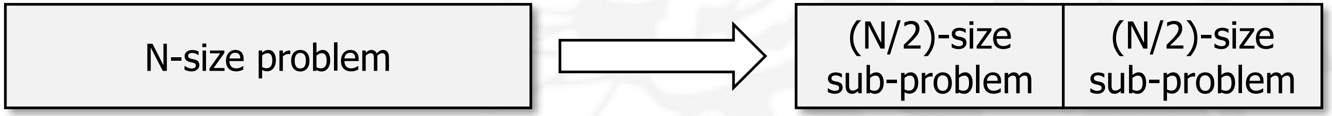
Deep Reinforcement Learning Implementation

Imitation Learning

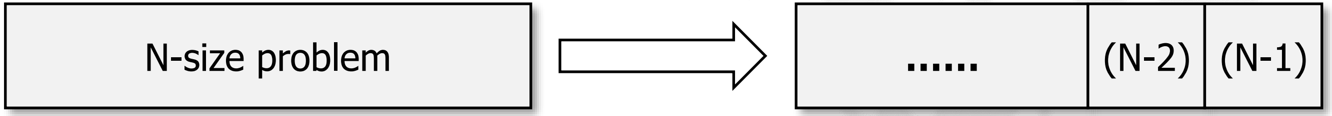
Autonomous Mobility Applications

- **Introduction**
- Applications
  - Fibonacci Number
  - Pascal's Triangle
  - Knapsack Problem

- Dynamic Programming
  - The term "programming" stands for "planning".
  - Usually used for optimization problems
  - In order to solve large-scale problems, (i) divide the problems into several sub-problems, (ii) solve the sub-problems, and (iii) obtain the solution of the original problem based on the solutions of the sub-problems, recursively
  - Difference from divide-and-conquer
    - Divide-and-conquer



    - Dynamic programming



- Introduction
- Applications
    - <u>**Fibonacci Number**</u>
    - Pascal’s Triangle
    - Knapsack Problem

- Fibonacci Number
  - 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, ……
  - $F(N) = F(N-2) + F(N-1)$ where $F(1) = 0$ and $F(2) = 1$ (Recursive!)

$$F(7) = F(5) + F(6)$$

$$F(8) = F(6) + F(7)$$

……	3	5	8	13	……

Table

$\boxed{F(7) = 8}$

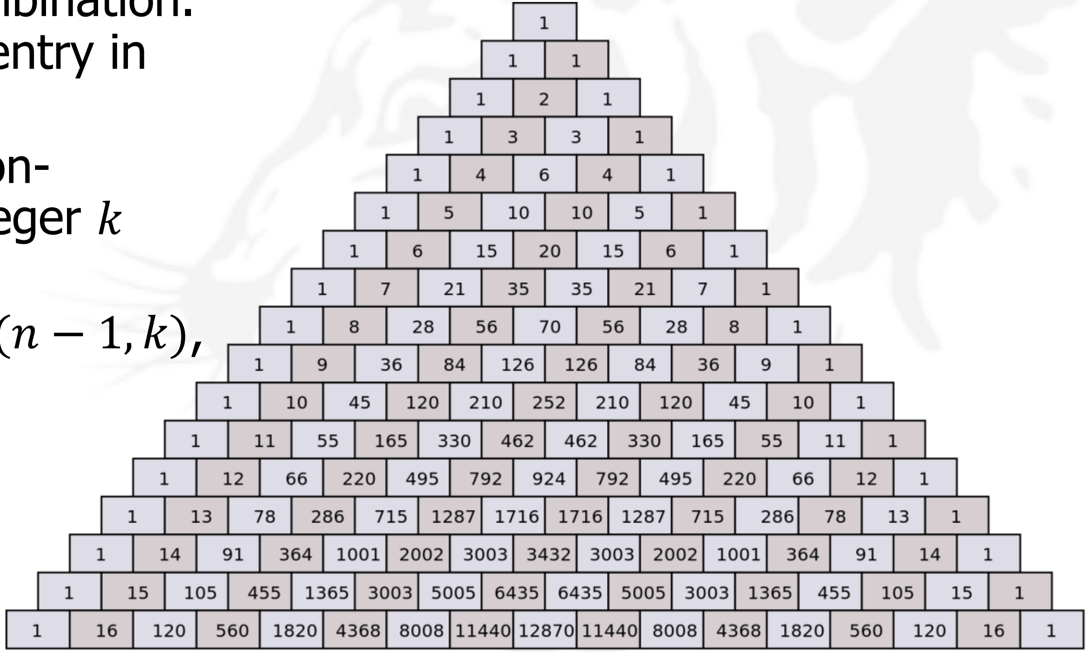
$\boxed{F(8) = 13}$

- Introduction
- Applications
  - Fibonacci Number
  - **<u>Pascal's Triangle</u>**
  - Knapsack Problem

- Pascal's Triangle
  - The entry in the n-th row and k-th column of Pascal's triangle is denoted $C(n,k)$ where C stands for combination. Note that the unique nonzero entry in the topmost row is $C(0,0) = 1$.
  - General Formulation for any non-negative integer n and any integer k between 0 and n:

$$C(n,k) = C(n-1,k-1) + C(n-1,k),$$

  - Recursive!



Artificial Intelligence and Mobility Lab

Introduction and Preliminaries

- Introduction and Applications
- Q-Learning
- Dynamic Programming
- **Markov Decision Process**

Deep Reinforcement Learning Theory

Deep Reinforcement Learning Implementation

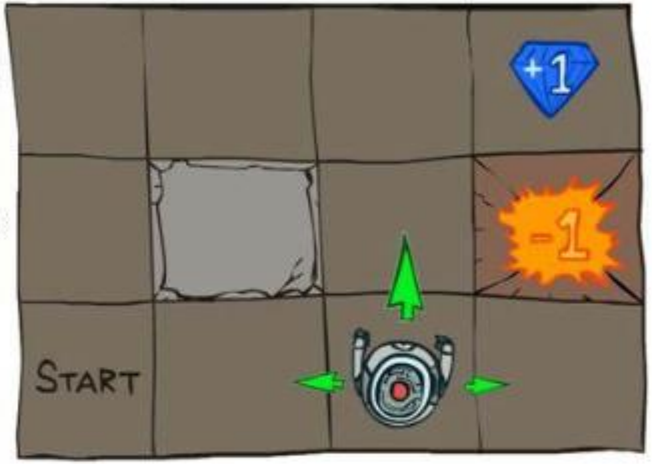
Imitation Learning

Autonomous Mobility Applications

- 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

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

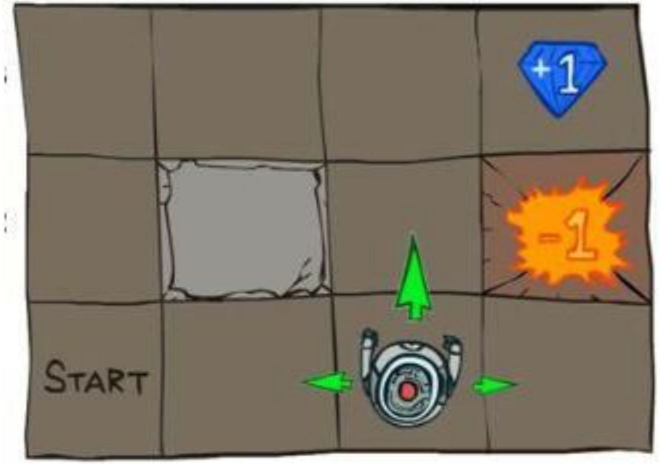
- Markov Decision Process (MDP) Components: <S, A, R, T, γ>
  - **S: Set of states**
  - A: Set of actions
  - R: Reward function
  - T: Transition function
  - γ: Discount factor



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

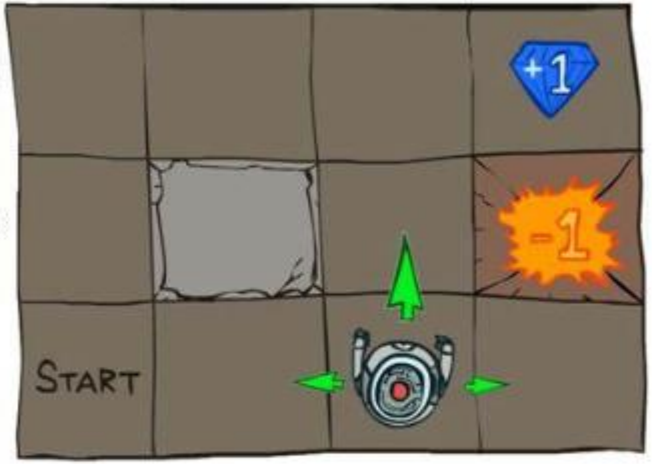
- s_0: starting state
- s: current state
- s': next state
- s_t: state at time t

- 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



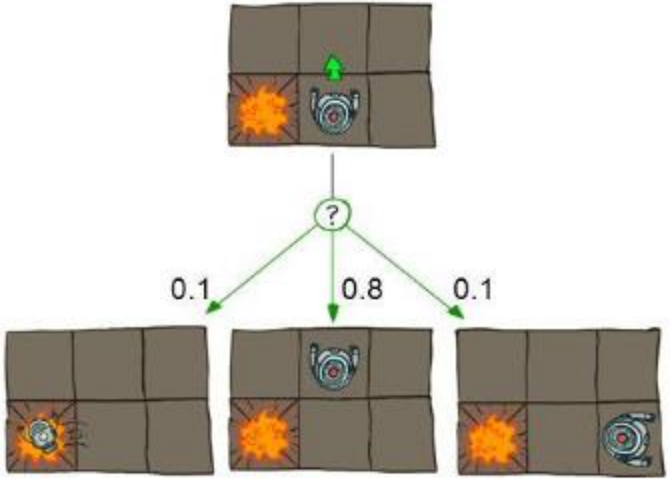
S: location (x, y) if the maze is a 2D grid
A: move up, down, left, or right
- $s \rightarrow s'$

- Markov Decision Process (MDP) 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?

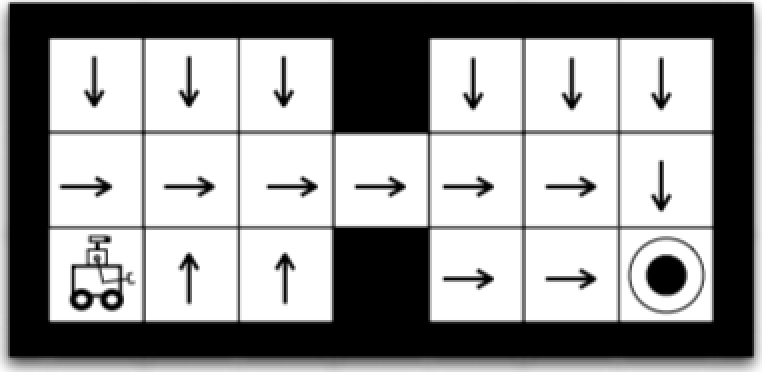
- $T = P(s'|s, a)$

- 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)]

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



Our goal:
Find π that maximizes expected return!

- 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

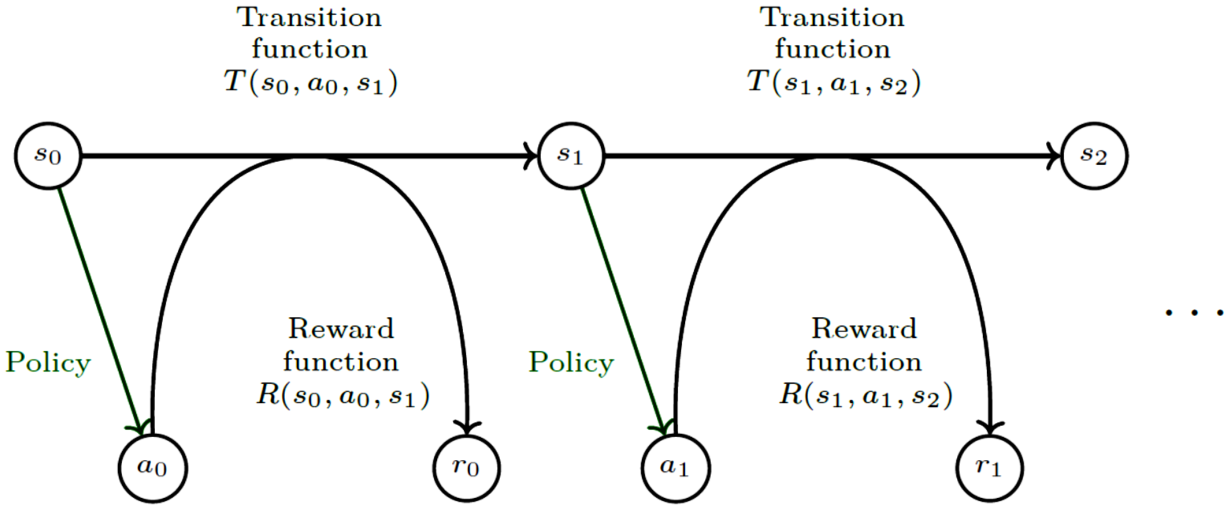
$$\pi^*(s) = \max_{\pi} R^{\pi}(s)$$

  - 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.

- **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 $R(s_t, a_t, s_{t+1}) \in R$.



Introduction and Preliminaries

Deep Reinforcement Learning Theory

- **Deep Neural Network Summary**
- Deep Q-Network (DQN)

Deep Reinforcement Learning Implementation

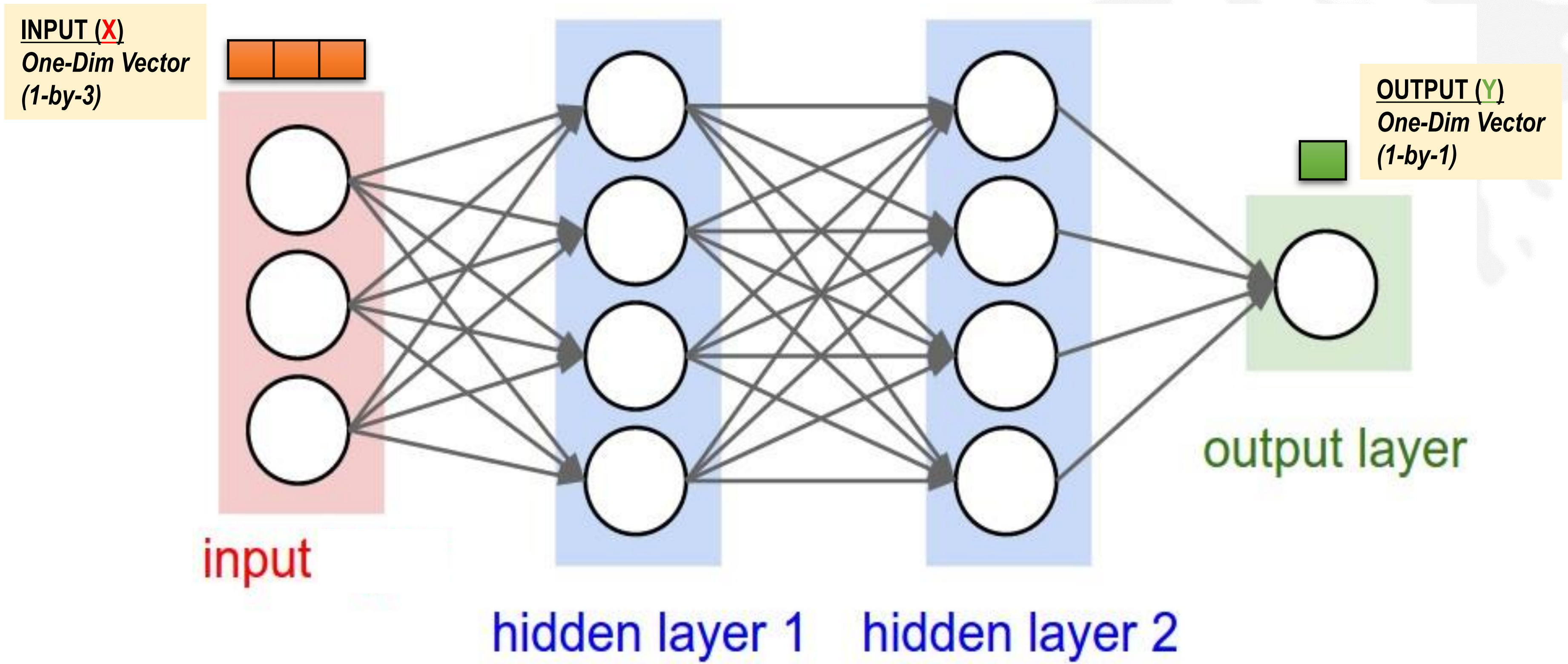
Imitation Learning

Autonomous Mobility Applications

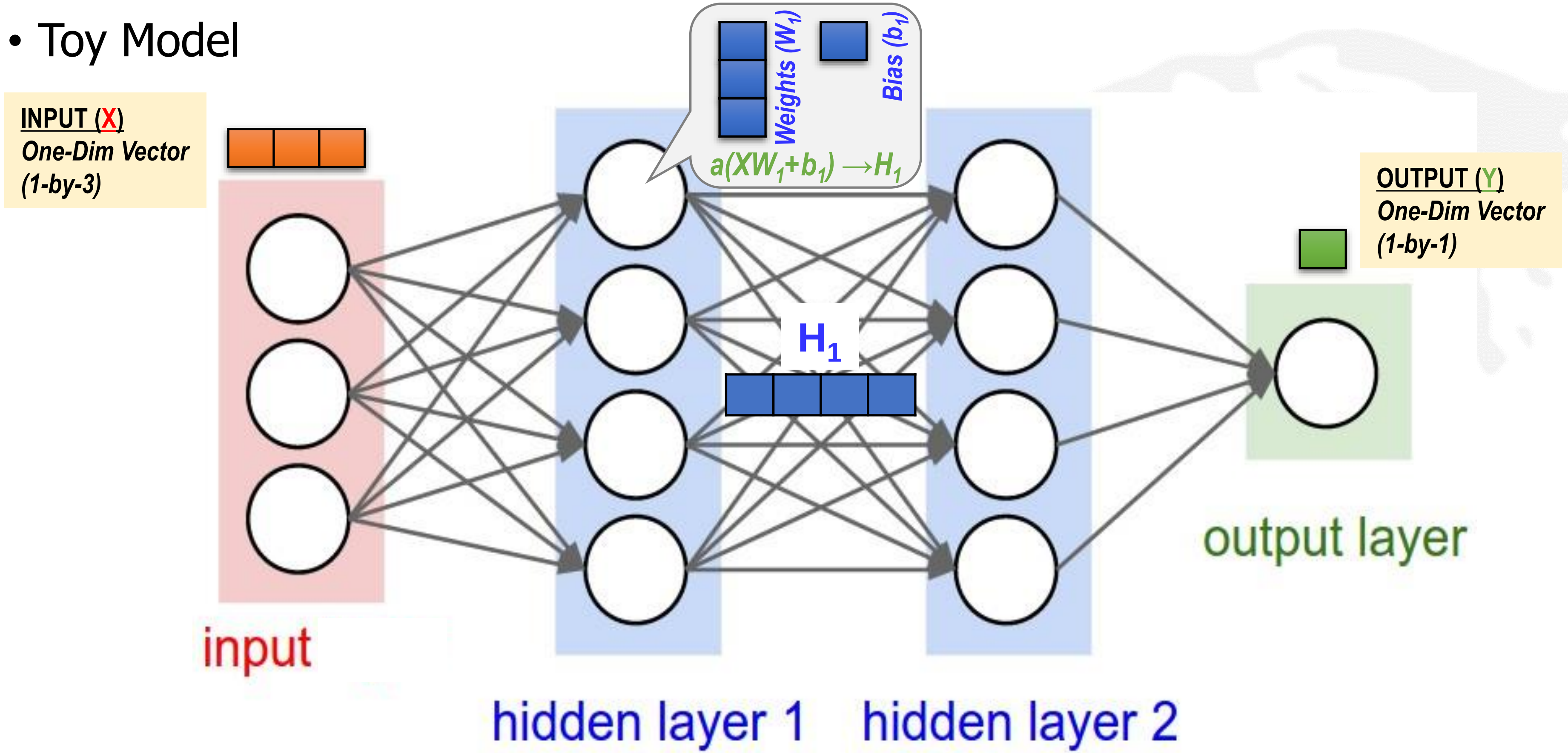
Reinforcement Learning (RL) vs. Deep Reinforcement Learning (DRL)

	RL (i.e., MDP)	DRL
Pros	Optimal	Fast Computation
Cons	Pseudo-Polynomial	Non-Optimal

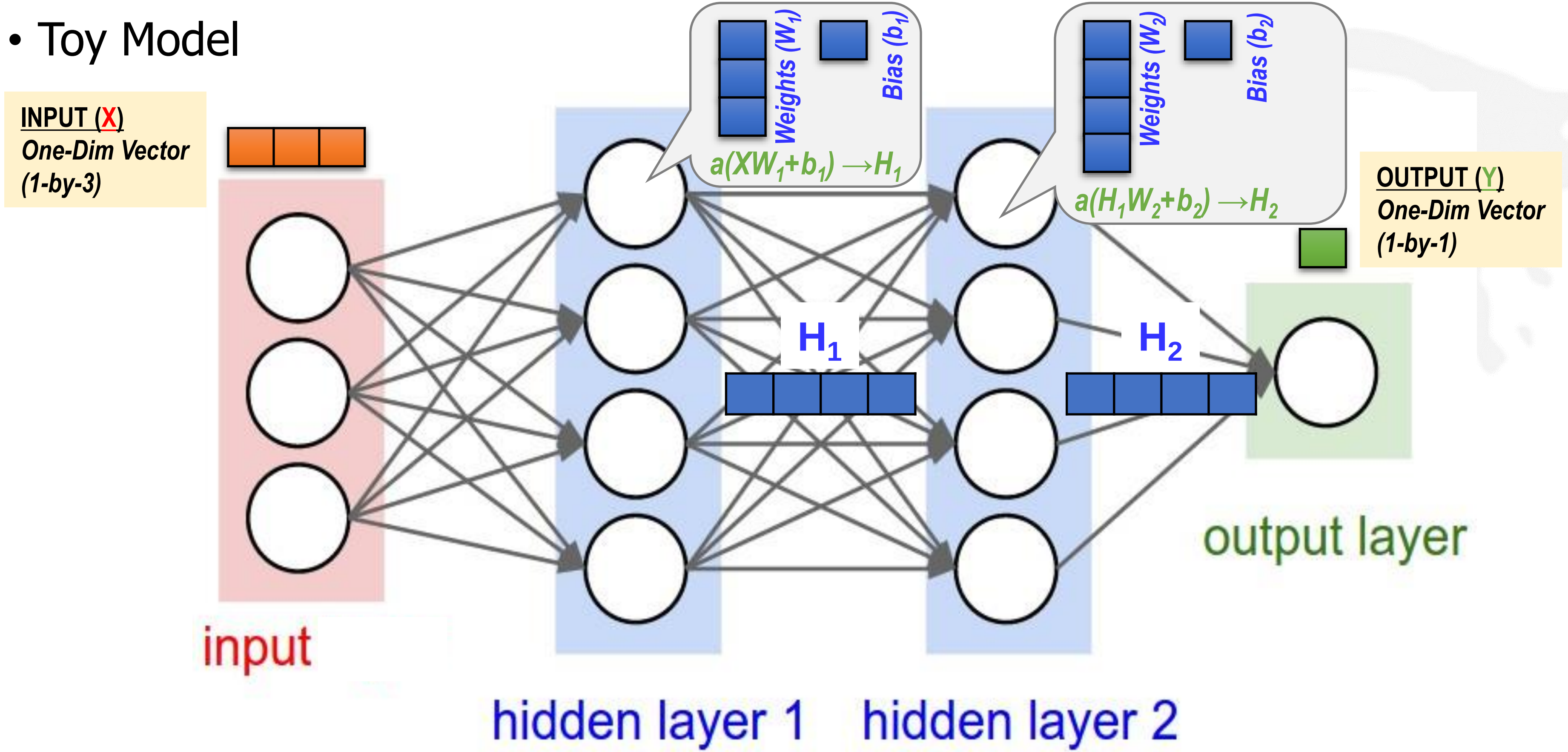
- Toy Model


INPUT (X)
One-Dim Vector
(1-by-3)
input
hidden layer 1
hidden layer 2
output layer
OUTPUT (Y)
One-Dim Vector
(1-by-1)

- Toy Model


INPUT (X)
One-Dim Vector
(1-by-3)
Weights (W_1)
Bias (b_1)
$a(XW_1+b_1) \rightarrow H_1$
H_1
OUTPUT (Y)
One-Dim Vector
(1-by-1)
input
hidden layer 1
hidden layer 2
output layer

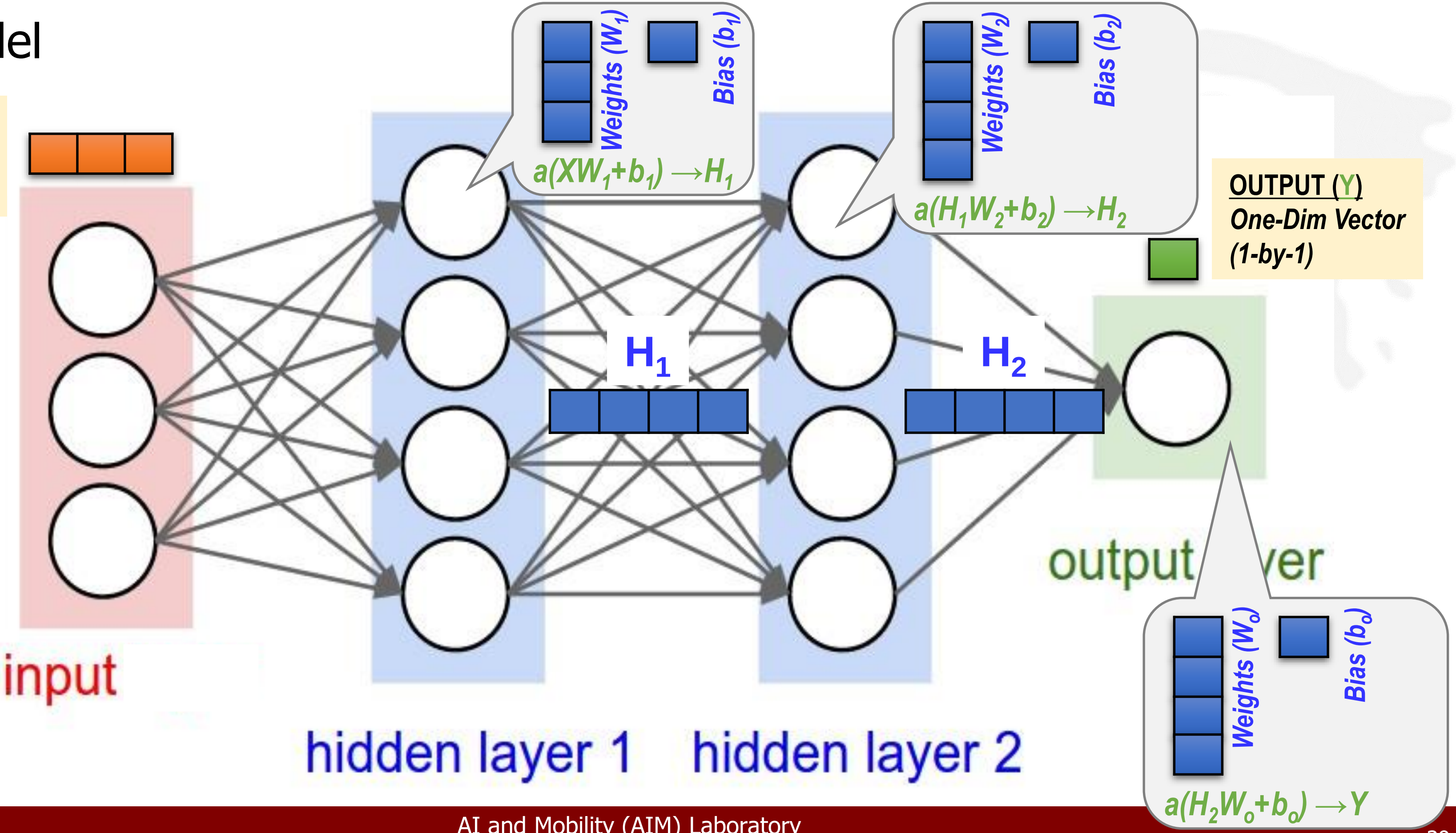
- Toy Model



INPUT (X)
One-Dim Vector
(1-by-3)
Weights (W_1)
Bias (b_1)
$a(XW_1+b_1) \rightarrow H_1$
Weights (W_2)
Bias (b_2)
$a(H_1W_2+b_2) \rightarrow H_2$
OUTPUT (Y)
One-Dim Vector
(1-by-1)
H_1
H_2
input
hidden layer 1
hidden layer 2
output layer

- Toy Model

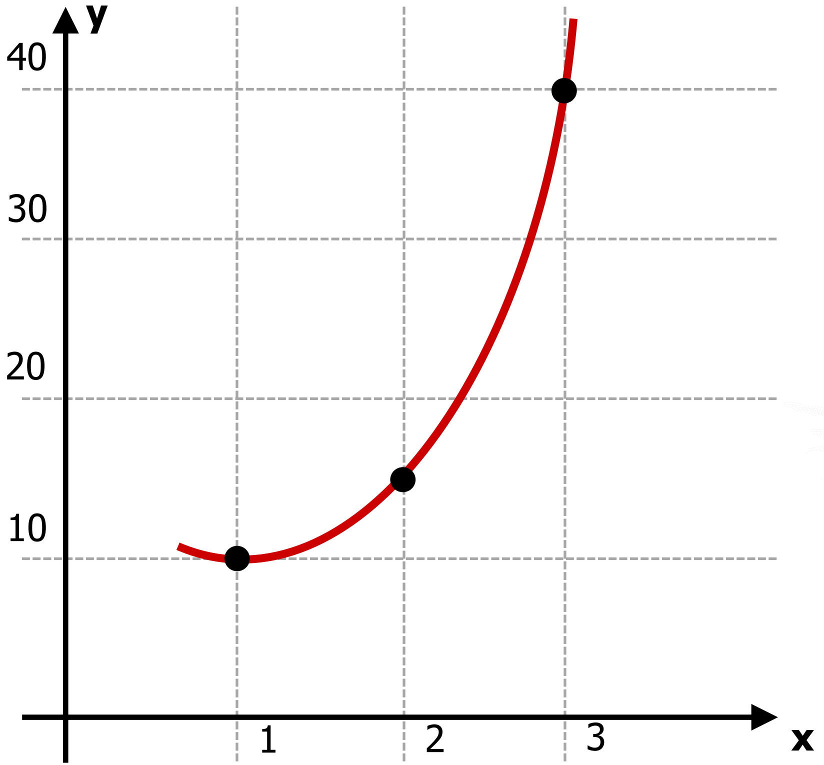
INPUT (X)
One-Dim Vector
(1-by-3)



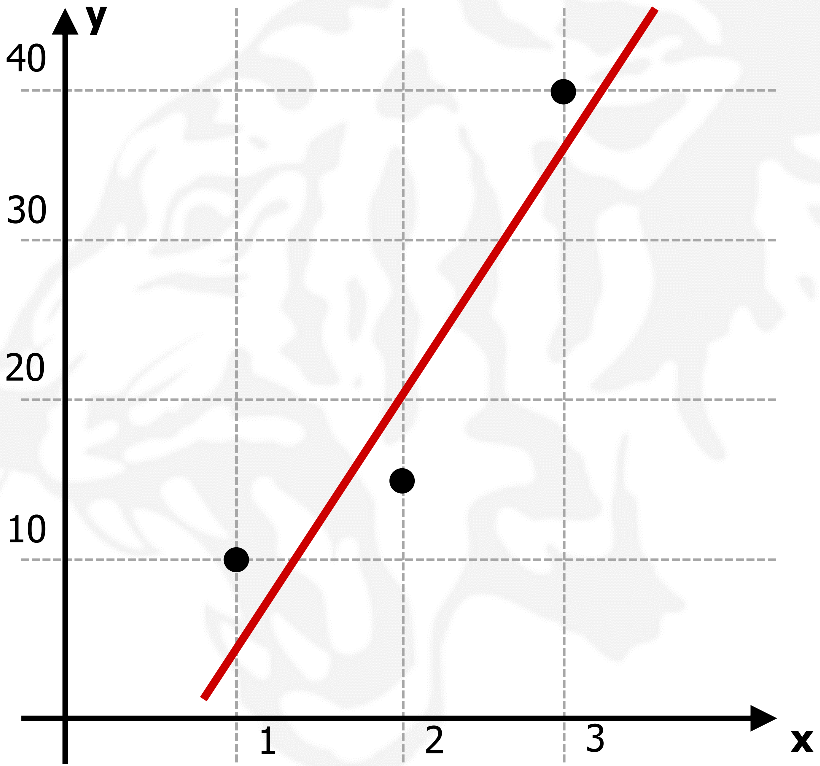
OUTPUT (Y)
One-Dim Vector
(1-by-1)

Interpolation vs. Linear Regression

Interpolation

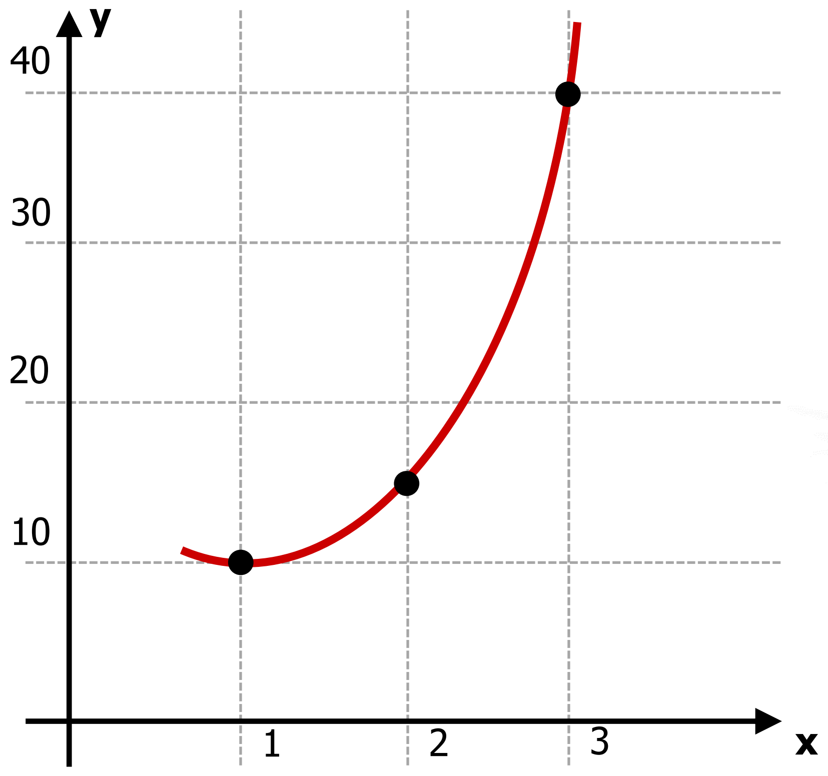


Linear Regression



Interpolation vs. Linear Regression

Interpolation

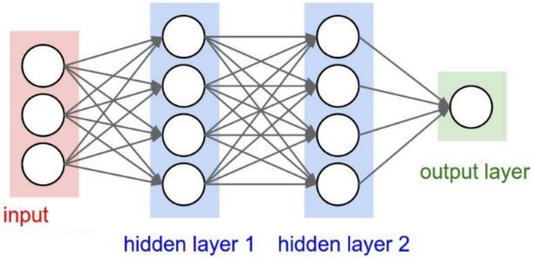


Interpolation with Polynomials

$$y = a_2 x^2 + a_1 x^1 + a_0$$

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

Is this related to
Neural Network Training?



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

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**

	RL (i.e., MDP)	DRL
Pros	Optimal	Fast Computation
Cons	Pseudo-Polynomial	Non-Optimal

Introduction and Preliminaries

Deep Reinforcement Learning Theory

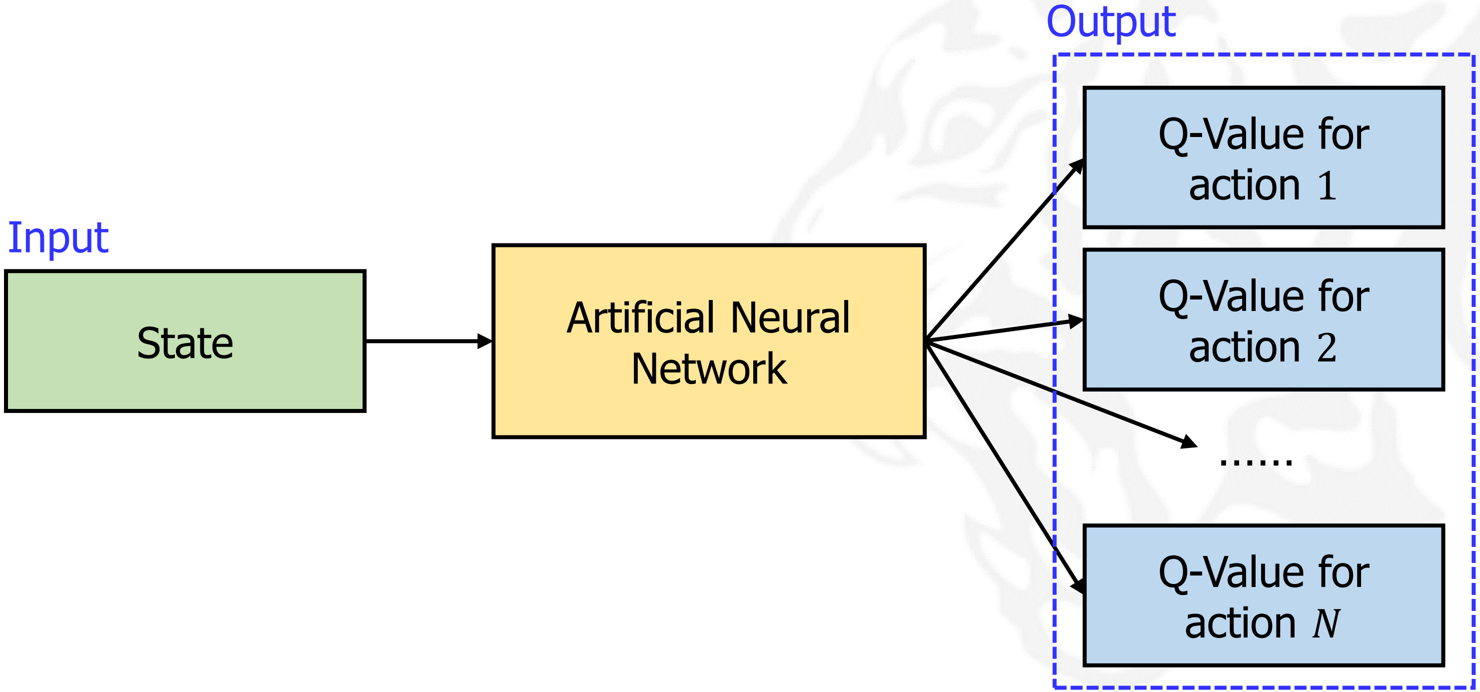
- Deep Neural Network Summary
- **Deep Q-Network (DQN)**

Deep Reinforcement Learning Implementation

Imitation Learning

Autonomous Mobility Applications

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



Lecture Roadmap

Introduction and Preliminaries

Deep Reinforcement Learning Theory

Deep Reinforcement Learning Implementation

Imitation Learning

Autonomous Mobility Applications

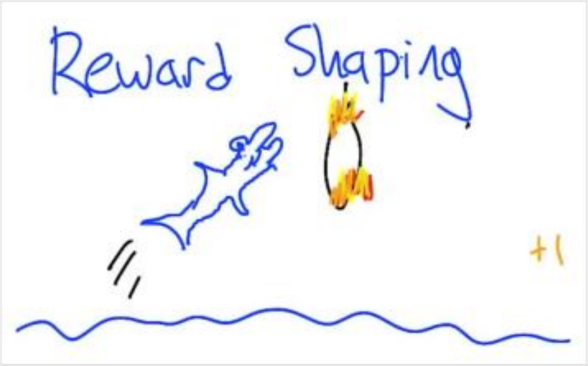
- Gameplay

Pro-Gamer

Trained Agent

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

- Problems of RL



1. Reward Shaping
2. Safe Learning
3. Exploration process

Imitation Learning **handles with** these problems through the demonstration of the experts.

Inverse Reinforcement Learning (IRL)

RL

Environment Model(MDP)

Reward Function R → Reinforcement Learning $\arg\max_\pi \mathrm{E}[\sum_t \gamma^t R(s_t)|\pi]$ → Optimal Policy π

IRL

Environment Model(MDP)

Reward Function R ← Inverse Reinforcement Learning ← Optimal Policy π

Expert Policy

R that explains Expert Trajectories

Expert Trajectories

$s_0, a_0, s_1, a_1, s_2, a_2, \cdots$

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

$$\operatorname{argmin}_{\boldsymbol{\theta}} E_{(s,a_E)\sim P^*} L(a_E, \pi_{\boldsymbol{\theta}}(s))$$

$$L(a_E, \pi_{\boldsymbol{\theta}}(s)) = (a_E - \pi_{\boldsymbol{\theta}}(s))^2$$

  - **Discussion**
    - Works well when P^* close to the distribution of states visited by $\pi_{\boldsymbol{\theta}}$
    - **Minimize 1-step deviation error** along the expert trajectories

- Starcraft2

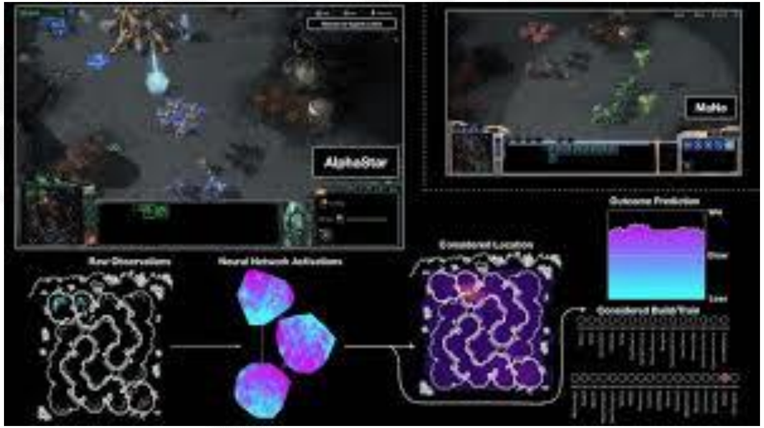
States: s = **minimap**, **screen**
Action: a = **select**, **drag**
Training set: $D = \{\tau \coloneqq (s, a)\}$ from expert
Goal: learn $\pi_\theta(s) \rightarrow$ a

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

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

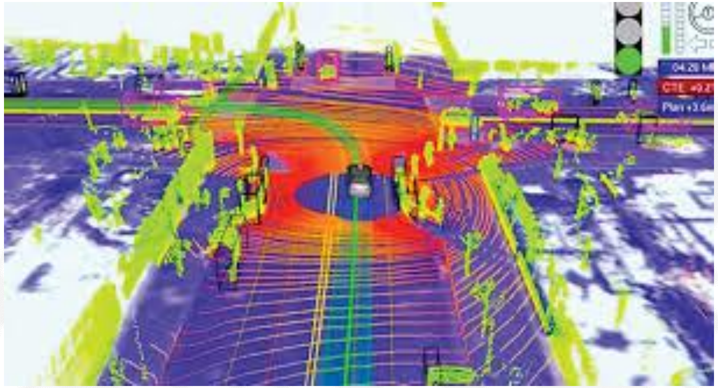
Rollout: sequentially execute $\pi_\theta(s_0)$ on initial state
- Produce trajectories $\boldsymbol{\tau}$

$\mathbf{P}(\boldsymbol{\tau}|\boldsymbol{\pi})$**:** 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

Introduction and Preliminaries

Deep Reinforcement Learning Theory

Deep Reinforcement Learning Implementation

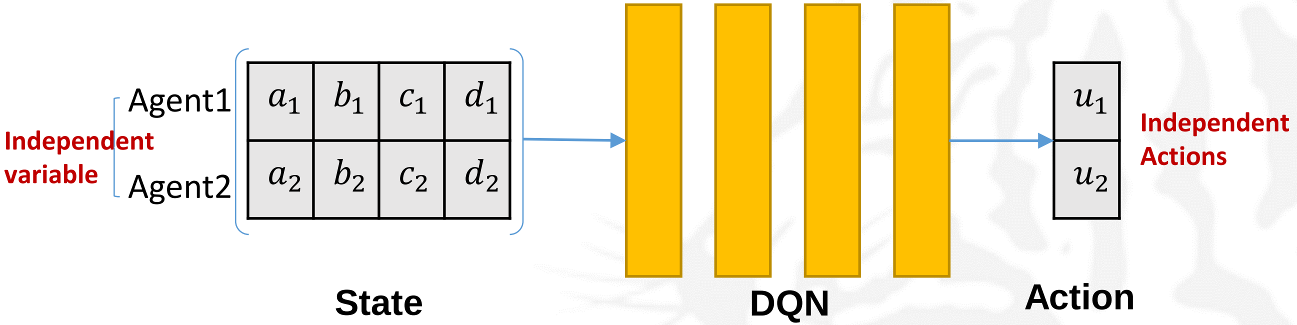
Imitation Learning

Autonomous Mobility Applications

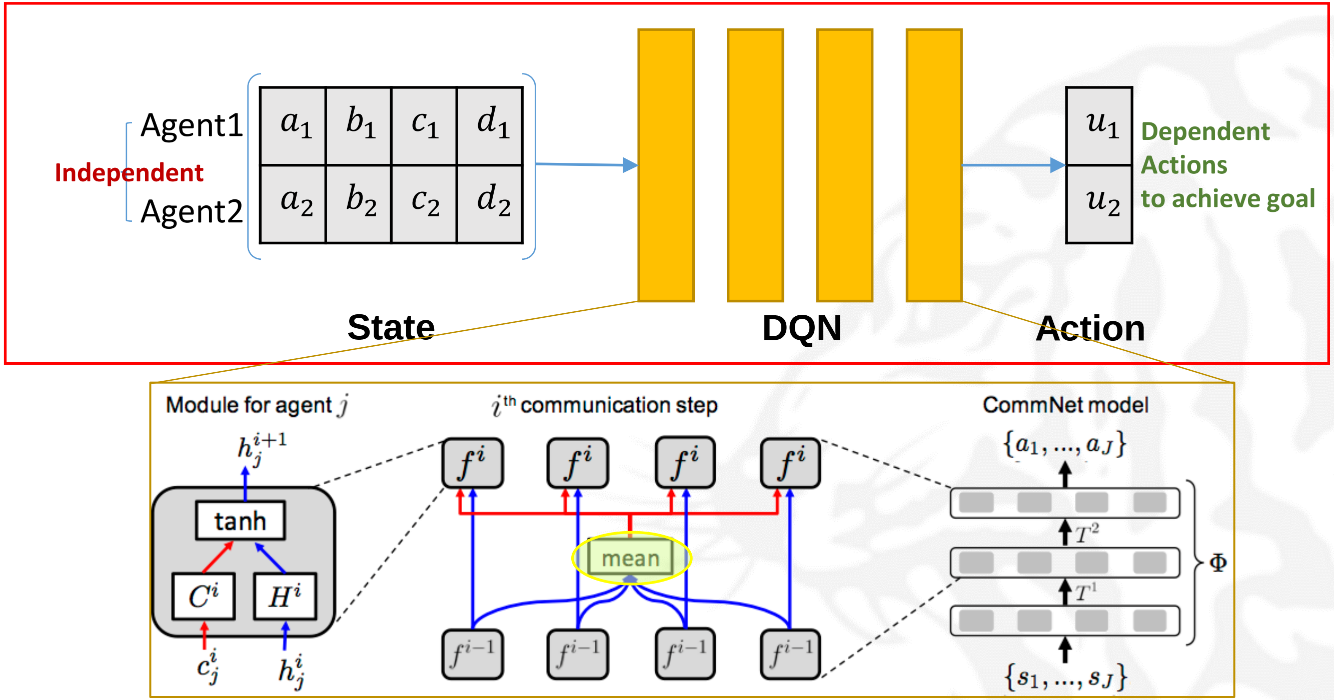
- **MADRL**
- Applications

Artificial Intelligence and Mobility Lab

- CTDE (Centralized Training and Distributed Execution)



DQN-based CommNet



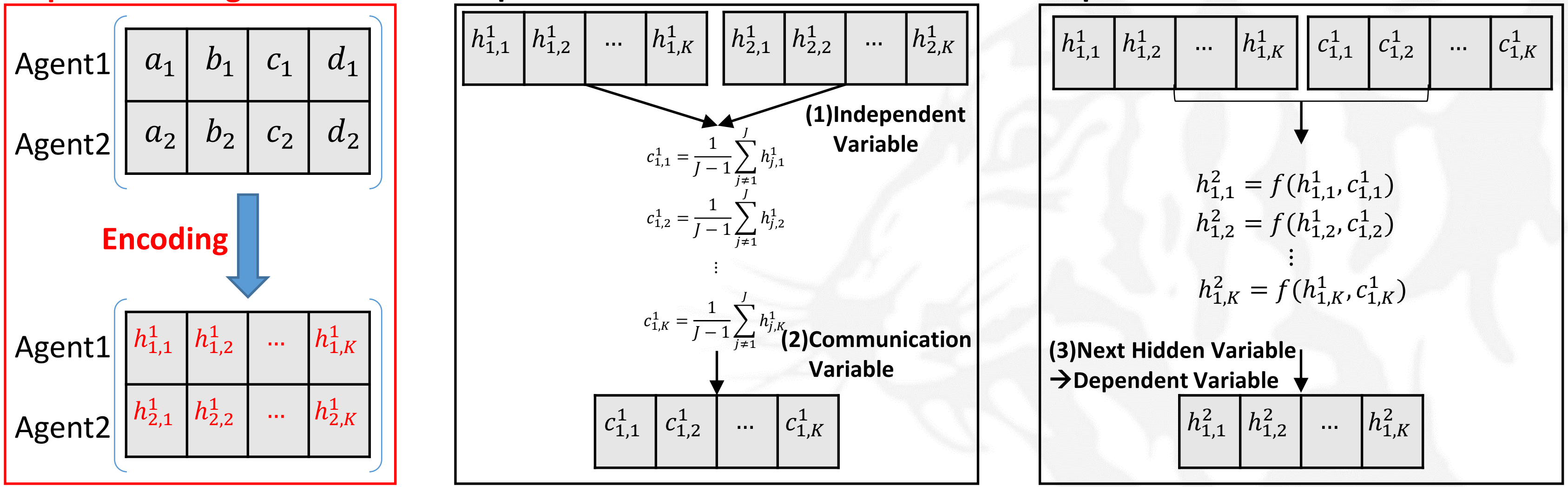
h_j^i : j-th agent's hidden state variable in i-th layer

c_j^i : j-th agent's communitive state variable in i-th layer

$$h_j^{i+1} = f^i(h_j^i, c_j^i)$$

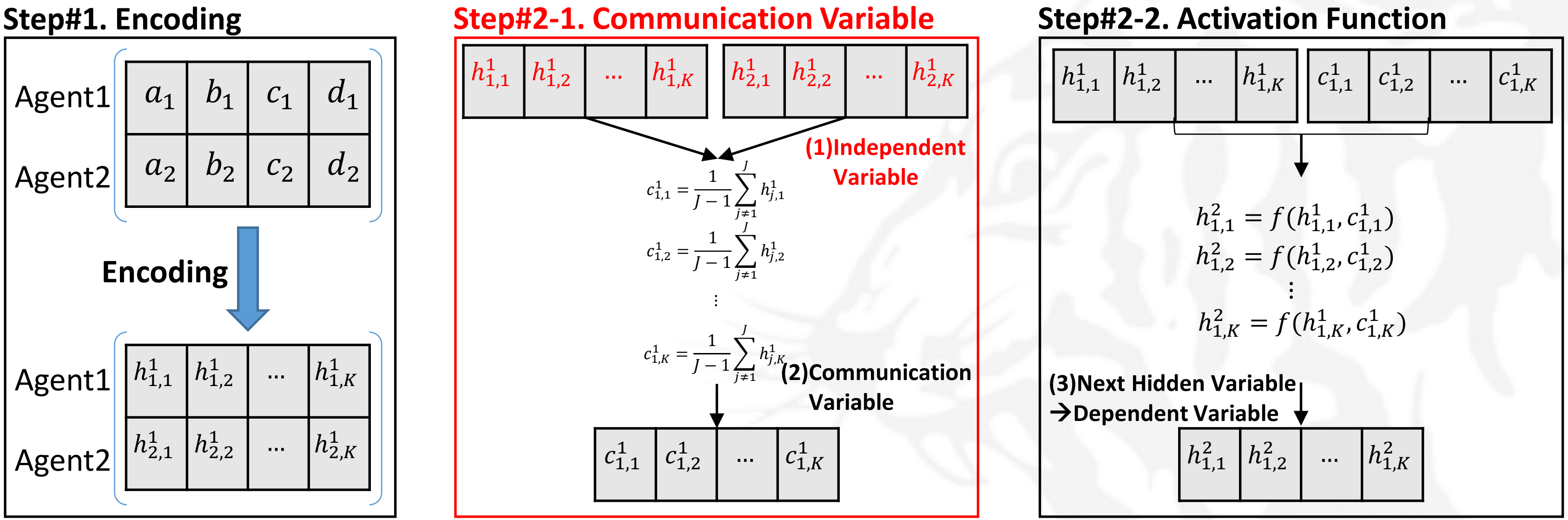
CommNet Mechanism

Step#1. Encoding **Step#2-1. Communication Variable** **Step#2-2. Activation Function**



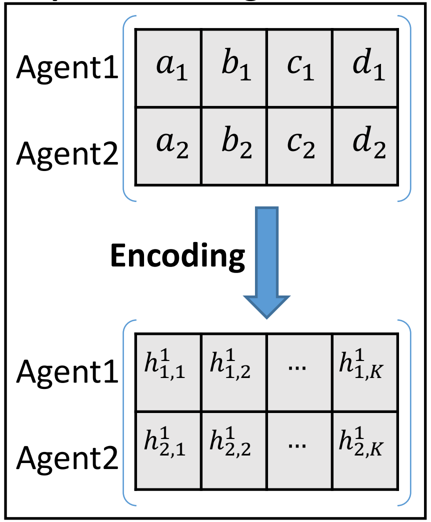
[3] S. Sukhbaatar et al., Learning Multiagent Communication with Backpropagation, *NIPS 2016*

CommNet Mechanism

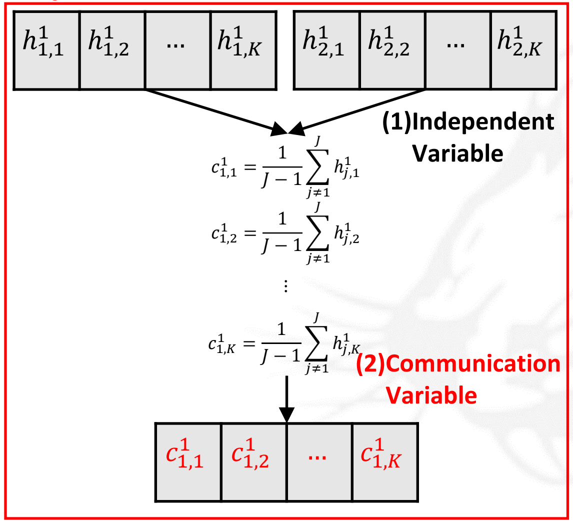


CommNet Mechanism

Step#1. Encoding



Step#2-1. Communication Variable



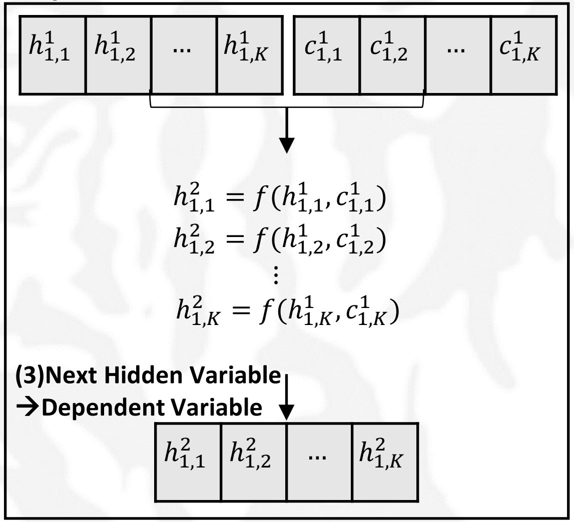
$$c^1_{1,1} = \frac{1}{J-1}\sum_{j\neq 1}^{J} h^1_{j,1}$$

$$c^1_{1,2} = \frac{1}{J-1}\sum_{j\neq 1}^{J} h^1_{j,2}$$

$$\vdots$$

$$c^1_{1,K} = \frac{1}{J-1}\sum_{j\neq 1}^{J} h^1_{j,K}$$

Step#2-2. Activation Function



$$h^2_{1,1} = f(h^1_{1,1}, c^1_{1,1})$$

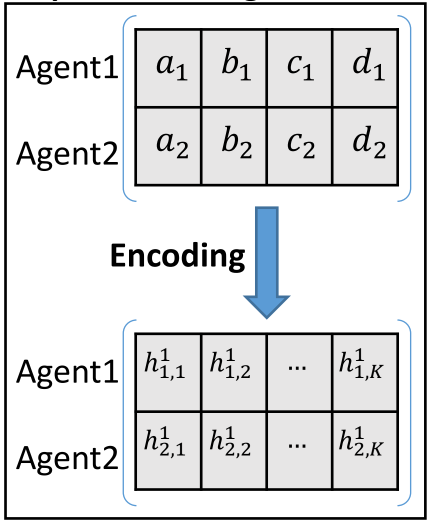
$$h^2_{1,2} = f(h^1_{1,2}, c^1_{1,2})$$

$$\vdots$$

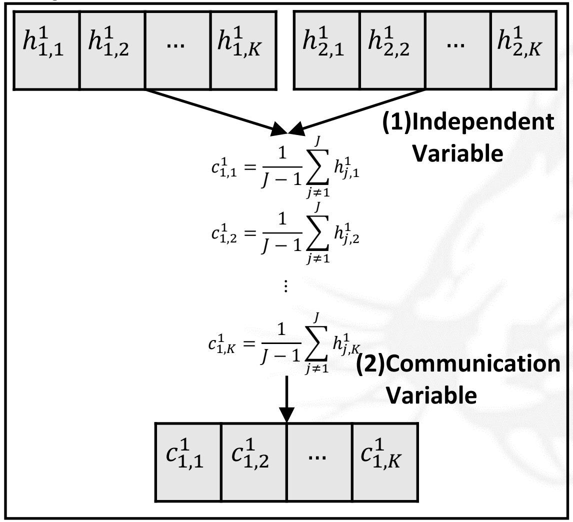
$$h^2_{1,K} = f(h^1_{1,K}, c^1_{1,K})$$

CommNet Mechanism

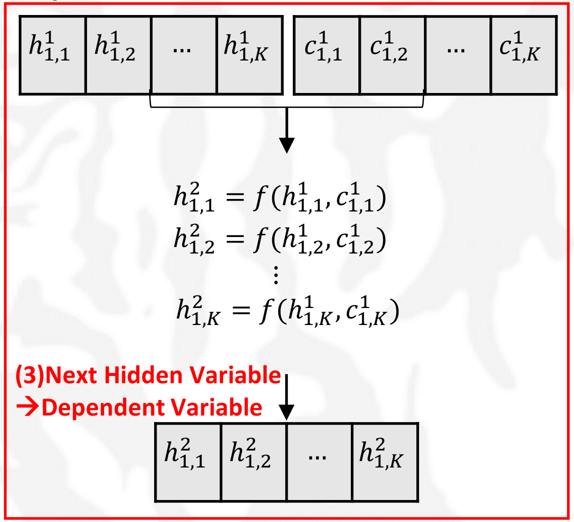
Step#1. Encoding



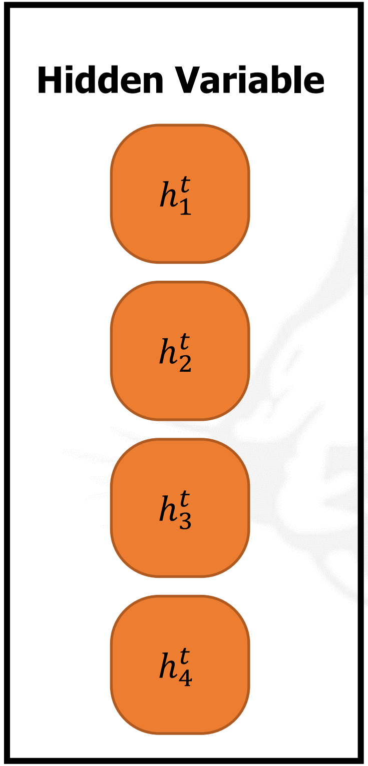
Step#2-1. Communication Variable



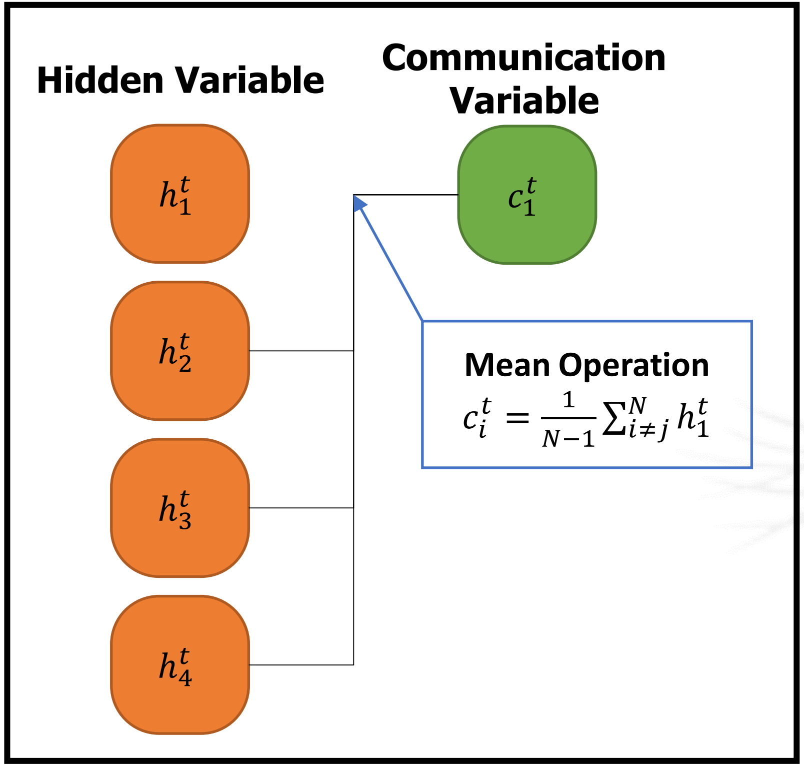
Step#2-2. Activation Function



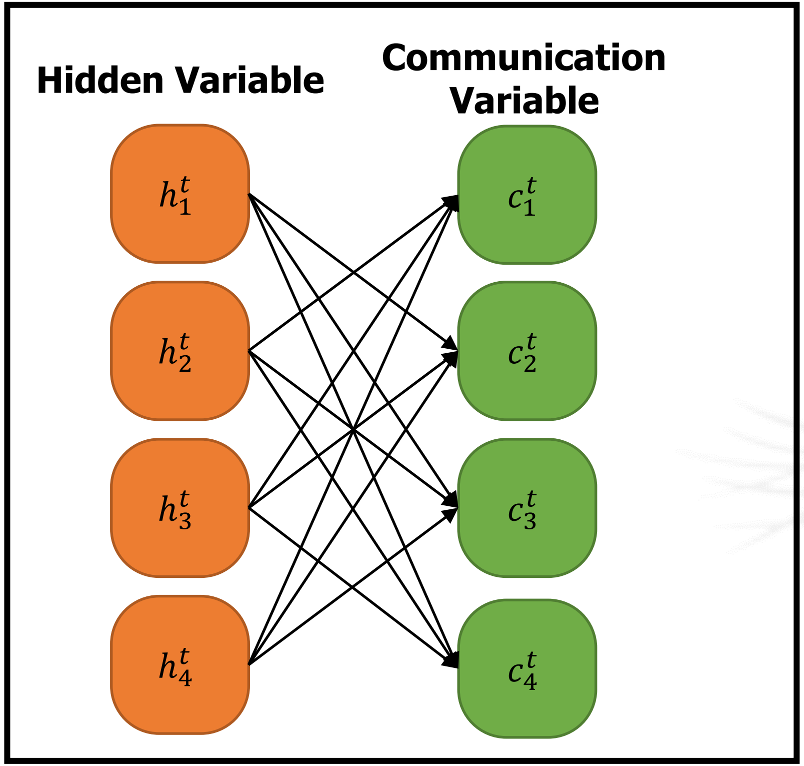
Summary of CommNet



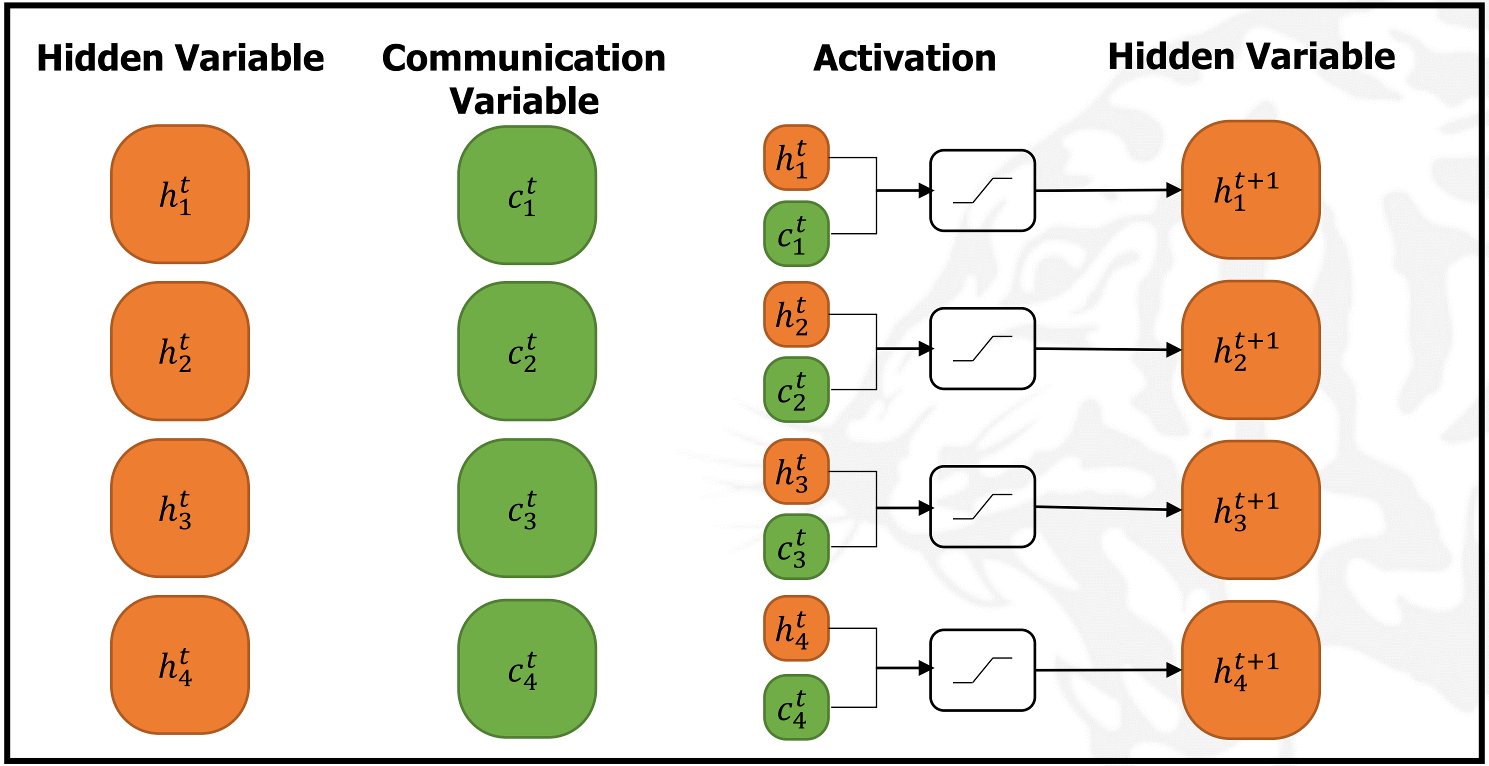
Summary of CommNet



Summary of CommNet

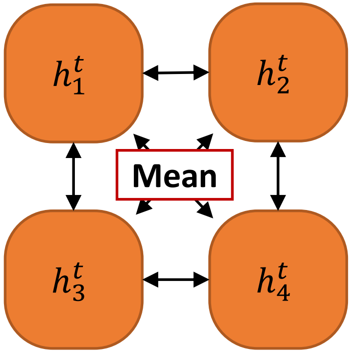


Summary of CommNet



Now we are curious that…

CommNet

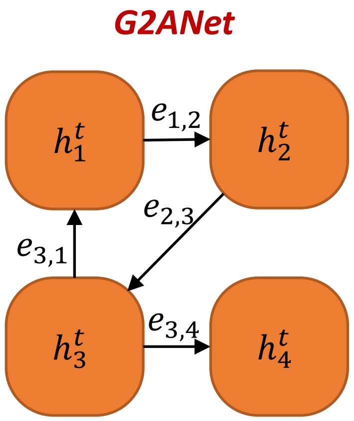


In Graph Approach.

1. Should the agent communicate with all agent?

2. Can we transfer only essential information between agents?

→ *G2ANet will be the solution to the above problem.*



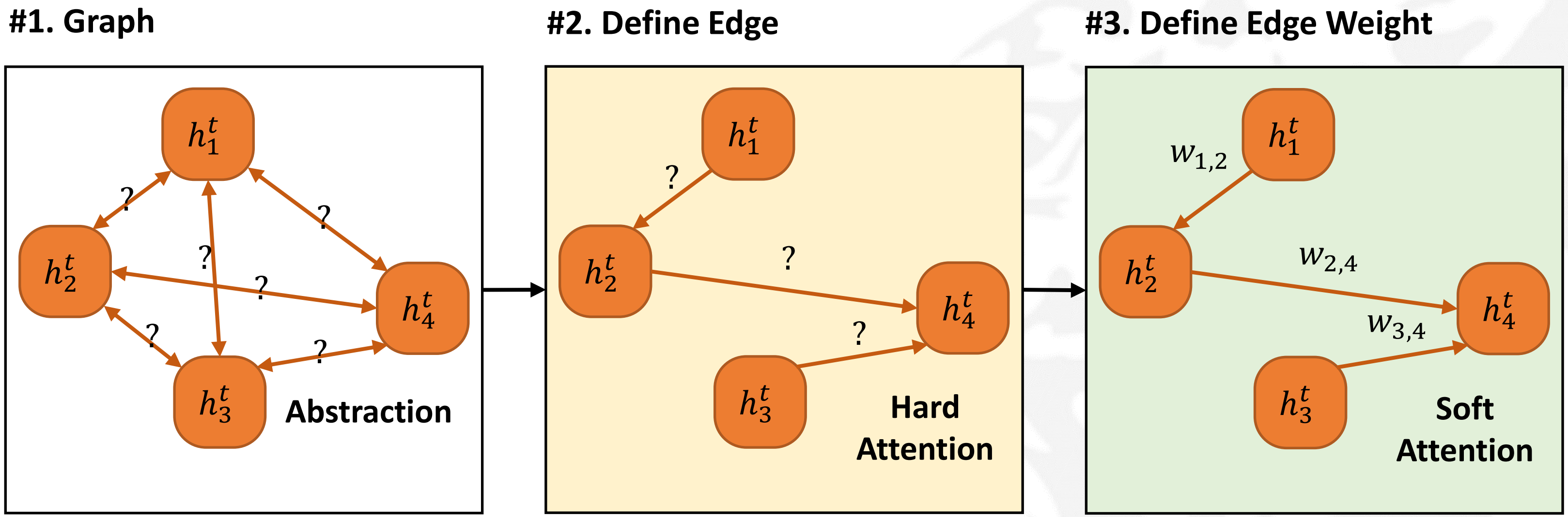
In Graph Approach.

1. Should the agent communicate with all agent?
2. Can we transfer only essential information between agents?

G2ANet **will be the solution to the above problem.**

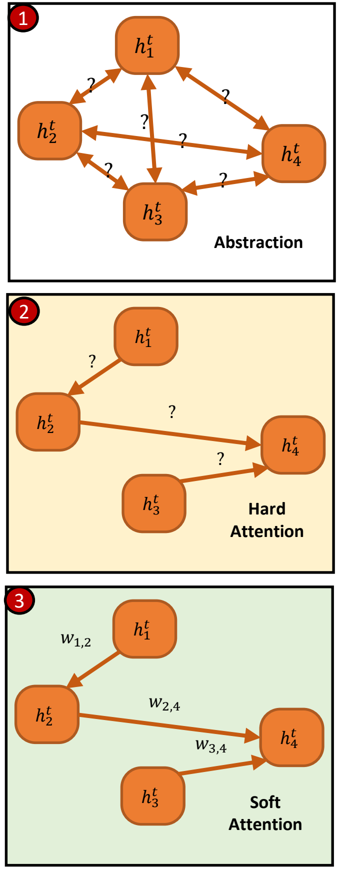
[4] Y. Liu et al., Multi-Agent Game Abstraction via Graph Attention Neural Network, *Proc. AAAI 2020*

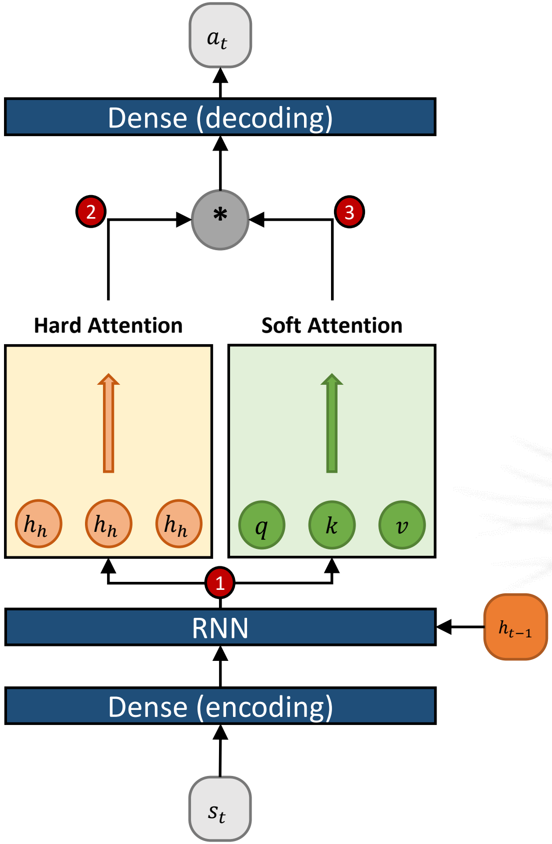
Before beginning *G2ANet* ...



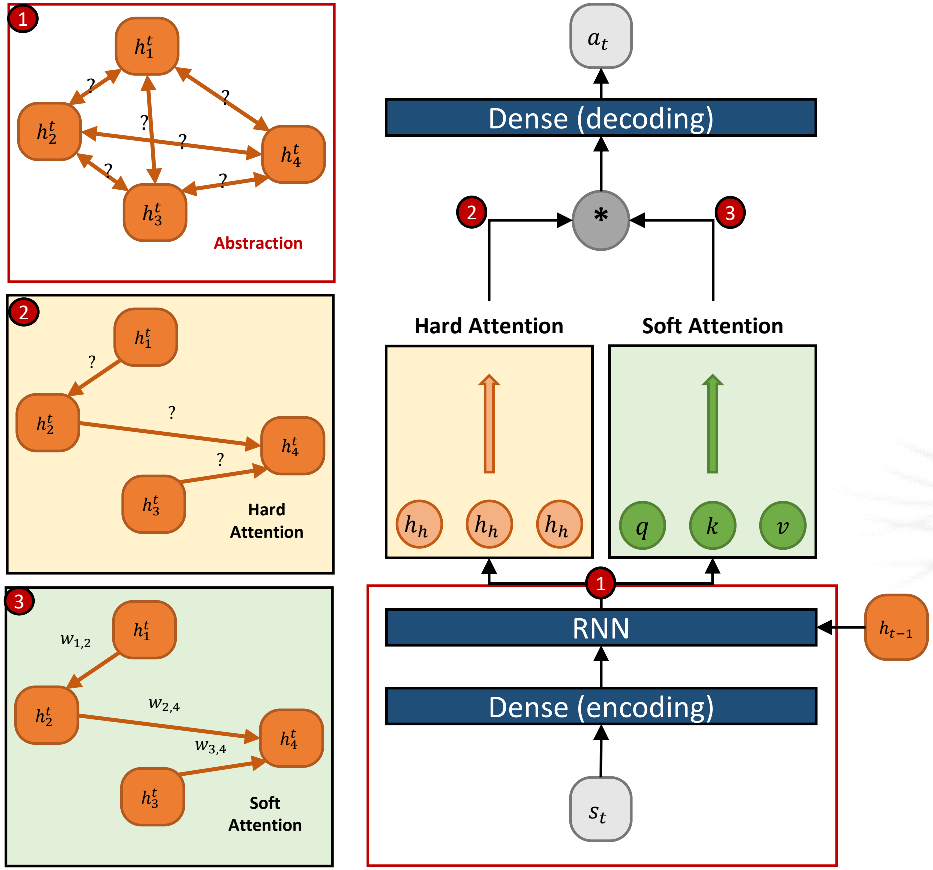
G2ANet Architecture

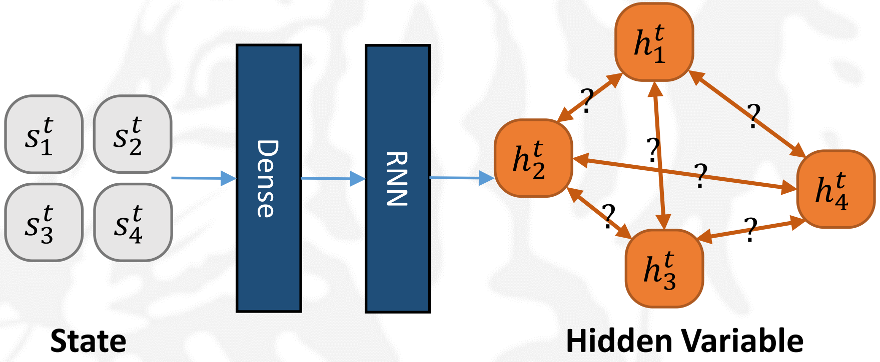
Artificial Intelligence and Mobility Lab



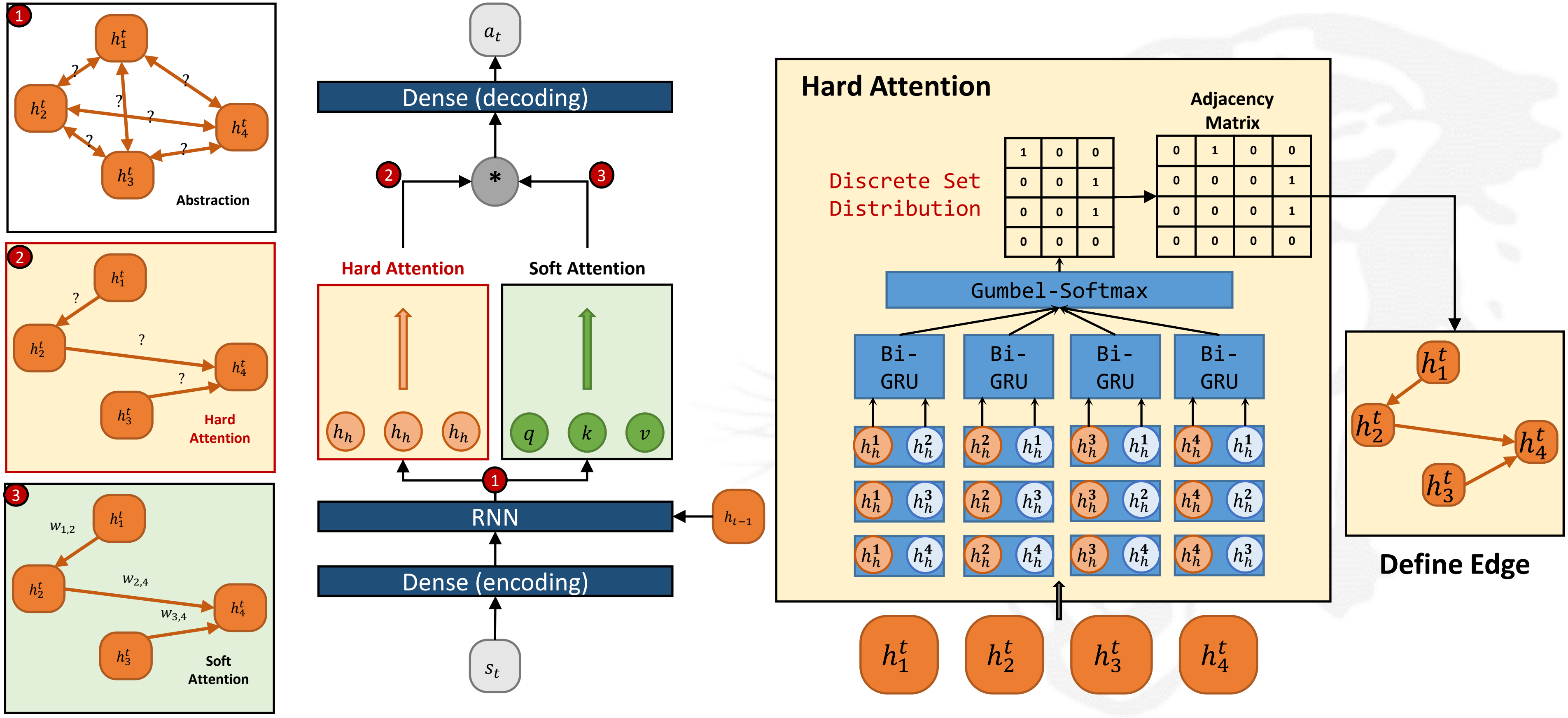


G2ANet Architecture: Abstraction

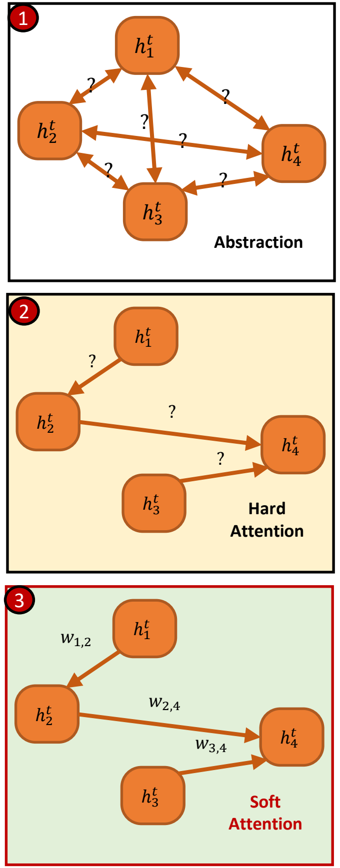


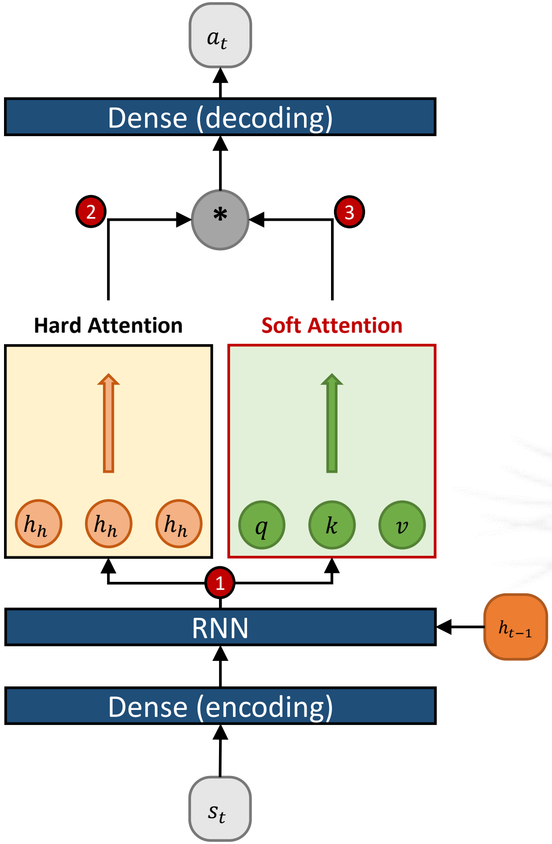


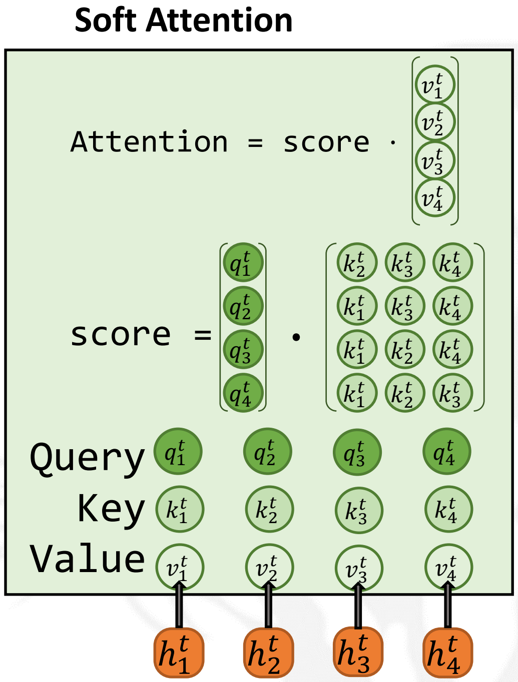
G2ANet Architecture: Hard Attention



G2ANet Architecture: Soft Attention







$$Score = \boldsymbol{q} \cdot \boldsymbol{k} = \boldsymbol{q}^T * \boldsymbol{k}$$

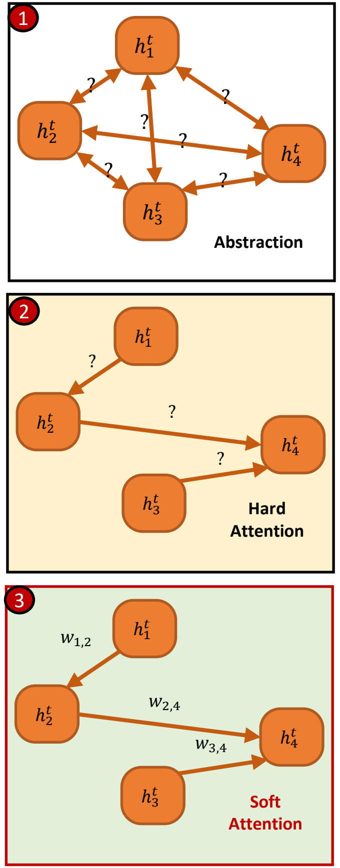
$$Score_{scaled} = \frac{Score}{\sqrt{n}}$$

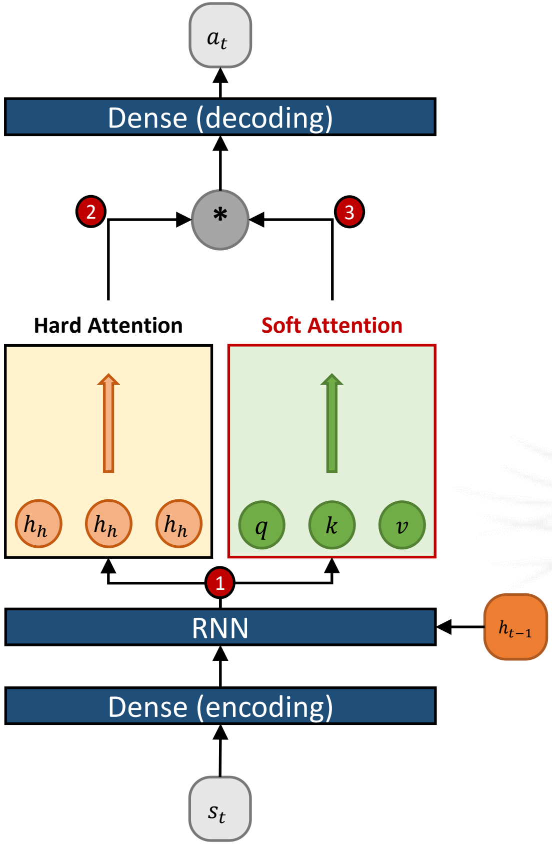
$$Attention(\boldsymbol{q}, \boldsymbol{k}, \boldsymbol{v}) = Score_{scaled} * \boldsymbol{v}$$

Soft Attention Output (message)

0.11	0.84	0.4
0.1	0.18	0.72
0.34	0.38	0.28
0.16	0.14	0.70

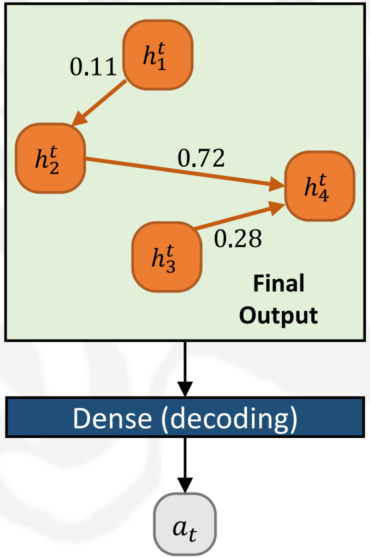
G2ANet Architecture: Soft Attention & GNN





Hard Attention Output (connection)		Soft Attention Output (message)		Final Output (connection & message)

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 0.11 & 0.84 & 0.4 \\ 0.1 & 0.18 & 0.72 \\ 0.34 & 0.38 & 0.28 \\ 0.16 & 0.14 & 0.70 \end{bmatrix} = \begin{bmatrix} 0.11 & 0 & 0 \\ 0 & 0 & 0.72 \\ 0 & 0 & 0.28 \\ 0 & 0 & 0 \end{bmatrix}$$



Introduction and Preliminaries

Deep Reinforcement Learning Theory

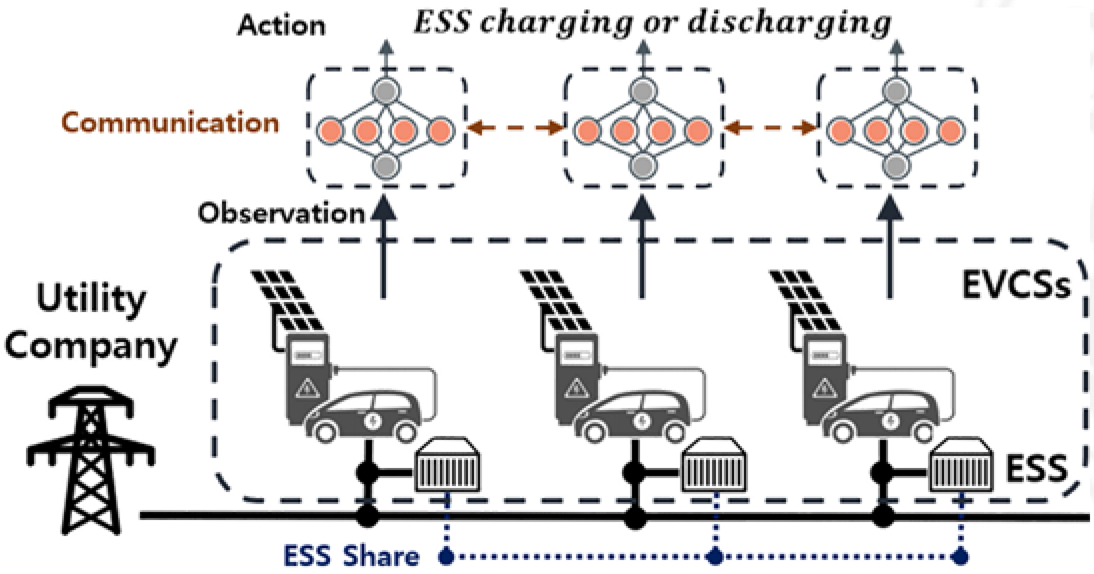
Deep Reinforcement Learning Implementation

Imitation Learning

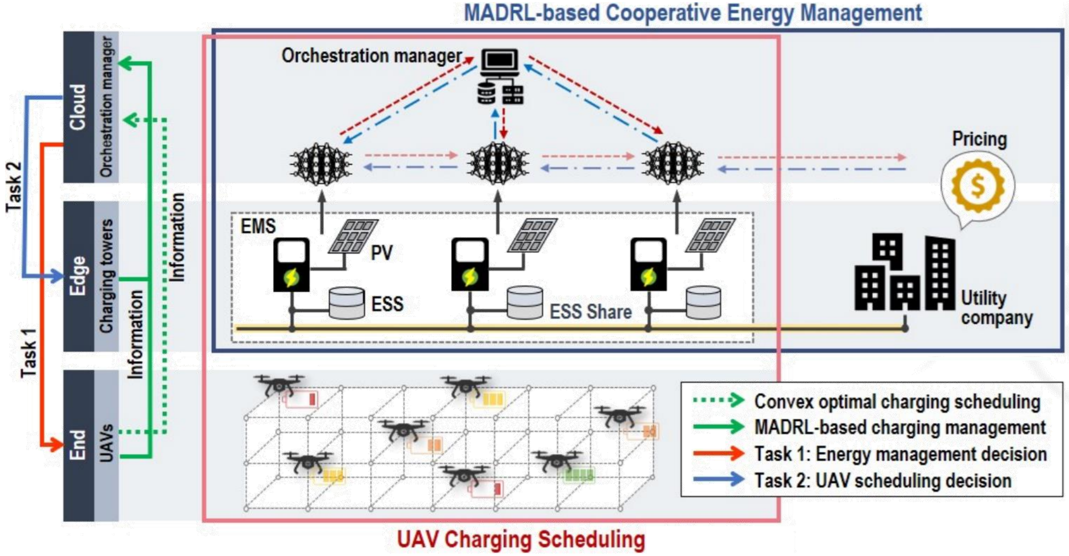
Autonomous Mobility Applications

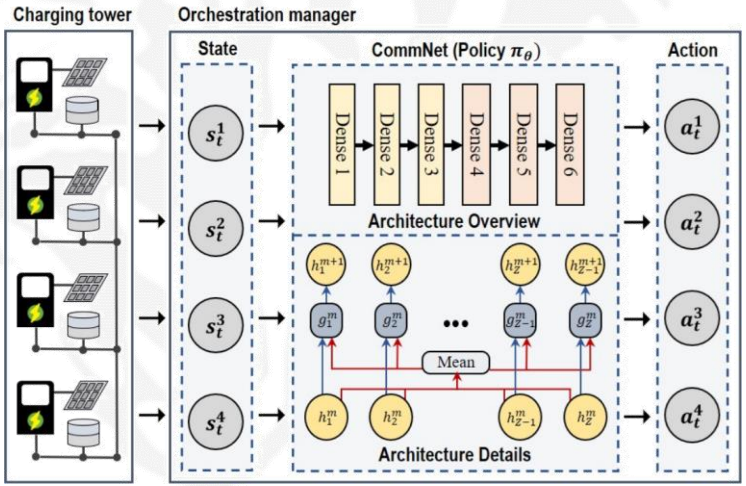
- MADRL
- **Applications**

- Electric Vehicle Charging
  - MyungJae Shin, Dae-Hyun Choi, and Joongheon Kim, "**Cooperative Management for PV/ESS-Enabled Electric-Vehicle Charging Stations: A Multiagent Deep Reinforcement Learning Approach**," IEEE Transactions on Industrial Informatics, 16(5):3493-3503, May 2020.



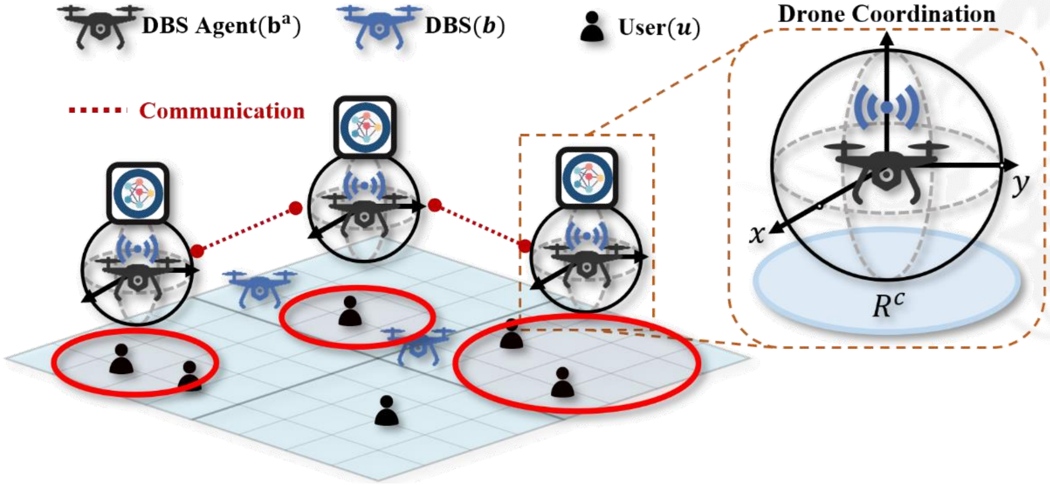
- Multi-UAV Charging
  - Soyi Jung, Won Joon Yun, MyungJae Shin, Joongheon Kim, and Jae-Hyun Kim, "**Orchestrated Scheduling and Multi-Agent Deep Reinforcement Learning for Cloud-Assisted Multi-UAV Charging Systems**," IEEE Transactions on Vehicular Technology, 70(6):5362-5377, June 2021.

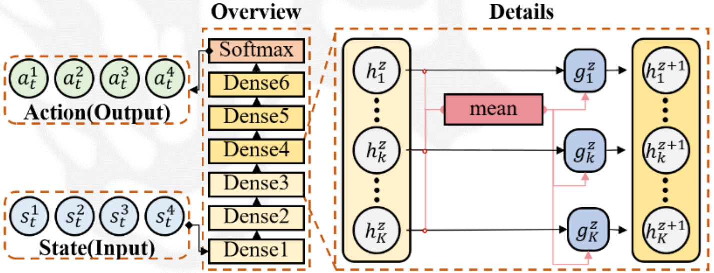




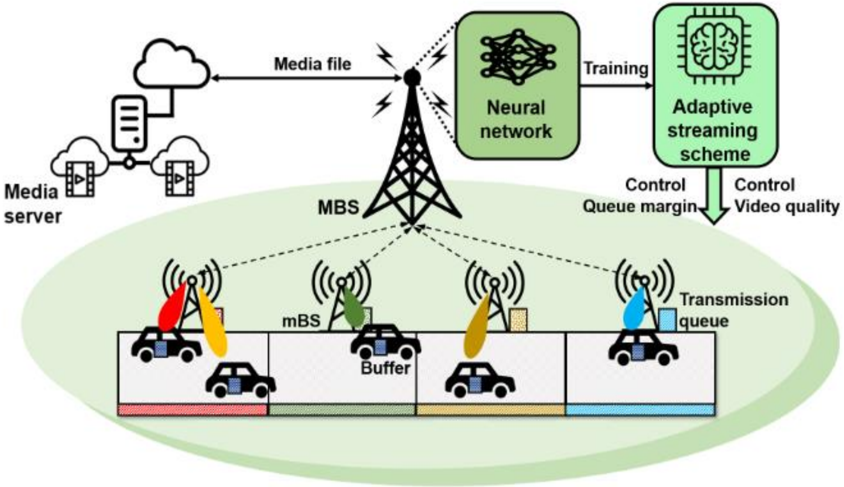
Artificial Intelligence and Mobility Lab

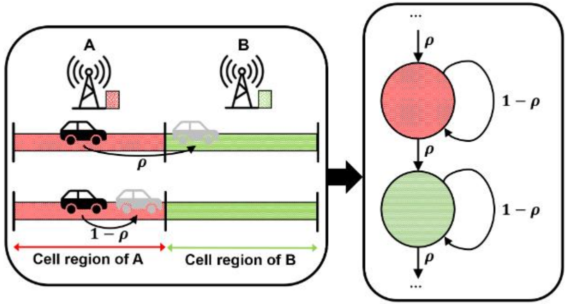
- Multi-UAV Surveillance
  - Won Joon Yun, Soohyun Park, Joongheon Kim, MyungJae Shin, Soyi Jung, David Mohaisen, and Jae-Hyun Kim, "**Cooperative Multi-Agent Deep Reinforcement Learning for Reliable Surveillance via Autonomous Multi-UAV Control**," IEEE Transactions on Industrial Informatics, (To Appear: October 2022).

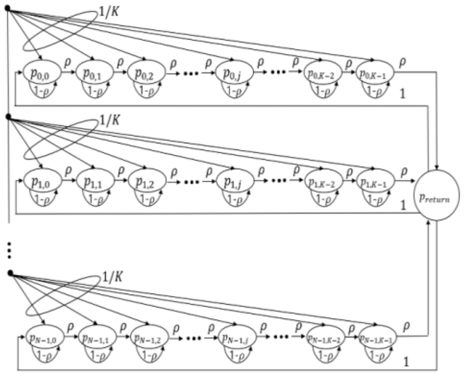




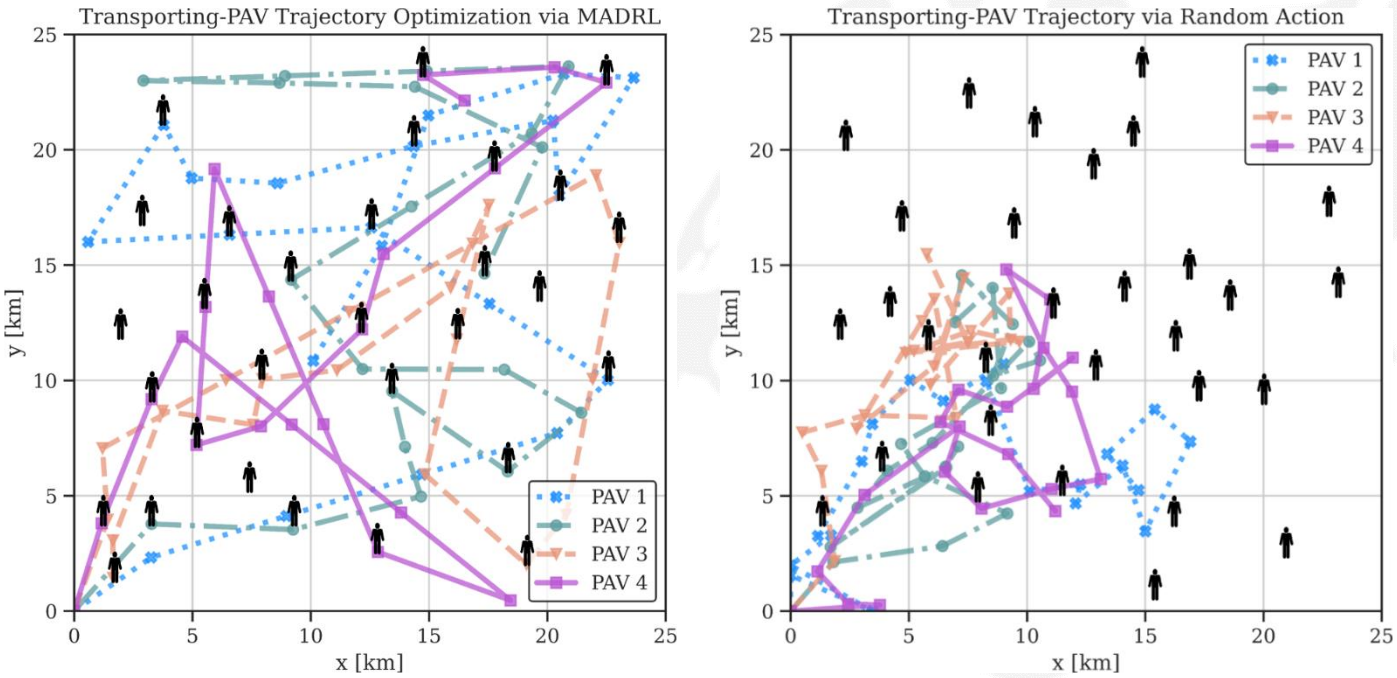
- Streaming in Connected Vehicles
  - Won Joon Yun, Dohyun Kwon, Minseok Choi, Joongheon Kim, Giuseppe Caire, and Andreas F. Molisch, "**Quality-Aware Deep Reinforcement Learning for Streaming in Infrastructure-Assisted Connected Vehicles**," IEEE Transactions on Vehicular Technology, 71(2):2002-2017, February 2022.







- Drone-Taxi Trajectory Learning
  - Won Joon Yun, Soyi Jung, Joongheon Kim, and Jae-Hyun Kim, "**Distributed Deep Reinforcement Learning for Autonomous Aerial eVTOL Mobility in Drone Taxi Applications**," ICT Express, 7(1):1-4, March 2021.



Artificial Intelligence and Mobility Lab

Thank you for your attention!

- More questions?
  - joongheon@korea.ac.kr