Module 6 : Reinforcement Learning

國立中山大學
資訊工程系
張雲南

5

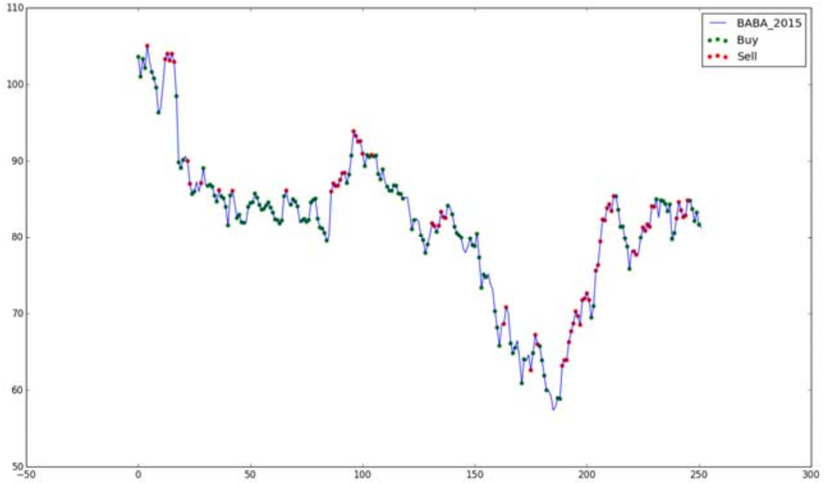
Reinforcement Learning

Application

- Autonomous helicopter
  - http://heli.stanford.edu/
  - https://www.youtube.com/watch?v=VCdxqn0fcnE
- Robotics
  - Towards Learning Robot Table Tennis
  - https://www.youtube.com/watch?v=SH3bADiB7uQ
  - Guided Cost Learning: Deep Inverse Optimal Control via Policy Optimization
  - https://www.youtube.com/watch?v=hXxaepw0zAw
- Save power
  - Transforming Cooling Optimization for Green Data Center via Deep Reinforcement Learning
- Game
  - AlphaGo

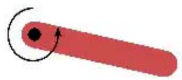
Application

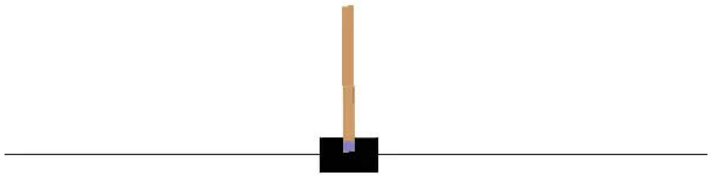
- NLP
  - IBM Watson
- Stock trading
  - https://github.com/llSourcell/Reinforcement_Learning_for_Stock_Prediction

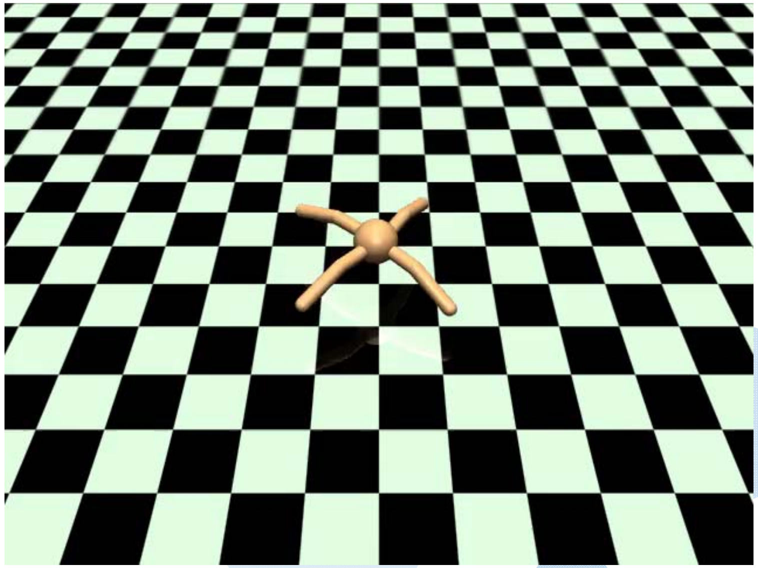


OpenAI gym

- A toolkit for developing and comparing reinforcement learning algorithms.

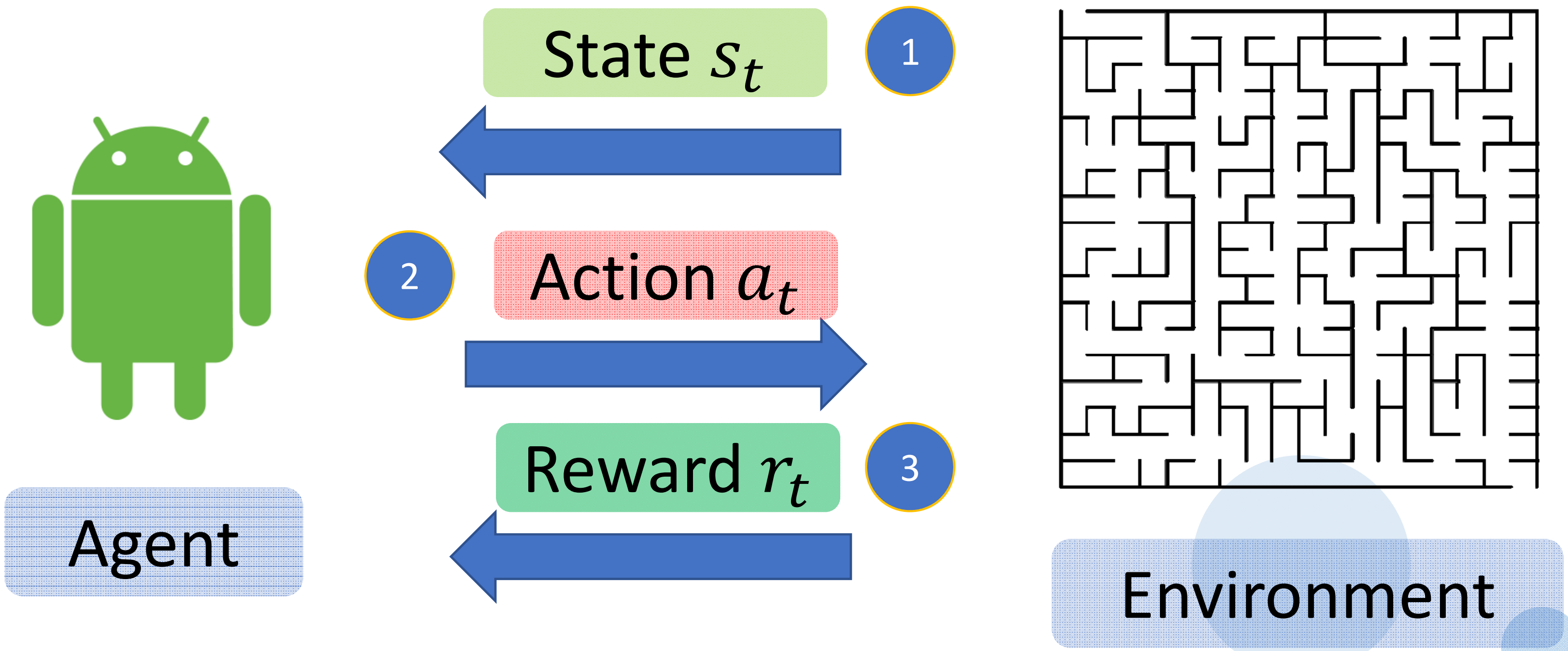




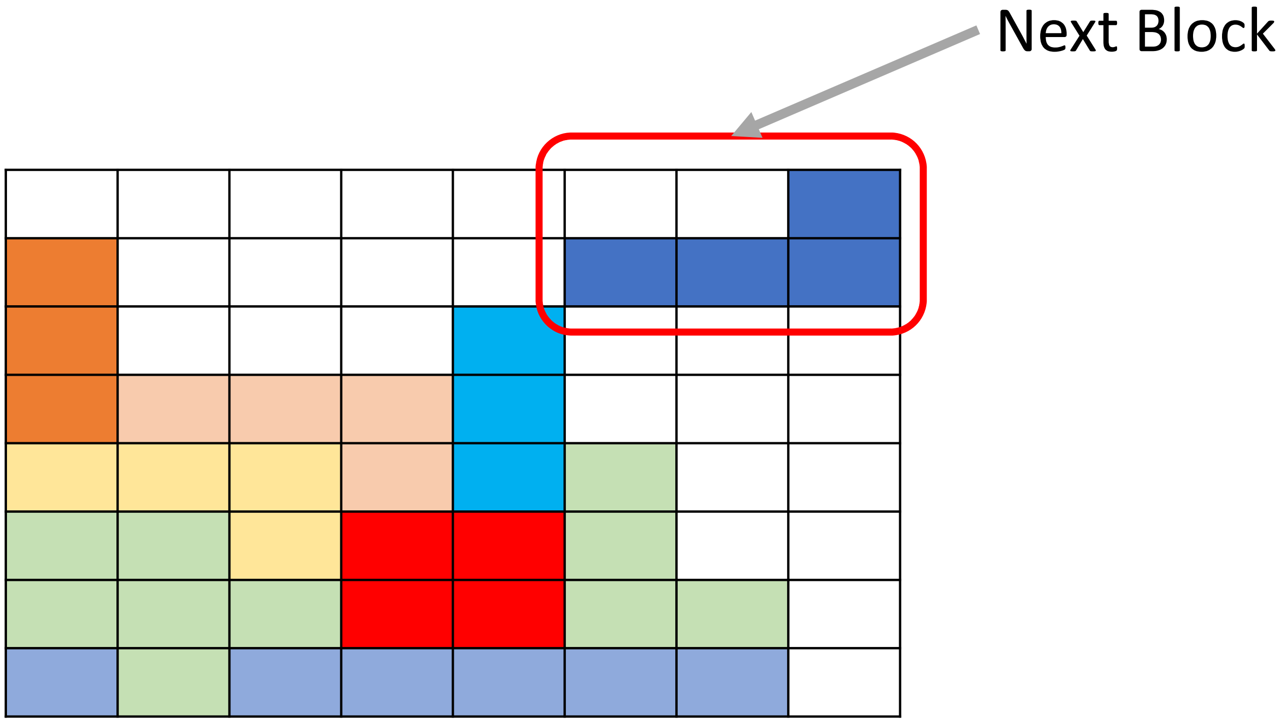


Reinforcement Learning

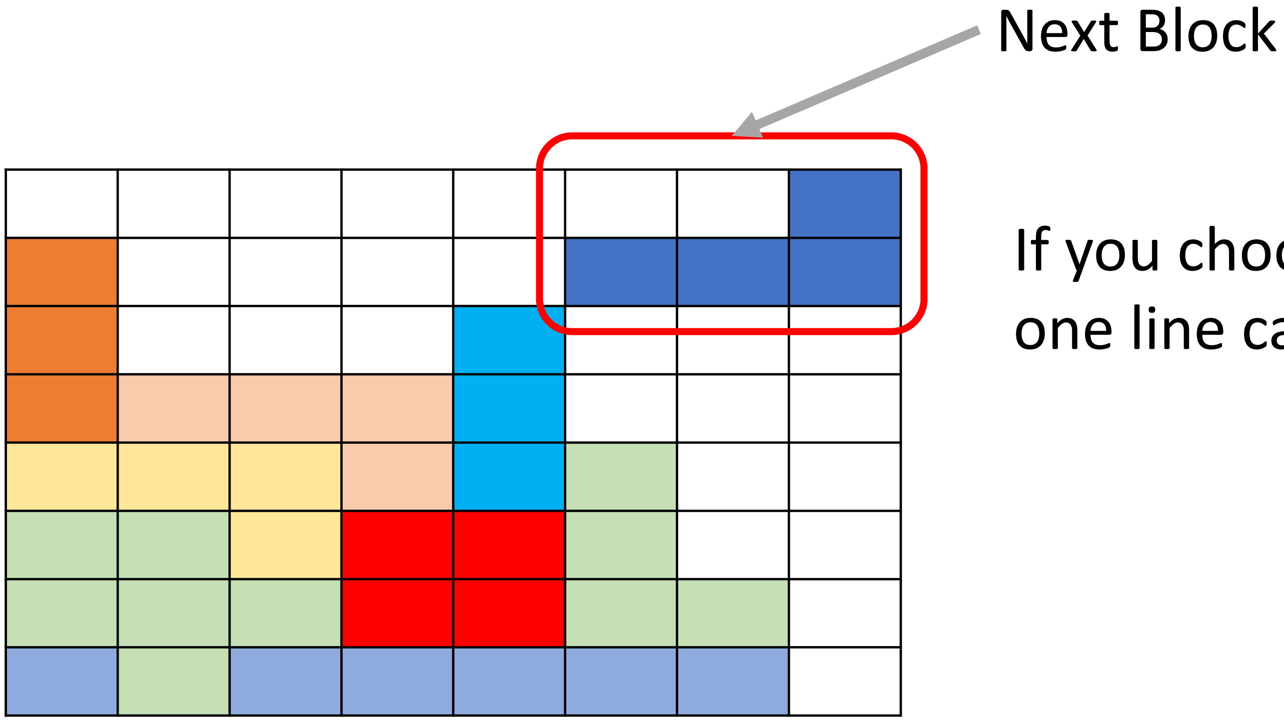
- Train an **Agent/Actor** to take actions that can receive best total reward.
  - It involves a whole sequence of actions.



Total Reward - Tetris

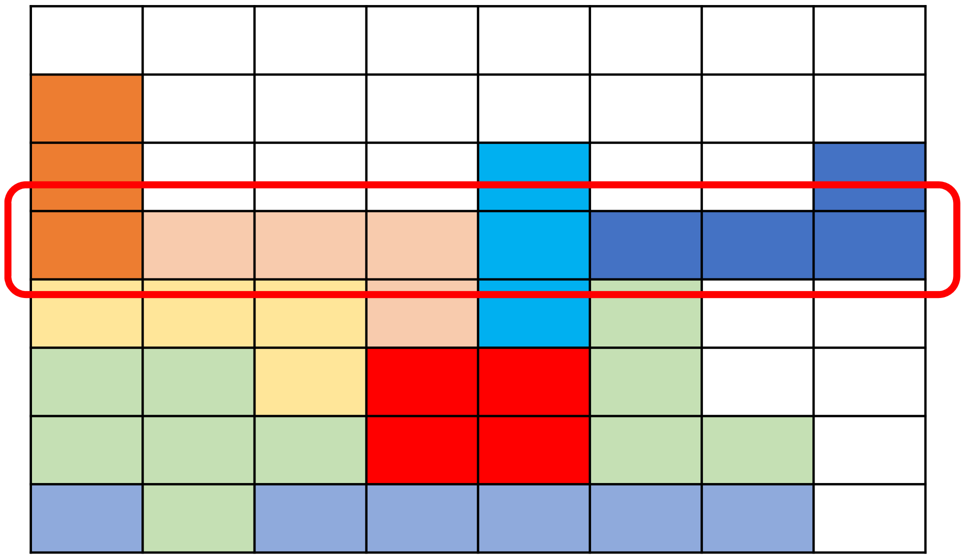


Total Reward - Tetris



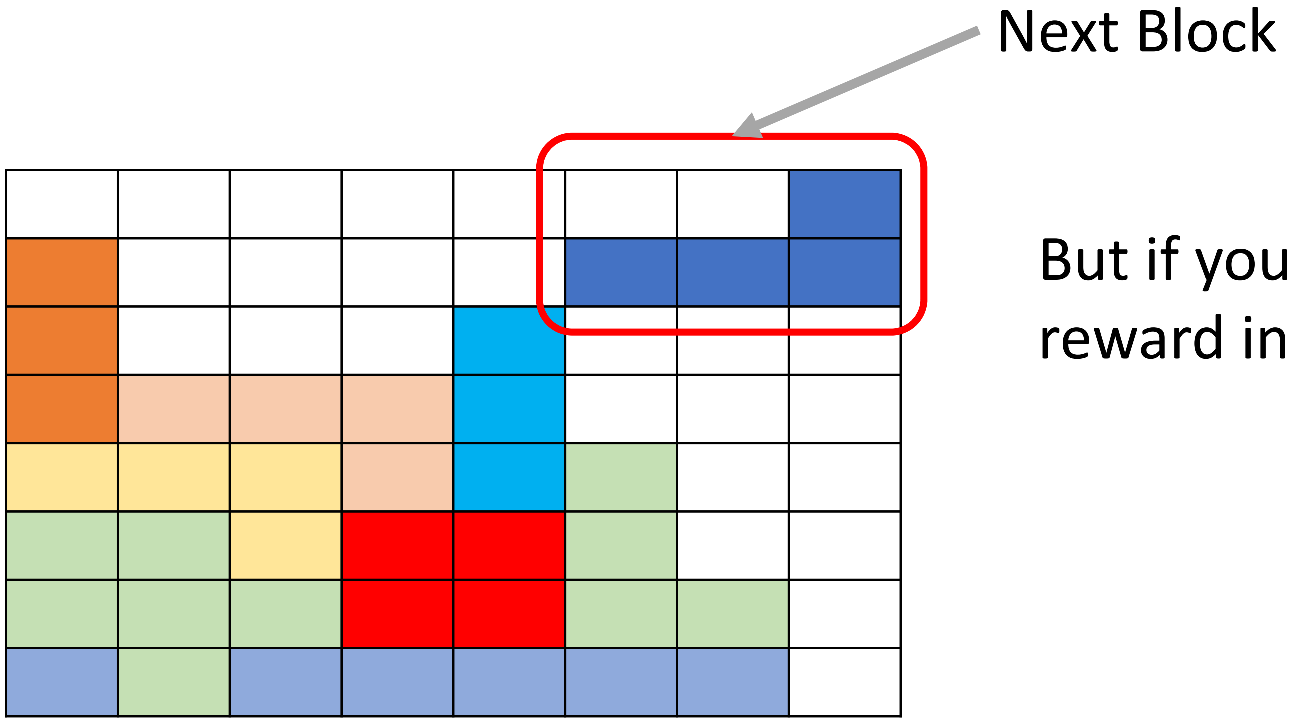
If you choose to go straight down, one line can be cleared.

Total Reward - Tetris



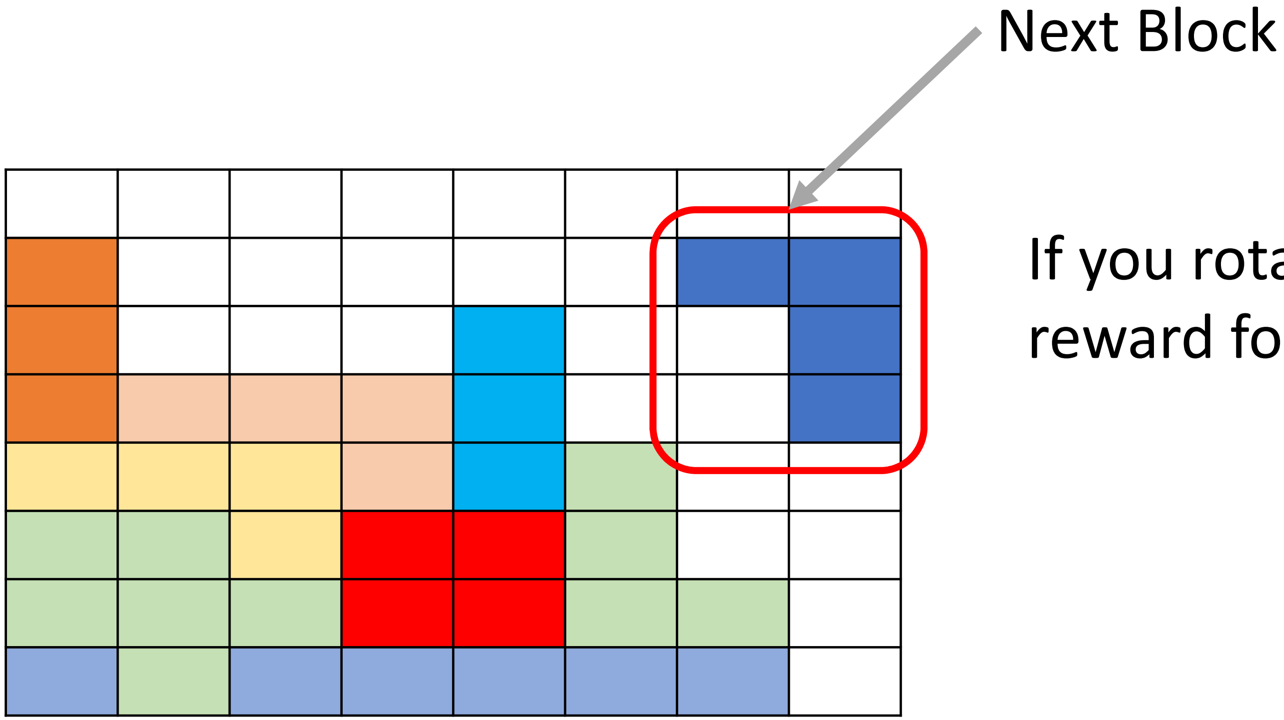
If you go straight down, you can get a reward of 1 line for your "straight-down" action .

Total Reward - Tetris



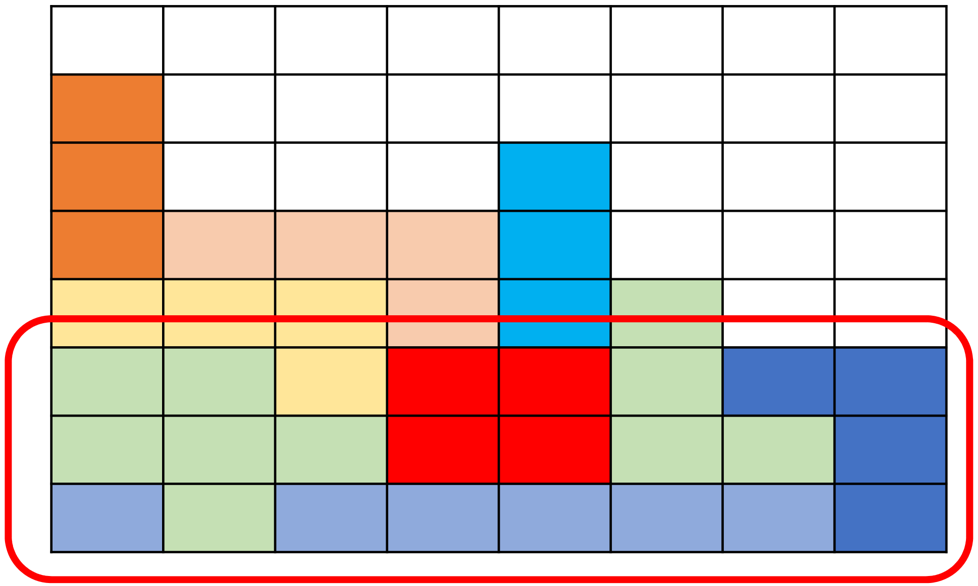
But if you rotate it, you don't get reward in this action.

Total Reward - Tetris



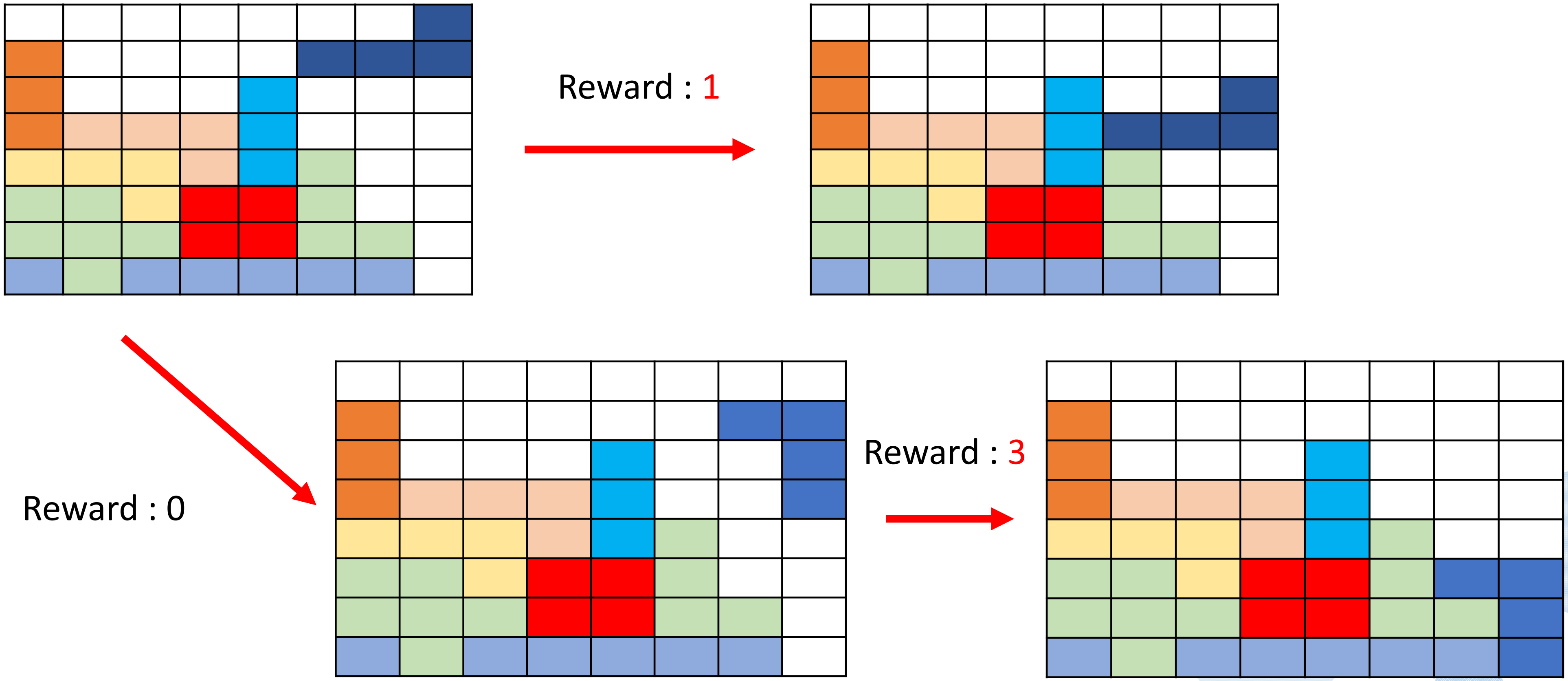
If you rotate it, you don’t get reward for this action.

Total Reward - Tetris



But you can get a higher reward (3 lines) in the next action.

Total Reward - Tetris



Reinforcement Learning

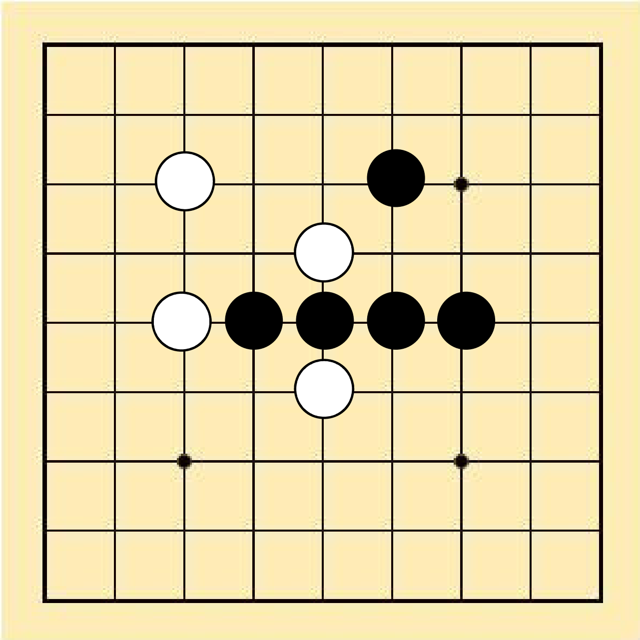
- In Gomoku, one who can form an unbroken chain of five is the winner

State s_t

Board

Environment

Opponent



Action a_t

Where to place the stone

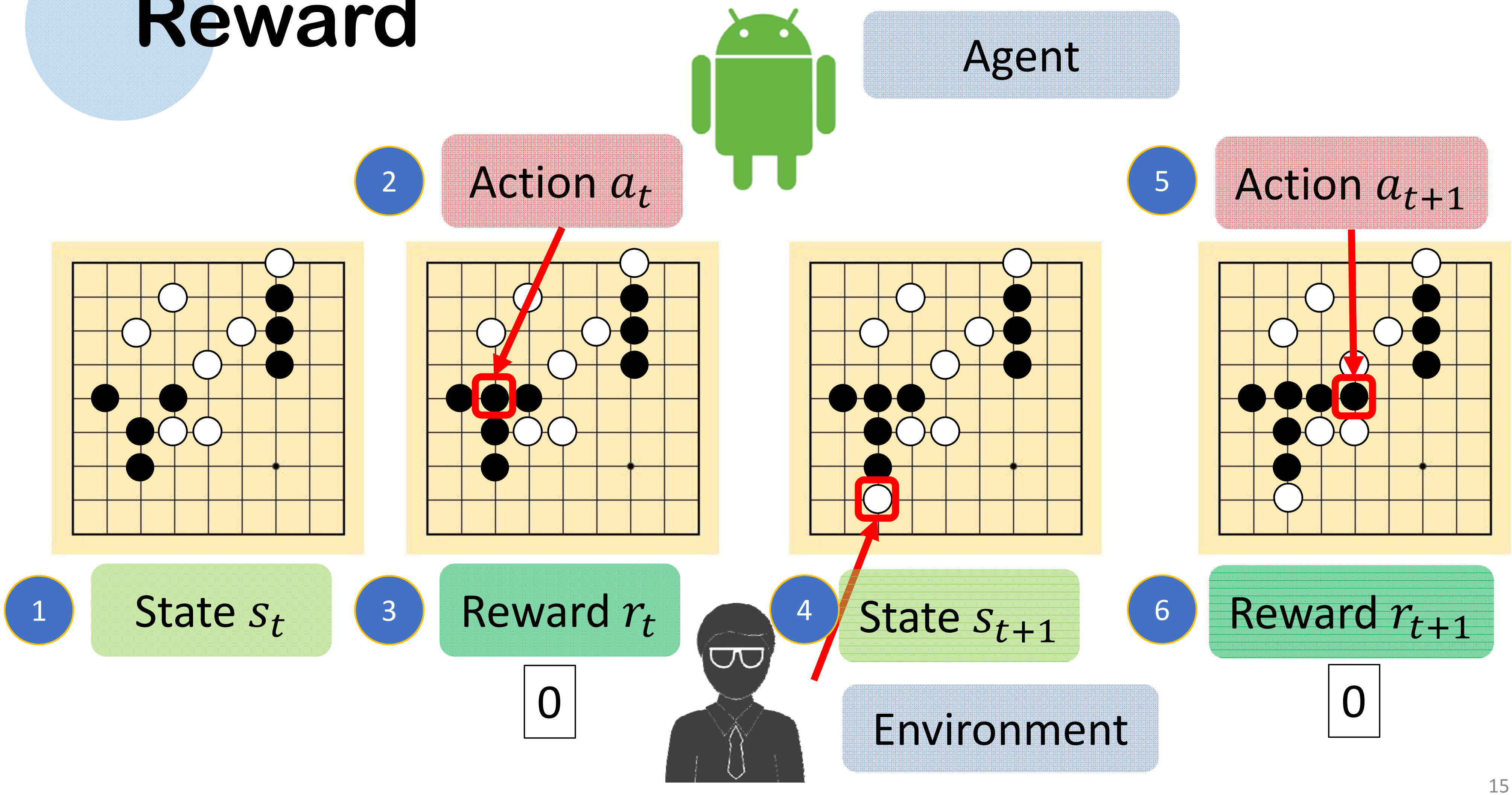
Reward r_t

Win : get 1 point

Not-Win : get 0

Loss: get -1 point

Reward



Agent
2
Action a_t
5
Action a_{t+1}
1
State s_t
3
Reward r_t
0
4
State s_{t+1}
6
Reward r_{t+1}
0
Environment

Autonomous helicopter

State Action Reward

Wind

Gravity

Agent

Motor control

Environment

-10/0/10/100

Reward Grading System

Autonomous helicopter

State Action Reward

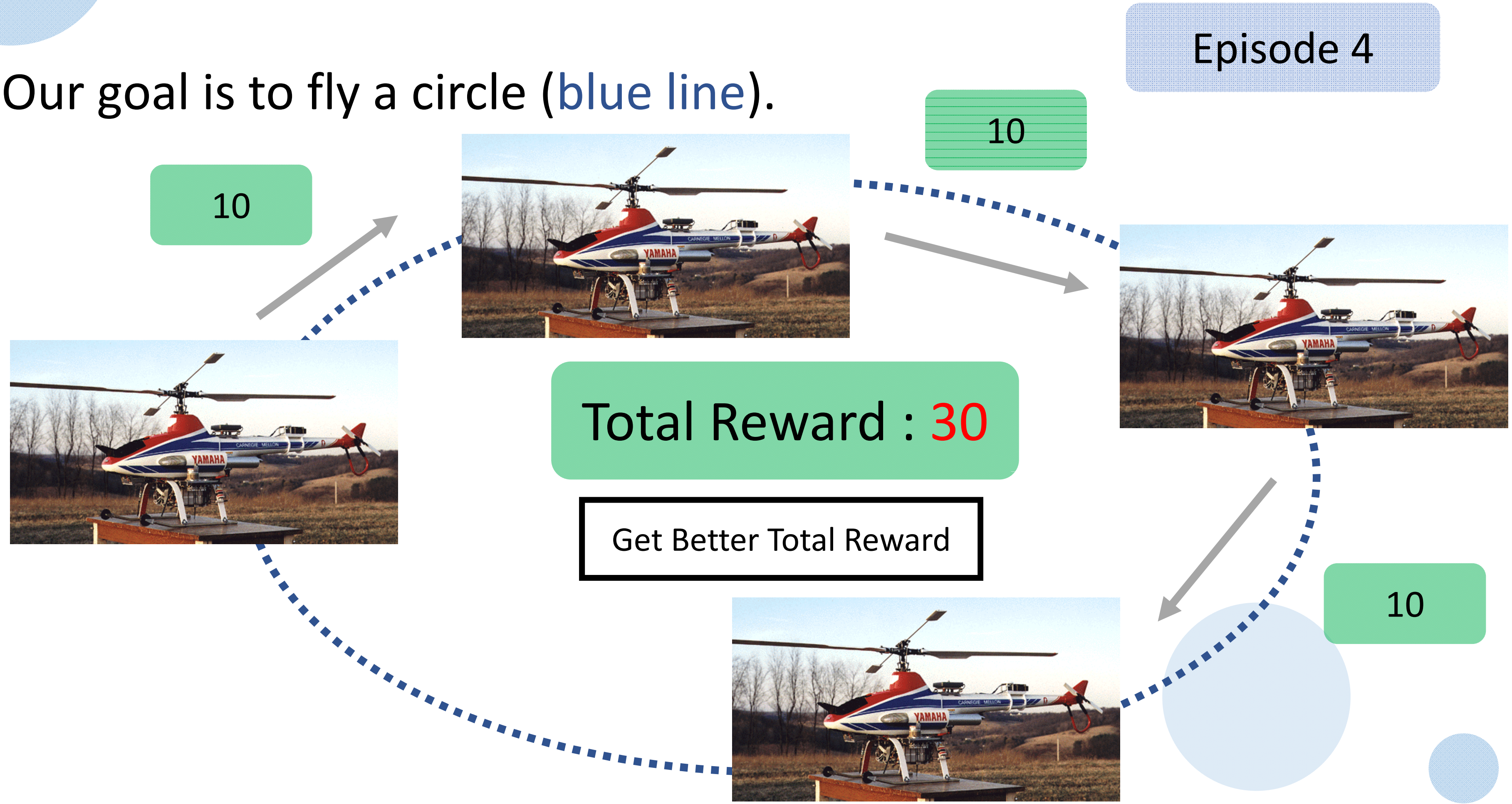
Episode 1

- Our goal is to fly a circle (blue line).

Autonomous helicopter

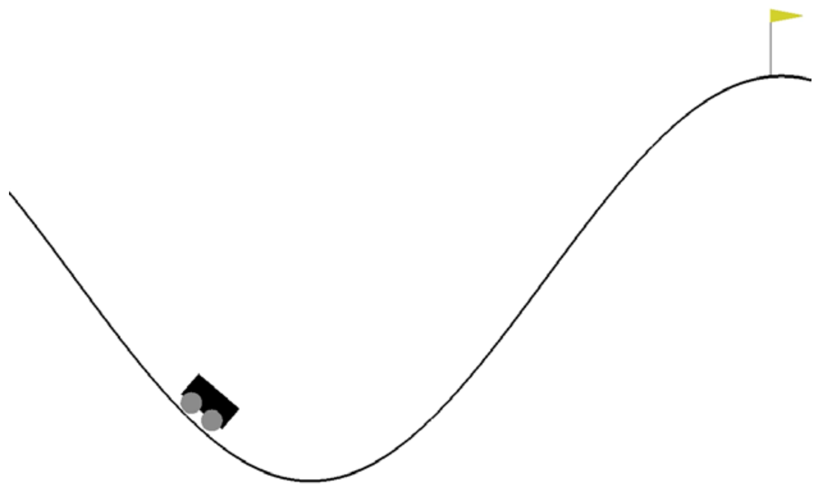
State Action Reward

◆Our goal is to fly a circle (blue line).



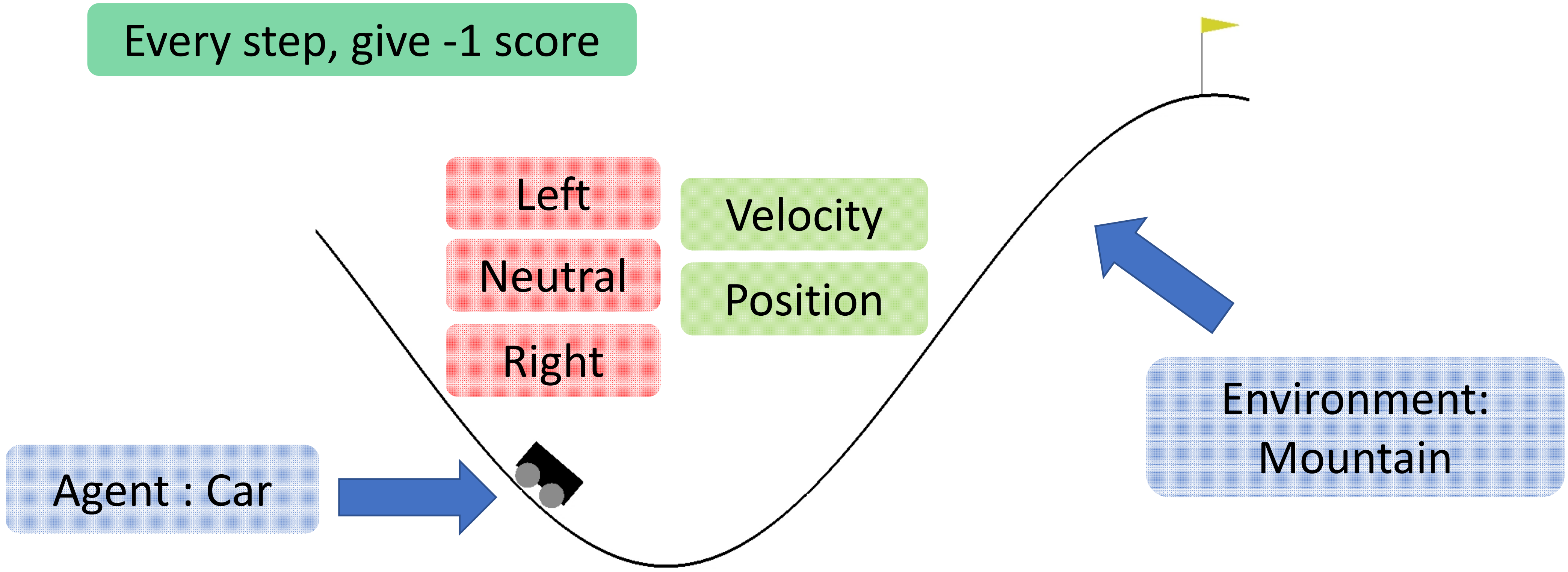
Mountain Car Problem

- Mountain Car is a problem in which an under-powered car must drive up a steep hill.
- Since gravity is stronger than the car's engine, even at full throttle, the car cannot simply accelerate to climb up the steep slope.

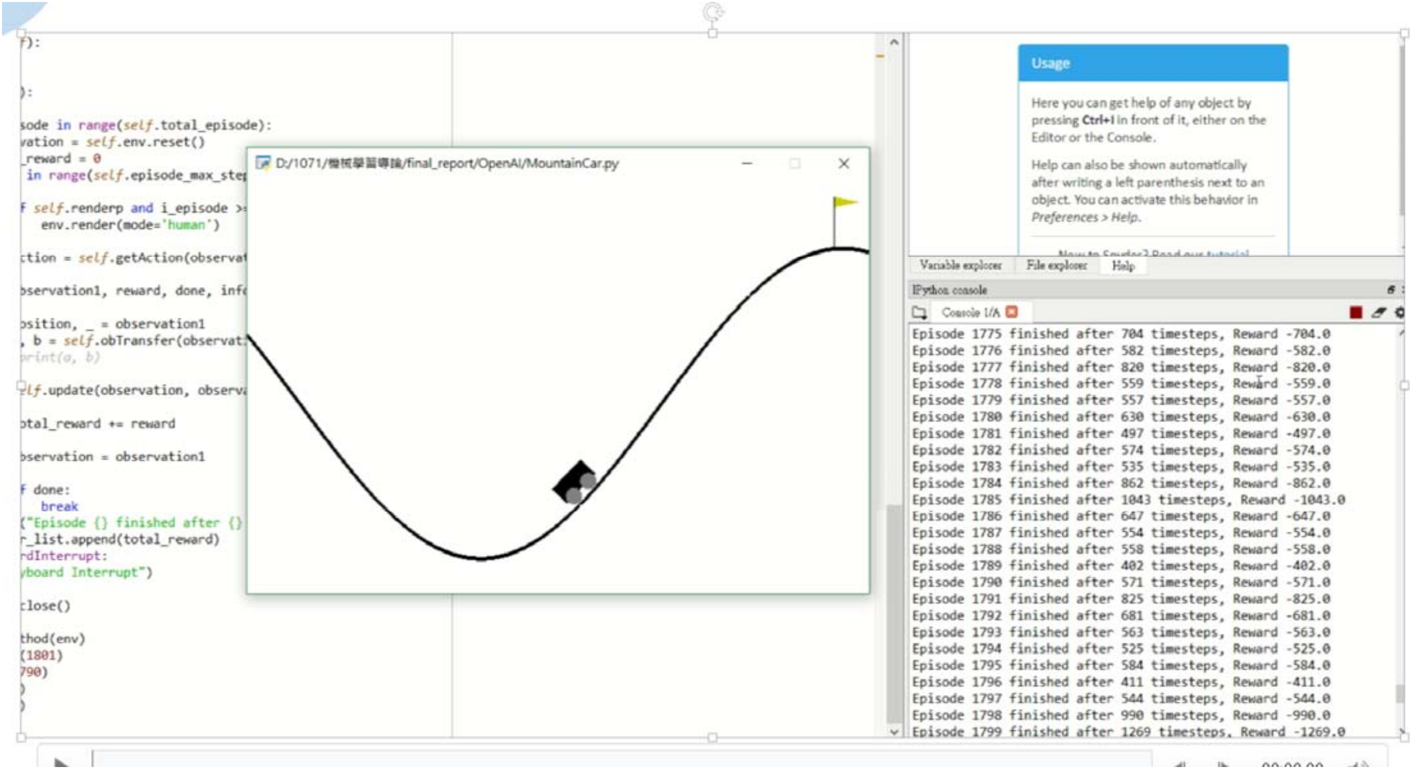


Mountain Car Problem

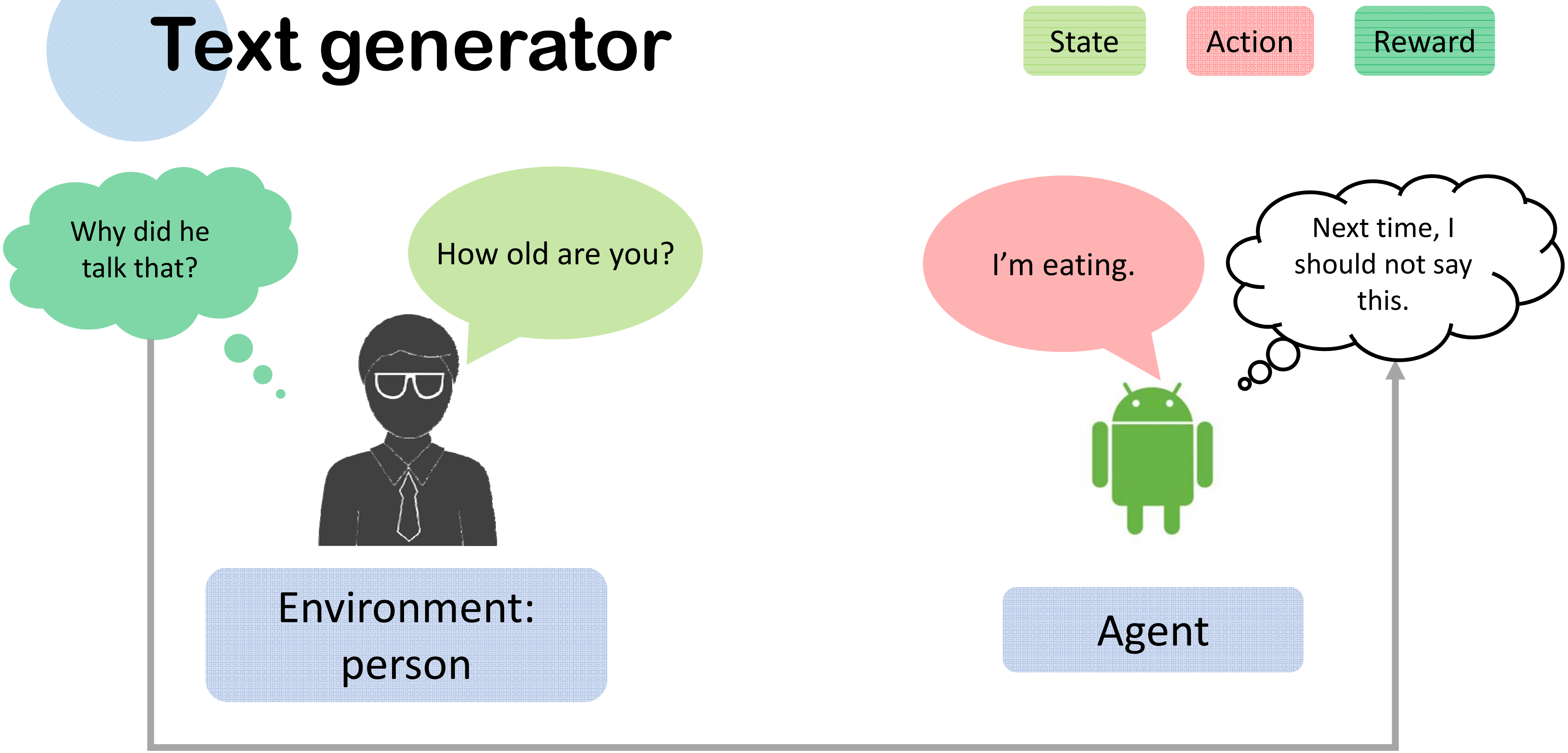
State Action Reward



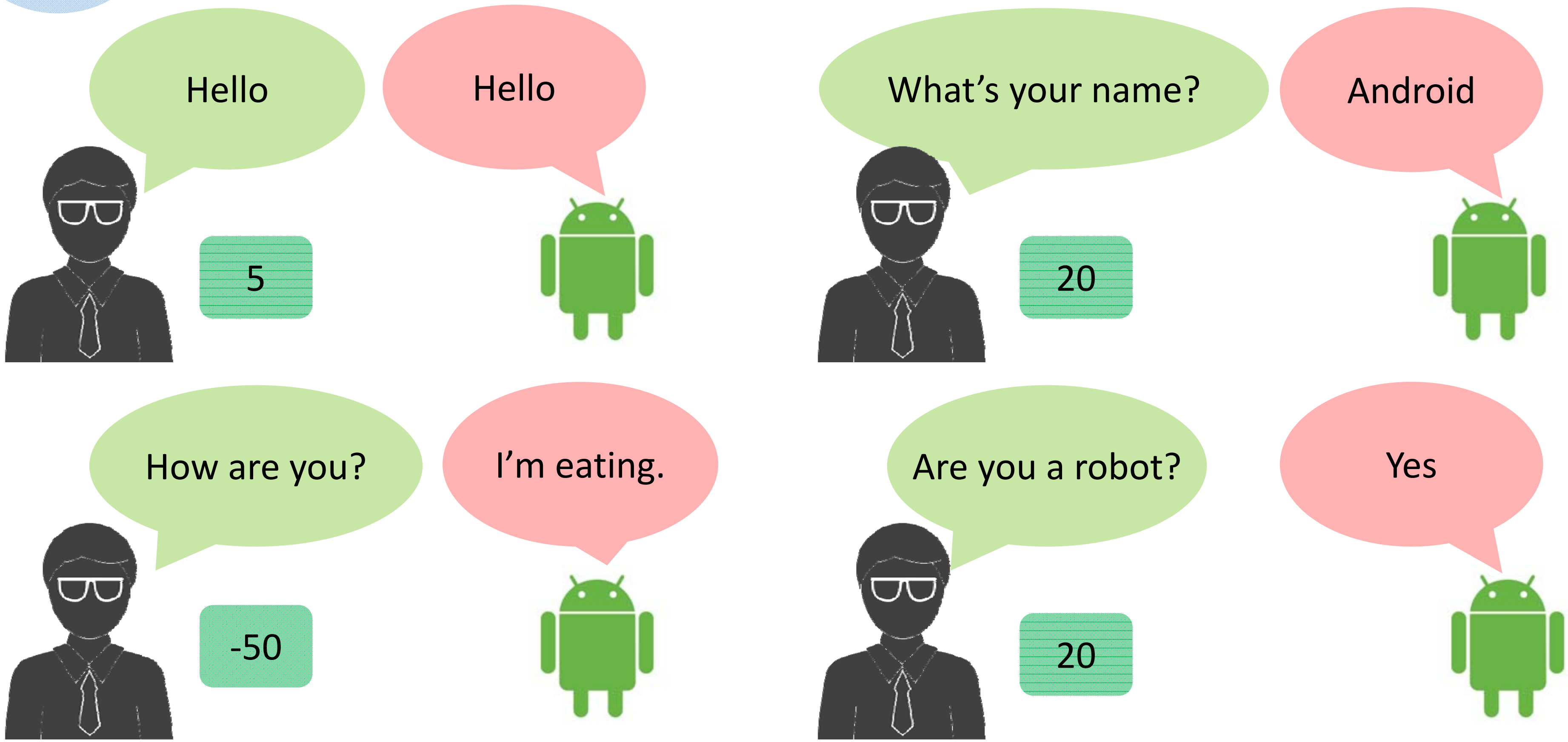
Mountain Car in OpenAI gym



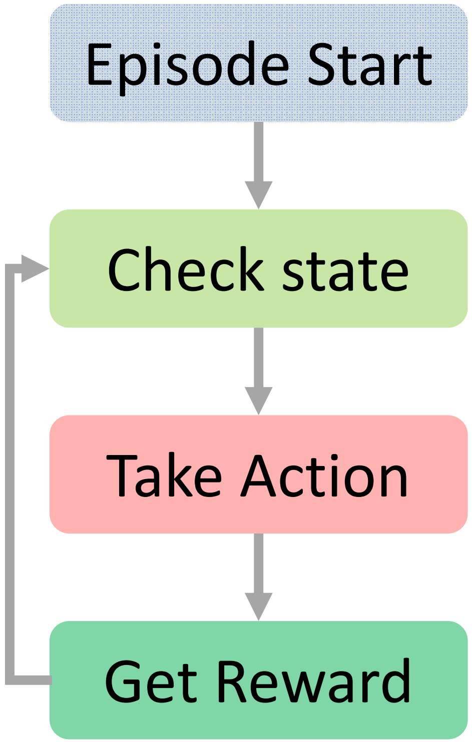
Text generator

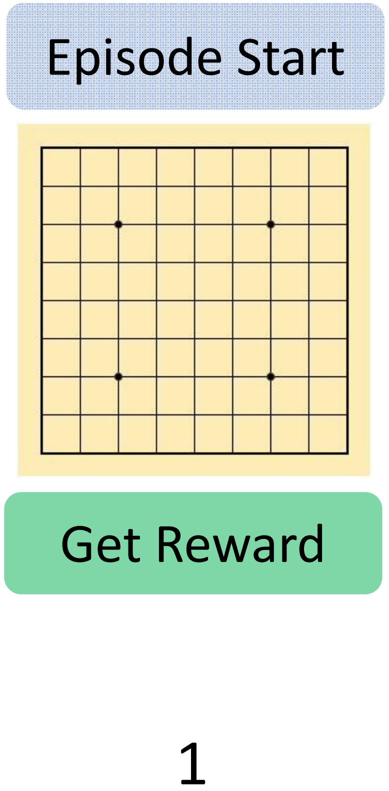


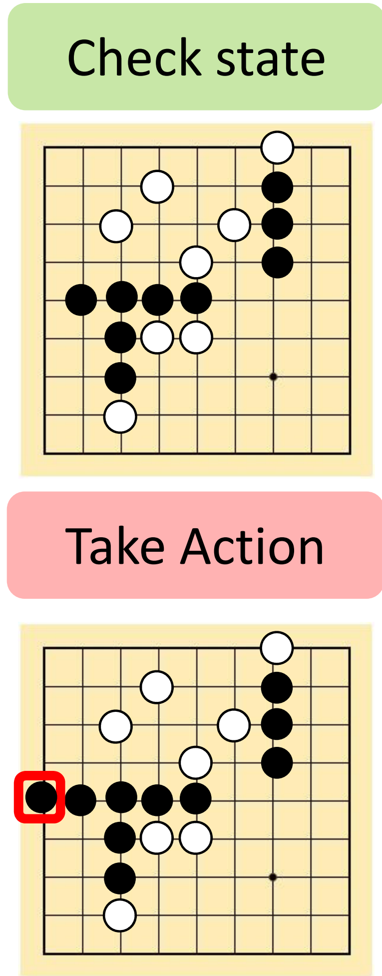
Text generator



Learning Process



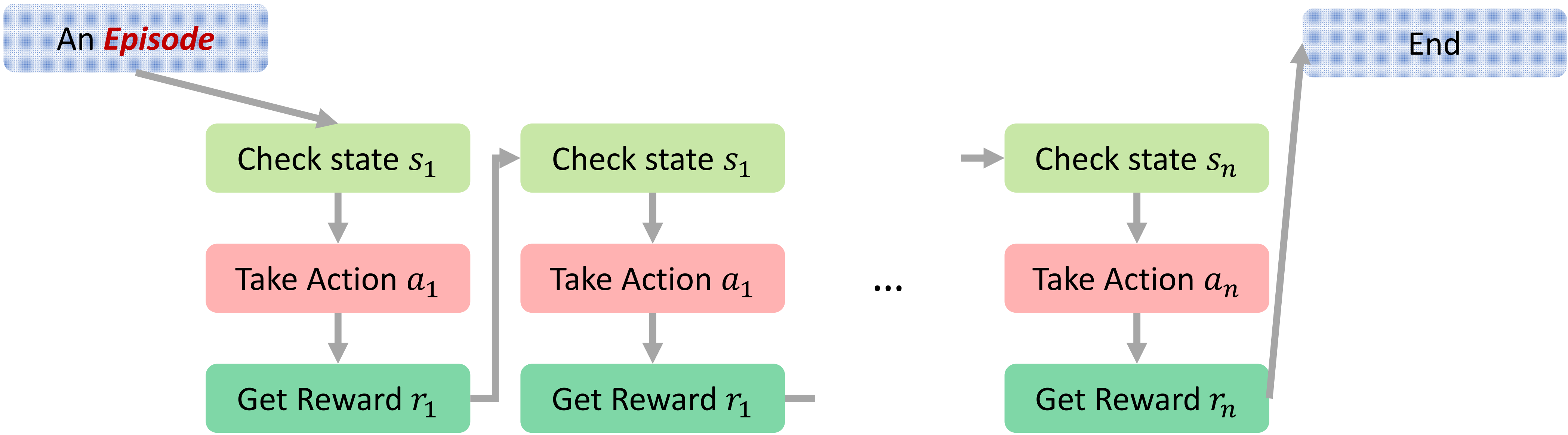




Learning Process

$$Data = \{(s_1, a_1), (s_2, a_2), (s_3, a_3), \ldots, (s_n, a_n)\}$$

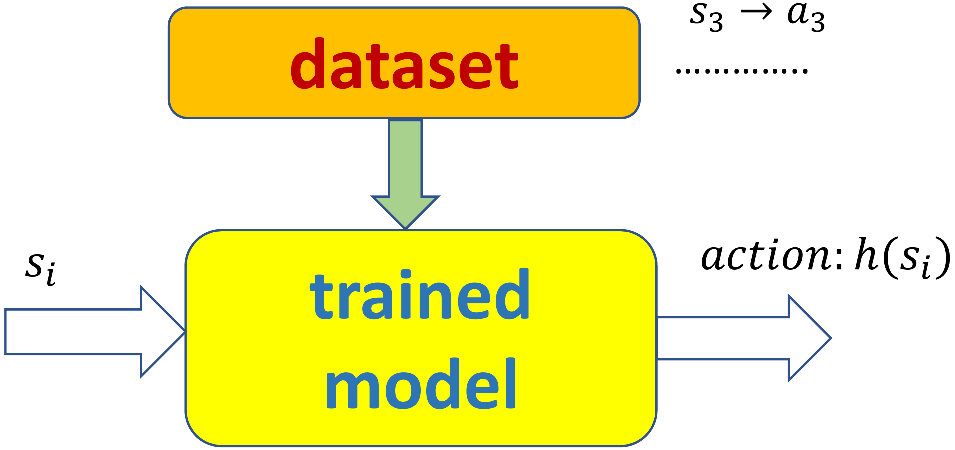
$$Reward = \{r_1,\ r_2,\ r_3, \ldots, r_n\}$$

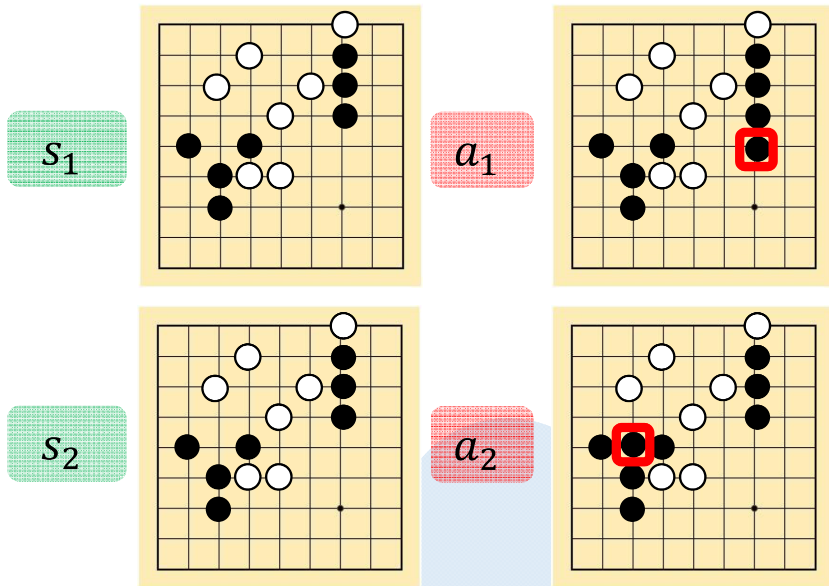


Supervised vs Reinforcement

- For supervised Learning, we should have the dataset of *(state, action)* pairs ready from previous experience.
  - Neglect about reward.
  - Train a machine learning model.
    - Copy the experience.

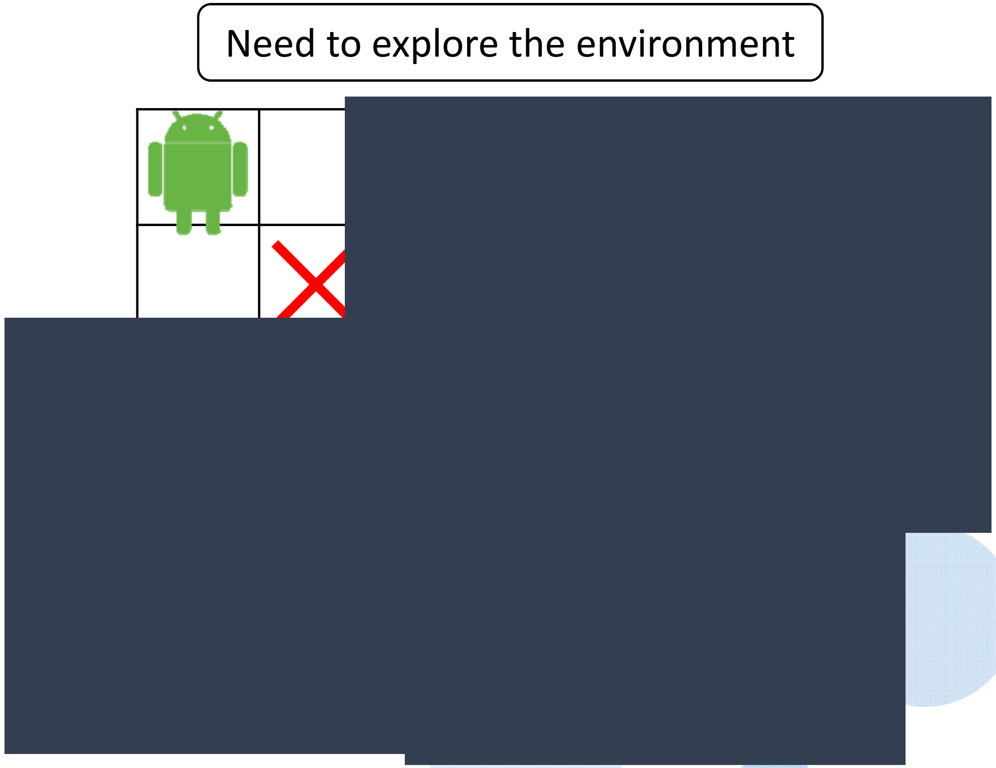
$s_1 \rightarrow a_1$
$s_2 \rightarrow a_2$
$s_3 \rightarrow a_3$
....





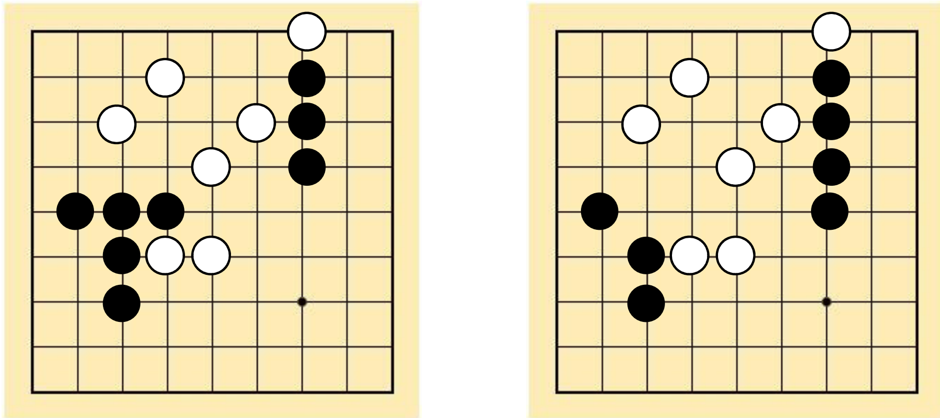
Supervised vs Reinforcement

- For reinforcement Learning, sometime the dataset is not ready.
- We need to explore during the learning process.
- We have to consider the total reward.



Value-based v.s. Policy-based

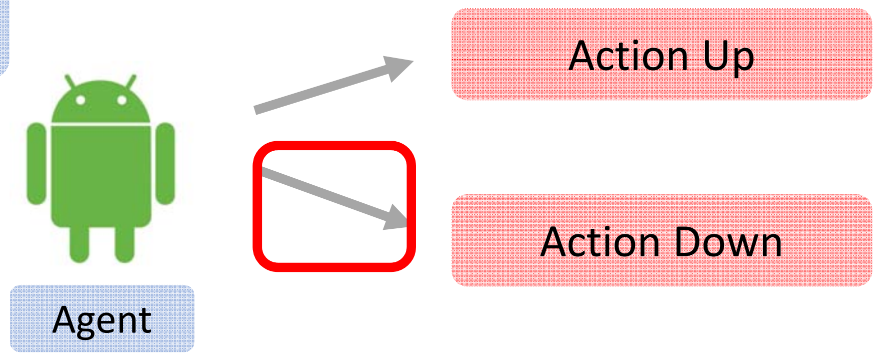
- Value-based: Learn a model to evaluate the "total reward" from the state.
  - Q-Learning
  - DQN



- Policy-based: Learn a model to decide "action" from the state.
  - Policy gradient

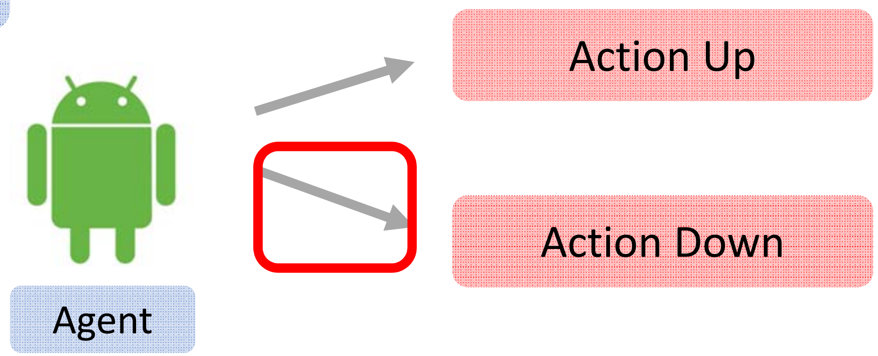
Value-based v.s. Policy-based

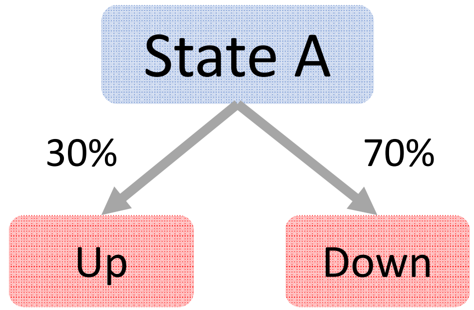
Value-based



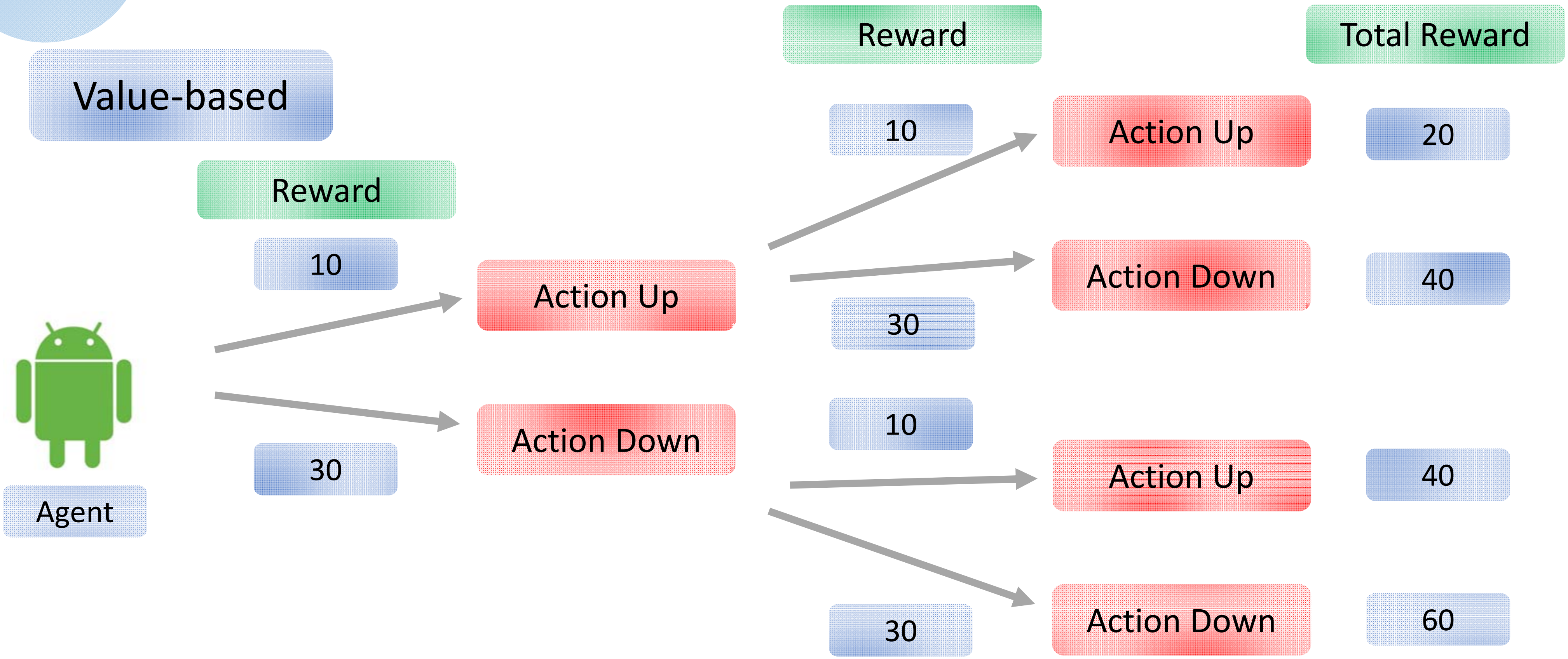
	Up	Down
State A	10	30
State B	30	21

Policy-based





Value-based

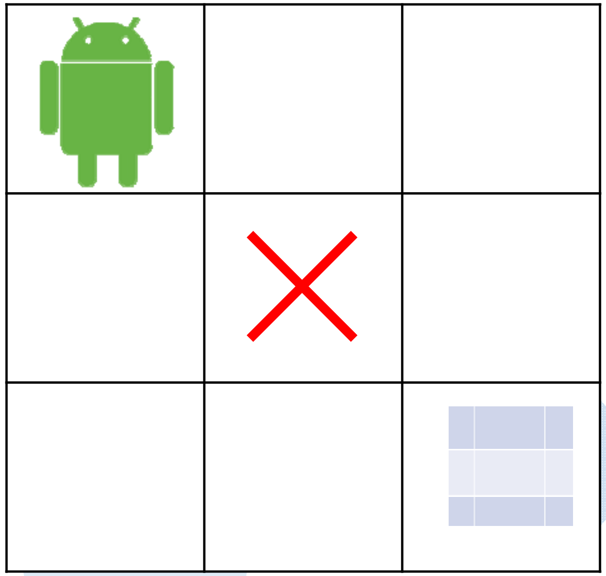

Value-based
Reward
Total Reward
Reward
10
Agent
30
Action Up
Action Down
10
Action Up
20
30
Action Down
40
10
Action Up
40
30
Action Down
60

Q-learning

- Keep update a so-called "**Q-table**".
- Q-table represents the expected future total reward for each *(state, action)* pair.
- Q-table is built and updated through exploration.

Q-table

State	a_1	a_2	$\cdots$	a_L
s_1				
s_2				
$\cdots$				
s_N				



Algorithm

Initialize Q(s, a) arbitrarily
Repeat (for each episode):
  Initialize s
  Repeat(for each step of episode):
    *Choose **a** from **s** using policy derived from Q (e.g. , **Ɛ-greedy**)*
    Take action a, observe r, s'

$$Q(s,a) \leftarrow Q(s,a) + \alpha\,[\underbrace{r_t + \gamma \, max_{a'}\; Q(s',a')}_{\text{Estimate of optimal future reward}} - Q(s,a)]$$

$$s \leftarrow s';$$

  until s is terminal

α: learning rate

γ: discount factor

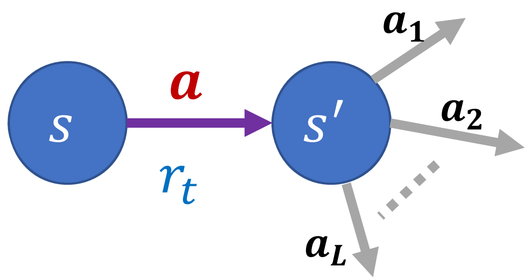
$$(1-\alpha)Q(s,a) + \alpha\,[r_t + \gamma \, max_{a'}\; Q(s',a')]$$

Choose the best action

Q-table

State	a_1	a_2	$\cdots$	a_L
s_1				
s_2				
$\cdots$				
s_N				

◆ After taking action a at state S and receiving the reward r_t, the expected reward of $Q(s,a)$ will be updated.



Discount Factor in Q learning

- The discount factor γ determines the importance of future rewards.

$$Q(s_1, a) = (1-\alpha)Q(s_1, a) + \alpha\, [r_t + \gamma\, max_{a'}\ Q(s_1{}', a')]$$

$$\alpha = 1 \quad Q(s_1, a) = r_1 + \gamma Q(s_2, a\,) = r_1 + \gamma[r_2 + \gamma Q(s_3, a\,)]$$

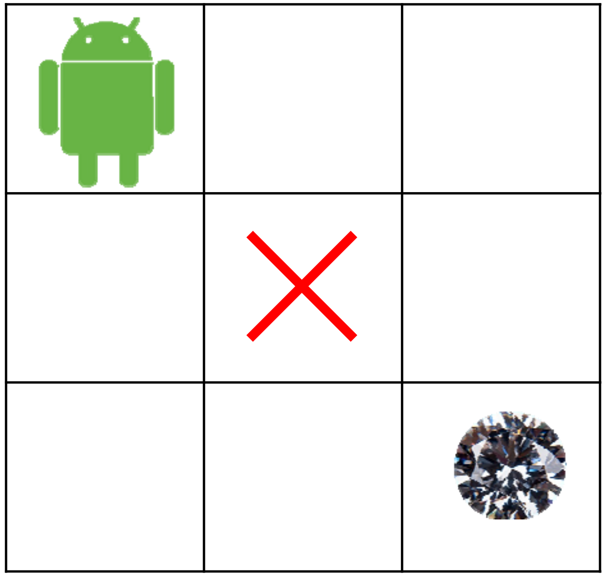
$$Q(s_1, a) = r_1 + \gamma r_2 + \gamma^2\, r_3 + \gamma^3\, r_4\ + \ldots$$

$$\gamma = 1 \qquad Q(s_1, a) = r_1 + r_2 + r_3 + r_4 + \ldots$$

$$\gamma = 0 \qquad Q(s_1, a) = r_1$$

Q-learning example

- We want to get the diamond.

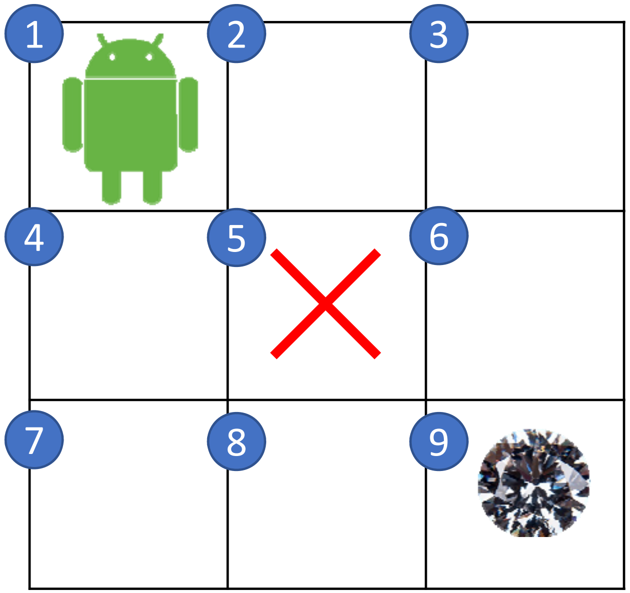


Action	Reward
Nothing	-1
Trap	-100
Diamond	10

Game over

Initial Q-table

- ◈We want to get a diamond.
- ◈X represents the direction we can't take.



Action	Reward
Nothing	-1
Trap	-100
Diamond	10

Q-table

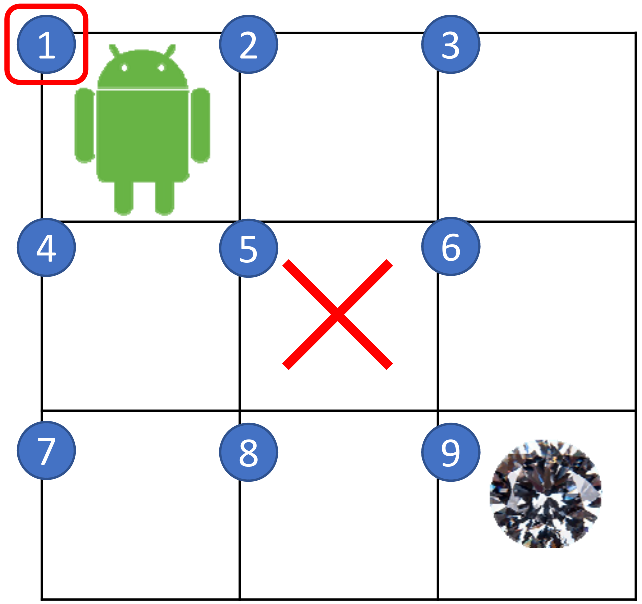
	Up	Down	Left	Right
1	X	0	X	0
2	X	0	0	0
3	X	0	0	X
4	0	0	X	0
5	0	0	0	0
6	0	0	0	X
7	0	X	X	0
8	0	X	0	0
9	0	X	0	X

Q-learning

Episode 1

Action	Reward
Nothing	-1
Trap	-100
Diamond	10

- We will find out the action that can achieve the expected maximum reward according to the Q-table.



	Up	Down	Left	Right
1	X	0	X	0
2	X	0	0	0
3	X	0	0	X
4	0	0	X	0
5	0	0	0	0
6	0	0	0	X
7	0	X	X	0
8	0	X	0	0
9	0	X	0	X

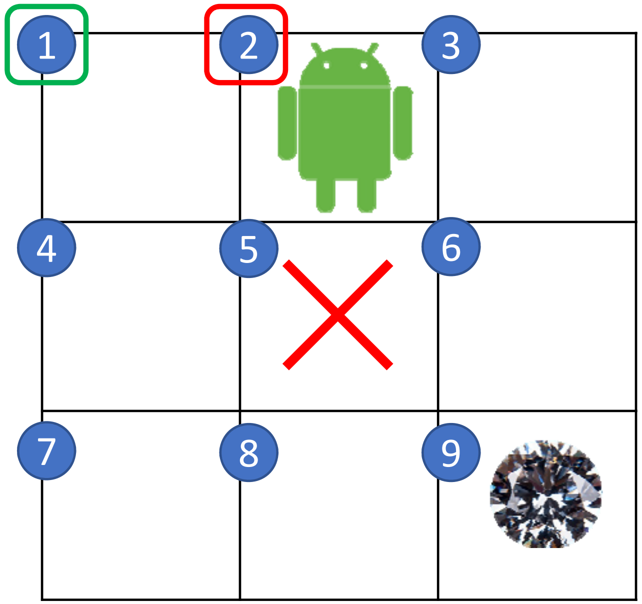
Q-learning

Episode 1

Action	Reward
Nothing	-1
Trap	-100
Diamond	10

- Update *Q(1, Right)*
- $(1-\alpha)Q(1, Right) + \alpha\,[-1 + \boxed{\gamma\, max_{a'}\ Q(2, a')}]$ = -1

1



Reward = -1

	Up	Down	Left	Right
1	X	0	X	**-1**
2	X	0	0	0
3	X	0	0	X
4	0	0	X	0
5	0	0	0	0
6	0	0	0	X
7	0	X	X	0
8	0	X	0	0
9	0	X	0	X

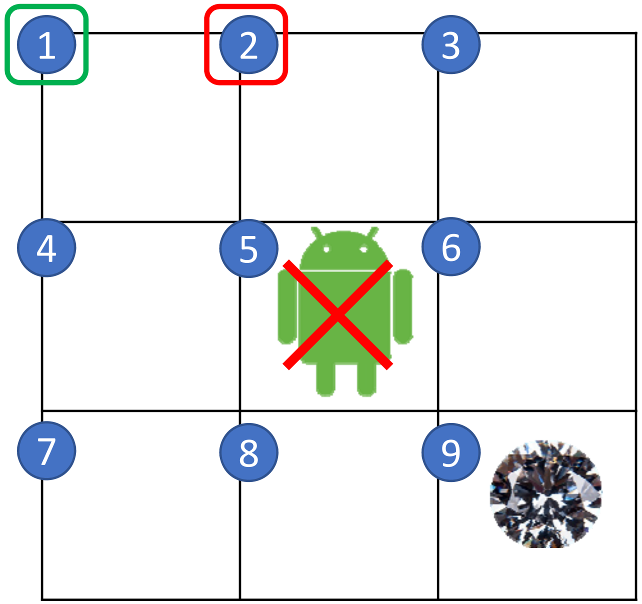
Update

Q-learning

Episode 1

Action	Reward
Nothing	-1
Trap	-100
Diamond	10

- Update *Q(2, Down)*
- $(1-\alpha)Q(2, Down) + \alpha\,[-100 + \gamma\, max_{a'}\; Q(5, a')]$= -100



Reward = -100

Game over

	Up	Down	Left	Right
1	X	0	X	-1
2	X	**-100**	0	0
3	X	0	0	X
4	0	0	X	0
5	0	0	0	0
6	0	0	0	X
7	0	X	X	0
8	0	X	0	0
9	0	X	0	X

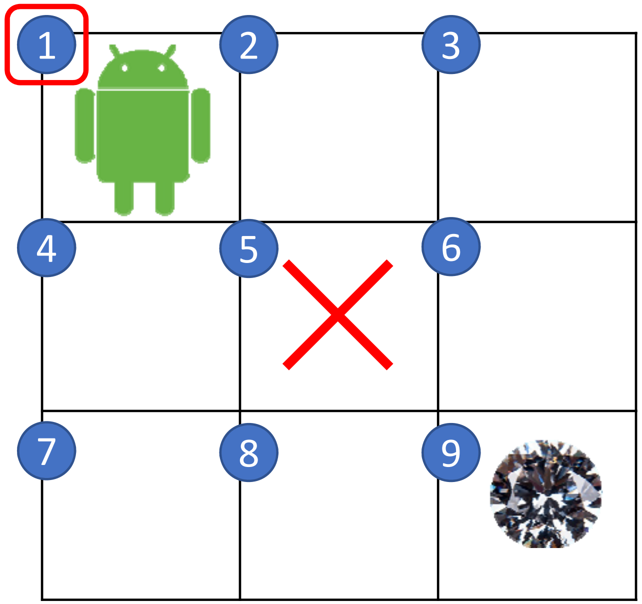
Update

Q-learning

Episode 2

Action	Reward
Nothing	-1
Trap	-100
Diamond	10

- We choose the action that we expect to get the most reward in the future.



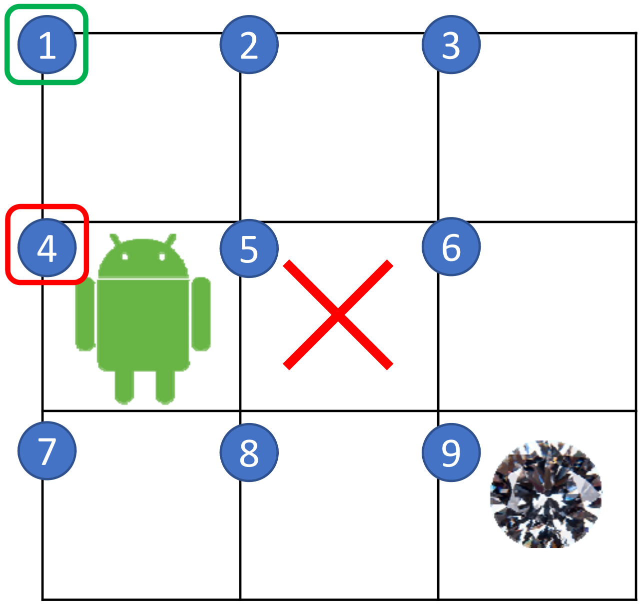
	Up	Down	Left	Right
1	X	0	X	-1
2	X	-100	0	0
3	X	0	0	X
4	0	0	X	0
5	0	0	0	0
6	0	0	0	X
7	0	X	X	0
8	0	X	0	0
9	0	X	0	X

Q-learning

Episode 2

Action	Reward
Nothing	-1
Trap	-100
Diamond	10

- Update *Q(1, Down)*
- $(1-\alpha)Q(1,Down) + \alpha\,[-1 + \gamma \max_{a'} Q(4,a')] = -1$



Reward = -1

	Up	Down	Left	Right
1	X	**-1**	X	-1
2	X	-100	0	0
3	X	0	0	X
4	0	0	X	0
5	0	0	0	0
6	0	0	0	X
7	0	X	X	0
8	0	X	0	0
9	0	X	0	X

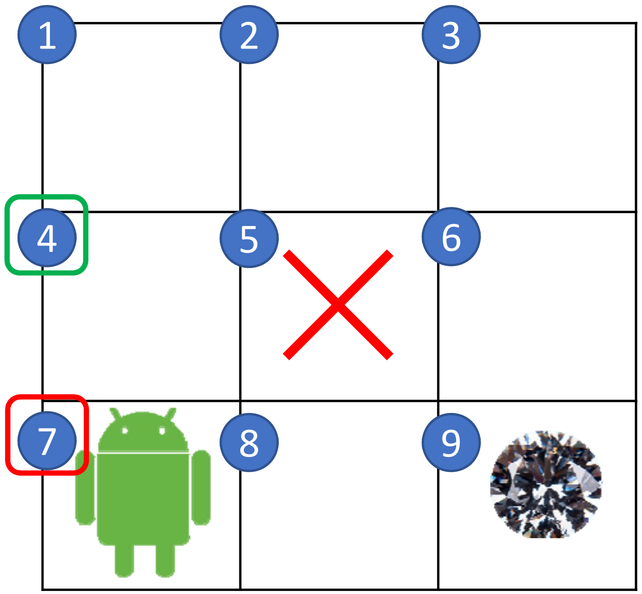
Update

Q-learning

Episode 2

Action	Reward
Nothing	-1
Trap	-100
Diamond	10

- Update *Q(4, Down)*
- $(1-\alpha)Q(4, Down) + \alpha\,[-1 + \gamma\, max_{a'}\ Q(7, a')]$= -1



Reward = -1

	Up	Down	Left	Right
1	X	-1	X	-1
2	X	-100	0	0
3	X	0	0	X
4	0	**-1**	X	0
5	0	0	0	0
6	0	0	0	X
7	0	X	X	0
8	0	X	0	0
9	0	X	0	X

Update

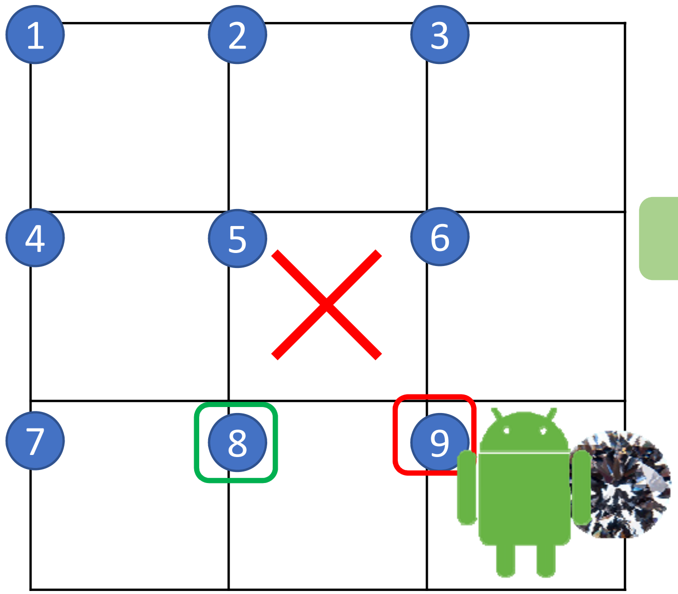
Q-learning

Episode 2

Action	Reward
Nothing	-1
Trap	-100
Diamond	10

◇Update *Q(8, Right)*

◇$(1-\alpha)Q(8, Right) + \alpha\,[10 + \gamma\, max_{a'}\ Q(9, a')]$= 10



Reward = 10

Game over

	Up	Down	Left	Right
1	X	0	X	-1
2	X	-100	0	0
3	X	**0**	0	X
4	0	**-1**	X	0
5	0	0	0	0
6	0	0	0	X
7	0	X	X	-1
8	0	X	0	**10**
9	0	X	0	X

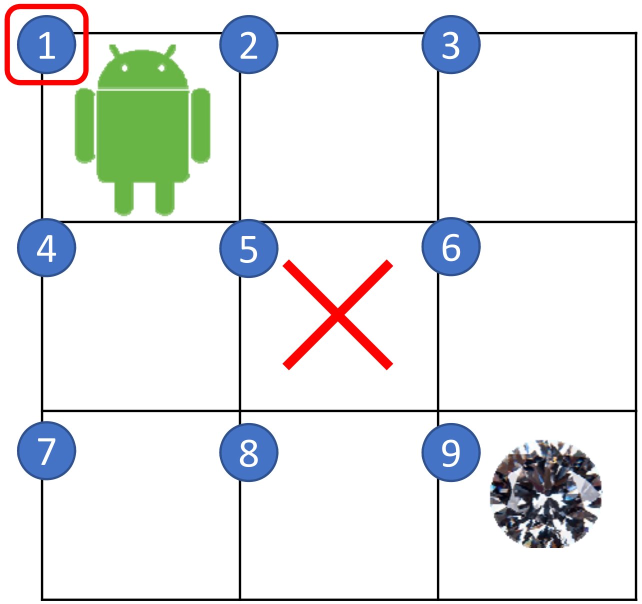
Update

Q-learning

Episode 400

Action	Reward
Nothing	-1
Trap	-100
Diamond	10

◇After many episodes, we will have an optimized Q-table.



$\alpha = 1$
$\gamma = 1$
$\varepsilon = 1$

$\alpha = 0.9$
$\gamma = 0.95$
$\varepsilon = 0.9$

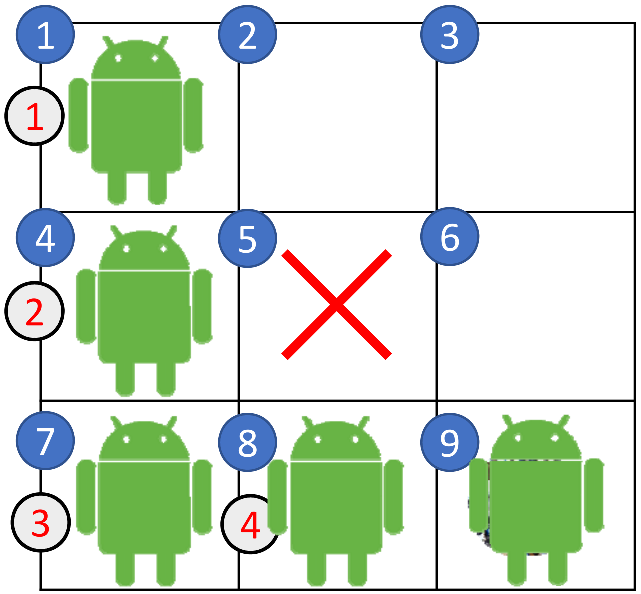
	Up	Down	Left	Right
1	X	5.72	4.44	5.72
2	X	-100	4.44	7.07
3	X	8.5	5.72	X
4	4.44	7.07	X	-100
5	0	0	0	0
6	0	10	-100	X
7	5.72	X	X	8.5
8	0	X	7.07	10
9	0	0	0	0

Q-learning

Episode 400

Action	Reward
Nothing	-1
Trap	-100
Diamond	10

◇We can find the best path according to Q-table.



	Up	Down	Left	Right
1	X	7	X	7
2	X	-100	6	8
3	X	9	7	X
4	6	8	X	-100
5	0	0	0	0
6	0	10	-100	X
7	7	X	X	9
8	0	X	8	10
9	0	0	0	0

Exploration and Exploitation

- Sometime we need to explore new paths for learning better.

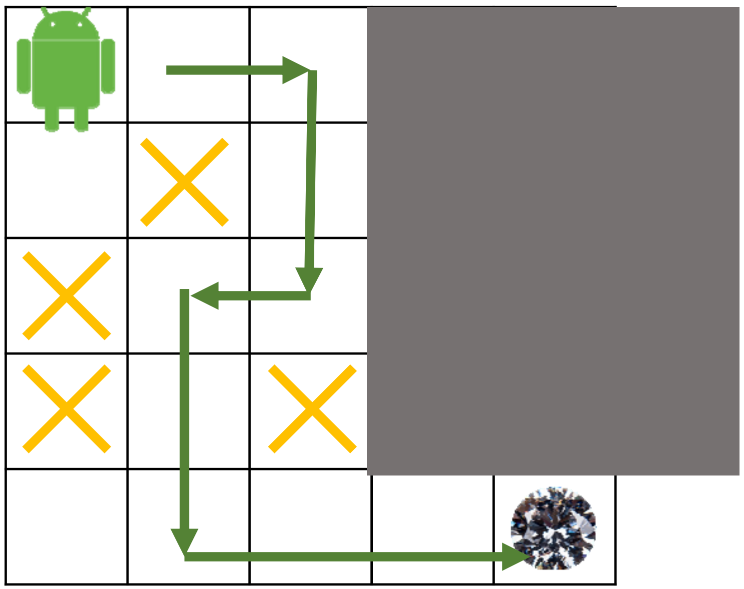
- Exploration
  - Try different actions even if you don't get the best reward.

- Exploitation
  - Choose the best action that gets the highest reward.

Exploration and Exploitation

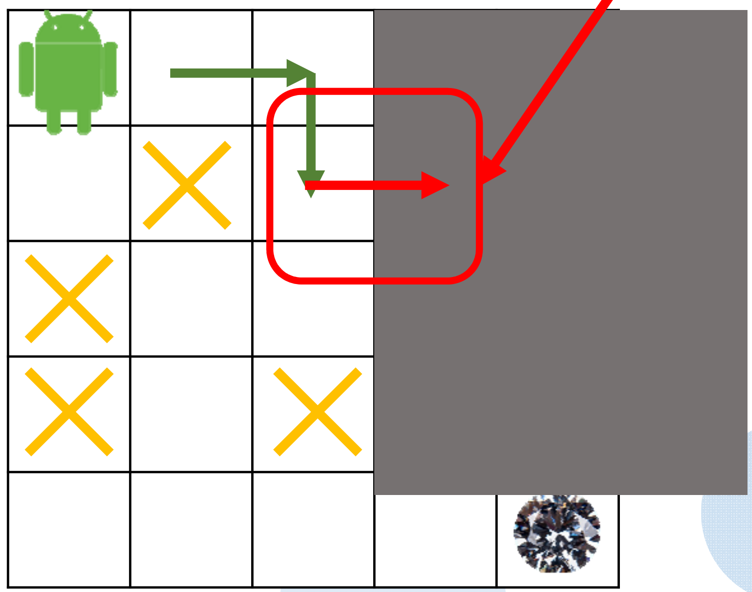
Exploitation

The path we chose to explore



Exploration

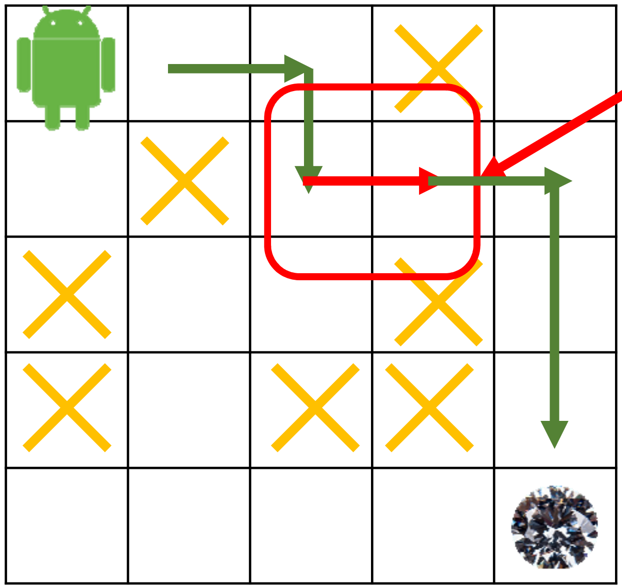
Try to explore new ways



Exploration and Exploitation

Exploration

Sometimes we can get better results(Shorter path).



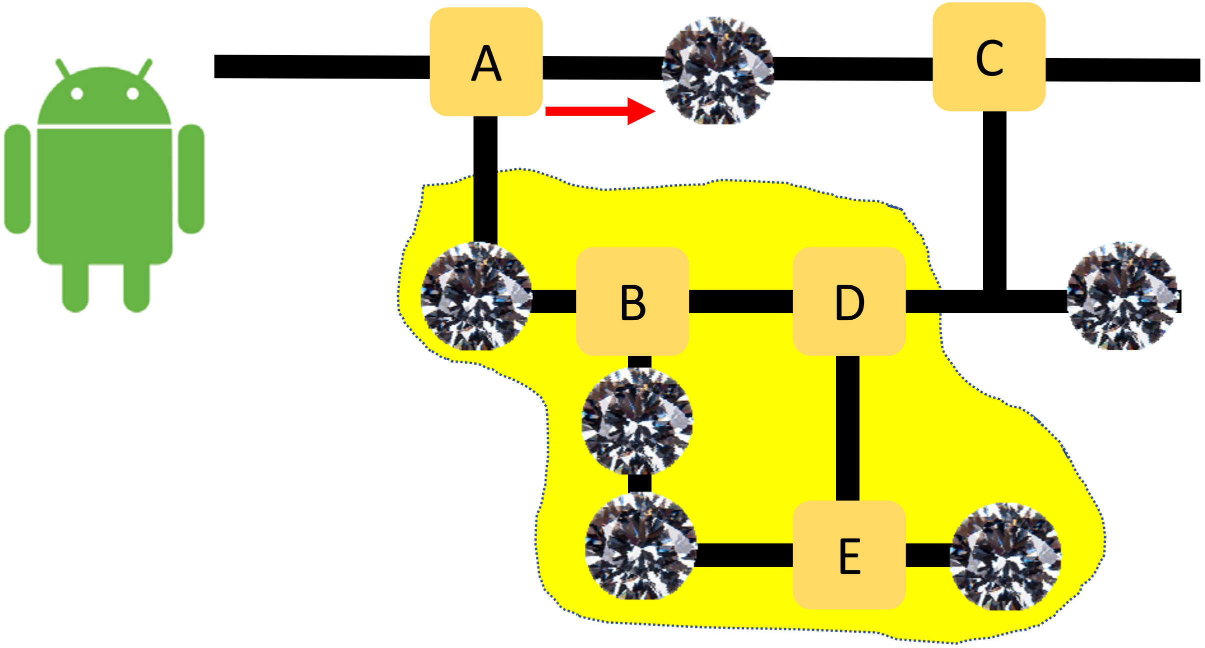
In Q-learning, this means that we sometimes choose actions that currently will not receive the most expected value.

	Up	Down	Left	Right
1	X	-20	X	10
2	X	-45	-53	-30
3	X	-40	20	X
…	…	…	…	…
n	…	…	…	…

Maximum expected reward

Epsilon Greedy

- Assume we just use '**Greedy**' policy.
- First time we choose '**Right**' in 'A'



Epoch = 0

state	Right	Down
A	1	0
B	0	0
C	0	0
D	0	0
E	0	0

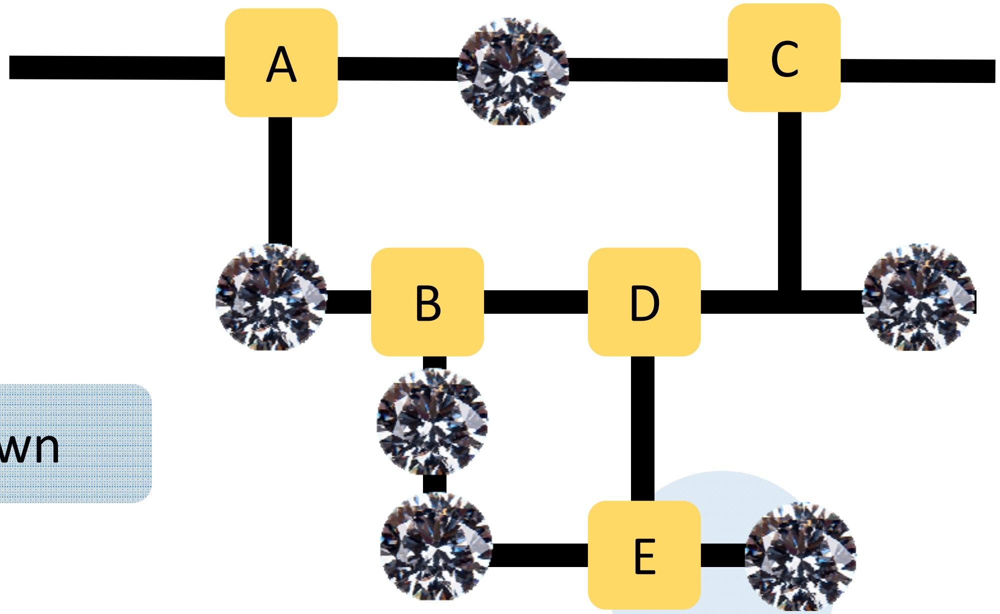
- We will never choose action '**Down**' at state **A**.

Epsilon Greedy

◇Sometime we choose action randomly in order to explore all possibilities

	Right	Down
A	1	0
B	0	0
C	0	0
D	0	0
E	0	0

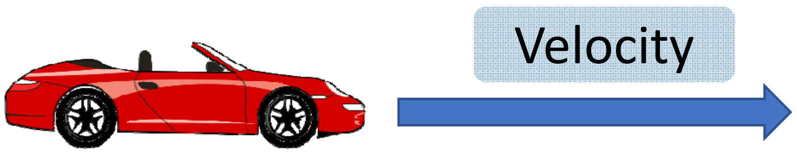
Down



Q-table explosion

◈If the number of *states/actions* is infinite or very large, Q-table becomes infeasible.

Infinite

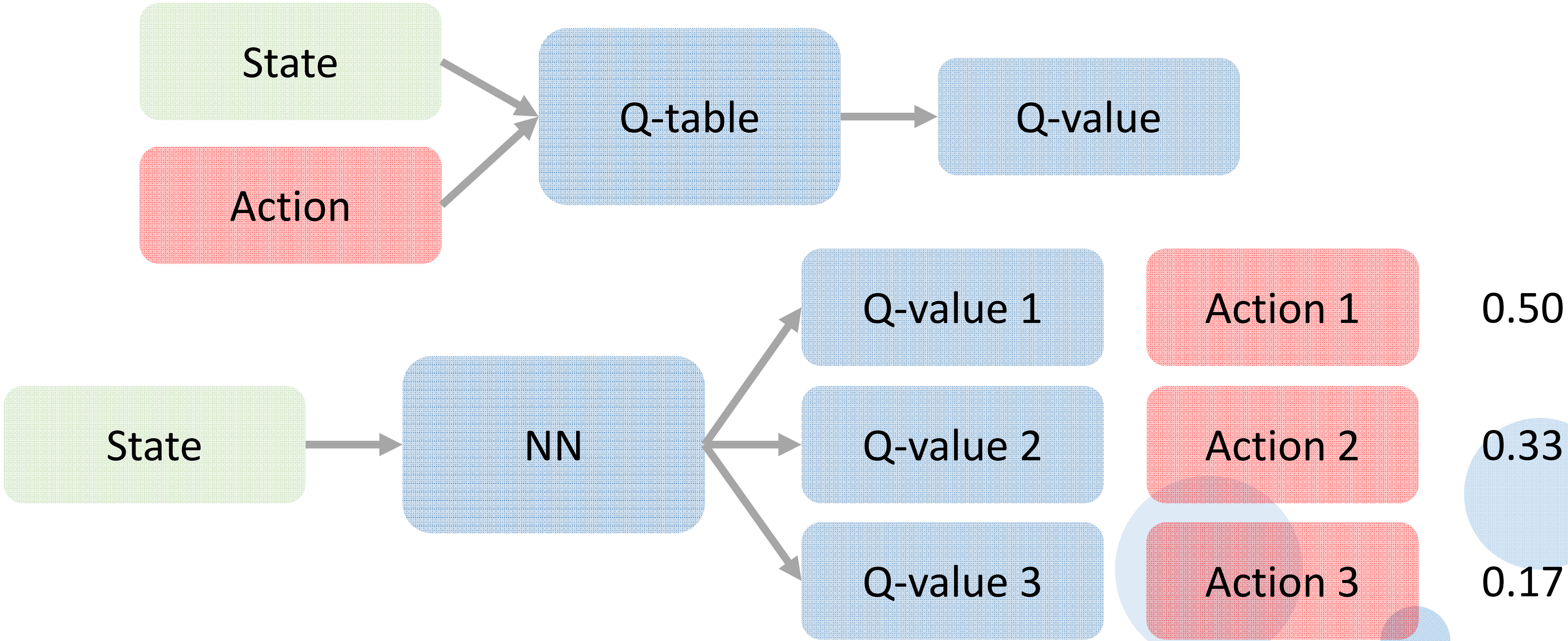


Large

◆ *How many possible states in GO?*

Deep Q Network(DQN)

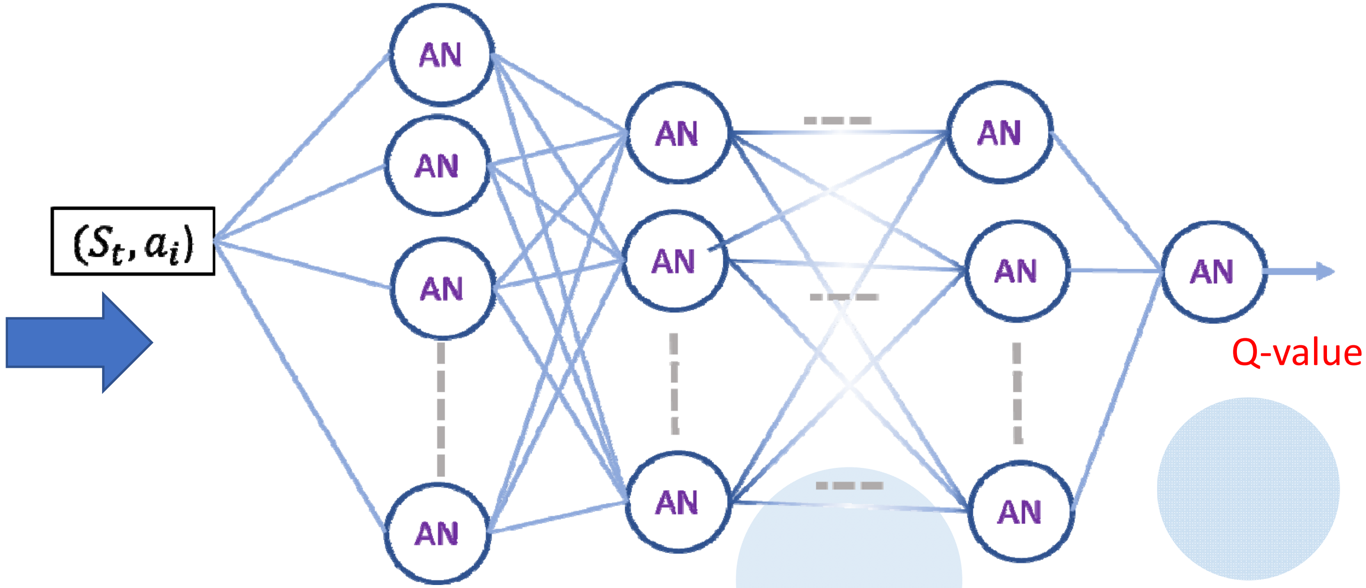
- Generate Q-value for each *(state, action)* pair.



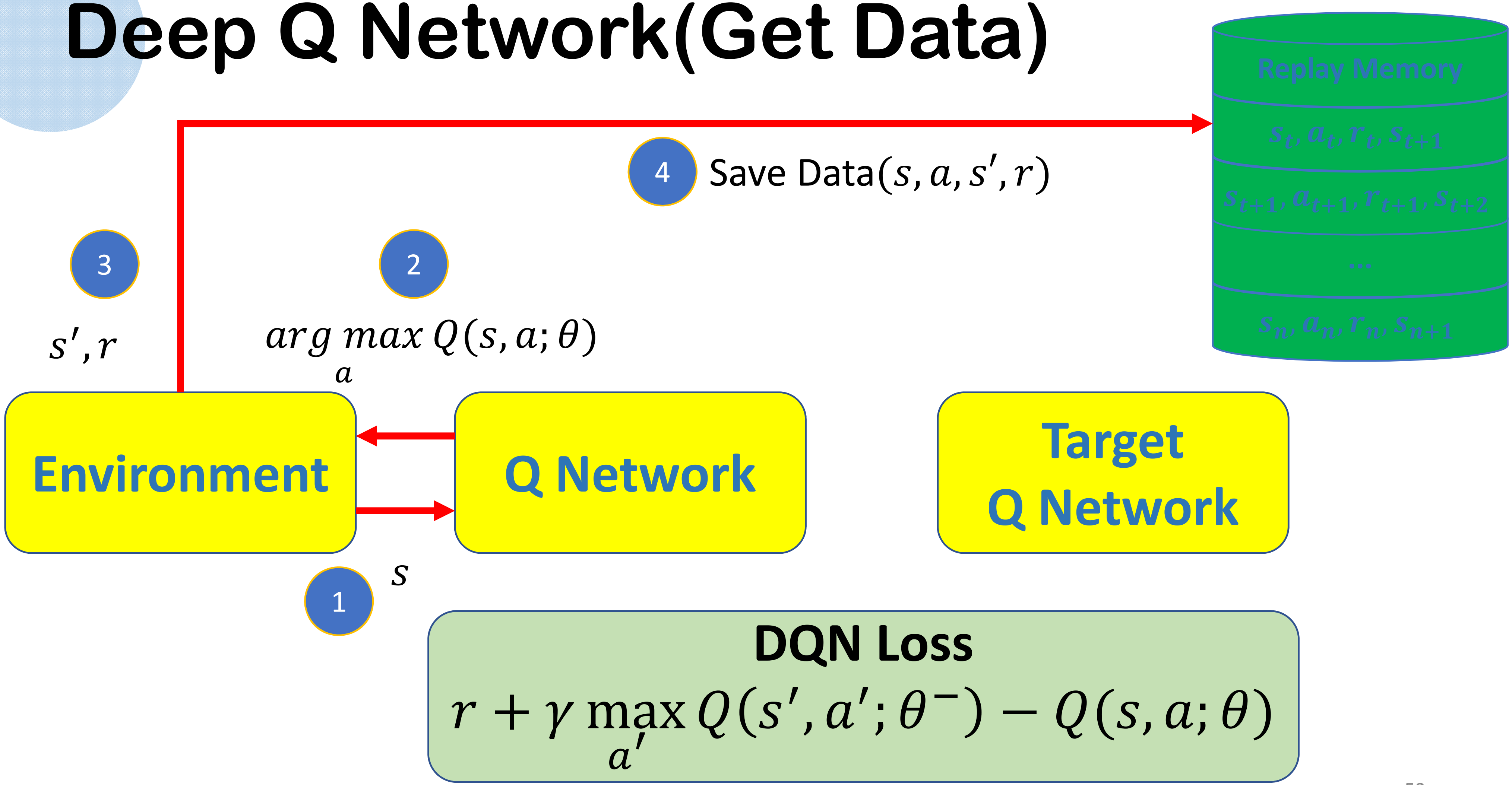
Build Q-table Model

- When # of *states* or *actions* is too big to record, we can use Neural Network to replace it.

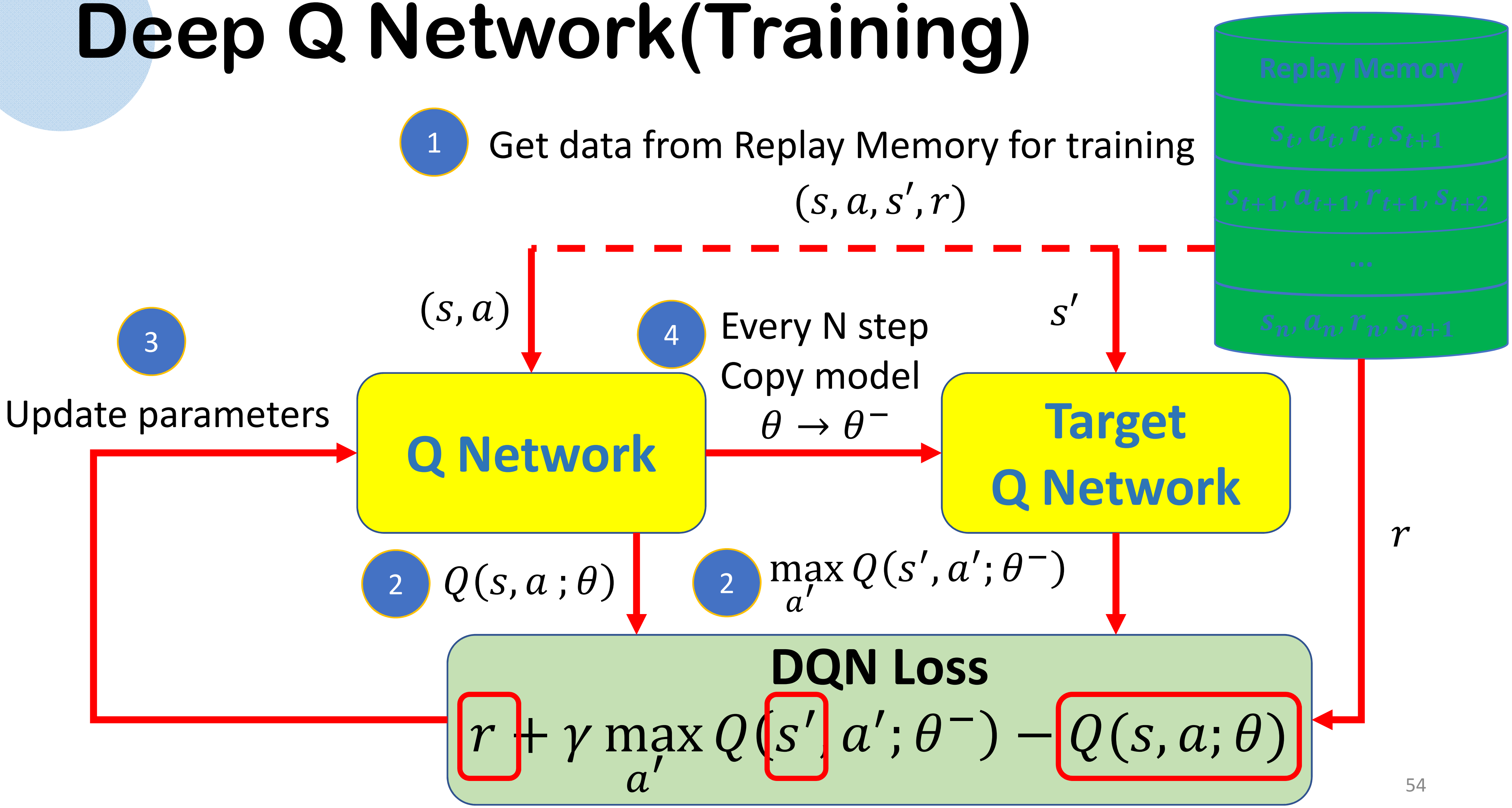
	Right	Down
S1	1	0
S2	0	0
S3	0	0
S4	0	0
S5	0	0
…	…	…



Deep Q Network(Get Data)



Deep Q Network(Training)



Fixed Q-targets

- Use two Network, Q Network and Target Q Network.
- Every N step, we will copy the Q Network to the Target Q Network.
- Target Q Network is the old Q Network

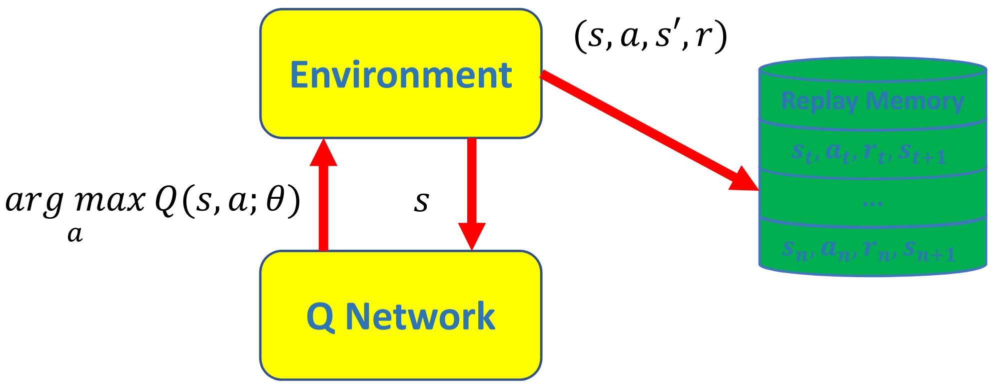
$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r_t + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$$

Q Network $\longrightarrow$ Target Q Network

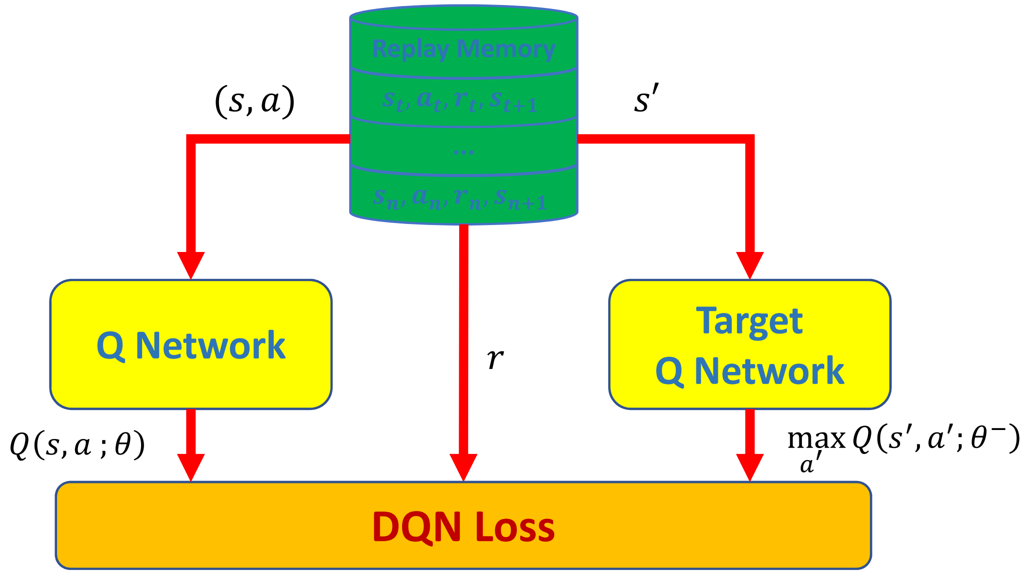
Every N step, copy model to Target Q Network

Experience Replay

Add data



Use data



Double Deep Q Network

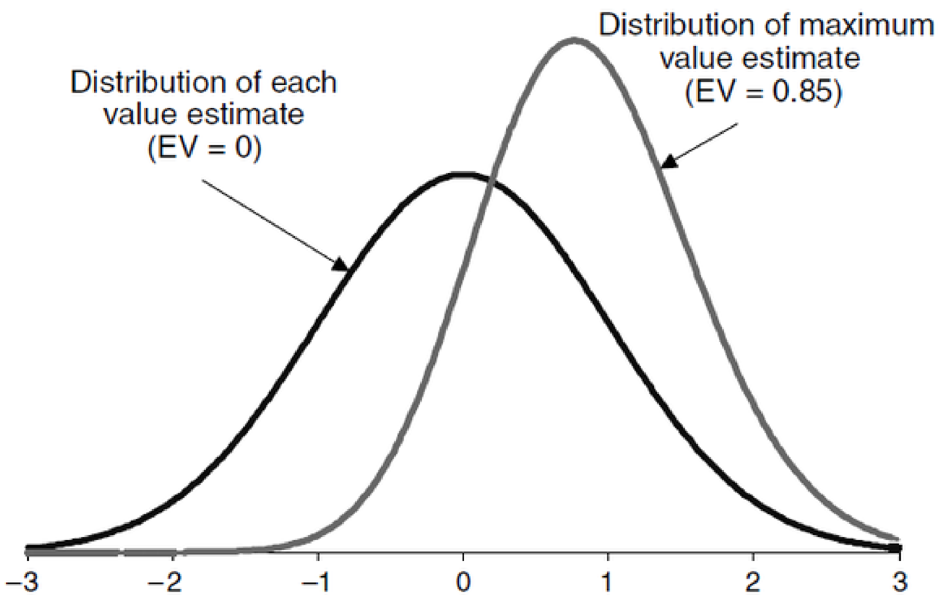
- Q-learning and Deep Q Learning tends to overestimate q-values.
- Because we take the max over all actions.

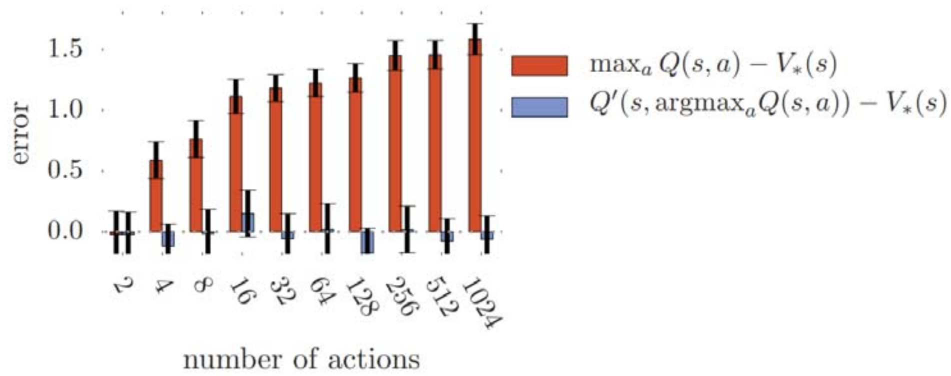
$$Q(s,a) \leftarrow Q(s,a) + \alpha\,[r_t + \gamma \boxed{max_{a'}\ Q(s',a')} - Q(s,a)]$$

- If all values were equally overestimated this would be no problem.
- But if the overestimations are not uniform, this might slow downlearning.

Overestimated

- *X1, X2 is sequence*
- $E(\max(X1, X2)) \geq \max(E(X1), E(X2))$





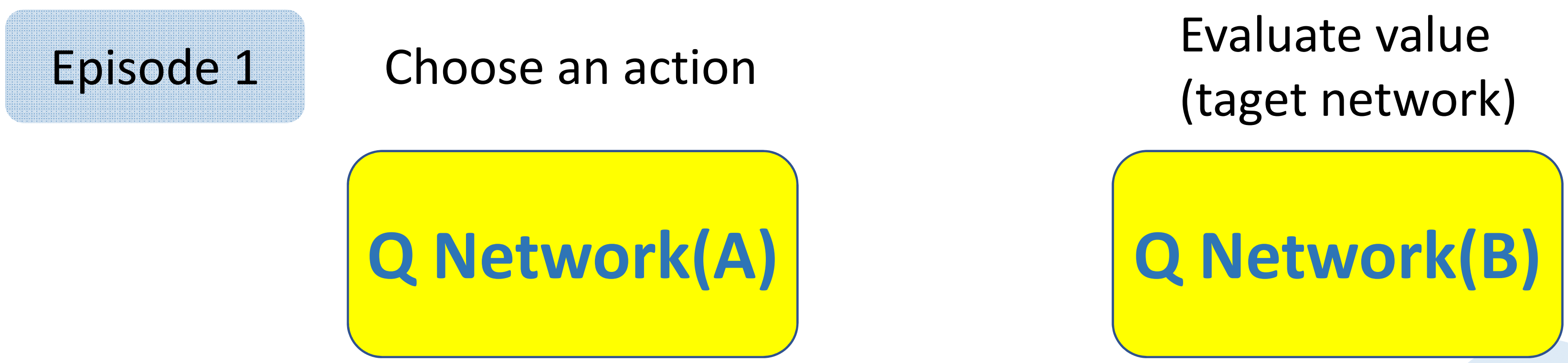
Double Q-learning

- Use two Q network, one of them evaluate, another choose an action.
- Randomly exchange behavior of two Q networks.

Initialize $Q_1(s,a)$ and $Q_2(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily
Initialize $Q_1(\textit{terminal-state}, \cdot) = Q_2(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
    Initialize S
    Repeat (for each step of episode):
        Choose A from S using policy derived from Q_1 and Q_2 (e.g., ε-greedy in $Q_1 + Q_2$)
        Take action A, observe R, S'
        With 0.5 probabilility:
            $Q_1(S,A) \leftarrow Q_1(S,A) + \alpha\Big(R + \gamma Q_2\big(S', \operatorname{argmax}_a Q_1(S',a)\big) - Q_1(S,A)\Big)$
        else:
            $Q_2(S,A) \leftarrow Q_2(S,A) + \alpha\Big(R + \gamma Q_1\big(S', \operatorname{argmax}_a Q_2(S',a)\big) - Q_2(S,A)\Big)$
        $S \leftarrow S'$
    until S is terminal

Double Q-learning

- Use two Q network, one of them evaluate, another choose an action.
- Randomly exchange behavior of two Q networks.



Double Q-learning

- Use two Q network, one of them evaluate, another choose an action.
- Randomly exchange behavior of two Q networks.

Episode 2

Evaluate value
(taget network)

Choose an action

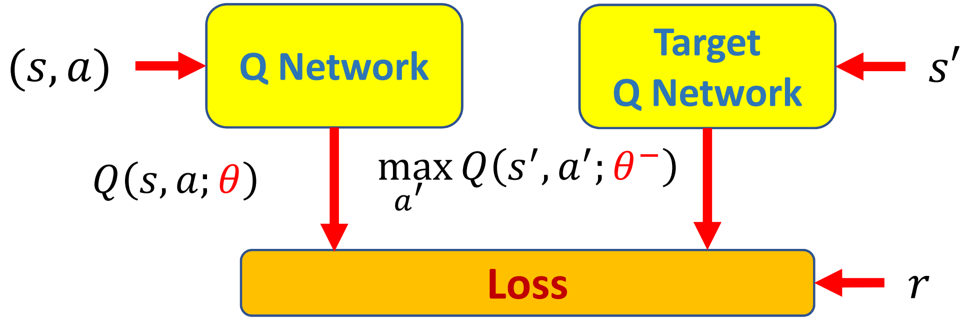
Change Behavier

Q Network(A)

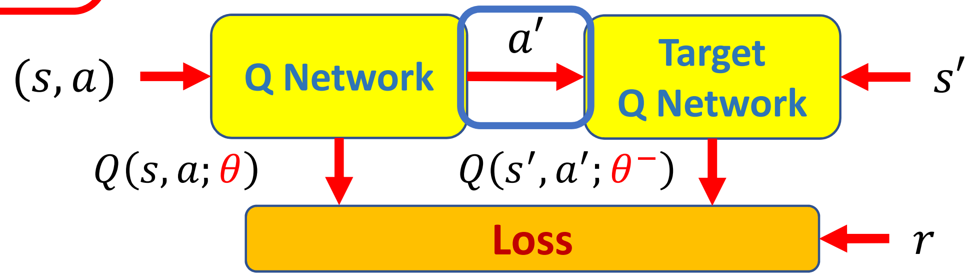
Q Network(B)

Double Deep Q Network

$$Y_t^Q = r_t + \gamma \boxed{max_{a'} \ Q(s', a'; \theta^-)}$$

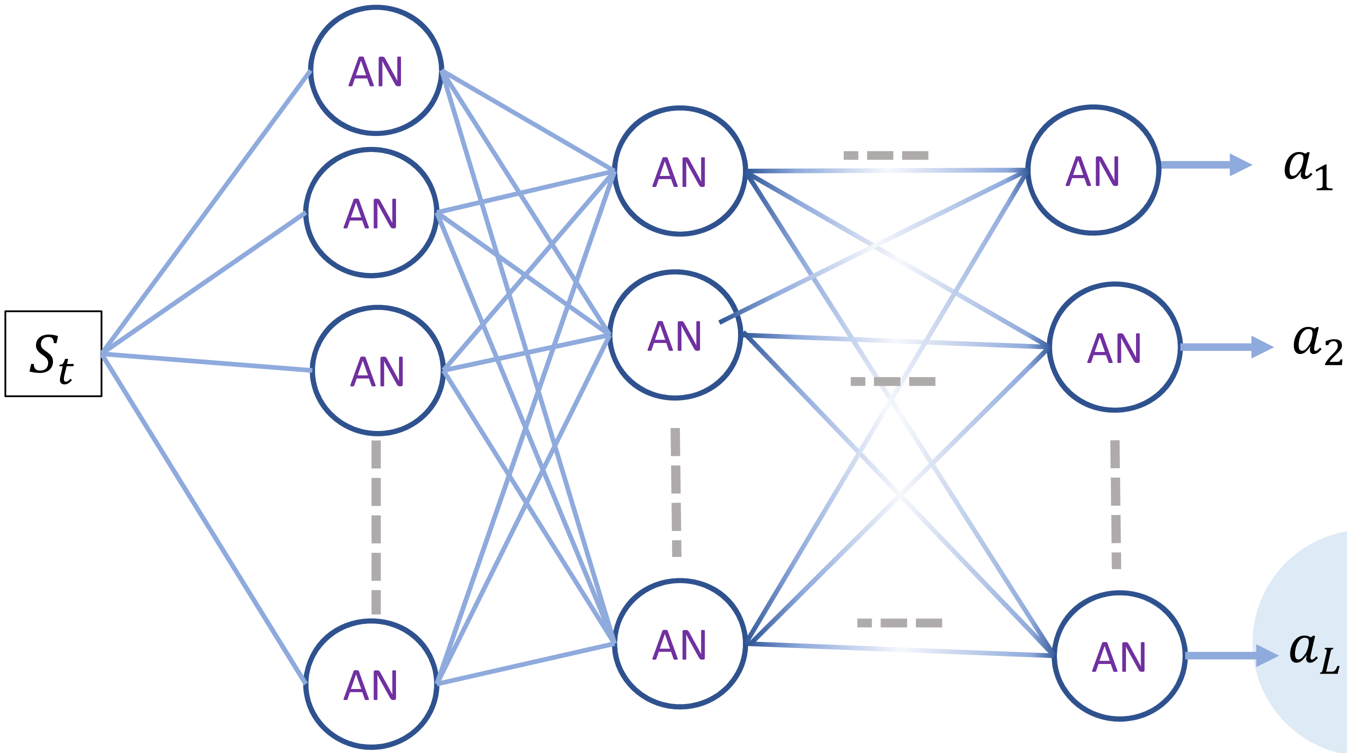


$$Y_t^{Double-Q} = r_t + \gamma \boxed{Q\left(s', \boxed{\arg\max_{a'} Q(s', a'; \theta)}; \theta^-\right)}$$



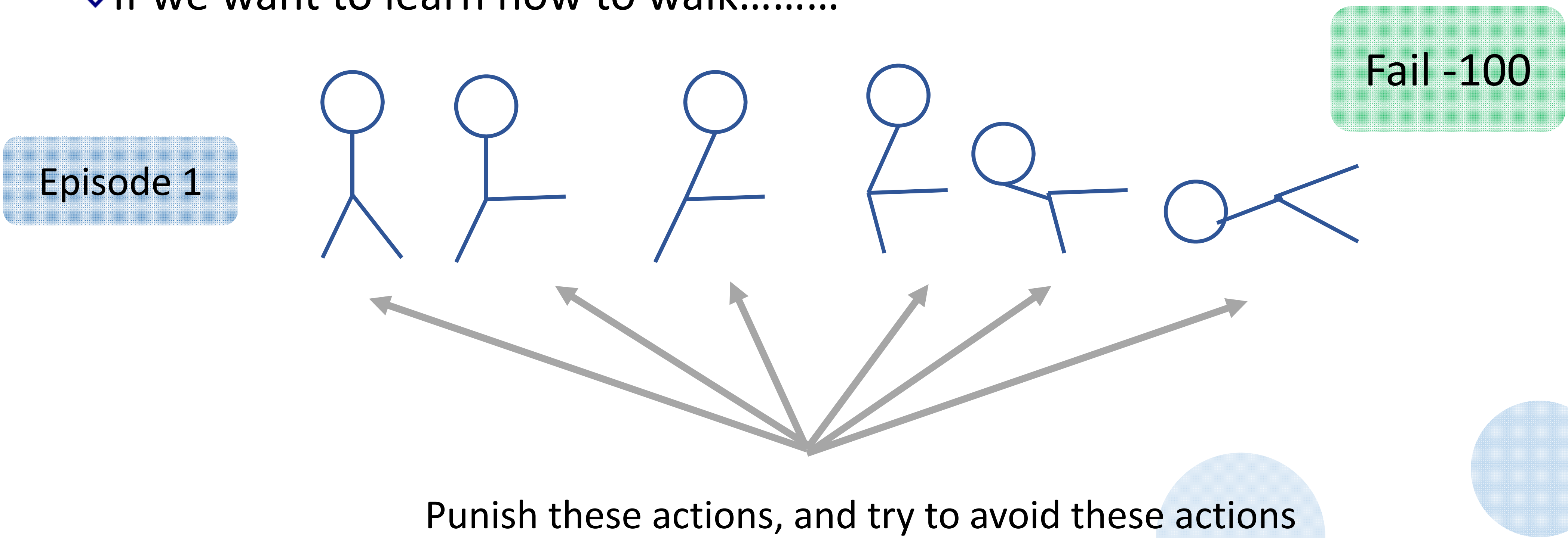
Policy Gradient

- Similar to multi-class classification problem.



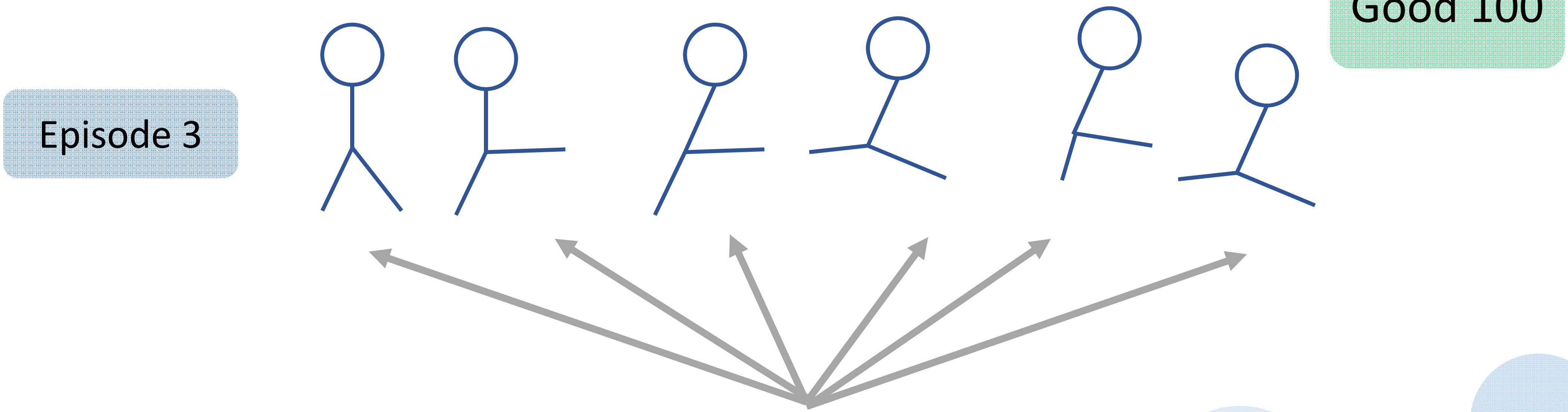
Policy gradient

- If we want to learn how to walk………



Policy gradient

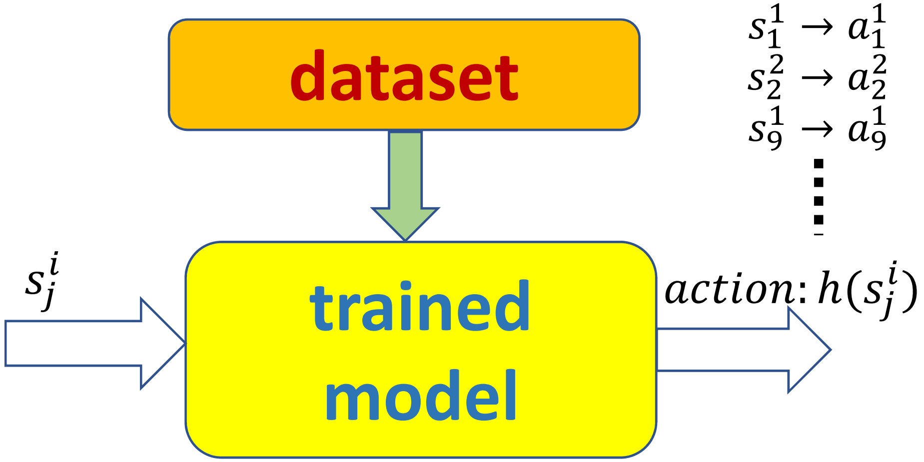
◆If we want to learn how to walk ………



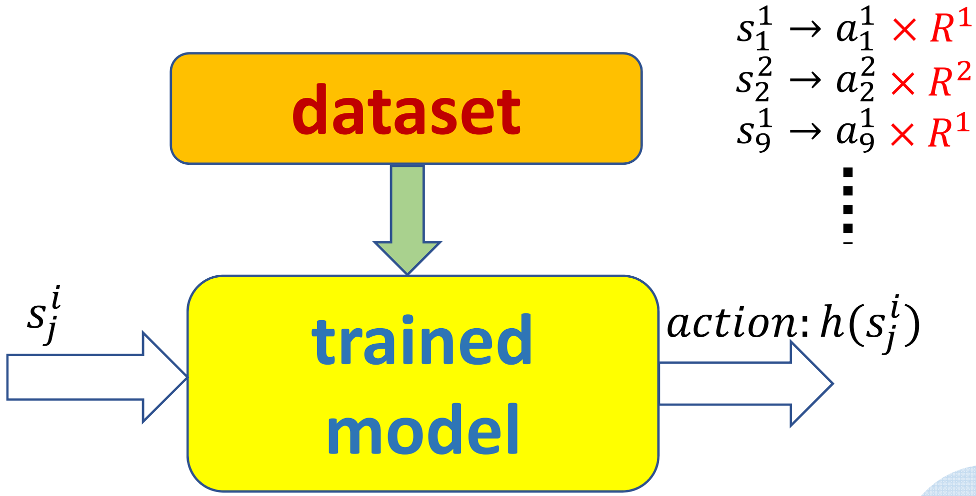
Reward these actions, and choose these actions more often.

Policy Gradient Training

◆Supervised Learning



◆Policy Gradient Training



Episode 1: $[(s_1^1 \to a_1^1), (s_2^1 \to a_2^1), \cdots, (s_{100}^1 \to a_{100}^1)]$ Total reward: $R^1 = r_1^1 + r_2^1 \cdots + r_{100}^1$

Episode 2: $[(s_1^2 \to a_1^2), (s_2^2 \to a_2^2), \cdots, (s_{70}^2 \to a_{70}^1)]$ Total reward: $R^2 = r_1^2 + r_2^2 \cdots + r_{70}^2$

$\vdots$ $\vdots$

Monte Carlo & Temporal-Difference

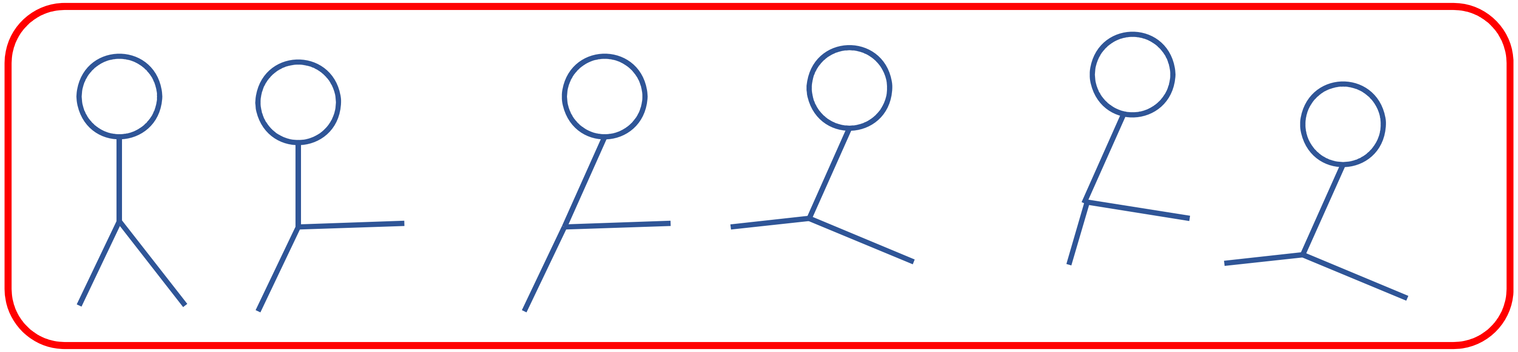
- Monte Carlo Policy Gradient
  - Update every episode.

- Temporal-Difference
  - Update every step.

Monte Carlo

- Monte Carlo Policy Gradient
  - Update every episode.

Episode 1



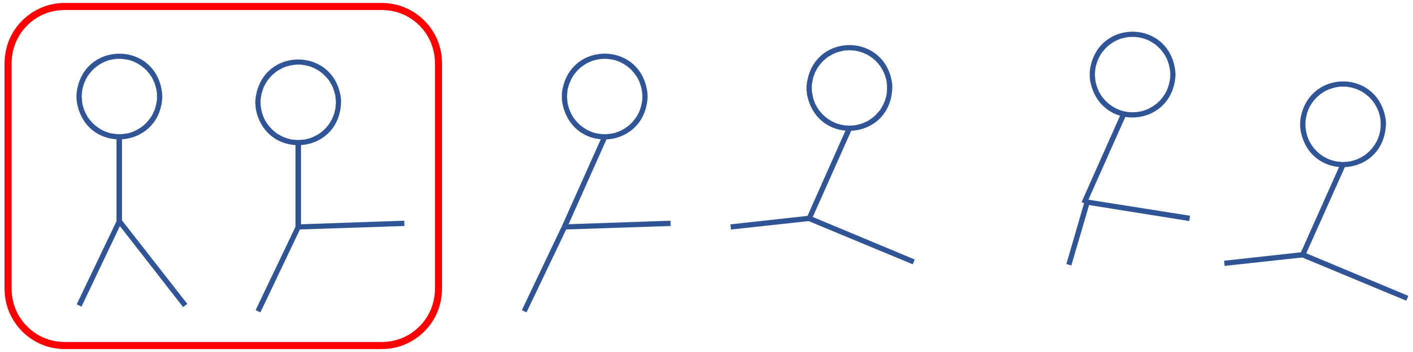
$$episode(1) = \{(s_1, a_1, r_1), (s_2, a_2, r_2), (s_3, a_3, r_3), \dots, (s_n, a_n, r_n)\}$$

Update 1 time

Temporal-Difference

- Temporal-Difference
  - Update every step.

Episode

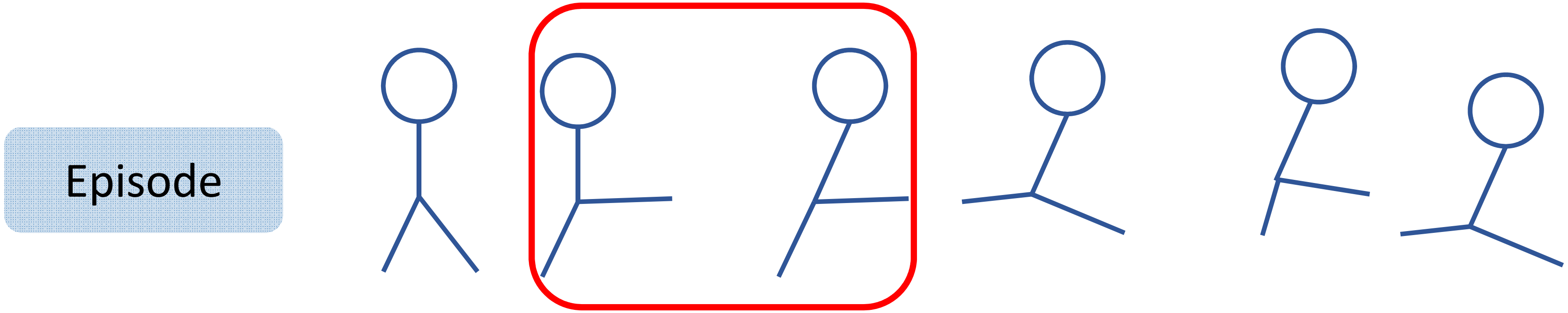


$$episode(1) = \{(s_1, a_1, r_1), (s_2, a_2, r_2), (s_3, a_3, r_3), \ldots, (s_n, a_n, r_n)\}$$

Update 5 times

Temporal-Difference

- Temporal-Difference
  - Update every step.

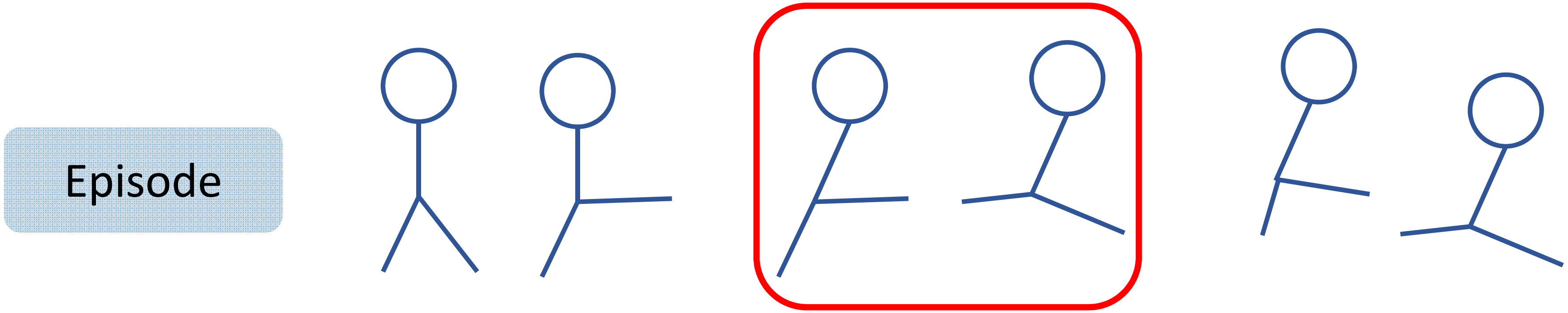


$$episode(1) = \{(s_1, a_1, r_1), (s_2, a_2, r_2), (s_3, a_3, r_3), \ldots, (s_n, a_n, r_n)\}$$

Update 2 times

Temporal-Difference

- Temporal-Difference
  - Update every step.



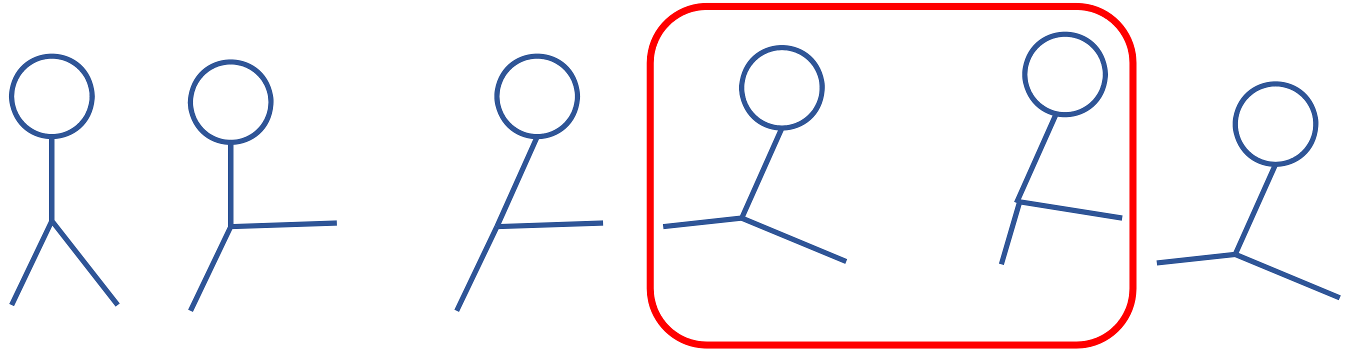
$$episode(1) = \{(s_1, a_1, r_1), (s_2, a_2, r_2), (s_3, a_3, r_3), \ldots, (s_n, a_n, r_n)\}$$

Update 3 times

Temporal-Difference

- Temporal-Difference
  - Update every step.

Episode

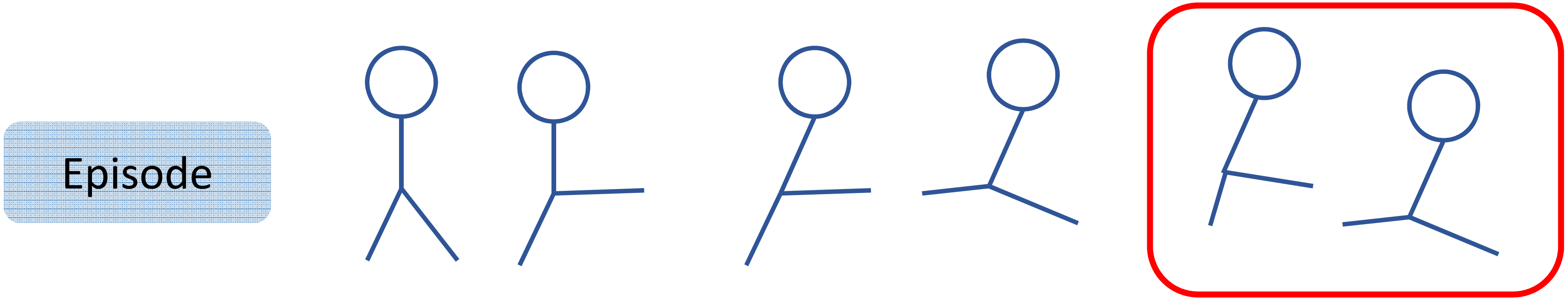


$$episode(1) = \{(s_1, a_1, r_1), (s_2, a_2, r_2), (s_3, a_3, r_3), \ldots, (s_n, a_n, r_n)\}$$

Update 4 times

Temporal-Difference

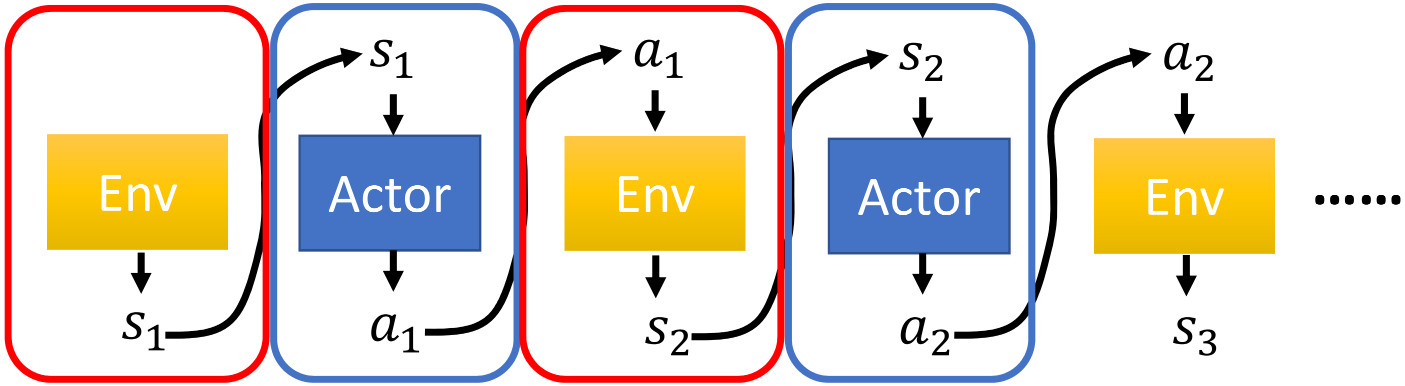
- Temporal-Difference
  - Update every step.



$$episode(1) = \{(s_1, a_1, r_1), (s_2, a_2, r_2), (s_3, a_3, r_3), \ldots, (s_n, a_n, r_n)\}$$

Update 5 times

Actor, Environment, Reward

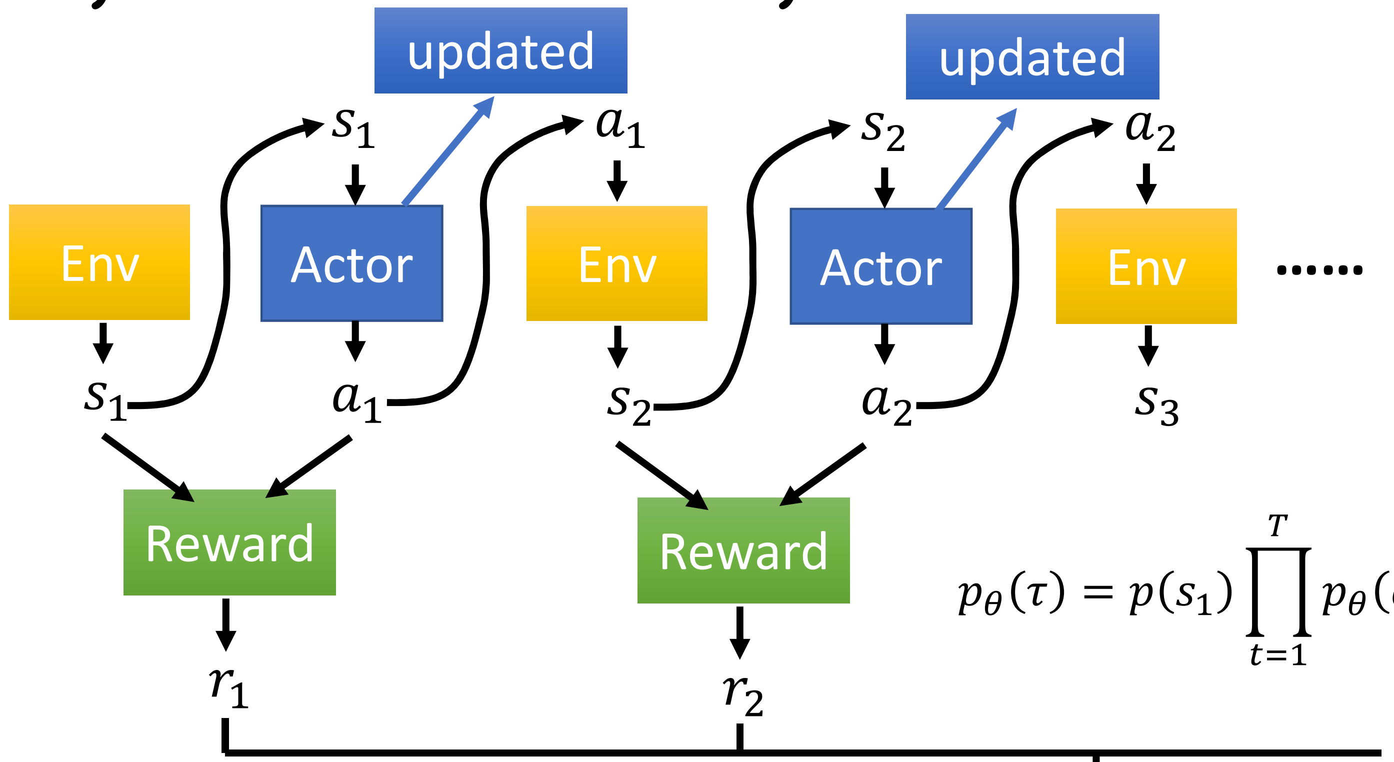


Trajectory $\tau = \{s_1, a_1, s_2, a_2, \cdots, s_T, a_T\}$

$$p_\theta(\tau) = \boxed{p(s_1)}\boxed{p_\theta(a_1|s_1)}\boxed{p(s_2|s_1,a_1)}\boxed{p_\theta(a_2|s_2)}p(s_3|s_2,a_2)\cdots$$

$$= p(s_1)\prod_{t=1}^{T} p_\theta(a_t|s_t)p(s_{t+1}|s_t,a_t)$$

Actor, Environment, Reward



$$p_\theta(\tau) = p(s_1)\prod_{t=1}^{T} p_\theta(a_t|s_t)p(s_{t+1}|s_t,a_t)$$

Expected Reward

$$\bar{R}_\theta = \sum_\tau R(\tau)p_\theta(\tau) = E_{\tau\sim p_\theta(\tau)}[R(\tau)]$$

$$R(\tau) = \sum_{t=1}^{T} r_t$$

Policy Gradient

$$\bar{R}_\theta = \sum_\tau R(\tau) p_\theta(\tau) \qquad \nabla \bar{R}_\theta = ?$$

$$\nabla f(x) = f(x) \nabla \log f(x)$$

$$\nabla \bar{R}_\theta = \sum_\tau R(\tau) \nabla p_\theta(\tau) \quad = \sum_\tau R(\tau) p_\theta(\tau) \frac{\nabla p_\theta(\tau)}{p_\theta(\tau)}$$

$R(\tau)$ do not have to be differentiable

It can even be a black box.

$$= \boxed{\sum_\tau} R(\tau) \boxed{p_\theta(\tau)} \nabla \log p_\theta(\tau)$$

$$= E_{\tau \sim p_\theta(\tau)}[R(\tau) \nabla \log p_\theta(\tau)] \approx \frac{1}{N} \sum_{n=1}^{N} R(\tau^n) \nabla \log p_\theta(\tau^n)$$

$$= \frac{1}{N} \sum_{n=1}^{N} \sum_{t=1}^{T_n} R(\tau^n) \nabla \log p_\theta(a_t^n | s_t^n)$$

Policy Gradient

$$\nabla \bar{R}_\theta = E_{\tau \sim p_\theta(\tau)}[R(\tau)\nabla log p_\theta(\tau)]$$

Given policy π_θ

τ^1: (s_1^1, a_1^1) $R(\tau^1)$

(s_2^1, a_2^1) $R(\tau^1)$

$\vdots$ $\vdots$

τ^2: (s_1^2, a_1^2) $R(\tau^2)$

(s_2^2, a_2^2) $R(\tau^2)$

$\vdots$ $\vdots$

only used once

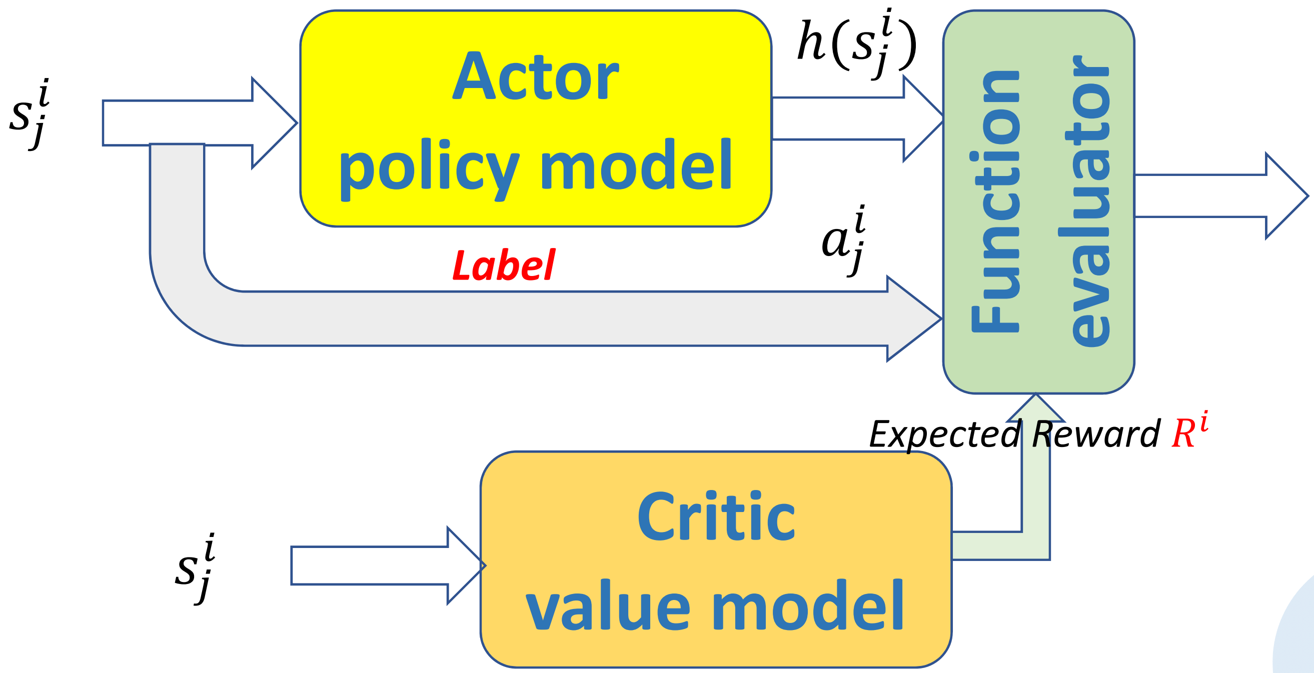
Update Model

$$\theta \leftarrow \theta + \eta \nabla \bar{R}_\theta$$

$$\nabla \bar{R}_\theta = \frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T_n} R(\tau^n)\nabla log p_\theta(a_t^n|s_t^n)$$

Data Collection

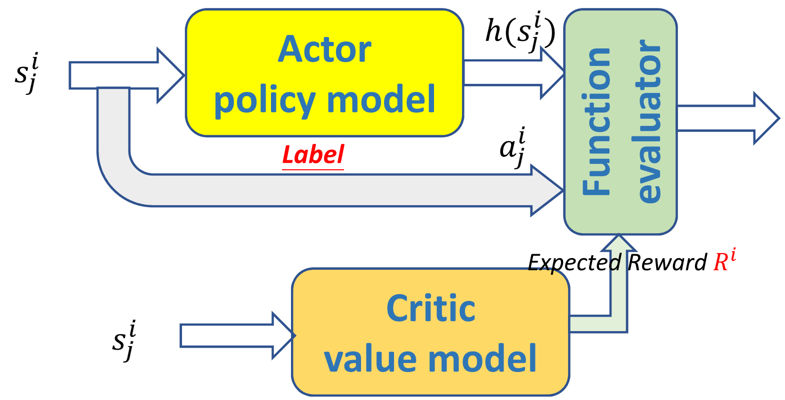
Actor-Critic



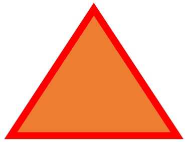
Actor-Critic

- Actor : Policy base
- Critic : Value base

- Actor selects the action, Critic evaluates the quality of the selected action



Actor-Critic



Actor

Critic

left

None

right

straight

Agent

Actor-Critic

Actor

left

right

straight

Agent

Critic

-10

Actor-Critic

Actor

left

right

straight

Agent

Critic

-20

Actor-Critic

Actor

left

right

straight

Agent

Critic

10

Actor-Critic

Actor

left

right

straight

Critic

10

Agent

Actor-Critic

◆Policy Gradient

$$\nabla \bar{R}_\theta \approx \frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T_n}\left(\boxed{\sum_{t'=t}^{T_n}\gamma^{t'-t}r_{t'}^{n}} - b\right)\nabla log p_\theta(a_t^n|s_t^n)$$

G_t^n : obtained via interaction

Actor-Critic

- Policy Gradient
- Use critic to calculate possible future rewards

$$\nabla \bar{R}_\theta \approx \frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T_n}\left(\boxed{\sum_{t'=t}^{T_n}\gamma^{t'-t} r_{t'}^n} - b\right)\nabla \log p_\theta(a_t^n|s_t^n)$$

G_t^n : obtained via interaction

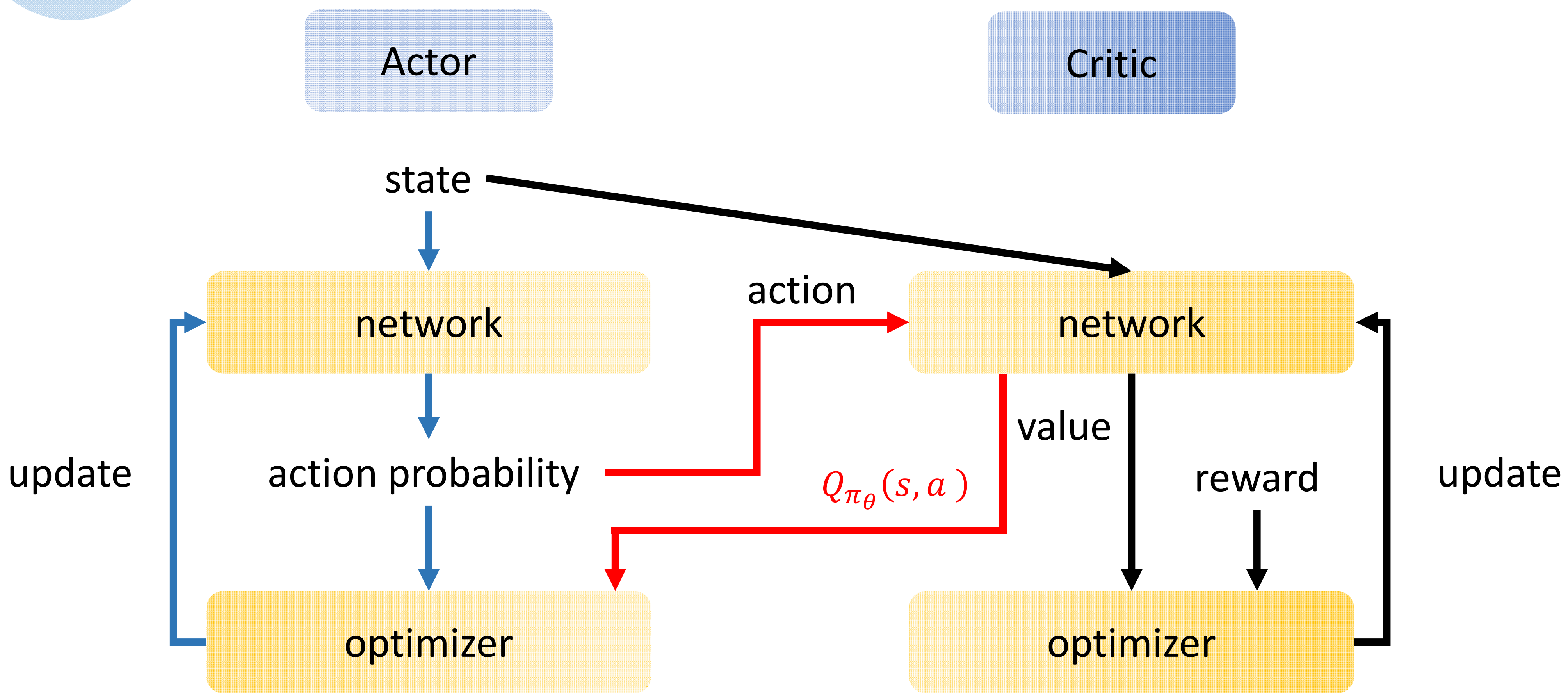
$$E[G_t^n] = Q^{\pi_\theta}(s_t^n, a_t^n)$$

Actor-Critic

$$\begin{aligned}
\nabla_\theta J(\theta) &= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\, G_t\right] && \text{REINFORCE} \\
&= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\; Q^w(s,a)\right] && \text{Q Actor-Critic} \\
&= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\; A^w(s,a)\right] && \text{Advantage Actor-Critic} \\
&= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\; \delta\right] && \text{TD Actor-Critic}
\end{aligned}$$

Actor-Critic

$$\nabla_\theta J(\theta) = E_{\pi_\theta}[\nabla_\theta log\pi_\theta(s,a) Q_{\pi_\theta}(s,a)]$$

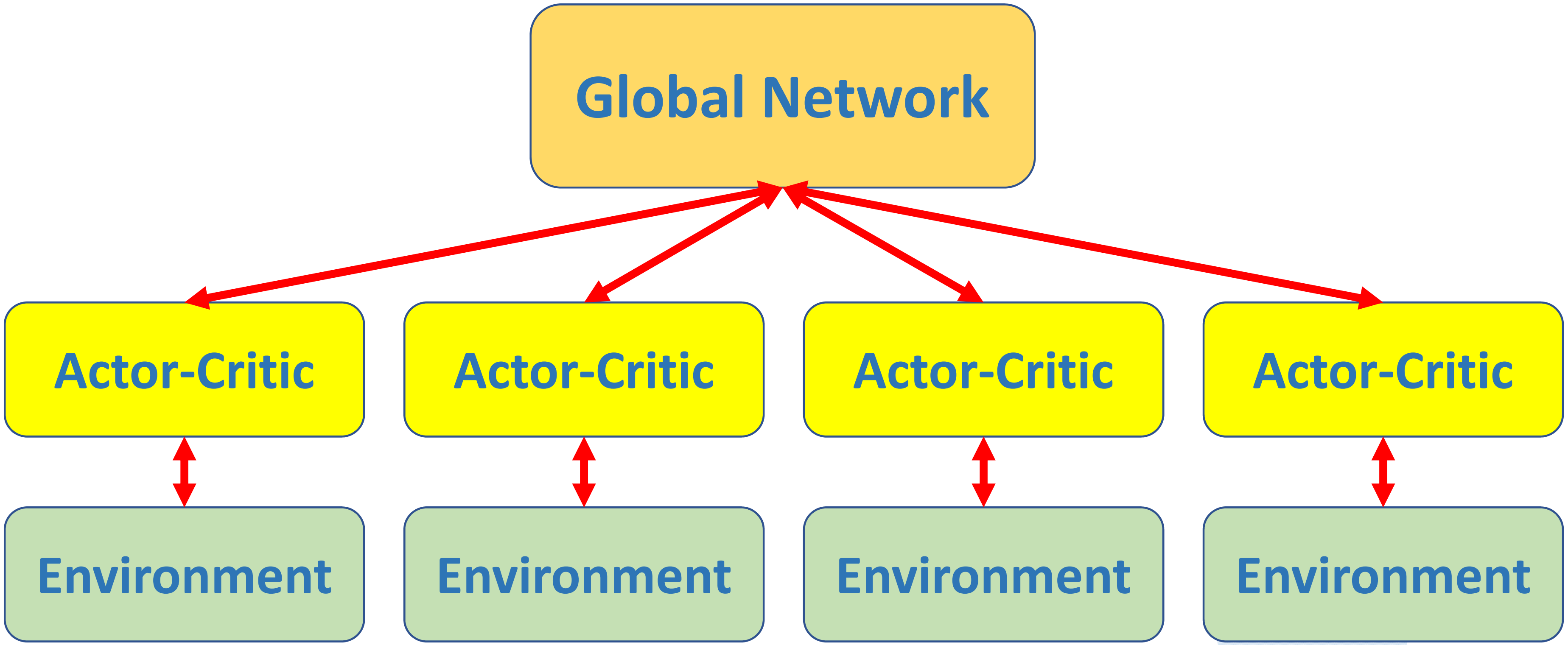


Deep Deterministic Policy Gradient

- Actor : Policy Gradient
- Critic : DGN

- DDPG can only be used for environments with continuous action spaces.

Asynchronous Advantage Actor-Critic (A3C)

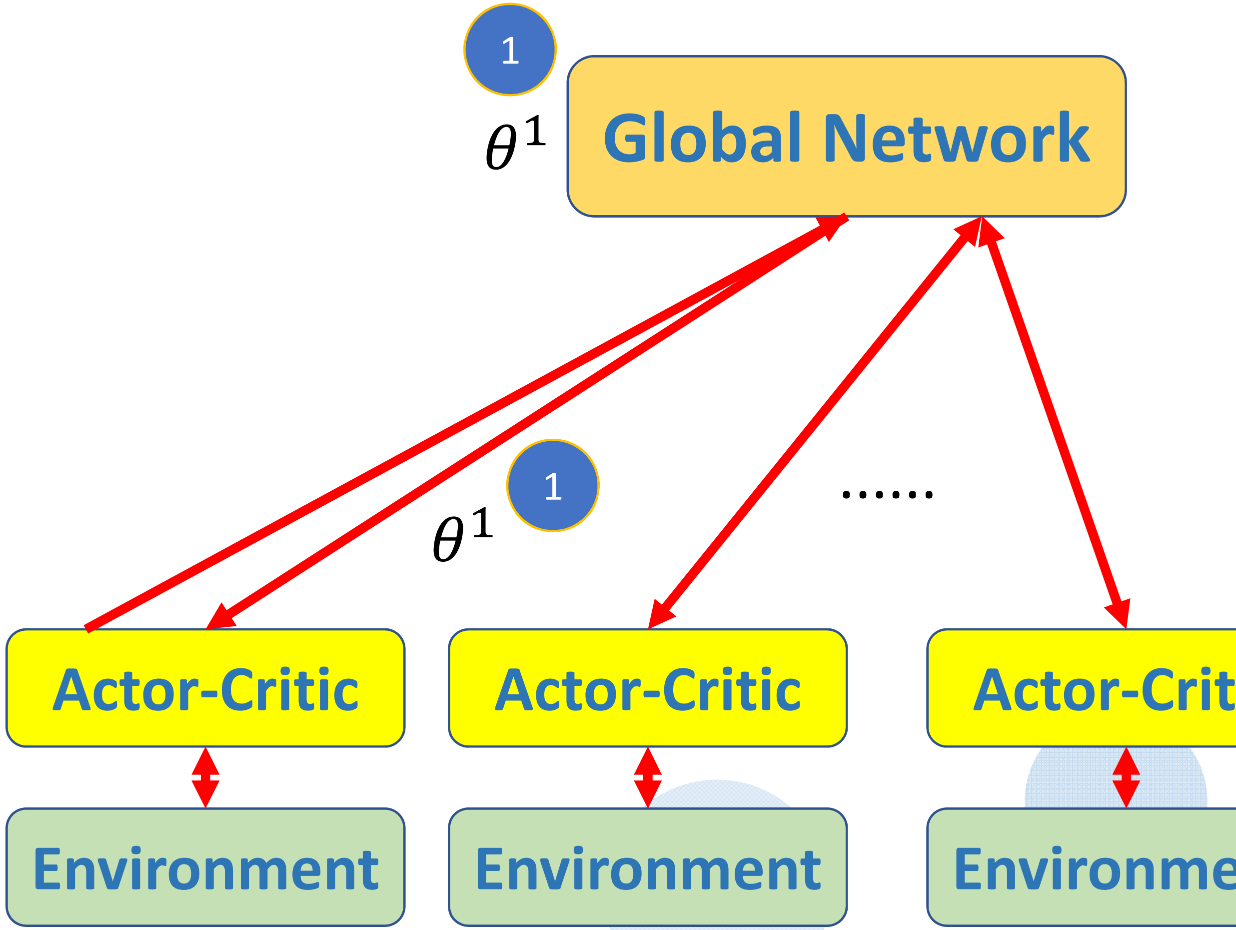


Asynchronous Advantage Actor-Critic (A3C)

- Train with a lot of Actor-Critic and use a central control for all Actor-Critic.
- Each Actor-Critic can upload their learned experience to the Global Network, and can also update their parameters using Global Network.

Asynchronous Advantage Actor-Critic (A3C)

- ◈1. Copy global parameters
- ◈2. Sampling some data
- ◈3. Compute gradients
- ◈4. Update global models

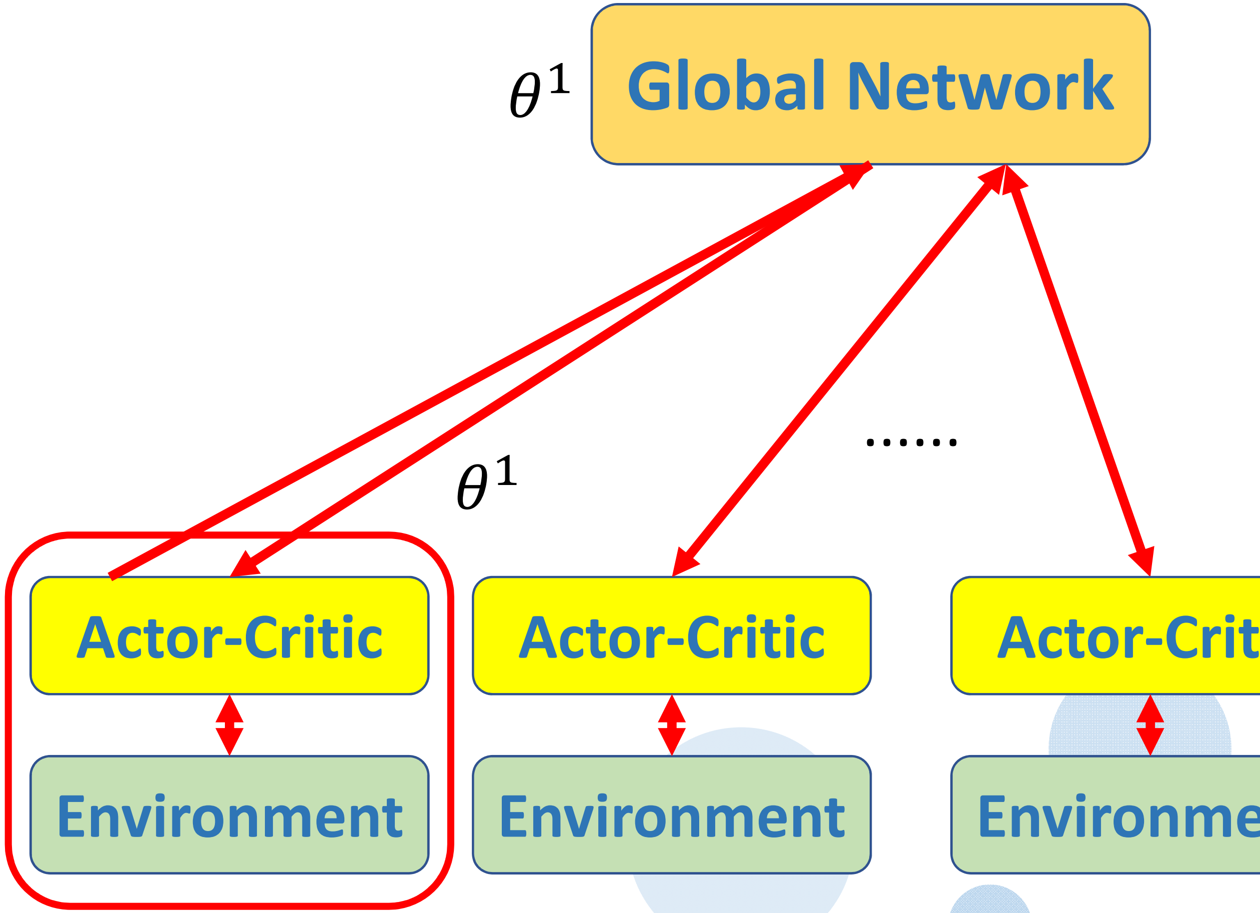


Asynchronous Advantage Actor-Critic (A3C)

- 1. Copy global parameters
- 2. Sampling some data
- 3. Compute gradients
- 4. Update global models

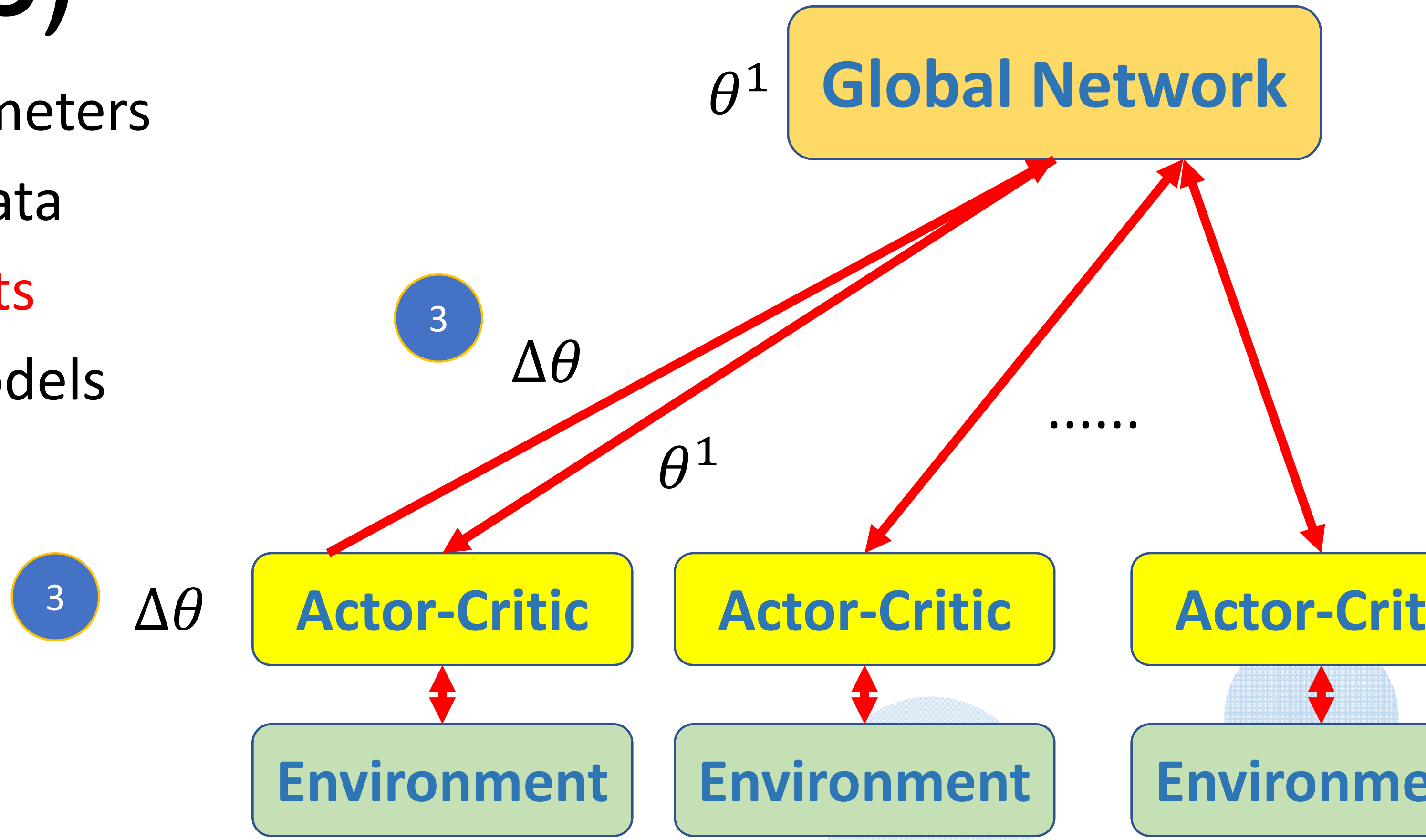
2

Interact with the environment to obtain information



Asynchronous Advantage Actor-Critic (A3C)

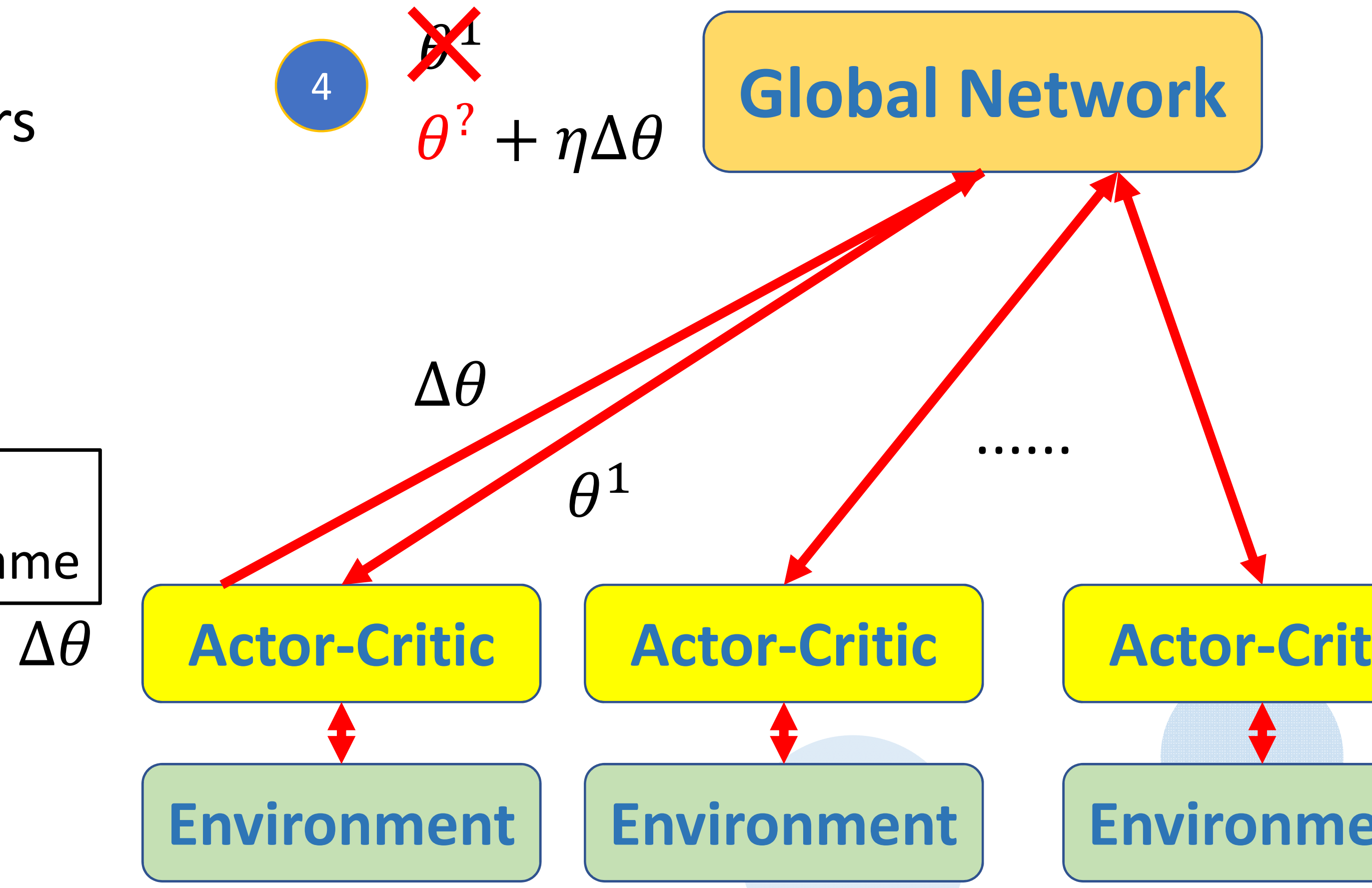
- ◈1. Copy global parameters
- ◈2. Sampling some data
- ◈3. Compute gradients
- ◈4. Update global models



Asynchronous Advantage Actor-Critic (A3C)

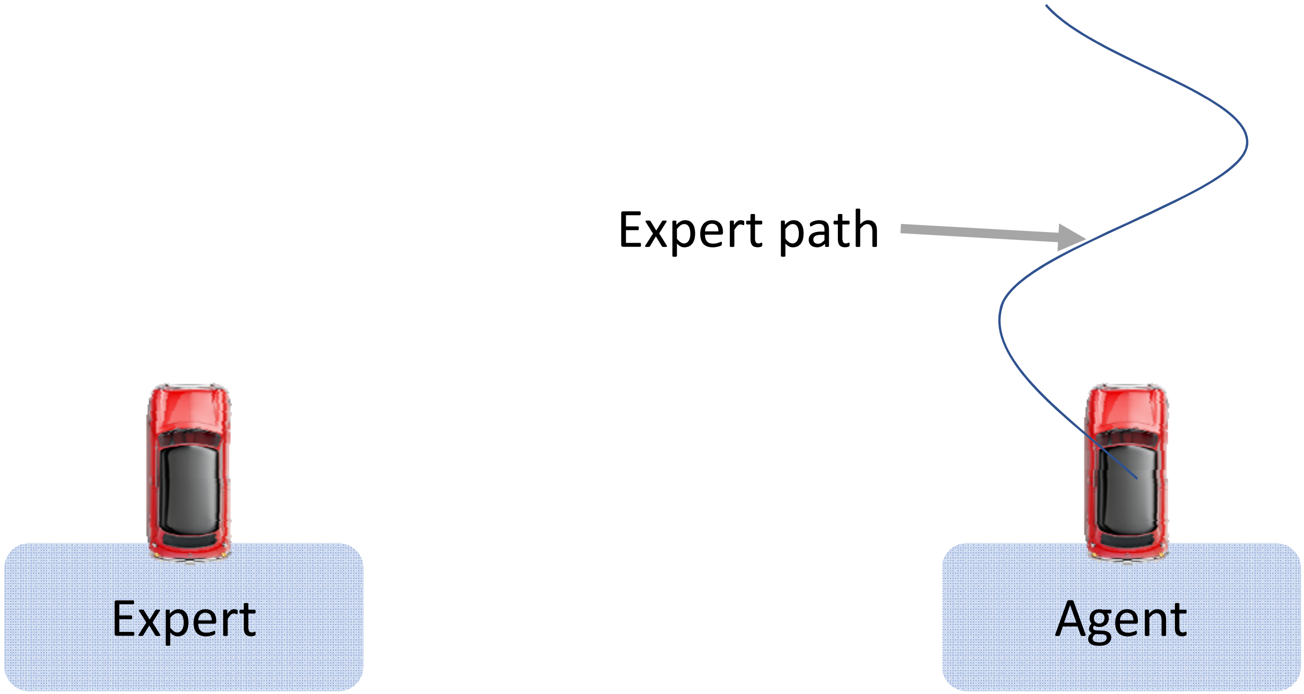
- 1. Copy global parameters
- 2. Sampling some data
- 3. Compute gradients
- 4. Update global models

Others may update the Global network, so θ may not be the same



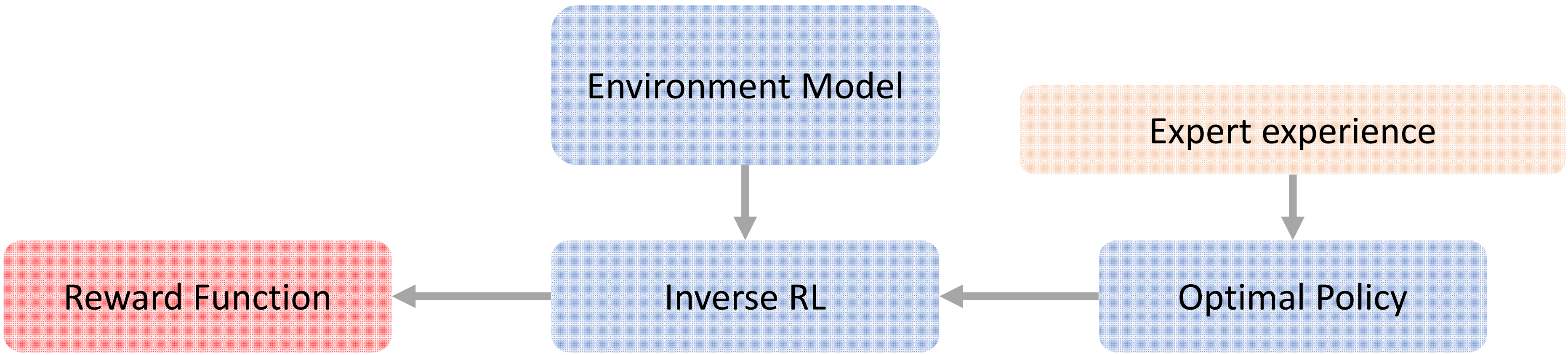
Imitation Learning

- We can learn by imitating the actions of experts.

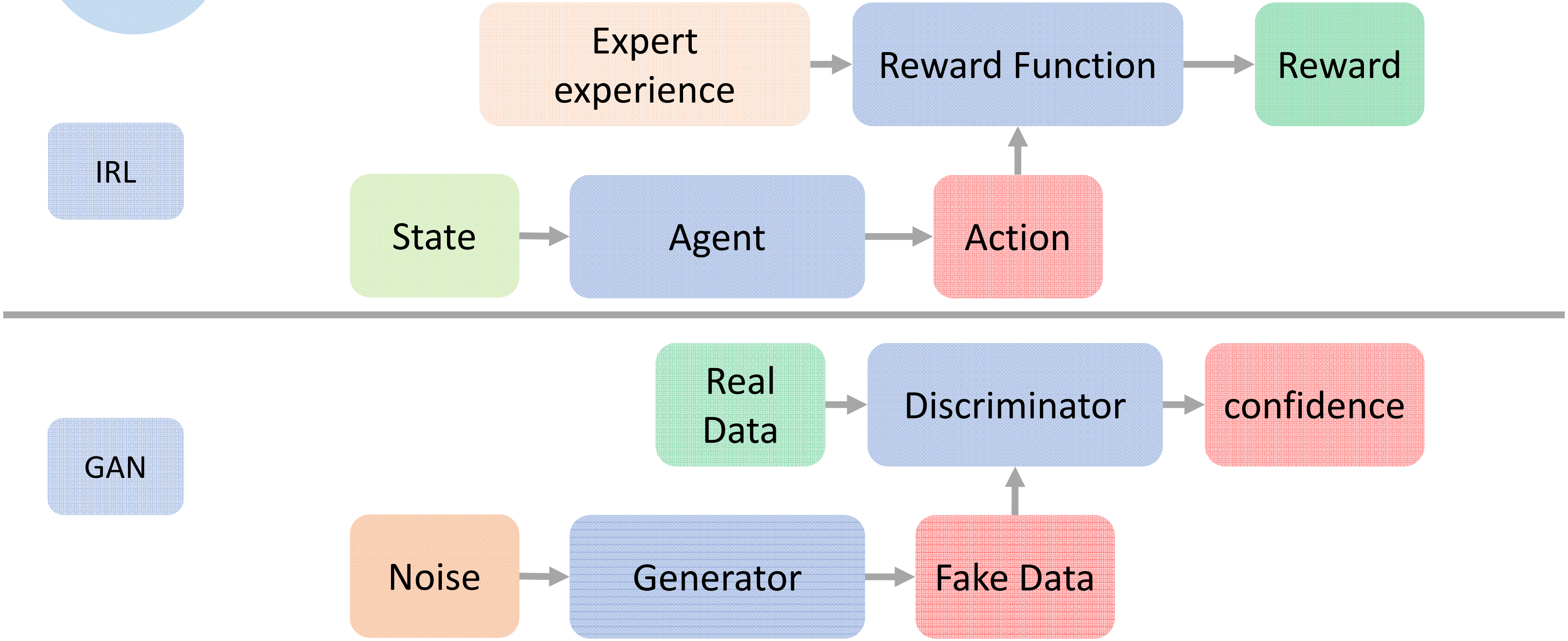


Inverse RL

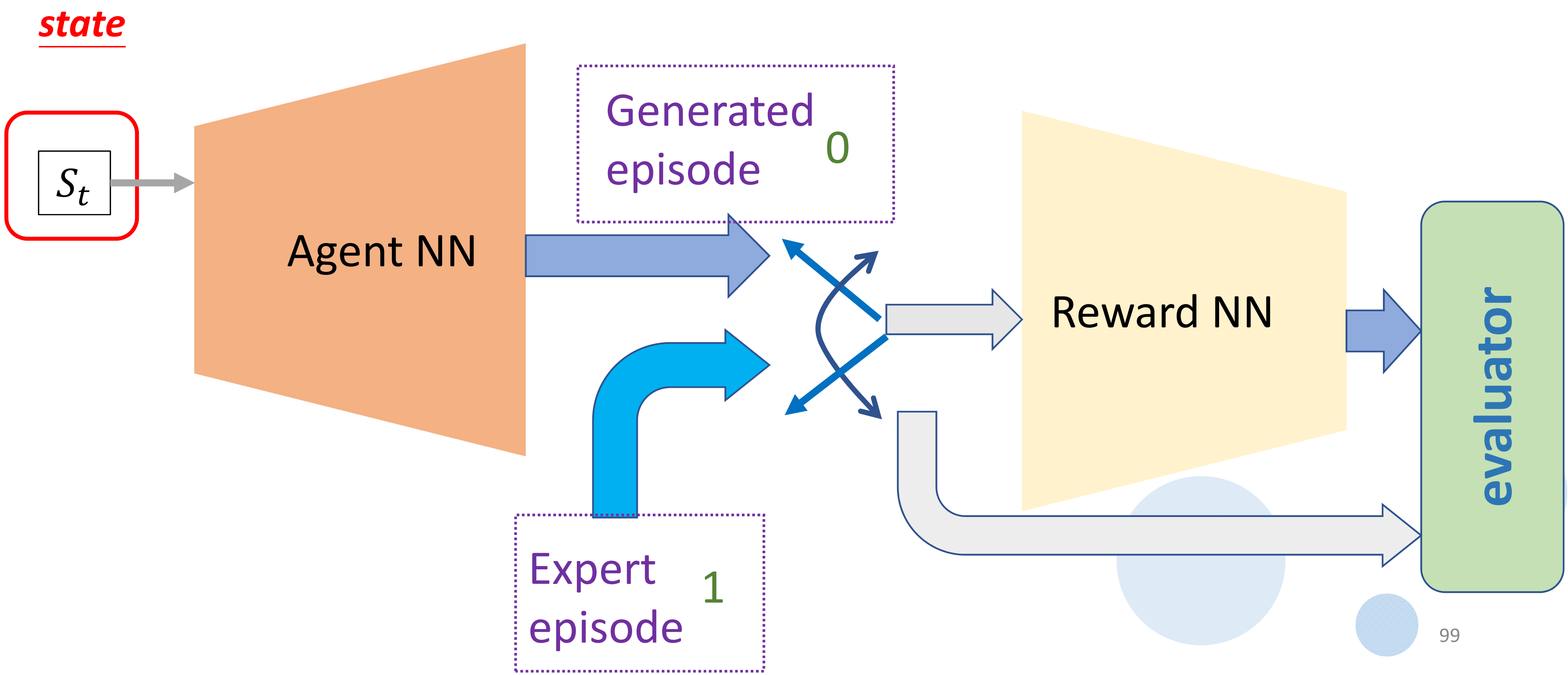
- We have some expert episodes available.
- Try to find a reward function, which can give expert episode large reward, while our episode generated by our agent low reward.



Inverse RL & GAN



Inverse RL



state
s_t
Agent NN
Generated episode
0
Expert episode
1
Reward NN
evaluator

Guided Cost Learning

Source http://rll.berkeley.edu/gcl/

Thanks for your listening!