Reinforcement Learning in RTB
A Tutorial

chaoqiyang(杨超琪)、junweilu(卢君苇)

2019 / 5 / 15

Tencent Social Ads, Beijing

Outline

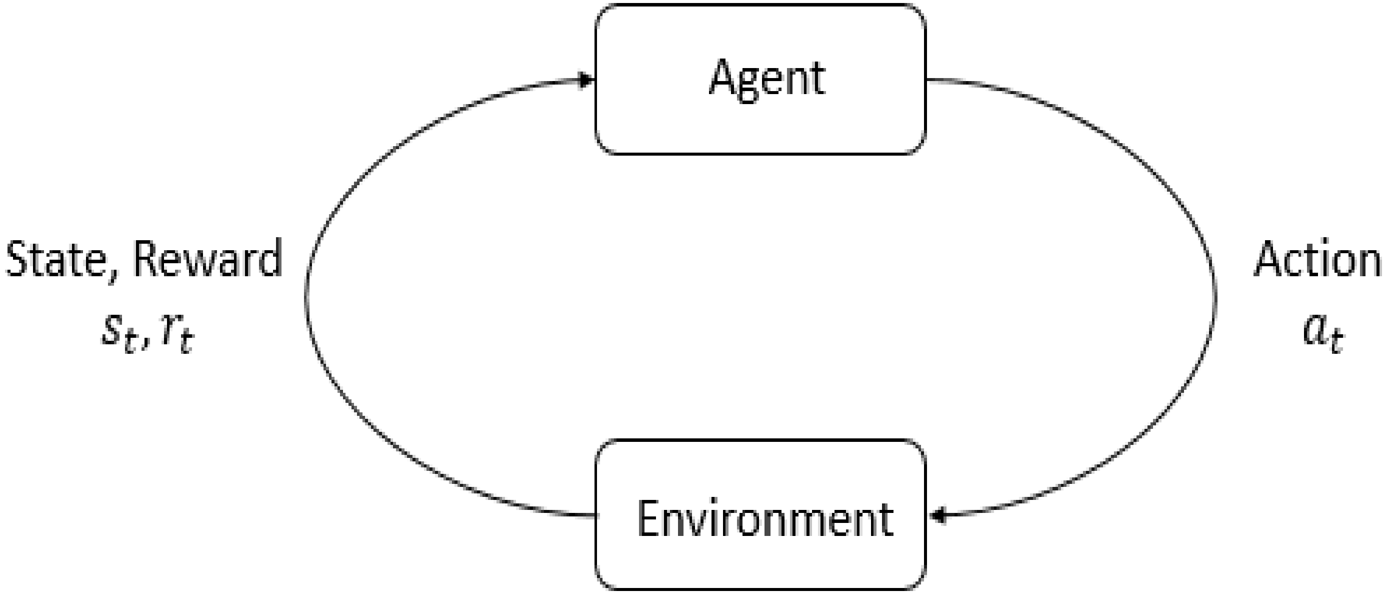
- **Introduction**
  - Basic concepts
  - An example
- **Markov decision process (MDP)**
- **RL categorization**
  - Policy gradient
  - Q-Learning
  - A3C
- **Application to ads**
  - Implementation of multi-process
- **Future direction**

Reinforcement Learning
An Introduction

Introduction

- **Supervised Learning**
  - Learning a mapping function to regression & classification
  - Learning from training data
- **Unsupervised Learning**
  - Learning approaches to dimension reduction, clustering, density estimation
- **Reinforcement Learning**
  - Learning to do sequential decision making
  - Learning from delayed reward

Basic concepts



① state s_0, agent gives a_0

② Get r_0 from environment

③ Transition to state s_1

④ state s_1, agent gives a_1

⑤ Get r_1 from environment

⑥ Transition to state s_2

⑦ state s_2, agent gives a_2

⑧ ……

- Trajectory: $(s_0, a_0, r_0, s_1, a_1, r_1, s_2, a_2, \ldots)$
- Transition: (s_0, a_0, r_0, s_1), (s_1, a_1, r_1, s_1) …
- From initial state s_0, we can sample many trajectories.

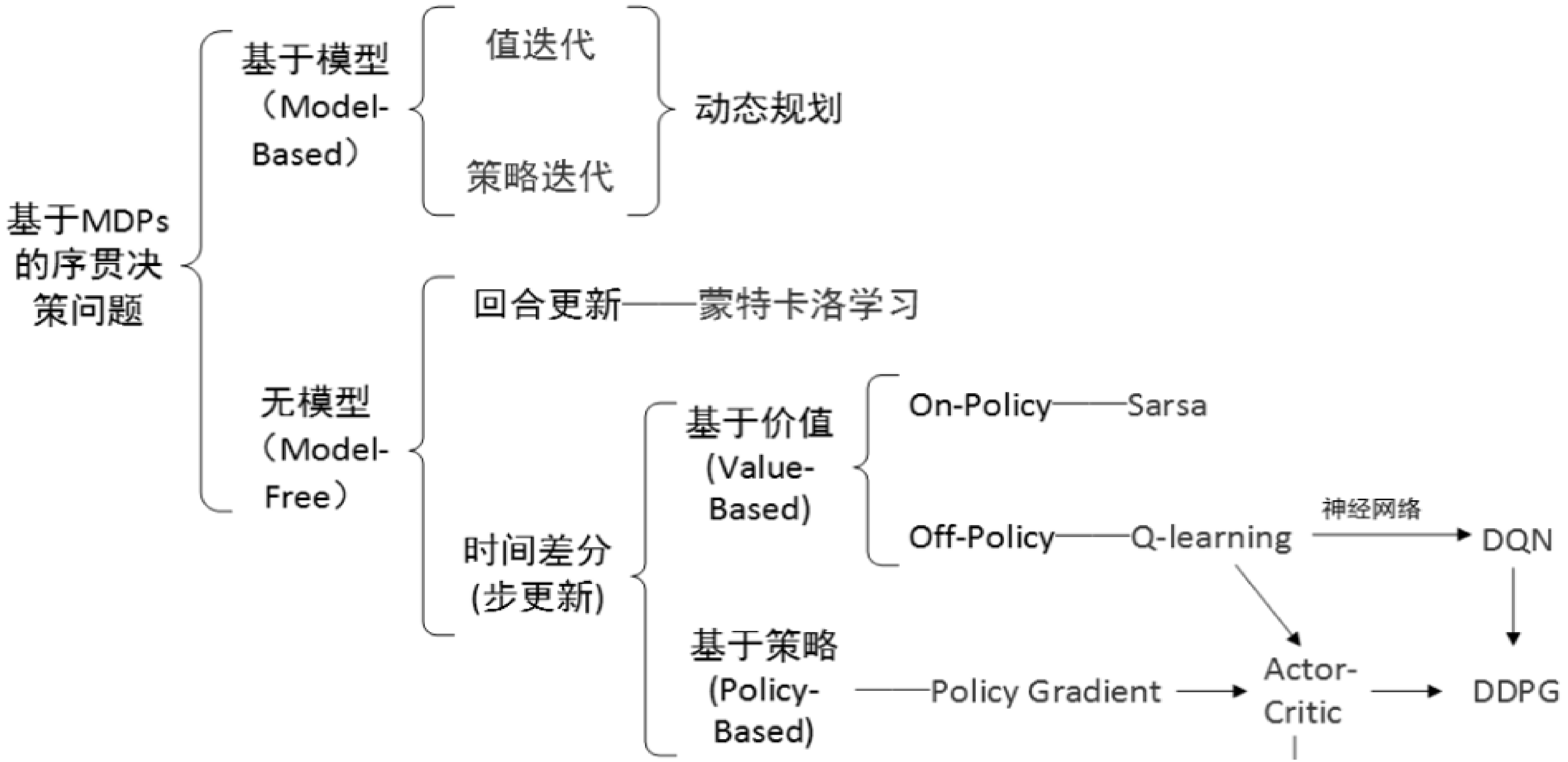
Basic concepts

- **State: feature vector of the current situation**
  - In Atari, A picture
  - In RTB, 〈bias, conversion, pcvr, pctr, …〉
- **Action: given by agent based on current state**
  - In Atari, left / right / up / down
  - In RTB, adjust factor
- **Reward: assessment of action, given by environment**
  - In Atari, reward=1 if succeed, reward=0 if fail
  - In RTB, reward = $\sqrt{\frac{gmv}{cost} \times \frac{gmv}{base\ gmv}}$

An example

- **State: a screenshot**
  - Picture
- **Action: left / right**
- **Reward: 1 / 0**
  - Break the brick, 1
  - Miss the ball, 0

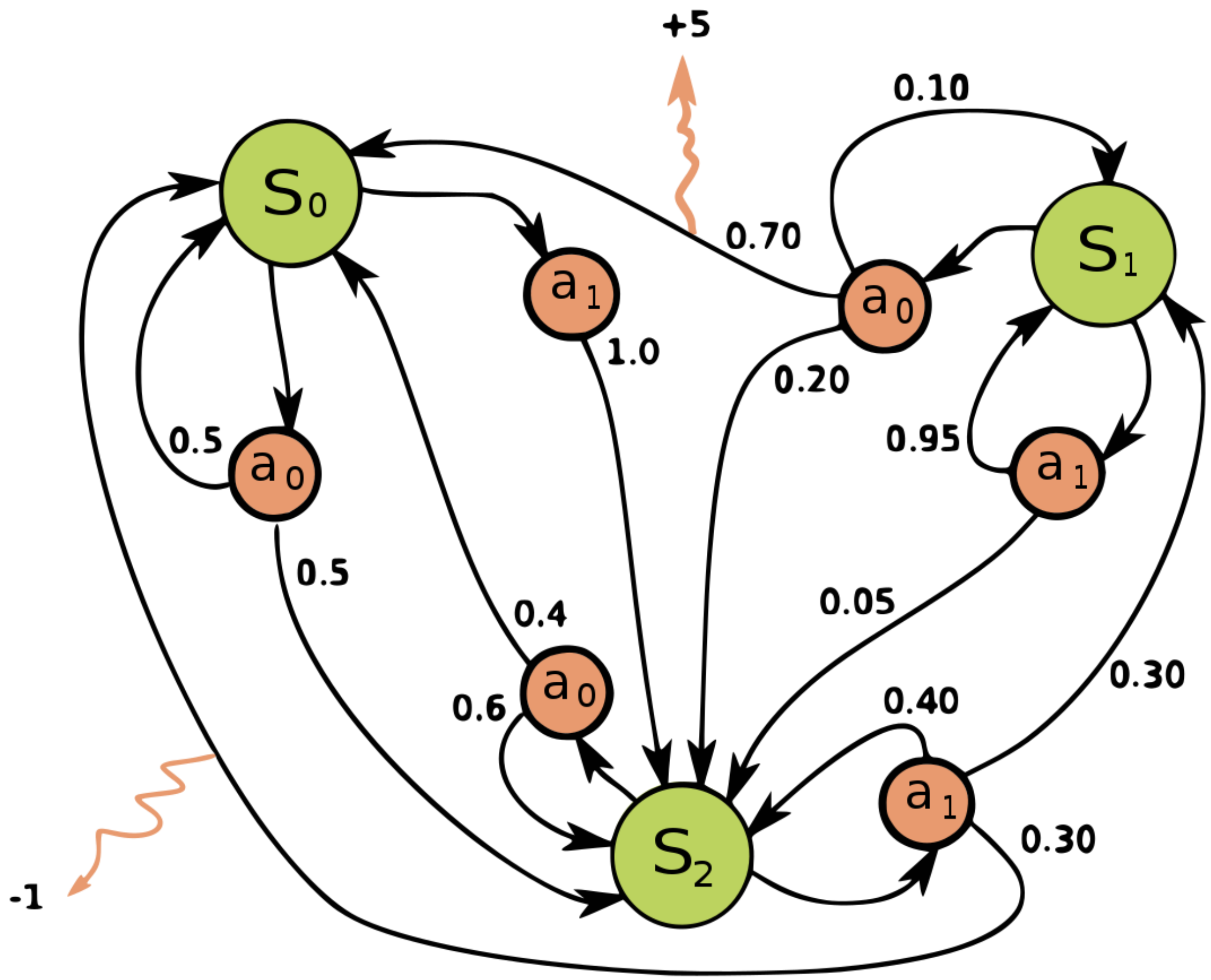
RL Tree



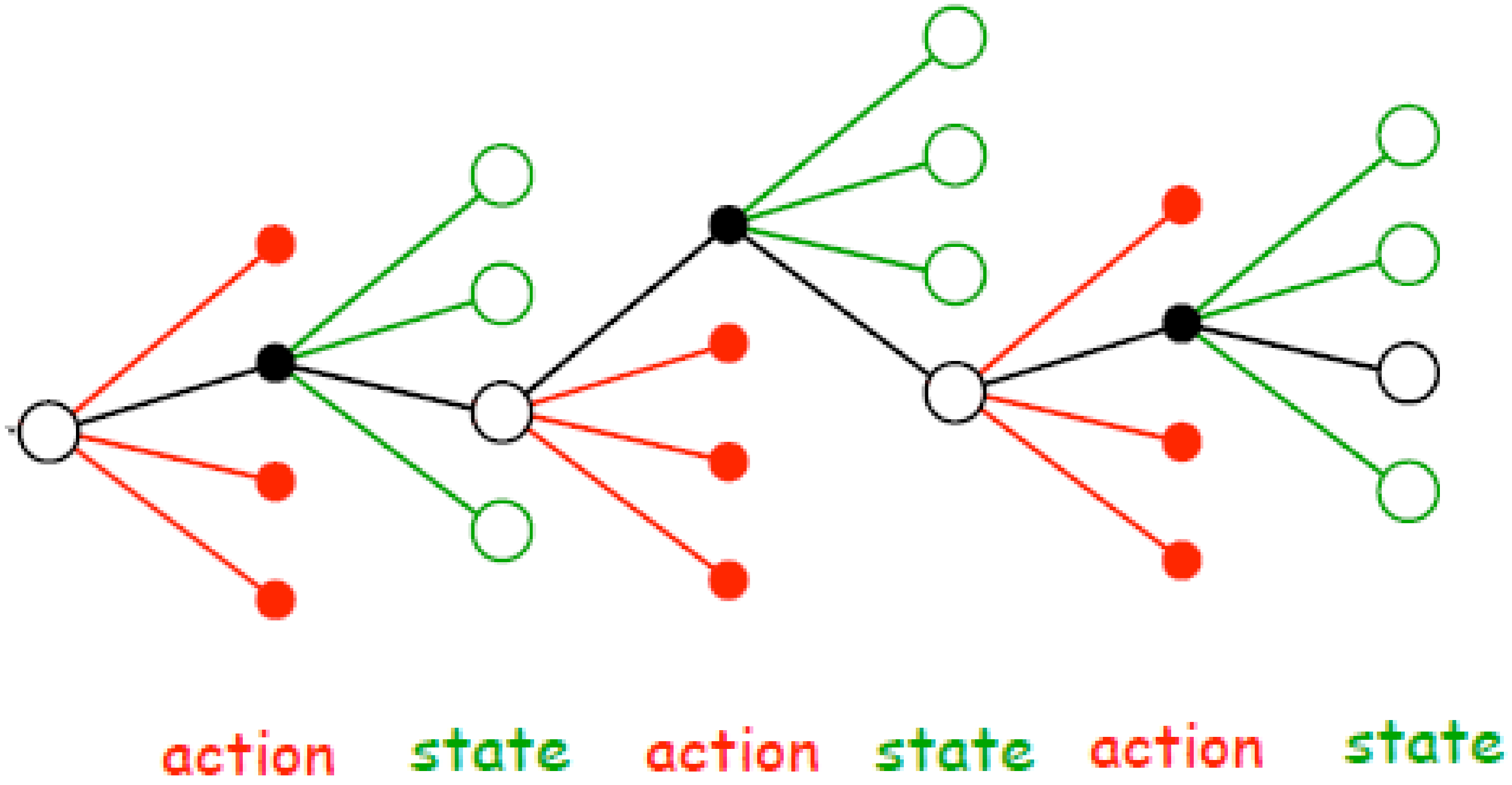
基于MDPs的序贯决策问题
基于模型（Model-Based）
值迭代
策略迭代
动态规划
无模型（Model-Free）
回合更新——蒙特卡洛学习
时间差分(步更新)
基于价值(Value-Based)
On-Policy——Sarsa
Off-Policy——Q-learning
神经网络
DQN
基于策略(Policy-Based)
Policy Gradient
Actor-Critic
DDPG

Markov Decision Process (MDP)
An Introduction

Markov decision process (MDP)



Markov decision process (MDP)

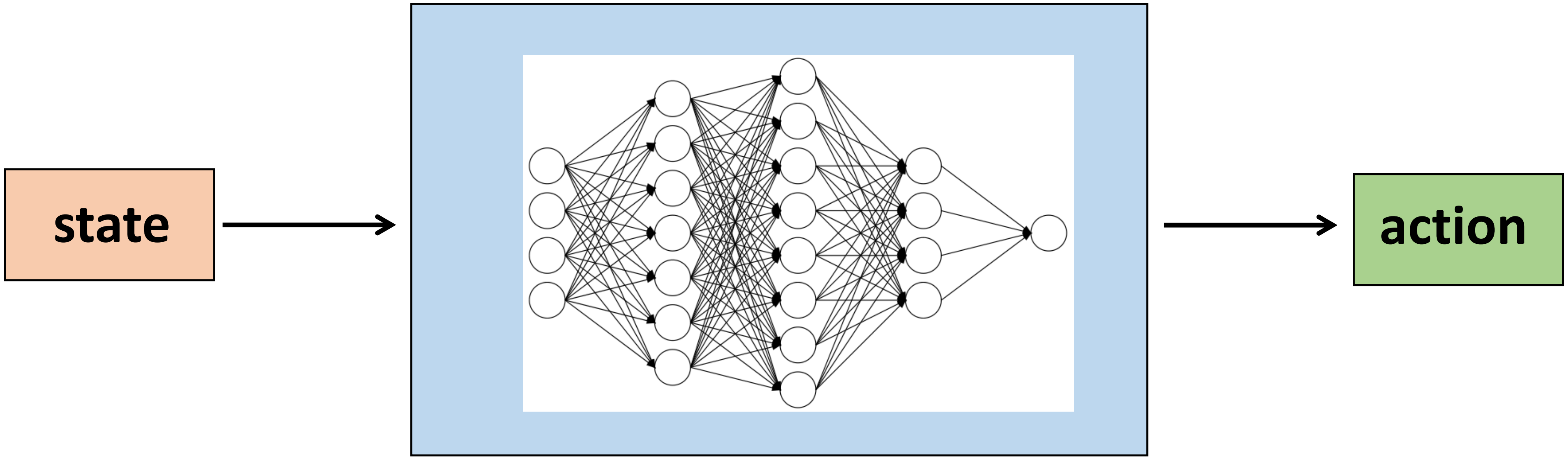


Markov assumption / property

- $Pr(s_{t+1} \mid s_t, a_t, s_{t-1}, a_{t-1}, \dots, s_0, a_0) = Pr(s_{t+1} \mid s_t, a_t)$
- $Pr(r_{t+1} \mid s_t, a_t, s_{t-1}, a_{t-1}, \dots, s_0, a_0) = Pr(r_{\mathrm{t+1}} \mid s_t, a_t)$
- **Transition probability:** $P_{ss'}^{a} = \Pr(s_{t+1} = s' \mid s_t = s, a_t = a)$
- **Reward function:** $R_{ss'}^{a} = \mathrm{E}\{\mathrm{r}_{t+1} \mid s_{t+1} = s', s_t = s, a_t = a\}$

Policy π

- **Neural Network**
  - $\pi(s) = a$
- **Action**
  - Discrete action
  - Continuous action: $\pi_1(s) = \mu,\ \pi_2(s) = \sigma,\ \mathbf{a} \sim N(\mu, \sigma)$

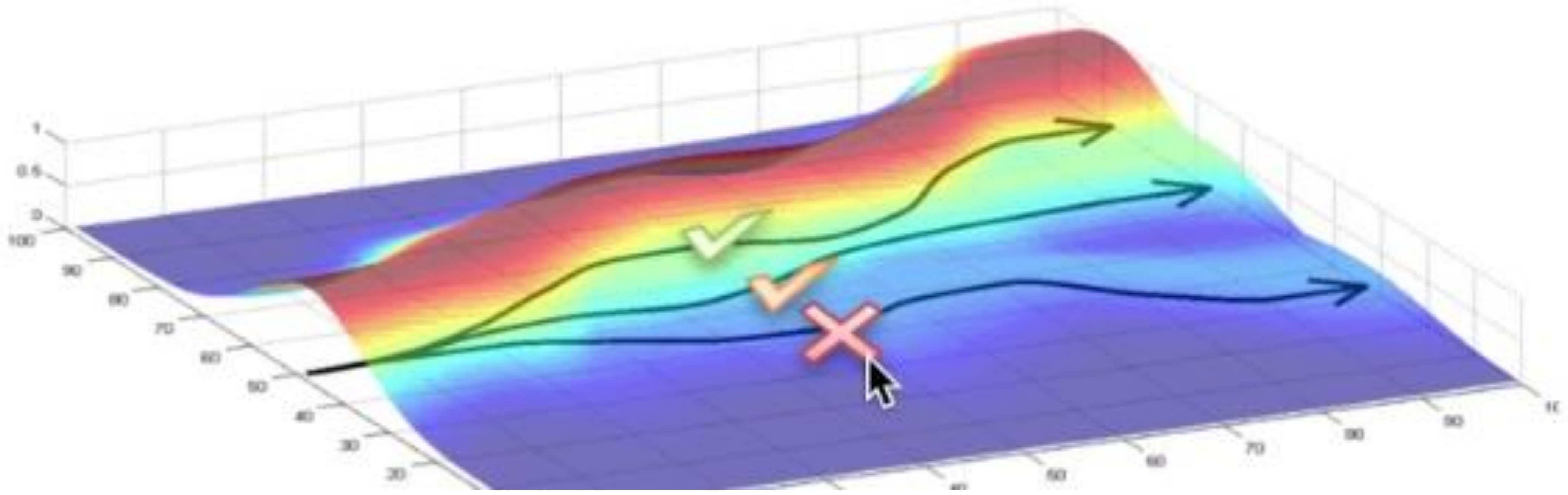


State-value function $V^{\pi}(s)$

- $V^{\pi}(s) = E_{\pi}\{r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots \mid s_0 = s\}$
- **Start from state s, play to the end of the game.**
- **Trajectory 1:** $(s, a_0^1, r_0^1, s_1^1, a_1^1, r_1^1, \dots)$**, decayed reward =** $r_0^1 + \gamma r_1^1 + \cdots$
- **Trajectory 2:** $(s, a_0^2, r_0^2, s_1^2, a_1^2, r_1^2, \dots)$**, decayed reward =** $r_0^2 + \gamma r_1^2 + \cdots$
- **Trajectory 3:** $(s, a_0^3, r_0^3, s_1^3, a_1^3, r_1^3, \dots)$**, decayed reward =** $r_0^3 + \gamma r_1^3 + \cdots$
- **…**

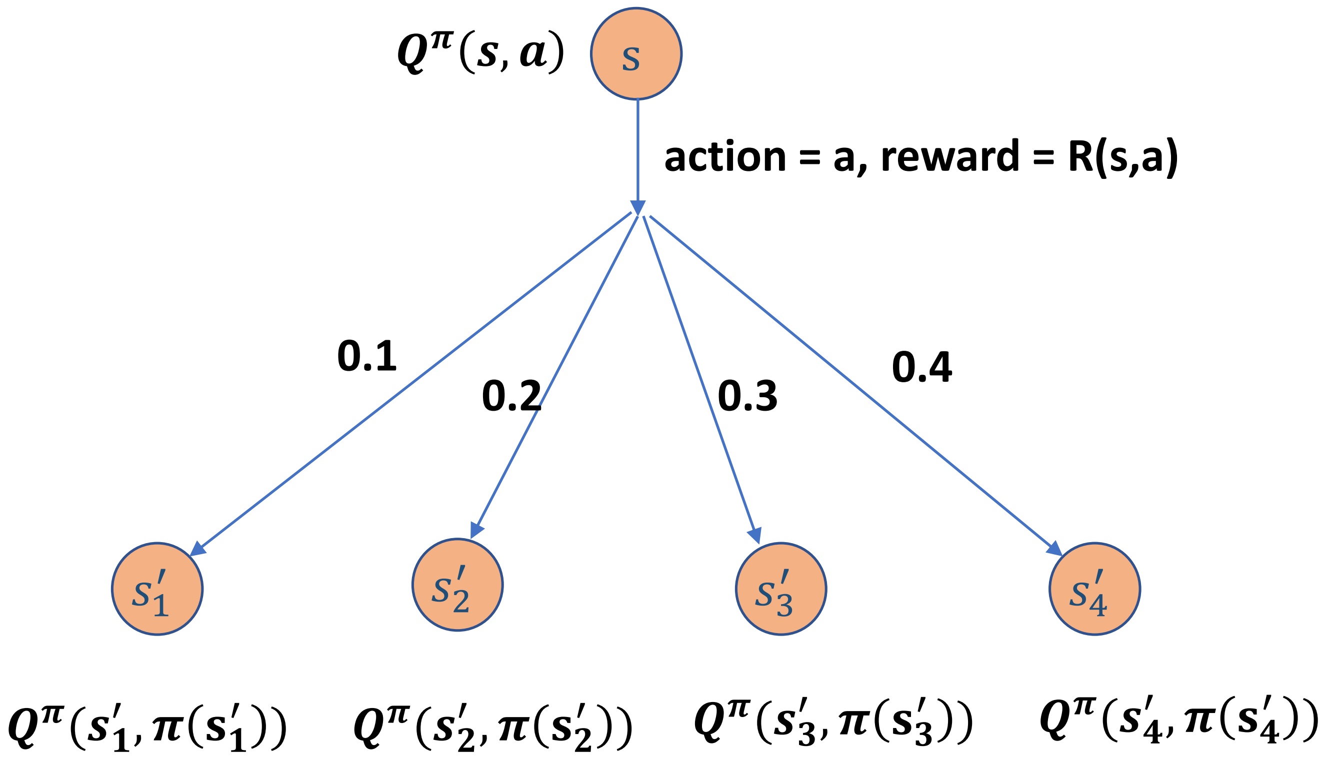
State-value function $V^{\pi}(s)$

- **Trajectory**



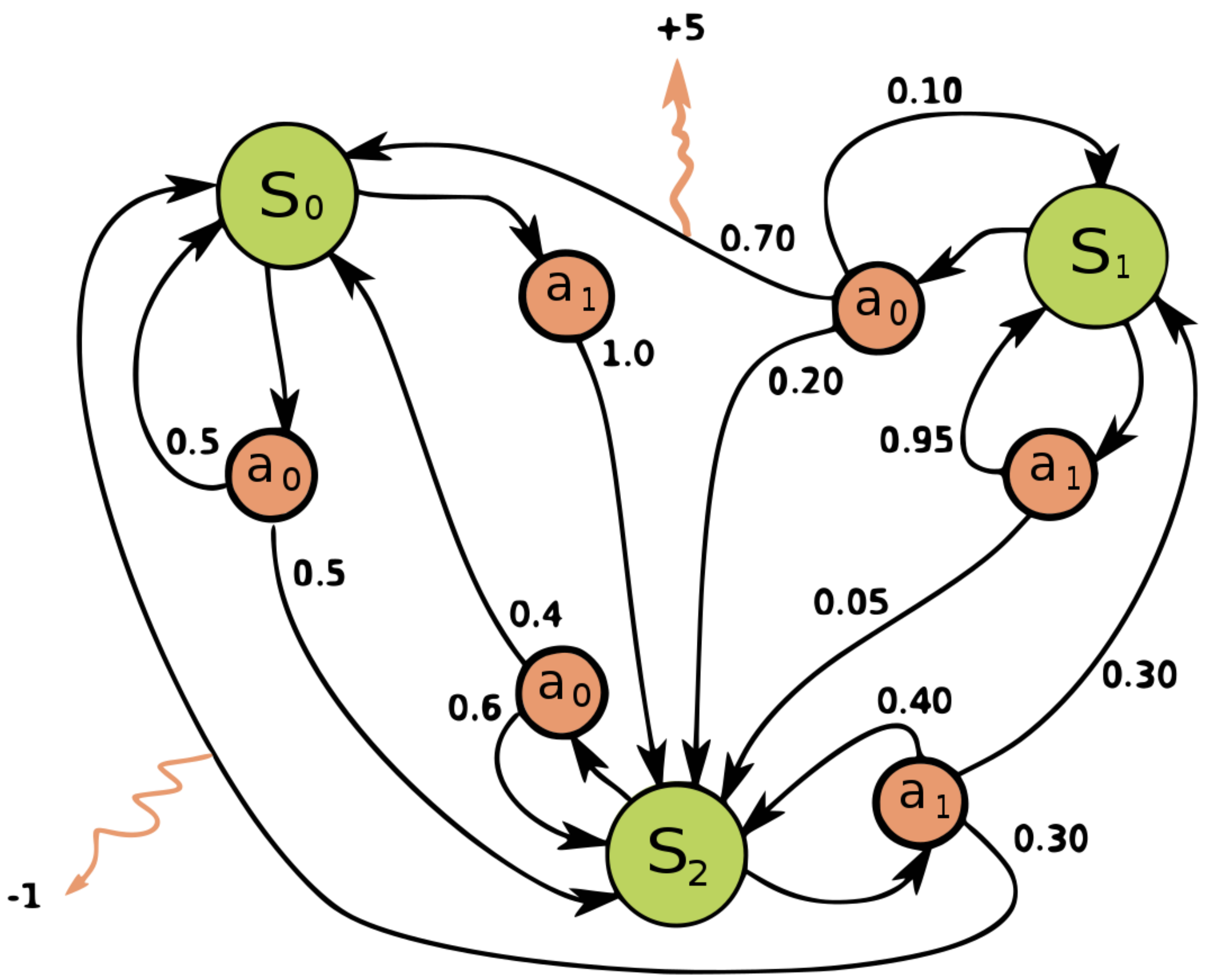
State-action function $Q^{\pi}(s,a)$

- $Q^{\pi}(s,a) = R(s,a) + \sum_{s' \in S} P(s' | s, \pi(s))\, Q^{\pi}(s', \pi(s'))$



- $V^{\pi}(s) = R(s, \pi(s)) + \sum_{s' \in S} P(s' | s, \pi(s))\, V^{\pi}(s')$

Markov decision process (MDP)



Relationship $V^{\pi}, \pi, Q^{\pi}(s, a)$

- $\pi^* = argmax_{\pi}\, V^{\pi}(s)$
- $V^*(s) = max_{\pi}\, V^{\pi}(s)$

- $\pi^* = argmax_{a \in A}\, [R(s, a) + \sum_{s' \in S} P(s' | s, a)\, V^*(s')]$
- $V^*(s) = max_{a \in A}\, [R(s, a) + \sum_{s' \in S} P(s' | s, a)\, V^*(s')]$

- $\pi^*(s) = argmax_{a \in A}\, Q^*(s, a)$
- $V^*(s) = max_{a \in A}\, Q^*(s, a)$

Policy iteration

- **Initialize** π
- **Compute** $V^{\pi}(s)$
  - $s_0, \pi(s_0), r_0, s_1, \pi(s_1), r_1, \ldots$
  - $\mathrm{V}^{\pi}(\mathrm{s}) = \mathrm{r}_0 + \gamma \mathrm{r}_1 + \gamma^2 \mathrm{r}_2 + \cdots$
- **Update** π
  - $\pi^{next}(s) = argmax_{a\in A}\left[R(s,a) + \sum_{s'\in S} P(s' | s,a)\, V^{\pi}(s')\right]$
- **Compute** $V^{\pi^{next}}(s)$
- **Update** π^{next}
- **…**

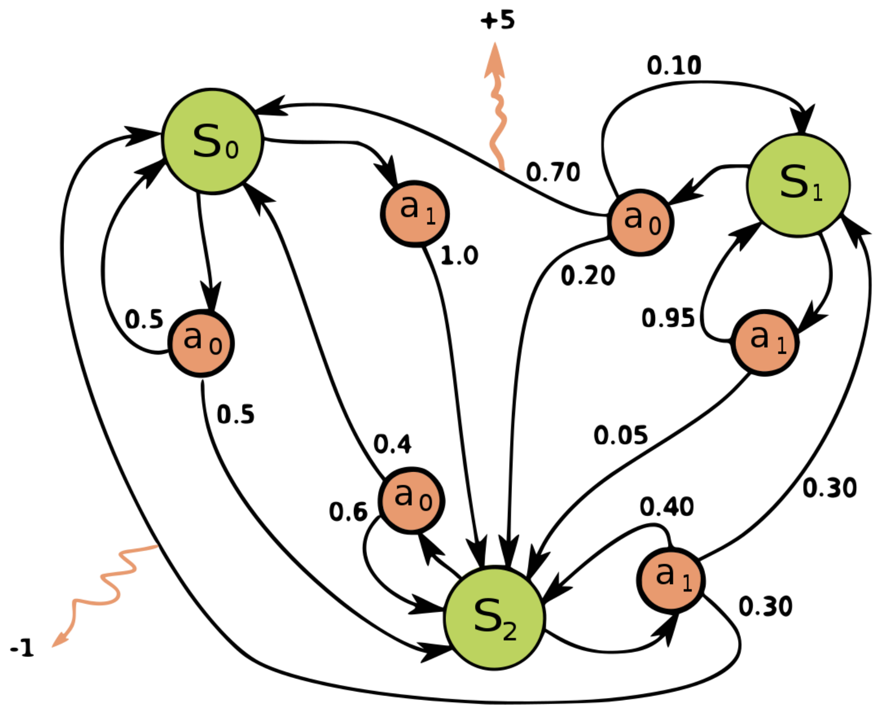
Value iteration

- **Initialize** $V(s)$
- **update** $V(s)$
  - $V(s) = max_{a \in A}\left[R(s, a) + \sum_{s' \in S} P(s' | s, a) V(s')\right]$
- **Compute** π
  - $\pi(s) = argmax_{a \in A}\left[R(s, a) + \sum_{s' \in S} P(s' | s, a) V(s')\right]$

- **Or we could iterate** $Q(s, a)$

Bellman equation

- $V(s) = \left[\mathrm{R}(s, \pi(s)) + \sum_{s' \in S} P(s' \mid s, \pi(s))\, V(s')\right]$
- $Q(s, \pi(a)) = \left[\mathrm{R}(s, \pi(s)) + \sum_{s' \in S} P(s' \mid s, \pi(s))\, Q(s', \pi(a))\right]$



Reinforcement Learning
--- Q-Learning

Q-Table

- **Refer table to choose action**
- **It can be updated**

Q(s, a)	a_1	a_2	a_3	a_4
s_1	-1	-3	**3**	2
s_2	$*$	$*$	$*$	$*$
s_3	$*$	$*$	$*$	$*$
s_4	$*$	$*$	$*$	$*$
s_5	$*$	$*$	$*$	$*$
s_6	$*$	$*$	$*$	$*$

An example

- State is the distance away from exit
- Action is forward or backward
- Initial Q-Table

Q(s, a)	5米	4米	3米	2米	1米
forward	1分	1分	1分	1分	10分
backward	1分	1分	1分	1分	-5分

- First round later

Q(s, a)	5米	4米	3米	2米	1米
forward	1分	1分	1分	5分	10分
backward	1分	1分	1分	-1分	-5分

An example

- **Second round later**

Q(s, a)	5米	4米	3米	2米	1米
forward	1分	1分	6分	8分	10分
backward	1分	1分	0分	-3分	-5分

- **Many rounds later**

Q(s, a)	5米	4米	3米	2米	1米
forward	3分	4分	6分	9分	10分
backward	0分	0分	-1分	-3分	-5分

Basic Q-Learning

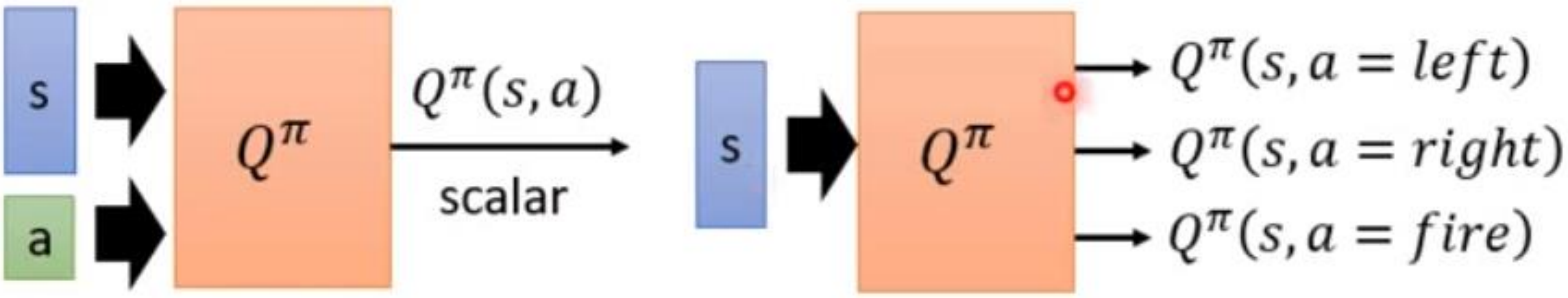
- **Update every entry in Q-Table**
- **In state s, take action a, get reward r, transition to state s'**
- **Gradient**
  - $r + \gamma max_{a'} Q(s', a') - Q(s, a)$
  - **Like Floyd Algorithm**
- **Update $Q(s, a)$**
  - $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma max_{a'} Q(s', a') - Q(s, a)]$

Basic Q-Learning

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., ε-greedy)
        Take action a, observe r, s'
        $Q(s,a) \leftarrow Q(s,a) + \alpha\big[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\big]$
        $s \leftarrow s'$;
    until s is terminal

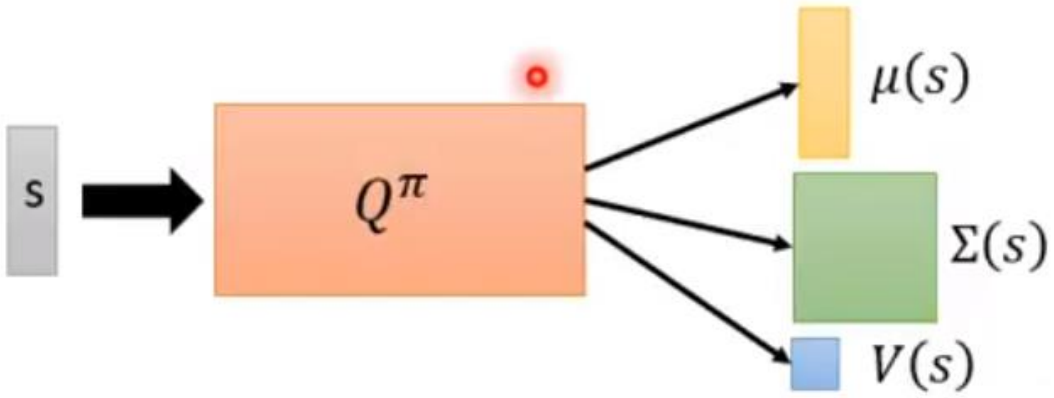
Deep Q-Network

- **If the state set is infinite?**
  - No Q-Table
- **Deep Neural Network**



Other variants

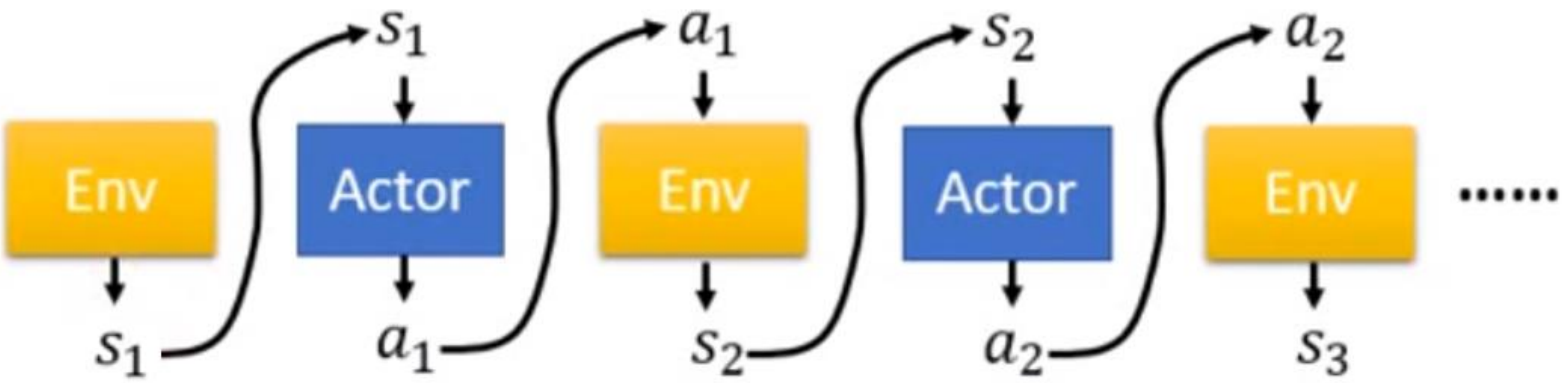
- **Double DQN**
  - One network to choose action
  - Another network to evaluate Q(s, a)
- **Continuous DQN**



- ……

Reinforcement Learning
--- Policy gradient

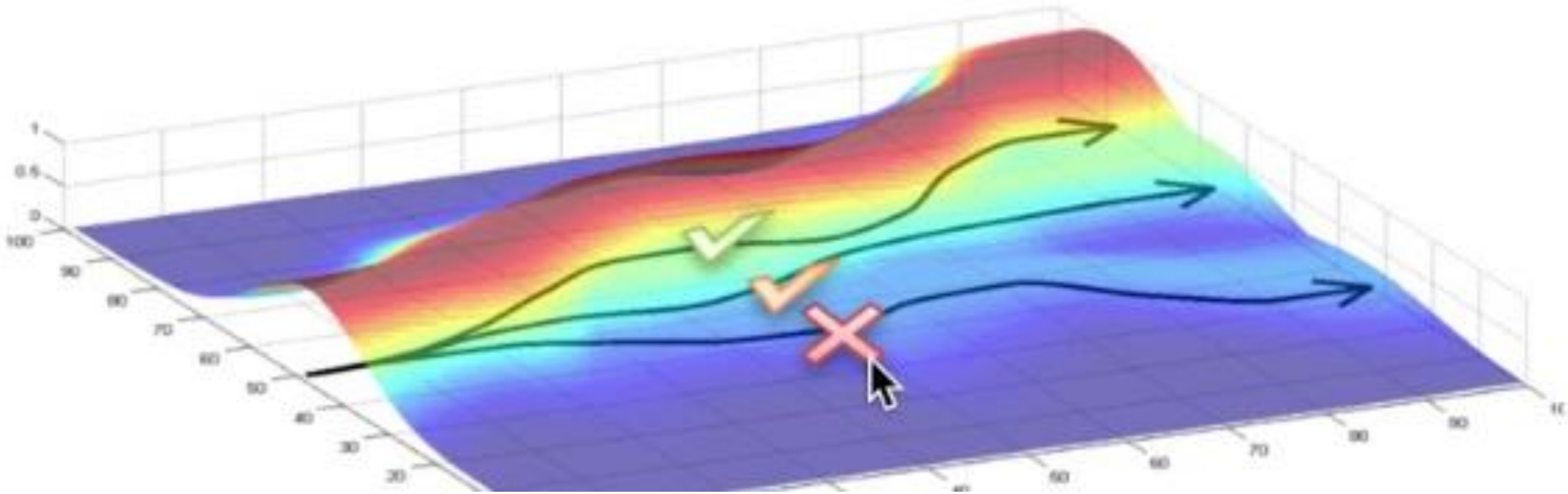
Policy gradient



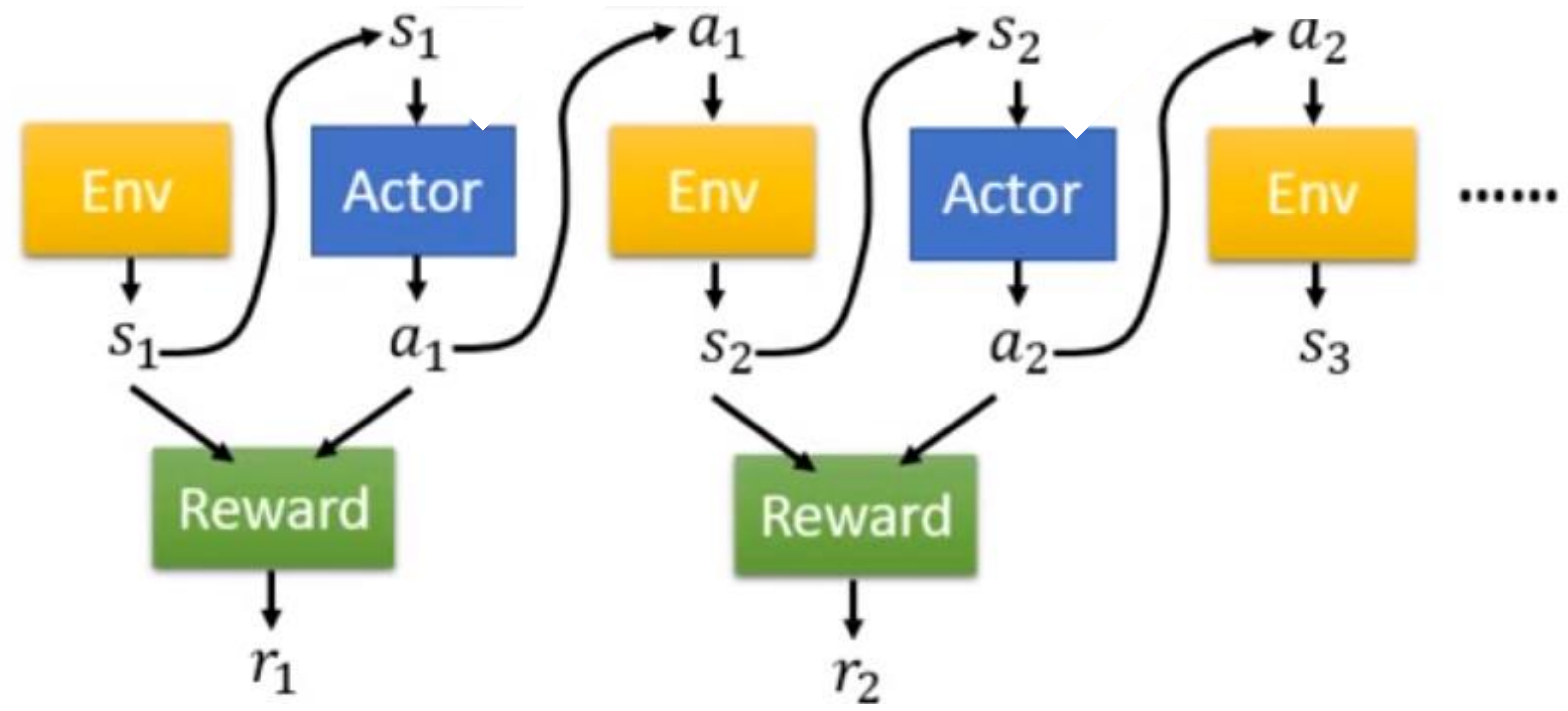
Trajectory $\tau = \{s_1, a_1, s_2, a_2, \cdots, s_T, a_T\}$

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

Trajectory



Policy gradient



- $\boldsymbol{Tajectory\ \tau}$
- $\boldsymbol{R(\tau) = \sum r}$
- $\boldsymbol{\widehat{R}_\theta = \mathrm{E}_\tau[R(\tau)] = \sum_\tau R(\tau) p_\theta(\tau)}$

Policy gradient

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

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

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

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

Proximal Policy Optimization (PPO)

- **On-policy**

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

- **Off-policy**

$$\nabla \bar{R}_\theta = E_{\underline{\tau \sim p_{\theta'}(\tau)}}\left[\frac{p_\theta(\tau)}{\underline{p_{\theta'}(\tau)}} R(\tau)\nabla log p_\theta(\tau)\right]$$

- **Importance sampling**
  - Assumption: If $p_\theta(\tau)$ *is similar to* $p_{\theta'}(\tau)$

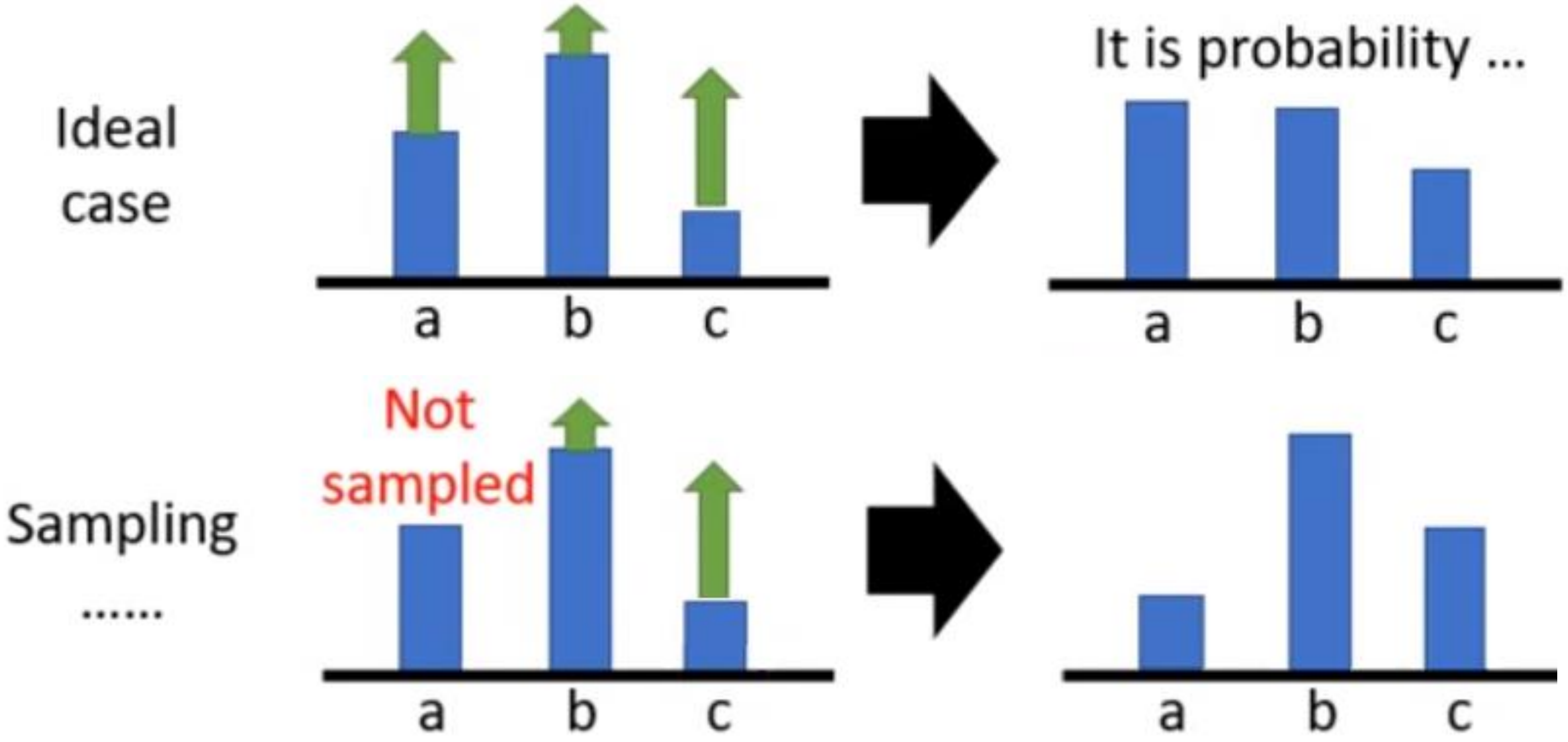
Other variants

- **Deep deterministic policy gradient (DDPG)**
- **Trust region policy optimization (TRPO)**
- **……**

Reinforcement Learning
--- Actor-Critic

Problem of policy gradient

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



Baseline reward

- **Basic policy gradient**

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

- **Add a baseline reward**

$$\nabla \bar{R}_\theta \approx \frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T_n} \boxed{(R(\tau^n) - b)}\nabla log p_\theta(a_t^n|s_t^n) \qquad b \approx E[R(\tau)]$$

advantage

Advantage actor-critic

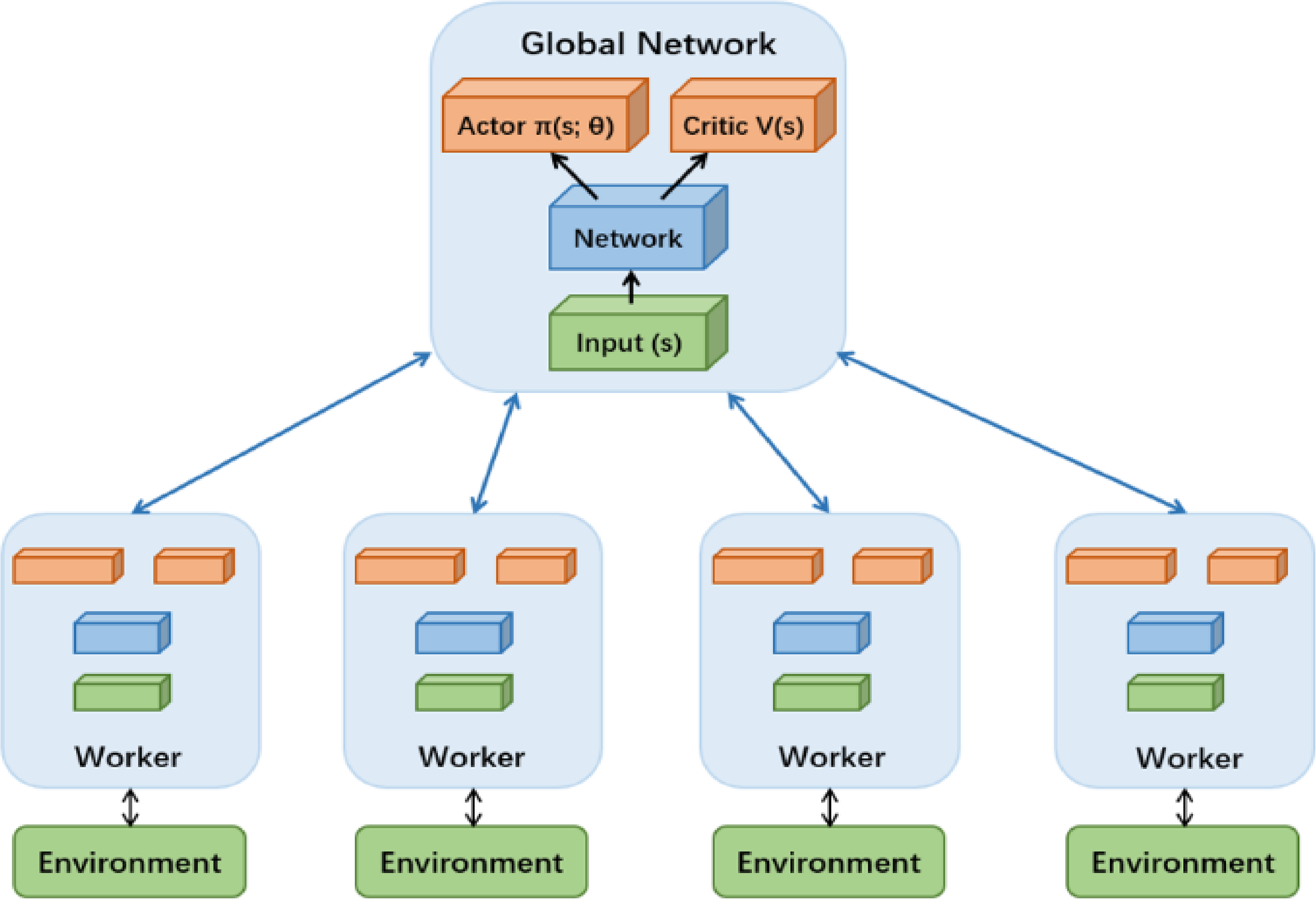
- **TD-error**

$$\nabla \bar{R}_\theta \approx \frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T_n}\left(r_t^n + V^\pi(s_{t+1}^n) - V^\pi(s_t^n)\right)\nabla log p_\theta(a_t^n|s_t^n)$$

- **Monte Carlo-error**

$$\nabla \bar{R}_\theta \approx \frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T_n}\left(\underbrace{\sum\nolimits_{t'=t}^{T_n}\gamma^{t'-t} r_{t'}^n - V^{\pi_\theta}(s_t^n)}_{\text{critic}}\right)\underbrace{\nabla log p_\theta(a_t^n|s_t^n)}_{\text{actor}}$$

Asynchronously Advantage Actor-Critic (A3C)



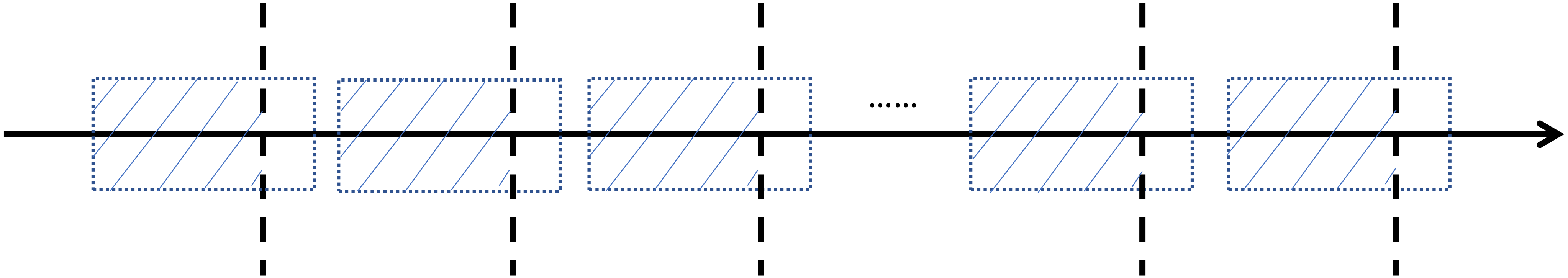
Reinforcement Learning Application in ads

Problem

- **Give adjusted factor α**
- **Choose click log**
  - $\mathrm{Max\_cpa} * \alpha > \mathrm{ecpa}$
- **Compute score**
  - $\sqrt{\frac{gmv}{cost} \times \frac{gmv}{base\ gmv}}$
  - 综合衡量指标R

	adgroupid	current	siteset	fraud_type	cost	report_conversion	target_cpa	max_cpa	report_time	ecpa
962	71979193	10	25.0	1.0	0.0	0	2625.0	4523.0	-1	1829.689427
963	71979193	10	25.0	1.0	0.0	0	2625.0	4523.0	-1	2293.199967
964	71979193	10	25.0	0.0	107.0	0	2625.0	4523.0	-1	2652.972370
965	71979193	10	25.0	0.0	20.0	0	2625.0	4523.0	-1	2691.336172
966	71979193	10	25.0	0.0	30.0	0	2625.0	4523.0	-1	2729.030843
967	71979193	10	25.0	0.0	56.0	0	2625.0	4523.0	-1	2748.947779
968	71979193	10	25.0	1.0	0.0	0	2625.0	4523.0	-1	2799.494196
969	71979193	10	25.0	0.0	67.0	0	2625.0	4523.0	-1	3016.184124
970	71979193	10	25.0	0.0	79.0	0	2625.0	4523.0	-1	3028.532611
971	71979193	10	25.0	0.0	100.0	0	2625.0	4523.0	-1	3088.102674
972	71979193	10	25.0	0.0	70.0	0	2625.0	4523.0	-1	3182.256031
973	71979193	10	25.0	0.0	106.0	0	2625.0	4523.0	-1	3268.750431
974	71979193	10	25.0	0.0	63.0	0	2625.0	4523.0	-1	3341.317137
975	71979193	10	25.0	0.0	69.0	0	2625.0	4523.0	-1	3348.984903
976	71979193	10	25.0	0.0	69.0	0	2625.0	4523.0	-1	3348.984903
977	71979193	10	25.0	1.0	0.0	1	2625.0	4523.0	11	3348.984903
978	71979193	10	25.0	0.0	75.0	0	2625.0	4523.0	-1	3349.138439
979	71979193	10	25.0	0.0	53.0	0	2625.0	4523.0	-1	3361.309150
980	71979193	10	25.0	0.0	61.0	0	2625.0	4523.0	-1	3411.860949
981	71979193	10	25.0	0.0	61.0	0	2625.0	4523.0	-1	3411.860949
...	...	...	...	...	...	...	...	...	...	...
1020	71979193	10	25.0	0.0	40.0	0	2625.0	4523.0	-1	4382.211683

Problem



- 1st interval
  - Conversion
  - Cost
- 2nd interval
  - Conversion
  - Cost
- 3rd interval
  - Conversion
  - Cost
- n-1th interval
  - Conversion
  - Cost
- nth interval
  - Conversion
  - Cost

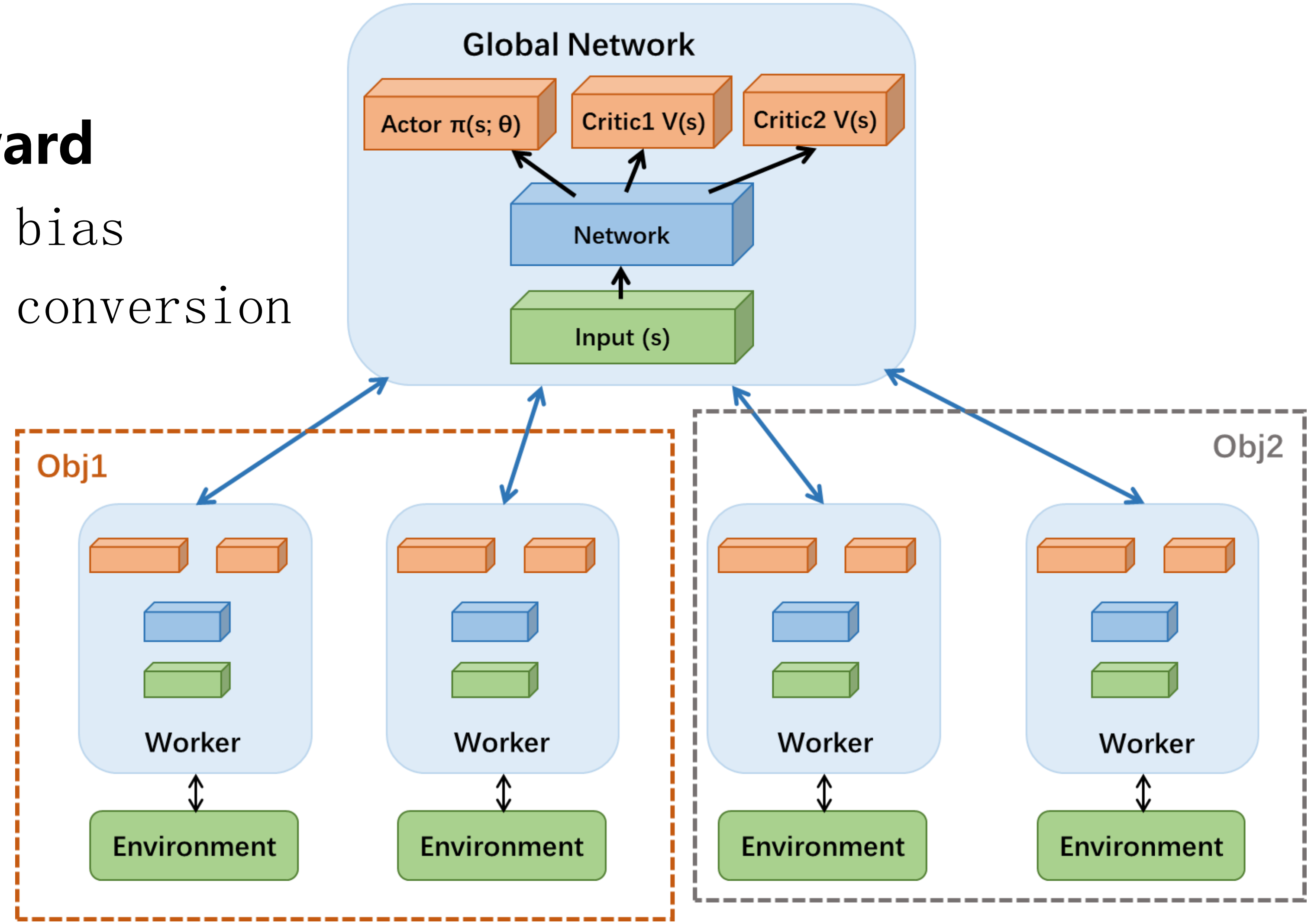
- **total**
  - Conversion
  - Cost
- **compute**
  - $\sqrt{\frac{gmv}{cost} \times \frac{gmv}{base\ gmv}}$
  - 综合衡量指标R

Experiment setting

- **State: feature vector**
  - $\langle$pError, iError, dError, timestamp, bias, $\cdots\rangle$
- **Action: adjust factor**
  - Discrete action
  - Continuous action, $a \sim N(\mu, \sigma)$
- **Reward: evaluation**
  - $\sqrt{\frac{gmv}{cost} \times \frac{gmv}{base\ gmv}}$
  - 综合衡量指标R
- **Method: A3C**

idea

- **Use separated reward**
  - Reward1: adjust bias
  - Reward2: adjust conversion



idea

- **Reward1: adjust bias**

$$Reward_1\left(\frac{CPA_{real}^{(j)}}{CPA_{target}^{(j)}}\right)=\begin{cases}-x^2, & 1.2<x\\ 1-x, & 1<x\leq 1.2\\ 1, & x\leq 1\end{cases}.$$

- **Reward2: adjust conversion**

$$Reward_2\left(\frac{conversions^{(i,j)}}{conversions_{PID}^{(i,j)}}\right)=\begin{cases}\frac{1}{1+e^x}, & 1<x\\ x, & 0.8<x\leq 1\\ x-0.8, & x\leq 0.8\end{cases},$$

Current result

- **The result is on training ads**

Model	Revenue	Cost	ROI
A2C	1.0019 (+0.19%)	1.0366 (+3.66%)	0.9665 (-3.35%)
DQN	0.9840 (-2.60%)	0.9765 (-2.35%)	1.0076 (+0.76%)
Agg-A3C	1.0625 (+6.25%)	1.0952 (+9.52%)	0.9702 (-2.98%)
O1-A3C	0.9744 (-2.56%)	0.9580 (-4.20%)	1.0170 (+1.70%)
O2-A3C	1.0645 (+6.45%)	1.0891 (+8.91%)	0.9774 (-2.26%)
MoTiAC	1.0421 (+4.21%)	1.0150 (+1.50%)	1.0267 (+2.67%)

Table 2: Comparative Result based on PID

Multi-process

- **Problem: large click log**
  - Share memory
- **Solution**
  - **Redis, multiprocessing.manager**
    - Not work (due to the overhead of pickle)
  - **Ctypes, multiprocessing.sharedctypes**
    - Work

Multi-process

```python

class MockAdRequestStructure(Structure):
    _fields_ = [
        ('ad_id_feature', AdIdFeatureStructure),
        ('bidding_info_list', (POINTER(AdBiddingInfoStructure) * common.DAY_OP_COUNT)),
        ('bidding_len_list', (c_int * common.DAY_OP_COUNT)),
        ('conversion_info_list', (POINTER(AdBiddingInfoStructure) * common.DAY_OP_COUNT)),
        ('conversion_len_list', (c_int * common.DAY_OP_COUNT)),
    ]

    def __init__(self, ad_request):
        self.ad_id_feature = AdIdFeatureStructure(ad_request.ad_id_feature)
        self.bidding_info_list, self.bidding_len_list = \
            assemble_list(ad_request.bidding_info_list)
        self.conversion_info_list, self.conversion_len_list = \
            assemble_list(ad_request.conversion_info_list)

```

- **Ctypes class**

```python

class MockAdRequest(object):
    '''根据点击日志仿真广告的一次请求
    '''
    def __init__(self):
        '''ad_id_feature存储广告的id类信息
           bidding_info_list存储每次点击的出价信息,是AdBiddingInfo的list
           conversion_info_list存储发生了转化的点击出价信息
        '''
        self.ad_id_feature = None
        self.bidding_info_list = [[] for _ in range(0, DAY_OP_COUNT)]
        self.conversion_info_list = [[] for _ in range(0, DAY_OP_COUNT)]

    def set_id_feature(self, array):
        '''根据点击日志设置广告的id类信息
        '''
        self.ad_id_feature = AdIdFeature(array)

    def set_request(self, index, array, start_time_stamp):
        '''根据点击日志设置广告的一次请求
        '''
        bidding_info = AdBiddingInfo(array, start_time_stamp)
        self.bidding_info_list[index].append(bidding_info)
        bidding_info = AdBiddingInfo(array, start_time_stamp)
        if bidding_info.conversion_num > 0:
            self.conversion_info_list[index].append(bidding_info)

    def transform(self):
        ''' 把点击日志按照ecpa进行排序,然后逐个累加,
            调价后则比调节后的ecpa小的点击仍然能够赢得竞价
        '''
```

- **python class**

Multi-process

- **Machine: 126G memory**
- **5 workers experiment**
  - 20190109 all ads loading
  - Shared memory: 20.2G / 126G
  - Not share: 84.5G / 126G
- **Similar running time**

Future Work About Our Project

Future work

- **Change the definition of state**

  - To address sparsity of state

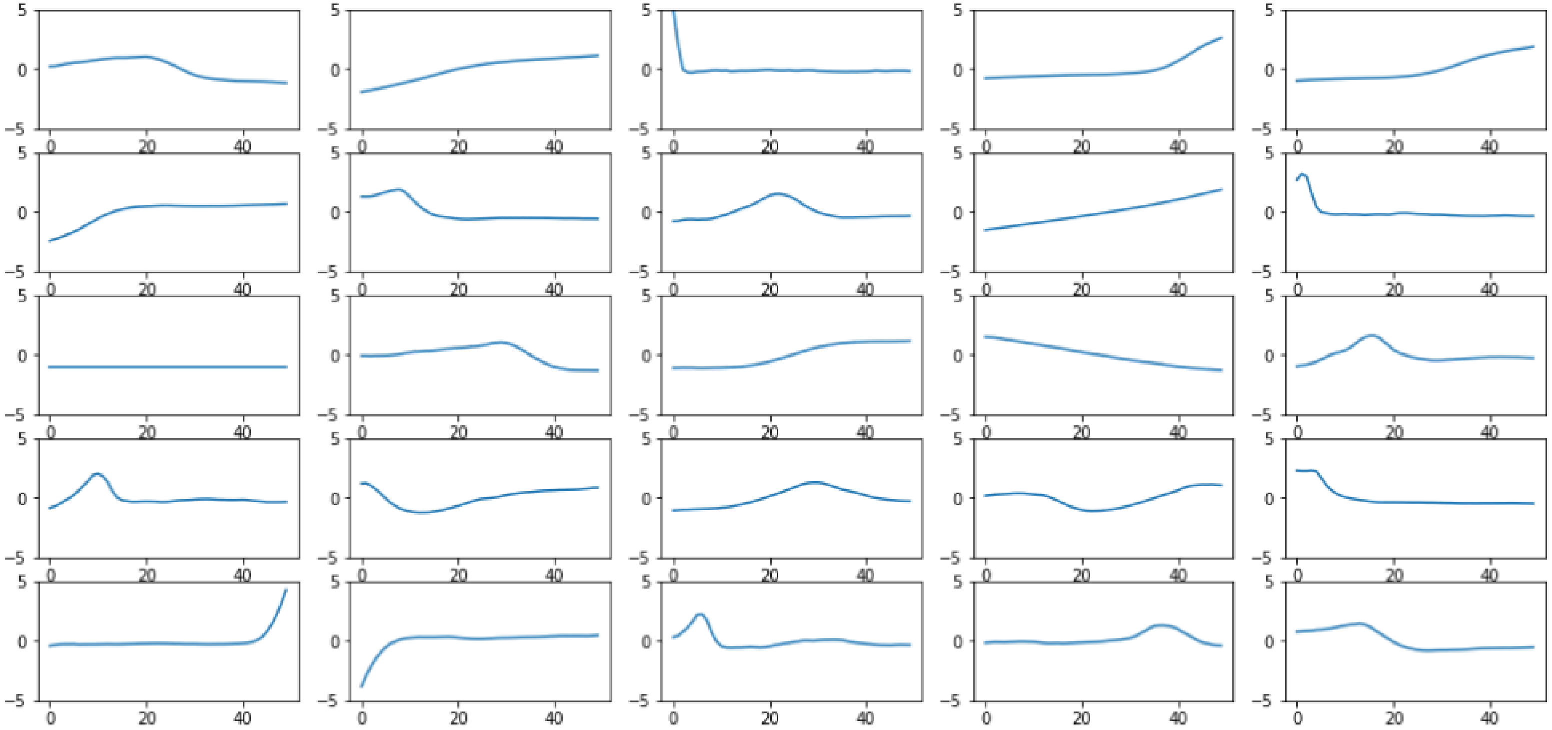
- **State definition**

  - 〈Bias pattern， conversion pattern〉

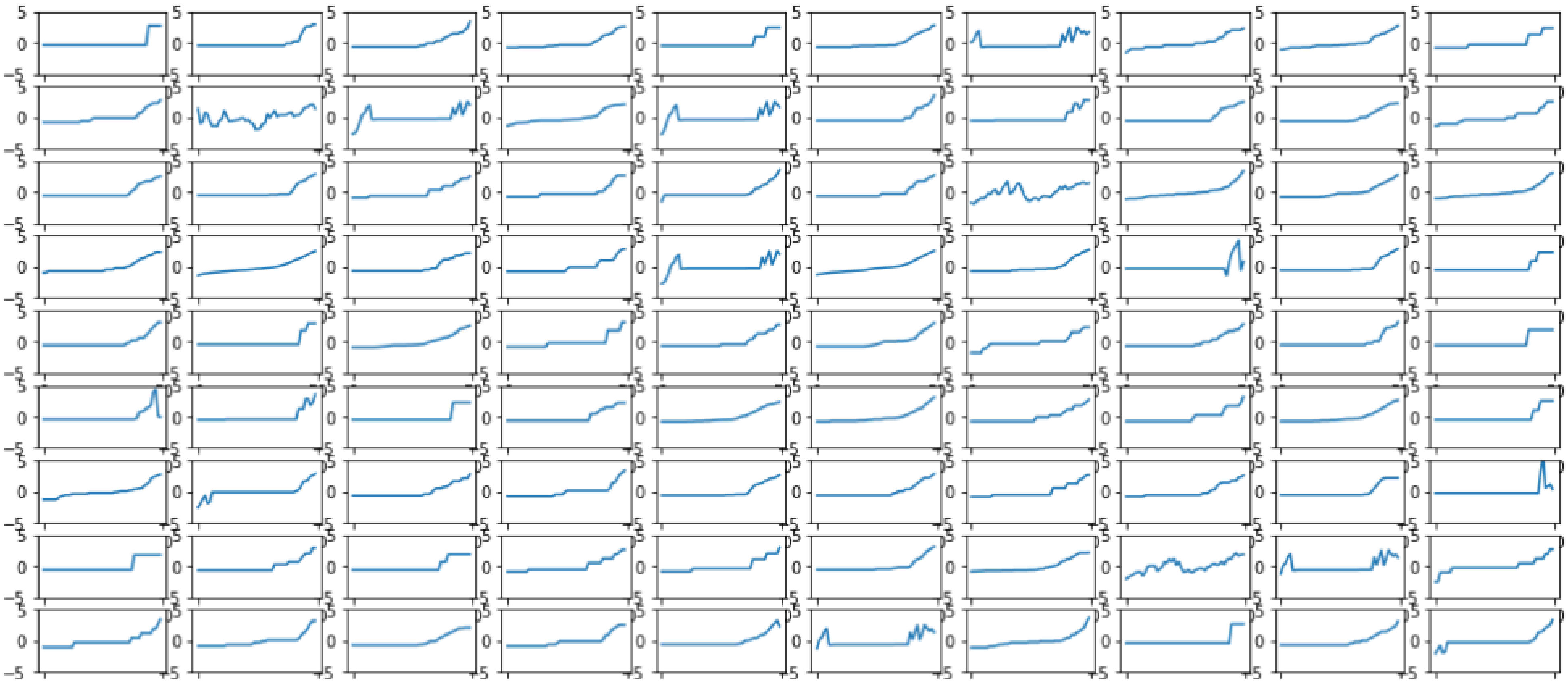
  - Clustering N bias/conversion pattern

Conversion clustering center

- **N = 25**

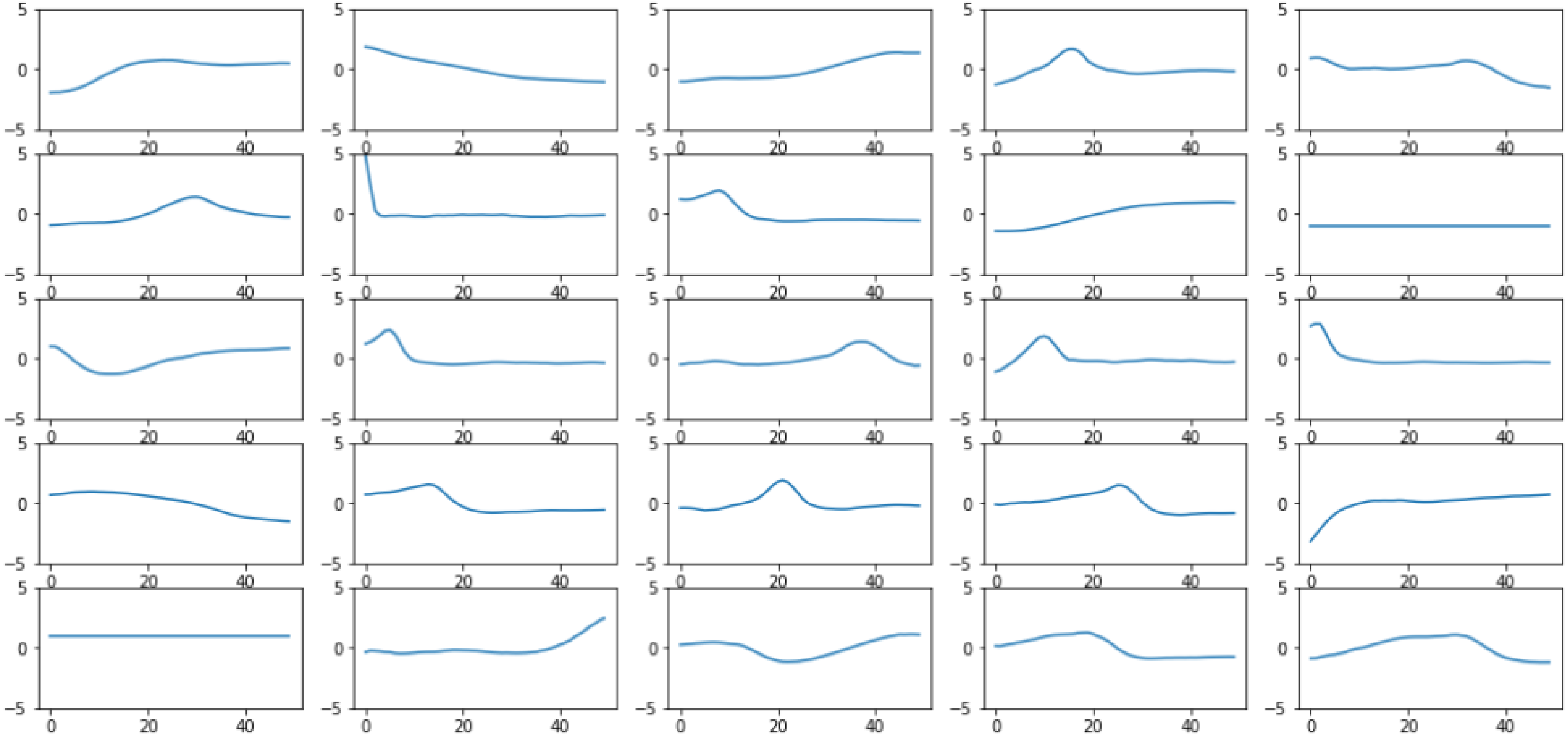


The 4th conversion cluster



Bias clustering center

- **N = 25**



The 4th bias cluster