



THE UNIVERSITY OF  
**SYDNEY**

# Advanced Machine Learning

(COMP 5328)

## Reinforcement Learning

Tongliang Liu



THE UNIVERSITY OF  
SYDNEY

# Review



THE UNIVERSITY OF  
SYDNEY

# Learning with noisy labels

## Problem Setup

- Given the training examples  $\{(X_i, \tilde{Y}_i)\}_{1 \leq i \leq n} \sim D_\rho(X, \tilde{Y})^n$ .
- The target is to learn a discriminant function  $f_n: \mathcal{X} \rightarrow \mathbb{R}$  such that the classifier predicts the correct label  $y$  given an observation  $x$ .



# Model Label Noise

A probabilistic model:

$$\rho_Y(X) = P(\tilde{Y}|Y, X),$$

where  $X$  is the feature,  $Y$  is the unobservable true label, and  $\tilde{Y}$  is the observed noisy label.

$$\rho_{+1}(X) = P(\tilde{Y} = -1|Y = 1, X); \rho_{-1}(X) = P(\tilde{Y} = 1|Y = -1, X).$$

Note that if there is no label noise, we have

$$P(\tilde{Y} = 1|Y = 1, X) = P(\tilde{Y} = -1|Y = -1, X) = 1$$

otherwise

$$P(\tilde{Y} = 1|Y = -1, X), P(\tilde{Y} = -1|Y = 1, X) \in (0,1).$$



# Model Label Noise

(1) Random Classification Noise (RCN):

$$\rho_Y(X) = P(\tilde{Y}|Y, X) = P(\tilde{Y}|Y); \rho_{+1}(X) = \rho_{-1}(X) = \rho.$$

(2) Class-Dependent Noise (CCN): *independent w.r.t instance  
depend on label*

$$\rho_Y(X) = P(\tilde{Y}|Y, X) = P(\tilde{Y}|Y); \rho_{+1}(X) = \rho_{+1}, \rho_{-1}(X) = \rho_{-1}.$$

(3) Instance- and Label-Dependent Noise (ILN): *depend on label & instance*

$$\rho_Y(X) = P(\tilde{Y}|Y, X).$$



# Random Classification Noise (RCN)

## How to Reduce the Effects of RCN?

Symmetric loss function is robust to RCN when the function class  $\mathcal{F}_{lin}$  is extended to the universal function space, which means the function in it can be of any form.

**Theorem I.** The losses satisfying the following symmetric criterion are robust to RCN:

$$L(f(X), +1) + L(f(X), -1) = C,$$

where  $C$  is a constant. That is

$$\arg \min_f R_{D,L}(f) = \arg \min_f R_{D_\rho,L}(f).$$

Van Rooyen, Brendan, Aditya Menon, and Robert C. Williamson. "Learning with symmetric label noise: The importance of being unhinged." Advances in Neural Information Processing Systems. 2015.



THE UNIVERSITY OF  
SYDNEY

# Class-dependent Label Noise

Modifying the loss function  $L$  to  $\tilde{L}$  such that

$$\arg \min_{f \in \mathcal{F}} R_{D,L}(f) = \arg \min_{f \in \mathcal{F}} R_{D_\rho, \tilde{L}}(f).$$

Methods: **Importance reweighting**, unbiased estimator, cost-sensitive loss, rank pruning.....

Liu, Tongliang, and Dacheng Tao. "Classification with noisy labels by importance reweighting." *IEEE Transactions on pattern analysis and machine intelligence* 38.3 (2016): 447-461.

Natarajan, Nagarajan, et al. "Learning with noisy labels." *Advances in neural information processing systems*. 2013.



THE UNIVERSITY OF  
SYDNEY

# Class-dependent Label Noise

$$R_{D,L}(f) = \mathbb{E}_{(X,Y) \sim D_\rho} [\beta(X, Y) L(f(X), Y)]$$

$$\text{where } \beta(x, y) = \frac{P_D(X=x, Y=y)}{P_{D_\rho}(X=x, \tilde{Y}=y)} = \frac{P_{D_\rho}(\tilde{Y}=y|X=x) - \rho - y}{(1-\rho+1-\rho-1)P_{D_\rho}(\tilde{Y}=y|X=x)}.$$

Liu, Tongliang, and Dacheng Tao. "Classification with noisy labels by importance reweighting." IEEE Transactions on pattern analysis and machine intelligence 38.3 (2016): 447-461.

# Learning with noisy labels

Let  $T$  be the following flip matrix (also called transition matrix), e.g.,

$$T = \begin{bmatrix} P(\tilde{Y} = 1|Y = 1) & P(\tilde{Y} = 1|Y = 2) & \dots & P(\tilde{Y} = 1|Y = C) \\ P(\tilde{Y} = 2|Y = 1) & P(\tilde{Y} = 2|Y = 2) & \dots & P(\tilde{Y} = 2|Y = C) \\ \vdots & \vdots & \vdots & \vdots \\ P(\tilde{Y} = C|Y = 1) & P(\tilde{Y} = C|Y = 2) & \dots & P(\tilde{Y} = C|Y = C) \end{bmatrix}.$$

If we assume that given the clean label, the noisy label is independent with the instance, we have that  $P(\tilde{Y}|Y) = P(\tilde{Y}|Y, X)$ , and that

Forward

$$[P(\tilde{Y} = 1|X), \dots, P(\tilde{Y} = C|X)]^\top = T [P(Y = 1|X), \dots, P(Y = C|X)]^\top,$$

or  $[P(Y = 1|X), \dots, P(Y = C|X)]^\top = T^{-1} [P(\tilde{Y} = 1|X), \dots, P(\tilde{Y} = C|X)]^\top$ .

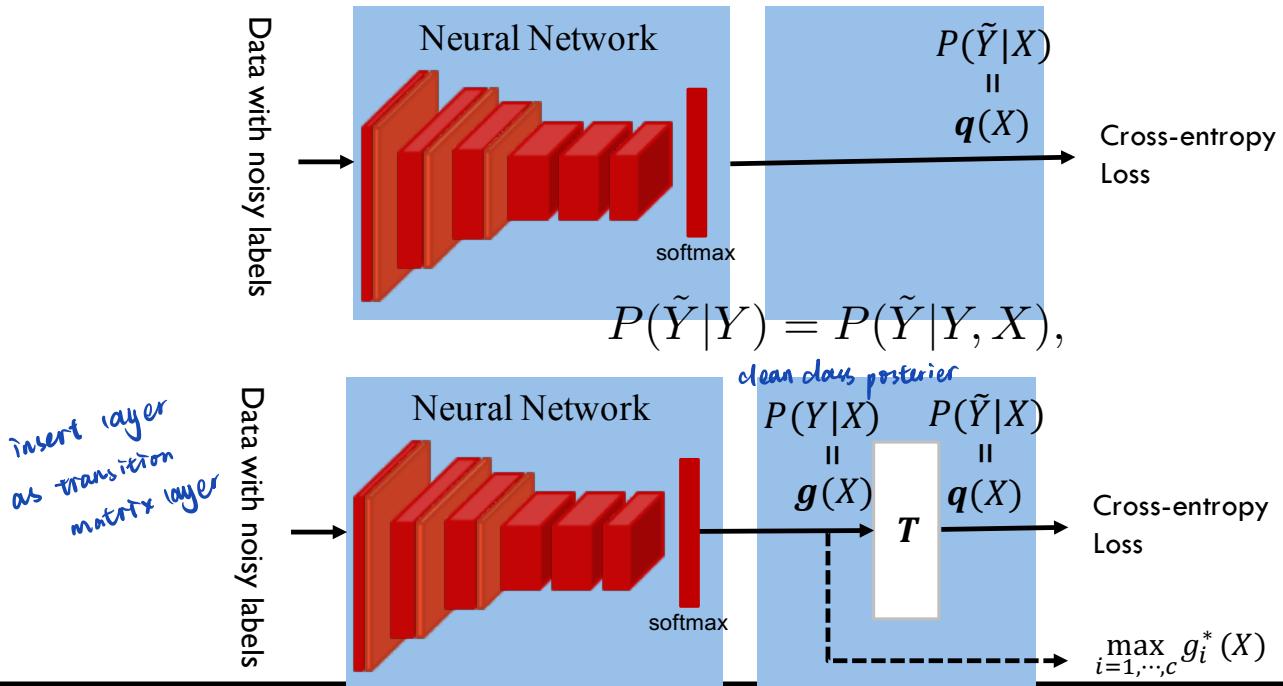
Backward

The above means that we can infer the clean class posterior by employing the noisy class posterior and the inverse transition matrix.

# Learning with noisy labels

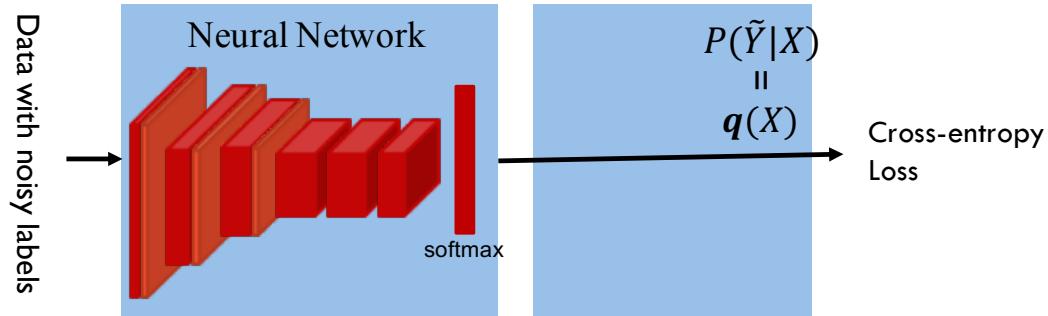
Forward learning:

$$[P(\tilde{Y} = 1|X), \dots, P(\tilde{Y} = C|X)]^\top = T [P(Y = 1|X), \dots, P(Y = C|X)]^\top.$$



# Learning with noisy labels

Backward learning:



We have that

$$P(Y|X) = \mathbf{g}(X) = T^{-1}q(X)$$

because  $[P(Y = 1|X), \dots, P(Y = C|X)]^\top = T^{-1}[P(\tilde{Y} = 1|X), \dots, P(\tilde{Y} = C|X)]^\top$ .



# Estimate the transition matrix

# Anchor Point Assumption

An instance  $x$  is called an anchor point for the  $i$ -th class, if

$$P(Y = i | X = x) = 1$$

In other words, the instance is with probability one in the  $i$ -th class. No change to be assigned to other classes.

Recall that, in the lecture of learning with label noise, we have exploited anchor points to estimate  $\rho_{+1}$  and  $\rho_{-1}$ .

$$P(\tilde{Y} = -1 | Y=1, x) \quad P(\tilde{Y} = 1 | Y=-1, x).$$



# Anchor Point Assumption

for binary case problem,

locate points with minimum value  $\rightarrow P(\tilde{Y} = -1 | X = x)$   
(which  $x$  achieves the min value)

## Noise rate estimation

$$P(\tilde{Y} = -1 | X = \mathbf{x}_{+1}) = (1 - \rho_{+1} - \rho_{-1}) P(Y = -1 | X = \mathbf{x}_{+1}) + \rho_{+1}$$

$$P(\tilde{Y} = +1 | X = \mathbf{x}_{-1}) = (1 - \rho_{+1} - \rho_{-1}) P(Y = +1 | X = \mathbf{x}_{-1}) + \rho_{-1}$$

↓  
0

$$P(\tilde{Y} = -1 | X = \mathbf{x}_{+1}) = \rho_{+1}$$

$$P(\tilde{Y} = -1 | X = \mathbf{x}) \geq \rho_{+1}$$

$$P(\tilde{Y} = +1 | X = \mathbf{x}_{-1}) = \rho_{-1}$$

$$P(\tilde{Y} = +1 | X = \mathbf{x}) \geq \rho_{-1}$$

We designed the following estimator:  $\rho_{-y} = \min_{X \in \mathcal{X}} P(\tilde{Y} = y | X)$

# Estimate the Transition Matrix

How to learn  $T$ ? Recall that

$$\begin{bmatrix} P(\tilde{Y} = 1|X) \\ \vdots \\ P(\tilde{Y} = C|X) \end{bmatrix} = \begin{bmatrix} P(\tilde{Y} = 1|Y = 1) & \cdots & P(\tilde{Y} = 1|Y = C) \\ \vdots & \ddots & \vdots \\ P(\tilde{Y} = C|Y = 1) & \cdots & P(\tilde{Y} = C|Y = C) \end{bmatrix} \begin{bmatrix} P(Y = 1|X) \\ \vdots \\ P(Y = C|X) \end{bmatrix}.$$

$T$

Let  $x^1$  be an anchor point for the first class, i.e.,

$$[P(Y = 1|X = x^1) = 1, P(Y = 2|X = x^1) = 0, \dots, P(Y = C|X = x^1) = 0]^\top.$$

We will show that given anchor points, the transition matrix can be learned.

# Estimate the Transition Matrix

Let  $x^1$  be an anchor point for the first class. We have

$$\begin{bmatrix} P(\tilde{Y} = 1|X = x^1) \\ \vdots \\ P(\tilde{Y} = C|X = x^1) \end{bmatrix} = \begin{bmatrix} P(\tilde{Y} = 1|Y = 1) & \cdots & P(\tilde{Y} = 1|Y = C) \\ \vdots & \ddots & \vdots \\ P(\tilde{Y} = C|Y = 1) & \cdots & P(\tilde{Y} = C|Y = C) \end{bmatrix} \begin{bmatrix} 1 \\ \vdots \\ 0 \end{bmatrix}$$
$$= \begin{bmatrix} P(\tilde{Y} = 1|Y = 1) \\ \vdots \\ P(\tilde{Y} = C|Y = 1) \end{bmatrix}.$$

Similarly, given the anchor points for the  $i$ -th class, we can learn the  $i$ -th column of the transition matrix, which equals

$$\begin{bmatrix} P(\tilde{Y} = 1|X = x^i) \\ \vdots \\ P(\tilde{Y} = C|X = x^i) \end{bmatrix}$$

where  $x^i$  represents an anchor point of the  $i$ -th class.

# Estimate the Transition Matrix

The problem remains to be how to find anchor points?

One intuitive way is to assume there are anchor points in the training sample and [treat instances with high i-th class posterior as the anchor points for the i-th class.]

For example, to find the anchor points for the first class, we can find instances  $x$  with high 1-th class posterior, i.e.,  $P(Y = 1|X = x)$ .

$$\Downarrow \cdot P(\tilde{Y}=1 | X) \nearrow$$



# Reinforcement Learning



# Supervised Learning and RL

## In supervised learning

- We have training data:  $S = \{x_i, y_i\}_{i=0}^k$ , Where  $x_i \in \mathcal{X}$  is features  $y_i \in \mathcal{Y}$  is a label.
- Goal: Learn a function  $f_S : \mathcal{X} \rightarrow \mathcal{Y}$  from the labeled dataset  $S$ .

## In reinforcement learning

- An agent interacts with the environment through different actions, and the environment provides numeric reward signals.
- Goal: Learn how to take actions in order to maximise the reward.



# Introduction of RL

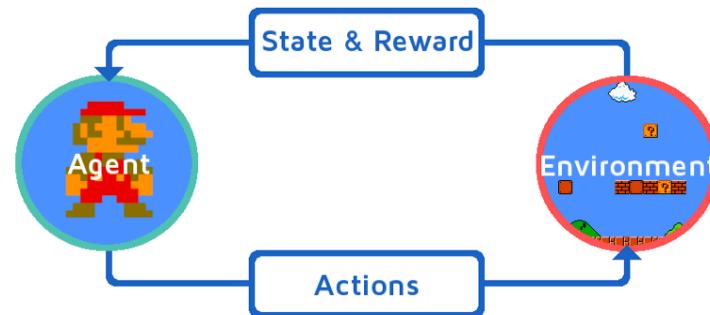
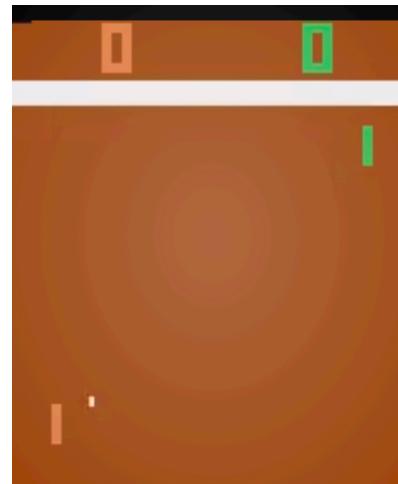
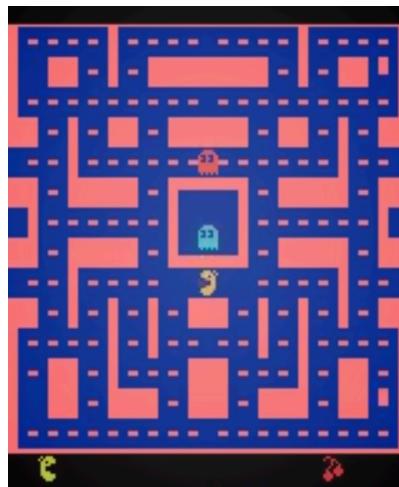


Figure: Atari Games on openAI gym and mario game.(<https://gym.openai.com/envs/#atari>)



# RL Framework (notation)

Let

- $\mathcal{A}$  to denote set of possible Actions
- $\mathcal{S}$  to denote set of possible States.
- $t$  to denote a time step.
- $r_t$  to denote an intermediate reward at time step  $t$ .
- $\pi$  to denote the policy (strategy) of an agent to choose its actions

# RL Framework (Procedure)



THE UNIVERSITY OF  
SYDNEY

An agent interacts with its environment according to following procedure.

- At each time step  $t$ , the agent is in some state  $S_t = s_t$ .
- It chooses an action  $A_t = a_t$ .
- Based on  $s_t$  and  $a_t$ , it receives reward  $R_t = r_t$  and goes into state  $S_{t+1} = s_{t+1}$ .

In general, the action  $A_t$ , reward  $R_t$  and next state  $S_{t+1}$  have the following probability distributions,

$$\pi(A_t|S_t = s_t), \quad P(S_{t+1}|A_t = a_t, S_t = s_t), \quad P(R_t|A_t = a_t, S_t = s_t).$$

# RL Framework (Objective)



THE UNIVERSITY OF  
SYDNEY

Given current state is  $s_0$ , we want to find the optimal policy (best strategy)  $\pi^*$  that maximises the expected cumulative reward, i.e.,

$$\pi^* = \arg \max_{\pi} \mathbb{E} \left[ \sum_{t \geq 0} \gamma^t r_t | S_0 = s_0, \pi \right],$$

with

$$a_t \sim \pi(A_t | S_t = s_t), \quad s_{t+1} \sim P(S_{t+1} | A_t = a_t, S_t = s_t),$$

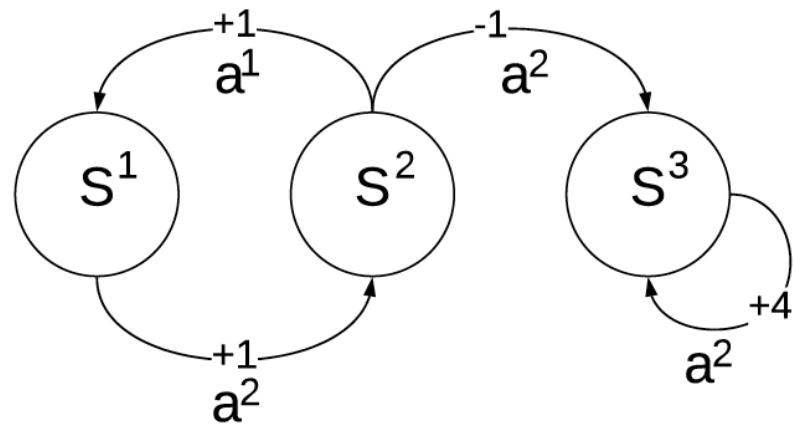
and a discounted factor

$$\gamma \in (0, 1)$$

# A Toy RL problem

(Diagram of the environment)

$$\mathcal{A} = \{a^1, a^2\}.$$
$$\mathcal{S} = \{S^1, S^2, S^3\}.$$



In this setting,

$$P(S_{t+1} = s^1 | A_t = a^1, S_t = s^2) = 1 \quad P(R_t = -1 | A_t = a^2, S_t = s^2) = 1$$

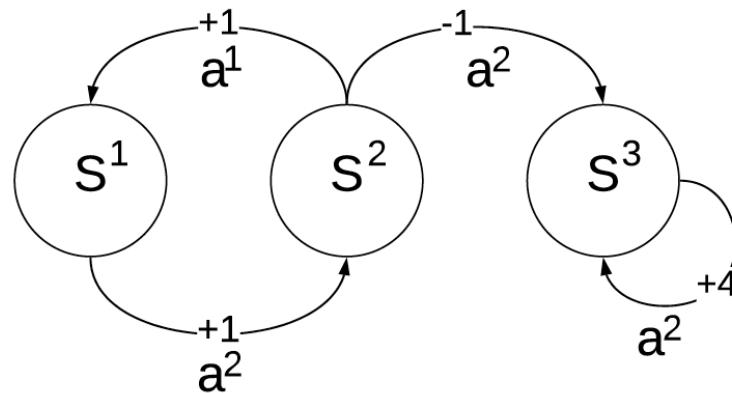


# A Toy RL problem

Given initial state  $s_0 = s^1$  which policy is better?

For all  $t \leq 0$ , we have

$$\begin{aligned}\pi^1(A_t = a^2 | S_t = s^1) &= 1, \quad \pi^1(A_t = a^1 | S_t = s^2) = 1, \quad \pi^1(A_t = a^2 | S_t = s^2) = 0, \\ \pi^1(A_t = a^2 | S_t = s^3) &= 1; \\ \pi^2(A_t = a^2 | S_t = s^1) &= 1, \quad \pi^2(A_t = a^1 | S_t = s^2) = 0, \quad \pi^2(A_t = a^2 | S_t = s^2) = 1, \\ \pi^2(A_t = a^2 | S_t = s^3) &= 1.\end{aligned}$$





# A toy RL problem

Trajectories produced by  $\pi^1$  and  $\pi^2$  as follows,

$\pi^1$	$t_0$	$t_1$	$t_2$	$t_3$	$t_4$	...
Sate	$s^1$	$s^2$	$s^1$	$s^2$	$s^1$	...
Action	$a^2$	$a^1$	$a^2$	$a^1$	$a^2$	...
Reward	+1	+1	+1	+1	+1	...
$\pi^2$	$t_0$	$t_1$	$t_2$	$t_3$	$t_4$	...
Sate	$s^1$	$s^2$	$s^3$	$s^3$	$s^3$	...
Action	$a^2$	$a^2$	$a^2$	$a^2$	$a^2$	...
Reward	+1	-1	+4	+4	+4	...

Under this environment, the optimal policy  $\pi^* = \pi^2!$

# Q-value function

Q-value function  $Q : S \times A \rightarrow \mathbb{R}$  determines the expected cumulative reward by following the policy  $\pi$  given current state is  $s_0$ , and action is  $a_0$ , specifically,

$$Q(s_0, a_0) = \mathbb{E} \left[ \sum_{t \geq 0} \gamma^t r_t | S_0 = s_0, A_0 = a_0, \pi \right],$$

help us to choose action  
 $\uparrow$   
 $=$

with

$$a_t \sim \pi(A_t | S_t = s_t), \quad s_{t+1} \sim P(S_{t+1} | A_t = a_t, S_t = s_t),$$

$$r_t \sim P(R_t | A_t = a_t, S_t = s_t)$$

and  $\gamma$  is the discounted factor.



# Optimal Q-value function

The optimal Q-value (denoted as  $Q^*$ ) is the maximum expected cumulative reward achievable by following the optimal policy  $\pi^*$  given that the current state is  $s_0$  and action is  $a_0$ , specifically,

$$Q^*(s_0, a_0) = \max_{\pi} \mathbb{E} \left[ \sum_{t \geq 0} \gamma^t r_t | S_0 = s_0, A_t = a_0, \pi \right]$$

with

$$a_t \sim \pi(A_t | S_t = s_t), \quad s_{t+1} \sim P(S_{t+1} | A_t = a_t, S_t = s_t)$$

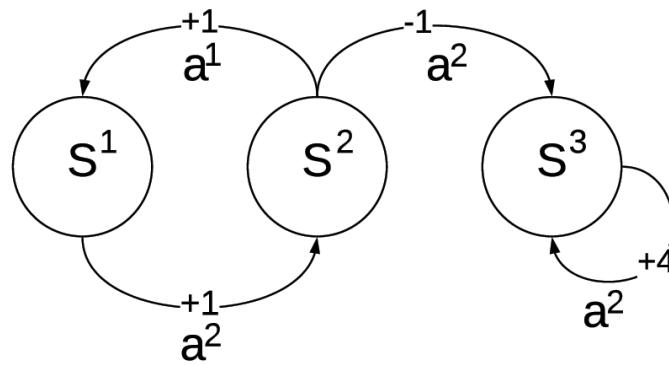
and  $r^t$  is the immediate reward for the state-action pair  $(s_t, a_t)$



# Optimal Q-value function (Example)

What is the Optimal Q-value start from state  $s^2$ , and action  $a_1$ , where for all  $t \leq 0$ , we have

$$\pi^*(A_t = a^2 | S_t = s^1) = 1, \pi^*(A_t = a^1 | S_t = s^2) = 0, \pi^*(A_t = a^2 | S_t = s^2) = 1,$$
$$\pi^*(A_t = a^2 | S_t = s^3) = 1.$$



# Optimal Q-value function (Example)

What is the Optimal Q-value start from state  $s^2$ , and action  $a_1$ ?

$$\begin{aligned}Q^*(s^2, a^1) &= 1 + \gamma 1 - \gamma^2 1 + \gamma^3 4 + \gamma^4 4 \dots \gamma^\infty 4 \\&= 1 + \gamma 1 - \gamma^2 1 + \sum_{t=3}^{\infty} \gamma^t 4,\end{aligned}$$

where  $\gamma \in (0, 1)$ .

# Update Q-value function

The optimal Q-value function  $Q^*$  also satisfies the following Bellman equation:

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

$$Q(s_0, a_0) = \mathbb{E} \left[ \sum_{t \geq 0} \gamma^t r_t | S_0 = s_0, A_0 = a_0, \pi \right]$$

# Update Q-value function

*this step*

In the RL problem, the environment **only returns immediate rewards**, the optimal Q-value function is not given, we could use the Bellman equation to update estimated Q-value function iteratively, i.e.,

$$\begin{aligned}
 Q(s, a) &= r + \gamma \mathbb{E}_{s' \sim \mathcal{S}} \left[ \max_{a'} Q(s', a') \right] \\
 &= \mathbb{E}_{s' \sim \mathcal{S}} \left[ Q(s, a) + r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right]. \\
 &= Q(s, a) + \mathbb{E}_{s' \sim \mathcal{S}} \left[ r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right].
 \end{aligned}$$

# Update Q-value function

Empirically, we could use the following equation to update Q-value function

$$Q_{i+1}(s, a) = Q_i(s, a) + \eta \left( r + \gamma \max_{a'} Q_i(s', a') - Q_i(s, a) \right),$$

where  $\eta \in (0, 1)$ , and  $i$  is the number of iterations.

Intuitively,  $r + \gamma \max_{a'} Q_i(s', a')$  is a more accurate approximation of the Q-value compared to  $Q_i(s, a)$  because it contains a true value  $r$ .

We add a hyper-parameter  $\eta$  because the estimation  $r + \gamma \max_{a'} Q_i(s', a')$  can not be fully trusted. If  $i$  goes to infinity,  $Q_i$  will converge to  $Q^*$  (see proof [I]).

# Store Q-values

To update and track the changes of Q-values for state-action pairs, we need to store Q-values in somewhere. We could use a lookup table to store Q-values, e.g.,

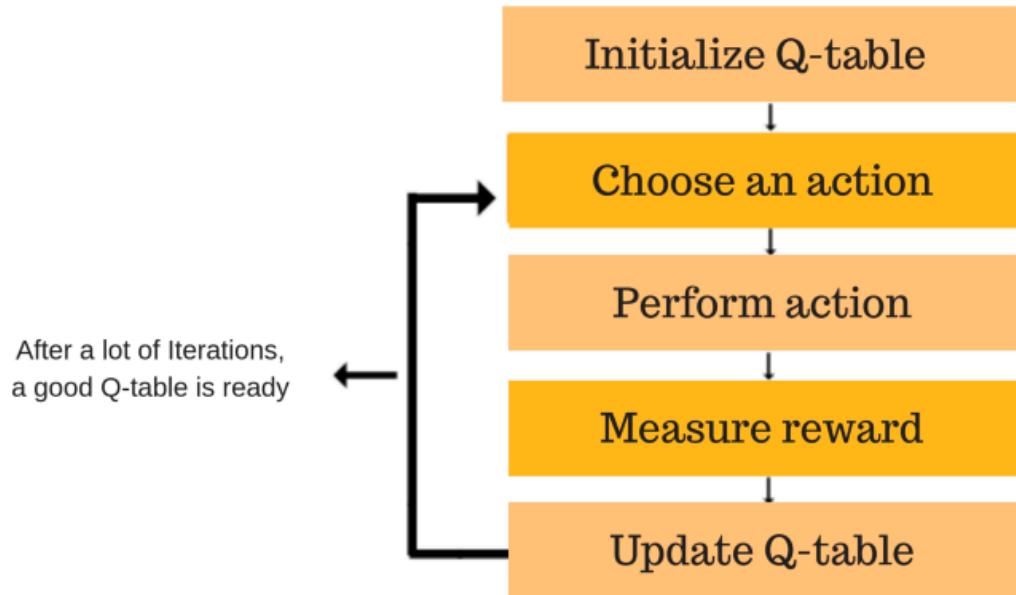
$Q_i$	$s^1$	$s^2$	$s^3$	...
$a^1$	5	2	-	...
$a^2$	-1	-6	-4	...
$a^3$	5	2	3	...
...	...	...	...	...

Table: A Q Table at  $i$ -th iteration

Each entry in this table is the estimated Q-value for a state-action pair at the  $i$ -th iteration. Then we update the values contained in the table by the updating rule to get the Q table at the  $i + 1$ -th iteration.



# Q-Table



Credit: <https://www.freecodecamp.org/news/a-brief-introduction-to-reinforcement-learning-7799af5840db/>



# Q-learning

## Algorithm 1 Q-Learning

Initialize the Q value  $Q(s, a)$  for all state-action pairs arbitrarily

**for** each episode **do**

    Initialize  $s$

**for**  $s$  is not a terminal state **do**

        Choose  $a = \arg \max_a Q(s, a)$

        Substitute action  $a$  and  $s$  into the environment and observe  $r, s'$

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

$s \leftarrow s'$  //only update the state but not action

**end**

**end**



# Sarsa

Update both state & action

## Algorithm 2 Sarsa

Initialize the Q value  $Q(s, a)$  for all state-action pairs arbitrarily

**for** each episode **do**

  Initialize  $s$

  Choose  $a = \arg \max_a Q(s, a)$

**for**  $s$  is not a terminal state **do**

    Substitute action  $a$  and  $s$  into the environment and observe  $r, s'$

    Choose  $a' = \arg \max_{a'} Q(s', a')$

$Q(s, a) \leftarrow Q(s, a) + \eta (r + \gamma Q(s', a') - Q(s, a))$

$s \leftarrow s'$

$a \leftarrow a' //$ action  $a$  is also updated

**end**

**end**

题主理解的off-policy是正确的，产生经验的policy和GPI(generalize policy iteration)里面做policy evaluation的policy不是同一个。

所以在MC方法里面要用importance sampling来修正两个经验分布不一样的问题。

Q-learning为什么是一个off-policy的方法？

和SARSA对比起来看

SARSA的policy evaluation是

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

Q-learning的policy evaluation是

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$$

在SARSA中，TD target用的是当前对  $Q^*$  的估计。

而在Q-learning中，TD target用的是当前对  $Q^*$  的估计，可以看作是在evaluate另一个greedy的policy，所以说它是off-policy learning

那么为什么Q-learning用  $Q^*$  作为TD target？

Q-learning利用的是bellman optimality equation，所以可以这样更新，深入证明就要看bellman equation。

因为Q-learning不需要importance sampling，所以是RL进程上的一个突破性算法

发布于 2017-12-06 12:12

▲ 赞同 39 ▾ 3条评论 分享 收藏 喜欢



# Off-policy and On-policy Learning

## (Difference between Q-Learning and Sarsa)

- **Target policy:** the policy to (choose the actions to) update Q-values.
- **Behaviour policy:** the policy to update the action.

If the target policy is consistent with the behavior policy, it is **on-policy algorithm**, and off-policy otherwise.



# Off-policy and On-policy Learning

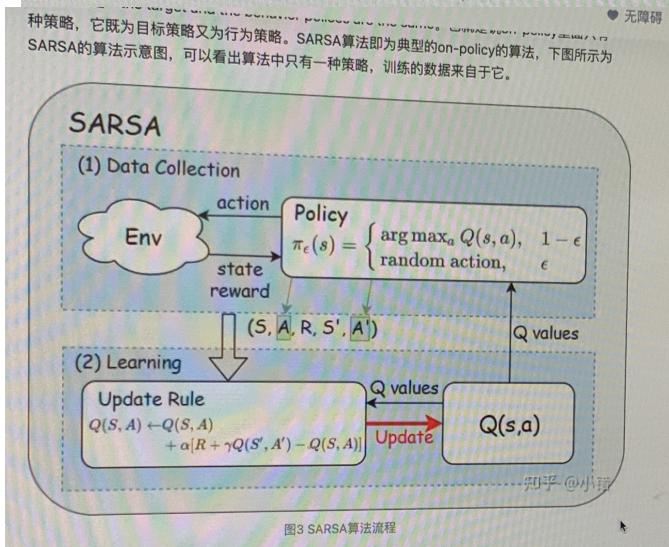
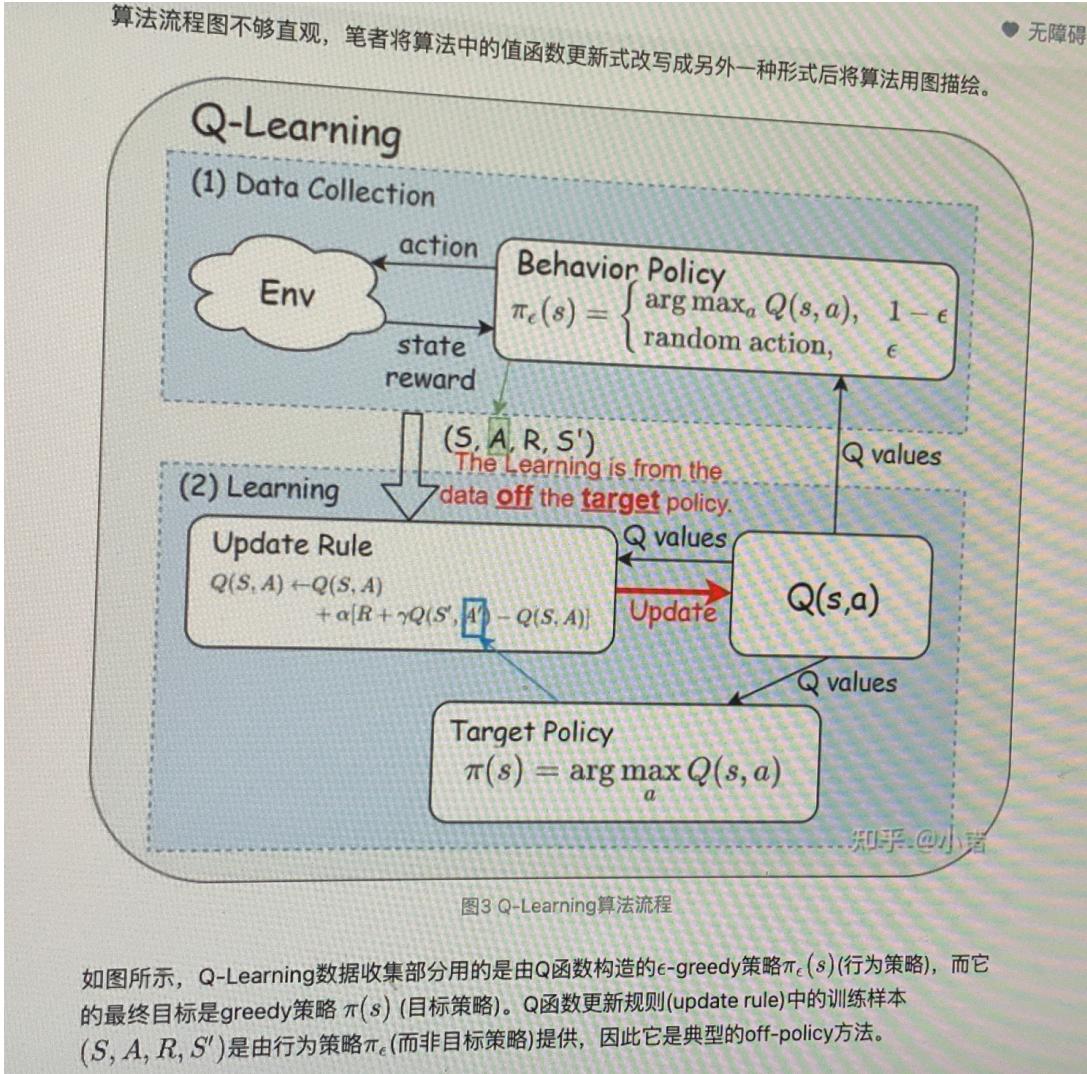
## (Difference between Q-Learning and Sarsa)

Sarsa is an **on-policy algorithm**, because it chooses the action  $a'$  to update Q-value, and it also choose the action  $a'$  as the next action.

*update action and state*

Q-learning is an **off-policy algorithm**, because it used the action  $a'$  to update Q-value, but  $a'$  may not be selected to be the next action.

*update only state*



困惑3：SARSA算法(on-policy)像是在进行值函数估计，为什么能收敛到最优策略？

更新式中用到的策略也是 $\epsilon$ -greedy算法。 $\epsilon$ 的概率是用随机动作，对策略的提升没有帮助；但另外有 $1 - \epsilon$ 的概率，按照 $\arg \max_a Q(s, a)$ 选取动作，即对策略进行提升。可以发现 $\epsilon$ -greedy策略是一种完全不同于用条件概率分布表示的策略，只要 $\epsilon$ 较小，它自带策略提升的功能。



# Deep Q-learning Network (DQN)

# Deep Q-learning Network

For some tasks both state and action could be continues. It is infeasible to store Q-value for every state-action pair into the Q table. Therefore, we use a neural network as a function approximator to store Q-values.



# Deep Q-learning Network

We parameterise  $Q$  by weight  $w$  of neural network, i.e.,  $Q_w$ .  
The objective function is to iteratively minimise

$$(r + \gamma \max_{a'} Q_w(s', a') - Q_w(s, a))^2,$$

where state  $s'$  is observed by substituting action  $a$  and  $s$  into the environment.

Note that gradient is applied only to  $\underbrace{Q_w(s, a)}_{\text{applied to current stage}}$ , not to  $Q_w(s', a')$



# DQN with Experience Replay

Learning from consecutive training examples (state and actions) is problematic, because there may exist strong temporal correlations between examples. To solve it, we can use a “experience bank” and randomly sample examples from the bank, specifically:

- Generating experiences (examples), each experience can be expressed by  $(s, a, r', s')$ .
- Using a database (experience bank) to store the experiences.
- Sample randomly from database and apply update, to minimise  $(r + \gamma \max_{a'} Q_w(s', a') - Q_w(s, a))^2$ .

*useful for learning : large objective function*



# DQN with Prioritised Experience Replay

Instead of randomly sampling examples from the "experience bank", we could add a weight to each example. The example with the larger weight will have higher chance to be sampled.

Specially, the weight of an example  $(s, a, r', s')$  can be positively related to  $(r + \gamma \max_{a'} Q_w(s', a') - Q_w(s, a))^2$ , in such way, the example which provides a larger update will have higher chance to be sampled.



# DQN with Prioritised Experience Replay

---

**Algorithm 3** DQN with Prioritised Experience Replay (simplified)

Initialize the Q value  $Q(s, a)$  for all state-action pairs arbitrarily

Initialize experience bank  $B = \emptyset$

Observe  $s_0$

**for**  $t = 1$  to  $T$  **do**

    Choose  $a = \arg \max_a Q_w(s_{t-1}, a)$

    Observe  $s_t$ ,  $r_t$  and  $\gamma_t$

    Store the transition  $(s_{t-1}, a_{t-1}, r_t, \gamma_t, s_t)$  in  $B$  with maximal priority

**if**  $t \equiv 0 \pmod K$  **to then**

        Sample a transition  $(s, a, r, \gamma, s', a')$  according to the priority

        Compute error  $\delta = (r + \gamma \max_{a'} Q_w(s', a') - Q_w(s, a))^2$

        Update the priority of current transition according to  $|\delta|$

        Update weights  $w \leftarrow w + \eta \nabla(r + \gamma \max_{a'} Q_w(s', a') - Q_w(s, a))^2$

**end**

**end**

---



# Policy Gradients

- Sometimes Q-function can be very complicated!  
For example, a robot grasping an object has a very high-dimensional state (image), and it is hard to learn exact value of every (state, action) pair.  
However the policy can be much simpler.

Why don't we directly learning the optimal policy?

- we can use the similar way as modelling Q-values, i.e., using a neural network to model policy  $\pi_\theta$ , where,  $\theta$  is the parameter of the network.

# Policy Gradients

The expected reward of a policy  $\theta$  is:

*maximise the expected  
cumulative reward*

$$J(\theta) = \mathbb{E} \left[ \sum_{t \geq 0} \gamma^t r_t | \pi_\theta \right],$$

$\uparrow$   
*parameterised by  $\theta$*

and we want to find the optimal policy (parameters)

$$\theta^* = \arg \max_{\theta} J(\theta).$$



# Calculate Gradient on $\theta$

Could we use gradient ascent on policy parameters?

Let  $r(\tau)$  be the reward of a trajectory:

$$\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \dots)$$

mathematically, we could write the expected reward  $J(\theta)$  as:

$$J(\theta) = \mathbb{E} \left[ \sum_{t \geq 0} \gamma^t r_t | \pi_\theta \right] = \mathbb{E}_{\tau \sim p(\tau; \theta)}^{policy} [r(\tau)] = \int_{\tau \sim p(\tau; \theta)} r(\tau) p(\tau; \theta) d\tau.$$

*to maximise  $\rightarrow$  gradient ascend*

# Calculate Gradient on $\theta$

We differentiate the expected reward  $J(\theta)$  :

$$\nabla_{\theta} J(\theta) = \int_{\tau \sim p(\tau; \theta)} r(\tau) \nabla_{\theta} p(\tau; \theta) d\tau,$$

where

- $r(\tau)$  is a mapping,
- $\nabla_{\theta} p(\tau; \theta)$  is the gradient for the distribution  $p(\tau; \theta)$ .



# Calculate Gradient on $\theta$

Suppose we have some trajectories  $\tau_1, \tau_2, \dots, \tau_n$ , could we approximate:

$$\nabla_{\theta} J(\theta) = \int_{\tau \sim p(\tau; \theta)} r(\tau) \nabla_{\theta} p(\tau; \theta) d\tau,$$

*not a distribution*

by summation (Monte Carlo methods)

*we can't approximate  
with empirical mean*

$$\nabla_{\theta} \hat{J}(\theta) = \frac{1}{n} \sum_{i=1}^n r(\tau_i) \nabla_{\theta} p(\tau; \theta)?$$

$$\begin{aligned} E[f(x)] &= \int f(x) p(x) dx \\ &\approx \frac{1}{n} \sum_{i=1}^n f(x_i) \end{aligned}$$

No! The integration is not an expected value! Because none of  $r(\tau)$  and  $\nabla_{\theta} p(\tau; \theta)$  are distributions! Then we can not approximate it by empirical mean!



# Calculate Gradient on $\theta$ (Log Derivative Trick)

$$\begin{aligned}\nabla_{\theta} J(\theta) &= \int_{\tau \sim p(\tau; \theta)} r(\tau) \nabla_{\theta} p(\tau; \theta) d\tau, \\ &= \int_{\tau \sim p(\tau; \theta)} r(\tau) p(\tau; \theta) \frac{\nabla_{\theta} p(\tau; \theta)}{p(\tau; \theta)} d\tau, \\ &\quad \text{f}(\tau_i | \theta) = r(\tau) \nabla_{\theta} \log p(\tau_i | \theta) \quad \downarrow \\ &= \int_{\tau \sim p(\tau; \theta)} \underbrace{r(\tau) p(\tau; \theta)}_{\text{f}(\tau_i | \theta)} \underbrace{\nabla_{\theta} \log p(\tau; \theta)}_{\text{f}(\tau_i | \theta)} d\tau, \\ &= \mathbb{E}_{\tau \sim p(\tau; \theta)} [r(\tau) \nabla_{\theta} \log p(\tau; \theta)].\end{aligned}$$

Then, we can proximate the expectation by

$$\nabla_{\theta} \hat{J}(\theta) = \frac{1}{n} \sum_{i=1}^n r(\tau_i) \nabla_{\theta} \log p(\tau_i; \theta).$$



# Calculate Gradient on $\theta$

Assume the **Markov condition holds**, i.e.,

$$P(s_{t+1}, a_t | s_t, s_{t-1}, \dots, s_0) = P(s_{t+1}, a_t | s_t),$$

then  $p(\tau; \theta) = \prod_{t \geq 0} p(s_{t+1} | s_t, a_t) \pi_\theta(a_t | s_t)$ . We have that,

$$\begin{aligned}\nabla_\theta \log p(\tau; \theta) &= \nabla_\theta \sum_{t \geq 0} \log p(s_{t+1} | s_t, a_t) + \log \pi_\theta(a_t | s_t), \\ &\stackrel{\textcolor{blue}{\nwarrow}}{=} \sum_{t \geq 0} \nabla_\theta \log p(s_{t+1} | s_t, a_t) + \nabla_\theta \log \pi_\theta(a_t | s_t), \\ &= \sum_{t \geq 0} \nabla_\theta \log \pi_\theta(a_t | s_t).\end{aligned}$$

Finally, we have a gradient estimator:

$$\nabla_\theta \hat{J}(\theta) = \frac{1}{n} \sum_{i=1}^n r(\tau_i) \sum_{t \geq 0} \nabla_\theta \log \pi_\theta(a_t | s_t) \stackrel{\textcolor{blue}{\nwarrow \text{ independent w.r.t } t}}{=} \frac{1}{n} \sum_{i=1}^n \sum_{t \geq 0} r(\tau_i) \nabla_\theta \log \pi_\theta(a_t | s_t).$$

# Policy Gradient

---

## Algorithm 4 Policy Gradient (simplified)

---

Initialize  $\theta$  arbitrarily

```
for each episode  $(s_0, a_0, r_0, \dots, s_T, a_T, r_T) \sim \pi_\theta$  do
     $\Delta\theta \leftarrow 0$ 
     $r \leftarrow 0$ 
    for  $t = 0$  to  $T$  do
         $\Delta\theta \leftarrow \Delta\theta + \alpha \nabla_\theta \log \pi_\theta(a_t | s_t)$ 
         $r \leftarrow r + \gamma r_t$ 
    end
     $\theta \leftarrow \theta + \eta \Delta\theta$ 
end
```

---