THE UNIVERSITY OF SYDNEY

Advanced Machine Learning (COMP 5328)

Reinforcement Learning

Tongliang Liu

Announcements

THE UNIVERSITY OF SYDNEY

- Assignment 2 is online now
- Assignment 2 due on 06/11/2022, 11:59pm
- Group-based (3-4 students per group). Find you teammates by yourselves.

Assignment 2

THE UNIVERSITY OF SYDNEY

Summary

The objective of this assignment is to design algorithms that are robust to label noise. Three input datasets are given. For each dataset, the training and validation data contains class-conditional random label noise, whereas the test data is clean. You need to build at least two different classifiers trained and validated on the noisy data, that can have a good classification accuracy on the clean test data.

THE UNIVERSITY OF SYDNEY

Review

Learning with noisy labels

THE UNIVERSITY OF SYDNEY

Problem Setup

- Given the training examples $\{(X_i, \tilde{Y}_i)\}_{1 \le i \le n} \sim D_\rho(X, \tilde{Y})^n$.

- The target is to learn a discriminant function f_n: $\mathcal{X} \to \mathbb{R}$ such that the classifier predicts the correct label y given an observation $\boldsymbol{x}$.

Model Label Noise

THE UNIVERSITY OF SYDNEY

A probabilistic model:

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

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

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

otherwise

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

Model Label Noise

THE UNIVERSITY OF SYDNEY

(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):

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

(3) Instance- and Label-Dependent Noise (ILN):

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

Random Classification Noise (RCN)

THE UNIVERSITY OF SYDNEY

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 1. 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.

Class-dependent Label Noise

THE UNIVERSITY OF SYDNEY

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.

Class-dependent Label Noise

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

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

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

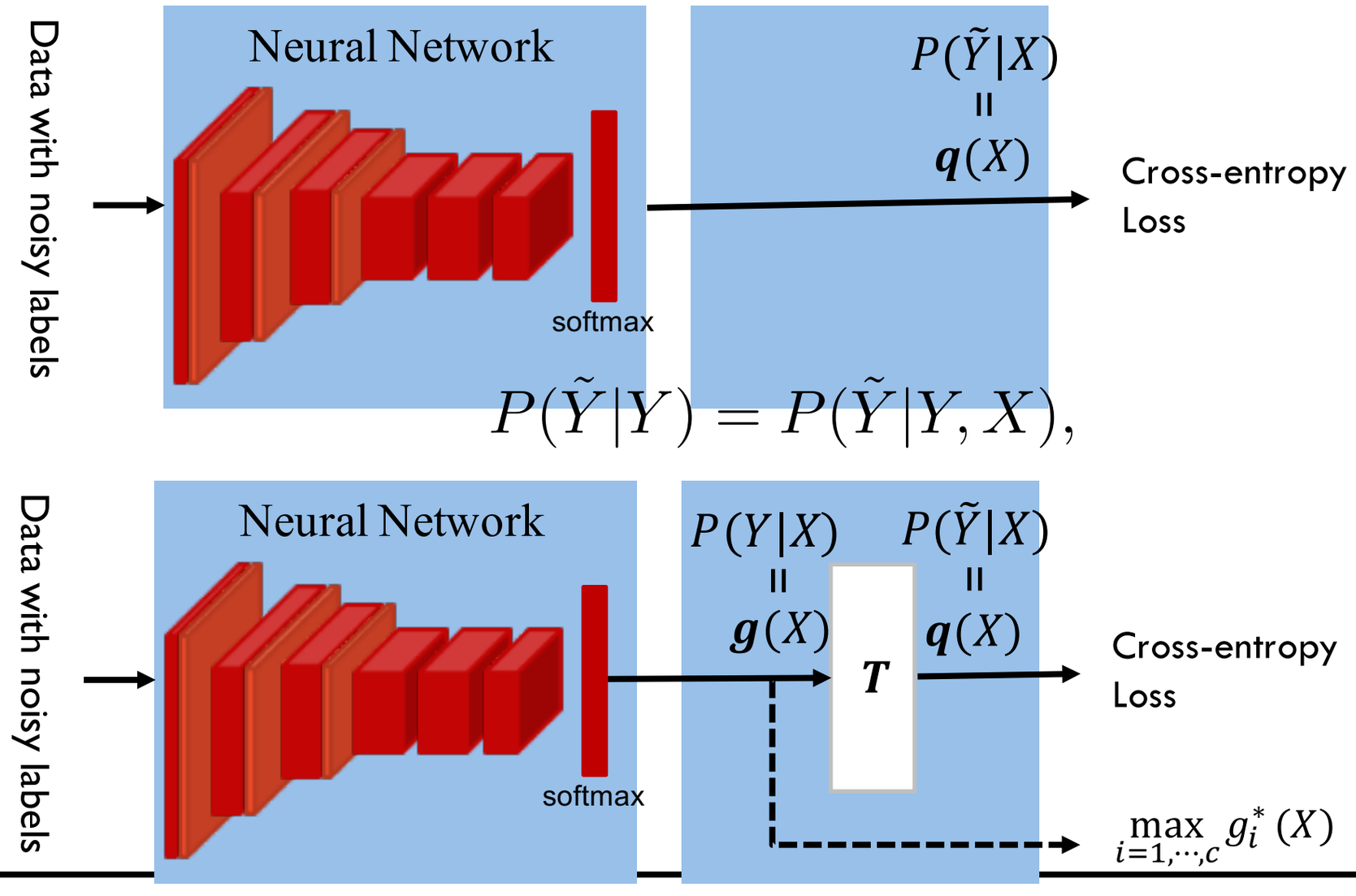
Backward

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

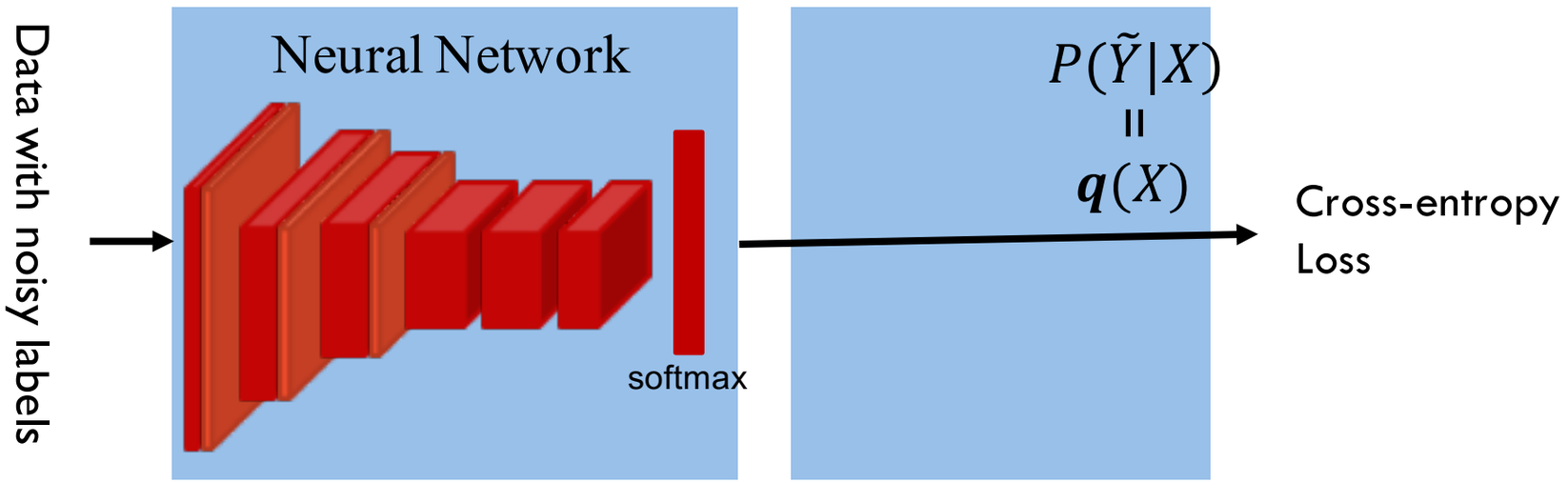
Forward learning:

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



Learning with noisy labels

Backward learning:



We have that $P(Y|X) = \boldsymbol{g}(X) = T^{-1}\boldsymbol{q}(X)$

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

THE UNIVERSITY OF SYDNEY

Estimate the transition matrix

Anchor Point Assumption

An instance x is call 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 ρ_{+1} and ρ_{-1}.

Anchor Point Assumption

THE UNIVERSITY OF SYDNEY

Noise rate estimation

$$P\left(\tilde{Y}=-1\middle|X=\boldsymbol{x}_{+1}\right)=(1-\rho_{+1}-\rho_{-1})P(Y=-1|X=\boldsymbol{x}_{+1})+\rho_{+1}$$

$$P\left(\tilde{Y}=+1\middle|X=\boldsymbol{x}_{-1}\right)=(1-\rho_{+1}-\rho_{-1})P(Y=+1|X=\boldsymbol{x}_{-1})+\rho_{-1}$$

0

$$P\left(\tilde{Y}=-1\middle|X=\boldsymbol{x}_{+1}\right)=\rho_{+1} \qquad P\left(\tilde{Y}=-1\middle|X=\boldsymbol{x}\right)\geq\rho_{+1}$$

$$P\left(\tilde{Y}=+1\middle|X=\boldsymbol{x}_{-1}\right)=\rho_{-1} \qquad P\left(\tilde{Y}=+1\middle|X=\boldsymbol{x}\right)\geq\rho_{-1}$$

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

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.

Estimate the Transition Matrix

How to learn T? Recall that

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

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

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

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., $P(Y = 1|X = x)$.

THE UNIVERSITY OF SYDNEY

Reinforcement Learning

Supervised Learning and RL

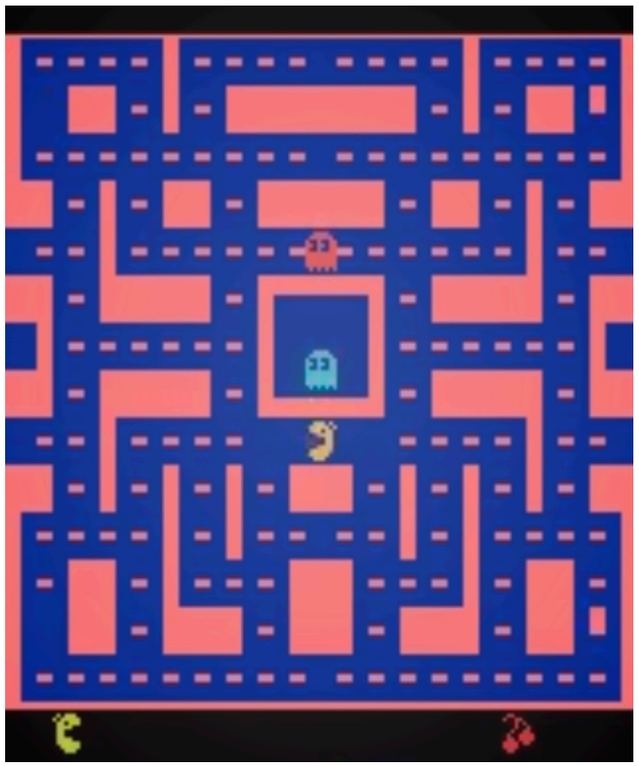
In supervised learning

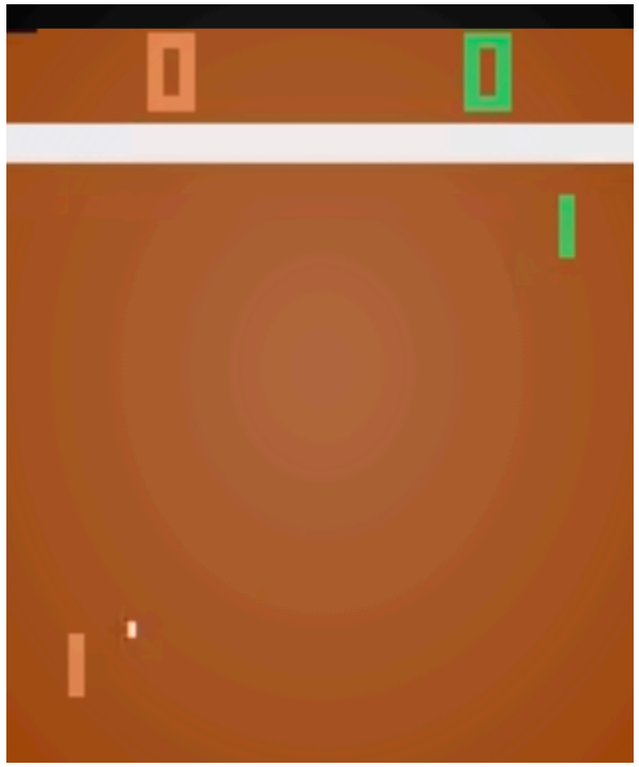
- We have training data: $S = \{x_i, y_i\}_{i=0}^{k}$,Where $x_i \in \mathcal{X}$ is features $y_i \in \mathcal{Y}$ is a label.
- Goal: Learn a function $f_S : \mathcal{X} \to \mathcal{Y}$ from the labeled dataset S.

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





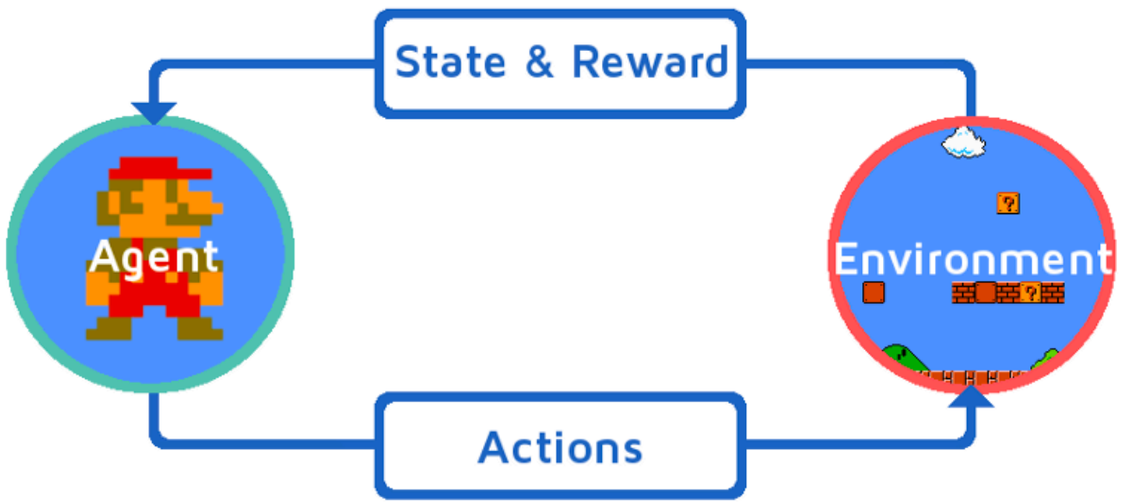


Figure: Atari Games on openAI gym and mario game.(https://gym.openai.com/envs/#atari)

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.
- π 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)

Given current state is s_0, we want to find the optimal policy (best strategy) π^* 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

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

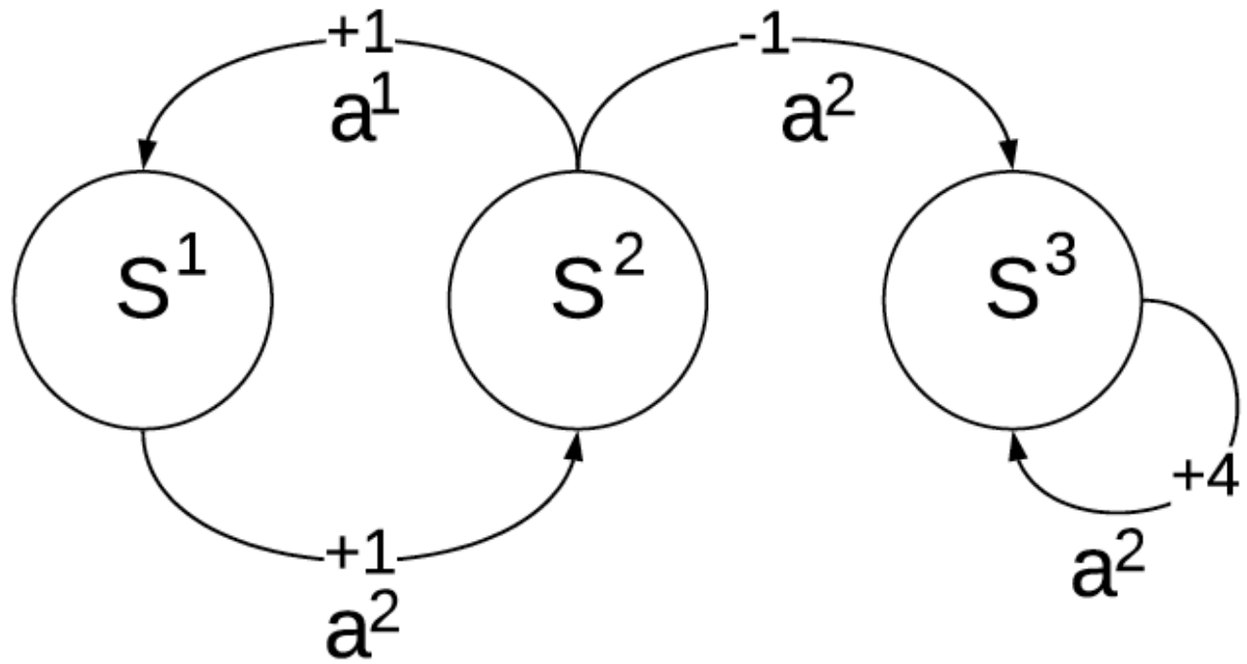
and a discounted factor

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

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,

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

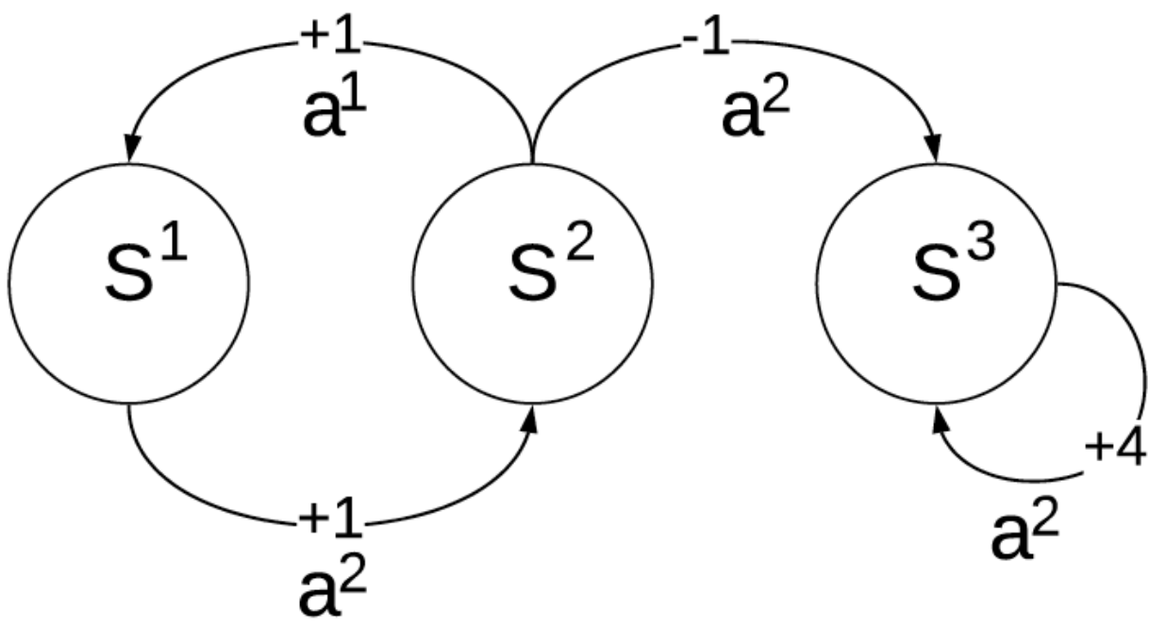
A Toy RL problem

THE UNIVERSITY OF SYDNEY

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

For all $t \leq 0$, we have

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



A toy RL problem

THE UNIVERSITY OF SYDNEY

Trajectories produced by π^1 and π^2 as follows,

π^1	t_0	t_1	t_2	t_3	t_4	$\ldots$
Sate	s^1	s^2	s^1	s^2	s^1	$\ldots$
Action	a^2	a^1	a^2	a^1	a^2	$\ldots$
Reward	$+1$	$+1$	$+1$	$+1$	$+1$	$\ldots$

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

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

Q-value function

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

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

with

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

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

and γ is the discounted factor.

Optimal Q-value function

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

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

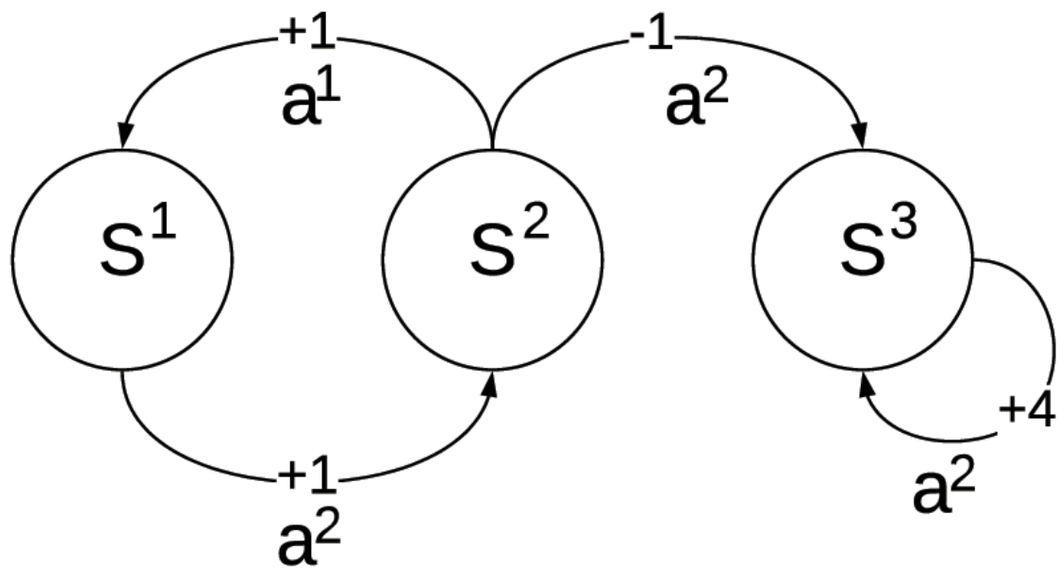
with

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

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 ?



Optimal Q-value function (Example)

THE UNIVERSITY OF SYDNEY

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

$$Q^*(s^2, a^1) = 1 + \gamma 1 - \gamma^2 1 + \gamma^3 4 + \gamma^4 4 \ldots \gamma^\infty 4$$

$$= 1 + \gamma 1 - \gamma^2 1 + \sum_{t=3}^{\infty} \gamma^t 4,$$

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

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 the 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 η 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 [1]).

[1] Proofs: http://users.isr.ist.utl.pt/~mtjspaan/readingGroup/ProofQlearning.pdf).

Store Q-values

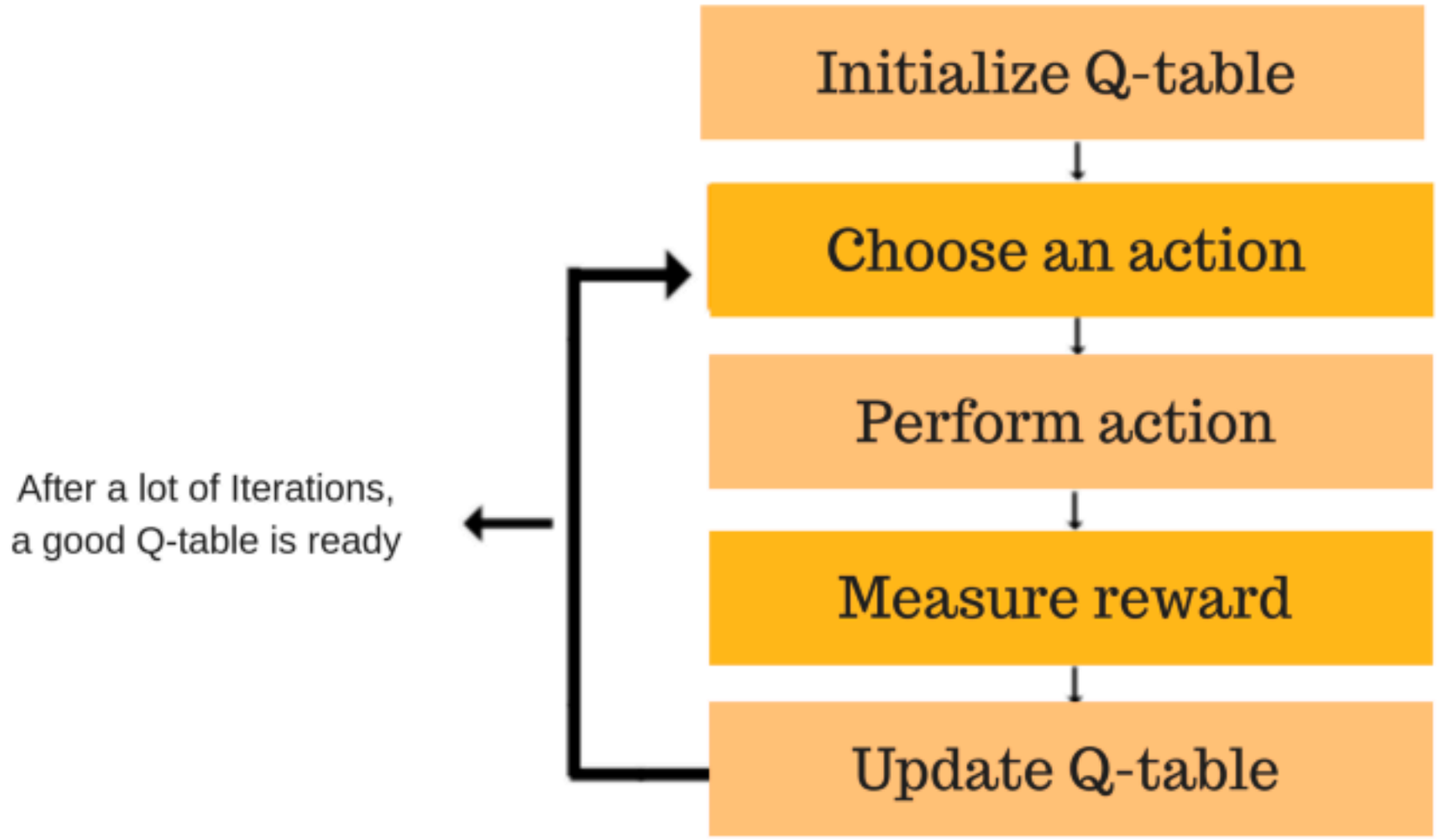
To update and track the changes of Q-values for state-action pairs, we need to store Q-values in somewhere. We could use a lookup table to store Q-values, e.g.,

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

Table: A Q Table at i-th iteration

Each entry in this table is the estimated Q-value for a state-acton 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'

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

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

  end

end

Sarsa

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

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

    $s \leftarrow s'$

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

  end

end

Off-policy and On-policy Learning (Difference between Q-Learning and Sarsa)

THE UNIVERSITY OF SYDNEY

- 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)

THE UNIVERSITY OF SYDNEY

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.

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.

THE UNIVERSITY OF SYDNEY

Deep Q-learning Network (DQN)

Deep Q-learning Network

THE UNIVERSITY OF SYDNEY

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 neutral 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 $Q_w(s, a)$, not to $Q_w(s', a')$

DQN with Experience Replay

Learning from consecutive training examples (state and actions) is problematic, because there may exists 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 $(r + \gamma \max_{a'} Q_w(s', a') - Q_w(s, a))^2$.

DQN with Prioritised Experience Replay

THE UNIVERSITY OF SYDNEY

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

THE UNIVERSITY OF SYDNEY

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 $a = \arg\max_a Q_w(s_{i-1}, a)$
  Observe s_t, r_t and γ_t
  Store the transition $(s_{t-1}, a_{t-1}, r_t, \gamma_t, s_t)$ in B with maximal priority
  if $t \equiv 0 \mod K$ *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 π_θ , where, θ is the parameter of the network.

Policy Gradients

The expected reward of a policy θ is:

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

and we want to find the optimal policy (parameters)

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

Calculate Gradient on θ

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:

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

Calculate Gradient on θ

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 θ

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

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

by summation (Monte Carlo methods)

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

No! The integration is not an expected value! Because non of $r(\tau)$ and $\nabla_\theta p(\tau;\theta)$ are distributions!Then we can not approximate it by empirical mean!

Calculate Gradient on θ (Log Derivative Trick)

$$\begin{aligned}
\nabla_\theta J(\theta) &= \int_{\tau \sim p(\tau;\theta)} r(\tau) \nabla_\theta p(\tau;\theta) d\tau, \\
&= \int_{\tau \sim p(\tau;\theta)} r(\tau) p(\tau;\theta) \frac{\nabla_\theta p(\tau;\theta)}{p(\tau;\theta)} d\tau, \\
&= \int_{\tau \sim p(\tau;\theta)} r(\tau) p(\tau;\theta) \nabla_\theta \log p(\tau;\theta) d\tau, \\
&= \mathbb{E}_{\tau \sim p(\tau;\theta)} \left[r(\tau) \nabla_\theta \log p(\tau;\theta) \right].
\end{aligned}$$

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 θ

Assume the Markov condition holds, i.e.,

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

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,

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

Finally, we have a gradient estimator:

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

Policy Gradient

Algorithm 4 Policy Gradient (simplified)

Initialize θ arbitrarily

for *each episode* $(s_0, a_0, r_0, \ldots, s_T, a_T, r_T) \sim \pi_\theta$ **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