THE UNIVERSITY OF SYDNEY

Advanced Machine Learning

(COMP 5328)

Week 11 Tutorial: Reinforcement Learning

Anjin Liu
anjin.liu@sydney.edu.au

Tutorial Contents

THE UNIVERSITY OF SYDNEY

- Review (20min):
  - Lecture 9: Reinforcement Learning
- Tutorial exercise & QA (40min):

Key points

- Introduction of Reinforcement Learning
- Q-Value, Q-Table
- Q-Learning, SARSA
- Off-policy and On-policy Learning
- DQN
- Policy Gradients

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.

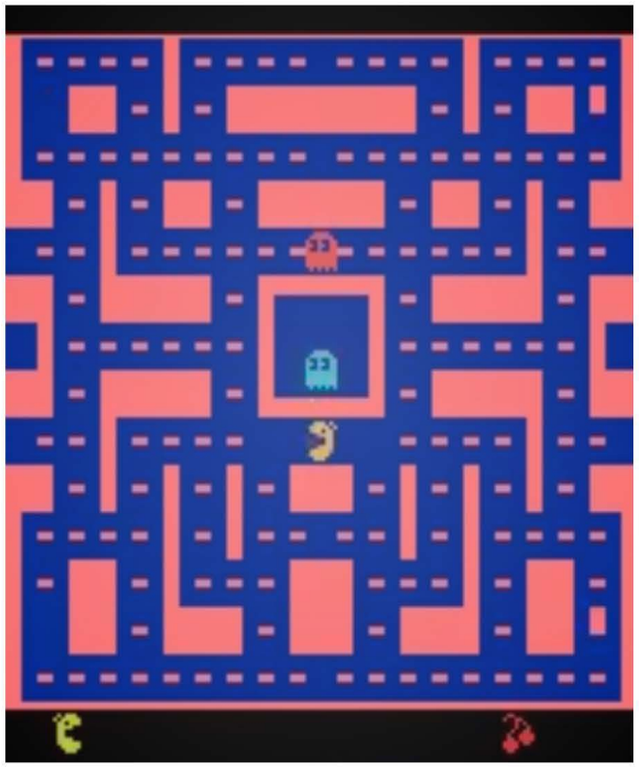
Decision Making

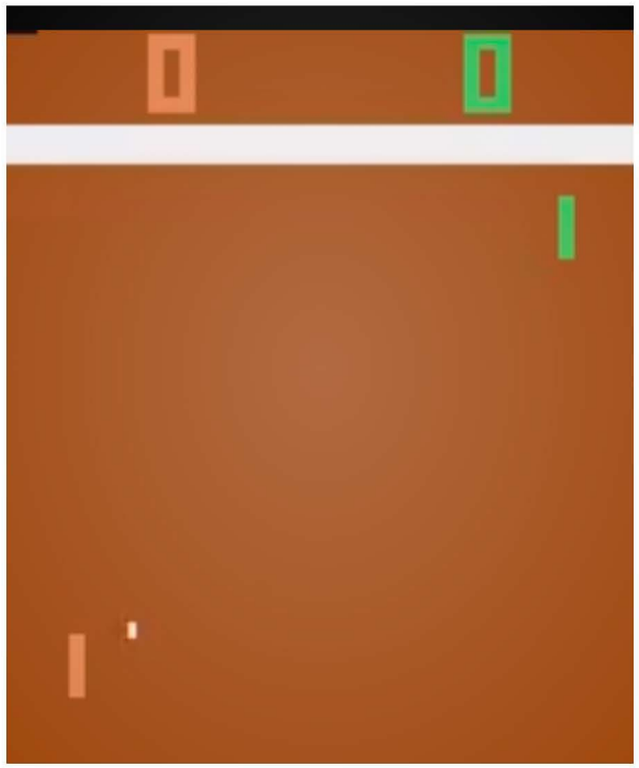
In reinforcement learning

- 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.

A Sequence of Decision Making

Introduction of RL





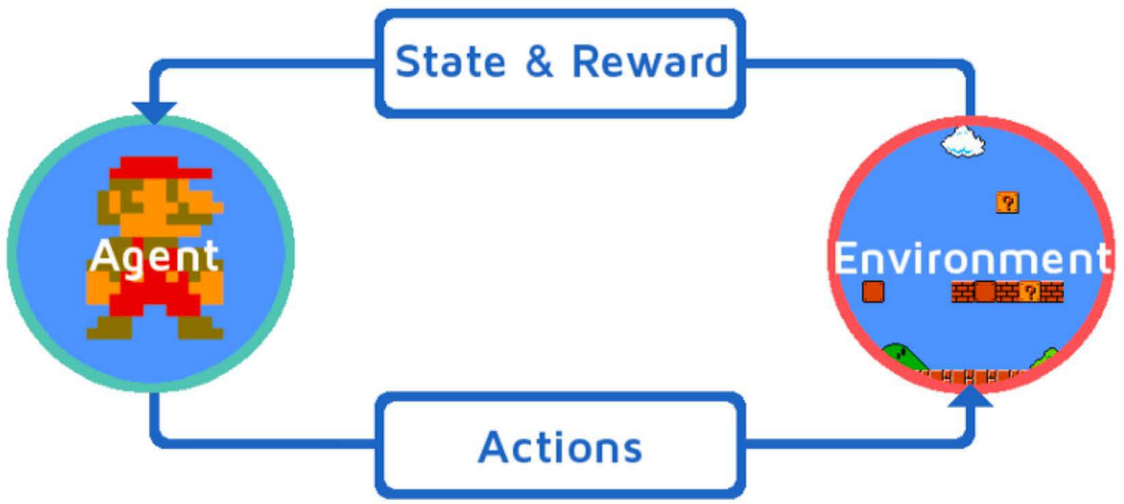


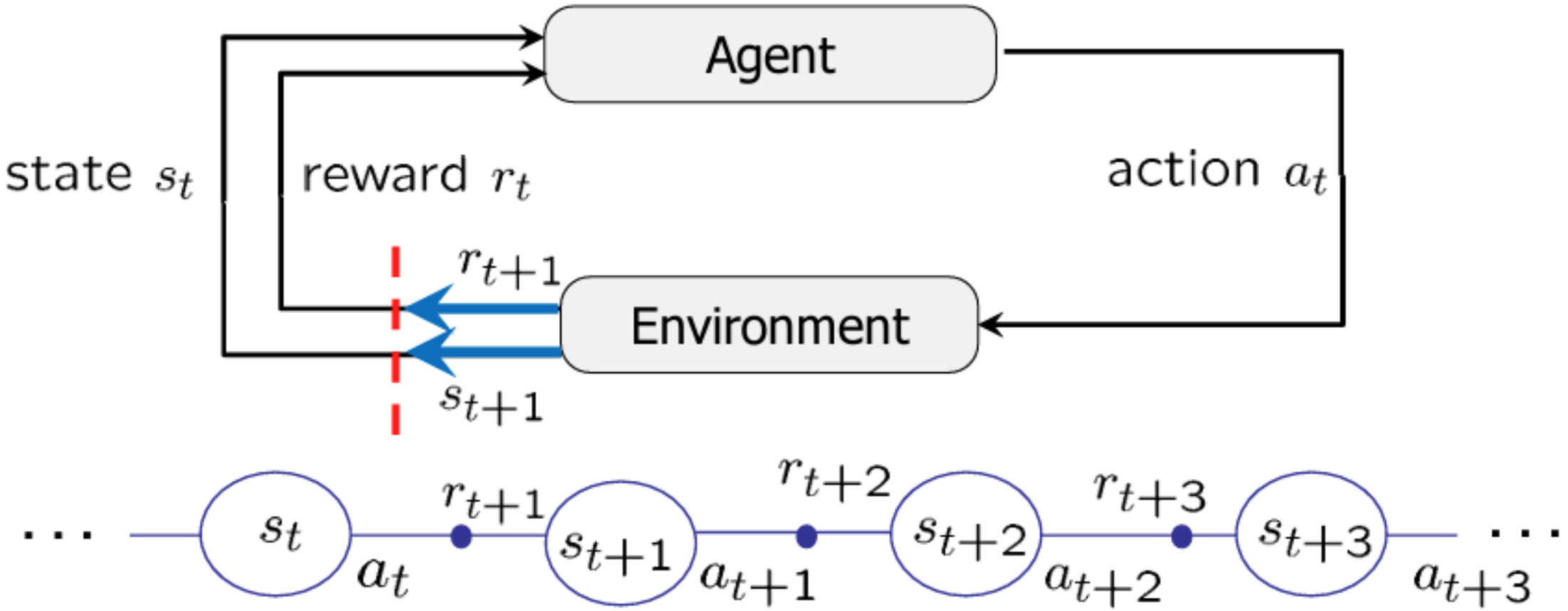
Figure: Atari Games on openAI gym and mario game.(https://gym.openai.com/envs/#atari)

RL Framework (notation)

THE UNIVERSITY OF SYDNEY

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



Goal at $t+1 = \max(r_{t+1} + r_{t+2} \ \ldots + r_T) = \max(r_{t+1}) + Goal(t+2)$

$= \max(r_{t+1}) + \max(r_{t+2}) + Goal(t+3)$

$= \max(r_{t+1}) + \max(r_{t+2}) + \max(r_{t+3}) + \cdots$

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) 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

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

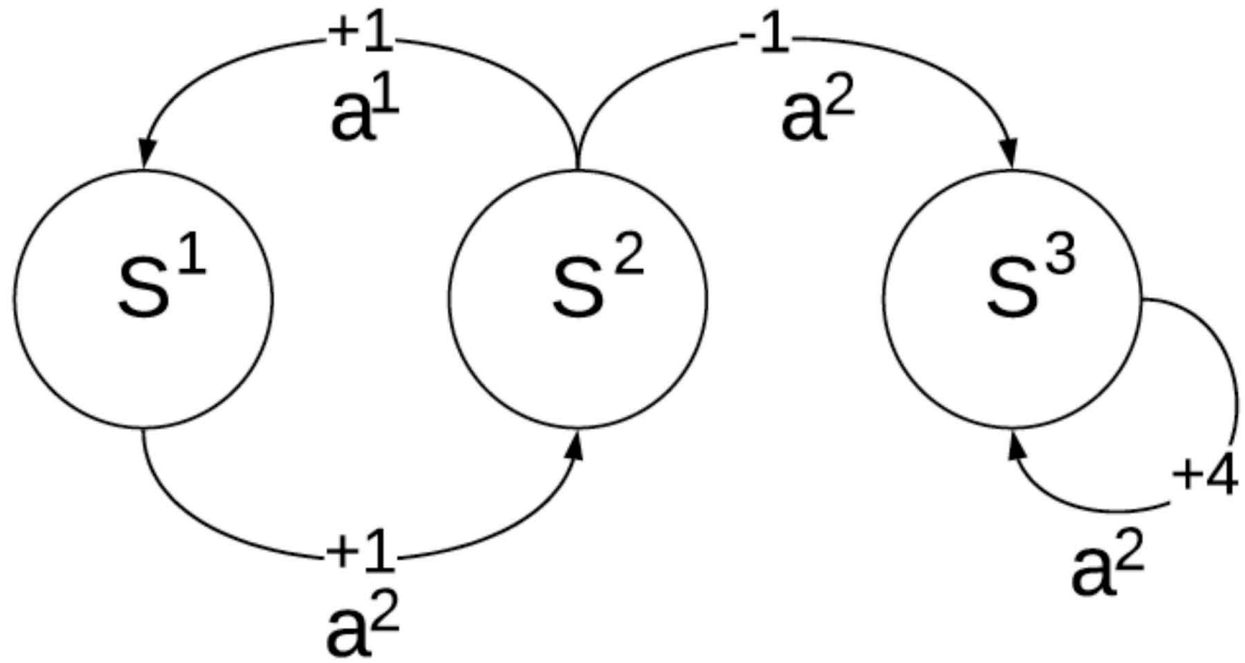
and a discounted factor

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

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,

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

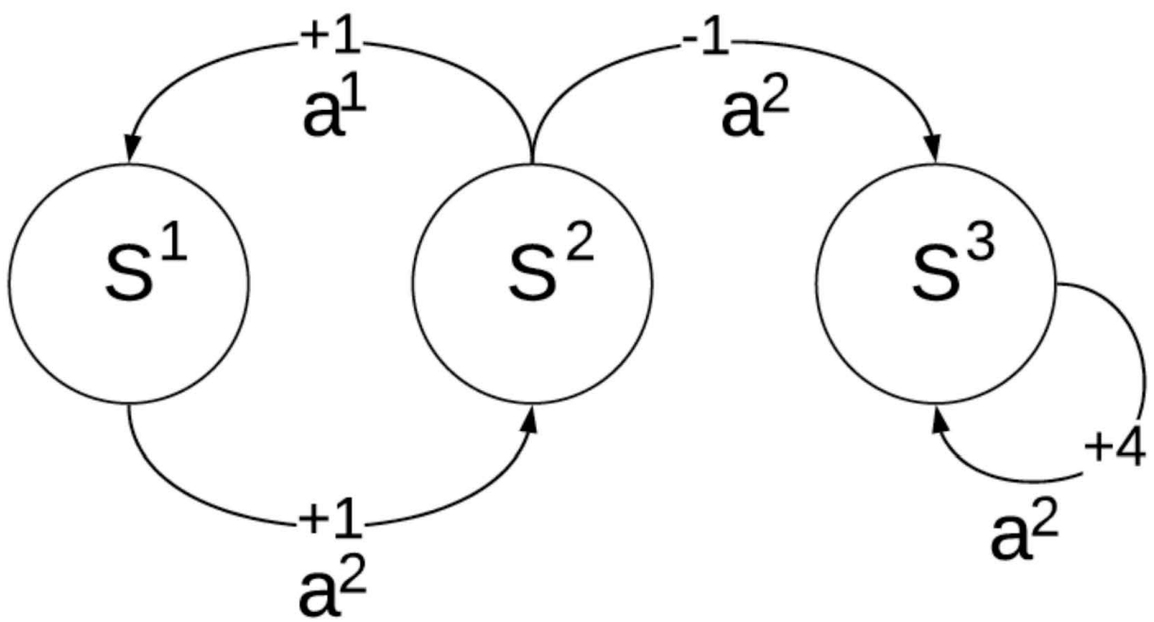
A Toy RL problem

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 $Q : S \times A \rightarrow \mathbb{R}$ 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 UNIVERSITY OF SYDNEY

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

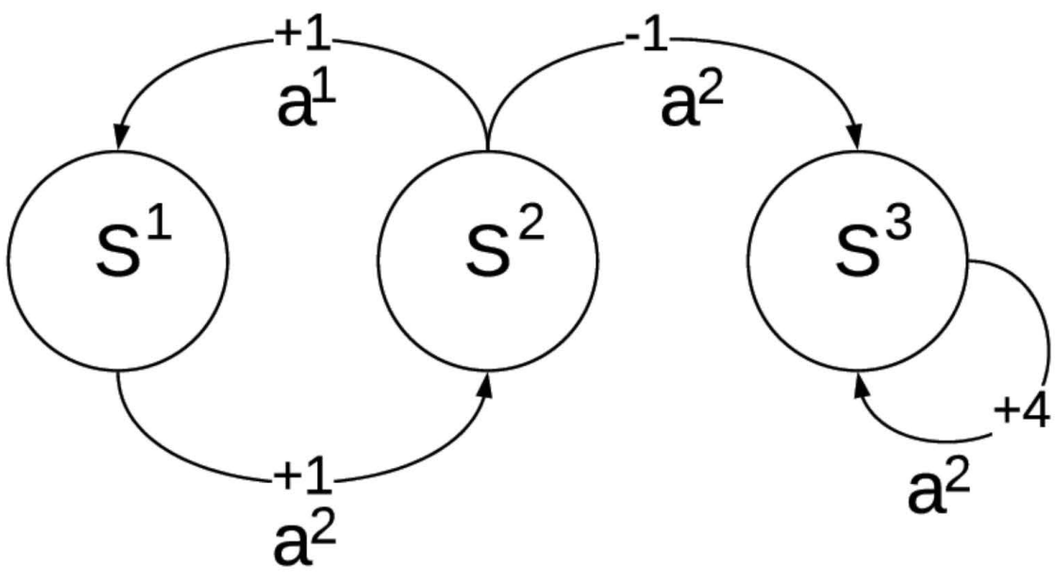
$$a_t \sim \pi(A_t | S_t = s_t),\ s_{t+1} \sim P(S_{t+1} | A_t = a_t, S_t = s_t)$$

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

Optimal Q-value function (Example)

THE UNIVERSITY OF SYDNEY

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:

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

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

Goal at $t+1 = \max(r_{t+1} + r_{t+2} \ \ldots + r_T) = \max(r_{t+1}) + Goal(t+2)$
$= \max(r_{t+1}) + \max(r_{t+2}) + Goal(t+3)$
$= \max(r_{t+1}) + \max(r_{t+2}) + \max(r_{t+3}) + \cdots$

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

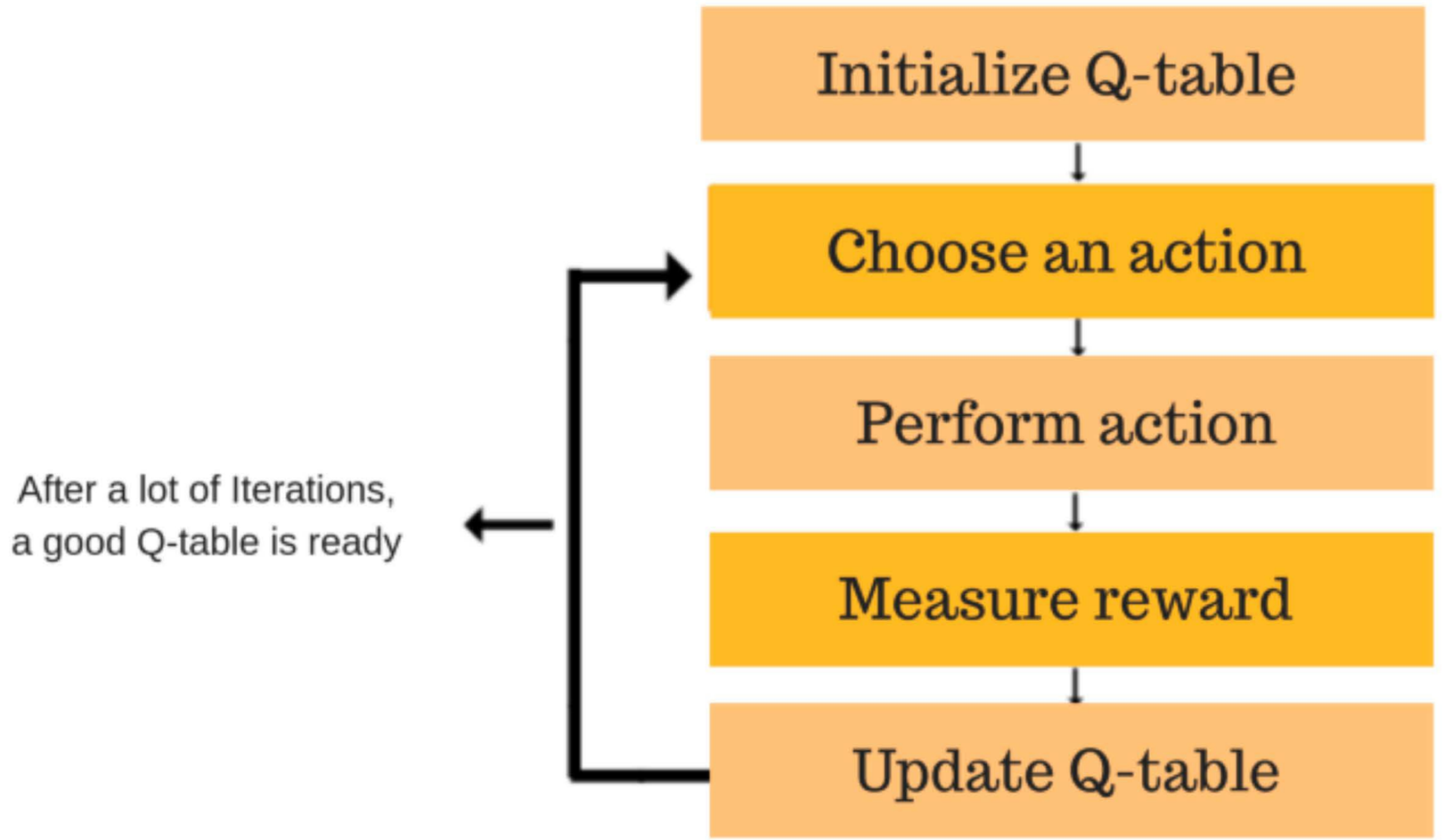
To update and track the changes of Q-values for state-action pairs, we need to store Q-values in somewhere. We could use a lookup table to store Q-values, e.g.,

Q_i	s^1	s^2	s^3	...
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-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

$$Q(s,a) = Q(s,a) + \alpha(r + \gamma Q(s',a') - Q(s,a))$$ (SARSA)

$$Q(s,a) = Q(s,a) + \alpha(r + \gamma \mathbf{max}_{a'} Q(s',a') - Q(s,a))$$ (Q-learning)

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 Gradient can be understood as "**directly learning a function that selects actions**," rather than "**learning action values first and then choosing the best one**.“

It optimizes the expected reward through gradient ascent, instead of maximizing the Q-value function.

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

Key points

- Introduction of Reinforcement Learning
- Q-Value, Q-Table
- Q-Learning, SARSA
- Off-policy and On-policy Learning
- DQN
- Policy Gradients