Reinforcement Learning

Tianchu.Zhao@uts.edu.au

What is reinforcement learning

- Reinforcement learning is a type of algorithm that let the computer to start with 0 knowledge and through repetitive trying, learn from the mistake to find the pattern in order to learn how to achieve objective

- Examples: Alpha Go, Atari games, Mario series

How does the computer learn

- Virtual Teacher
    - Teacher only gives them feedback if they do it right/wrong but not the steps or how to improve
- How does the machine study
    - They remember the behaviours that corresponds to low/high score and the next time they do the test they will perform the behaviour that corresponds to high score for a better result

Compare to supervised learning

- Supervised learning
  - we have data points that contain features and labels
- Reinforcement learning
  - we don’t have any data points
  - through repetitive trial and error, we obtain data points and the label

Different types of reinforcement learning

- Values learning
  - Q Learning
  - Sarsa
  - Deep Q Network
- Policy Learning
  - Policy Gradients

Model-free vs Model based

- Model Free
- the model does not try to understand the environment (obtain the feedback of environment as it is, without further analysing)
- Model based
- able to predict the future reward and determine next action base on future reward (without actually perform the move to obtain the reward)

Monte-Carlo vs temporal difference update

- Monte-Carlo update
- we only update our action policy after a game has finished
- Temporal difference update
- update at every step in the game (no need to wait till the game finished)

On-Policy vs Off-Policy learning

- On-Policy
- machine must playing the game itself
- Off-Policy
- machine can learn the behaviour from the past through observing to calibrate its action policy

Q-learning

- action rules
- When we do things, there are result that associated with it
- eg, when we are at University, if we do assignment, we pass the subject, but if we don't the assignment, we fail
- We have a Q table that defines the action, reward and state

	a1	a2
s1	-2	1
s2	-4	2

How Q learning updates

- If we are at state 1, because action 2 has higher reward than action 1, we pick action 2 to go to state 2 (we perform Q(s1, a2)
- then we imagine we do something on s3 without actually doing it
- from the table we know that Q(s2, a2) is bigger than Q(s2, a1)
- we use Q(s2, a2) times a decay value gamma (eg 0.9) plus a reward R when we perform action2 (R is the previous reward, at the moment is 0) to obtain approximate value
- this becomes the real world value of Q(s1, a2)
- then we calculate the difference between real world value and the approximate value
- The new approximate Q value then becomes this difference times by alpha + the previous Q value
- (this is off policy because we don’t actually perform the action on s2)

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

Parameters

- e-greedy (Epsilon greedy)
- if epsilon = 0.9
- then 90% of the time we will use Q table to find the best behaviour
- and 10% of the time to choose random behaviour

- alpha(a) = learning rate

- gamma(γ) = decay of future reward

gamma in Q learning

- gamma calculates how much the previous value is retained in the current computation
- $Q(s1) = r2 + \gamma\, Q(s2) = r2 + \gamma(r3 + \gamma Q(s3)) = r2 + \gamma(r3 + \gamma(r4 + \gamma Q(s4)))=..$
- $Q(s1) = r2 + \gamma\, r3 + \gamma^2\, r4 + \gamma^3\, r5 + ...$

- When $\gamma = 1$
- $Q(s1) = r2 + r3 + r4 + r5$
- When $\gamma = (0\text{-}1)$
- $Q(s1) = r2 + \gamma\, r3 + \gamma^2\, r4 + \gamma^3\, r5 + ...$
- When $\gamma = 0$
- $Q(s1) = r2$

Sarsa (State-action-reward-state-action)

- Similar to Q learning
- We have a Q table to help us pick the decision that has the greatest reward
- They (Q learning and Sarsa) differs at how they updates the Q value
- Sarsa is on-policy
- It will perform the action to obtain the reward instead of observing the reward

Initialize $Q(s, a)$ arbitrarily
Repeat (for each episode):
    Initialize s
    Choose a from s using policy derived from Q (e.g., ε-greedy)
    Repeat (for each step of episode):
        Take action a, observe r, s'
        Choose a' from s' using policy derived from Q (e.g., ε-greedy)
        $Q(s, a) \leftarrow Q(s, a) + \alpha\big[r + \gamma Q(s', a') - Q(s, a)\big]$
        $s \leftarrow s'$; $a \leftarrow a'$;
    until s is terminal

Sarsa lambda optimisation

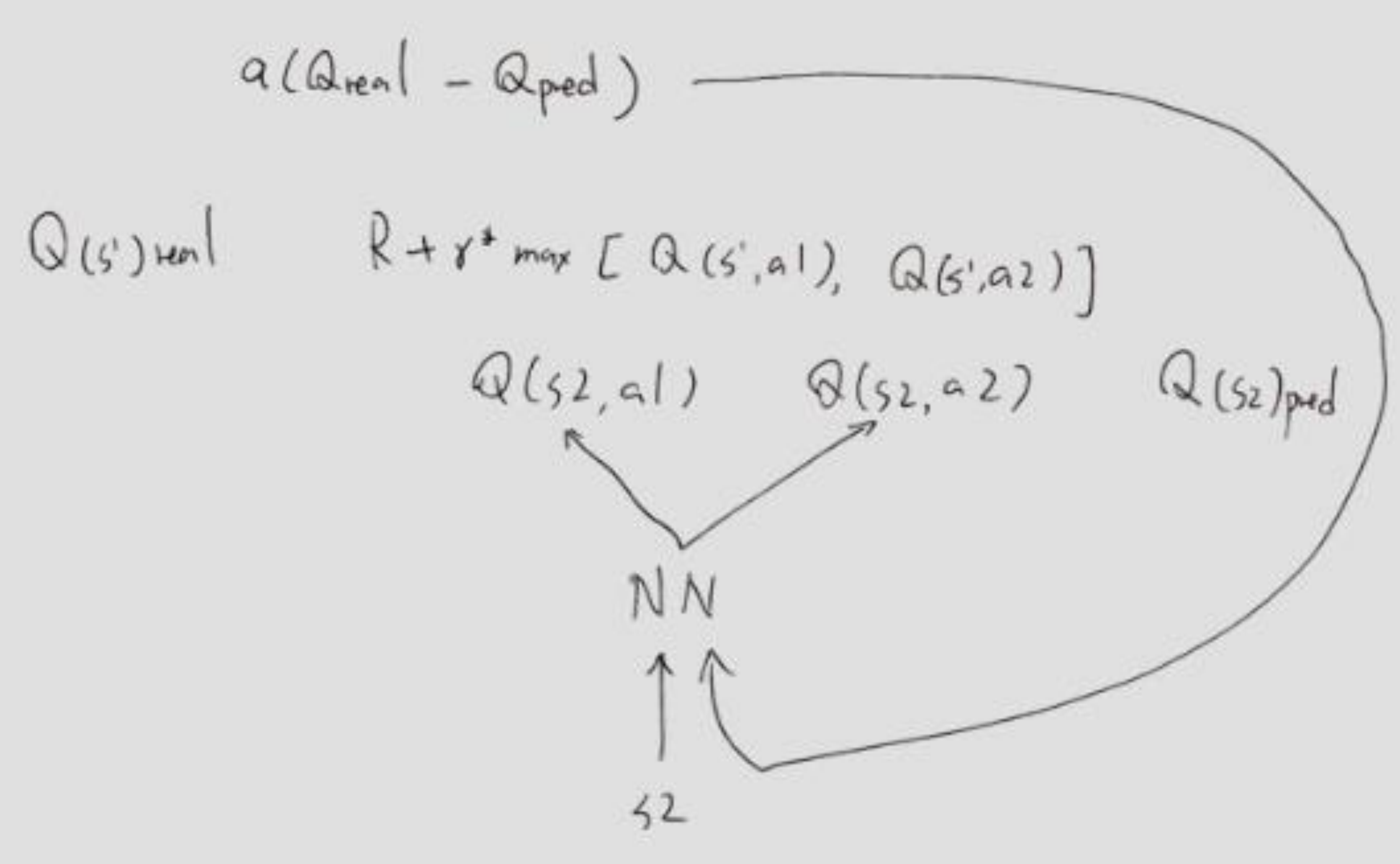
- previously we update Q table every step
- but it maybe more optimum to update after all steps (or at some intermediate steps)
- lambda helps to perform above
- Sarsa(lambda)
- Sarsa(0), we update at every step
- Sarsa(0-1), the cloest step to final reward will have more weight when update (decay)
- Sarsa(1), we update after episode

Initialize $Q(s, a)$ arbitrarily, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
Repeat (for each episode):
    $E(s, a) = 0$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
    Initialize S, A
    Repeat (for each step of episode):
        Take action A, observe R, S'
        Choose A' from S' using policy derived from Q (e.g., ε-greedy)
        $\delta \leftarrow R + \gamma Q(S', A') - Q(S, A)$
        $E(S, A) \leftarrow E(S, A) + 1$
        For all $s \in \mathcal{S}, a \in \mathcal{A}(s)$:
            $Q(s, a) \leftarrow Q(s, a) + \alpha \delta E(s, a)$
            $E(s, a) \leftarrow \gamma \lambda E(s, a)$
        $S \leftarrow S'$; $A \leftarrow A'$
    until S is terminal

Deep Q Learning (DQN)

- previously are all traditional method
- DQN is neural network based

- If we have a complex problem (GO)
- we won’t have a large enough table (Q table) to store all possible moves
- we use state and action as a input to the neural network and outputs the Q value
- or (alternatively) we use state as input, and the output is the best possible action
- DQN is off policy


a(Q_real - Q_pred)
Q(s')_real
R + γ* max [Q(s',a1), Q(s',a2)]
Q(s2,a1)
Q(s2,a2)
Q(s2)_pred
NN
s2

Algorithm 1: deep Q-learning with experience replay.

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights θ

Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$

For episode = 1, M **do**

  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$

  For $t = 1$,T **do**

    With probability ε select a random action a_t

    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$

    Execute action a_t in emulator and observe reward r_t and image x_{t+1}

    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$

    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D

    Sample random minibatch of transitions $\left(\phi_j, a_j, r_j, \phi_{j+1}\right)$ from D

    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}\left(\phi_{j+1}, a'; \theta^{-}\right) & \text{otherwise} \end{cases}$

    Perform a gradient descent step on $\left(y_j - Q\left(\phi_j, a_j; \theta\right)\right)^2$ with respect to the network parameters θ

    Every C steps reset $\hat{Q} = Q$

  End For

End For

Policy Gradient

- All the previous algorithms are reward oriented (we get a reward, then determine the action base on the reward)
- Policy gradient doesn’t provide this value-based action
- It outputs action straight away
- This allows picking an action from a continuous space instead of discrete space like Q learning, this allows faster experimentation

function REINFORCE

Initialise θ arbitrarily

for each episode $\{s_1, a_1, r_2, ..., s_{T-1}, a_{T-1}, r_T\} \sim \pi_\theta$ **do**

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

$\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) v_t$

end for

end for

return θ

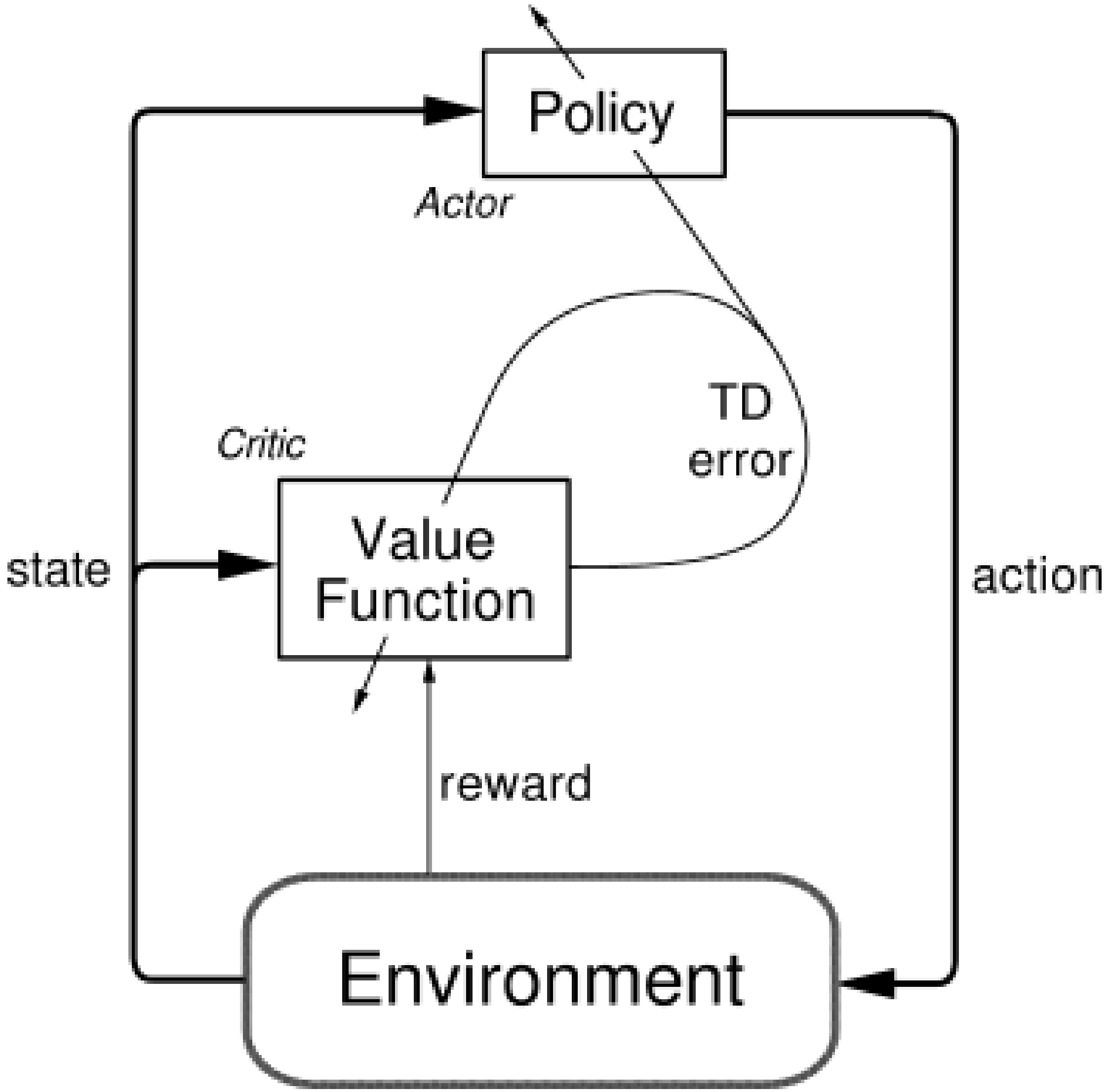
end function

Actor Critic (AC)

- Combines policy gradient and q learning
- we can see actor as policy gradient that is able to perform action straight away which q learning is unable to do
- we can see critic as q learning, we can update every step in a episode instead of waiting for the full episode to finish

Table 1: A Template for Incremental AC Algorithms.

1:	**Input:**	
	• Randomized parameterized policy $\pi(\cdot\|\cdot;\boldsymbol{\theta})$,	
	• Value function feature vector $\boldsymbol{f}(s)$.	
2:	**Initialization:**	
	• Policy parameters $\boldsymbol{\theta} = \boldsymbol{\theta}_0$,	
	• Value function weight vector $\boldsymbol{v} = \boldsymbol{v}_0$,	
	• Step sizes $\alpha = \alpha_0, \quad \beta = \beta_0, \quad \xi = c\alpha_0$,	
	• Initial state s_0.	
3:	**for** $t = 0, 1, 2, \ldots$ **do**	
4:	**Execution:**	
	• Draw action $a_t \sim \pi(a_t\|s_t;\boldsymbol{\theta}_t)$,	
	• Observe next state $s_{t+1} \sim p(s_{t+1}\|s_t, a_t)$,	
	• Observe reward r_{t+1}.	
5:	**Average Reward Update:**	$\hat{J}_{t+1} = (1-\xi_t)\hat{J}_t + \xi_t r_{t+1}$
6:	**TD error:**	$\delta_t = r_{t+1} - \hat{J}_{t+1} + \boldsymbol{v}_t^\top \boldsymbol{f}(s_{t+1}) - \boldsymbol{v}_t^\top \boldsymbol{f}(s_t)$
7:	**Critic Update:**	algorithm specific (see the text)
8:	**Actor Update:**	algorithm specific (see the text)
9 :	**endfor**	
10:	**return** Policy and value-function parameters $\boldsymbol{\theta}, \boldsymbol{v}$	


Policy
Actor
TD
error
Critic
Value
Function
state
action
reward
Environment

Deep Deterministic Policy Gradient (DDPG)

- Improvement on Actor Critic
- Deep Deterministic Policy Gradient (DDPG)
- Basically DQN + Actor Critic
- with extra tuning1 (instead sample from policy gradient we have a deterministic action (no sampling, keep performing the same action), this speeds the learning)
- with extra tuning2 (both action and state has 2 neural networks (4 total), 1 pair for the approximate value, 1 pair for the real value

Asynchronous Advantage Actor-Critic (A3C)

- Interpret this as a distributed AC

(Bonus)
How to start a research (eg PhD, MPhil)

- Assume you have select topic

- Look for review paper on the topic (eg keywords: **review** of blabla, summary of blabla, blabla literature review, benchmark, **survey**)
- review/survey paper is usually a summary of the topic at a given period, usually paper like this comes out about every 3 years.

- To search paper
- To go UTS library -> search google scholar -> database -> click it, log in and save the link

- If you are new to the topic, it is better to go through the whole review paper to understand the background and the latest technique on the topic

- Then we also have theory/application paper (the reference paper from the review paper). These paper will have similar intro (you can skip if you have read a few papers).

- Usually people start by reading the abstract to see if the paper is helpful, and if it is they will go on to read the conclusion and interesting sections.

- A strong foundations is usually the pathway to great research idea.
- People usually start by replicating a research paper to fully understand the algorithm before proceed further

Grand Plan (besides the lecture material)

- (✓) Neural Network Foundation (update to ppt, tbd)
- (✓) Ensemble/Optimise your neural network
- (✓) Convolution Neural Network
- (✓) Principal Component Analysis / Big data
- (✓) Reinforcement Learning
- (☐) Recurrent Neural Network
- (☐) Generative Adversarial Network (+ unsupervised + symmetric nn)
- (☐) Natural Language Processing