

Deep Reinforcement Learning and its application to games

A/Prof Richard Yi Da Xu, Kelvin Deng Chen

`richardxu.com`

University of Technology Sydney (UTS)

September 4, 2019

- ▶ A video from Google DeepMind's Deep Q-learning playing Atari Breakout:
<https://www.youtube.com/watch?v=TmPfTpjtdgg>
- ▶ Mnih, Volodymyr, et al. "Playing atari with deep reinforcement learning." *arXiv preprint arXiv:1312.5602* (2013).
- ▶ code is also available
<https://github.com/kuz/DeepMind-Atari-Deep-Q-Learner>

N.B.

- ▶ Apologies for those have seen it before

significance of this demo shows it's possible to use Neural Network to learn how to play a game, based on:

- ▶ sequences of screen images
- ▶ scores the game receives
- ▶ goal is to learn the best policy for **actions** to take

Surely you **don't need a menu** to learn how to play Atari. i.e., it's **model-free**!

Forget about the Neural network for a second, how is Reinforcement Learning (RL) different to conventional supervised learning?

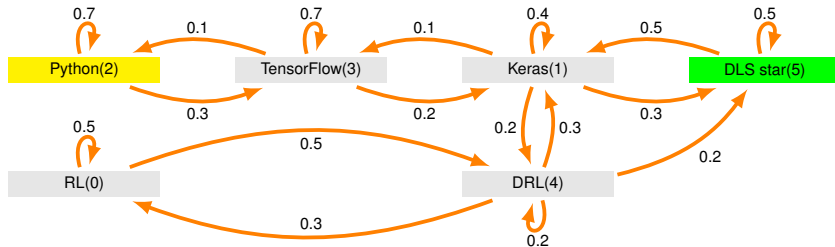
- ▶ No data label like supervised learning, i.e., no “*best-action-for-that-screen*” label
- ▶ only **reward** signal
- ▶ feedback in **delayed**, not instantaneous
- ▶ data are not i.i.d., (consecutive frames are similar)
- ▶ agent's actions affects the subsequent data it receives.

Let's get started with some RL background.

another way to look at it:

- ▶ RL uses training information that **evaluates the actions** taken rather than **instructs by giving correct actions**.
- ▶ a need for active exploration: explicit trial-and-error search for good behavior.
- ▶ **purely evaluative feedback** indicates how good the action taken is, but not whether it is the best or the worst action possible.
- ▶ **purely instructive feedback** indicates correct action to take, independently of the action actually taken. [supervised learning](#)

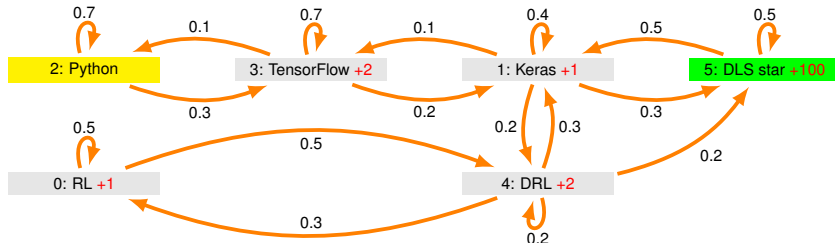
- ▶ **marketing**
customer's attributes s , marketing actions a , customer signs up r
- ▶ **drone control**
all available sensor data a , controls s , not crashing r
- ▶ **chatbot**
conversations to-date s , things that a robot will say a , customer satisfaction r



- ▶ one may start from **python** and generate sequences with transition probabilities to end up in **DLS star**. examples:
 - ▶ Python, Python, Python, TensorFlow, Keras, DLS star
 - ▶ Python, Python, Python, TensorFlow, TensorFlow, Keras, DRL, DRL RL, DLS star
 - ▶ Python, Python, TensorFlow, TensorFlow, Keras, DRL, DLS star
 - ▶ The question is: how we may able to measure “how good” each path? ...

Markov Reward Process

Let's add some rewards to being at each of the state:



What we care is the **total return** G_t : sum of **discounted** reward from time-step t

$$G_t = R_{t+1} + \gamma R_{t+2} + \dots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \quad \text{where } \gamma \in [0, 1]$$

note that G_t is a random variable

exercise what happens when $\gamma = 0$ and $\gamma = 1$

Markov Random Process: Bellman Equation (new)

- ▶ **state value function** $V(s)$ of MRP is expected total return starting from state s

$$\begin{aligned} V(s) &= \mathbb{E}_{s_{t+1}, s_{t+2}, \dots, r_{t+1}, r_{t+2}, \dots} [G_t | s_t = s] \\ &= \mathbb{E}_{s_{t+1}, s_{t+2}, \dots, r_{t+1}, r_{t+2}, \dots} [R_{t+1} + \underbrace{\gamma (R_{t+2} + \gamma R_{t+3} + \dots)}_{G_{t+1}}] \end{aligned}$$

- ▶ $\mathbb{E}[\cdot]$ needs the integrate over $(s_1, s_2, \dots \in \mathcal{S}, r_1, r_2, \dots \in \mathcal{R})$:
- ▶ s_1, s_2, \dots and r_1, r_2, \dots are generated in the following fashion:

$$s_0 \rightarrow (s_1, r_1) \quad s_1 \rightarrow (s_2, r_2) \dots$$

- ▶ for clarity, we let $s_t \rightarrow s_0$ and $s_{t+k} \rightarrow s_k$:

Markov Random Process: Bellman Equation (new)

- suppose we have a **universal state value function** $V(\cdot)$:

$$V(\cdot) = \sum_{s_0} \Pr(s_0) \sum_{s_1, r_1} \Pr(s_1, r_1 | s_0) \sum_{s_2, r_2} \Pr(s_2, r_2 | s_1) \sum_{s_3, r_3} \dots [r_1 + \gamma(r_2 + \gamma r_3 + \dots)]$$

- however, we usually specify value of $v_\pi(s_0)$ to evaluate:

$$\begin{aligned} V(s_0) &= \sum_{s_1, r_1} \Pr(s_1, r_1 | s_0) \underbrace{\left(r_1 + \gamma \sum_{s_2, r_2} \Pr(s_2, r_2 | s_1) \sum_{s_3, r_3} \dots [r_2 + \gamma(r_3 + \gamma r_4 + \dots)] \right)}_{V(s_1) \triangleq \mathbb{E}[G_{t+1} | s_1]} \\ &\quad \underbrace{\hspace{10em}}_{V(s_0) \triangleq \mathbb{E}[G_t | s_0]} \\ &= \mathbb{E}_{s_1, r_1} [r_1 + \gamma V(s_1) | s_0] \\ &= \mathbb{E}_{s_1} [R_1 + \gamma V(s_1) | s_0] \text{ if } R_1 \text{ is deterministic} \end{aligned}$$

Markov Random Process: Bellman Equation (new)

$$V(s_0) = \mathbb{E}_{s_1} [R_1 + \gamma V(s_1) | s_0]$$

- ▶ **Bellman equations:** value of the current state, $v(s)$ breaks up into (1) **immediate** and (2) **future** rewards.
- ▶ state value function $V(s)$ is written in a consecutive time steps
- ▶ difficult to estimate: because $V(s)$ also depends on various other $V(s')$ which occur at different times

Bellman Equation in matrix form

- ▶ to simplify, making R_t deterministic

$$V(s_0) = \mathbb{E}_{s_1} [R_1 + \gamma V(s_1) | s_0]$$

- ▶ say $s \in \{1, \dots, n\}$:

$$\underbrace{V(s_0 = 1)}_{v(1)} = \mathbb{E}_{s_1} \left[\underbrace{R_1(s_0 = 1)}_{R_1} + \gamma V(s_1) | s_0 = 1 \right]$$

$$V(s_0 = 2) = \mathbb{E}_{s_1} [R_1(s_0 = 2) + \gamma V(s_1) | s_0 = 2]$$

...

take the first line,

$$v(1) = \mathbb{E}_{s_1} [R_1 + \gamma V(s_1) | s_0 = 1]$$

$$= R_1 + \gamma \mathbb{E} [V(s_1) | s_0 = 1]$$

$$= R_1 + \gamma \left(\sum_{s_1=1}^n v(s_1) \Pr(1 \rightarrow s_1) \right)$$

$$= R_1 + \gamma \left(\sum_{j=1}^n v(j) \Pr(1 \rightarrow j) \right)$$

...

$$\Rightarrow v(n) = R_n + \gamma \left(\sum_{j=1}^n v(j) \Pr(k \rightarrow j) \right)$$

Bellman Equation in matrix form (2)

$$\begin{aligned}v(k) &= R_k + \gamma \left(\sum_{j=1}^n v(j) \Pr(k \rightarrow j) \right) \\&= R_k + \gamma \mathcal{P}_{k,:}^\top \mathbf{v} \\ \Rightarrow \mathbf{v} &= \mathbf{R} + \gamma \mathcal{P} \mathbf{v} \\ \Rightarrow \begin{bmatrix} v(1) \\ \vdots \\ v(n) \end{bmatrix} &= \begin{bmatrix} R_1 \\ \vdots \\ R_n \end{bmatrix} + \gamma \begin{bmatrix} \mathcal{P}_{1,1} & \dots & \mathcal{P}_{1,n} \\ \vdots & & \vdots \\ \mathcal{P}_{n,1} & \dots & \mathcal{P}_{n,n} \end{bmatrix} \begin{bmatrix} v(1) \\ \vdots \\ v(n) \end{bmatrix}\end{aligned}$$

the solution to **MRP** is straight forward:

$$\begin{aligned}\mathbf{v} &= \mathbf{R} + \gamma \mathcal{P} \mathbf{v} \\ (I - \gamma \mathcal{P}) \mathbf{v} &= \mathbf{R} \\ \mathbf{v} &= (I - \gamma \mathcal{P})^{-1} \mathbf{R}\end{aligned}$$

Markov Decision Process (MDP)

- ▶ now agent has **actions**
- ▶ concept of **policy** π : take a state s_t as input and decides an action a_t

$$\pi(a|s) = \Pr(A_t = a | S_t = s)$$

- ▶ a policy is time-invariant (or stationary) and stochastic
- ▶ next state for an agent, now also depends on its action taken:

$$\mathcal{P}_{s \rightarrow s'}^a = \Pr(S_1 = s' | S_0 = s, A_0 = a)$$

- ▶ multiple transition matrix \mathcal{P} each depends on the a taken
- ▶ once fixed π , MDP becomes MRP with transition probability $\mathcal{P}_{s \rightarrow s'}^\pi$:

$$\mathcal{P}_{s \rightarrow s'}^\pi = \sum_{a \in \mathcal{A}} \pi(a|s) \mathcal{P}_{s \rightarrow s'}^a$$

Markov Decision Process: Bellman Equation (new)

- ▶ given a policy π , **state value function** $v(s)$ is expected total return starting from state s

$$\begin{aligned} v_{\pi}(s) &= \mathbb{E}_{\pi}[G_t | s_t = s] \\ &= \mathbb{E}_{\pi}\left[R_{t+1} + \underbrace{\gamma(R_{t+2} + \gamma R_{t+3} + \dots)}_{G_{t+1}}\right] \end{aligned}$$

- ▶ $\mathbb{E}_{\pi}[\cdot]$ needs the integrate over $(a_0, a_1, \dots \in \mathcal{A}, s_0, s_1, \dots \in \mathcal{S}, r_1, r_2, \dots \in \mathcal{R})$:
- ▶ then chain of changes are then:

$$s_0 \rightarrow a_0, \quad (s_0, a_0) \rightarrow (s_1, r_1), \quad s_1 \rightarrow a_1, \quad (s_1, a_1) \rightarrow (s_2, r_2), \quad \dots$$

- ▶ for clarity, we let $s_t \rightarrow s$ and $s_{t+1} \rightarrow s'$:

Markov Decision Process: Bellman Equation (new)

- suppose we have a **universal state value function** $V_\pi(\cdot)$, i.e., no matter what the current state and action is:

$$\begin{aligned}
 & V_\pi(\cdot) \\
 &= \sum_{s_0} \Pr(s_0) \sum_{a_0} \pi(a_0|s) \sum_{s_1, r_1} \Pr(s_1, r_1 | s_0, a_0) \sum_{a_1} \pi(a_1|s_1) \sum_{s_2, r_2} \Pr(s_2, r_2 | s_1, a_1) \sum_{a_2} \cdots \sum_{s_3, r_3} \cdots \\
 & \quad [r_1 + \gamma(r_2 + \gamma r_3 + \dots)]
 \end{aligned}$$

- however, we do know the value $v_\pi(s_0)$:

$$\begin{aligned}
 & V_\pi(s_0) \\
 &= \sum_{a_0} \pi(a_0|s) \sum_{s_1, r_1} \Pr(s_1, r_1 | s_0, a_0) \underbrace{\left(r_1 + \gamma \sum_{a_1} \pi(a_1|s_1) \sum_{s_2, r_2} \Pr(s_2, r_2 | s_1, a_1) \sum_{a_2} \cdots \sum_{s_3, r_3} \cdots [r_2 + \gamma(r_3 + \gamma r_4 + \dots)] \right)}_{V_\pi(s_1) \stackrel{\Delta}{=} \mathbb{E}_\pi[G_{t+1} | s_1]} \\
 & \quad \underbrace{\hspace{15em}}_{V_\pi(s_0) \stackrel{\Delta}{=} \mathbb{E}_\pi[G_t | s_0]} \\
 &= \sum_{a_0} \pi(a_0|s) \sum_{s_1, r_1} \Pr(s_1, r_1 | s_0, a_0) (r_1 + \gamma \mathbb{E}_\pi[G_{t+1} | s_1])
 \end{aligned}$$

Bellman equation extends to $Q(s, a)$

summarise slides from before:

$$\begin{aligned}V_{\pi}(s) &= \sum_a \pi(a|s) \sum_{s', r'} \Pr(s', r' | s, a) (r' + \gamma v_{\pi}(s')) \\&= \sum_a \pi(a|s) \sum_{s', r'} \Pr(s', r' | s, a) (r' + \gamma \mathbb{E}_{\pi}[G_{t+1} | s']) \\&= \sum_a \pi(a|s) \mathbb{E}_{(s', r') \sim} [r' + \gamma v_{\pi}(s')]\end{aligned}$$

insert a to obtain Q function:

$$\begin{aligned}Q_{\pi}(s, a) &= \sum_{s', r'} \Pr(s', r' | s, a) (r' + \gamma v_{\pi}(s')) \\&= \sum_{s', r'} \Pr(s', r' | s, a) (r' + \gamma \mathbb{E}_{\pi}[G_{t+1} | s']) \\&= \mathbb{E}_{(s', r') \sim} [r' + \gamma v_{\pi}(s')]\end{aligned}$$

- ▶ this version is called **Bellman Expectation equation**
- ▶ expectation equation is linear in V , so you can solve for V using simple linear algebra
- ▶ Bellman Expectation equation is usually used to evaluate a known policy π

Bellman equation extends to $Q(s, a)$

since any policy π works, then:

$$Q_{\pi_*}(s, a) = \mathbb{E}_{(s', r') \sim \pi} [r' + \gamma v_{\pi_*}(s')]$$
$$\text{or } Q_*(s, a) = \mathbb{E}_{(s', r') \sim \pi} [r' + \gamma v_*(s')]$$

- ▶ this version is called **Bellman Optimality Equation**
- ▶ nonlinear (due to the max operation) in V , so there is no closed-form solution
- ▶ Bellman Expectation equation is usually used to evaluate a known policy π
- ▶ Many algorithms need to find the optimal solution (Q-learning, value iteration, policy iteration etc)
- ▶ optimality equation is used to learn the optimal policy π_*

Bellman optimality

- ▶ we know best $V_*(s)$ must be the best action from an optimal (state, action) pair: $Q_*(s, a)$:

$$V_*(s) = \max_a Q_*(s, a)$$

- ▶ and from before:

$$\begin{aligned} V_*(s) &= \max_a \underline{Q_*(s, a)} \\ &= \max_a \underline{\mathbb{E}_{(s', r') \sim} [r' + \gamma V_*(s')]} \quad \text{from last page} \\ &= \max_a \sum_{s', r'} \Pr(s', r' | s, a) (r' + \gamma V_*(s')) \\ &= \max_a \sum_{s', r'} \Pr(s', r' | s, a) (r' + \gamma \max_{a'} Q_*(s', a')) \\ &= \max_a \mathbb{E}_{(s', r') \sim} \left[r' + \gamma \max_{a'} Q_*(s', a') \middle| s \right] \\ &= \max_a \mathbb{E}_{(s', r') \sim} \left[r' + \gamma \max_{a'} Q_*(s', a') \right] \quad \text{removed } |s \text{ for clarity} \\ \implies Q_*(s) &= \mathbb{E}_{(s', r') \sim} \left[r' + \gamma \max_{a'} Q_*(s', a') \right] \end{aligned}$$

- ▶ also $r' \triangleq r'(s, \pi(s), s')$

loop $\forall s \in \mathbb{S}$

loop $\forall a \in \mathcal{A}$

$$Q(s, a) = \sum_{s' \in \mathbb{S}} \mathcal{P}_{s \rightarrow s'}^a [R(s, a, s') + \gamma V(s')]$$

$$V(s) = \max_a Q(s, a)$$

$$\pi_*(S) = \arg \max_a \sum_{s' \in \mathbb{S}} \mathcal{P}_{s \rightarrow s'}^a [R(s, a, s') + \gamma V(s')]$$

- ▶ implicitly solve for the state values under an ideal policy
- ▶ no need to define an actual policy during the iterations (although “current” policy is computable each iteration)

Policy Iteration

- ▶ choose an arbitrary policy π'
- ▶ **while** before some stopping criteria:
 - $\pi = \pi'$
 - compute the value function $V_\pi(1), \dots, V_\pi(n)$ using policy π :

$$V_\pi(s_0) = R(s, \pi(s_0)) + \gamma \sum_{s_1 \in \mathbb{S}} \mathcal{P}_{s_0 \rightarrow s_1}^{a_0} V_\pi(s_1)$$

improve the policy at each state:

$$\pi'(s_0) = \arg \max_{a_0} \left[R(s, a_0) + \gamma \sum_{s_1 \in \mathbb{S}} \mathcal{P}_{s_0 \rightarrow s_1}^{a_0} V_\pi(s_1) \right]$$

- ▶ start a policy π_0 and iterate towards π_* , by estimating state value associated with the policy, and making changes to action choices
- ▶ after each policy iteration, re-calculate value function for that policy. means you also work with value functions measure actual policies

Question: Is Q-Learning a Value Iteration or Policy Iteration?

- ▶ given π , let's work out how good is v^π :

$$\begin{aligned} V^\pi(s) &= \mathbb{E}_\pi [R_t | s_t = s] \\ &= \mathbb{E}_\pi [R_{t+1} + \gamma V^\pi(s_{t+1}) | s_t = s] \\ &= \sum_a \pi(s, a) \sum_{s'} \mathcal{P}_{ss'}^a [R_{ss'}^a + \gamma V^\pi(s')] \end{aligned}$$

- ▶ given you are in state s , instead of following $\pi(s)$, what if we choose **another** policy, such that $a \neq \pi(s)$:
- ▶ look at $Q^\pi(s, a)$, i.e., a value function for taking a particular action a , instead of average of all actions:

$$\begin{aligned} Q^\pi(s, a) &= \mathbb{E}_\pi [R_{t+1} + \gamma V^\pi(S_{t+1} | S_t = s, a_t = a)] \\ &= \sum_{s'} \mathcal{P}_{ss'}^a [R_{ss'}^a + \gamma V^\pi(s_{t+1})] \end{aligned}$$

- ▶ a should be chosen iff $Q^\pi(s, a) > V^\pi(s)$

Policy Improvement theorem

- ▶ if choosing $a \neq \pi(s)$ implies $Q^\pi(s, a) \geq V^\pi(s)$ for s
- ▶ then we choose policy π' for state s , and π for other $s' \neq s$
- ▶ this policy π' is at least as good as π , i.e., $V^{\pi'}(s) \geq V^\pi(s)$, i.e:

$$Q^\pi(s, a) > V^\pi(s) \implies V^{\pi'}(s) > V^\pi(s)$$

- ▶ proof:

$$\begin{aligned} V^\pi(s) &\leq Q^\pi(s, \pi'(s)) \quad \text{replace } s' \leftarrow \pi'(s) \\ &= \sum_{s'} \mathcal{P}_{ss'}^{\pi'(s)} [R_{ss'}^{\pi'(s)} + \gamma V^\pi(s_{t+1})] \quad \text{obviously } \pi' \text{ gives different } s' \text{ than } \pi \\ &\equiv \mathbb{E}_{\pi'(s)} [R_{t+1} + \gamma V^\pi(S_{t+1}) | S_t = s] \\ &\leq \mathbb{E}_{\pi'(s)} [R_{t+1} + \gamma Q^\pi(S_{t+1}, \pi'(S_{t+1})) | S_t = s] \quad \text{apply recursively to } V^\pi(S_{t+1}) \\ &= \mathbb{E}_{\pi'(s)} [R_{t+1} + \gamma \mathbb{E}_{\pi'(s')} [R_{t+2} + \gamma V^\pi(S_{t+2})] | S_t = s] \\ &\equiv \mathbb{E}_{\pi'} [R_{t+1} + \gamma \mathbb{E}_{\pi'} [R_{t+2} + \gamma V^\pi(S_{t+2})] | S_t = s] \\ &\vdots \\ &\leq \mathbb{E}_{\pi'} [R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots | S_t = s] \\ &= V^{\pi'}(s) \end{aligned}$$

Policy Improvement theorem (2)

- ▶ if apply this strategy to all states to get a new greedy policy:

$$\pi'(s) = \arg \max_a [Q^\pi(s, a)] \implies V^{\pi'} \geq V^\pi$$

- ▶ when $V^{\pi'} = V^\pi \implies$, and we know,

$$V^\pi(s) = \sum_a \pi(s, a) \sum_{s'} \mathcal{P}_{ss'}^a [R_{ss'}^a + \gamma V^\pi(s')]$$

we know:

$$V^{\pi'}(s) = \max_a \sum_{s'} \mathcal{P}_{ss'}^a [R_{ss'}^a + \gamma V^\pi(s')]$$

- ▶ a form of Bellman optimality equation
- ▶ therefore, $V^\pi = V^{\pi'} = V^*$

Bellman's equation: Three ways of solving it

$$V_{\pi}(s_0) = \mathbb{E}_{\pi}[G_t | s_0]$$

– could be approximated by **Monte-carlo**, i.e., sample s_1, s_2, \dots and compute G_t

$$= \mathbb{E}_{\pi} \left[r(s_0, \pi(s_0), s_1) + \gamma V_{\pi}(s_1) \right]$$

– could be approximated by **Temporal Difference**

$$= \sum_{a_0} \pi(a_0 | s_0) \sum_{s_1} \mathcal{P}_{s_0 \rightarrow s_1}^{a_0} \left[r(s_0, \pi(s_0), s_1) + \gamma V_{\pi}(s_1) \right]$$

– could be solved exactly by **Dynamic programming**

Solve Bellman's equation using Temporal Difference

$$V_{\pi}(s_0) = \sum_{a_0} \pi(a_0 | s_0) \sum_{s_1, r_1} \Pr(s_1, r_1 | s_0, a_0) (r_1 + \gamma V_{\pi}(s_1))$$

- drop $|s$ again for clarity:

$$\begin{aligned} V^{\pi}(s) &= \mathbb{E}_{s'} \left[r(s, \pi(s), s') + \gamma V^{\pi}(s') \right] \\ \implies V^{\pi}(s) + \eta V^{\pi}(s) &= V^{\pi}(s) + \eta \left(\mathbb{E}_{s'} \left[r(s, \pi(s), s') + \gamma V^{\pi}(s') \right] \right) \\ \implies V^{\pi}(s) &= V^{\pi}(s) + \eta \left(\mathbb{E}_{s'} \left[r(s, \pi(s), s') + \gamma V^{\pi}(s') \right] - V^{\pi}(s) \right) \end{aligned}$$

- instead of compute this expectation, in **each iteration** t , we sample a new state $\tilde{s}' \sim \Pr(s' | \dots)$

$$V_{t+1}^{\pi}(s) = V_t^{\pi}(s) + \eta \left(r(s, \pi(s), \tilde{s}') + \gamma V_t^{\pi}(\tilde{s}') - V_t^{\pi}(s) \right)$$

- note that the last equation is called **temporal difference**

Action-value (Q) function

- ▶ action-valued function $Q^\pi(s, a) = \mathbb{E}[G_t | S_t = s, A_t = a, \pi]$:
- ▶ expected total return starting from state s , taking action a , and then follow policy π
- ▶ Stochastic policy π :

$$V^\pi(s) = \mathbb{E}_{a \sim \pi(s)}[Q^\pi(s, a)]$$

- ▶ deterministic policy:

$$v^*(s) = \max_{a'} Q^*(s, a')$$

- ▶ from before;

$$\begin{aligned} V^*(s) &= \max_a \left(\mathbb{E}_{s'} \left[r(s, a, s') + \gamma \underbrace{V^*(s')}_{\text{by definition}} \middle| s \right] \right) \\ &= \max_a \left(\underbrace{\mathbb{E}_{s'} \left[r(s, a, s') + \gamma \left(\max_{a'} Q^*(s', a') \right) \right] s}_{Q^*(s, a) \text{ by definition}} \right) \end{aligned}$$

- ▶ therefore:

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

Action-value (Q) function

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

- drop $|s, a$, let's solve this by **temporal difference**:

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

$$\implies \textcolor{red}{Q}^\pi(\textcolor{red}{s}, \textcolor{red}{a}) + \eta Q^\pi(s, a) = \textcolor{red}{Q}^\pi(\textcolor{red}{s}, \textcolor{red}{a}) + \eta \left(\mathbb{E}_{s'} \left[r(s, a, s') + \gamma \left(\max_{a'} Q^\pi(s', a') \right) \right] \right)$$

$$\implies Q^\pi(s, a) = Q^\pi(s, a) + \eta \left(\mathbb{E}_{s'} \left[r(s, a, s') + \gamma \left(\max_{a'} Q^\pi(s', a') \right) \right] - Q^\pi(s, a) \right)$$

- instead of compute this expectation, in **each iteration** t , we sample a new state $(\tilde{s}', \tilde{a}) \sim \Pr(s', a | \dots)$.

Q-Learning: recursively:

$$Q(s, \tilde{a}) = Q(s, \tilde{a}) + \eta \left(\underbrace{r(s, \tilde{a}, \tilde{s}') + \gamma \left(\max_{a'} Q(\tilde{s}', a') \right)}_y - Q(s, \tilde{a}) \right)$$

- let $\eta = 1$:

$$Q(s, \tilde{a}) = r(s, \tilde{a}, \tilde{s}') + \gamma \left(\max_{a'} Q(\tilde{s}', a') \right)$$

Require: choice of γ Rewards matrix R

```
1:  $Q \leftarrow \mathbf{0}$ 
2: for each episode do
3:   randomise initiate state  $s_0$ 
4:   while goal state not reached do
5:     select  $(a, s') \sim \text{Pr}(a, s' | \cdot)$ 
6:      $Q(s, a) \leftarrow r(s, a, s') + \gamma (\max_{a'} Q(s', a'))$ 
7:      $s_t \leftarrow s_{t+1}$ 
8:   end while
9: end for
```

Question: Is Q-Learning a Value Iteration or Policy Iteration?

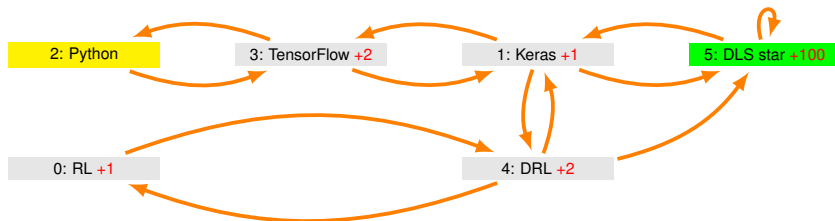
- ▶ if $a = \arg \max_a Q^\pi(s, a)$, then by setting $\pi'(a|s) = 1$, this policy is at least as good as π regardless of what π is
- ▶ if $Q^\pi(s, a) > V^\pi(s)$, then a is better than average since,

$$V^\pi(s) = \mathbb{E}_{a \sim \pi(a|s)}[Q^\pi(s, a)]$$

- ▶ obviously, we should increase $\pi(a|s)$ if $Q^\pi(s, a) > V^\pi(s)$

Q-Learning example

We took the example from the Markov Reward Process example earlier:



- ▶ there is small immediate rewards by going from one module to another
- ▶ you get a final large reward by becoming DLS star
- ▶ let $\gamma = 0.5$
- ▶ in this special example, $a = s'$, i.e., the action is to turn into the next state (module of studies).
- ▶ assume equal probabilities for all edges.

Q-Learning example: episode 1

$$R = \begin{matrix} S \downarrow, A \rightarrow & \text{RL}(0) & \text{Ke}(1) & \text{Py}(2) & \text{TF}(3) & \text{DRL}(4) & \text{DLS}^*(5) \\ \text{RL}(0) & - & - & - & - & 2 & - \\ \text{Ke}(1) & - & - & - & 2 & 2 & 100 \\ \text{Py}(2) & - & - & - & 2 & - & - \\ \text{TF}(3) & - & 1 & 0 & - & - & - \\ \text{DRL}(4) & 1 & 1 & - & - & - & 100 \\ \text{DLS}^*(5) & - & 1 & - & - & - & 100 \end{matrix}$$

before

$$Q = \begin{matrix} S \downarrow, A \rightarrow & \text{RL}(0) & \text{Ke}(1) & \text{Py}(2) & \text{TF}(3) & \text{DRL}(4) & \text{DLS}^*(5) \\ \text{RL}(0) & 0 & 0 & 0 & 0 & 0 & 0 \\ \text{Ke}(1) & 0 & 0 & 0 & 0 & 0 & 0 \\ \text{Py}(2) & 0 & 0 & 0 & 0 & 0 & 0 \\ \text{TF}(3) & 0 & 0 & 0 & 0 & 0 & 0 \\ \text{DRL}(4) & 0 & 0 & 0 & 0 & 0 & 0 \\ \text{DLS}^*(5) & 0 & 0 & 0 & 0 & 0 & 0 \end{matrix}$$

after

$$Q = \begin{matrix} S \downarrow, A \rightarrow & \text{RL}(0) & \text{Ke}(1) & \text{Py}(2) & \text{TF}(3) & \text{DRL}(4) & \text{DLS}^*(5) \\ \text{RL}(0) & 0 & 0 & 0 & 0 & 0 & 0 \\ \text{Ke}(1) & 0 & 0 & 0 & 0 & 0 & 100 \\ \text{Py}(2) & 0 & 0 & 0 & 0 & 0 & 0 \\ \text{TF}(3) & 0 & 0 & 0 & 0 & 0 & 0 \\ \text{DRL}(4) & 0 & 0 & 0 & 0 & 0 & 0 \\ \text{DLS}^*(5) & 0 & 0 & 0 & 0 & 0 & 0 \end{matrix}$$

- ▶ $s \sim \Pr(s|\cdot) = 1$, i.e. Keras
- ▶ at $s = 1$, it has **allowable actions**: go to state $\{3, 4, 5\}$, i.e., $a \in \{3, 4, 5\}$
- ▶ $(a, s') \sim \Pr(a, s'|\cdot) = (5, 5)$
- ▶ at $s' = 5$, it has **allowable actions**: $a' \in \{1, 5\}$:

$$\begin{aligned} Q(s, a) &= r(s, a, s') + \gamma (\max_{a'} Q(s', a')) \\ &= R(1, s' = 5) + 0.5 \max[Q(s' = 5, 1), Q(s' = 5, 5)] \\ &= 100 + 0.5 \times 0 = 100 \end{aligned}$$

- ▶ set $s \leftarrow s' \implies s = 5$, i.e., goal state, end

Q-Learning example: episode 2, Iteration 1

$$R = \begin{matrix} S \downarrow, A \rightarrow & \text{RL}(0) & \text{Ke}(1) & \text{Py}(2) & \text{TF}(3) & \text{DRL}(4) & \text{DLS}^*(5) \\ \text{RL}(0) & - & - & - & - & 2 & - \\ \text{Ke}(1) & - & - & - & 2 & 2 & 100 \\ \text{Py}(2) & - & - & - & 2 & - & - \\ \text{TF}(3) & - & 1 & 0 & - & - & - \\ \text{DRL}(4) & 1 & 1 & - & - & - & 100 \\ \text{DLS}^*(5) & - & 1 & - & - & - & 100 \end{matrix}$$

before

$$Q = \begin{matrix} S \downarrow, A \rightarrow & \text{RL}(0) & \text{Ke}(1) & \text{Py}(2) & \text{TF}(3) & \text{DRL}(4) & \text{DLS}^*(5) \\ \text{RL}(0) & 0 & 0 & 0 & 0 & 0 & 0 \\ \text{Ke}(1) & 0 & 0 & 0 & 0 & 0 & 100 \\ \text{Py}(2) & 0 & 0 & 0 & 0 & 0 & 0 \\ \text{TF}(3) & 0 & 0 & 0 & 0 & 0 & 0 \\ \text{DRL}(4) & 0 & 0 & 0 & 0 & 0 & 0 \\ \text{DLS}^*(5) & 0 & 0 & 0 & 0 & 0 & 0 \end{matrix}$$

after

$$Q = \begin{matrix} S \downarrow, A \rightarrow & \text{RL}(0) & \text{Ke}(1) & \text{Py}(2) & \text{TF}(3) & \text{DRL}(4) & \text{DLS}^*(5) \\ \text{RL}(0) & 0 & 0 & 0 & 0 & 0 & 0 \\ \text{Ke}(1) & 0 & 0 & 0 & 0 & 0 & 100 \\ \text{Py}(2) & 0 & 0 & 0 & 0 & 0 & 0 \\ \text{TF}(3) & 0 & 51 & 0 & 0 & 0 & 0 \\ \text{DRL}(4) & 0 & 0 & 0 & 0 & 0 & 0 \\ \text{DLS}^*(5) & 0 & 0 & 0 & 0 & 0 & 0 \end{matrix}$$

- ▶ $s \sim \Pr(s|\cdot) = 3$
- ▶ at $s = 3$, it has **allowable actions**: go to state $\{1, 2\}$, i.e., $a \in \{1, 2\}$
- ▶ $(a, s') \sim \Pr(a, s'|\cdot) = (1, 1)$
- ▶ at $s' = 1$, it has **allowable actions**: $a' \in \{3, 4, 5\}$:

$$\begin{aligned} Q(s, a) &= r(s, a, s') + \gamma \left(\max_{a'} Q(s', a') \right) \\ &= R(3, \mathbf{1}) + 0.5 \max[Q(\mathbf{1}, 3), Q(\mathbf{1}, 4), Q(\mathbf{1}, 5)] \\ &= 1 + 0.5 \times 100 = 51 \end{aligned}$$

- ▶ set $s \leftarrow s' \implies s = \mathbf{1}$, i.e., **not** a goal state, keep on going

Q-Learning example: episode 2, Iteration 2

$S \downarrow, A \rightarrow$

| | RL(0) | Ke(1) | Py(2) | TF(3) | DRL(4) | DLS*(5) |
|---------|-------|-------|-------|-------|--------|---------|
| RL(0) | — | — | — | — | 2 | — |
| Ke(1) | — | — | — | 2 | 2 | 100 |
| Py(2) | — | — | — | 2 | — | — |
| TF(3) | — | 1 | 0 | — | — | — |
| DRL(4) | 1 | 1 | — | — | — | 100 |
| DLS*(5) | — | 1 | — | — | — | 100 |

$R =$

before

$S \downarrow, A \rightarrow$

| | RL(0) | Ke(1) | Py(2) | TF(3) | DRL(4) | DLS*(5) |
|---------|-------|-------|-------|-------|--------|---------|
| RL(0) | 0 | 0 | 0 | 0 | 0 | 0 |
| Ke(1) | 0 | 0 | 0 | 0 | 0 | 100 |
| Py(2) | 0 | 0 | 0 | 0 | 0 | 0 |
| TF(3) | 0 | 51 | 0 | 0 | 0 | 0 |
| DRL(4) | 0 | 0 | 0 | 0 | 0 | 0 |
| DLS*(5) | 0 | 0 | 0 | 0 | 0 | 0 |

$Q =$

after

$S \downarrow, A \rightarrow$

| | RL(0) | Ke(1) | Py(2) | TF(3) | DRL(4) | DLS*(5) |
|---------|-------|-------|-------|-------|--------|---------|
| RL(0) | 0 | 0 | 0 | 0 | 0 | 0 |
| Ke(1) | 0 | 0 | 0 | 0 | 0 | 100 |
| Py(2) | 0 | 0 | 0 | 0 | 0 | 0 |
| TF(3) | 0 | 51 | 0 | 0 | 0 | 0 |
| DRL(4) | 0 | 0 | 0 | 0 | 0 | 0 |
| DLS*(5) | 0 | 0 | 0 | 0 | 0 | 0 |

$Q =$

- ▶ $s = 1$ from previous iteration
- ▶ at $s = 1$, it has **allowable actions**: go to state $\{3, 4, 5\}$, i.e., $a \in \{3, 4, 5\}$
- ▶ $(a, s') \sim \Pr(a, s' | \cdot) = (5, 5)$
- ▶ at $s' = 5$, it has **allowable actions**: $a' \in \{1, 5\}$:

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

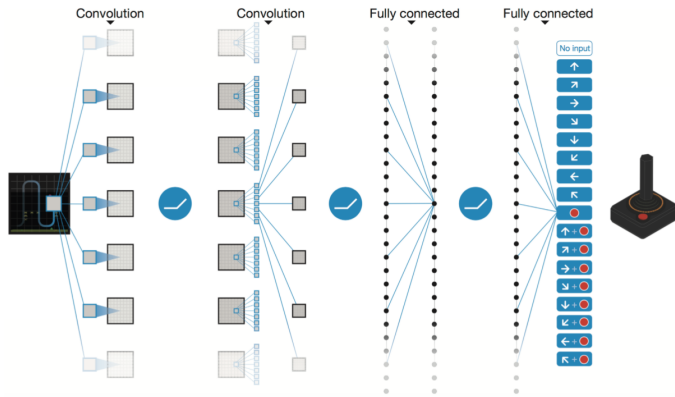
$$\begin{aligned} Q(1, 5) &= R(1, 5) + 0.5 \max[Q(5, 1), Q(5, 5)] \\ &= 100 + 0.5 \times 0 = 100 \end{aligned}$$

- ▶ set $s \leftarrow s' \implies s = 5$, i.e., goal state, end the state-action table gets updated until convergence.

- ▶ the states are far too many!
- ▶ need a **function approximator** to estimate the action-value function,
 $Q(s, a|\theta) \approx Q^*(s, a)$
- ▶ guess what? Deep Neural Network helps!

Represent $Q(s, a)$ using neural networks

- ▶ The figure below represents a row of the Q function table earlier:



Conv [16] \rightarrow ReLU \rightarrow Conv [32] \rightarrow ReLU \rightarrow FC [256] \rightarrow ReLU \rightarrow FC [|A|]

- ▶ these are **not** softmax functions.

abstracted algorithm for Deep Q-Learning

Require: Initialize an empty replay memory

Require: Initialize the DQN weights θ

- 1: **for** each episode **do**
- 2: **for** $t = 1, \dots, T$ **do**
- 3: with probability ϵ select \tilde{a} random action
- 4: otherwise, select:

$$\tilde{a} = \max_a (Q^*(s, a|\theta))$$

- 5: perform \tilde{a} and receive rewards r_t and state s' .
- 6: add tuple (s, \tilde{a}, r_t, s') into replay memory
- 7: Sample a mini-batch of tuples (s_j, a_j, r_j, s'_j) from the replay memory
- 8: and perform stochastic gradient descent on the DQN, based on the loss function:

$$\left(\underbrace{r_j + \gamma (\max_{a'} Q(s'_j, a'|\theta^-))}_{y_j} - Q(s_j, a_j|\theta) \right)^2$$

- 9: **end for**
- 10: **end for**

innovation

- ▶ freeze parameters of target network $Q(s'_j, a'|\theta^-)$ for fixed number of iterations
- ▶ while updating the online network $Q(s; a; \theta_i)$ by gradient descent

- ▶ same values θ both to select and to evaluate an action:

$$\begin{aligned}y_j &= r_j + \gamma(\max_{a'} Q(s'_j, a' | \theta)) \\ &= r_j + \gamma(Q(s'_j, \arg \max_a Q(s'_j, a, \theta) | \theta))\end{aligned}$$

- ▶ more likely to select overestimated values
- ▶ resulting in overoptimistic value estimates
- ▶ the solution is:

$$y_j = r_j + \gamma(\max_{a'} Q(s'_j, \arg \max_a Q(s'_j, a, \theta) | \theta'))$$

- ▶ still estimating value of policy according to current values defined by θ
- ▶ use second set of weights θ' to **fairly** evaluate value of this policy

- ▶ CNN and RNN are two of the building blocks in Deep Learning
- ▶ People have been putting them into many existing machine learning frameworks, and have generated many interesting stuff
- ▶ but there is plenty still needs to be explored!