Deep Reinforcement Learning: A brief introduction

A/Prof Richard Yi Da Xu

Yida.Xu@uts.edu.au

Wechat: aubedata

https://github.com/roboticcam/machine-learning-notes

University of Technology Sydney (UTS)

March 22, 2018

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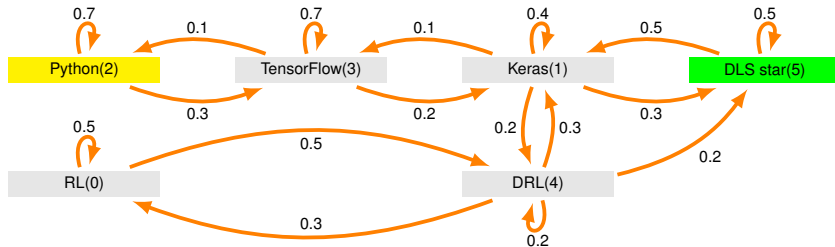 customer's attributes s, marketing actions a, customer signs up r
- **drone control**
 all avaiable sensor data a, controls s, not crashing r
- **chatbot**
  conversations to-date s, things that a robot will say a, customer satisfaction r

- ▶ 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? . . .

Let's add some rewards to being at each of the state:
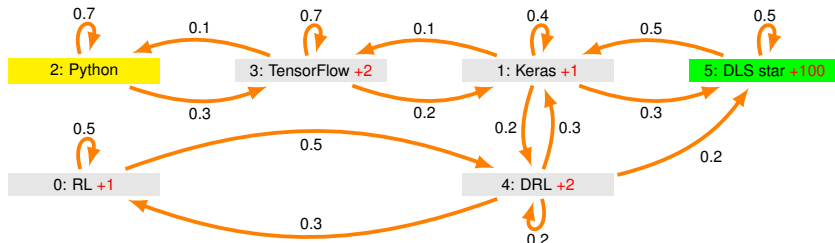
What we care is the **total return** G_t: sum of **discounted** reward from time-step t

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

note that G_t is a random variable
exercise what happens when $\gamma = 0$ and $\gamma = 1$

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

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

► $\mathbb{E}[.]$ needs the integrate over $(s_1, s_2, \cdots \in \mathcal{S}, r_1, r_2, \cdots \in \mathcal{R})$:

► $s_1, s_2, \ldots$ and $r_1, r_2, \ldots$ are generated in the following fashion:

$$s_0 \rightarrow (s_1, r_1) \qquad s_1 \rightarrow (s_2, r_2) \ldots$$

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

▶ suppose we have a **universal state value function** $V(.)$:

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

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

$$V(s_0) = \sum_{s_1, r_1} \Pr(s_1, r_1 | s_0) \left(r_1 + \gamma \underbrace{\sum_{s_2, r_2} \Pr(s_2, r_2 | s_1) \sum_{s_3, r_3} \ldots [r_2 + \gamma(r_3 + \gamma r_4 + \ldots)]}_{V(s_1) \overset{\Delta}{=} \mathbb{E}[G_{t+1} | s_1]} \right)$$

$$\underbrace{\phantom{V(s_0) = \sum_{s_1, r_1} \Pr(s_1, r_1 | s_0) \left(r_1 + \gamma \sum_{s_2, r_2} \Pr(s_2, r_2 | s_1) \sum_{s_3, r_3} \ldots [r_2 + \gamma(r_3 + \gamma r_4 + \ldots)] \right)}}_{V(s_0) \overset{\Delta}{=} \mathbb{E}[G_t | s_0]}$$

$$= \mathbb{E}_{s_1, r_1} [r_1 + \gamma V(s_1) | s_0]$$

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

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

▶ **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

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

▶ say $s \in \{1, \ldots, n\}$:

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

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

$$\cdots$$

take the first line,

$$\begin{aligned}
v(1) &= \mathbb{E}_{s_1} \left[R_1 + \gamma V(s_1) | s_0 = 1 \right] \\
&= R_1 + \gamma \mathbb{E} \left[V(s_1) | s_0 = 1 \right] \\
&= R_1 + \gamma \left(\sum_{s_1 = 1}^{n} v(s_1) \Pr(1 \to s_1) \right) \\
&= R_1 + \gamma \left(\sum_{j=1}^{n} v(j) \Pr(1 \to j) \right)
\end{aligned}$$

$$\cdots$$

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

$$v(k) = R_k + \gamma \left(\sum_{j=1}^{n} v(j) \Pr(k \to j) \right)$$

$$= R_k + \gamma \mathcal{P}_{k,:}^{\top} \mathbf{v}$$

$$\implies \mathbf{v} = \mathbf{R} + \gamma \mathcal{P} \mathbf{v}$$

$$\implies \begin{bmatrix} v(1) \\ \vdots \\ v(n) \end{bmatrix} = \begin{bmatrix} R_1 \\ \vdots \\ R_n \end{bmatrix} + \gamma \begin{bmatrix} \mathcal{P}_{1,1} & \dots & \mathcal{P}_{1,n} \\ \vdots & & \vdots \\ \mathcal{P}_{n,1} & \dots & \mathcal{P}_{n,n} \end{bmatrix} \begin{bmatrix} v(1) \\ \vdots \\ v(n) \end{bmatrix}$$

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

$$\mathbf{v} = \mathbf{R} + \gamma \mathcal{P} \mathbf{v}$$

$$(I - \gamma \mathcal{P}) v = R$$

$$\mathbf{v} = (I - \gamma \mathcal{P})^{-1} R$$

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

$$\mathcal{P}^{a_0}_{s_0 \to s_1} = \Pr(S_1 = s_1|S_0 = s_0, A_0 = a)$$

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

$$\mathcal{P}^{\pi}_{s_0 \to s_1} = \sum_{a_0 \in \mathcal{A}} \pi(a_0|s_0)\mathcal{P}^{a_0}_{s_0 \to s_1}$$

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

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

- $\mathbb{E}_\pi[.]$ needs the integrate over $(a_0, a_1, \dots \in \mathcal{A}, s_0, s_1, \dots \in \mathcal{S}, r_1, r_2, \dots \in \mathcal{R})$:

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

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

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

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

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

$$
\begin{aligned}
V_\pi(s_0) \\
= \sum_{a_0} \pi(a_0|s) \sum_{s_1,r_1} \Pr(s_1,r_1|s_0,a_0) \Bigg(r_1 + \gamma \underbrace{\sum_{a_1} \pi(a_1|s_1) \sum_{s_2,r_2} \Pr(s_2,r_2|s_1,a_1) \sum_{a_2} \cdots \sum_{s_3,r_3} \cdots [r_2 + \gamma(r_3 + \gamma r_4 + \dots)]}_{v_\pi(s_1) \overset{\Delta}{=} \mathbb{E}_\pi[G_{t+1}|s_1]} \Bigg)
\end{aligned}
$$

$$
\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}_{v_\pi(s_0) \overset{\Delta}{=} \mathbb{E}_\pi[G_t|s_0]}
$$

$$
= \sum_{a_0} \pi(a_0|s) \sum_{s_1,r_1} \Pr(s_1,r_1|s_0,a_0)(r_1 + \gamma \mathbb{E}_\pi[G_{t+1}|s_1])
$$

summarise slides from before:

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

$$= \sum_{a_0} \pi(a_0|s_0) \sum_{s_1, r_1} \Pr(s_1, r_1|s_0, a_0)(r_1 + \gamma \mathbb{E}_\pi[G_{t+1}|s_1])$$

$$= \sum_{a_0} \pi(a_0|s_0) \mathbb{E}_{(s_1, r_1) \sim} [r_1 + \gamma v_\pi(s_1)]$$

insert a_0 to obtain Q function:

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

$$= \sum_{s_1, r_1} \Pr(s_1, r_1|s_0, a_0)(r_1 + \gamma \mathbb{E}_\pi[G_{t+1}|s_1])$$

$$= \mathbb{E}_{(s_1, r_1) \sim} [r_1 + \gamma v_\pi(s_1)]$$

since any policy π works, then:

$$Q_{\pi_*}(s_0, a_0) = \mathbb{E}_{(s_1, r_1) \sim} [r_1 + \gamma v_{\pi_*}(s_1)]$$

▶ we know best $V_*(s)$ must be the best action from an optimal (state, action) pair: $Q_{\pi*}(s_0, a_0)$:

$$V_*(s_0) = \max_{a_0} Q_{\pi*}(s_0, a_0)$$

▶ and from before:

$$
\begin{aligned}
V_*(s_0) &= \max_{a_0} Q_{\pi*}(s_0, a_0) \\
&= \max_{a_0} \mathbb{E}_{(s_1, r_1) \sim} [r_1 + \gamma v_{\pi*}(s_1)] \qquad \text{from previous page} \\
&= \sum_{s_1, r_1} \Pr(s_1, r_1 | s_0, a_0)(r_1 + \gamma v_{\pi*}(s_1)) \\
&= \sum_{s_1, r_1} \Pr(s_1, r_1 | s_0, a_0)(r_1 + \gamma \max_{a_1} Q_{\pi*}(s_1, a_1)) \\
&= \max_{a_0} \mathbb{E}_{(s_1, r_1) \sim} \left[r_1 + \gamma \max_{a_1} Q_{\pi*}(s_1, a_1) \right] \\
&= \max_{a_0} \mathbb{E}_{(s_1, r_1) \sim} \left[r_1 + \gamma \max_{a_1} Q_{\pi*}(s_1, a_1) \Big| s_0 \right] \text{ removed } |s_0 \text{ for clarity previously} \\
\implies Q_*(s_0) &= \mathbb{E}_{(s_1, r_1) \sim} \left[r_1 + \gamma \max_{a_1} Q_{\pi*}(s_1, a_1) \Big| s_0 \right]
\end{aligned}
$$

▶ also $r_1 \triangleq r_1(s_0, \pi(s_0), s_1)$

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

► drop $|s$ again for clarity:

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

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

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

► 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**

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

— could be approximated by Monte-carlo, i.e., sample $s_1, s_2, \ldots$ 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

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

— could be solved exactly by Dynamic programming

- choose an arbitrary policy π'
- **while** before some stopping criteria:

  $\pi = \pi'$

  compute the value function $V_\pi(1), \ldots V_\pi(n)$ using policy π:

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

  improve the policy at each state:

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

Value Iteration

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

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

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

$$V(s_0) = \max_{a_0} Q(s_0, a_0)$$

- 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;

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

$$= \max_a \underbrace{\left(\mathbb{E}_{s'} \left[r(s, a, s') + \gamma \big(\max_{a'} Q^*(s', a') \big) \Big| s \right] \right)}_{Q^*(s', a') \text{ by definition}}$$

- therefore:

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

Action-value (Q) function

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

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

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

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

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

▶ 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:

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

▶ let $\eta = 1$:

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

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 $(a, s') \sim \Pr(a, s'|.)$
6:         compute $\max_{a'} Q(s', a')$
7:         $Q(s, a) \leftarrow r(s, a, s') + \gamma\left(\max_{a'} Q(s', a')\right)$
8:         $s_t \leftarrow s_{t+1}$
9:      **end while**
10: **end for**

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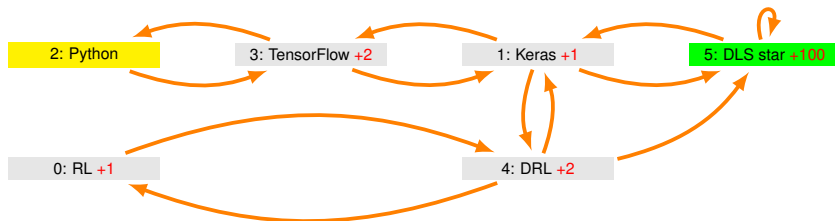
- there is small immediate rewards by going from one module to another
- you get a final large reward by becoming DLS star
- let $\gamma = 0.5$
- in this special example, $a = s'$, i.e., the action is to turn into the next state (module of studies).
- assume equal probabilities for all edges.

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

before

S ↓, A →	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	0
Py(2)	0	0	0	0	0	0
TF(3)	0	0	0	0	0	0
DRL(4)	0	0	0	0	0	0
DLS*(5)	0	0	0	0	0	0

$Q = $ (for the table above)

after

S ↓, A →	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	0	0	0	0	0
DRL(4)	0	0	0	0	0	0
DLS*(5)	0	0	0	0	0	0

$Q = $ (for the table above)

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

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

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

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

before

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

after

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

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

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

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

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

before

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

after

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

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

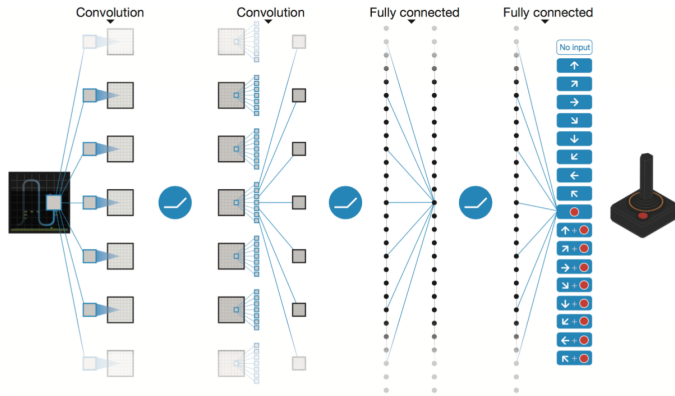$$Q(s, a) = r(s, a, s') + \gamma \left(\max_{a'} Q(s', a') \right)$$
$$Q(1, 5) = R(1, 5) + 0.5 \max[Q(5, 1), Q(5, 5)]$$
$$= 100 + 0.5 \times 0 = 100$$

- 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!

▶ 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.

Require: Initialize an empty replay memory
Require: Initialize the DQN weights θ
1: **for** each episode **do**
2:    **for** $t = 1, \ldots T$ **do**
3:       with probability ϵ select $\tilde{a}$ random action
4:       otherwise, select:

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

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:

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

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:

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

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

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

- 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!