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)

February 18, 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**!

Reinforcement Learning (RL)

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.

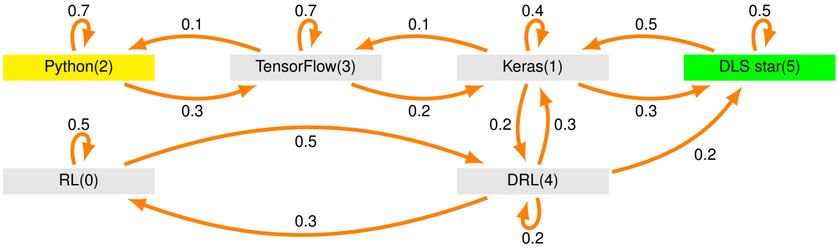
another way to look at it:

- 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

Application of RLs

- **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

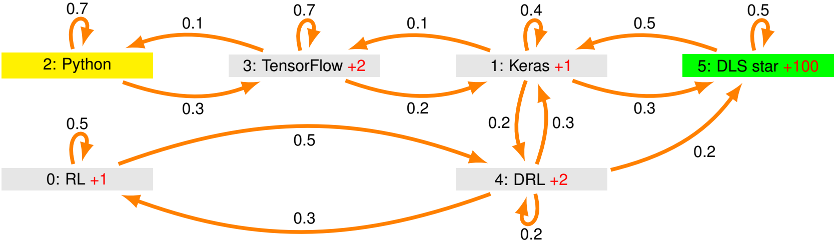
Markov Process



- 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

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

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

Markov Random Process: State value function

- state value function $v(s)$
- is expected total return starting from state s

$$
\begin{aligned}
v(s) \equiv v(S_t) &= \mathbb{E}_{s_{t+1}, s_{t+2}, \dots} \left[G_t | s_t = s \right] \\
&= \mathbb{E}_{s_{t+1}, s_{t+2}, \dots} \left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots | s_t = s \right]
\end{aligned}
$$

- $v(s)$ indicates the long future value, if you are currently at state s

Markov Random Process: Bellman Equation

- state value function $v(s)$ is expected total return starting from state s

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

- we know that, $\mathbb{E}_{x,y}[r(X) + r(X, Y)] = \mathbb{E}_x\left[r(X) + \mathbb{E}_y[r(X, Y)]\right]$

$$
\begin{aligned}
v(s) &= \mathbb{E}_{s_{t+1}}\left[R_{t+1} + \gamma \mathbb{E}_{s_2,s_3,\dots}\left[G_{t+1}|s_{t+1}\right]|s_t = s\right] \\
&= \mathbb{E}_{s_{t+1}}\left[R_{t+1} + \gamma v(S_{t+1})|s_t = s\right]
\end{aligned}
$$

- **Bellman equations**: value of the current state, $v(s)$ breaks up into (1) **immediate** and (2) **future** rewards.

Bellman Equation in matrix form

$$v(s_t) \equiv v(s) = \mathbb{E}_{s_1}\left[R_{t+1} + \gamma v(S_{t+1}) | s_t = s\right]$$

- say $s \in \{1, \ldots, n\}$:

$$\underbrace{v(s_t = 1)}_{v(1)} = \mathbb{E}_{s_1}\left[\underbrace{R_{t+1}(s_t = 1)}_{R_1} + \gamma v(S_{t+1}) | s_t = 1\right]$$

$$v(s_t = 2) = \mathbb{E}_{s_1}\left[R_{t+1}(s_t = 2) + \gamma v(S_{t+1}) | s_t = 2\right]$$

$$\ldots$$

take the first line,

$$\begin{aligned}
v(1) &= \mathbb{E}_{s_1}\left[R_1 + \gamma v(S_{t+1}) | s_t = 1\right] \\
&= R_1 + \gamma \mathbb{E}\left[v(S_{t+1}) | s_t = 1\right] \\
&= R_1 + \gamma \left(\sum_{s_{t+1}=1}^{n} v(s_{t+1}) \Pr(1 \to s_{t+1}) \right) \\
&= R_1 + \gamma \left(\sum_{j=1}^{n} v(j) \Pr(1 \to j) \right) \\
&\ldots \\
\implies v(k) &= R_k + \gamma \left(\sum_{j=1}^{n} v(j) \Pr(k \to j) \right)
\end{aligned}$$

Bellman Equation in matrix form (2)

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

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

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

- 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:

$$\mathcal{P}^a_{s \to s'} = \Pr(S_{t+1} = s' | S_t = s, A_t = a)$$

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

$$\mathcal{P}^\pi_{s \to s'} = \sum_{a \in \mathcal{A}} \pi(a|s) \mathcal{P}^a_{s \to s'}$$

- is the expected return starting from state s, and then follow policy π:

$$v^{\pi}(s) = \mathbb{E}[G_t | s_t = s]$$

- for example, $v^{\pi}(s_0)$:

$$v^{\pi}(s_0) = \mathbb{E}_{s_1, s_2, s_3, \dots}\left[\sum_{t=0}^{\infty} \gamma^t \underbrace{r_t(s_t, \pi(s_t), s_{t+1})}_{\text{reward}} | s_0\right]$$

- the **optimal state value** is computed using **best policy**:

$$V^*(s_0) = \max_{\pi}\left(\mathbb{E}_{s_1, s_2, s_3, \dots}\left[\sum_{t=0}^{\infty} \gamma^t r_t(s_t, \pi(s_t), s_{t+1}) | s_0\right]\right)$$

$$V^*(s_0) = \max_\pi \left(\mathbb{E}_{s_1,s_2,s_3,\dots} \left[\sum_{t=0}^{\infty} \gamma^t \underbrace{r_t(s_t, \pi(s_t), s_{t+1})}_{} \Big| s_0 \right] \right)$$

$$= \max_\pi \left(\mathbb{E}_{s_1} \left[\mathbb{E}_{s_2,s_3,\dots} \left[\gamma^0 r_0(s_0, \pi(s_0), s_1) + \sum_{t=1}^{\infty} \gamma^t r_t(s_t, \pi(s_t), s_{t+1}) \right] \Big| s_0 \right] \right)$$

$$= \max_\pi \left(\mathbb{E}_{s_1} \left[\gamma^0 r_0(s_0, \underbrace{\pi(s_0)}_{a_0}, s_1) + \mathbb{E}_{s_2,s_3,\dots} \left[\sum_{t=1}^{\infty} \gamma^t r_t(s_t, \pi(s_t), s_{t+1}) \Big| s_1 \right] \Big| s_0 \right] \right)$$

since $a_t = \pi(s_t) \implies \max_\pi f(\pi(s_t)) = \max_{a_t} f(\pi(s_t))$ **and** π still takes care of rest of $a_1, \dots$

$$= \max_{a_0} \left(\mathbb{E}_{s_1} \left[\gamma^0 r_0(s_0, \pi(s_0), s_1) + \max_\pi \left(\mathbb{E}_{s_2,s_3,\dots} \left[\sum_{t=1}^{\infty} \gamma^t r_t(s_t, \pi(s_t), s_{t+1}) \Big| s_1 \right] \right) \Big| s_0 \right] \right)$$

we like the first γ to be 1

$$= \max_{a_0} \left(\mathbb{E}_{s_1} \left[\underbrace{\gamma^0}_{=1} r_0(s_0, \pi(s_0), s_1) + \gamma \underbrace{\max_\pi \left(\mathbb{E}_{s_2,s_3,\dots} \left[\sum_{t=1}^{\infty} \gamma^{t-1} r_t(s_t, \pi(s_t), s_{t+1}) \Big| s_1 \right] \right)}_{V^*(s_1)} \Big| s_0 \right] \right)$$

Dynamic programming

$$
\begin{aligned}
V^*(s_0) &= \max_{\pi} \left(\mathbb{E}_{s_1, s_2, s_3, \dots} \left[\sum_{t=0}^{\infty} \gamma^t r_t(s_t, \pi(s_t), s_{t+1}) \middle| s_0 \right] \right) \\
&= \max_{a_0} \left(\mathbb{E}_{s_1} \left[r_0(s_0, \pi(s_0), s_1) + \underbrace{\gamma \max_{\pi} \left(\mathbb{E}_{s_2, s_3, \dots} \left[\sum_{t=1}^{\infty} \gamma^{t-1} r_t(s_t, \pi(s_t), s_{t+1}) \middle| s_1 \right] \right)}_{V^*(s_1)} \middle| s_0 \right] \right) \\
&= \max_{a_0} \left(\mathbb{E}_{s_1} \left[r_0(s_0, \pi(s_0), s_1) + \gamma V^*(s_1) \middle| s_0 \right] \right)
\end{aligned}
$$

Bellman's equation

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

- drop $|s$, one has:

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

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

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

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

Bellman's equation: Three ways of solving it

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left[G_t | s_t = s\right]$$

– could be approximated by Monte-carlo, i.e., sample $s_{t+1}, s_{t+2}, \dots$ and compute G_t

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

– could be approximated by Temporal Difference

$$= \sum_a \pi(a|s) \sum_{s'} \mathcal{P}^a_{s \to s'} \left[r(s, \pi(s), s') + \gamma V^{\pi}(s')\right]$$

– could be solved exactly by Dynamic programming

Action-value (Q) function

- action-valued function $Q^\pi(s, a) = \mathbb{E}\big[G_t | S_t = s, A_t = a, \pi\big]$:
- 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;

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

- therefore:

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

Action-value (Q) function

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

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

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

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

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

- 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:

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

- let $\eta = 1$:

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

- advantage function:

$$A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s)$$

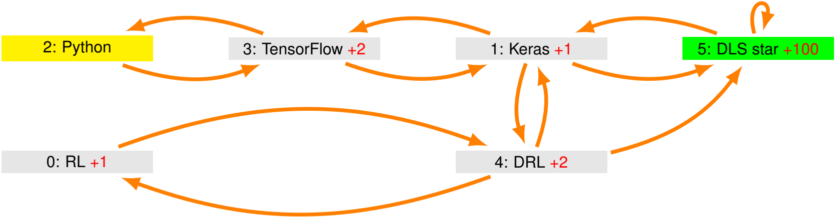
$$\begin{aligned}
&\mathbb{E}_{a\sim\pi(s)}[A^{\pi}(s,a)] \\
=&\mathbb{E}_{a\sim\pi(s)}[Q^{\pi}(s,a) - V^{\pi}(s)] \\
=&\mathbb{E}_{a\sim\pi(s)}[Q^{\pi}(s,a)] - V^{\pi}(s) = 0
\end{aligned}$$

  - V measures how good in a particular state s.
  - Q measures value of choosing a particular action when in this state.
  - A measures **relative importance** of each action.

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: $Q(s, a) \leftarrow r(s, a, s') + \gamma\big(\max_{a'} Q(s', a')\big)$
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.

Q-Learning example: episode 1

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

- $s \sim \Pr(s|.) = 1$, 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**: $a' \in \{1, 5\}$:

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

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

before

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

after

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

Q-Learning example: episode 2, Iteration 1

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

before

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

after

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

- $s \sim \Pr(s|.) = 3$
- 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**: $a' \in \{3, 4, 5\}$:

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

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

Q-Learning example: episode 2, Iteration 2

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

before

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

after

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

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

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

$$Q(1, 5) = R(1, 5) + 0.5 \max[Q(5, 1), Q(5, 5)]$$

$$= 100 + 0.5 \times 0 = 100$$

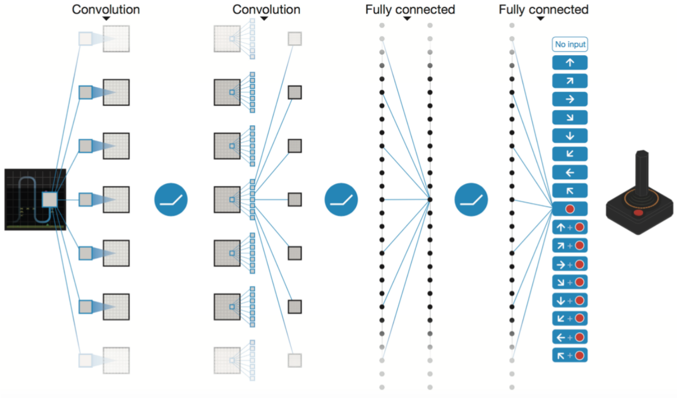
- set $s \leftarrow s' \implies s = 5$, i.e., goal state, end

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!

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

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



$$\text{Conv } [16] \rightarrow \text{ReLU} \rightarrow \text{Conv } [32] \rightarrow \text{ReLU} \rightarrow \text{FC } [256] \rightarrow \text{ReLU} \rightarrow \text{FC } [|A|]$$

- these are **not** softmax functions.

abstracted algorithm for Deep Q-Learning

Require: Initialize an empty replay memory
Require: Initialize the DQN weights θ

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

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

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

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

  end for
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

double Deep Q-Leanring

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

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

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

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

- 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

In summary

- 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!