Reinforcement Learning: A Brief Introduction

Yiwen Song

gavinsyw@sjtu.edu.cn

Reference Book:

Machine Learning by Zhihua Zhou, Tsinghua University Press, Chap. 6, 16

Reinforcement Learning: An Introduction by Richard S. Sutton and Andrew G. Barto, MIT Press, Chap. 1-6, 9, 16, 17

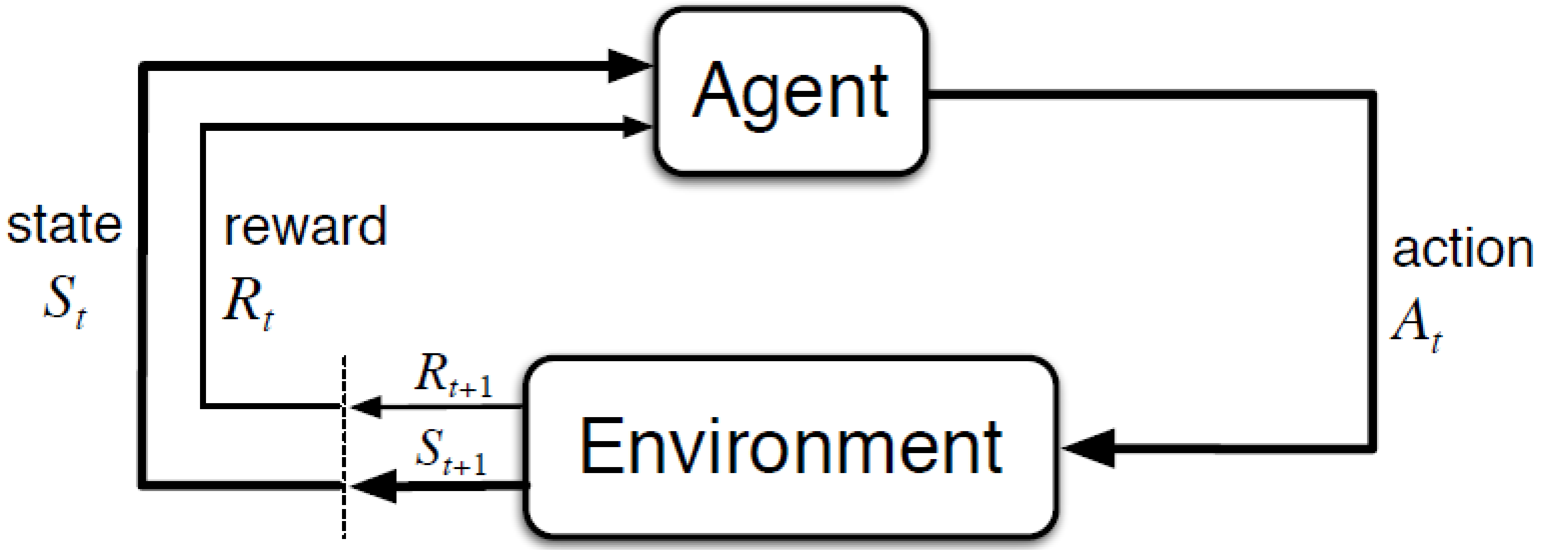
Contents

- Markov Decision Process: the Environment
- Traditional RL: TD, Q-, and SARSA
- Convergence: Whether & How fast
- Continuous State: Value Function Approximation
- Deep RL: Pros and Cons
- Industrial Examples

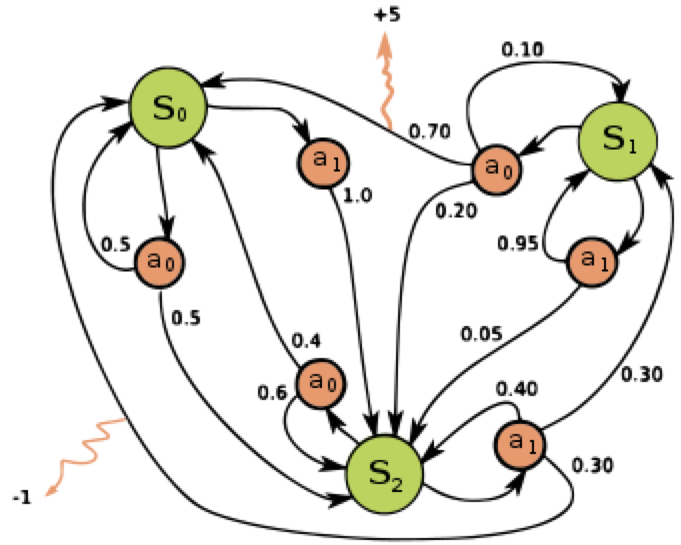
Markov Decision Process: the Environment

[Wikipedia, 2019] https://en.wikipedia.org/wiki/Reinforcement_learning.
[Wikipedia, 2019] https://en.wikipedia.org/wiki/Markov_decision_process
[Z. Zhou, 2016] Machine Learning, Tsinghua University Press
[S. Richard, et al., 2018] Reinforcement Learning: An Introduction, MIT Press

The interaction of an agent and the environment forms the basic model of RL.



The Markov Decision Process is a 4(5) tuple $(\boldsymbol{S}, \boldsymbol{A}, \boldsymbol{P_a}, \boldsymbol{R_a}, \gamma)$



There are 2 traditional solutions for MDP: Policy Iteration and Value Iteration

- $V(s) := \sum_{s'} P_{\pi(s)}(s, s')\left(R_{\pi(s)}(s, s') + \gamma V(s')\right)$
- $\pi(s) \ := \arg\max_a \left[\sum_{s'} P(s'|s, a)\left(R(s'|s, a) + \gamma V(s')\right)\right]$

- OR

- $V_{i+1}(s) \ := \max_a \left[\sum_{s'} P_a(s, s')\left(R_a(s, s') + \gamma V(s')\right)\right]$

Traditional RL: MC, TD, Q-, and SARSA

[Wikipedia, 2019] https://en.wikipedia.org/wiki/Temporal_difference_learning

[Wikipedia, 2019] https://en.wikipedia.org/wiki/Q-learning

[Wikipedia, 2019] https://en.wikipedia.org/wiki/State-action-reward-state-action

[Z. Zhou, 2016] Machine Learning, Tsinghua University Press

[S. Richard, et al., 2018] Reinforcement Learning: An Introduction, MIT Press

[R. Sutton, et al., 1995] Temporal difference learning and TD-Gammon, *Communications of the ACM*

[C. Watkins, et al., 1992] Q-learning, *Machine Learning*

[G. Rummery, et al., 1994] On-line Q-learning using connectionist systems, *Technical Report CUED/F-INFENG/TR*

Monte-Carlo Method solves the problem without complete knowledge of environment

- Input: environment E, action space A, start state x_0, total step T
- Output: policy π
- 2 steps: 1. Estimating v_π; 2. Gaining π^*
- $V(x_t, a_t) = \frac{V(x_t,a_t)\times n+R}{n+1}, R = \frac{1}{T-t}\sum_{i=t+1}^{T} r_i$
- ϵ-greedy policy for evaluating policy:

$$\pi(x) = \begin{cases} \arg\max_{a'} V(x, a'), \text{with probability } 1-\epsilon \\ \text{randomly choose an action from } A, \text{with probability } \epsilon \end{cases}$$

First-visit MC prediction, for estimating $V \approx v_\pi$

Input: a policy π to be evaluated

Initialize:

  $V(s) \in \mathbb{R}$, arbitrarily, for all $s \in \mathcal{S}$

  $Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Loop forever (for each episode):

  Generate an episode following π: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$

  $G \leftarrow 0$

  Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:

    $G \leftarrow \gamma G + R_{t+1}$

    Unless S_t appears in $S_0, S_1, \ldots, S_{t-1}$:

      Append G to $Returns(S_t)$

      $V(S_t) \leftarrow \text{average}(Returns(S_t))$

On-policy first-visit MC control (for ε-soft policies), estimates $\pi \approx \pi_*$

Algorithm parameter: small $\varepsilon > 0$

Initialize:

$\quad \pi \leftarrow$ an arbitrary ε-soft policy

$\quad Q(s,a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

$\quad Returns(s,a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Repeat forever (for each episode):

$\quad$ Generate an episode following π: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$

$\quad G \leftarrow 0$

$\quad$ Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:

$\quad\quad G \leftarrow \gamma G + R_{t+1}$

$\quad\quad$ Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1 \ldots, S_{t-1}, A_{t-1}$:

$\quad\quad\quad$ Append G to $Returns(S_t, A_t)$

$\quad\quad\quad Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$

$\quad\quad\quad A^* \leftarrow \arg\max_a Q(S_t, a)$ $\qquad$ (with ties broken arbitrarily)

$\quad\quad\quad$ For all $a \in \mathcal{A}(S_t)$:

$$\pi(a|S_t) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(S_t)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(S_t)| & \text{if } a \neq A^* \end{cases}$$

Temporal Difference (TD) Learning is an advance of Monte-Carlo Method applying DP

- Apply DP in the process of sampling (experiencing). Monte-Carlo Algorithm don't care what happened before

$$V(s_t) \leftarrow V(s_t) + \alpha[G_t - V(s_t)]$$

- However, TD consider both the observed reward and the estimate

$$V(s_t) \leftarrow V(s_t) + \alpha[R_{t+1} + \gamma V(s_{t+1}) - V(s_t)]$$

- Thus the estimating procedure goes faster

Tabular TD(0) for estimating v_π

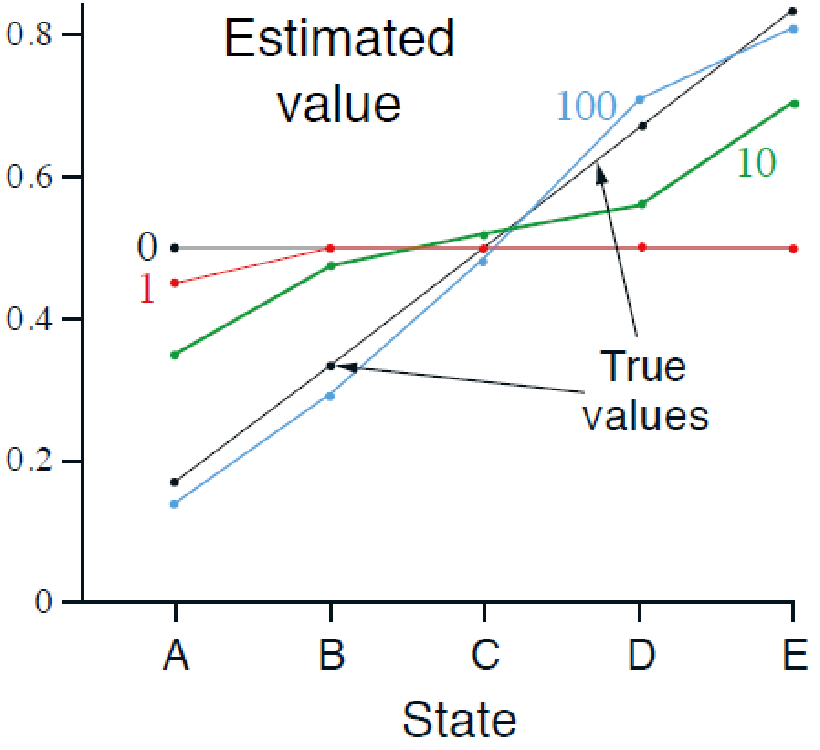
Input: the policy π to be evaluated
Algorithm parameter: step size $\alpha \in (0, 1]$
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(\textit{terminal}) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    $A \leftarrow$ action given by π for S
    Take action A, observe R, S'
    $V(S) \leftarrow V(S) + \alpha\big[R + \gamma V(S') - V(S)\big]$
    $S \leftarrow S'$
  until S is terminal

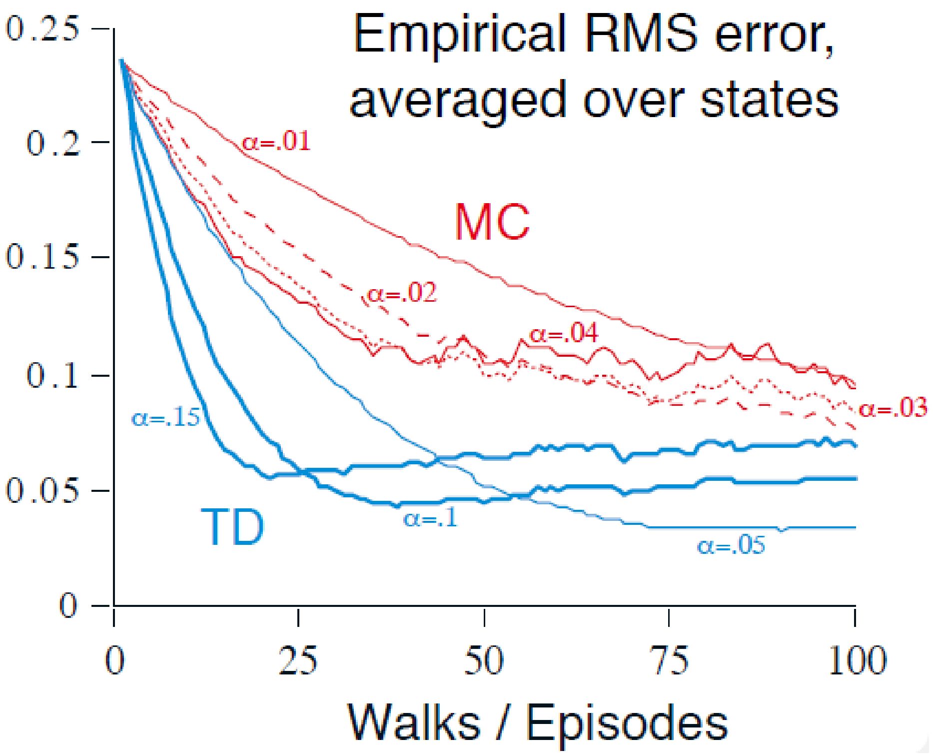
The example **Random Walk** shows how TD performs better than MC

- A random walk is a **Markov Reward Process (MRP)**.
- Starting point: C; move left or right with same probability; terminates on extreme left or right.
- True value: $v_\pi(\mathrm{A}\sim\mathrm{E}) = \frac{1}{6} \sim \frac{5}{6}$

Result of TD(0), with different episodes



Comparison between TD(0) and MC, starting at point C



Q-Learning: Off-policy TD Control

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
    Initialize S
    Loop 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

State-Action-Reward-State-Action (SARSA): On-policy TD-Control

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Choose A from S using policy derived from Q (e.g., ε-greedy)
  Loop 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

Convergence: Whether and How fast?

[S. Singh, et al., 2000] Convergence Results for Single-Step On-Policy Reinforcement-Learning Algorithms, *Machine Learning*

[V. Borkar, et al., 2000] The O.D.E. Method for Convergence of Stochastic Approximation and Reinforcement Learning, *SIAM Control Optimization*

[W. Beggs, 2005] On the convergence of reinforcement learning, *Journal of Economic Theory*

[C. Szepevari, et al., 1999] A unified analysis of value-function-based reinforcement-learning algorithms, *Neural Computation*

[S. Russel, et al., 1999] Convergence of reinforcement learning with general function approximators, *IJCAI*

[J. Tsitsiklis, 2003] On the Convergence of Optimistic Policy Iteration, *The Journal of Machine Learning Research*

Convergence of Reinforcement Learning

http://www.cs.cmu.edu/afs/cs.cmu.edu/project/learn-43/lib/idauction2/.g/web/glossary/converge.html

		Not Residual			Residual		
		Fixed distribution(On-policy)	Fixed Distribution	Usually-greedy distribution	Fixed distribution(On-policy)	Fixed Distribution	Usually-greedy distribution
Markov Chain	Lookup table	Y	Y		Y	Y	
	Averager	Y	Y		Y	Y	
	Linear	Y	N		Y	Y	
	Non-linear	N	N		Y	Y	
MDP	Lookup table	Y	Y	Y	Y	Y	Y
	Averager	Y	Y	N	Y	Y	Y
	Linear	N	N	N	Y	Y	Y
	Non-linear	N	N	N	Y	Y	Y
POMDP	Lookup table	N	N	N	Y	Y	Y
	Averager	N	N	N	Y	Y	Y
	Linear	N	N	N	Y	Y	Y
	Non-linear	N	N	N	Y	Y	Y

We should assure convergence when applying RL algorithms. However,

- **Convergence may be very slow when the state space is huge**, which is a major weakness of reinforcement learning.
- Two challenges:
  - No general theorem to infer convergence of a specific RL model.
  - Convergence difficult to infer for
    - **Non-i.i.d. samples**
    - **Dynamically changing learning policy (Usually in DRL)**

There are some tricks, however, to assure convergence speed, sacrificing convergence.

- For example:
- A simple TD-RL model (Q-, SARSA) converges as $t \to \infty$.
- The convergence ratio is $\frac{c_1}{t} + c_2 \log(1 + \frac{c_3}{t}) \to 0$
- By some means we can achieve a convergence ratio $c_4 e^{-c_5 t} + c_6$, sacrificing convergence for learning speed.

[S. Zou, 2019] Non-asymptotic Analysis of Reinforcement Learning Algorithms with Function Approximation (Seminar)

Continuous State: Value Function Approximation

[Z. Zhou, 2016] Machine Learning, Tsinghua University Press

[S. Richard, et al., 2018] Reinforcement Learning: An Introduction, MIT Press

[L. Busoniu, et al., 2010] Reinforcement Learning Dynamic Programming Using Function Approximation, Hall/CRC Press

Initiative of Value Function Approximation

- Consider an MDP with continuous state space.
- Can we just discretize the state space?
- Sometimes, yes. But not always yes.

- Therefore, why not just use a function to represent the value function?

$$V(s): \mathbb{R}^n \rightarrow \mathbb{R}, \qquad S: \ \mathbb{R}^n \rightarrow \mathbb{R}^n$$

The simplest case of VFA is **Linear-VFA**

- Take state space $X = \mathbb{R}^n$, and let $V_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{\theta}^T \boldsymbol{x}$
- We want to minimize the RMS error of our approximation:
$$E_{\boldsymbol{\theta}} = \mathbb{E}_{\boldsymbol{x}\sim\pi}\left[\left(V^{\pi}(\boldsymbol{x}) - V_{\boldsymbol{\theta}}(\boldsymbol{x})\right)^2\right]$$
- Apply gradient descent and we have
$$\boldsymbol{\theta} = \boldsymbol{\theta} + \alpha\left(V^{\pi}(\boldsymbol{x}) - V_{\boldsymbol{\theta}}(\boldsymbol{x})\right)\boldsymbol{x}$$
- Apply TD-Learning and we have the update method for $\boldsymbol{\theta}$
$$\boldsymbol{\theta} = \boldsymbol{\theta} + \alpha(r + \gamma\boldsymbol{\theta}^T \boldsymbol{x}' - \boldsymbol{\theta}^T \boldsymbol{x})\boldsymbol{x}$$

Semi-gradient TD(0) for estimating $\hat{v} \approx v_\pi$

Input: the policy π to be evaluated
Input: a differentiable function $\hat{v} : \mathcal{S}^+ \times \mathbb{R}^d \to \mathbb{R}$ such that $\hat{v}(\text{terminal},\cdot) = 0$
Algorithm parameter: step size $\alpha > 0$
Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)

Loop for each episode:
    Initialize S
    Loop for each step of episode:
        Choose $A \sim \pi(\cdot|S)$
        Take action A, observe R, S'
        $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[R + \gamma\hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})\big]\nabla\hat{v}(S,\mathbf{w})$
        $S \leftarrow S'$
    until S is terminal

More complex models work, such as Kernel-VFA

- Use **Kernel Function** as approximation for value function.
- Common Kernel Functions:

Name	Expression	Parameters
Linear Kernel	$\mathcal{K}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{x}_i^T \boldsymbol{x}_j$	
Polynomial Kernel	$\mathcal{K}(\boldsymbol{x}_i, \boldsymbol{x}_j) = (\boldsymbol{x}_i^T \boldsymbol{x}_j)^d$	$d \geq 1$
Gaussian Kernel	$\mathcal{K}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp\left(-\frac{\lvert\lvert \boldsymbol{x}_i - \boldsymbol{x}_j \rvert\rvert^2}{2\sigma^2}\right)$	$\sigma > 0$
Laplace Kernel	$\mathcal{K}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp\left(-\frac{\lvert\lvert \boldsymbol{x}_i - \boldsymbol{x}_j \rvert\rvert}{\sigma}\right)$	$\sigma > 0$
Sigmoid Kernel	$\mathcal{K}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \tanh(\beta \boldsymbol{x}_i^T \boldsymbol{x}_j + \theta)$	$\beta > 0, \theta < 0$

Deep RL: Pros and Cons

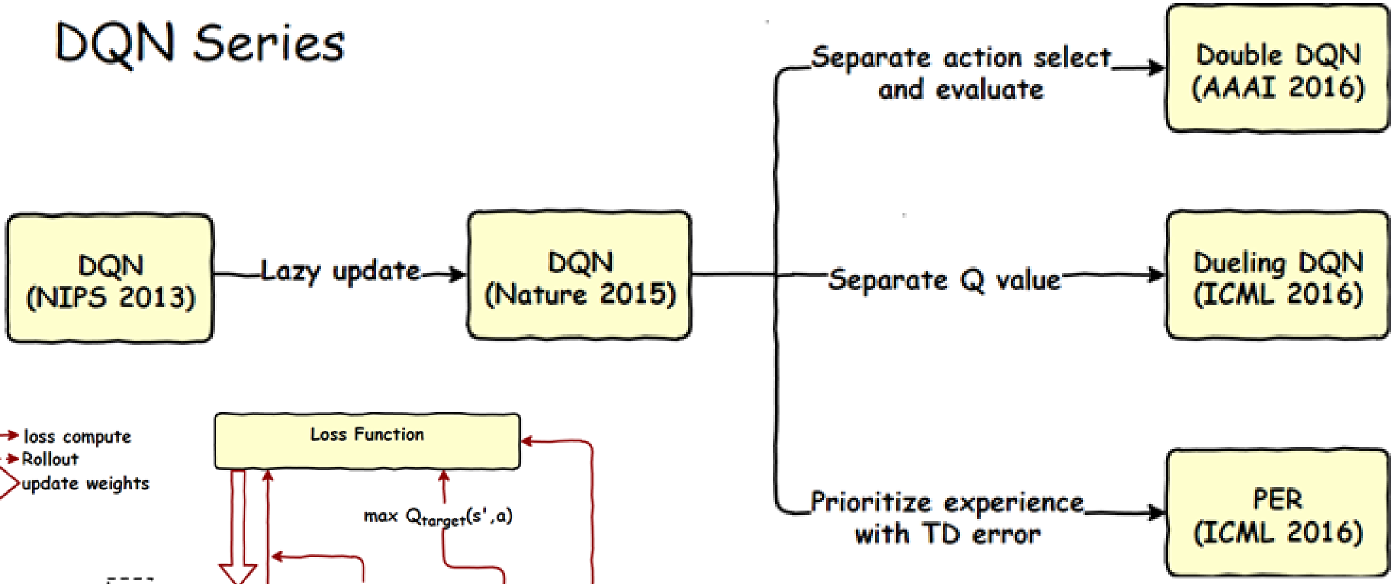
[P. Dimitri, 1996] Neuro-Dynamic Programming, MIT Press

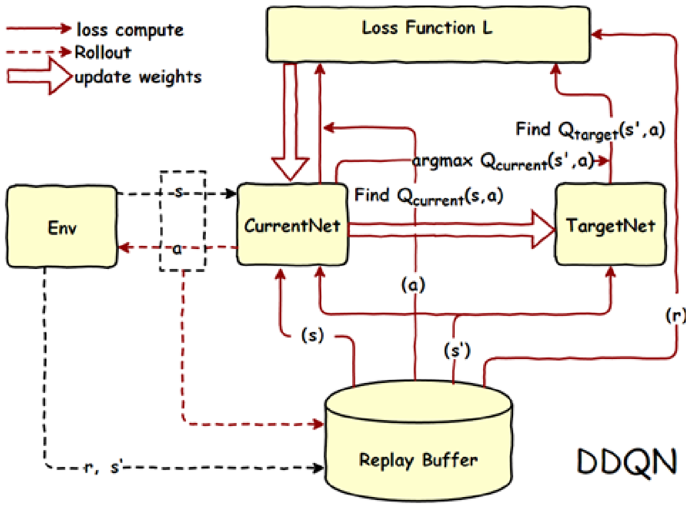
[V. Mnih, et al., 2013] Playing Atari with Deep Reinforcement Learning, *NIPS*

[V. Mnih, et al., 2015] Human-level control through deep reinforcementlearnin, *Nature*

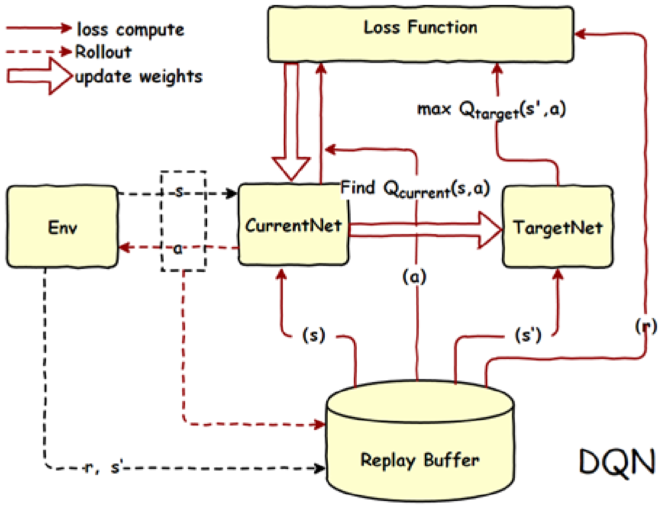
[V. Hasselt, et al, 2015] Deep Reinforcement Learning with Double Q-learning, *AAAI*

Revolution of DQN





$$L = (r + \gamma\, Q_{target}(s', \mathrm{argmax}_a Q_{current}(s',a)) - Q_{current}(s,a))^2$$



$$L = (r + \gamma\, \max Q_{target}(s',a) - Q_{current}(s,a))^2$$

Advantage of DQN

- Compared to Traditional Q
  - Multi-sampling during a step
  - Random sampling increases efficiency for continuous data
  - Avoid learning divergence by experiencing playback
- Compared to other Neural Networks
  - Better Interpretability
  - Can be manually corrected by **Apprenticeship Learning**

Disadvantage of DQN

- Uncertainty of convergence
- Great cost in hardware
- Sensitive to disturbance during training

Industrial Examples: When do we use RL?

[S. Richard, et al., 2018] Reinforcement Learning: An Introduction, MIT Press

[S. David, et al., 2018] A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play, *Science*

Apprenticeship Learning is usually applied in Industrial RL.

- Two ways:
  - **Direct Apprenticeship Learning**: Learn from an expert’s action first, then use on-policy training to get the policy;
  - **Inverse Reinforcement Learning**: discover the reward function from an expert’s action.
- Apprenticeship learning helps to improve the efficiency of learning.

DeepMind in reinforcement learning: **Chess, Go & RTS Games**