A Brief Introduction to Reinforcement Learning

Minlie Huang (黄民烈)

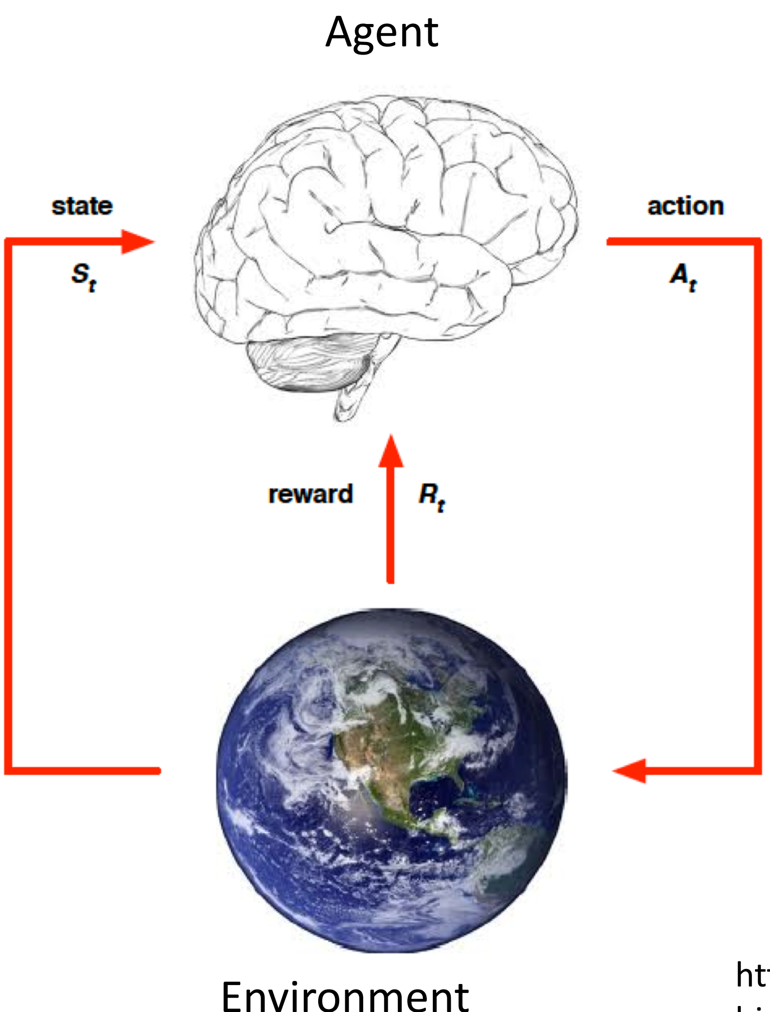
Dept. of Computer Science,

Tsinghua University

aihuang@tsinghua.edu.cn

http://coai.cs.tsinghua.edu.cn/hml

Reinforcement Learning

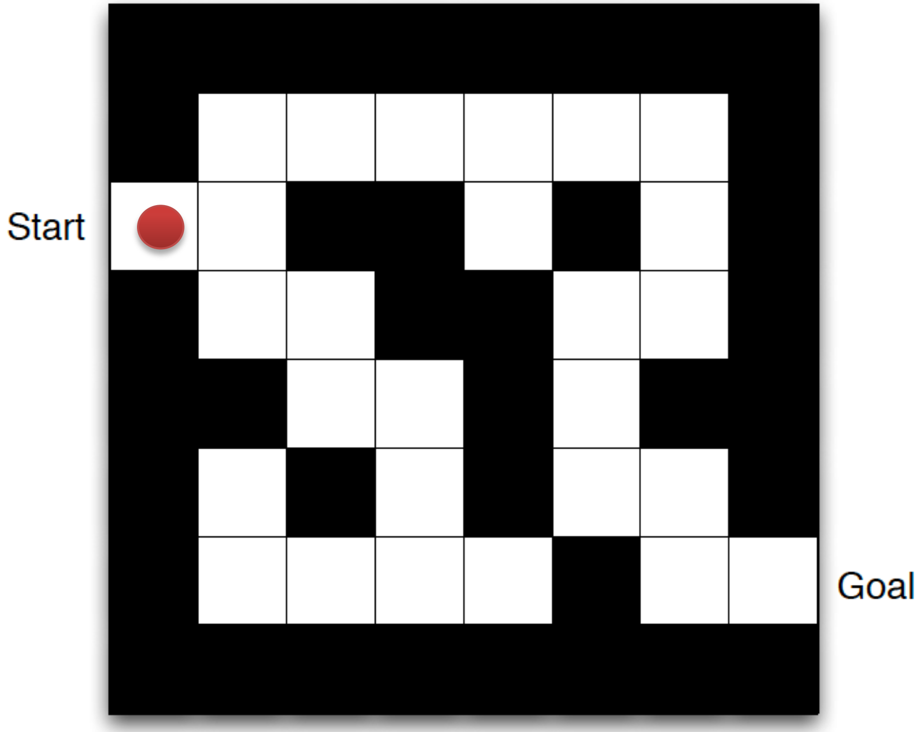


At each step ***t***:

- The agent receives a **state** S_t from environment
- The agent executes **action** A_t based on the received state
- The agent receives scalar **reward** R_t from the environment
- The environment transform into a new state S_{t+1}

http://www.cs.ucl.ac.uk/staff/d.silver/web/Teaching_files/intro_RL.pdf

Maze Example

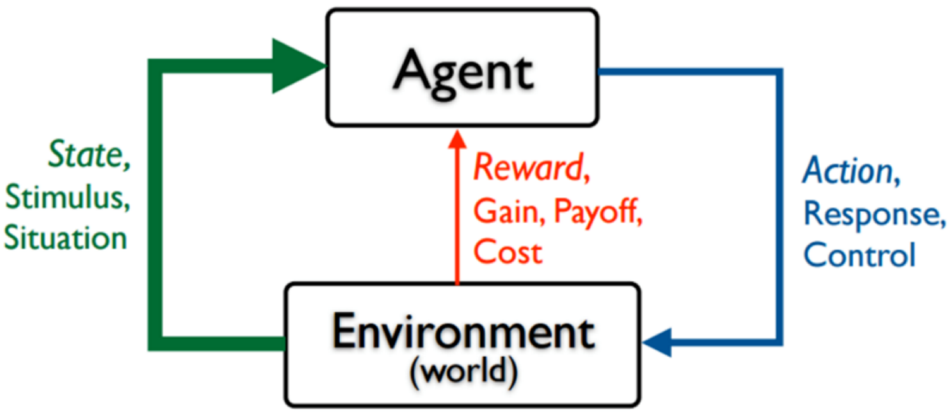


States: Agent's location
Actions: N, E, S, W
Rewards:

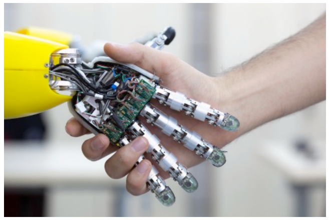
- 100 if reaching the goal
- -100 if reaching the dead end
- -1 per time-step

http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching_files/intro_RL.pdf

Deep Reinforcement Learning



Deep learning to represent states, actions, or policy functions



Robotics, control

Self-driving

Language interaction

System operating

Reinforcement Learning

- Markov Decision Process (**MDP**)

$\mathbf{s}_t$ – state
$\mathbf{o}_t$ – observation
$\mathbf{a}_t$ – action

$\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$ – policy

$\boxed{\pi_\theta(\mathbf{a}_t|\mathbf{s}_t) \text{ – policy (fully observed)}}$

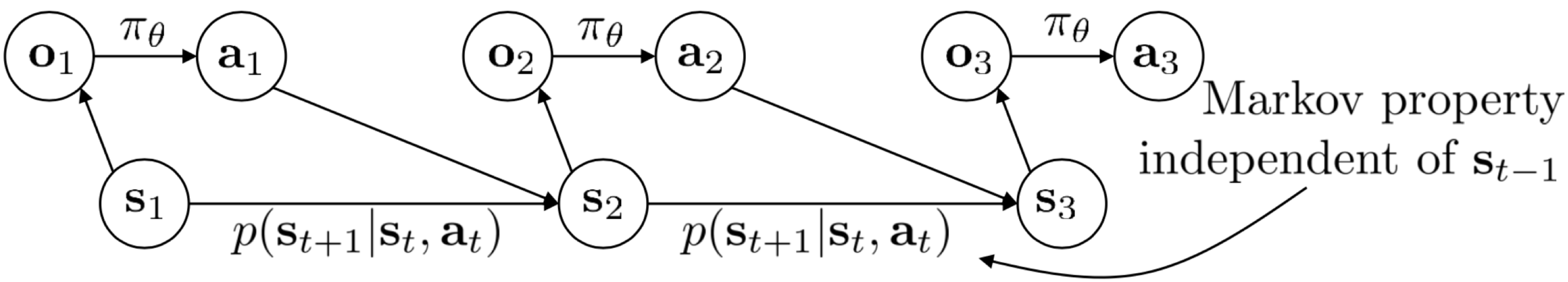
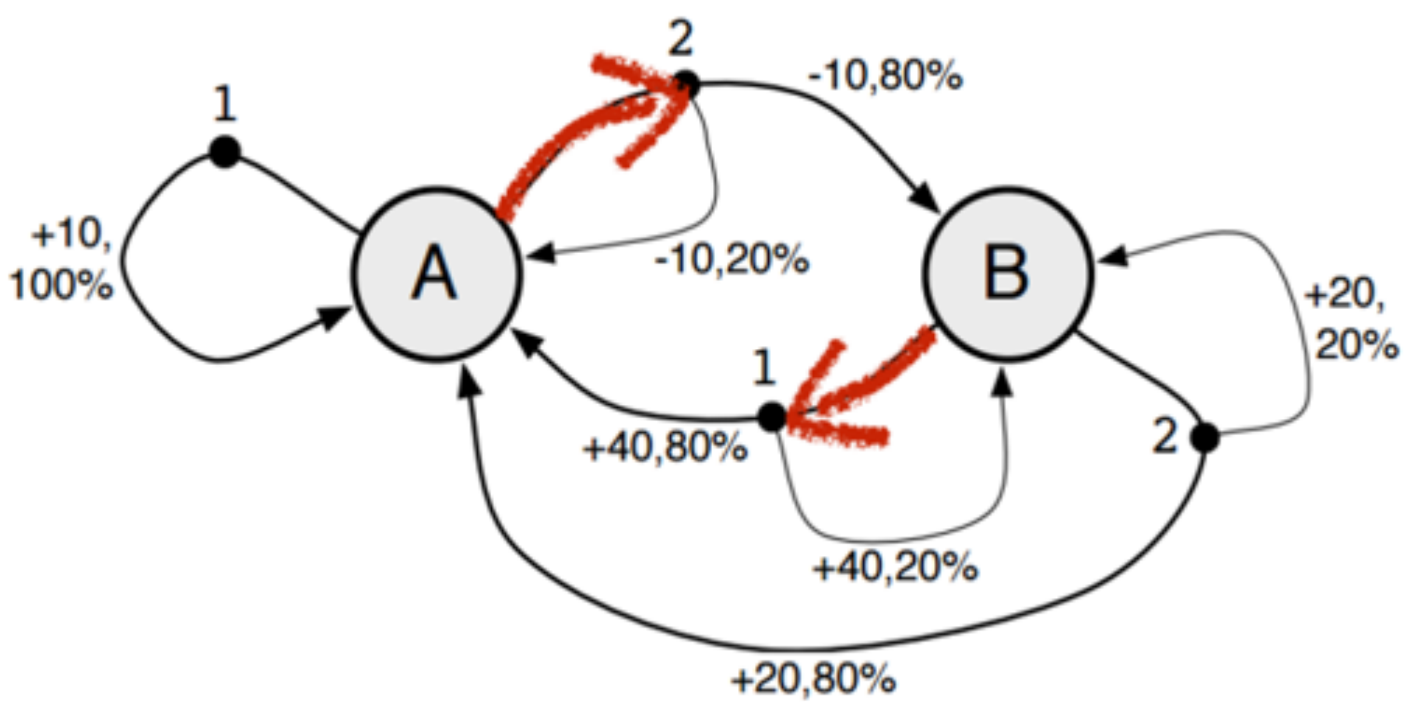


Figure from ICML Tutorial by Sergey Levine and Chelsea Finn

A MDP Game

If the agent is at state A, which action would you prefer?

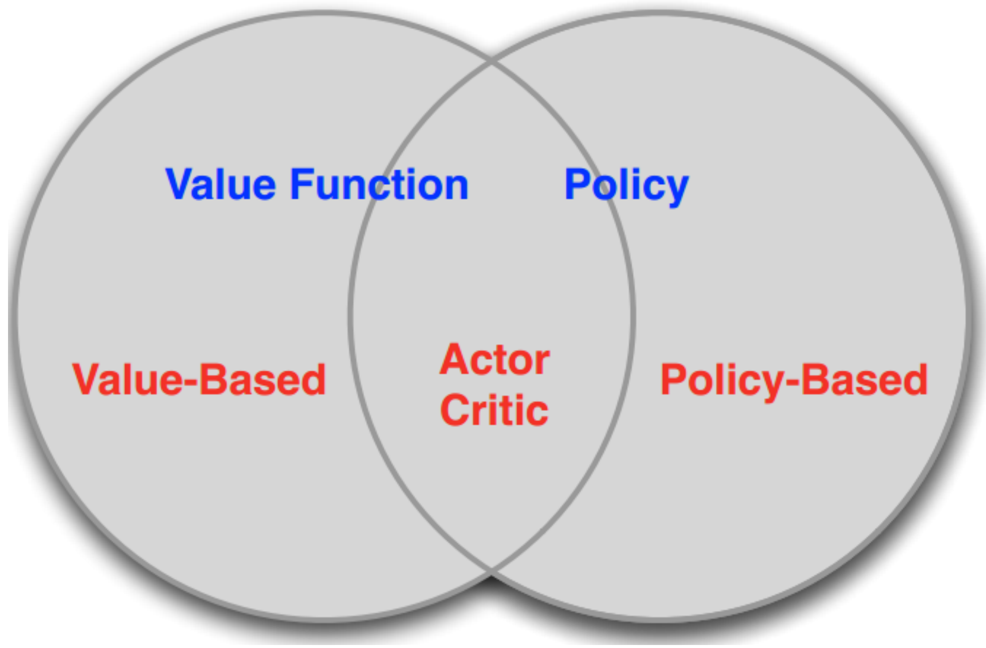


Reinforcement Learning

- Policy: Agent's behavior strategy
  - How to choose an action given the current state in order to maximize the total reward
- Deterministic policy： $\boldsymbol{a = \pi(o)}$
- Stochastic policy： $\boldsymbol{a \sim \pi(o) = p(a|o)}$
- **Fully observable:** $\boldsymbol{o = s}$
- **Partially observable:** $\boldsymbol{o \neq s}$

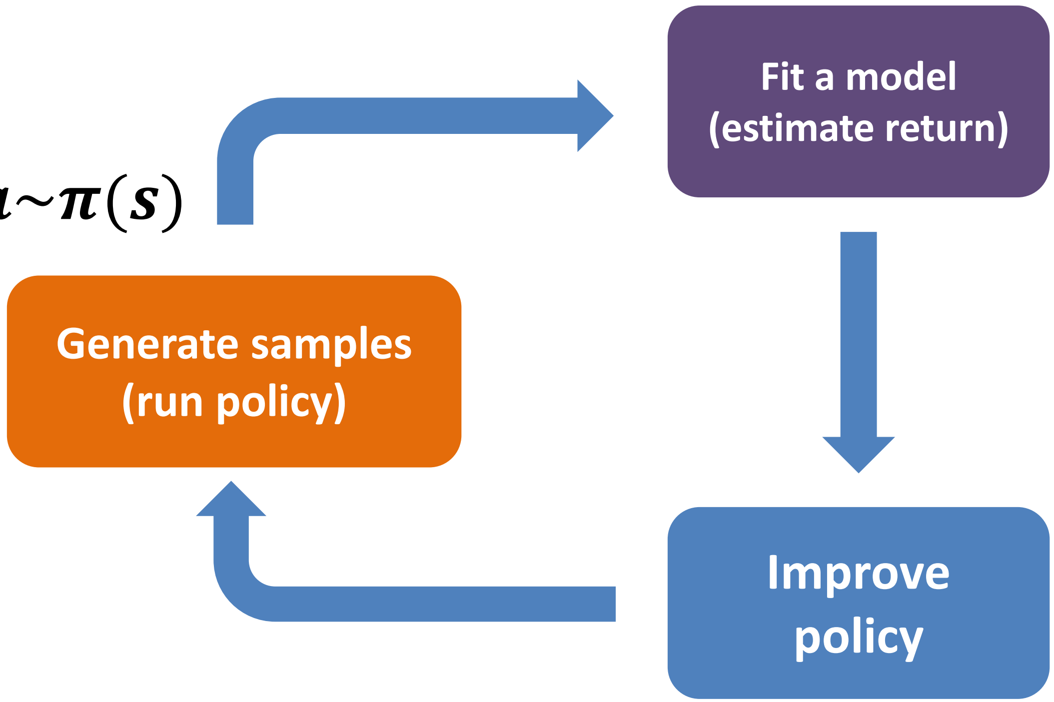
Value-Based RL

- Value Based
  - Learnt Value Function
  - Implicit Policy
- Policy Based
  - No Value Function
  - Learnt Policy
- Actor-Critic
  - Learnt Value Function
  - Learnt Policy



Q-Learning: Procedure

$a \sim \pi(s)$



Update $q_\pi(s, a)$
with $\Delta q_\pi(s, a)$

$$\pi(s) = \operatorname*{argmax}_a q_\pi(s, a)$$

Value Function

- A **value function** is a prediction of **future reward**
  - How much reward will I get from **action** ***a*** given **state** ***s*** under policy $\boldsymbol{\pi}$?
- **Q-value function** (action-value function) gives expected total reward
  - Each state-action pair ***(s, a)*** has an entry ***q(s, a)***

Value Function

- **Q-value function** gives expected total reward
  - Sample an trajectory with policy π
  - $\ldots S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}, R_{t+2}, S_{t+2}, \ldots$
  - Discount factor γ : between [0, 1], **how far ahead in time** the algorithm looks

$$q_\pi(s, a) = \mathbb{E}\left[\underbrace{R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots}_{\text{Delayed reward is taken into consideration}} \,\middle|\, S_t = s, A_t = a, A_{t+1:\infty} \sim \pi\right]$$

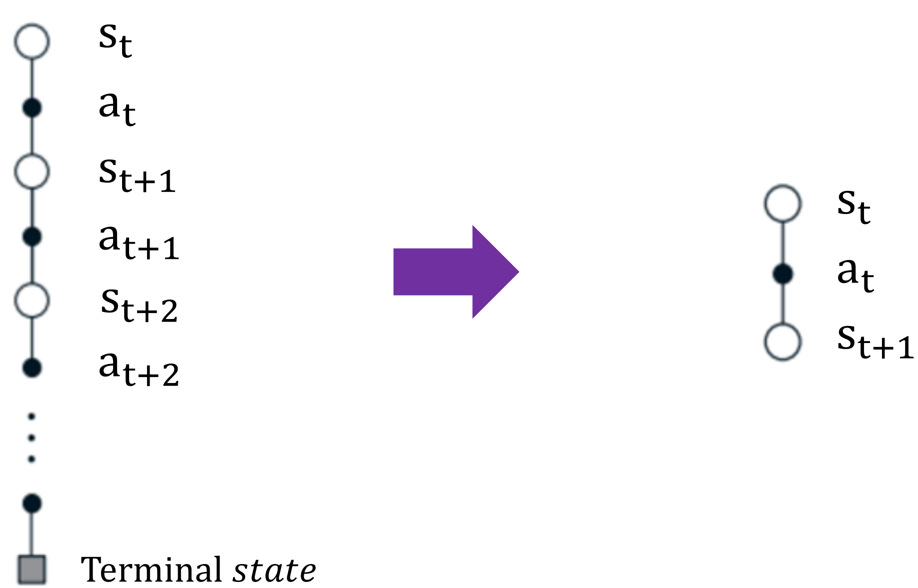
- **Future rewards!**

Value Function

- Bellman equation (with a particular policy π)

$$q_\pi(s,a) = \mathbb{E}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \,\middle|\, S_t = s, A_t = a, A_{t+1:\infty} \sim \pi\right]$$

$$q_\pi(s,a) = \mathbb{E}\left[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \,\middle|\, S_t = s, A_t = a, A_{t+1} \sim \pi\right]$$



Optimal Value Function

- An optimal value function is the maximum achievable value

$$q^*(s,a) = \max_{\pi}[q_\pi(s,a)] = q_{\pi*}(s,a)$$

- Act optimally:

$$\pi^*(s) = \underset{a}{\operatorname{argmax}}[q^*(s,a)]$$

- Optimal value maximizes over all decisions

$$q^*(s,a) = R_{t+1} + \gamma \max_{\mathrm{a}_{t+1}} \mathrm{R}_{t+2} + \gamma^2 \max_{\mathrm{a}_{t+2}} \mathrm{R}_{t+3}$$

$$= R_{t+1} + \gamma \max_{\mathrm{a}_{t+1}} \mathrm{q}^*(\mathrm{s}_{t+1}, \mathrm{a}_{t+1})$$

Q-Learning: Optimization

- Optimal q-values obey Bellman function

$$q^*(s,a) = E_{s'}[R + \gamma \max_{a'} q^*(s',a'|s,a)$$

- Minimizing mean-squared error of **value function** between *approximate value* and *true value (target)*

$$\Delta \boldsymbol{q}(\boldsymbol{s},\boldsymbol{a}) = \underbrace{\boldsymbol{\alpha}(\boldsymbol{R} + \boldsymbol{\gamma} \max_{\boldsymbol{a}'} \boldsymbol{q}(\boldsymbol{s}',\boldsymbol{a}')}_{\textit{Biased}\text{ true value}} - \underbrace{\boldsymbol{q}(\boldsymbol{s},\boldsymbol{a})}_{\text{Approximate value}})$$

- Policy $\boldsymbol{\pi}(\boldsymbol{s})$: choose the action that has the maximum **profit**

Q-Learning: Trial and Error

- Exploration & Exploitation : ε-greedy
  - Require sufficient exploration to enable learning
  - $a_t = \begin{cases} \arg\max_a q(s_t, a) & \text{with probability } 1 - \epsilon \\ \text{random action} & \text{with probability } \epsilon \end{cases}$

Q-Learning: Algorithm

Initialize q arbitrarily

Repeat (for each episode):

Initialize s

Repeat (for each time step of episode):

Take action from s according to policy (e.g. ε-greedy) $a \sim \pi(s)$

Observe reward r and transit to new state s'

$$q(s,a) \leftarrow (1-\alpha)q(s,a) + \alpha\left(r + \gamma \max_{a'} q(s',a')\right)$$

$$s \leftarrow s'$$

until s is terminal

Q-Learning: Review

- Strength
  - Learn from experience sampled from policy
  - Choose the best action for each state to get highest long-term reward
- Weakness
  - Over-estimate the action values using the maximum value as approximation for the maximum expected value
    → Double Q-learning
  - Large memory to store q values with increasing numbers of states and actions
    → Deep Q-learning

Double Q-learning

- Double estimator
  - sometimes underestimates rather than overestimates the maximum expected value

---

Initialize Q^A,Q^B,s
repeat
    Choose a, based on $Q^A(s,\cdot)$ and $Q^B(s,\cdot)$, observe r, s'
    Choose (e.g. random) either UPDATE(A) or UPDATE(B)
    if UPDATE(A) **then**
        Define $a^* = \arg\max_a Q^A(s',a)$
        $Q^A(s,a) \leftarrow Q^A(s,a) + \alpha(s,a)\left(r + \gamma Q^B(s',a^*) - Q^A(s,a)\right)$
    else if UPDATE(B) **then**
        Define $b^* = \arg\max_a Q^B(s',a)$
        $Q^B(s,a) \leftarrow Q^B(s,a) + \alpha(s,a)(r + \gamma Q^A(s',b^*) - Q^B(s,a))$
    end if
    $s \leftarrow s'$
until end

Function Approximation

- Represent value function by Q-network with weights θ

$$q(s, a, \theta) \approx q_*(s, a)$$

- May use back-propagation to compute $\nabla_\theta q$ in neural networks

- Popular choice: linear approximation, kernel methods, decision trees, deep learning models

Deep Q-Learning

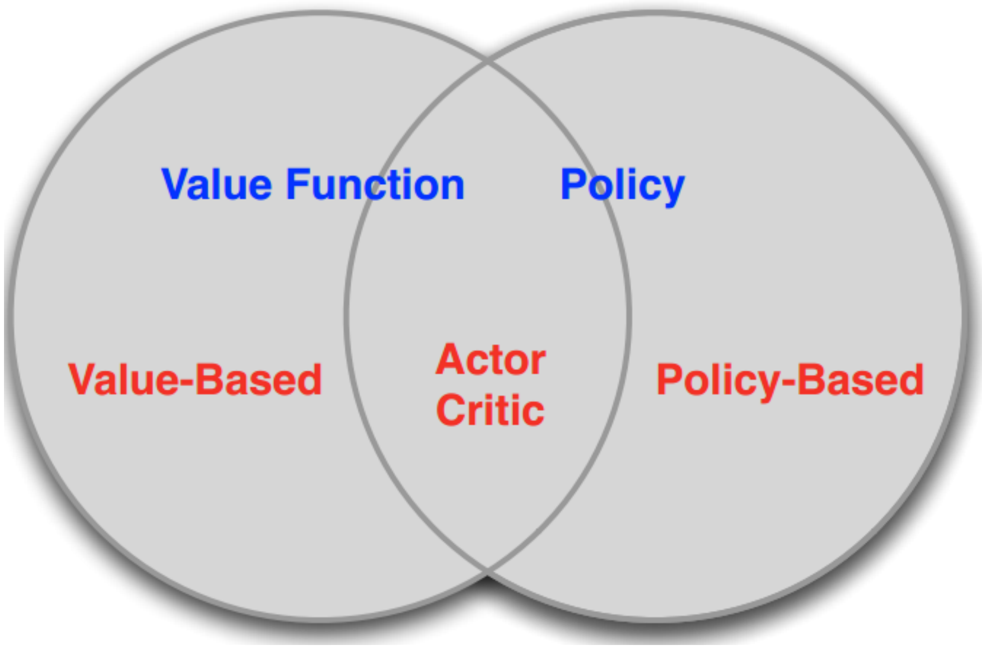
- Advantage
  - possible to apply the algorithm to larger problems, even when the state space is continuous
  - generalize earlier experiences to previously unseen states
- Challenge
  - unstable or divergent when a nonlinear function approximator such as a neural network is used to represent Q-value

Q-learning: Varients

- Deep Q Network (DQN)
  - *Playing Atari with Deep Reinforcement Learning*, V. Mnih et al., NIPS 2013 Workshop
- Double DQN
  - *Deep Reinforcement Learning with Double Q-learning*, H. van Hasselt et al., AAAI 2016
- Dueling DQN
  - *Dueling Network Architectures for Deep Reinforcement Learning*, Z. Wang et al., ICML 2016

Policy-Based RL

- Value Based
  - Learnt Value Function
  - Implicit Policy
- Policy Based
  - No Value Function
  - Learnt Policy
- Actor-Critic
  - Learnt Value Function
  - Learnt Policy



Policy-Based RL

- Directly parametrize the policy: $\pi_\theta(s,a) = P[a|s,\theta]$
- Comparing to Value-Based RL:
  - Approximate the value function: $q_\theta \approx q^\pi(s,a)$
  - Generate policy from the value function
    - Greedy: taking the action that can maximize ***q(s,a)***
    - Epsilon-greedy: small probability to explore new

$$- \; a_t = \begin{cases} \arg\max_a q(s_t, a) & \text{with probability } 1-\epsilon \\ \text{random action} & \text{with probability } \epsilon \end{cases}$$

Advantages

- Advantages:
  - Better convergence properties
  - Effective in high-dimensional or continuous action spaces
  - Can learn stochastic policies
- Disadvantages:
  - Converge to a local optimum
  - High variance

Policy-based RL

- Markov Chain:

$$p_\theta(s_1, a_1, \dots, s_T, a_T) = p(s_1) \prod_{t=1}^{T} \pi_\theta(a_t|s_t) p(s_{t+1}|s_t, a_t)$$

- Goal is to maximize the reward:

$$\theta^* = argmax_\theta E_{(s,a)\sim p_\theta(s,a)}[r(s,a)]$$

- Let $\tau = (s, a)$ be the state-action sequence:

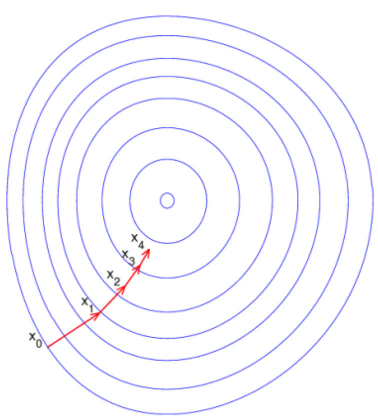
$$\theta^* = argmax_\theta E_{(\tau)\sim p_\theta(\tau)}[r(\tau)]$$

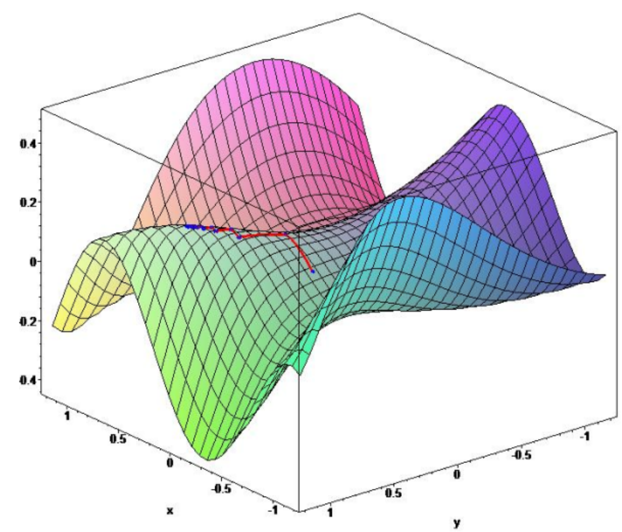
Policy Search

- Goal: find best θ for policy $\boldsymbol{\pi_\theta(\tau)}$
- How to measure the quality of policy $\boldsymbol{\pi_\theta(\tau)}$?
- Objective Function $\boldsymbol{J(\theta)}$
  - In Policy Gradient, $\boldsymbol{J(\theta) = E_{\tau\sim\pi_\theta(\tau)}[\sum r(\tau)]}$
- An optimization problem: find $\boldsymbol{\theta}$ to maximize $\boldsymbol{J(\theta)}$
  - Gradient Descent

Policy Gradient

- Search for a local maximum $J(\theta)$ by ascending the gradient of the policy: $\Delta\theta = \alpha \nabla_\theta J(\theta)$
- $\nabla_\theta J(\theta)$ is the policy gradient
- And α is learning rate





Policy Gradient

$$J(\theta) = E_{\tau\sim\pi_\theta(\tau)}[r(\tau)] = \int \pi_\theta(\tau) r(\tau) d\tau$$

$$\nabla_\theta J(\theta) = \int \nabla_\theta \pi_\theta(\tau) r(\tau) d\tau$$

$$= \int \pi_\theta(\tau) \nabla_\theta log \pi_\theta(\tau) r(\tau) d\tau$$

$$\nabla_\theta \pi_\theta(\tau) = \pi_\theta(\tau) \frac{\nabla_\theta \pi_\theta(\tau)}{\pi_\theta(\tau)} = \pi_\theta(\tau) \nabla_\theta log \pi_\theta(\tau)$$

Policy Gradient

$$\nabla_\theta J(\theta) = \int \underline{\nabla_\theta \pi_\theta(\tau)} r(\tau) d\tau$$

$$= \int \underline{\pi_\theta(\tau) \nabla_\theta log \pi_\theta(\tau)} r(\tau) d\tau$$

$$= E_{\tau \sim \pi_\theta(\tau)} [\underline{\nabla_\theta log \pi_\theta(\tau)} r(\tau)]$$

$$= E_{\tau \sim \pi_\theta(\tau)} \left[\left(\sum_{t=1}^{T} \nabla_\theta log \pi_\theta(a_t | s_t) \right) \left(\sum_{t=1}^{T} r(s_t, a_t) \right) \right]$$

One-Step MDPs

- A simple class of one-step MDPs
  - Start in state $\boldsymbol{s} \sim \boldsymbol{d}(\boldsymbol{s})$
  - Obtain reward $\boldsymbol{r} = \boldsymbol{R}_{\boldsymbol{s},\boldsymbol{a}}$ after one time-step
  - Then terminate
- Compute **policy gradient** with likelihood ratios:

$$J(\theta) = E_{\tau \sim \pi_\theta(\tau)}\left[\sum r(\tau)\right] = \sum_{s \in S} d(s) \sum_{a \in A} \pi_\theta(\tau) R_{s,a}$$

$$\nabla_\theta J(\theta) = \sum_{s \in S} d(s) \sum_{a \in A} \pi_\theta(\tau) \nabla_\theta \log \pi_\theta(\tau)\, R_{s,a}$$

$$= E_{\tau \sim \pi_\theta(\tau)}[\nabla_\theta \log \pi_\theta(\tau)\, r(\tau)]$$

Policy Gradient Theorem

- The policy gradient theorem:
  - Generalizes the likelihood ratio approach to multi-step MDPs
  - Replaces instantaneous reward r with long-term value $R^{\pi}(\tau)$

Policy Gradient Theorem

For any differentiable policy $\pi_\theta(\tau)$

For any policy objective function $J(\theta)$

the policy gradient is

$$\nabla_\theta J(\theta) = E_{\tau\sim\pi_\theta(\tau)}[\nabla_\theta \log \pi_\theta(\tau)\, \mathrm{R}^{\pi}(\tau)]$$

REINFORCE

- How to update the policy with policy gradient?
- Using stochastic gradient ascent
- Using **delayed reward** v_t as an unbiased sample of $R^{\pi}(s, a)$

function REINFORCE

Initialise $\boldsymbol{\theta}$ *arbitrarily*

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

    For *t=1 to T-1* ***do***

$$\boldsymbol{\theta} = \boldsymbol{\theta} + \alpha \nabla_\theta \log \pi_\theta(s_t, at)\, v_t$$

    end for

end for

Return $\boldsymbol{\theta}$

Review

- Policy-Based RL: directly parametrize the policy
- Objective Function $J(\theta)$: measure the quality of policy

$$J(\theta) = E_{\tau \sim \pi_\theta(\tau)}\left[\sum r(\tau)\right]$$

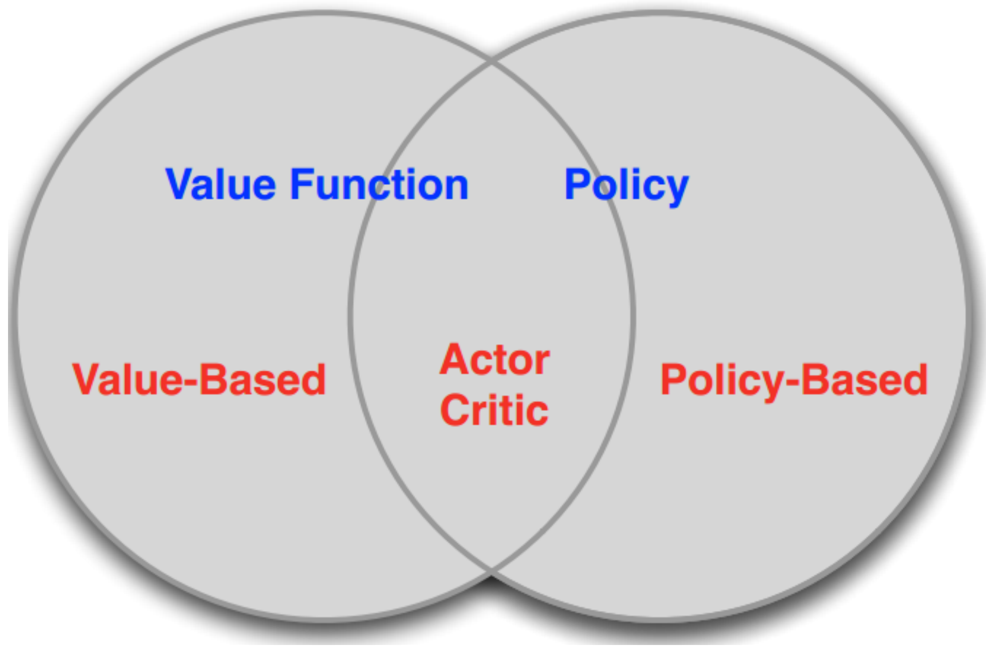
- Policy Gradient Theorem:

$$\nabla_\theta J(\theta) = E_{\tau \sim \pi_\theta(\tau)}[\nabla_\theta \log \pi_\theta(\tau)\, \mathrm{R}^\pi(\tau)]$$

- REINFORCE algorithm: stochastic gradient ascent and delayed reward

Actor-Critic

- Value Based
  - Learnt Value Function
  - Implicit Policy
- Policy Based
  - No Value Function
  - Learnt Policy
- Actor-Critic
  - Learnt Value Function
  - Learnt Policy



Actor-Critic

- Policy gradient still has high variance ☹
- Use a critic to evaluate the action just selected
- State-value function & state-action value function

$$V_w(s) = E_{a \sim \pi_\theta(a|s)}[q(a,s)]$$

- Actor-Critic maintains two sets of parameters
  - Actor (Policy Gradient) : Updates policy parameters θ
  - Critic (Function Approximation) : Updates parameters *w* of the state-value function

Actor-Critic

- Approximate policy gradient

$$\nabla_\theta J(\theta) = E_{\tau\sim\pi_\theta(\tau)}[\nabla_\theta \log \pi_\theta(\tau)\, \mathrm{R}^\pi(\tau)]$$

$$\rightarrow E_{\tau\sim\pi_\theta(\tau)}[\nabla_\theta \log \pi_\theta(\tau)\, \underbrace{(r + \gamma \mathrm{V}_w(\tau') - \mathrm{V}_w(\tau))}_{\text{TD error in Q-learning}}]$$

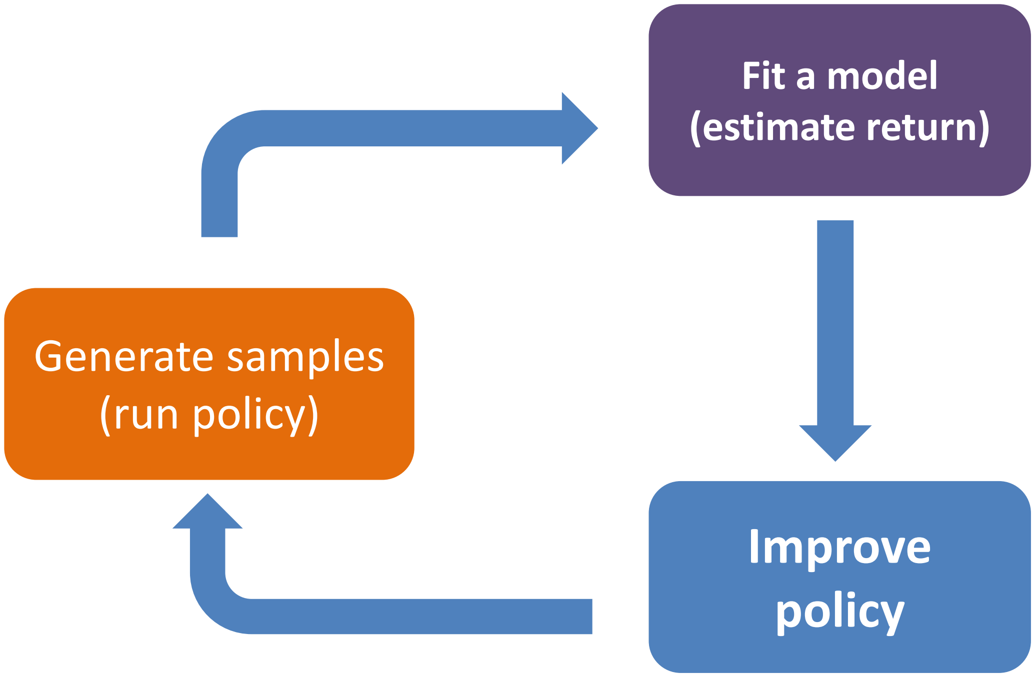
- Fit $\mathrm{V}_w(s)$ to sampled reward sums $\sum R^\pi(s', a') \approx R^\pi(s, a) + \mathrm{V}_w(s')$

$$L(w) = \frac{1}{2}||\overbrace{r + \gamma \mathrm{V}_w(\mathrm{s}')}^{\text{Target}} - \mathrm{V}_w(\mathrm{s})||^2$$

Actor-Critic: Procedure

$w \leftarrow w - \nabla_w L(w)$

$a \sim \pi(s)$



$\theta \leftarrow \theta + \nabla_\theta J(\theta)$

Actor-Critic: Algorithm

Initialize actor $\boldsymbol{\pi}$ with $\boldsymbol{\theta}$, critic $\boldsymbol{V}$ with w arbitrarily

Repeat (for each episode):

Initialize s

Repeat (for each time step of episode):

Take action from s according to actor $a \sim \pi_{\theta}(s)$

Observe reward r and transit to new state s'

Evaluate TD error $\mathrm{A}(s) = r + \gamma \mathrm{V}_w(s') - \mathrm{V}_w(s)$

$w \leftarrow w - \alpha_w \nabla_w ||\mathrm{A}(s)||^2$

Calculate $\boldsymbol{A(s)}$ using new critic V_w

$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha_\theta \nabla_\theta \log \pi_\theta(\mathrm{s})\mathrm{A}(s)$

$s \leftarrow s'$

until s is terminal

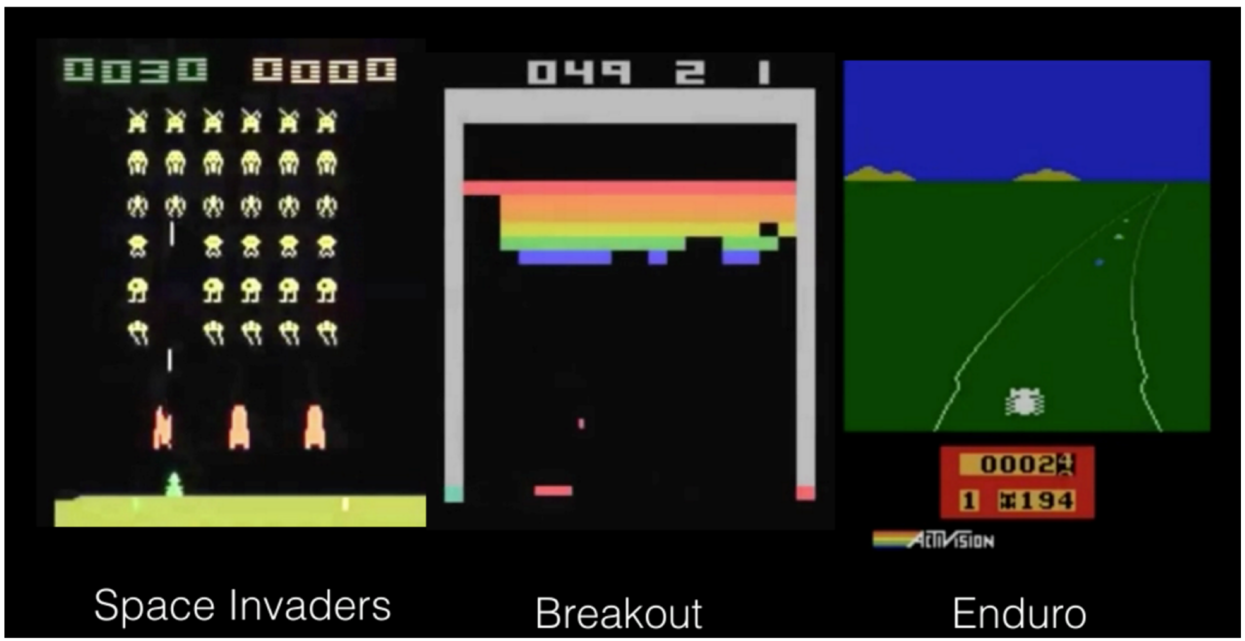
⊙Summary and Discussions

Reinforcement Learning

- **Sequential decision:** current decision affects future decision
- **Trial-and-error: just try, do not worry making mistakes**
  - **Explore** (new possibilities)
  - **Exploit** (with the current best policy)
- **Future reward:** maximizing the future rewards instead of just the intermediate rewards at each step
  - Remember **q(s,a)**

Difference to Supervised Learning

- Supervised learning: given a set of samples (x_i, y_i), estimate f: $X \rightarrow Y$



Difference to Supervised Learning

- You know what a true goal is, but do not know how to achieve that goal
- Through interactions with environment (trial-and-error)
- Many possible solutions (policies), which is optimal?