Stanford CS234 Reinforcement Learning

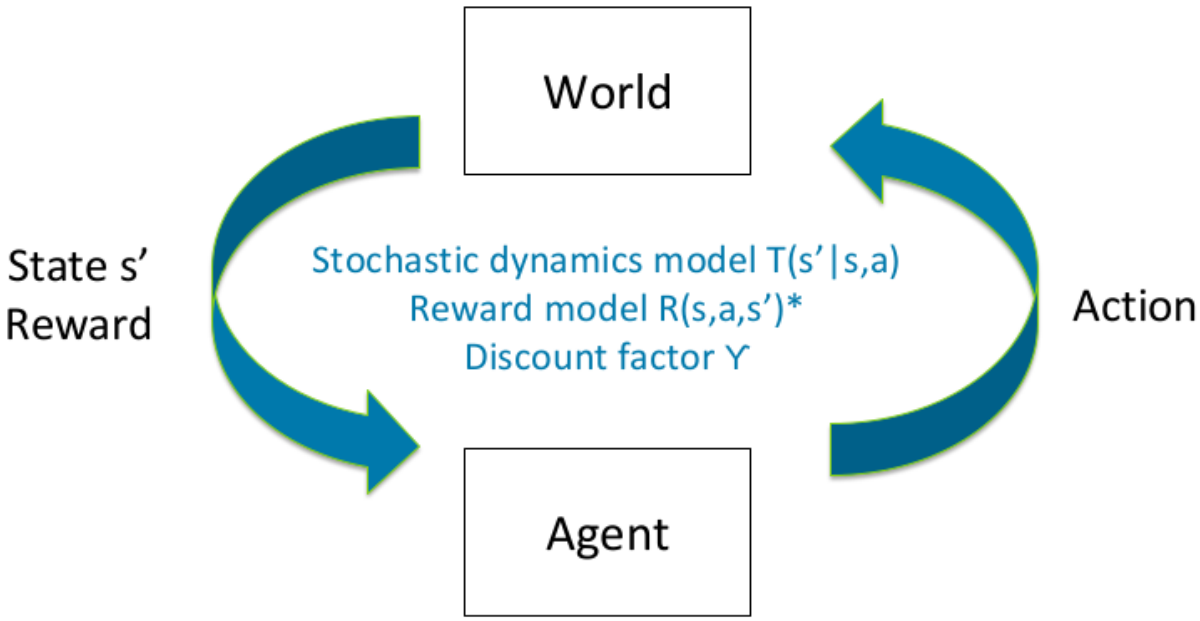
`Stanford` `RL` `Python`

- **Author: Yunqiu Xu**
- 课程总结:
  - 课程资源: http://web.stanford.edu/class/cs234/syllabus.html
  - 比较基础的RL教程, 偏重原理及Q learning部分, 对PG, AC等一带而过
  - 我之前上了 *UNSW COMP9444* , 对DL以及RL有一定了解, 再看这门课巩固下RL的部分
  - 进阶: 之后可以看 *UCB CS249-112* 扩宽广度

---

1. Introduction

- RL involves:
  - Optimization
  - Generalization
  - Exploration
  - Delayed consequences
- We have known the difference between supervised learning, unsupervised learning and RL, here another notation "Imitation Learning" is introduced:
  - Compared with RL, supervised and unsupervised learning do not have exploratin and delayer consequences, but optimization and generalization
  - Imitation learning has optimization, generalization and delayer consequences
  - Assumes input demos of good policies $\rightarrow$ reduce RL to supervised
  - Benefits:
    - For SL, there has been good tools
    - Avoid exploration problem
    - Large data about outcomes of decisions
  - Limitations:
    - Expensive to capture
    - Limited by collected data
- Marcov Decision Process: $< S, A, R, T, \Upsilon >$



Policy mapping from state → action

- S : states set
- A : action set
- R : reward function, e.g. $r_t = R(s_t, a_t)$
- T : dynamics model $p(s_{t+1}|s_t, a_t)$
  - **Markov property**: an action only depends on current state instead of all previous states
  - i.e. $p(s_{t+1}|s_1, a_1, s_2, \ldots, s_t, a_t) = p(s_{t+1}|s_t, a_t)$
- Υ : discount factor

- Methods of optimality:
  - Finite horizon reward: $\sum_{i=0}^{h-1} r_{t+i}$, easy to compute
  - **Infinite discounted reward**: $\sum_{i=0}^{\infty} \gamma^i r_{t+i}$, easy to prove theorem
  - Average reward: $lim_{h\to\infty} \frac{1}{h} \sum_{i=0}^{h-1} r_{t+i}$, hard to deal with
- Policy $\pi^* : S \to A$, method to choose action given current state
  - Value of policy $V^{\pi}(s)$: start from s, the expected discount sum of rewards if choose actions based on π

$$V^{\pi}(s) = E\left[\sum_{i=0}^{\infty} \gamma^i R(s_i, \pi(s_i))|s_0 = s\right]$$

  - Optimal policy π^*: $V^{\pi^*}(s)$ is the maximum expected discounted reward starting from s
  - **Bellman equation** : decompose the value function by the sum of current reward and future rewards based on Markov property:

$$V^{\pi}(s) = max\left[R(s, \pi(s)) + \gamma \sum_{s' \in S} p(s'|\pi(s), s) V^{\pi}(s')\right]$$

- Q learning:

- Q learning can be seen a variation of value based learning
- Difference between TD-learning and Q-learning:
  - TD: $S \rightarrow V$
  - Q: $S \times A \rightarrow Q$
- Change Bellman equation:

$$Q(s,a) = R(s,a) + \gamma \sum_{s' \in S} p(s'|a,s) V^{\pi}(s')$$

- TD-learning V.S. Q-learning from COMP9444
  - TD Determistic: $V(s_t) \leftarrow r_t + \gamma V(s_{t+1})$
  - TD Stocastic: $V(s_t) \leftarrow V(s_t) + \eta[r_t + \gamma V(s_{t+1}) - V(s_t)]$
  - Q Determistic: $Q(s_t, a_t) \leftarrow r_t + \gamma \max_{a' \in A} Q(s_{t+1}, a')$
  - Q Stocastic: $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \eta\left[r_t + \gamma \max_{a' \in A} Q(s_{t+1}, a') - Q(s_t, a_t)\right]$

---

2. From MDP Planning to RL Basics

- Last course : value iteration via Bellman backup

1. Initialize $V_0(s_i)=0$ for all states s_i,
2. Set k=1
3. Loop until [finite horizon, convergence]
   - For each state s,

$$V_{k+1}(s) = \max_a \left[r(s,a) + \gamma \sum_{s' \in S} p(s'|a,s) V_k(s') \right]$$

4. Extract Policy

Bellman backup

2.1 Value-based V.S. Policy-based

Policy Iteration	Value Iteration
Fewer Iterations	More iterations
More expensive per iteration	Cheaper per iteration

- Value-based learning is sure to converge, but ...
- Weakness of value-based learning:
  - Delayed reinforcement
  - Finite search space
  - Need to generalizatin in real environment (lookup table is insufficient)
- Policy iteration:
  - Compute infinite horizon value of a policy
  - Update the parameters of policy
  - Policy gradient

2.2 Policy iteration

1. i=0; Initialize $\pi_0(s)$ randomly for all states s
2. Converged = 0;
3. While i == 0 or $|\pi_i - \pi_{i-1}| > 0$
   - i=i+1
   - Policy evaluation V^{π}
   - Policy improvement

- Policy evaluation:

Policy Iteration Can Take At Most |A|^|S| Iterations (Size of # Policies)

1. i=0; Initialize $\pi_0(s)$ randomly for all states s
2. Converged = 0;
3. While i == 0 or $|\pi_i - \pi_{i-1}| > 0$
   - i=i+1
   - Policy evaluation: Compute V^{π}
   - Policy improvement:

$$Q^{\pi_i}(s,a) = r(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) V^{\pi_i}(s')$$

$$\pi_{i+1}(s) = \arg\max_a Q^{\pi_i}(s,a)$$

- Police improvement:
  - Compute Q value of different 1st action and then following π_i

$$Q^{\pi_i}(s,a) = r(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) V^{\pi_i}(s')$$

  - Use to extract a new policy

$$\pi_{i+1}(s) = \arg\max_a Q^{\pi_i}(s,a)$$

$$Q^{\pi_i}(s,a) = r(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) V^{\pi_i}(s')$$

$$\max_a Q^{\pi_i}(s,a) \geq V^{\pi_i}(s)$$

$$\pi_{i+1}(s) = \arg\max_a Q^{\pi_i}(s,a)$$

- So if take $\pi_{i+1}(s)$ then followed π_i forever,
  - expected sum of rewards would be at least as good as if we had always followed π_i
- But new proposed policy is to always follow π_{i+1} …

- An easier version in COMP9444: PG for epidemic task

for each trial
    run trial and collect states s_t, actions a_t, and reward r_{total}
    for $t = 1$ to length(trial)
        $\theta \leftarrow \theta + \eta(r_{\text{total}} - b)\nabla_\theta \log \pi_\theta(a_t|s_t)$
    end
end

2.3 TD-learning and Q-learning

- From last course we have compared the difference between TD and Q, following is an example of TD-learning:

$$V^{\pi}(s) = r(s, \pi(s)) + \gamma \sum_{s' \in S} p(s'|s, \pi(s)) V^{\pi}(s')$$

- Maintain estimate of $V^{\pi}(s)$ for all states
  - Update $V^{\pi}(s)$ each time after each transition (s, a, s', r)
  - Likely outcomes s' will contribute updates more often
  - Approximating expectation over next state with samples
  - Running average

Decrease learning rate over time (why?)

$$V_{samp}(s) = r + \gamma V^{\pi}(s')$$

$$V^{\pi}(s) = (1 - \alpha) V^{\pi}(s) + \alpha V_{samp}(s)$$

- 随着时间的推移, 我们要逐渐降低随机性

S1	S2	S3	S4	S5	S6	S7
Okay Field Site +1						Fantastic Field Site +10

- Policy: TryLeft in all states, use alpha = 0.5, γ=1
- Set V^{π}=[0 0 0 0 0 0 0],

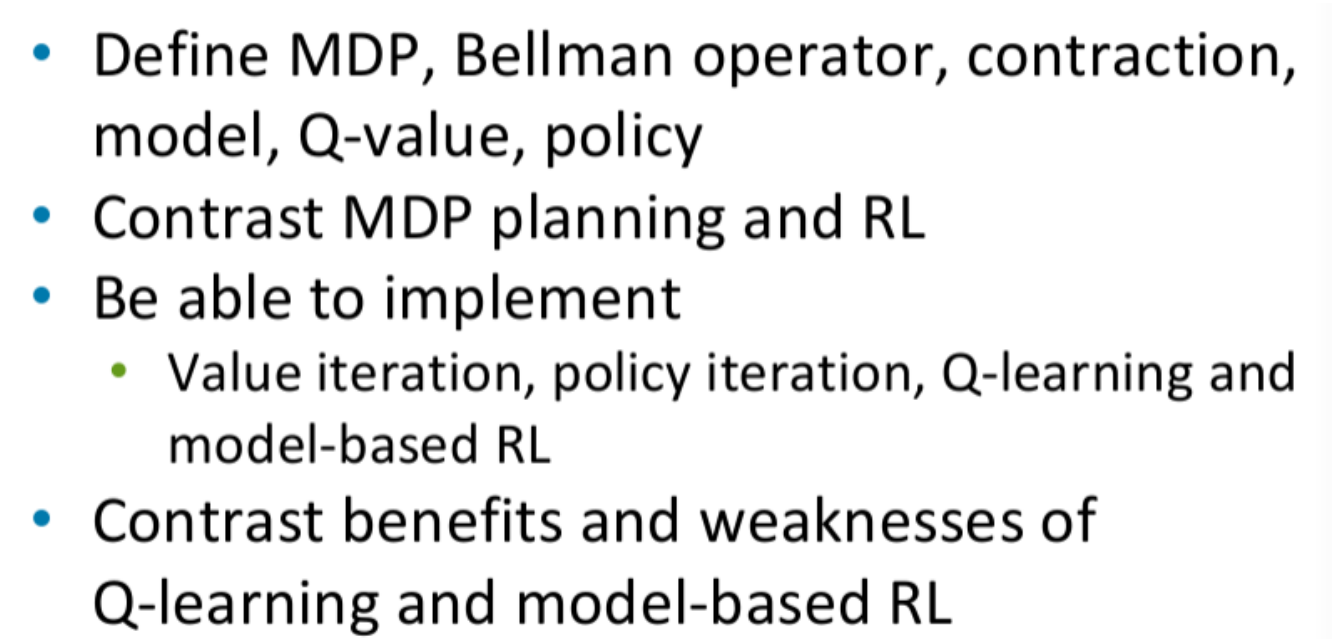
- Start: S3, TryLeft → S2
  - $V_{samp}(S3) = 0 + 1 * 0 = 0$
  - $V^{\pi}(S3) = (1 - 0.5) * 0 + 0.5 * 0 = 0$
- S2, TryLeft → S1
  - $V_{samp}(S2) = 0 + 1 * 0 = 0$
  - $V^{\pi}(S2) = (1 - 0.5) * 0 + 0.5 * 0 = 0$
- S1, TryLeft → S1
  - $V_{samp}(S1) = 1 + 1 * 0 = 1$
  - $V^{\pi}(S1) = (1 - 0.5) * 0 + 0.5 * 1 = 0.5 \Rightarrow [\,0.5\ 0\ 0\ 0\ 0\ 0\ 0\,]$

2.4 Explore V.S. Exploit

- E-greedy:

- With prob 1-e → argmax
- With prob e → random action
- Decay epsilon overtime, eventually will follow optimal policy all the time

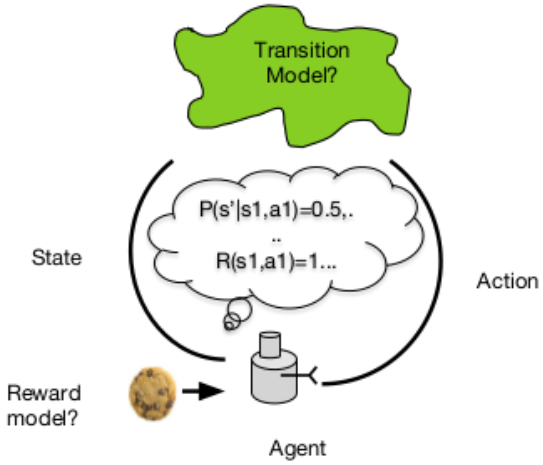
2.5 Conclusion



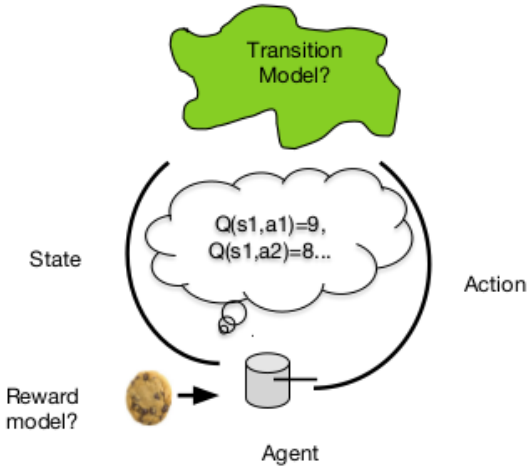
---

3. Monte Carlo and Generation

Model-based RL:
Agent estimates a reward & dynamics model… and then computes V/Q



Model-free RL:
Agent directly estimates Q / V



3.1 MC Methods: Model-free RL

- Update using Monte Carlo or TD-learning
  - TD-learning
    - Updates V estimate after each (s,a,r,s') tuple
    - Uses biased estimate of V
  - MC
    - Unbiased estimate of V
    - Can only update at the end of an episode

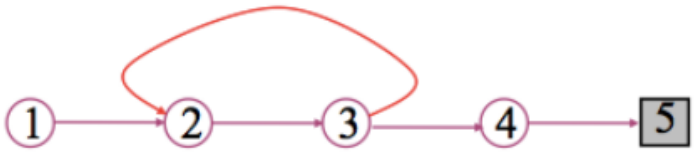
- MC methods: $Experience \rightarrow values, policy$
- Learn from complete sample returns, rrgard value as mean return
  - This course will only consider episodic tasks (MC 是回合更新)
  - No bootstrapping, all episodes must terminate
- Can be used in 2 ways:
  - Model free: most useful when
  - Simulated: 可以复习下COMP9414中simulated和situated的区别
- Advantages:
  - Can learn directly from interaction with environment
  - No need for full models
  - No need to learn about ALL states (no bootstrapping)

    - Less harmed by violating Markov property
- Issue: maintaining sufficient exploration ⇒ exploring starts, soft policies

3.2 MC Policy Evaluation

- Learn $v_\pi(s)$ from episodes of exp based on policy π
    - 之前返回的total discounted reward然后取期望值, 对于MC则使用empirical mean return

Idea: Average returns observed after visits to s:



    - 2 visit ways:
        - Every-Visit MC: collect and average returns for every time s is visited in an ep
        - First-Visit MC: collect and average returns for the first time s is visited in an ep
- Every Visit
    - To evaluate state s
    - Every time-step t that state s is visited in an episode,
    - Increment counter: $N(s) \leftarrow N(s) + 1$
    - Increment total return: $S(s) \leftarrow S(s) + G_t$
    - Value is estimated by mean return $V(s) = S(s)/N(s)$
    - By law of large numbers $V(s) \rightarrow v_\pi(s)$ as $N(s) \rightarrow \infty$
- First Visit

- To evaluate state s
- The first time-step t that state s is visited in an episode,
- Increment counter: $N(s) \leftarrow N(s) + 1$
- Increment total return: $S(s) \leftarrow S(s) + G_t$
- Value is estimated by mean return $V(s) = S(s)/N(s)$
- By law of large numbers $V(s) \rightarrow v_\pi(s)$ as $N(s) \rightarrow \infty$

- An example

S1	S2	S3	S4	S5	S6	S7
Okay Field Site +1						Fantastic Field Site +10

  - Policy: TryLeft (TL) in all states, use γ=1, H=4
  - Start in state S3, take TryLeft, get r=0, go to S2
  - Start in state S2, take TryLeft, get r=0, go to S2
  - Start in state S2, take TryLeft, get r=0, go to S1
  - Start in state S1, take TryLeft, get r=+1, go to S1
  - Trajectory = (S3,TL,0,S2,TL,0,S2,TL,0,S1,TL,1,S1)
  - First visit MC estimate of all states?
  - Every visit MC estimate of S2? $G_t = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{T-1} R_T$
    - Every-Visit MC: average returns for every time s is visited in an episode
    - First-visit MC: average returns only for first time s is visited in an episode
  - TD estimate of all states (init at 0)

$$V_{samp}(s) = r + \gamma V^\pi(s')$$

$$V^\pi(s) = (1-\alpha)V^\pi(s) + \alpha V_{samp}(s)$$

- Incremental MC Updates
  - Incremental mean

$$\mu_k = \frac{1}{k}\sum_{j=1}^{k} x_j = \mu_{k-1} + \frac{1}{k}(x_k - \mu_{k-1})$$

  - Update

- Update V(s) incrementally after episode $S_1, A_1, R_2, ..., S_T$
- For each state S_t with return G_t

$$N(S_t) \leftarrow N(S_t) + 1$$
$$V(S_t) \leftarrow V(S_t) + \frac{1}{N(S_t)}(G_t - V(S_t))$$

- In non-stationary problems, it can be useful to track a running mean, i.e. forget old episodes.

$$V(S_t) \leftarrow V(S_t) + \alpha(G_t - V(S_t))$$

3.3 MC for Q value

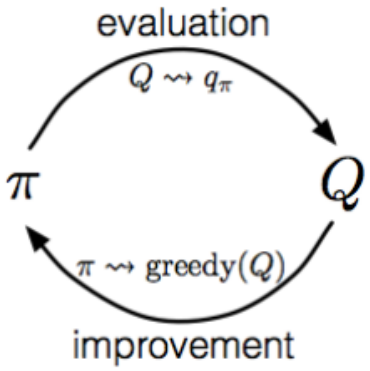
- q_π(s,a) - average return starting from state s and action a following π

$$\begin{aligned} q_\pi(s,a) &= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a] \\ &= \sum_{s',r} p(s',r|s,a)\Big[r + \gamma v_\pi(s')\Big]. \end{aligned}$$

- Converges asymptotically if every state-action pair is visited
- Exploring starts: Every state-action pair has a non-zero probability of being the starting pair

- MC control

$$\pi_0 \xrightarrow{\text{E}} q_{\pi_0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} q_{\pi_1} \xrightarrow{\text{I}} \pi_2 \xrightarrow{\text{E}} \cdots \xrightarrow{\text{I}} \pi_* \xrightarrow{\text{E}} q_*$$



- MC policy iteration step: Policy evaluation using MC methods followed by policy improvement
- Policy improvement step: greedify with respect to value (or action-value) function

- Greedy policy
  - For any action-value function q, the corresponding greedy policy is the one that:
    - For each s, deterministically chooses an action with maximal action-value:

$$\pi(s) \doteq \arg\max_a q(s,a).$$

  - Policy improvement then can be done by constructing each π_{k+1} as the greedy policy with respect to $q_{\pi k}$.

- On-policy MC control
  - Learn about policy currently executing
  - Policy is eternally soft: for all s and a, $\boldsymbol{\pi(a|s) > 0} \Rightarrow$ get rid of exploring starts

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
$Q(s,a) \leftarrow$ arbitrary
$Returns(s,a) \leftarrow$ empty list
$\pi(a|s) \leftarrow$ an arbitrary ε-soft policy

Repeat forever:
(a) Generate an episode using π
(b) For each pair s, a appearing in the episode:
$G \leftarrow$ return following the first occurrence of s, a
Append G to $Returns(s,a)$
$Q(s,a) \leftarrow \text{average}(Returns(s,a))$
(c) For each s in the episode:
$A^* \leftarrow \arg\max_a Q(s,a)$
For all $a \in \mathcal{A}(s)$:

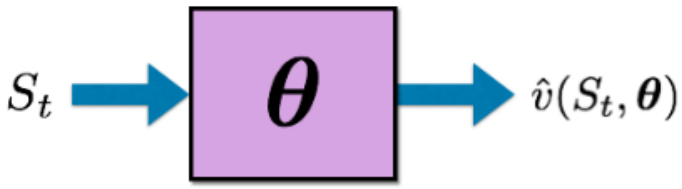
$$\pi(a|s) \leftarrow \begin{cases} 1-\varepsilon+\varepsilon/|\mathcal{A}(s)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(s)| & \text{if } a \neq A^* \end{cases}$$

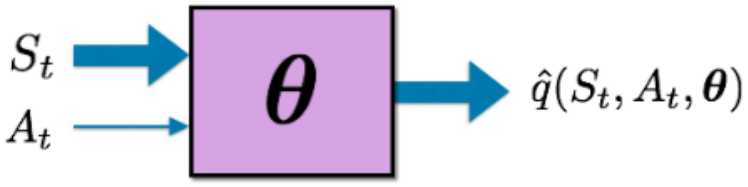
3.4 Generalization

- Why:
  - Smoothness assumption
  - More denerally, dim reduction / compression
  - Reduce memory / compute / experience
- Key idea of function approximation: replace lookup table with a function

Value Function Approximation (VFA)

- Value function approximation (VFA) replaces the table with a general parameterized form:





Linear Value Function Approximation (VFA)

- Represent value function by a linear combination of features

$$\hat{v}(S, \mathbf{w}) = \mathbf{x}(S)^\top \mathbf{w} = \sum_{j=1}^{n} \mathbf{x}_j(S)\mathbf{w}_j$$

- Objective function is quadratic in parameters w

$$J(\mathbf{w}) = \mathbb{E}_\pi \left[(v_\pi(S) - \mathbf{x}(S)^\top \mathbf{w})^2 \right]$$

- Update rule is particularly simple

$$\nabla_{\mathbf{w}} \hat{v}(S, \mathbf{w}) = \mathbf{x}(S)$$

$$\Delta \mathbf{w} = \alpha(v_\pi(S) - \hat{v}(S, \mathbf{w}))\mathbf{x}(S)$$

- Update = step-size × prediction error × feature value
- Later, we will look at the neural networks as function approximators.

- MC / TD / Control with VFA P38 - 43
- Other form of VFA: Action-VFA / Linear Action-VFA
- SGD with experience replay: 可以顺便复习下9444第三次作业

- Given experience consisting of ⟨state, value⟩ pairs

$$\mathcal{D} = \{\langle s_1, v_1^\pi \rangle, \langle s_2, v_2^\pi \rangle, \ldots, \langle s_T, v_T^\pi \rangle\}$$

- Repeat
  - Sample state, value from experience

$$\langle s, v^\pi \rangle \sim \mathcal{D}$$

  - Apply stochastic gradient descent update

$$\Delta \mathbf{w} = \alpha(v^\pi - \hat{v}(s, \mathbf{w}))\nabla_{\mathbf{w}} \hat{v}(s, \mathbf{w})$$

- Converges to least squares solution

---

4. Recap and More on Model Free Methods and Approximation

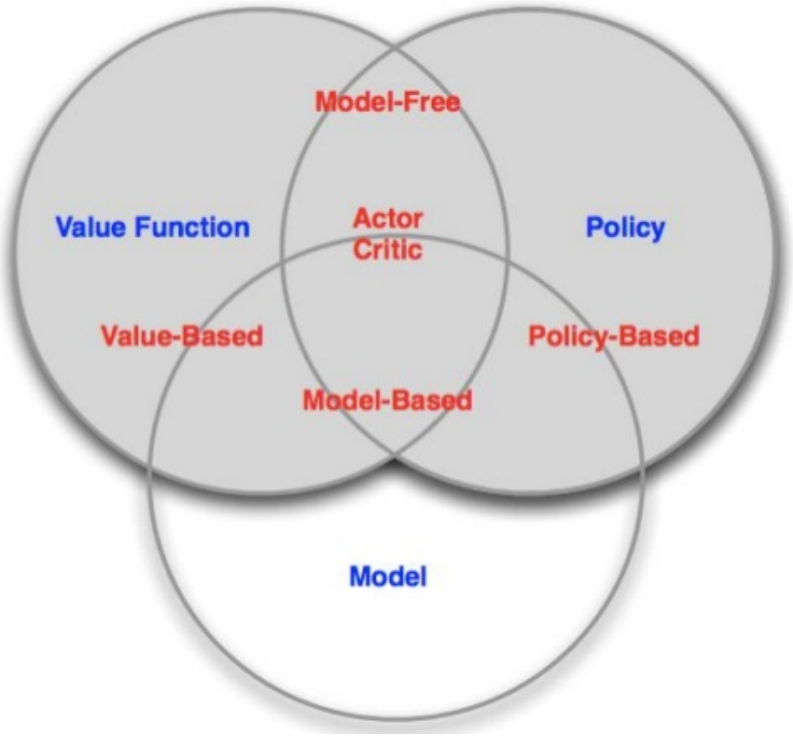
- 感觉这周主要是对之前的复习

4.1 MDP planning:

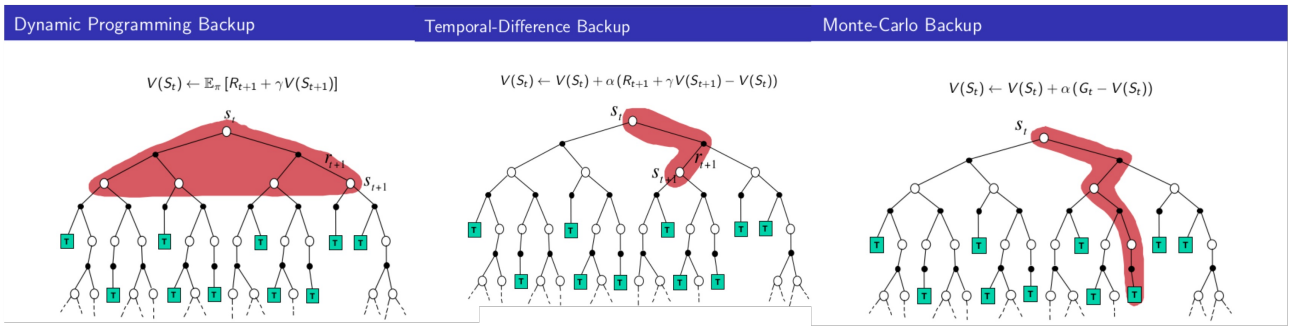
- Bellman equation
- Value iteration
- Policy iteration

4.2 RL with lookup table

- Model-based and model-free
  - Model based: estimate dynamics & reward & do MDP planning
  - Model free: Q-learning & MC evaluation



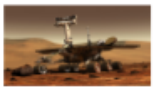
- Different kinds of Backup



- Bootstraping and Sampling
  - Bootstraping: update involves an estimate
  - Sampling: update samples an expectation

Method	Bootstrap	Sample
MC	N	Y
DP	Y	N
TD	Y	Y

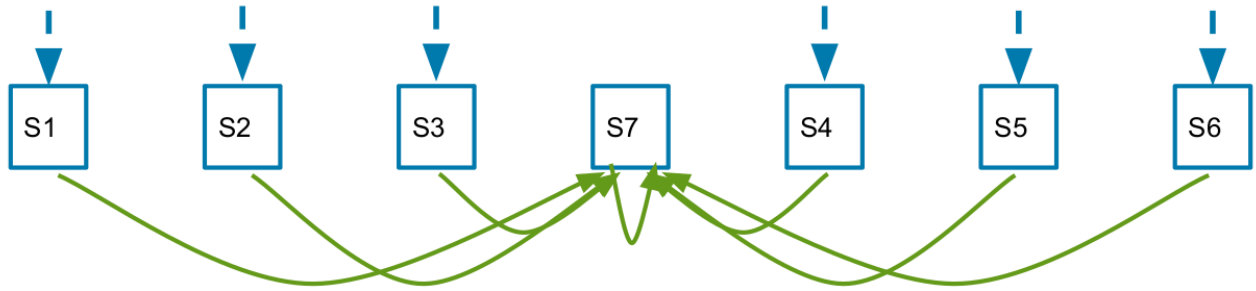
- An example:

S1	S2	S3	S4	S5	S6	S7
Okay Field Site +1						Fantastic Field Site +10

- Policy: TryLeft (TL) in all states, use (discount) γ=1
- 1 episode yielded trajectory = (S3,TL,0,S2,TL,0,S2,TL,0,S1,TL,1,S1)
- First visit MC estimate of all states (assume 0 if unvisited) [1 1 1 0 0 0 0]
- TD estimate of all states (init at 0) [α 0 0 0 0 0 0]
- **What if did updates not once, but many times?**
  - Repeatedly do TD updates on past tuples (pick any tuple and do update)
  - Repeatedly do MC on past episodes (pick any episode and do update)
- **What would be the resulting MC estimate in this case?**
- **What about the TD estimate? Try doing TD on the following tuples:**
  - **(**S3,TL,0,S2,TL,0,S2,TL,0,S1,TL,1,S1) (see solution above, start from there)
  - **then** (S2,TL,0,S2), (S1,TL,1,SL), (S1,TL,1,SL), (S1,TL,1,SL),
  - What are the resulting TD estimates?
- **Bonus: Does the order in which one samples prior (s,a,r,s') tuples impact the convergence rate of TD? If yes, what is a good order? Does it depend on alpha?**

4.3 Generalization in Model-free RL

- Linear VFA
- An example: Mars rover
  - 0 reward everywhere
  - Slightly different dynamics
- MC with VFA

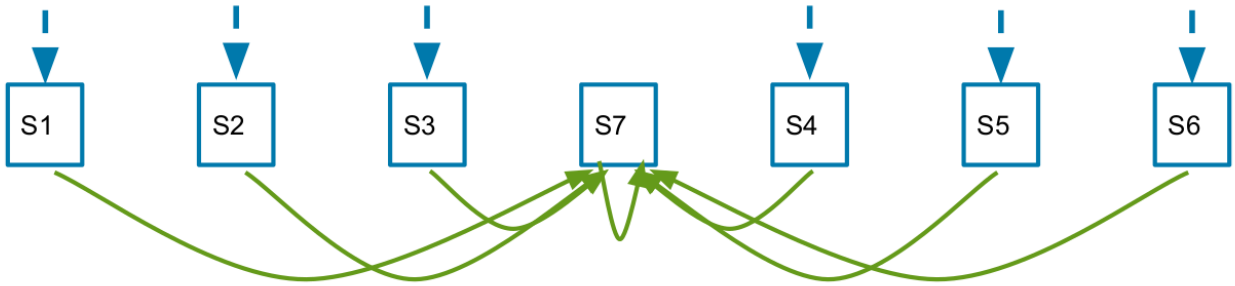


- a1 takes to states S1...S6 with probability 0.99/6
- a2 goes to state S7 with probability .99
- with prob 0.01 go to a terminal state s8 & episode ends
- Reward is 0 everywhere
- $\tilde{V}(s) = \sum_i w_i f_i(s)$, Gradient of $\tilde{V}$ wrt $w_i = f_i(s)$
- Policy is always take a2.
- One episode: act for 100 timesteps
- Observe [s1,a2,0,s7,a2,0,s7,... 0]
- Consider feature representation of 8 features
  - State s1 = [2 0 0 0 0 0 0 0 1] …
  - State s6 = [0 0 0 0 0 0 2 0 1]
  - State s7 = [0 0 0 0 0 0 0 1 2].
- Assume initial **w** = [1 1 1 1 1 1 1 1]
- Can true value of V^{π} be represented with these linear features (e.g. with $\tilde{V}$)?
- What is initial $\tilde{V}(s1)$?
- What is new weight vector using MC with VFA?
- Imagine we get the same trajectory many times. What will we converge to?

MC with VFA
Generate episode of length T
for t=0,1,...T-1

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha[G_t - \tilde{V}(s_t,\mathbf{w})]\mathbf{f}(s_t)$$

- TD with VFA



- a1 takes to states S1...S6 with probability 0.99/6
- a2 goes to state S7 with probability .99
- with prob 0.01 go to a terminal state s8 & episode ends
- Reward is 0 everywhere
- $\tilde{V}(s) = \Sigma_i w_i f_i(s)$, Gradient of V wrt $w_i = f_i(s)$
- Policy is always take a2.
- One episode: act for 100 timesteps
- Observe [s1,a2,0,s7,a2,0,s7,... 0]
- Consider feature representation of 8 features
  - State s1 = [2 0 0 0 0 0 0 1] …
  - State s6 = [0 0 0 0 0 0 2 0 1]
  - State s7 = [0 0 0 0 0 0 0 1 2].
- Assume initial **w** = [1 1 1 1 1 1 1], γ=1
- What is new weight vector using TD learning with VFA at end of episode?
- Imagine we get the same trajectory many times. What will we converge to?

TD(0) with VFA
For each episode
Initialize S
Repeat for each step of the episode
Choose a, observe r, s'
$\mathbf{w} \leftarrow \mathbf{w} + \alpha[r + \gamma \tilde{V}(s') - \tilde{V}(s,\mathbf{w})]\mathbf{f}(s)$

---

5. DNN

- 这周主要讲一些DL的基础, 已在其他课程进行学习, 可以快速过一遍

---

6. CNNs and DQN

- CNN 已在其他课程进行学习, 这里快速过一遍, 着重看DQN

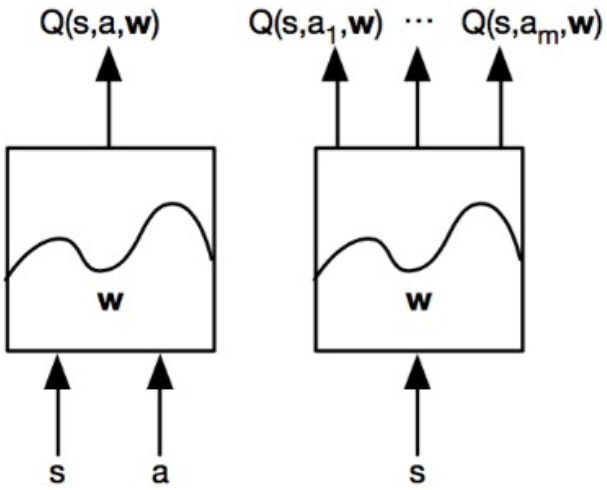
6.1 Basic DQN

- Why use DNN : for real world, there are too many states that lookup table may not be sufficient → we need generalization
- For Q-learning, we represent optimal Q-values as Bellman equation:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \eta\left[r_t + \gamma \max_{a' \in A} Q(s_{t+1}, a') - Q(s_t, a_t)\right]$$

  - In DQN we represent value function by Q-network with weights w:

$$Q(s,a,\mathbf{w}) \approx Q^*(s,a)$$



- Thus we can update weights by minimize MSE loss by SGD

$$\left[r_t + \gamma \max_{a' \in A} Q_w(s_{t+1}, a') - Q_w(s_t, a_t)\right]^2$$

- Some weakness $rightarrow$ experience replay
  - Correlations between samples
  - Non-stationary targets

6.2 DQN with Experience Replay

- Why use : there can be temporal correlations between samples, if we choose similar samples, it's bad for learning
- How to use :
  - build dataset from experience $< s, a, r, s' >$
  - sample experiences from dataset to update weights

s_1, a_1, r_2, s_2
s_2, a_2, r_3, s_3
s_3, a_3, r_4, s_4
...
$s_t, a_t, r_{t+1}, s_{t+1}$

$\rightarrow$ s, a, r, s'

  - To deal with non-stationarity, the weights of target net (w^-) are fixed
    - 在Nature版本的DQN中, 用了target net和eval net两个部分
    - Target net: calculate the one-step lookahead Q-Values, only updated after a certain number of episodes
    - 注意这个和DDQN不同, 这个相当于是用隔一定时间才更新的Target net来防止不稳定

$$I = \left[r + \gamma \max_{a' \in A} Q_{w^-}(s', a') - Q_w(s, a) \right]^2$$

$$\mathcal{L}_i(w_i) = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}_i} \left[\left(\underbrace{r + \gamma \max_{a'} Q(s', a'; w_i^-)}_{\text{Q-learning target}} - \underbrace{Q(s, a; w_i)}_{\text{Q-network}} \right)^2 \right]$$

- Whole process:
  - Given experience consisting of ⟨state, value⟩, or <state, action/value> pairs

$$\mathcal{D} = \{\langle s_1, v_1^\pi \rangle, \langle s_2, v_2^\pi \rangle, \ldots, \langle s_T, v_T^\pi \rangle\}$$

  - Repeat
    - Sample state, value from experience

$$\langle s, v^\pi \rangle \sim \mathcal{D}$$

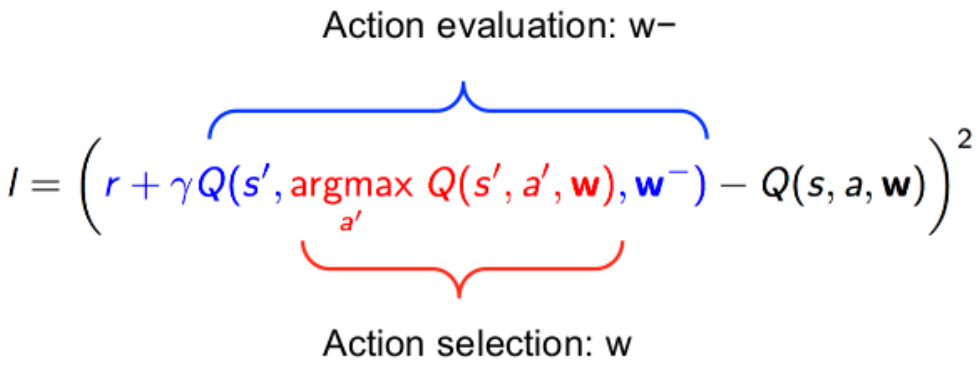
    - Apply stochastic gradient descent update

$$\Delta \mathbf{w} = \alpha(v^\pi - \hat{v}(s, \mathbf{w})) \nabla_{\mathbf{w}} \hat{v}(s, \mathbf{w})$$

- Some examples:

Game	Linear	Deep netowrk	DQN with fixed Q	DQN with replay	DQN with replay and fixed Q
Breakout	3	3	10	241	317
Enduro	62	29	141	831	1006
River Raid	2345	1453	2868	4102	7447
Sequest	656	275	1003	823	2894
Space Invaders	301	302	373	826	1089

6.3 Double DQN



- Why use: 若使用相同的网络(权重)用于选择动作和评估动作 $rightarrow$ confirmation bias (over estimation), 即Q值过大
- How to use:
  - Build 2 independent networks: Q_{new} is used to choose action, Q_{old} is used to evaluate action
  - Sample Q_new and Q_old randomly to be updated on each step

  $$Q_{new}(s_t, a_t) \leftarrow Q_{new}(s_t, a_t) + \eta\left[r_t + \gamma Q_{old}(s_{t+1}, argmax_{a' \in A} Q_{new}(s_{t+1}, a')) - Q_{new}(s_t, a_t)\right]$$
- Persudo code: here Q_1 is Q_{new} and Q_2 is Q_{old}

  Initialize $Q_1(s,a)$ and $Q_2(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily
  Initialize $Q_1(terminal\text{-}state, \cdot) = Q_2(terminal\text{-}state, \cdot) = 0$
  Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
  Choose A from S using policy derived from Q_1 and Q_2 (e.g., ε-greedy in $Q_1 + Q_2$)
  Take action A, observe R, S'
  With 0.5 probabilility:
  $$Q_1(S,A) \leftarrow Q_1(S,A) + \alpha\Big(R + \gamma Q_2\big(S', \text{argmax}_a\, Q_1(S',a)\big) - Q_1(S,A)\Big)$$
  else:
  $$Q_2(S,A) \leftarrow Q_2(S,A) + \alpha\Big(R + \gamma Q_1\big(S', \text{argmax}_a\, Q_2(S',a)\big) - Q_2(S,A)\Big)$$
  $S \leftarrow S'$;
  until S is terminal

---

7. Deel RL Continue

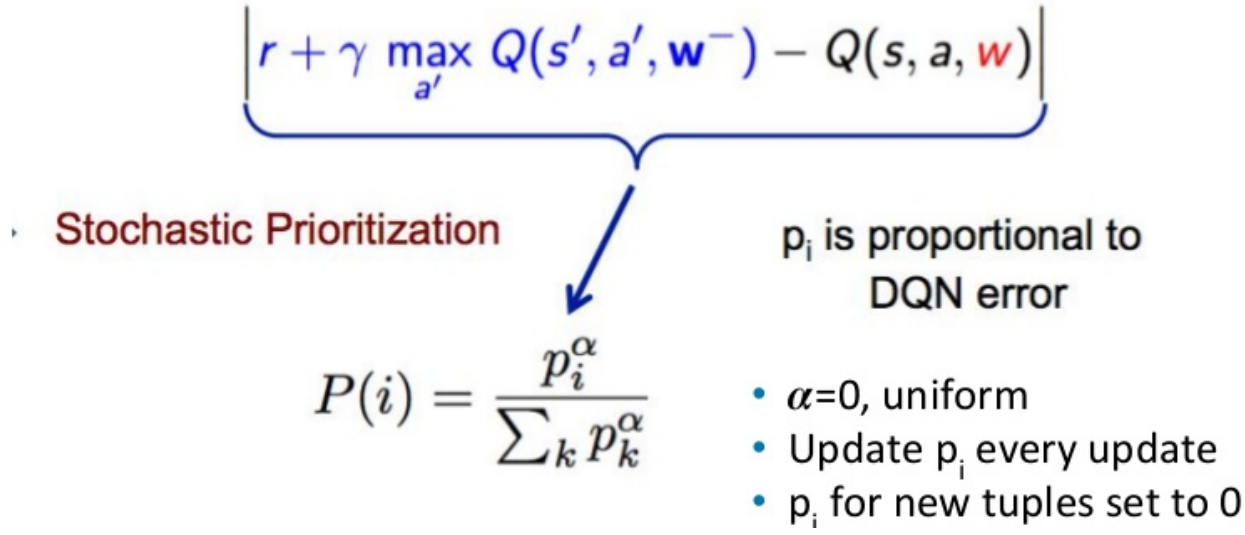
- In last course we discussed DQN, DQN with experience replay, fixed target weights and DDQN

7.1 DQN with Prioritized Experience Replay

- Why use: sometimes we can weight the importance of the experience, this will be useful when the

number of positive and negative samples is inbalance, e.g. MountainCar

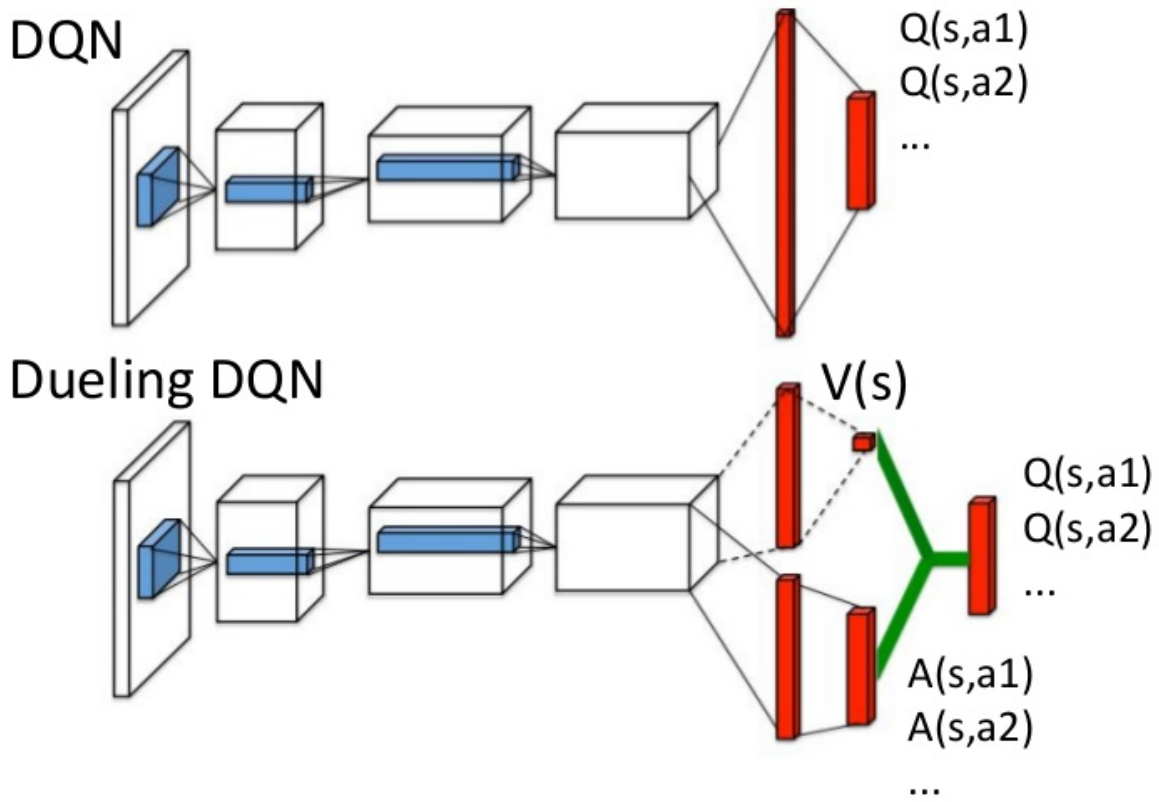
- How to use: sample experience using priority, where the priority is proportion to error, i.e. pay attention to those "more prone to being wrong"



7.2 Dueling DQN with Advantage Function

- Why use: Features need to pay attention to determine value may be different than those need to determine action benefit
- How to use: replace Q function with advantage function, which is to represent the advantage or disadvantage to take an action a at state s, comparing to taking the action preferred by current policy

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



- Identifiability:
  - Unidentifiable
  - Option 1: Force A(s,a) = 0 for action a taken from

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + \left(A(s,a;\theta,\alpha) - \max_{a' \in |A|} A(s,a';\theta,\alpha)\right)$$

  - Option 2: Use mean as baseline (**more stable**)

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + \left(A(s,a;\theta,\alpha) - \frac{1}{|A|}\sum_{a'} A(s,a';\theta,\alpha)\right)$$

7.3 Summary of Model-free DQN and Practice Tips

- Model-free DQN:
  - Stabilize target
  - Prioritized experience replay
  - Advantage function
- Tips:
  - Pong can be a reliable task to test
  - Large replay buffers $\rightarrow$ more robust, memory efficient is key
  - DQN converges slowly, e.g. for ATARI maybe 10-40M frames to see awesome result
  - Try Huber loss on Bellman error

$$L(x) = \begin{cases} x^2/2 & \text{if } |x| \leq \delta \\ \delta|x| - \delta^2/2 & \text{otherwise} \end{cases}$$

- Try DDQN
- When experimenting, run at least 2 different seeds
- Try high learning rate in initial period
- Tty non-standard exploration schedules

7.4 TD V.S. MC

- Challenges of Target for Model Free RL

$$\mathbb{E}_{s,a,r,s' \sim \mathcal{D}_i}\left[\left(\underbrace{r + \gamma \max_{a'} Q(s', a'; w_i^-)}_{\text{Q-learning target}} - \underbrace{Q(s, a; w_i)}_{\text{Q-network}}\right)^2\right]$$

- Running stochastic gradient descent
- Ideally what should Q-learning target be?
  - Q(s,a)
  - But we don't know that
  - Could be use Monte Carlo estimate (sum of rewards to the end of the episode)

- TD V.S. MC P25-42, 这里加的David Silver的图, 需要了解两种方法的优缺点

- MC has high variance, zero bias
  - Good convergence properties
  - (even with function approximation)
  - Not very sensitive to initial value
  - Very simple to understand and use
- TD has low variance, some bias
  - Usually more efficient than MC
  - TD(0) converges to $v_\pi(s)$
  - (but not always with function approximation)
  - More sensitive to initial value

---

8. Exploration

- 首先还是继续上次课TD和MC的对比
- Tradeoff between exploration and exploitation
  - Exploration: try new things that maybe good
  - Exploitation: based on past good experiences
- Performance of RL algorithms
  - Convergence
  - Asymptotically optimal
  - Probably approximately correct (PAC)
  - Minimize / sublinear regret

8.1 PAC RL

- Given an input ε and δ, with probability at least 1-δ
- On all but N steps,
- Select action a for state s whose value is ε-close to V*
  $|Q(s,a) - V^*(s)| < \varepsilon$
- where N is a polynomial function of $(|S|, |A|, \delta, \varepsilon, \Upsilon)$

- Much stronger criteria
  - Bounding number of mistakes we make
  - Finite and polynomial

- Q-learning with e-Greedy is not PAC
  - Need eventually to be taking bad actions only small fraction of the time
  - Bad (random) action could yield poor reward on this and many future time steps
  - If want PAC MDP algorithm using e'-greedy exploration, need $e' < \varepsilon(1-\Upsilon)$
  - *Q-learning with optimistic initialization & learning rate = (1/t) and e'-greedy exploration is not PAC
    - Even though will converge to optima
    - Thm 10 in A.Strehl thesis 2007

- PAC RL Approaches:
  - Typically model-based or model-free
  - Formally analyze **how much experience** is needed to estimate a **good** Q-function to **achieve high reward** in real world

$p(s' \mid s, a)$ known $\rightarrow$ Compute ε-optimal policy

Bound $\left(\hat{p}(s' \mid s, a) - p(s' \mid s, a)\right) \rightarrow$ Bound error in policy calculated using $\hat{p}(s' \mid s, a)$

- How many samples are needed to build a PAC RL model

Sample complexity = # steps on which may not act well (could be far from optimal)

(R-MAX and E^3) = Poly(# of states)

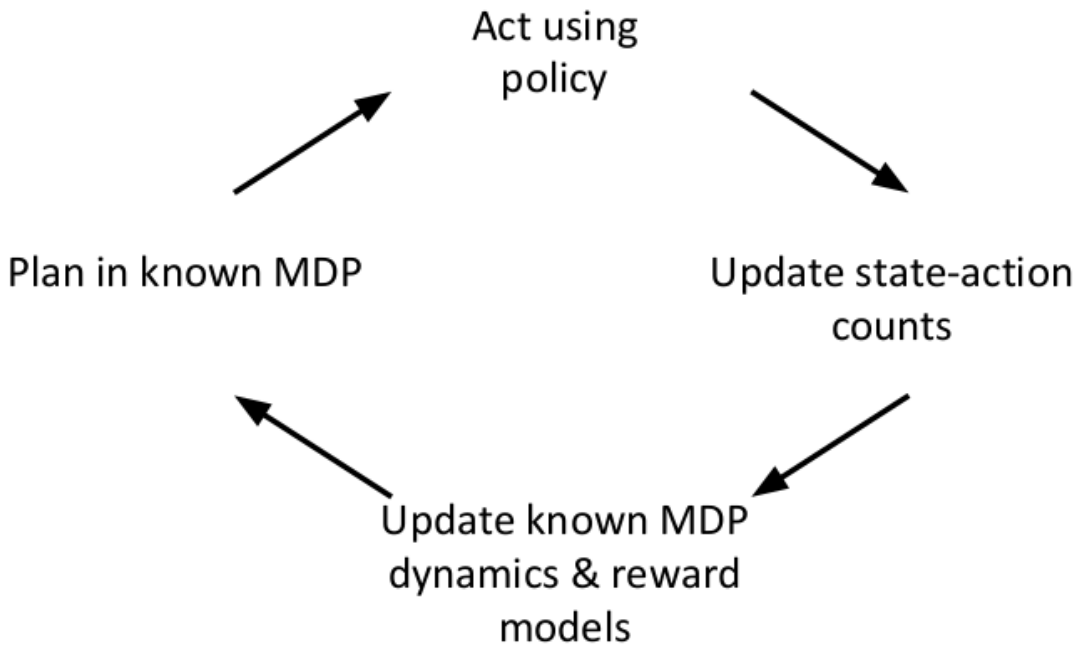
- Important ideas in PAC RL
  - Bound error over model estimates
    - Relate amount of samples to accuracy of parameters
  - Be optimistic with respect to model / Q uncertainty
    - Consider how world could be
    - Solve policy for that world
    - Act accordingly

8.3 Model-based RL: R-max algorithm

- Model based RL, repeat:
  - Given data seen so far
  - Build an explicit model of MDP
  - Compute policy for it

- For current state, select action based on policy, get next state and reward

- R-max algorithm



- An example:

Known/ Unknown

	S2	S2	S3	S4	…
↑	U	U	U	U	
→	U	U	U	U	
↓	U	U	U	U	
←	U	U	U	U	

Reward

	S2	S2	S3	S4	…
↑	R_{max}	R_{max}	R_{max}	R_{max}	
→	R_{max}	R_{max}	R_{max}	R_{max}	
↓	R_{max}	R_{max}	R_{max}	R_{max}	
←	R_{max}	R_{max}	R_{max}	R_{max}	

Transition Counts

	S2	S2	S3	S4	…
↑	0	0	0	0	
→	0	0	1	0	
↓	0	0	0	0	
←	0	0	0	0	

Increment counts for state-action tuple

Known/ Unknown

	S2	S2	S3	S4	…
↑	U	U	U	U	
→	U	U	K	U	
↓	U	U	U	U	
←	U	U	U	U	

Reward

	S2	S2	S3	S4	…
↑	R_{max}	R_{max}	R_{max}	R_{max}	
→	R_{max}	R_{max}	R	R_{max}	
↓	R_{max}	R_{max}	R_{max}	R_{max}	
←	R_{max}	R_{max}	R_{max}	R_{max}	

Transition Counts

	S2	S2	S3	S4	…
↑	3	3	4	3	
→	2	4	5	0	
↓	4	0	4	4	
←	2	2	4	1	

If counts for (s,a) > N, (s,a) becomes known: **use observed data to estimate transition & reward model** for (s,a) when planning

- Sample complexity:

$$\left(\frac{SA}{\varepsilon(1-\gamma)^2} \underbrace{\frac{S}{\varepsilon^2(1-\gamma)^4}} \right)$$

γ=.9, ε=.1
How many steps?

samples need per (s,a) pair

On all but the above number of steps, chooses action whose expected reward is close to expected reward of action take if knew model parameters, with high probability

---

9. Midterm review

---

10. Sample Efficiency

- Exploration is important
- Optimism under uncertainty can
  - Yield formal bounds on algorithm's performance
  - Have practical benefits
- Regret and PAC have some limitations, PAC-uniform is a new theoretical framework to get us closer to what we want in practice
- Still a large gap between bounds and practical performance

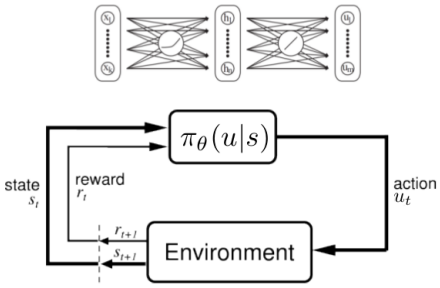
---

11. Sample Efficient Policy Search

- Super effective and popular
  - Robotics, vision, NLP, ….
- Gradient based approaches are most common
  - Find a local optima
  - Empirically often good enough
  - Can use complex models easily (neural nets)
- When data is expensive and policy class is small
  - Bayesian optimization can be very effective
- When data is expensive and large policy class
  - Can use importance sampling
  - For some important optimal stopping problems, can leverage reuse data with no penalty
  - But still open challenge

---

12. Policy Gradients



- Consider control policy parameterized by parameter vector θ

$$\max_\theta \quad \mathrm{E}[\sum_{t=0}^{H} R(s_t)|\pi_\theta]$$

- Often stochastic policy class (smooths out the problem):

  $\pi_\theta(u|s)$: probability of action u in state s

- Why use:
  - Often π can be simpler than Q or V
    - E.g., robotic grasp
  - V: doesn't prescribe actions
    - Would need dynamics model (+ compute 1 Bellman back-up)
  - Q: need to be able to efficiently solve $\arg\max_u Q_\theta(s,u)$
    - Challenge for continuous / high-dimensional action spaces*

- CS234把这部分作为客座, 可以看下COMP9444
  - **Model-based**
    - *Pathwise Derivatives (PD) / BackPropagation Through Time (BPTT)*
      - Deterministic dynamics
      - Stochastic dynamics / Reparameterization trick
      - Variance reduction (-> SVG, DDPG)

    Assumes:
    - f known, differentiable
    - R known, differentiable
    - π_θ (known), differentiable

  - **Model-free**
    - *Parameter Perturbation / Evolutionary Strategies*
    - *Likelihood Ratio (LR) Policy Gradient*
      - Derivation
      - Connection w/Importance Sampling
      - Variance reduction
      - Step-sizing / Natural Gradient / Trust Regions (TRPO)
      - Generalized Advantage Estimation (GAE) / Asynchronous Actor Critic (A3C)

    Assumes:
    - f -- no assumptions
    - R -- no assumptions
    - π_θ -- (known), stochastic

  - **Stochastic Computation Graphs:** general framework for PD / LR gradients