SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

01.112 Machine Learning, Spring 2018
Lecture Notes for Week 13

22. Reinforcement Learning (II)

Last update: Thursday 12th April, 2018 15:21

We have learned value iteration in the last class, where we derived the update functions based on the following:

$$V^*(s) = \max_a Q^*(s,a) = Q^*(s,\pi^*(s)) \quad (1)$$

$$Q^*(s,a) = \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')] \quad (2)$$

$$V^*(s) = \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')] \quad (3)$$

(Sub (2) into (1))

$$= \sum_{s'} T(s,\pi^*(s),s')[R(s,\pi^*(s),s') + \gamma V^*(s')] \quad (4)$$

Sub (1) into (2)

Alternatively, we could replace $V^*(s')$ in Eq (2) by the right-hand side of Eq (1). We arrive at the following recurrence function that involves Q-values only:

$$Q^*(s,a) = \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma \max_{a'} Q^*(s',a')] \quad (5)$$

The Q-Value Iteration Algorithm

- Start with $Q_0^*(s,a) = 0$ for all $s \in S, a \in A$.
- Given $Q_i^*(s,a)$, calculate the Q-values for all states (depth $i+1$) and for all actions a:

$$Q_{i+1}^*(s,a) \leftarrow \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma \max_{a'} Q_i^*(s',a')] \quad (6)$$

- Repeat the above step until convergence.

This algorithm has the same convergence guarantees as its value iteration counterpart. As before, the optimal policy can be easily recovered from the Q-values as:

$$\pi^*(s) = \arg\max_a Q^*(s,a) \quad (7)$$

Policy Iteration (optional)

Besides value iteration and Q-value iteration, there is also policy iteration algorithm that iteratively improves the policy directly. The algorithm is as follows.

- Randomly initialize the policy π.
- Find the values for the states under the policy π by solving the following system of linear equations:

$$V^{\pi}(s) = \sum_{s'} T(s, \pi(s), s') \Big(R(s, \pi(s), s') + \gamma V^{\pi}(s') \Big) \text{ for all } s. \tag{8}$$

  where $V^{\pi}(\cdot)$'s are unknowns, and all others variables have known values.

  This step is also called *policy evaluation*.

- Find the improved policy π'. For all s:

$$\pi'(s) \leftarrow \arg\max_{a} \sum_{s'} T(s, a, s') \Big(R(s, a, s') + \gamma V^{\pi}(s') \Big) \tag{9}$$

- If $\pi' = \pi$, return the final policy π (or π') as the optimal policy π^*. Otherwise, set $\pi = \pi'$ and repeat the above two steps until convergence ($\pi = \pi'$).

The algorithm involves finding the values for each state at each iteration, based on a particular policy π. This step is done via solving a system of linear equations, which can be performed analytically. The algorithm also comes with a guarantee: the policy always converges to *an* optimal policy eventually.

Reinforcement learning

Now, we will consider a set-up where neither reward nor transitions are known a priori. Our robot can travel in the grid, moving from one state to another, collecting rewards along the way. The model of the world is unknown to the robot other than the overall Markov structure. The robot could do one of two things. First, it could try to learn the model, the reward and transition probabilities, and then solve the optimal policy using the algorithms for MDPs described above. Another option is to try to learn the Q-values directly.

Model-based learning We first assume that we can collect information about transitions involving any state s and action a. Under this assumption, we can learn T and R through experience, by collecting outcomes for each s and a.

$$T(s, a, s') = \frac{\text{Count}(s, a, s')}{\text{Count}(s, a)} \tag{10}$$

$$R(s, a, s') = \frac{\sum_t R_t(s, a, s')}{\text{Count}(s, a, s')} \tag{11}$$

where $R_t(s, a, s')$ is the reward we observed (for the t-th time) when starting in state s, taking action a, and transitioning to s'. If the reward is noisy, observed rewards R_t may vary from one instance to another. In reality, this naive approach is highly ineffective for any non-trivial state space. The best we can do is randomly explore, taking actions and moving from one state to another. Most likely, we will be unable to reach many parts of the state space in any complex environment. Moreover, the learned model would be quite large as we'd have to store all the states and possible transitions.

Model-free Learning Can we learn how to act without learning a full model? Remember:

$$\pi^*(s) = \arg\max_a Q^*(s, a) \tag{12}$$

We have shown that Q-values can be learned recursively, assuming we have access to T and R. Since this information is not provided to us, we will consider Q-learning algorithm, a sample based Q-value iteration procedure.

To better understand the difference between model-based and model-free estimation, consider the task of computing the expected value of a function $f(x)$: $E[f(x)] = \sum_x p(x)f(x)$

- **Model-based computation**: First estimate $p(x)$ from samples and then compute expectation:

$$x_i \approx p(x), i = 1, \ldots, k \tag{13}$$

$$\hat{p}(x) = \frac{\text{Count}(x)}{k} \tag{14}$$

$$E[f(x)] \approx \sum_x \hat{p}(x) f(x) \tag{15}$$

- **Model-free estimation**: estimate expectation directly from samples

$$x_i \approx p(x), i = 1, \ldots, k \tag{16}$$

$$E[f(x)] \approx \frac{1}{k} \sum_{i=1}^{k} f(x_i) \tag{17}$$

Now we will apply the model-free learning approach to the estimation of Q-values. Recall,

$$Q^*_{i+1}(s, a) \leftarrow \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma \max_{a'} Q^*_i(s', a')] \tag{18}$$

We will repeatedly take one action at a time, and observe the reward and the next state. We can compute:

$$\text{Sample 1}: \quad R(s,a,s'_1) + \gamma \max_{a'} Q_i(s'_1, a') \tag{19}$$
$$\text{Sample 2}: \quad R(s,a,s'_2) + \gamma \max_{a'} Q_i(s'_2, a') \tag{20}$$
$$\dots \tag{21}$$
$$\text{Sample } k: \quad R(s,a,s'_k) + \gamma \max_{a'} Q_i(s'_k, a') \tag{22}$$

Now we can average all the samples, to obtain the Q-value estimate:

$$Q_{i+1}(s,a) \leftarrow \frac{1}{k} \sum_{l=1}^{k} \left[R(s,a,s'_l) + \gamma \max_{a'} Q_i(s'_l, a') \right] \tag{23}$$

which, for large k, would be very close to the Q-value iteration step. We are almost there. In practice, we only observe the states when we actually move. Therefore, we cannot really collect all these sample at once. Instead, we will update the Q-values after every experience (s, a, s', r), where r is the reward.

Assume we have observed $k-1$ samples related to (s,a) so far, and now we just observed the k-th sample and we would like to update the Q-value.

The update function is:

$$Q_{new}(s,a) \leftarrow \frac{(k-1)Q_{old}(s,a) + R(s,a,s'_k) + \gamma \max_{a'} Q_{old}(s'_k, a')}{k} \tag{24}$$

Simplifying the formula, we arrive at the following update function:

$$Q(s,a) \leftarrow Q(s,a) + \frac{1}{k} \left[R(s,a,s'_k) + \gamma \max_{a'} Q(s'_k, a') - Q(s,a) \right] \tag{25}$$

This fact leads to the following Q-learning algorithm.

Q-learning Algorithm

- Collect a sample: s, a, s' and $R(s,a,s')$.
- Update Q-values, by incorporating the new sample into a running average over samples:

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha \left[R(s,a,s') + \gamma \max_{a'} Q(s', a') \right] \tag{26}$$
$$= Q(s,a) + \alpha \left[R(s,a,s') + \gamma \max_{a'} Q(s', a') - Q(s,a) \right] \tag{27}$$

where the learning rate α takes the role of $1/k$ in the earlier sample average example. During the iterations of the algorithm, likely s' will contribute more often to the Q-values estimate. As the algorithm progresses, old estimates fade, making the Q-value more consistent with more recent samples.

You may have noticed that the form of the update closely resembles stochastic gradient decent. In fact, it has the same convergence conditions as the gradient ascent algorithm. Each sample corresponds to (s, a), *i.e.*, being in state s and taking action a. We can assign a separate learning rate for each such case, *i.e.*, $\alpha = \alpha_k(s, a)$, where k is the number of times that we saw (s, a). Then, in order to ensure convergence, we should have

$$\sum_k \alpha_k(s, a) \to \infty \tag{28}$$

$$\sum_k \alpha_k^2(s, a) < \infty \tag{29}$$

Obviously $\alpha_k(s, a) = 1/k$ satisfies the above two conditions.

Exploration/Exploitation Trade-Off In the Q-learning algorithm, we haven't specified how to select an action for a new sample. One option is to do it fully randomly. While this exploration strategy has a potential to cover a wide spectrum of possible actions, most likely it will select plenty of suboptimal actions, and leads to a poor exploration of the relevant (high reward) part of the state space. Another option is to exploit the knowledge we have already obtained during previous iterations. Remember that once we have Q estimates, we can compute a policy. Since our estimates are noisy in the beginning, and the corresponding policy is weak, we wouldn't want to follow this policy completely. To allow for additional exploration, we select a random action every once in a while. Specifically, with probability ϵ, the action is selected at random and with probability $1-\epsilon$, we follow the policy induced by the current Q-values. Over time, we can decrease ϵ, to rely more heavily on the learned policy as it improves based on the Q-learning updates.

Learning Objectives

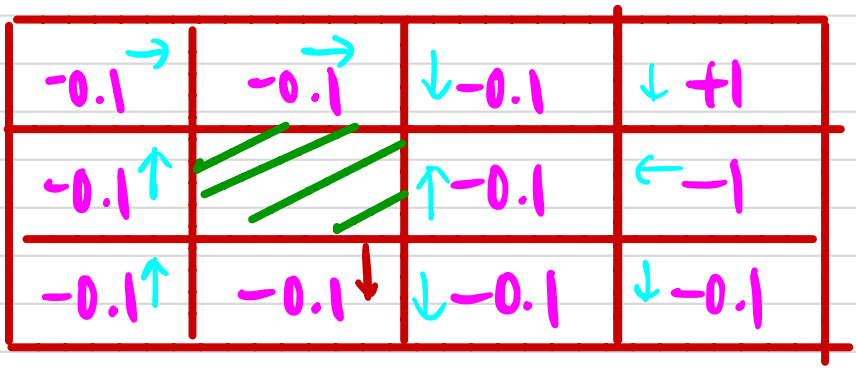
You need to know:

1. What is Q-value iteration and how to do Q-value iteration for a simple MDP problem
2. What is Q-learning and how to perform Q-learning for a simple reinforcement learning problem

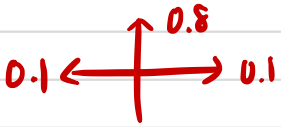
W12 D2 Recap (Markov Decisn Process (MDP))

A · A set of states S
B · A set of actions A → Reinforcem Learng
· Reward Functn : R(s,a,s') or R(s')
· Transition Probabilities: T(s,a,s') = P(s'|s,a)

Aim: Policy (Actn for ea state for all states)



stay on spot = 80%



Assumptn
Robot can determine its posn for all states.
↓
Partially Observable Markov Decisn

$\pi(s)$
$V^{\pi}(s)$: ē expected long-term reward for state s if we follow policy π
$Q^{\pi}(s)$: ē expected long-term reward for taking actn a at state s, and following policy π thereafter.

$$V^*(s) = \max_a Q^*(s,a) \quad (1)$$

$$= Q^*(s, \pi^*(s))$$ - Max Q value is due to actn as part of optimal policy

$$\pi^*(s) = \arg\max_a Q^*(s,a) \quad (2)$$

$$Q^*(s,a) = \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma \cdot V^*(s')\right] \quad (3)$$

- $\sum_{s'}$: a that leads from s → s'
- $R(s,a,s')$: Immediate Reward
- $V^*(s')$: Long-Term Reward from state s'

Haven't went thru 'policy initialization' algo.

Policy 1
$\pi_1(1,A) = \rightarrow$
$\pi_1(2,A) = \leftarrow$

Policy 2
$\pi_2(1,A) = \downarrow$
$\pi_2(2,A) = \downarrow$

A. Value Iteration Algo.

Steps
1. Set $V_0^*(s) = 0$ for all s
2. $V_{i+1}^*(s) \leftarrow \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma \cdot V_i^*(s')]$
3. Repeat (2) until convergence

To prevent backtracking → use shortcut.
→ Record a that gives us maximum. Each state s w/ max. a
- Each set of a stored will be optimal actions.

B. Q-value iteration. Algo

1. Set $Q_0^*(s,a) = 0$ for all s, a
   Sub Eqn (1) into (2)
2. $Q_{i+1}^*(s,a) = \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma \max_{a'} Q_i^*(s',a')]$
3. Repeat (2) until convergence.

C. Policy Iteration Algo (Optional)
└ Not tested.

$V^\pi(s) = \sum_{s'} T(s,\pi(s),s')[R(s,\pi(s),s') + \gamma \cdot V^\pi(s')]$

- $V^\pi(s)$: unknown; π: particular π
- $T(s,\pi(s),s')$: known
- $R(s,\pi(s),s')$: known
- γ: known

1 state, 1 eqn
n state, n eqn, n unknown (Use Gaussian Elimination).

Step: Find expected long-term reward. Compute for n states. Then use Gaussian Elimination.

W13 01 - Reinforcement Learning (Only give states & actions)
- Robot just take actn from current state.
→ Estimates $T(s,a,s')$ and $R(s,a,s')$

Be at state s and repeat a. → $T(s,a,s') = \frac{\text{count}(s,a,s')}{\text{count}(s,a)}$

$$R(s,a,s') = \frac{1}{k}\sum_{k=1}^{k} r_k(s,a,s')$$

OR

$$= \frac{\sum_t R_t(s,a,s')}{\text{count}(s,a,s')}$$

— reward differs by 0.1 due to sensor problem.

Not feasible for robot to repeat actn a at states multiple times.

↳ Modal-based approach.

Better Soln

Model-Free Approach

Gambler → H: +20 , T: −10

1.	H	+20	
2.	H	+20	$= \frac{20(1)+20}{2}$
3.	T	+10	$= \frac{20\times2+(-10)}{3}$
	T	+5	$= \frac{10\times3+(-10)}{4}$
	H		
	T		
	T		
	T		
(k−1)	T		V
k	T		$\frac{V(k-1)+(-10)}{k}$

$P(H) = 0.3 \quad (3/10)$

$P(T) = 0.7 \quad (7/10)$

$P(H)(20) + P(T)(-10)$

$= 0.3(20) + (0.7)(10)$

$= -1$

Sample 1: $R(s,a,s_1') + \gamma \max_{a'} Q_i(s_1', a')$

2: $R(s,a,s_2') + \gamma \max_{a'} Q_i(s_2', a')$

$\vdots$

k: $R(s,a,s_k') + \gamma \cdot \max_{a'} Q_i(s_k', a')$

$$\rightarrow Q_{new}(s,a) \leftarrow \frac{Q_{old}(s,a)(k-1) + R(s,a,s_k' + \gamma \max_{a'} Q_{old}(s_k', a')}{k}$$

Robot walks around. Based on actn a in state s from many states, find its respective Q(s,a) and update.

$$= Q(s,a) + \frac{R(s,a,s_k') + \gamma \cdot \max_{a'} Q(s_k', a') - Q(s,a)}{k}$$

Steps (Stochastic Gradient Descent) — Q-Learning Algo.

① Collect a sample for (s,a,s')

② $Q(s,a) \leftarrow Q(s,a) + \alpha_k [R(s,a,s') + \gamma \cdot \max_{a'} Q(s',a') - Q(s,a)]$

where $\sum_{k=1}^{\infty} \alpha_k \rightarrow \infty$ and $\alpha_k \rightarrow \infty$

$\sum_{k=1}^{\infty} \alpha_k^2 < \infty$

How to prove

A. $1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} \quad > \infty$

$> 1 + \frac{1}{2} + (\frac{1}{4}+\frac{1}{4}) + (\frac{1}{8}+\frac{1}{8}+\frac{1}{8}+\frac{1}{8}) + \ldots$

$\searrow \infty$

B. $1 + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} + \ldots \quad < +\infty$

$< 1 + \frac{1}{1\times2} + \frac{1}{2\times3} + \frac{1}{3\times4} + \ldots$

$\searrow = 1 + (1 - \frac{1}{2}) + (\frac{1}{2} - \frac{1}{3}) + \ldots$

$= 2 < +\infty$

Robot Explore initially. After some time, find Q of env. & follow policy. Explore → Exploit

EXPLORATION - EXPLOITATN TRADE-OFF

Exam Review

FOCUS

1. Learning $P(a|y=+1) = \frac{count(a, y=+1)}{count(y=+1)}$
2. Markov Blanket (Spouse, Children, Parent)

X: a b c d Y: + −

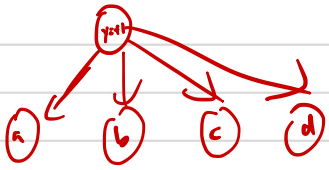
Naive Bayes — Generative Model

↓

Special Case of Bayesian Network

$P(X,y) = P(y)P(X|y) = P(y=+1) \cdot P(a|y=+1) P(b|y=+1) P(c|y=+1) P(d|y=+1)$

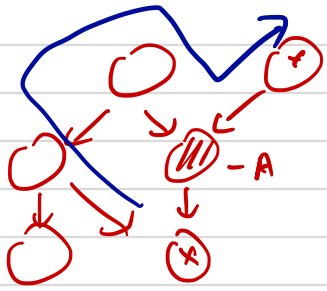
(marginal probabilities)



Bayesian Network — Generative Model

No testing on Gibbs Sampling.

Bayes Ball Network



X, Y given. If A not black, open, dependent
If A black, dependent then blue path

Hidden Markov Models — Special Case of Bayesian Network

$P(x_1 \dots x_n, y_0, y_1, \dots, y_n, y_{n+1}) = \prod_{i=0} a_{y_i, y_{i+1}} \prod_{i=1}^{n} b_{y_i}(x_i)$

transition

→ Supervised learning

Given : X, Y

Find : $\theta = \{a, b\}$

Step . $\Pi \rightarrow \log \rightarrow \Sigma \rightarrow \partial \rightarrow = 0$

$a_{uv} = \frac{\text{count}(u,v)}{\text{count}(u)}$

$b_u(o) = \frac{\text{count}(u \rightarrow o)}{\text{count}(u)}$

(Training)

* Decodg / Testg / Ev

From Traing phase : Obtain θ ie a, b.

Have Testg X

Find : Y

Forward Path Calculate Scores.

Viterbi Algo. to solve. for Y. (Back-Trace Whole Path)

→ Unsupervised Learng of HMM

- Use EM (E-Step, M-Step)

Assign membership update Model Param.

→ Hard EM (randomly assign membership / or model param)

E step : Find membership. (Decodg Procedure)

M-step : Have, X and Y → update model param.

→ Soft EM

- Diff in E-Step (Soft, Not Hard Membership)

In M-step : instead of integer count, fractional / exp. count → a, b = θ. Obtained.

Soft EM,

$$\text{count}(u,v) = \sum_{i=1}^{m} \text{count}^{(i)}(u,i)$$

i-th instance. $\text{count}(u,v) = \sum_{Y} P(Y|X)\ \text{count}(u,v \text{ in } X,Y)$

$$= \sum_{Y} P(Y|X) \cdot \sum_{j=0}^{n} \text{count}(u,v \text{ in } X,Y \text{ @ posn } j)$$

$$= \sum_{j=0}^{n} \sum_{Y} P(Y|X)\ \text{count}(u,v \text{ in } X,Y \text{ @ posn } j)$$

$$= \sum_{j=0}^{n} \sum_{Y} P(y_0, y_1, \ldots, y_i, y_{j+1}, \ldots, y_n, y_{n+1}|X) \cdot \sigma(y_j = u, y_{j+1} = v)$$

$$= \sum_{j=0}^{n} \sum_{Y} P(y_0, y_i, \ldots, y_j = u, y_{j+1} = v, \ldots, y_n, y_{n+1}|X)$$

$$= \sum_{j=0}^{n} P(y_j = u, y_{j+1} = v|X) \rightarrow \text{INFERENCE PROB}$$

$\text{count}(u) = \sum_{j=1}^{n} P(y_j = u \mid x)$

→ **FORWARD - BACKWARD ALGO**

$P(y_j = u, y_j = v \mid x_1 \dots x_n)$

$$P(y_j = u \mid x) = \frac{P(x_1 \dots x_{j-1}, y_j = u) \cdot P(x_j \dots x_n \mid y_j = u)}{\sum_{u'} P(x_1 \dots x_{j-1}, x_j = u') \cdot P(x_j \dots x_n \mid y_j = u')}$$

From Graph

$\alpha_u(j)$ $\beta_u(j)$

$\alpha_{u'}(j) \cdot \beta_{u'}(j)$