Reinforcement learning (CS6700)
Written assignment #3

Aravind S
EE14B013

13th Mar. 2017

1 Problem 1

No discounting here. Easy!

1.1 Part (a)

Monte-Carlo estimates are average of rewards observed from a particular state (from all episodes) till termination. Multiple visits to a state are ignored as we are doing first-visit Monte-Carlo.

For state B,

$$V(B) = \frac{(1) + (0+2) + (0+0+1) + (0+2) + (0+0+1) + (0+2) + (1) + (1) + (1) + (1)}{10} = 1.30$$

For state A,

$$V(A) = \frac{(0+1) + (2) + (0+0+2) + (0+0+0+1) + (2) + (0+1) + (2) + (0+1)}{8} = 1.50$$

1.2 Part (b)

Based on possible transitions we can obtain an estimate of transition probabilities. The possible transitions here include $A \to B$, $A \to T$, $B \to A$ and $B \to T$ where T is a terminal state. We can build a transition matrix from the episodes.

	A	B	T
A	0	6	4
B	5	0	7
T	0	0	0

where $a_{ij}(i^{th}row, j^{th}column)$ denotes the number of transitions from $i \to j$.

Now, $p(s_{t+1} = j \,|\, s_t = i) = \frac{n(s_{t+1}=j \,|\, s_t=i)}{n(s_t=i)}$ where $n(...)$ denotes number of occurences of $(...)$ in the episodes. Calculating this for all possible transitions we get,

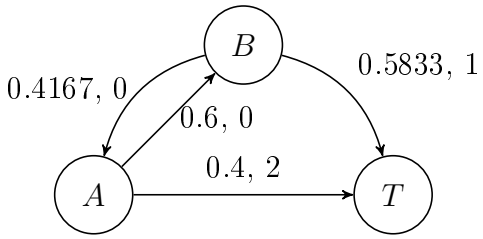
Transition	Probability	Value
$A \to B$	$p(s_{t+1} = B \mid s_t = A)$	0.6000
$A \to T$	$p(s_{t+1} = T \mid s_t = A)$	0.4000
$B \to A$	$p(s_{t+1} = A \mid s_t = B)$	0.4167
$B \to T$	$p(s_{t+1} = T \mid s_t = B)$	0.5833

We can build a transition matrix for rewards as shown below.

	A	B	T
A	0	$\frac{0+0+0+0+0+0+0}{7} = 0$	$\frac{2+2+2+2}{4} = 2$
B	$\frac{0+0+0+0+0}{5} = 0$	0	$\frac{1+1+1+1+1+1+1}{7} = 1$
T	0	0	0

where $R_{ij}(i^{th}row, j^{th}column)$ denotes the average reward obtained for transition $i \to j$.

The state-transition diagram is shown below.



1.3 Part (c)

Batch $TD(0)$ converges to the certainty-equivalence estimate for the Markov model of the system. The Markov model is constructed based on the given episodes only (shown in part (b)).

$V(T) = 0$. Based on transition probabilities,

$V(B) = 0.5833(R(B,T) + V(T)) + 0.4167(R(B,A) + V(A))$

$V(A) = 0.4(R(A,T) + V(T)) + 0.6(R(A,B) + V(B))$

Solving the above system of equations, we get,

State s	Value $V(s)$
A	1.5333
B	1.2222
T	0

2 Problem 2

2.1 Part (a)

State-space $S = \{laughter\,(L),\ silent\,(S)\}$

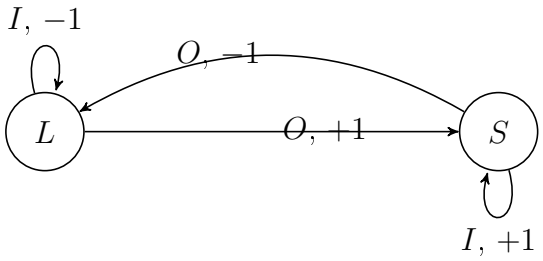
Actions $A = \{playing\,organ\,(O),\ lighting\,incense\,(I)\}$

Discount factor $\gamma = 0.9$.

Let us consider the following way of action-encodings. We will assume that if you burn incense, you won't play organ. And if you play organ, you don't burn incense. It doesn't matter as far as we get the

optimal policy in the end. Also, encoding this way results in faster convergence of iterative methods.

The state transition diagram is given below.



2.2 Part (b)

2.2.1 Policy iteration

- Let us assume an initial estimate of policy, $\pi_0 = \{L : I, S : I\}$. The symbols are defined in the previous part. Note that in vector notation, the order is: L, S (for states).
- $p_{\pi_0} = \begin{matrix} 1 & 0 \\ 0 & 1 \end{matrix}$. $r_{\pi_0} = \begin{matrix} -1 \\ +1 \end{matrix}$ $V_{\pi_0} = (I - \gamma p_{\pi_0})^{-1} r_{\pi_0} = \begin{matrix} -10 \\ 10 \end{matrix}$
- Now, $\pi_1(L) = argmax_a\{O : 1 + 0.9(10), I : -1 + 0.9(-10)\} = argmax_a\{O : 10, I : -10\} = O$
  $\pi_1(S) = argmax_a\{O : -1 + 0.9(-10), I : 1 + 0.9(10)\} = argmax_a\{O : -10, I : 10\} = I$
  Therefore, $\pi_1 = \{L : O, S : I\}$.
- $p_{\pi_1} = \begin{matrix} 0 & 1 \\ 0 & 1 \end{matrix}$. $r_{\pi_1} = \begin{matrix} +1 \\ +1 \end{matrix}$ $V_{\pi_1} = (I - \gamma p_{\pi_1})^{-1} r_{\pi_1} = \begin{matrix} 10 \\ 10 \end{matrix}$
- Now, $\pi_2(L) = argmax_a\{O : 1 + 0.9(10), I : -1 + 0.9(10)\} = argmax_a\{O : 10, I : 8\} = O$
  $\pi_2(S) = argmax_a\{O : -1 + 0.9(10), I : 1 + 0.9(10)\} = argmax_a\{O : 8, I : 10\} = I$
  Therefore, $\pi_2 = \{L : O, S : I\}$.
- $\pi_1 = \pi_2$. Thereforce, policy iteration converged to the optimal policy $\pi^* = \{L : O, S : I\}$.
- And optimal value function $V^* = V_{\pi_1} = \{L : 10, S : 10\}$.

2.2.2 Value iteration

- Let us assume an initial estimate of value function, $V_0 = \{L : 0, S : 0\}$.
- Now, $V_1(L) = max_a\{O : 1 + 0.9(0), I : -1 + 0.9(0)\} = 1$.
  $V_1(S) = max_a\{O : -1 + 0.9(0), I : 1 + 0.9(0)\} = 1$.
- $V_2(L) = max_a\{O : 1 + 0.9(1), I : -1 + 0.9(1)\} = 1.9$.
  $V_2(S) = max_a\{O : -1 + 0.9(1), I : 1 + 0.9(1)\} = 1.9$.
- $V_3(L) = max_a\{O : 1 + 0.9(1.9), I : -1 + 0.9(1.9)\} = 2.71$.
  $V_3(S) = max_a\{O : -1 + 0.9(1.9), I : 1 + 0.9(1.9)\} = 2.71$.
- $V_4(L) = max_a\{O : 1 + 0.9(2.71), I : -1 + 0.9(2.71)\} = 3.439$.
  $V_4(S) = max_a\{O : -1 + 0.9(2.71), I : 1 + 0.9(2.71)\} = 3.439$.
- Following the trend, we can observe that V^*converges to $\{L : 10, S : 10\}$ (sum of G.P with $a = 1, r = 0.9$).

2.3 Part (c)

Now we know that $V^* = \{L : 10, S : 10\}$. Knowing this, we can compute $Q^*(s, a)$ with one-step look ahead at V^*.

$Q^*(L, O) = 1 + 0.9V^*(S) = 10$
$Q^*(L, I) = -1 + 0.9V^*(S) = 8$
$Q^*(S, O) = -1 + 0.9V^*(L) = 8$
$Q^*(S, I) = 1 + 0.9V^*(L) = 10$

2.4 Part (d)

Advice to At Wits End is to follow the optimal policy as it ensures that the house remains silent. Since currently laughing sound is heard, At Wits End should play his organ and after that the house becomes silent. Now, he should switch and light an incense. He should keep doing this forever i.e he should keep burning incense and the house will be silent :). Stay happy At Wits End!

3 Problem 3

If we consider the given grid-world, there are totally 6 states out of which 2 are terminal states. From the remaining states, we can take some actions as shown below.

		S		
T2	*	A	B	T1

It is given that there is a reward of $+5$ units when $T2$ is reached, $+10$ units when $T1$ is reached and any transition to $*$ results in a reward of a units.

		$\downarrow$		
X	$\leftarrow$ $\rightarrow$	$\leftarrow$ $\rightarrow$	$\leftarrow$ $\rightarrow$	X

From state A, we won't take the "up" action because there is no reward obtained in the process and the only possible action from state S is coming down again. So effectively we wouldn't have accumulated any reward in these 2 steps but our future rewards will get discounted by γ^2 from here. As a result that action is not preferred (because we can directly turn left or right and somehow reach the terminal states - getting non-zero rewards in the process).

Therefore there are totally 8 policies possible. Since, it is a small state-space the problem can be analysed in a brute force fashion.

3.1 Policy 1

		$\downarrow$		
X	$\leftarrow$	$\leftarrow$	$\leftarrow$	X

State	Value
S	$a\gamma + 5\gamma^2$
B	$a\gamma + 5\gamma^2$
A	$a + 5\gamma$
$*$	5

3.2 Policy 2

		$\downarrow$		
X	$\leftarrow$	$\leftarrow$	$\rightarrow$	X

State	Value
S	$a\gamma + 5\gamma^2$
B	10
A	$a + 5\gamma$
$*$	5

3.3 Policy 3

		$\downarrow$		
X	$\leftarrow$	$\rightarrow$	$\leftarrow$	X

State	Value
S	0
B	0
A	0
$*$	5

3.4 Policy 4

		$\downarrow$		
X	$\leftarrow$	$\rightarrow$	$\rightarrow$	X

State	Value
S	$10\gamma^2$
B	10
A	10γ
$*$	5

3.5 Policy 5

		$\downarrow$		
X	$\rightarrow$	$\leftarrow$	$\leftarrow$	X

State	Value
S	$\frac{a\gamma^3}{1-\gamma^2}$
B	$\frac{a\gamma^3}{1-\gamma^2}$
A	$\frac{a\gamma^2}{1-\gamma^2}$
$*$	$\frac{a\gamma}{1-\gamma^2}$

3.6 Policy 6

		$\downarrow$		
X	$\rightarrow$	$\leftarrow$	$\rightarrow$	X

State	Value
S	$\frac{a\gamma^3}{1-\gamma^2}$
B	10
A	$\frac{a\gamma^2}{1-\gamma^2}$
$*$	$\frac{a\gamma}{1-\gamma^2}$

3.7 Policy 7

		$\downarrow$		
X	$\rightarrow$	$\rightarrow$	$\leftarrow$	X

State	Value
S	0
B	0
A	0
$*$	0

3.8 Policy 8

		$\downarrow$		
X	$\rightarrow$	$\rightarrow$	$\rightarrow$	X

State	Value
S	$10\gamma^2$
B	10
A	10γ
$*$	$10\gamma^2$

Conclusion

For policy π_i to be optimal, $V_{\pi_i}(s) \geq max\, V_{\pi_j}(s)$ for $i \neq j$ for all states s. Applying the above rule for each policy, we get the following results.

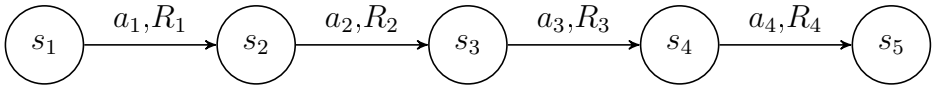
- Policy 1 and 2 can never be optimal for any γ.
- Policy 4 and 8 will be optimal for $a < 0$ and $\gamma > 0$ or $a = 0$ and $\gamma > \frac{1}{\sqrt{2}}$.
- Policy 5 and 6 will be optimal for $\gamma > \frac{\sqrt{a^2+400}-a}{20}$ and for $a > 0$.
- Policy 3 and 7 can never be optimal for any γ.

4 Problem 4

4.1 Part (a)

If we zero out the eligibility traces after 3 time steps i.e when they fall below $(\gamma\lambda)^3$, we will end up getting a variant of G_t^λ. The proof is given below.

Consider a transition as given below.



Now lets formulate eligibility for each of the 5 states for 5 time steps.

$E_i(s_j)$	s_1	s_2	s_3	s_4	s_5
e_0	0	0	0	0	0
e_1	1	0	0	0	0
e_2	$\gamma\lambda$	1	0	0	0
e_3	$(\gamma\lambda)^2$	$\gamma\lambda$	1	0	0
e_4	$(\gamma\lambda)^3$	$(\gamma\lambda)^2$	$\gamma\lambda$	1	0
e_5	0	$(\gamma\lambda)^3$	$(\gamma\lambda)^2$	$\gamma\lambda$	1

Now let V denote the value function estimate for all the states as a vector i.e $V_i = V(s_i)$. When we do online updates to V using eligibility traces, say $V(s_1)$ - will get updated till time epoch 4 as its eligibility goes to 0 after that. Also, we are assuming that s_1 is not visited till its eligibility goes to 0 (This is

not necessary, but makes the proof simple :)). So let us consider the total reward which is used to update $V(s_1)$.

$$V \leftarrow V + \alpha\delta e^{(t)}$$

where $e^{(t)}$ denotes eligibility vector (for all states) at time instant t.

Now for $V(s_1)$, the updates are as follows.

$$V(s_1) \leftarrow V(s_1) + \alpha\{R_1 + \gamma V(s_2) - V(s_1) + \gamma\lambda\{R_2 + \gamma V(s_3) - V(s_2)\} + (\gamma\lambda)^2\{R_3 + \gamma V(s_4) - V(s_3)\} + (\gamma\lambda)^3\{R_4 + \gamma V(s_5) - V(s_4)\}\}$$

Now this big summation can be reduced by simple math manipulations. Lets decompose,

$$R_1 = R_1(1-\lambda) + R_1(1-\lambda)\lambda + R_1(1-\lambda)\lambda^2 + R_1(1-\lambda)\lambda^3 + R_1\lambda^4$$

$$R_2 = R_2(1-\lambda) + R_2(1-\lambda)\lambda + R_2(1-\lambda)\lambda^2 + R_2\lambda^3$$

$$R_3 = R_3(1-\lambda) + R_3(1-\lambda)\lambda + R_3\lambda^2$$

So the above expression can be rewritten as given below.

$V(s_1) \leftarrow V(s_1) + \alpha\{(1-\lambda)\{\{\lambda^0\{R_1 + \gamma V(s_2)\} + \lambda^1\{R_1 + \gamma R_2 + \gamma^2 V(s_3)\} + \lambda^2\{R_1 + \gamma R_2 + \gamma^2 R_3 + \gamma^3 V(s_4)\} + \lambda^3\{R_1 + \gamma R_2 + \gamma^2 R_3 + \gamma^3 R_4 + \gamma^4 V(s_5)\}\} + \lambda^4\{R_1 + \gamma R_2 + \gamma^2 R_3 + \gamma^3 R_4 + \gamma^4 V(s_5)\}\} - V(s_1)\}$

i.e $G^{\lambda-eff} = \sum_{i=1}^{4} \lambda^{i-1} G^{(i)} + \lambda^4 G^{(4)}$ where $G^{(i)} = i - step TD\, return.$

4.2 Part (b)

From the above analysis, we can generalise it to $(\gamma\lambda)^n$ truncation case (after n time steps).
$G^{\lambda-eff} = \sum_{i=1}^{n+1} \lambda^{i-1} G^{(i)} + \lambda^{n+1} G^{(n+1)}$ where $G^{(i)} = i - step TD\, return.$

5 Problem 5

A Markov state provides complete information which is required for making decisions from the particular state. It can be some representation of the history of all the past states, actions and rewards. It need not contain all the details about the history but will contain information which is essential for future decisions. In other words, best policy obtained using Markov states is as good as the one which we obtain, if we use complete histories. Using this idea, lets see what happens in the Broken vision system problem.

When we see the first scene, we just see a portion of the environment - just a single snapshot. We can't see what is behind us? We can't see occluded objects? In this case, we can say that we have access to a Markov state if suppose we know the rules of the environment dynamics. This is because if the environment operates based on some set of rules and the maximum information which we can get from the environment is: the first scene, we do have access to a Markov state. It contains information based on

which all future decisions can be taken.

On the other hand, if the camera is broken we won't have images of the environment. These images might prove crucial to a future decision which we might take, and we are missing that! So, in this case we don't have access to a Markov state as we might not have all the required information needed to make a decision in the future.

6 Problem 6

Q-learning is an off-policy method and using it, we can learn optimal policies even if we follow random exploratory policies to generate episodes. Similarly, if we use on-policy methods, we need to use importance sampling if the behaviour policy and target policy are different.

Now suppose we have the optimal policy in hand. We need to learn the value function of an arbitrary policy while following the optimal policy. To learn the value function of an arbitrary policy (not optimal) we need to use some on-policy method say SARSA or Monte-Carlo. And we need to generate trajectories from that policy. Since, we are using the optimal policy to generate trajectories we can use importance sampling to weigh the updates and hence make sure that the estimated Q-values correspond to the arbitrary policy.

But this method has a higher variance and you can reduce it a bit using Weighted importance sampling at the cost of added bias. Another important thing to ensure is that the optimal policy should cover the arbitrary policy i.e it should be stochastic in places wherever the arbitrary policy is stochastic. This is important otherwise the weight blows to ∞.

7 Problem 7

The bhajji vendor's business depends on a variety of factors. All he can do is to buy raw materials or make bhajjis and sell them. His goal is to get maximum profit inspite of the stochasticity involved in this problem.

In RL terms, all these factors constitute the "state" of the vendor. It will contain all information which will be required to take future decisions. The "actions" which can be taken by the vendor are - { buy raw materials, sell bhajjis }. Now the "reward" which the vendor gets - should be such that we get an optimal policy considering the dependencies on all the factors.

The state representation contains:

$$\begin{gathered} last\ delivery\ time\ for\ all\ raw\ materials\ (for\ previous\ order)\ [tod_{last}] \\ whether\ end\ sem\ exams\ are\ going\ on\ in\ IIT\ [is\_exam\_going\_on] \\ expiry\ date\ for\ all\ raw\ materials\ [toe] \\ current\ quantities\ of\ raw\ materials\ [Q_r] \end{gathered}$$

The action-set is: $\{buy\ raw\ materials\ from\ a\ particular\ vendor, sell\ bhajjis\}$. Note that the state gets changed when an action is taken - learnt using simulation. That's where RL comes into play.

All penalties or positive feedback are provided through rewards. The reward function should consider the following factors into account: profit which he gets (depends on number of bhajjis sold, selling price, cost price and the holding price - depends on current time and last time of delivery of raw materials), penalty imposed based on current time, time of last delivery and expiry date. The profit term takes care of the under-ordering problem whereas the penalty term takes care of the over-ordering problem. So, a possible expression for reward is:

$$R = \lambda(s.p - c.p - H(t - tod_{last})) + max(t - tod_{last}, toe - tod_{last})$$

where λ is a hyperparameter, $s.p$ denotes selling price, $c.p$ denotes cost price and H denotes the holding cost per unit time.

Based on the above formulation if we sample transitions from various episodes (based on the data collected by him), we can find the optimal policy - which gives the action to take in the current state. Thus, the given problem can be solved using RL.

8 Problem 8

8.1 Part (a)

Return $G_t = R_{t+1} + R_{t+2} + R_{t+3} + ... + R_{t+k}$.

Now,

$V(s) = E[G_t|s_t = s] = E[R_{t+1} + R_{t+2} + ... + R_{t+k}|s_t = s] = E[R_{t+1}|s_t = s] + E[R_{t+2} + R_{t+3} + ... + R_{t+k+1}|s_{t+1} = s'] - E[R_{t+k+1}|s_t = s]$

Therefore,

$V(s) = E[R_{t+1}|s_t = s] + V(s') - E[R_{t+k+1}|s_t = s]$

The optimality equation is:

$V^*(s) = max_a \sum_{s',r} p(s', r|s, a)\{r + (V^*(s') - E[R_{t+k+1}|s_t = s])\}$

8.2 Part (b)

Monte-Carlo learning algorithm is preferred. This is because, the Monte-carlo estimate is unbiased. It has lower variance than normal MC method as we are using $k - step$ sum of rewards.

TD learning can't be used as the returns are just some of rewards which we get in the next k steps. So, it doesn't make sense to bootstrap from the next state. Even if we follow Bellman optimality equation and bootstrap, it will have lesser variance when compared to the previous method, but at the cost of added bias.

9 Problem 9

The proof of Bellman Optimality equation involves usage of Banach fixed point theorem and triangle inequality. To use Banach fixed point theorem, we need to prove that the transformation $L = max_a\{r(s, a) + \gamma p(j|s, a)V(j)\}$ is a contraction. This transformation is a contraction only when $\gamma < 1$. So if $\gamma = 1$, this transformation will not be a contraction and as a result Banach fixed point theorem cannot be applied. This means that, the proof of convergence doesn't hold when $\gamma = 1$.

When we go through the proof of convergence of Bellman optimality equation, we will find that the bound used for proving that L is a contraction is a very loose bound.

$$|LV(s) - LU(s)| \leq \gamma \sum_j P(j|s, a_s^*)(V(j) - U(j))$$

The above expression gets reduced to,

$$|LV(s) - LU(s)| \leq \sum_j P(j|s, a_s^*)(V(j) - U(j))$$

$$|LV(s) - LU(s)| \leq ||V - U||\{\sum_j P(j|s, a_s^*)\}$$

$$|LV(s) - LU(s)| \leq ||V - U||$$

Note that ||..|| is max-norm and the bound used in the above step is a very loose bound. As a result, we still maintain the equality sign and there is no contracting factor in the expression. If we use an alternative way to bound this and hence obtain a way to prove,

$$||LV - LU|| < \lambda||V - U||$$

where $0 \leq \lambda < 1$, then we can prove that L is a contraction which means Banach fixed point theorem will hold and hence there is a unique fixed point to Bellman optimality equation. Thus, it will guarantee the existence of a unique solution.

10 Problem 10

Many real world problems have a large number of states and hence traditional DP methods fail due to memory/time constraints. So we resort to methods like Realtime dynamic programming which obtains better estimates for the states which are more frequently visited. But the algorithm can be improved if we exploit the symmetries involved in the problem. Similar states tend to have similar optimal actions and hence number of computations can be minimised if these symmetries are used while solving the MDP.

Since the accuracy of estimates depends on the number of samples which was used to estimate them, our goal is to determine the sampling strategy based on symmetries so that it performs better than existing RTDP algorithm, when we have state-spaces with symmetries.

Algorithm

The idea is to use a grouping function G which measures how similar 2 states are, and groups similar states together. In other words, G denotes clusters of states, grouped based on symmetries.

- Generate trajectories following a policy π.
- For each step (s, a) in the episode, do the following:
  - If it is similar to some (s', a') which was seen previously, discard this current sample. Start moving from (s', a').
  - Else, simulate and find r, s'.
  - Using policy π, now choose a' - action from state s'.
  - Perform the following steps for all states t which are reachable from (s, a):

    - $P'(s,a,s'') \leftarrow P'(s,a,s'') + P(s,a,t)$ where s'' are states similar to t.
    - For states not similar to t, set $P'(s,a,t) \leftarrow P(s,a,t)$.
  - If (s,a) is unseen, $Q(s,a) \leftarrow 0$.
  - $Q(s,a) \leftarrow R(s,a) + \sum_{s''} \gamma P'(s,a,s'') max_{a''} Q(s'',a'')$

Advantages

- If we exploit symmetries in the problem, we are actually reducing the MDP to a less complex one. There are methods which reduce the MDP, store it and then we run vanilla-RTDP on that. But the above algorithm uses the notion of similar states and evaluates Q values without constructing the reduced MDP.
- If we construct the reduced MDP, it will take a very long time as we need to assign irrelevant states into some other states in the reduced MDP. But here, we can get near-optimal policies without construction of the reduced MDP.
- It is faster than the vanilla-MDP as the state space is reduced using symmetry information.

Note that the notion of similarity can be defined in many ways: we can use k-NN on feature representations obtained from deep network, we can do some sort of clustering of those representations etc. Once you have grouped similar states, the above algorithm samples transitions effectively so as to account for symmetries.

11 Problem 11

We have a bandit problem in which the parameters on which the policy depends are the preferences of the actions and the action selection probabilities are determined using a softmax relationship as:

$$\pi_t(a_j) = \frac{e^{p_t(a_j)}}{\sum_{i=1}^{n} e^{p_t(a_i)}}$$

where $p_t(a)$ is the preference of action a at time t.

The REINFORCE update equation is:

$$\theta_{t+1} \leftarrow \theta_t + \alpha_t (r_t - b_t) \nabla_{p_t} log(\pi_t(a_t)) \frac{\partial p_t}{\partial \theta_t}$$

where baseline b_t is defined as $b_{t+1} = b_t + \beta(r_t - b_t)$.

Solving we get,

$$\theta_{t+1} \leftarrow \theta_t + \alpha_t (r_t - b_t) \{1 - \pi_t(a_t)\} \frac{\partial p_t}{\partial \theta_t}$$

12 Problem 12

Let us consider a Gaussian parameterization for the same. Parameters are mean μ and variance σ^2 of the Normal distribution and baseline $b_t = 0$.

$$\pi_t(a;\mu_t,\sigma_t) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{(a-\mu_t)^2}{\sigma_t^2}}$$

The REINFORCE update equation is:

$$\theta_{t+1} \leftarrow \theta_t + \alpha_t(r_t - b_t)\nabla_{\theta_t} log(\pi_t(a_t))$$

Here $\theta_t = \begin{matrix}\mu_t \\ \sigma_t\end{matrix}$. Solving we get,

$$\mu_{t+1} \leftarrow \mu_t + \alpha_t r_t(a_t - \mu_t)$$

$$\sigma_{t+1} \leftarrow \sigma_t + \frac{\alpha_t r_t\{(a_t - \mu_t)^2 - \sigma_t^2\}}{\sigma_t}$$