HW4: Reinforcement Learning

CS6300: Artificial Intelligence **Jake Pitkin**
University of Utah

1 TD and Q in Blockworld

1. **According to direct estimation, what are the values for every state in the grid?**

State	Calculation	Value
(1,1)	(93 + 95 + 94)/3	94
(1,2)	(96 + 95)/2	95.5
(1,3)	(97 + 96)/2	96.5
(2,1)	94/1	94
(2,2)	-	-
(2,3)	(98 + 97 + 98)/3	97.666
(3,1)	-	-
(3,2)	-	-
(3,3)	(99 + 99)/2	99
(4,1)	-	-
(4,2)	-	-
(4,3)	(100 + 100)/2	100

   Assuming the value of states aren't initialized to zero. If they are, the dashes will be zeroes.

2. **According to model-based learning, what are the transition probabilities for every (state, action, state) triple. Don't bother listing all the ones that we have no information about.**

s	a	s′	T(s, a, s′)
(1,1)	up	(2,1)	1/3
(1,1)	up	(1,2)	2/3
(1,2)	up	(1,3)	1
(1,3)	up	(2,3)	1
(1,3)	right	(2,3)	1
(2,1)	left	(1,1)	1
(2,3)	right	(3,3)	2/3
(2,3)	right	(2,3)	1/3
(3,3)	right	(4,3)	1
(4,3)	exit	(done)	1

3. **Suppose that we run Q-learning. However, instead of initializing all our Q values to zero, we initialize them to some large positive number ("large" with respect to the maximum reward possible in the world: say, 10 times the max reward). I claim that this will cause a Q-learning agent to initially explore a lot and then eventually start exploiting. Why should this be true? Justify your answer in a short paragraph.**

   Consider the update for a Q-learning step:

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha * sample$$

$$sample = R(s,a,s') + \gamma \max_{a'} Q(s',a')$$

   We know initially all $Q(s,a)$ are much larger than the maximum reward possible in the world. On the first iteration we will get the first sample and we update a $Q(s,a)$. For any non-zero α, the new value of $Q(s,a)$ will be smaller than its previous value as we know $Q(s,a) > sample$.

   Now let's consider following iterations. In *sample* we take the max, with respect to a', of $Q(s',a')$. We know that if $Q(s',a')$ had been taken in previous iterations it's value decreased (if it hasn't converged yet). Therefore, we will be more likely to choose q values that haven't been explored yet.

   This will lead to more exploration initially until q values begin to converge and we start exploiting.

2 Policy Gradient

Compute a closed form solution for $g(s_0, a)$.

We are given that:

$$g(s_0, a) = \nabla_\theta \ log\left[\frac{\exp\left(\sum_{i=1}^{n} \theta_i f_i(s_0, a)\right)}{\sum_{a'} \exp\left(\sum_{i=1}^{n} \theta_i f_i(s_0, a')\right)}\right]$$

First we can apply the *log* rule $log(a/b) = log(a) - log(b)$:

$$g(s_0, a) = \nabla_\theta \ log\big(\exp\big(\sum_{i=1}^{n} \theta_i f_i(s_0, a)\big)\big) - \nabla_\theta log\big(\sum_{a'} \exp\big(\sum_{i=1}^{n} \theta_i f_i(s_0, a')\big)\big)$$

Let's consider the partial gradient. We can cancel out the *log* and *exp* terms from the left piece and take the partial gradient for term k:

$$\frac{d}{d\theta_k} \sum_{i=1}^{n} \theta_i f_i(s_0, a) - \frac{d}{d\theta_k} log\big(\sum_{a'} \exp\big(\sum_{i=1}^{n} \theta_i f_i(s_0, a')\big)\big)$$

$$f_k(s_0, a) - \frac{d}{d\theta_k} log\big(\sum_{a'} \exp\big(\sum_{i=1}^{n} \theta_i f_i(s_0, a')\big)\big)$$

For the left piece, we will use the chain rule to handle the *log* and take the partial gradient for term k:

$$f_k(s_0, a) - \frac{\frac{d}{d\theta_k} \sum_{a'} \exp\big(\sum_{i=1}^{n} \theta_i f_i(s_0, a')\big)}{\sum_{a'} \exp\big(\sum_{i=1}^{n} \theta_i f_i(s_0, a')\big)}$$

$$f_k(s_0, a) - \frac{\sum_{a'} f_k(s_0, a') \exp\big(\sum_{i=1}^{n} \theta_i f_i(s_0, a')\big)}{\sum_{a'} \exp\big(\sum_{i=1}^{n} \theta_i f_i(s_0, a')\big)}$$

Notice this contains the definition of $\pi_\theta(s, a)$ and can make a replacement to simplify our answer:

$$f_k(s_0, a) - \sum_{a'} f_k(s_0, a') \pi_\theta(s_0, a')$$

This produces a scalar value as it's just entry k of the gradient. We combine all n partials to produce the gradient:

$$g(s_0, a) = f(s_0, a) - \sum_{a'} f(s_0, a') \pi_\theta(s_0, a')$$

Explain in a few sentences *why* this leads to a sensible update for gradient ascent?

Using the closed form, this gives the following gradient estimate:

$$\nabla_\theta J(\theta) = \sum_a \pi_\theta(s_0, a) R(a)(f(s_0, a) - \sum_{a'} f(s_0, a')\pi_\theta(s_0, a'))$$

Which we will use to update θ at step $k+1$ of gradient ascent as such:

$$\theta_{k+1} = \theta_k + \alpha \nabla J(\theta_k)$$

The softmax function π_θ converts values into action probabilities (via the wikipedia for the softmax function). In our update we are considering each action (first summation) and determining it's expected value. From this, we consider all other actions (second summation) and sum their expected value. This sum is subtracted from the action a's expected value. As such, we can determine an updated θ vector estimate and compute its gradient estimate. This estimate is scaled by a learning rate and the actual θ vector is updated. The goal is after enough iterations we arrive at some local maxima.