COMP 552 HW 1

Kevin McCoy

September 27th, 2023

1. Rock Paper Scissors

1.1

$$a \in \{Rock, Paper, Scissors\}$$

$$\mathbb{P}(r|a) = \begin{cases} 0.1\mathbb{1}_{r=-1} + 0.8\mathbb{1}_{r=0} + 0.1\mathbb{1}_{r=+1} & a = Rock \\ 0.1\mathbb{1}_{r=-1} + 0.1\mathbb{1}_{r=0} + 0.8\mathbb{1}_{r=+1} & a = Paper \\ 0.8\mathbb{1}_{r=-1} + 0.1\mathbb{1}_{r=0} + 0.1\mathbb{1}_{r=+1} & a = Scissors \end{cases}$$

$$q_*(a) = \begin{cases} 0 & a = Rock \\ 0.7 & a = Paper \\ -0.7 & a = Scissors \end{cases}$$

1.2

The optimal policy is given by:

$$\pi_*(a) = \underset{a}{argmax}\ q_*(a) = Paper$$

And the expected cumulative reward is 7000.

1.3

$$q_*(a) = \begin{cases} 1 - p_{rock} - 2p_{paper} & a = Rock \\ 2p_{rock} + p_{paper} - 1 & a = Paper \\ p_{paper} - p_{rock} & a = Scissors \end{cases}$$

The optimal policy is given by:

$$\pi_*(a) = \underset{a}{argmax}\ q_*(a)$$

1.4

$$a \in \{Rock, Paper, Scissors\}$$

Alice's new policy is effectively

$$\pi_{Alice}(a) = \begin{cases} 0.45 & a = Rock \\ 0.1 & a = Paper \\ 0.45 & a = Scissors \end{cases}$$

Thus we have:

$$\mathbb{P}(r|a) = \begin{cases} 0.1\mathbb{1}_{r=-1} + 0.45\mathbb{1}_{r=0} + 0.45\mathbb{1}_{r=+1} & a = Rock \\ 0.45\mathbb{1}_{r=-1} + 0.1\mathbb{1}_{r=0} + 0.45\mathbb{1}_{r=+1} & a = Paper \\ 0.45\mathbb{1}_{r=-1} + 0.45\mathbb{1}_{r=0} + 0.1\mathbb{1}_{r=+1} & a = Scissors \end{cases}$$

$$q_*(a) = \begin{cases} 0.35 & a = Rock \\ 0 & a = Paper \\ -0.35 & a = Scissors \end{cases}$$

The optimal policy is given by:

$$\pi_*(a) = \underset{a}{argmax}\ q_*(a) = Rock$$

And the expected cumulative reward is 3500.

1.5

My strategy is to avoid playing the item that beats their last throw. For example, if my opponent plays rock, I will not play paper on the next turn. The reasoning behind this is that my opponent may be discouraged from repeating their play across turns.

$$\pi_{Robot}(a) = \begin{cases} Rock\ or\ Scissors & b_{t-1} = Rock \\ Rock\ or\ Paper & b_{t-1} = Paper \\ Paper\ or\ Scissors & b_{t-1} = Scissors \end{cases}$$

1.6

Multi-arm bandits could be okay, but it depends on what variety of MAB is used. Upper Confidence Bound, as well as a constant α value / learning rate, which prioritize new information over old would both work well. This would allow the model to update their beliefs as Alice's policy changes.

1.8

- One application of MAB could be playing a game of roulette, especially if the board is not fair. The agent can then choose between options $\{00, 0, 1, ..., 36\}$ as well as $\{red, black\}$. Choosing a number would be high risk high reward, and the color would be a safer bet but return less (the conditional reward distribution). If the roulette game is unfair / rigged, the agent could learn over time the optimal strategy.
- Another application of MAB is playing the board game 'Guess Who?'. Typically, a player can make up any unique yes or no question to ask, but to simply the situation we can limit questions to involve basic facial characteristics. The agent could choose between questions (which serve as the actions), and receive a reward based on how many cards the opposing player faces down (the conditional reward distribution).

2. Prior Knowledge in Bandits

2.1

Algorithm 1 Modified Simple Bandit Problem

$Q(a) \leftarrow 0$
$N(a) \leftarrow 0$
$q_*(a^2) \triangleq \mathbb{E}_{r \sim Pr(r|a^2)}(r)$
while TRUE **do**
$A \leftarrow \begin{cases} \arg\max_a(\{Q(a^1), q_*(a^2), q_*(a^3)\}) & w.p.\ 1-\epsilon \\ a^1 & w.p.\ \epsilon \end{cases}$
$R \leftarrow bandit(A)$
$N(A) \leftarrow N(A) + 1$
$Q(A) \leftarrow Q(A) + \frac{1}{N(A)}[R - Q(A)]$
end while

The modified version of the simple bandit algorithm should converge faster, as it can focus it's exploration on the unknown reward distribution of the first action a^1.

2.2

Algorithm 2 Modified Gradient Bandit

$\theta_1 \leftarrow 5, \theta_2 \leftarrow q_*(a^2), \theta_3 \leftarrow q_*(a^3)$
$\theta \leftarrow [\theta_1, \theta_2, \theta_3]$
while TRUE **do**
$A \sim \pi_\theta(a)$ ▷ $\pi_\theta(a) \triangleq \frac{e^{\theta_a}}{\sum_b e^{\theta_b}}$
$R \leftarrow bandit(A)$
for $i = 1, 2, ..., n$ **do**
$\theta_i \leftarrow \theta_i + \alpha R[\mathbb{1}_{(a_i = A)} - \pi(a_i)]$
end for
end while

Thus, the parameter for each action is weighted according to the expected reward. θ_1 is set to 5, which is higher than the max of 1, to encourage early exploring of the first action.

2.3

The original algorithm I wrote is not robust to this error. We can make it robust by changing it so that:

Algorithm 3 More Robust Modified Simple Bandit Problem

$q_*(a^2) \triangleq \mathbb{E}_{r \sim Pr(r|a^2)}(r)$

$Q(a) \leftarrow [0, q_*(a^2), q_*(a^3)]$
$N(a) \leftarrow [x, y, z]$
while TRUE **do**
$A \leftarrow \begin{cases} \arg\max_a Q(a) & w.p.\ 1-\epsilon \\ \text{random action} & w.p.\ \epsilon \end{cases}$
$R \leftarrow bandit(A)$
$N(A) \leftarrow N(A) + 1$
$Q(A) \leftarrow Q(A) + \frac{1}{N(A)}[R - Q(A)]$
end while

Now the known and possibly erroneous expected rewards are the initial values of our estimates. These estimates can change over time. x, y, z can act has our confidence in our prior beliefs. Higher values of x, y, z will take the algorithm longer to converge back to the correct value.

3. Gradient Bandit Algorithm: Variance

3.1

Because the softmax distribution is used, no probability in π can ever equal 0, and thus there will always be a non-zero probability of choosing a specific action. In other words, all actions will eventually be chosen and have a chance to influence the policy. The policy gradient algorithm also incorporates exploration via the initialization of the parameters $H(a)$.

3.2

Each action is equally probable with probability $1/k$. If each parameter is initialized to 10, the previous answer stays the same. This is due to the fact that the softmax function is used, thus it is not the absolute values that matter, but the differences between them.

3.3

$$\begin{aligned}
\mathbb{E}[-B(\mathbb{1}_{(a=a_t)} - \pi_t(a))] &= -B\mathbb{E}[\mathbb{1}_{(a=a_t)} - \pi_t(a)] && (1)\\
&= -B\sum_a [\mathbb{1}_{(a=a_t)} - \pi_t(a)]\pi_t(a) && (2)\\
&= 0 && (3)
\end{aligned}$$

Thus the introduction of a baseline B does not change the value of the gradient.

3.4

The term B can only be pulled out of the expectation if it does not depend on the action chosen (i.e. the random variable of interest, a). Thus B can vary with time as long as it does not depend on the action chosen.

3.5

Since this baseline is the average of *past* rewards, it does not depend on a, and thus it can be pulled out of the expectation. This will yield the same work as in 3.3.

3.6

$$\begin{aligned}
\mathbb{V}(X) &= \mathbb{E}(X^2) - (\mathbb{E}(X))^2 && (4)\\
&= \mathbb{E}(((r_t - B)g)^2) - (\mathbb{E}((r_t - B)g))^2 && (5)\\
&= \mathbb{E}(r_t^2 g^2) - 2\mathbb{E}(Bg^2 r_t) + \mathbb{E}(B^2 g^2) - (\mathbb{E}((r_t - B)g))^2 && (6)\\
&= \mathbb{E}(r_t^2 g^2) - 2B\mathbb{E}(g^2 r_t) + B^2\mathbb{E}(g^2) - (\mathbb{E}(r_t g))^2 && (7)\\
& && (8)
\end{aligned}$$

$$\mathcal{Q.E.D.}$$

3.7

We can find:

$$\frac{\partial}{\partial B}\mathbb{V}(X) \overset{set}{=} 0$$

We have:

$$\frac{\partial}{\partial B}\mathbb{V}(X) = \frac{\partial}{\partial B}\left[\mathbb{E}(r_t^2 g^2) - 2B\mathbb{E}(g^2 r_t) + B^2\mathbb{E}(g^2) - (\mathbb{E}(r_t g))^2\right] \tag{9}$$

$$= -2\mathbb{E}(g^2 r_t) + 2B\mathbb{E}(g^2) \stackrel{set}{=} 0 \tag{10}$$

$$\implies B_* = \frac{\mathbb{E}(g^2 r_t)}{\mathbb{E}(g^2)} \tag{11}$$

This is equivalent to the answer given.

$$\mathcal{Q}.\mathcal{E}.\mathcal{D}.$$

3.8

I suspect the average reward was chosen as the baseline as it is much easier to compute. Calculating the expectations in my equation (11) above is not trivial. The average reward is also much more intuitive, and shows how one can make the estimated gradient impervious to the scale of the rewards being given.

4. Markov Property

4.1

$$\mathbb{P}(X_t|X_{t-1}, X_{t-2}, ..., X_1) = \mathbb{P}(X_t|X_{t-1})$$

4.2

The sequence $(x_0, x_1, ...)$ is Markovian because the robot's next state does not depend on any state other than the one it is currently in.

4.3

My previous answer does not depend on the starting position of the robot.

4.4

$$s_t \in \{-4, -3, -2, -1, 0, 1, 2, 3, 4\}$$

where s_t is the distance from the robot to the green or blue square, whichever is the goal. This distance is defined to be negative if the goal is to the left of the robot, and positive if the goal is to the right.

4.5

One could also define the state such that

$$s_t \in \{(x_b, x_t)\}$$

which are the tuples consisting of the robots location and the location of the blue square.

5. Value Functions and Optimal Policies

5.1

We know that:

$$\max_{\pi} q_{\pi}(s,a) = \max_{\pi}[R(s,a) + \gamma \sum_{s'} T(s'|s,a)v(s')] \tag{12}$$

$$= R(s,a) + \gamma \sum_{s'} T(s'|s,a) \max_{\pi} v_{\pi}(s') \tag{13}$$

$$= R(s,a) + \gamma \sum_{s'} T(s'|s,a)v_{\alpha}(s') \tag{14}$$

$$= q_{\alpha}(s,a) \tag{15}$$

5.2

$$\max_{\pi} v_{\pi}(s) = \max_{\pi}[\sum_{a} \pi(a|s)q(s,a)] \tag{16}$$

$$= \sum_{a} \pi(a|s) \max_{\pi} q_{\pi}(s,a) \tag{17}$$

$$= \sum_{a} \pi(a|s)q_{\beta}(s,a) \tag{18}$$

$$= v_{\beta}(s) \tag{19}$$

5.3

No, the two policies are not necessarily unique. However, they are both optimal by their definition, and have equal value functions.

6. Autonomous Navigation

6.1

$$S = \{(i,j) : i,j \in 0:4\} \setminus \{(2,1),(2,2),(2,3)\}$$

$$A = \{N, S, E, W, NW, NE, SE, SW, wait\}$$

In this case, $|S| = 22$ and $|A| = 9$.

6.2

$$T(s'|s_0 = (x,y), a_0) = \begin{cases} s' = (x, y+1) & a_0 = N \text{ w.p. } 0.9 \\ s' = (x, y-1) & a_0 = S \text{ w.p. } 0.9 \\ s' = (x+1, y) & a_0 = E \text{ w.p. } 0.9 \\ s' = (x-1, y) & a_0 = W \text{ w.p. } 0.9 \\ s' = (x-1, y+1) & a_0 = NW \text{ w.p. } 0.7 \\ s' = (x+1, y+1) & a_0 = NE \text{ w.p. } 0.7 \\ s' = (x+1, y-1) & a_0 = SE \text{ w.p. } 0.7 \\ s' = (x-1, y-1) & a_0 = SW \text{ w.p. } 0.7 \\ s' = (x, y) & a_0 = wait \end{cases}$$

However, this is only the *intended* behavior, and the robot will fail occasionally. Thus, when $s' \notin S$ or when the action fails, an allowable action is chosen at random.

$$R(s,a,s') = \begin{cases} -1 & s' \text{ is white} \\ -100 & s' \text{ is red} \\ 100 & s' \text{ is green} \end{cases}$$

6.3

The choice of reward function is not unique. There are many other ways to represent our desired behavior of the agent. For example,

$$R(s,a,s') = \begin{cases} 0 & s' \text{ is white} \\ -10 & s' \text{ is red} \\ 10 & s' \text{ is green} \end{cases}$$

6.4

The state space can be modeled as $S = \mathbb{R}^2$, with the origin centered at the car and units measured in meters. Additionally, the current speed and direction of the car can be modelled using a vector. Then the action space can be modeled as $A = \mathbb{R}^2$, the set of all acceleration vectors in the Cartesian plane. The magnitude of this vector denotes the acceleration of the car, which decides how firmly to press the brake or pedal. The direction of the vector denotes the direction the car attempts to drive in, which decides how to turn the wheel. It is assumed that this car stays in Drive. The transition function could then be the vector addition $\overrightarrow{V'} = \overrightarrow{V} + \overrightarrow{a}t$, where t is the time the car will enact this action. Finally, the reward function could encode the distance to other cars and pedestrians, with higher rewards for being further away from all obstacles. The reward function should also be high for reaching its destination safely.

7. Stochastic Gradient Descent

7.1

Algorithm 4 Supervised Gradient Descent

Initialize $\theta \leftarrow \{\theta_1, ..., \theta_n\}$
Set learning rate α, learning threshold δ

while $\Delta < \delta$ **do**
    Compute gradient $\nabla_\theta J(\theta)$
    $\theta' \leftarrow \theta + \alpha \nabla_\theta J(\theta)$
    $\Delta \leftarrow \alpha \nabla_\theta J(\theta)$
end while

7.2

Algorithm 5 Supervised Stochastic Gradient Descent

Initialize $\theta \leftarrow \{\theta_1, ..., \theta_n\}$
Set learning rate α, learning threshold δ

while $\Delta < \delta$ **do**
    Compute gradient $\nabla_\theta J(\theta) = \mathbb{E}[g(\theta)]$
    $\theta' \leftarrow \theta + \alpha g(\theta)$
    $\Delta \leftarrow \alpha g(\theta)$
end while

7.3

Stochastic gradient descent is very useful when the true gradient cannot be computed, as is the case with our bandits example. SGD is also useful as it is usually more computationally efficient than vanilla GD.

7.4

Stochastic gradient descent will exhibit higher variance as it uses an estimation of the gradient, not the actual gradient itself. This higher variance is usually balanced by the computational efficiency benefits of stochastic gradient descent.