HW 4- My Computer is Now Better at Blackjack Than I Am

ENGR 3H

Due 11:59 PM, Thursday, May 30

1 Deal or No Deal? (40 Points)

A quick refresher on the rules of blackjack for those who might not be familiar: the goal is to get a total value on your cards that is as close to 21 as possible without going over 21. Cards have the same value as the number shown on the card, except for face cards (jacks, queens, and kings), which are all worth 10, and aces, which are worth either 1 or 11, depending on what's better for the player. Play starts with the player and the dealer each receiving 2 cards; the player can only see one of the dealer's cards. The player then chooses whether to receive another card or keep their current cards; they continue to do this until they decide to keep their cards or exceed 21, at which point the dealer's other card is revealed. The dealer then takes another card while their total is under 17. If the dealer's score is 17 or more, the dealer stops taking cards and compares scores with the player. If the player has a total that is higher than the dealer's total (or the dealer exceeded 21), they win! If it's less, they lose. A tie is called a push, and no one makes any money.

We're going to use Q-learning to try to teach our computers to play blackjack.

1.1 Choose Your Weapon

We can make the Q-learning quite complicated or more simple; for this assignment, you are only required to implement a simplified version of blackjack, but you are allowed and encouraged to try to get a better performance by increasing the complexity once you've got the basics working. Blackjack is a great game for simple Q-learning because its action space is only two actions: stay, meaning the player keeps their current card total, and hit, meaning the player gets an additional card. We can define the state space in a number of different ways, with increasing complexity:

1. The player's current point total;
2. The player's current point total and the number of aces the player has (since this can affect the point total);

3. The player's current point total, the number of aces, the player has, and the card that is visible from the dealer's hand;
4. The *specific cards* the player has as well as what is visible from the dealer's hand.

NOTE: Unless you're feeling really comfortable, I wouldn't try the last of these; since the number of cards the player has changes, this can get very messy to deal with. Not impossible, but not fun. Everyone is responsible for implementing Q-learning using at least the first option (the player's current point total), but you may try to implement any of the other options instead.

Now that we've defined our state-action space, let's look at how we're going to put together our code.

1.2 Initialize, Shuffle, and Deal (15 Points)

We need to start by initializing a bunch of variables and setting up our iteration loop. We want to train our Q-learning network by playing N games of blackjack. Initially, we're going to favor exploration; we'll set our initial exploration likelihood $\epsilon = 1$, but each game we're going to adjust it by a factor of β as $\epsilon_{new} = \beta\epsilon$. I recommend choosing a value of β very close to 1 ($> .99$) in the interest of making sure you fully explore the space. We also need to set our discounting value γ; this should be between 0 and .99. We also need an update rate, α, which should be between 0 and 1. We also need to initialize our Q-Table; we'll set all the initial values to 0.

For simplicity, we're going to start each game with the dealer's turn (this will allow us to immediately check the value when we stay on the player's turn). First, shuffle your deck; you will find the MATLAB function `randperm` useful here. Now deal 2 cards to the dealer, compute their total, and decide if the dealer will take the next card in the deck or stay. Continue to do this until the dealer has a total above 17, remembering they bust if the total is above 21. You may wish to set values above 21 to -1 for easier comparison later.

1.3 Time to Learn (25 Points)

Now the player needs to play the game and learn from it. The initial state the player is in is defined by the first two cards they are dealt. From there, the player selects an action either randomly (with probability ϵ) or based on the best available action in their current state. Then, we updated the reward table using the Q-learning update equation:

$$Q\left(s_{curr}, a_{curr}\right) = (1-\alpha)\, Q\left(s_{curr}, a_{curr}\right) + \alpha\left(r + \gamma \max_a Q\left(s_{new}, a\right)\right). \quad (1)$$

Here, r is the reward acquired by taking the action a_{curr} from the state s_{curr} and we consider the estimated best action from the resulting state, discounted by a factor of γ. The reward for going bust is -1, while the reward for staying and winning is +1. The reward for staying and losing is also -1, while every other action has a reward of 0. An example update proceeds as follows:

1. We begin with a 5 and an 8, so our player's score is 13.
2. We randomly select to take another card instead of staying. Our current state-action space coordinates are $(13, \text{hit})$.
3. The new card's value is 9.
4. Our new total is 22, which is greater than 21. As a result, we've lost. The reward associated with taking the action we took, therefore, was -1. Since there are no more actions to take, we don't need to worry about our new position in the state-action space; we just update with $r = -1$. So we have $Q(13, \text{hit}) = (1 - \alpha) Q(13, \text{hit}) + \alpha(-1)$.
5. Suppose instead of a 9, we had gotten an 8. Then, we'd have reached 21. At 21, staying always produces a better value than hitting (you always lose by hitting, you either win or push by staying), so the action with the best value from the new state is to stay. Then, our update would have looked like $Q(13, \text{hit}) = (1 - \alpha) Q(13, \text{hit}) + \alpha(0 + \gamma Q(21, \text{stay}))$.

2 Let's Play a Game (10 Points)

Now that you've completed teaching your computer to play blackjack, let's see how it does. Either in a separate script or after training in the same script you used for Q-learning, simulate playing 100 games of blackjack using your learned policy. Compute and display your total winnings, remembering you lose 1 if you lose and gain 1 if you win.