CSC449 - PA3

Colton Snyder

2 December 2022

1 Description of My Program

For my program I followed the True Online SARSA(λ) pseudocode on page 307 of the Sutton and Barto book. For the features vector, I used the Fourier Basis Functions with the state values, and then just appended the action value at the end. To change the order of Fourier Basis that is used, update line 29 of my code "fourierBasisOrder = 3" to use the order that is desired. You can also change the hyperparameters used for the learning algorithm by updating the values of the constants on lines 23 through 26. To run the program run "python3 pa3.py [¡episodes¿ [¡fileName¿]]" where episodes is the number of episodes that you would like to train the model for and fileName is the name of a file that you would like to write the total reward achieved in each episode to.

2 Learning Curves for Different Orders of Basis Functions

I trained my model using Fourier Basis functions of various orders. For each test, I trained for 100 episodes and recorded how long it took for each episode to reach the top of the right hill. Each time, the model ran with the hyperparameters defined as follows: $\alpha = 0.001$, $\lambda = 0.9$, $\epsilon = 0.01$, and $\gamma = 1.0$. The learning curve for each of these runs can be seen in the following figure.

As you can see, the first episode took the longest for all three runs, and for each increase in the order, from 3 to 5 and from 5 to 7, the length of that first episode about halved. Each run took only a couple of episodes before the length of the episodes converged and became pretty stable around about 150 steps. The higher order basis functions converged slightly faster; the Order 7 run became pretty stable after only about 2 episodes, but the episode lengths did then start increasing closer to 200 or 250 steps after about the 50th episode. The Order 3 run took a little longer to stabilize, going from near 6000 steps to about 1000 steps, then back up to 3000, before settling around the desired 150 steps. However, it continued to have some episodes throughout the entire run that still jumped up to over 500 steps, whereas the runs with higher order basis functions never had spikes of this magnitude after they settled around the 150 steps mark.

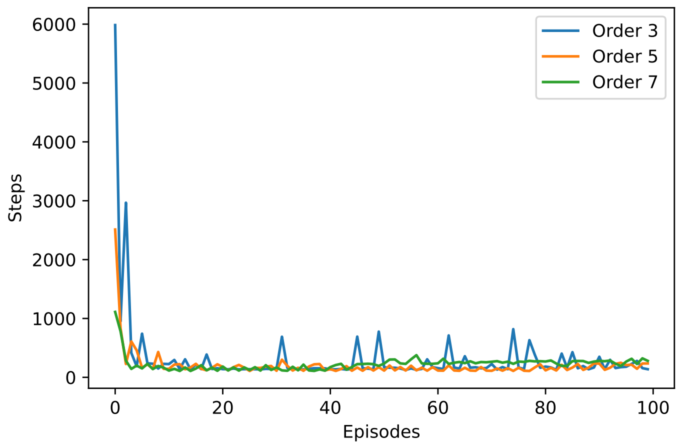

Figure 1: Learning Curves for the model under Fourier Basis functions of Orders 3, 5, and 7

3 State Value Graphs

After finding the learning curves for each of the three orders of Fourier basis functions, I then trained the model for 1,000 episodes under the three orders of Fourier basis and saved the weights that the model was using at the end of the last episode. I then used those weight vectors to determine the value of each state by walking across the entire state space, calculating my basis functions at that state under each of the three possible actions, the value of that state-action pair by taking the dot product of the basis functions and the weights vector, and then taking the greatest value of those. Since the rewards for this problem are always negative, I plotted the negative of this state value for each state.

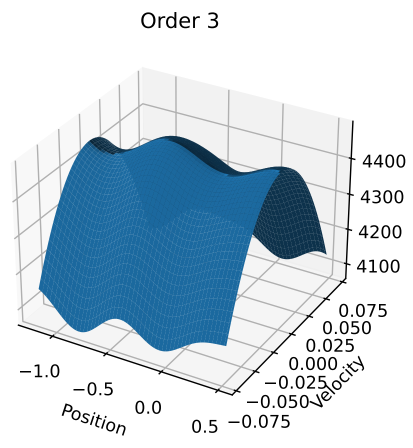

Figure 2: The state-value surface for Order 3 Fourier Basis Functions

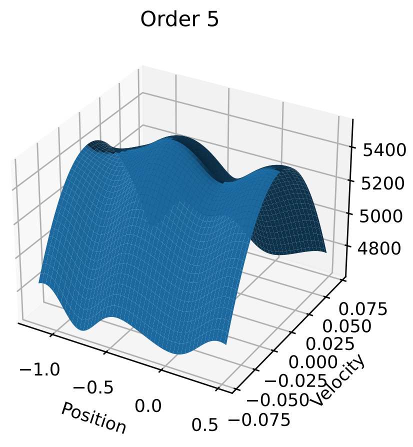

Figure 3: The state-value surface for Order 5 Fourier Basis Functions

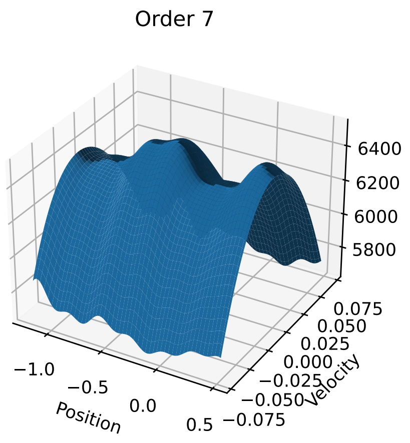

Figure 4: The state-value surface for Order 7 Fourier Basis Functions

Again, since these plots show the negative of the value for the state, the higher the point in the surface, the less the model wants to be there. Each of the three orders for basis function give the same result the worst value to about the same spot, a position of -0.5 and a velocity of 0. This would be the worst state to be in, since this is the very bottom of the valley with no momentum in any direction, so it's very hard for the car to start moving in either direction, forwards or backwards. The three different models all also show that having 0 velocity at a position of about 0.45 is worse than having 0 velocity at a position of -1, which makes sense because if you run out of speed at position 0.45, you are very close to the target, but have to reverse all the ways away from your goal to the top of the opposite hill before running back forwards to have enough speed to get past where you are currently stuck and to the goal. But if you have 0 speed at position -1.0, then if you just step on the gas and hold it, you should be able to get to the top of the mountain and reach your goal.

4 Setting γ to Less Than One

So far we have been using a $\gamma = 1.0$, this means that we don't discount our expected returns as we are learning. However, if we change that value to something less than one, say 0.5, then, since the perfect solution to this problem requires several dozen steps to reach the goal, having our discount factor, γ, less than one, we will discount the expected return values while we are training, so the Q values that we calculate get limited to values near $-1/(1-\gamma)$. This means for even smaller values of γ our state-values get even more limited and the entire state-space ends with at about the same value, so training is severely slowed down, and even about halted for any values of γ much less than 0.9.

If we change the reward function for this problem such that each step gives a reward of zero except we give a positive reward upon reaching the goal, then the model has a very tough time learning the solution. The model tries exploring, but since almost every step it takes gives a 0 reward, it calculates the return for each step to be 0, especially if we start the weights vector to be all 0s, so it can't update the weights in any direction. Only if it happens to randomly take the right steps to reach the goal does it get any feedback for how to update the weights, and will slightly adjust them. Once it does happen to reach the goal for the first time, is it able to continue the slight updates to its weights. On the future episodes, it will be able to start updating the weights a little bit more as it now won't have all 0s as the weights vector. If we keep our γ at 1.0, then we quickly hide the little bit that we learned from reaching the goal that one time because the Q estimates for every state will still be very close to zero across the board, so our delta's between the estimate for current state's Q and next state's Q will still be about zero, leaving us at still just randomly exploring all the time. If we use a γ less than one, than we still start by randomly exploring, but once we get that first reward from reaching the goal, we don't as quickly hide that little bit of information gained and can actually learn. From my couple of experiments, I was never able to complete an episode by starting the weights at zeros, but if I started the weights at random values between 0 and 1, then it was able to eventually find the goal after about a hundred thousand frames. Using the initial γ of one, the model never was able to have an episode that lasted fewer than 20,000 frames in the first 200 episodes of training, but using a γ of as small as just 0.99, it was able to quickly get to the point of every episode lasting fewer than 5,000 frames with even a couple episodes lasting fewer than 300 frames, although the average was still around about 2,000.