Alex Hanson

For Dr. Pyeatt class

CSC 449/549 - Advanced Topics in Artificial Intelligence Deep Reinforcement Learning

2 December 2022

Programming Assignment 3: Mountain Car

Implemented SARSA (λ) to solve the Mountain Car problem. Using linear function approximation with Fourier basis functions.

The problem

The Mountain car problem is a problem with the task of driving a car up a steep hill. The catch is that the car's engine is not powerful enough to drive straight up the hill meaning to solve the problem the driver must first back up away from the goal then driving forward then repeat this to build up enough momentum to get up the hill. This problem contains two continuous variables, the car's position, and velocity.

State variables
Velocity = (-0.07, 0.07)
Position = (-1.2, 0.6)
Actions
motor = (left, neutral, right)
Reward
For each time step:
reward = -1
Update function
For each time step:
Action = [-1, 0, 1]
Velocity = Velocity + (Action) * 0.001 + cos (3 * Position) * (-0.0025)
Position = Position + Velocity
Starting condition
Position = -0.5
Velocity = 0.0
Termination condition
Simulation ends when:
Position >= 0.6

SARSA (λ)

SARSA (λ) is a variant of SARSA that includes eligibility traces. The inclusion of eligibility traces gives the agent a way to remember what states it has seen leading up to its current state. This allows for previous states to be updated as well. This is a major difference compared to SARSA which will just update one state at a time taking a while to propagate back to a starting position.

Linear function approximation with Fourier basis functions

Function approximation is a technique to approximate a state's value. This is an approximation not an exact value of the state meaning it will never find the exact value of a state, but it will find an approximation of the state. The benefit of this is it can dramatically increase the speed that a solution is found.

For this problem we will be using the Fourier series as a linear function approximation. To do this we have to compute a Fourier base vector. This is a collection of repeating values. For an order 3 Fourier base vector it would contain the numbers 0,1,2, and 3 and every combination of them placed into a 2d array. The reason for doing a 2d array is because our state space contains two dimensions. We can then use this array to compute our Fourier approximation of a state.

$$\phi(x) = cos(\pi * c \cdot x)$$

This produces the basis function for a given state x. The c value is the array we calculated earlier in this section.

With scaling gradient descent lower order terms do not have to have the same learning rate α. Having lower order have a higher α results in them quickly approximating the correct over all shape of the value function. Scaling the higher order terms to have a lower α makes it so that they can have a higher precision to make smaller adjustments to the function approximation. To do this the learning rate of basis function ϕ_i is set to:

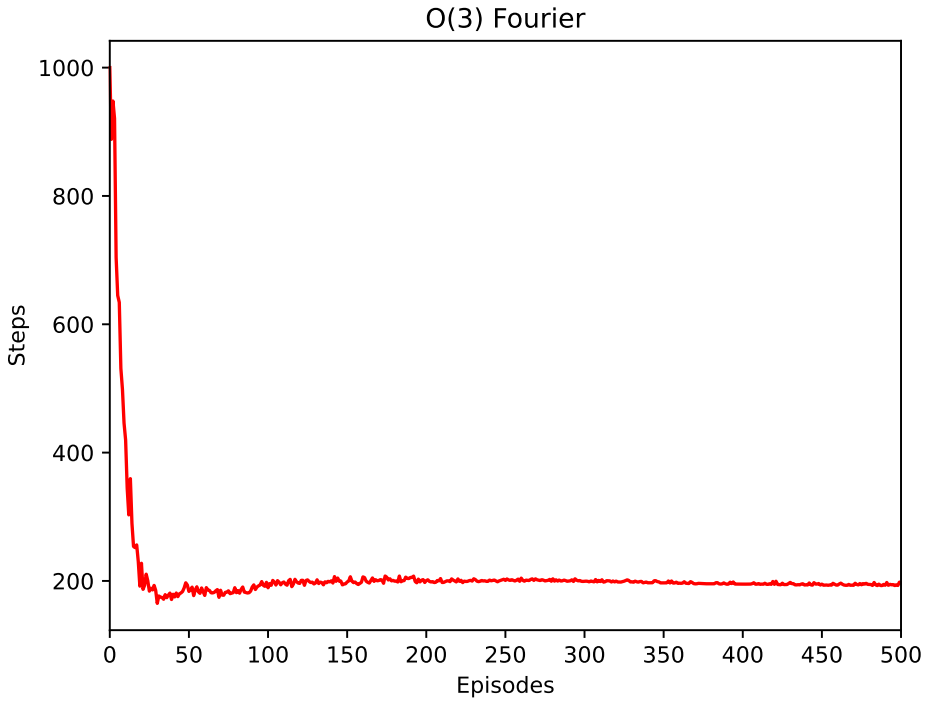
$$\alpha_i = \alpha_1 / ||c^i||_2$$

Converging

The implementation does indeed converge. Under these training rates the agent takes about 20 episodes to converge. Included with this file are multiple gifs these are trained agents ran over 1,000 episodes that are animated to show that they do indeed solve the problem.

α	0.001
ϵ	0.01
γ	1.0
λ	0.9

This graph shows 500 episodes run over 50 different samples of the training.

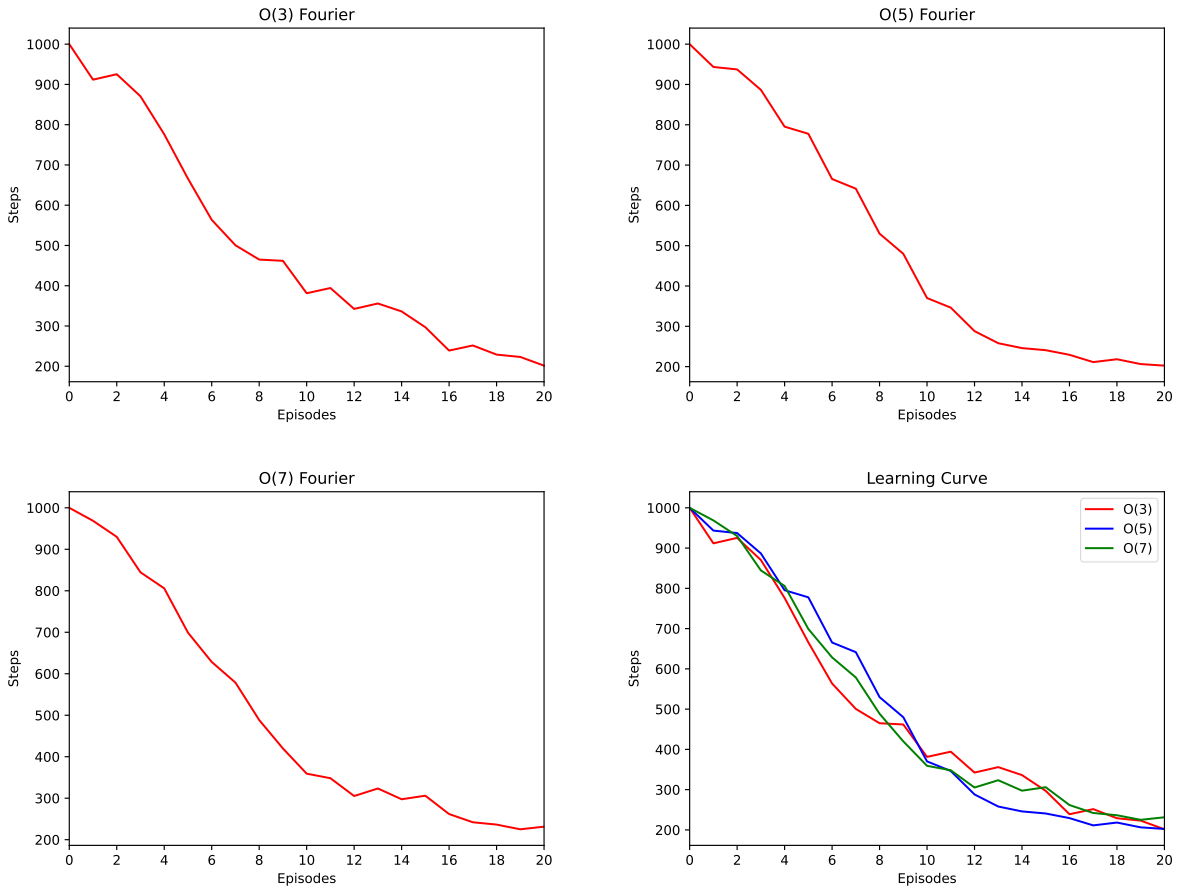


Learning curves

Depending on the order of the Fourier base function can get different results. With the higher order Fourier bases taking slightly longer to converge but still converge in under 20 episodes. The first 3 graphs show the learning curve for order 3, 5 and 7 respectively with the fourth graph showing the same graphs plotted on one for convenience of comparison.

α	0.001
ϵ	0.01
γ	1.0
λ	0.9

Graphs are 20 episodes averaged over 100 samples.



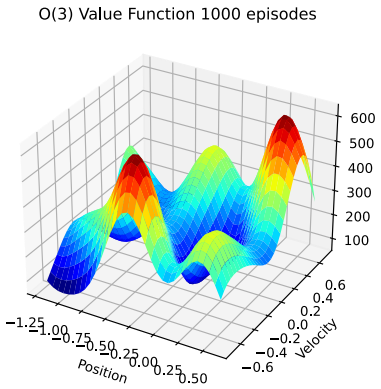
Surface plot

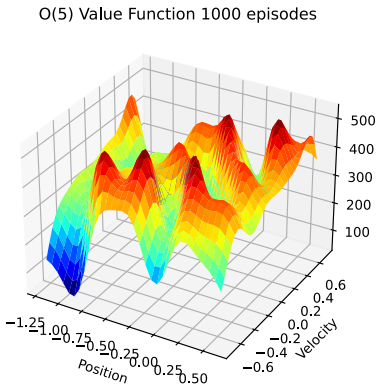
Again, depending on the order Fourier base used can produce different results. With higher order values having more precision but taking longer to converge.

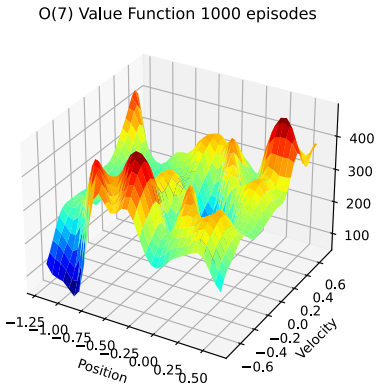
Plotting the surface plot was a big challenge for this project. With these being the initial surface plots generated for order 3, 5 and 7 respectively.

α	0.001
ϵ	0.01
γ	1.0
λ	0.9

Surface plot over 1 agent ran for 1,000 episodes.





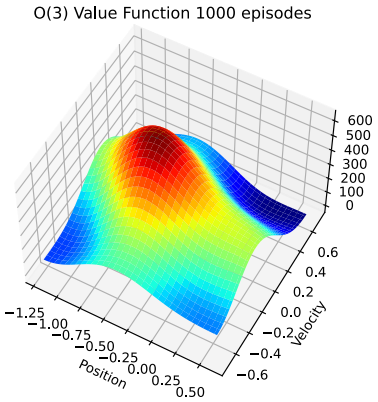


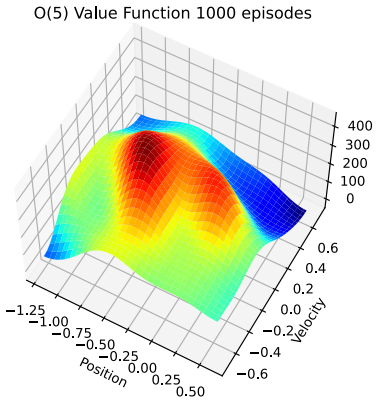
Notice how ridged and bumpy these plots are. There is little to no smoothness to their value functions. Many different vairations were attempted to fix this problem because it was not a problem in the algorithm because the agent is able to converge to a solution and solve the problem. There is an error in the generation of the graph. First potential solution was to try and max the Q value instead of sum the Q values for each state. This did not solve the problem. Second potential solution was to try and min the Q value this also did not solve the problem. After a lot of other attempts figured out the problem. The

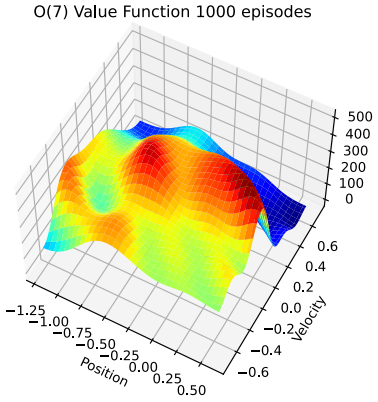
problem is that when calculating the value of a given state. The posititon and velocity of the state was being used before it was normilized to values between zero and one. This caused the graphs to be very up and down. Here are the corrected graphs with the state space normilized to between zero and one.

α	0.001
ϵ	0.01
γ	1.0
λ	0.9

Surface plot over 1 agent ran for 1,000 episodes.







What would happen if γ was less than 1 and the solution was many steps long?

The agent may never find a solution. This is caused by the fact that with a γ of less than 1. With the reduced γ when or if the agent makes it to the reward the reward is not spread out to the other states. Ran 1000 epsisodes with different values of γ to show this.

γ = 0.5 - did not converge

γ = 0.75 – did not converge, was able to solve once but with the γ value it was not able to reproduce.

γ = 0.85 - same as 0.75

γ = 0.9 - same as 0.75

γ = 0.95 - slightly better than 0.75

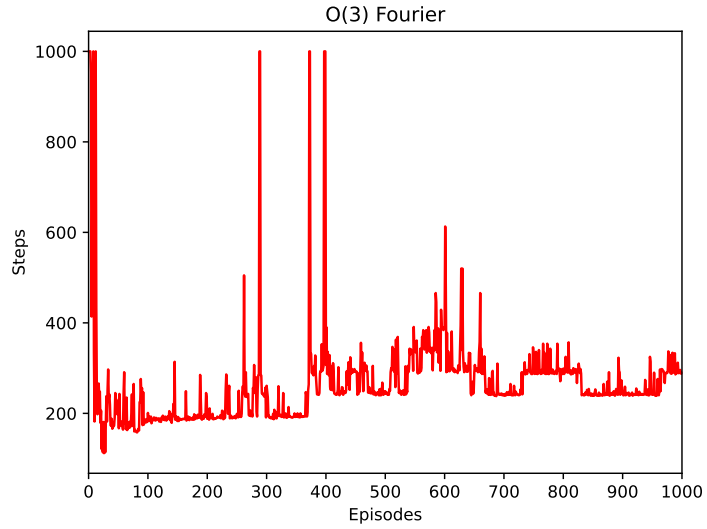
γ = 0.96 - no better than 0.95

γ = 0.97 - no better than 0.95

γ = 0.98 - better than 0.95 but still did not converge

γ = 0.99 – this was very interesting this value for γ was able to solve many times but included a lot of inconsistent episodes where it took significantly more time steps to solve.

Graph for γ = 0.99



What would happen if we had a zero step cost and a postive goal reward, for the case where $\gamma = 1$ and the case where $\gamma < 1$?

Intesresting with $\gamma = 1$ and timestep reward = 0 it was not able to converge in 1000 episodes. I initially thought that it would be able to converge. Was not able to get it to converge with any value of γ when timestep reward is zero. This makes sense because without having a negative reward for each timestep the agent has no means of telling how it is preforming.

References

Reinforcement Learning second edition

By Richard Sutton and Andrew Barto

Value Function Approximation in Reinforcement Learning using the Fourier Basis

By George Konidaris, Sarah Osentoski and Philip Thomas