Cart Pole Using Fourier SARSA(λ)

Ryan Finn | December 12th, 2022

Abstract

Fourier basis functions were implemented as a linear value function approximation scheme in the SARSA(λ) algorithm, demonstrating its effectiveness, at different parameters, for learning the inverted pendulum (or cart pole) problem described in Sutton and Barto, 1998. The experiments vary function order, discount (γ) values, and trace decay (λ) values.

Introduction

The cart pole problem introduces a goal to balance a pole pinned to a cart by pushing the cart either forward or backward. The state space is both four dimensional and continuous. Both the cart and pole have mass.

Unlike other problems, such as the mountain car problem where the goal was to minimize the amount of time spent in non-terminal states, this problem requires the time spent to be maximized. Entering the terminal state is punished with a reward of 0, while staying in non-terminal states is encouraged with a reward of 1.

Figure 1: Model animation of an inverted pendulum on a moving cart

Background

Fourier SARSA(λ) (see Appendix) was implemented with accumulating traces, $\alpha = 0.001$, $\varepsilon = 0.05$, and with a max step count per episode of 500. Three orders of FSL were trained; O(3), O(5), O(7) - each with a max episode count of 1,000.

As recommended by Konidaris and Osentoski, each basis function ϕ_i was assigned a corresponding $\alpha_i = \alpha / \| \mathbf{c}^i \|$, where $\mathbf{c}^i = \langle c_1, c_2, \ldots, c_d \rangle$, $c_j \in \{0, 1, \ldots, n\}$. In the case where $i = 0$, $\alpha_0 = \alpha$ because $\mathbf{c}^0 = \mathbf{0}$ and so $\| \mathbf{c}^0 \| = 0$. d is the number of degrees of freedom of the model which, in the case of the Cart Pole problem described in Sutton & Barto, is four (cart position, cart velocity, pole angle, pole angular velocity). n is the order for the basis functions. In the case of FSL, this results in $(n + 1)^d$ total basis functions when using the reduced $\phi_i(\mathbf{s}) = \cos(\pi \mathbf{c}^i \cdot \mathbf{s})$, where $\mathbf{s} = \langle s_1, s_2, \ldots, s_d \rangle$, $s_j \in [0, 1]$.

Related Work

FSL was also implemented in the mountain car problem (as described by Sutton and Barto), and showed great performance, especially at order three. Konidaris and Osentoski also implemented FSL on problems like the swing-up acrobat, discontinuous room, and the mountain car, comparing its performance to other basis functions (polynomial, radial, and proto-value).

Experimental Setup

Normally, the cart velocity and pole angular velocity are unbounded, allowed to reach infinite speeds. This is impractical (and inaccurate), as normalizing the state values becomes impossible. For the purposes of this experiment the two velocities were bounded between -5 and 5 units / time. This was chosen as in testing the two velocities never exceeded a magnitude of 3 or 4 units / time. Also, in a real life application both the cart and pole would have a finite maximum velocity.

The cart position is bounded between -2.4 and 2.4 units. Exceeding these bounds results in an automatic terminal state and will end the episode. It is possible to redefine this behavior to allow the cart to simply come to a sudden stop at these edges, or even bounce off of them with some defined coefficient of restitution. This could allow FSL to learn to use the edges to its advantage and correct the pole using the sudan change in momentum, which would be an interesting experiment in itself. However, in a real life application, colliding with environment bounds may not be preferred, so FSL is punished in this experiment if it does so.

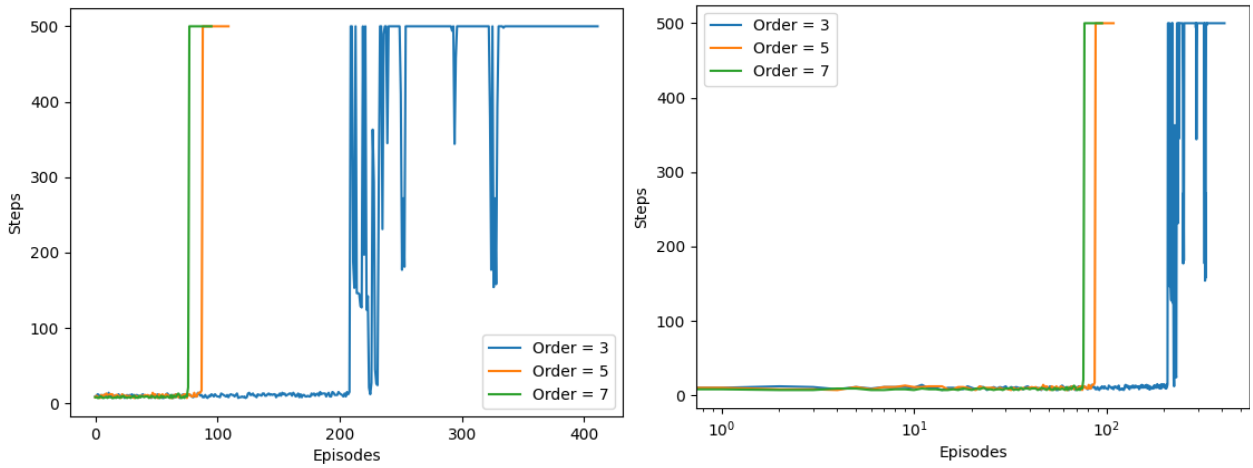
The pole is defined as non-terminal if it remains in an upright position anywhere between -12° to 12°, exceeding this range will result in a terminal state and exit the episode.

Only two actions can be taken: push the cart left (force of -10), or push the cart right (force of 10). It may be more realistic to add a third action to not push the cart at all. More actions may even be added to make the action space more granular, allowing both small pushes and large pushes.

Every episode will initialize the system in a random, non-terminal state. Transitioning to a non-terminal state is rewarded with a value of 1. Transitioning to a terminal state gives no reward (value of 0).

Results

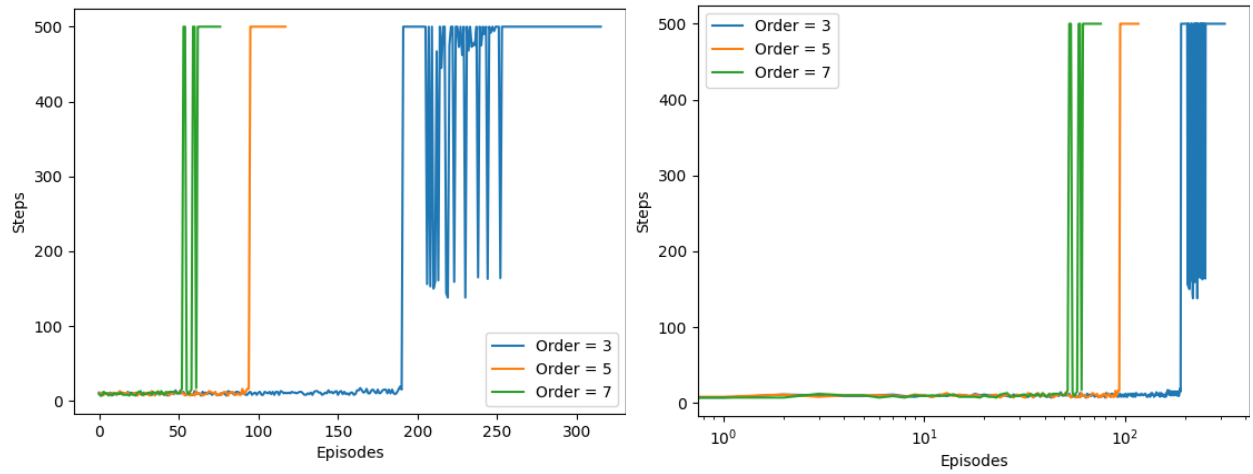
The below figures show the learning curves (steps per episode) for various combinations of γ and λ, for orders 3, 5, and 7.



(a) (b)

Figure 2: ɣ = 1, λ = 0.9

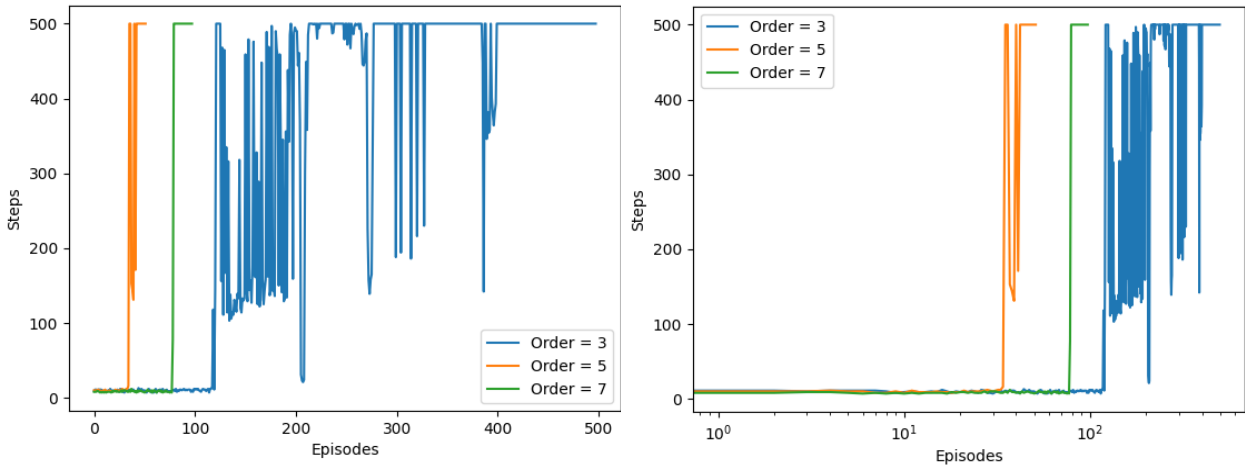
Orders 5 and 7 seem to converge very rapidly, and with very little noise. Whereas O(3) took many more episodes before converging, and was quite noisy in the process. It's also interesting to see how similar O(5) and O(7) are as compared to how similar O(3) and O(5) are.



(a) (b)

Figure 3: ɣ = 0.95, λ = 0.9

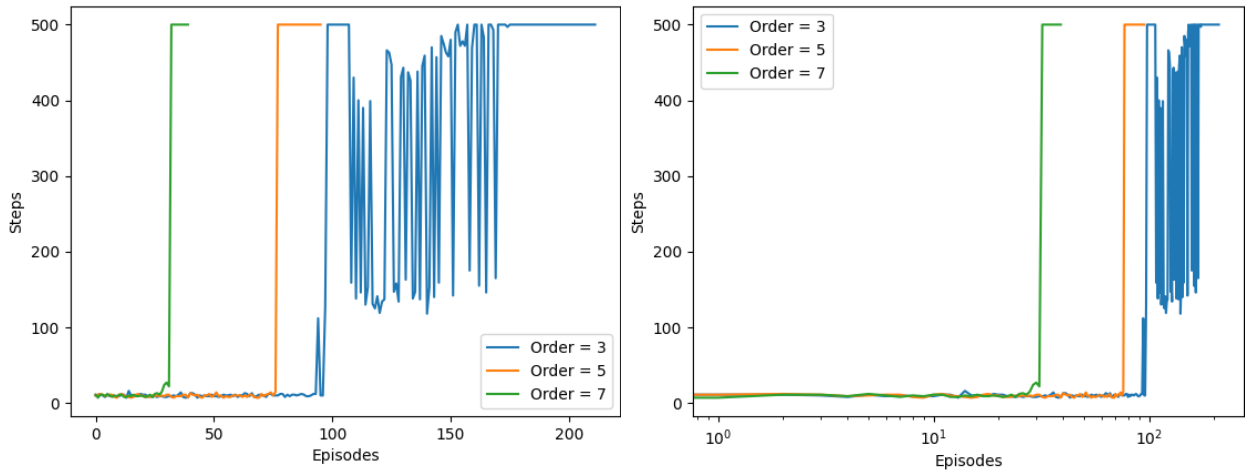
Similar to when ɣ = 1, orders 5 and 7 converge rapidly, but this time there is a much more noticeable difference between their relative performance. O(7)performs notably better than O(5) this time, and outperforms itself when ɣ is 1.



(a) (b)

Figure 4: ɣ = 1, λ = 0.5

In this case all three orders perform much closer to each other. However, notice that it's O(5) that actually learns the fastest, rather than O(7) like in the previous two cases. I'm unsure as to the cause of O(5)'s sudden performance boost when reducing λ.



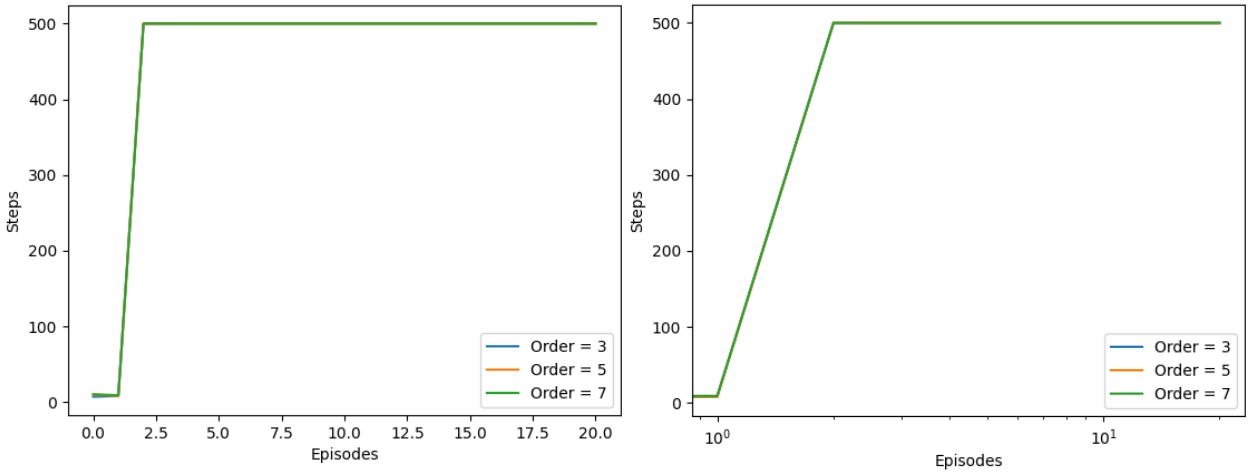
(a) (b)

Figure 5: ɣ = 0.95, λ = 0.5

Unusually, O(7) once again overtakes O(5), even with the lowered λ. In this case all three orders performed the best versus their counterparts in the other combinations of ɣ and λ, with O(7) converging in less than 50 episodes.

Comments

Normally the transition rewards would be defined differently; staying non-terminal gives no reward (0) and going terminal punishes with a negative reward (-1). In fact, this results in a significantly faster learn time for all three orders and for all combinations of ɣ and λ. Unfortunately, the learning time is so fast that it becomes hard to inspect performance differences, as every combination converges within three or less episodes.



(a) (b)

Figure 6: Corrected rewards, ɣ = 1, λ = 0.9, showing only the first 20 episodes

Defining the transition rewards as laid out in the experiment revealed the performance differences much more, and so this is why they were defined that way. In reality, they should be defined as 0 and -1 instead of 1 and 0, as it simply makes more sense, reduces the risk of overflow errors, and results in better learning performance.

Appendix

FSL Algorithm

$$\mathbf{w} = \begin{bmatrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix}, \quad \begin{matrix} \#columns = \#actions \\ \#rows = N = (n+1)^d \end{matrix}$$

$$\boldsymbol{\phi}(s) = \begin{bmatrix} \phi_1(s) \\ \phi_2(s) \\ \vdots \\ \phi_N(s) \end{bmatrix}$$

$$\boldsymbol{\alpha} = \begin{bmatrix} \alpha_1 & \alpha_1 & \cdots & \alpha_1 \\ \alpha_2 & \alpha_2 & \cdots & \alpha_2 \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_N & \alpha_N & \cdots & \alpha_N \end{bmatrix}, \quad \#columns = \#actions$$

```
For each Episode:
      Init s
      a ~ ε-greedy(s, w)
```

$$\mathbf{z} = \begin{bmatrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix}, \quad \begin{matrix} \#columns = \#actions \\ \#rows = N = (n+1)^d \end{matrix}$$

```
      For each Step in episode:
            z[:, a] ← z[:, a] + ϕ(s)
            Take action a, observe s’ and R
            δ ← R - Q(s, a, w)  # Q = w[:, a]·ϕ(s), or -1 if s is terminal

            If terminal:
                  w ← w + δz·α
                  Break

            a’ ~ ε-greedy(s’, w)
            δ ← δ + ɣQ(s’, a’, w)
            w ← w + δz·α
            z ← ɣλz
            s = s’
            a = a’
```

References

Konidaris, G., and Osentoski, S. 2011. Value function approximation in reinforcement learning using the Fourier basis. Technical Report UM-CS-2011-19, Department of Computer Science, University of Massachusetts, Amherst.

Sutton, R., and Barto, A. 1998. Reinforcement Learning: An Introduction. Cambridge, MA: MIT Press