ASSIGNMENT

Continuous-State Sarsa: Mountain Car

Rodrigo Pérez Dattari
March 28, 2019

1 INTRODUCTION

The goal of this assignment is to modify a MATLAB implementation of Sarsa that uses state discretization. This reinforcement learning (RL) algorithm must be used to solve the Mountain Car problem. The objective is to use a continuous space approximation instead of the state discretrization, specifically, a radial basis function (RBF) approximator. The properties of RBF value function approximators must be studied and compared with the discrete state case.

The specific objectives of this work are the following:

- Reproduce the results shown in the slides 22 and 23 of the presentation attached to the assignment.
- Compare the radial basis function approximator to the state discretization.
- Discuss the influence on the performance of the number of discrete states vs the number of basis functions.

2 SARSA

The main idea of the Sarsa algorithm is to learn a policy by finding an action value function $Q(s,a)$. This function is an estimate of the expected return for any given state-action pair; thus, a policy can be described by:

$$\pi(s,a) = \arg\max_a Q(s,a) \tag{2.1}$$

If $Q(s,a)$ is represented by a table, every time a state-action pair is visited, this table can be updated using the following formula:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a)] \tag{2.2}$$

Where α is the learning rate and γ the discount factor. The term $r + \gamma Q(s', a')$ corresponds to an approximation of the actual return G. This approximation is based on bootstraping the future reward with the same action value function that is being learned. This is the Sarsa version based on Temporal-Difference(0), Sarsa(0).

RL methods based on discrete action value functions are limited when the state space of the problem is big. Finding an expected return for every state-action pair of the Q table can become extremely time consuming, if not, unfeasible. For these cases, an alternative is to approximate Q with a continuous function $\hat{Q}$. In this case, the objective is to modify the parameter vector θ of the function $\hat{Q}$ that approximates Q to minimize the mean squared error between them. The parameters of $\hat{Q}$ are updated using stochastic gradient descent (SGD):

$$\theta \leftarrow \theta + \alpha[r + \gamma \hat{Q}(s', a') - \hat{Q}(s, a)]\nabla \hat{Q}(s, a) \tag{2.3}$$

3 Radial Basis Functions

A well-known case of function approximation in RL is that in which $\hat{Q}$ is a linear function of the weight vector θ. For instance, a linear combination of gaussian RBFs can be used as a linear function approximator. If $\phi(s, a)$ corresponds to a vector of basis functions which is weighted by θ, equation 2.3 can be evaluated as follows:

$$\hat{Q}(s, a) = \phi(s, a) \cdot \theta \Longrightarrow \theta \leftarrow \theta + \alpha[r + \gamma \hat{Q}(s', a') - \hat{Q}(s, a)]\phi(s, a) \tag{3.1}$$

When these basis functions are normalized gaussians:

$$\phi'_i(s, a) = \exp\left(-\frac{1}{2}[s - c_i]B_i^{-1}[s - c_i]\right) \qquad \phi_i(s, a) = \frac{\phi'_i(s, a)}{\sum_{i'=1}^{N} \phi'_{i'}(s, a)} \tag{3.2}$$

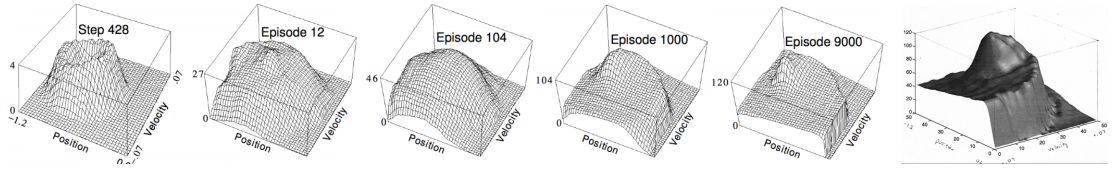
Where N is the number of basis functions and the subindex i refers to the i-th basis function.

4 Experiments, Results and Discussion

Normalized gaussian RBFs were used in this work. To make $\hat{Q}(s, a)$ depend on a , every discrete action is associated to a different set of weights, so all the weights that do not correspond to an action are taken to be zero.

4.1 Value Function Approximation

The first specific objective of this assignment was to reproduce the following results:



(a) (b)

Figure 4.1: (a) Cost-to-go function evolution (b) Cost-to-go function approximated with RBFs

The manifolds showed in Figure 4.1 are the cost-to-go function: $-\max_a \hat{Q}(s,a)$. In (a) Sarsa(λ) was used, which reduces the bias inserted by bootstrapping while gaining variance. Also, tile coding was used instead of RBFs (problem specified in [1]). With respect to (b), RBFs were used but it is not specified which type of Sarsa.

The results obtained when reproducing Figure 4.1 (a) with RBFs were the following.

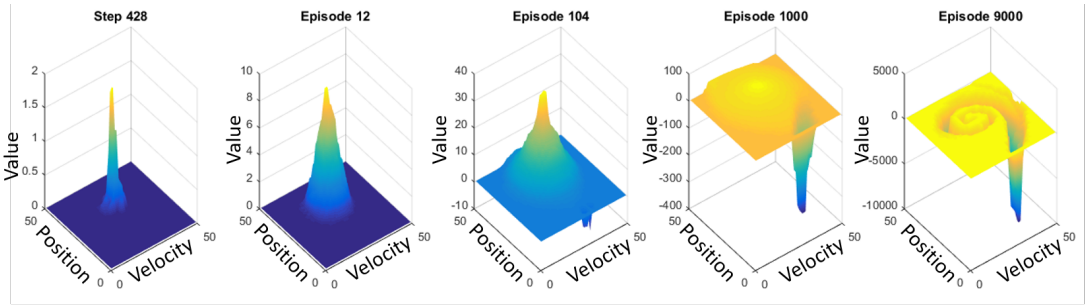


Figure 4.2: Cost-to-go evolution with RBFs. $\alpha = 0.1$, $\gamma = 0.99$, $\epsilon_{init} = 0.9$, $\epsilon_{decay} = 0.9996$

By comparing Figure 4.1 (a) with Figure 4.2 it can be observed that in the first episodes a more localized function is obtained, which becomes broader as the episodes pass. In the last episodes, a more detailed function is obtained, which takes the shape of an spiral. In the last two images of both sequences, a similar structure is observed towards the center of the function, but at the corners the shapes seem to differ.

Given that in this work Sarsa(0) is used, the values of the cost-to-go function are biased, as a consequence of bootsrapping. This makes this function to have negative values when the lower value should be zero (as it can be seen in Figure 4.1).

With respect to the corners of the shapes, they appear to be different because, as it can be seen in Figure 4.4 (b), they were never visited, thus, never updated. Figure 4.4 (b) shows the states visited by the agent of Figure 4.2. Yellow means that the state was explored, blue, that it was not. So, to make a better comparison, the functions can be compared only in the states that were explored. Having this into account, it is possible to observe that actually Figure 4.2 has a lot of common characteristics with 4.1 (a).

4.1.1 Cost-to-go function for different number of RBFs

The number of RBFs that parametrize an action value function should define its resolution. In Figures 4.3 and 4.4 this was tested using 10x10, 20x20 and 30x30 weights.

It can be seen that the manifold that is the most similar to the one in Figure 4.1 (b) is the one with 20x20 radial basis functions. The manifold with 30x30 RBFs has even more resolution than the benchmark, where is possible to observe a more detailed spiral.

4.2 Continuous/Discrete State Comparison

Discrete and Continuous state space Sarsa was trained with the same number of centers (RBF mean / discrete state). This value was modified between experiments. The idea was to observe

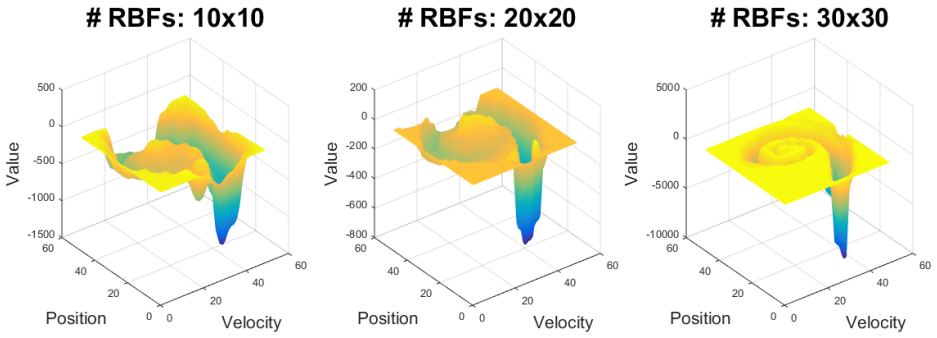
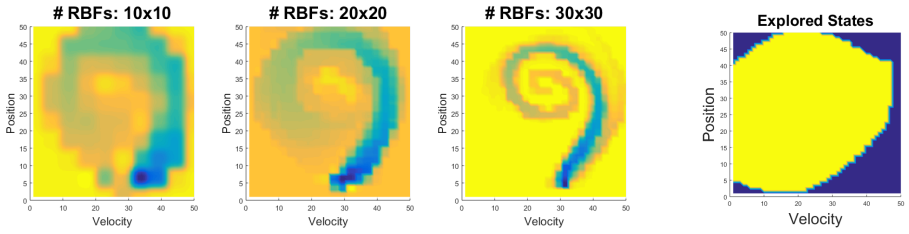


Figure 4.3: Cost-to-go function for different number of RBFs.



(a) (b)

Figure 4.4: (a) Cost-to-go function for different number of RBFs, upper view (b) States explored

that the continuous-state case is able to converge with fewer centers and that it can achieve better performance, given that it interpolates the data.

As it can be seen in Figure 4.5, the continuous-state Sarsa was able to learn a decent policy when having 3x3, 6x6, 10x10 and 20x20 centers. On the other hand, the discrete-state Sarsa was not able to learn useful policies with 3x3 and 6x6 centers.

Nevertheless, it is important to notice that not much hyperparameter tuning was made in these experiments, which can be crucial for RL algorithms to converge in some cases. The value of α was 0.1 in all of these experiments, which is not optimal, but this also indicated that using function approximation can help making the algorithm more independent from hyperparameter tuning.

With respect to the number of centers in the value functions, there is a trade-off between the function resolution and learning feasibility. For instance, approximating a value function with a lot of basis functions could be appealing given that great detail could be achieved. But, eventually the same problem of discrete states will arise; the state space will be too big for it to be approachable. In Figure 4.5, the blue solid line shows a learning curve with 20x20 RBFs, it can be seen that the other solid curves, that have less RBFs, outperform it. With that number of basis functions, it is slower for the function to converge and more training time should be considered, for instance.

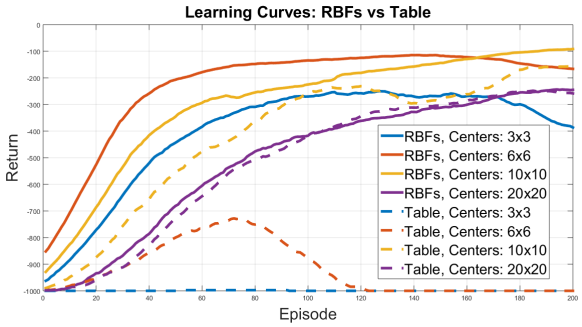


Figure 4.5: Learning curves for different number of centers (RBFs or discrete states). Five learning processes were done, averaged and smoothed for each curve. $\alpha = 0.1$, $\gamma = 0.99$, $\epsilon_{init} = 9$, $\epsilon_{decay} = 0.99$

5 Code Modifications

Two functions were deleted: `BuildQTable` and `DiscretizeState`. They were strictly necessary to the discrete case only. The following functions were modified/created:

- `BuildTheta`: creates the weight vector.
- `phi`: evaluates $\phi(s,a)$.
- `EvaluateQFunction`: Evaluates the $\hat{Q}$ function for a given action with $\hat{Q} = \phi \cdot \theta$.
- `BuildStateList`: generates the centers of the basis functions and their deviation.
- `UpdateSarsa`: updates the weights of $\hat{Q}$ with equation 3.1.
- `GetBestAction`: gets $\arg\max_a \hat{Q}$.
- `GetValueFunction`: generates a matrix with the cost-to-go function.
- `Norm`: normalizes the state between 0 and 1.

References

[1] R. Sutton and A. Barto, Reinforcement Learning: An Introduction, second edition, 2018, p.197-256.

[2] L. Busoniu, R. Babuska, B. De Schutter and D. Ernst, Reinforcement Learning and Dynamic Programming using Function Approximators, 2009, p.43-51.