CS6700 Course Project: Q Learning and Tile-coding for BSuite Environments

Kartik Bharadwaj (CS20S020), Richa Verma (CS20D020)

June 3, 2021

The problem state involves both discrete (Catch) and continuous (Cartpole and Mountaincar) environments from BSuite tool by Deepmind. The task is to design an RL agent, either tabular or using linear function approximation to facilitate learning. We primarily learn Q learning to solve the environments. Continuous state and action spaces posses a challenge for tabular RL. To this end, we use tile coding to encode features for continuous state spaces.

1 Tile Coding: An Overview

Tile coding is a technique of encoding continuous state spaces using grids. One such grid is called as one tiling. Multiple such grids are used to encode features. Sometimes, depending on the environment, it helps to use as many tilings as possible. It's also true that using more number of tilings is computationally expensive.

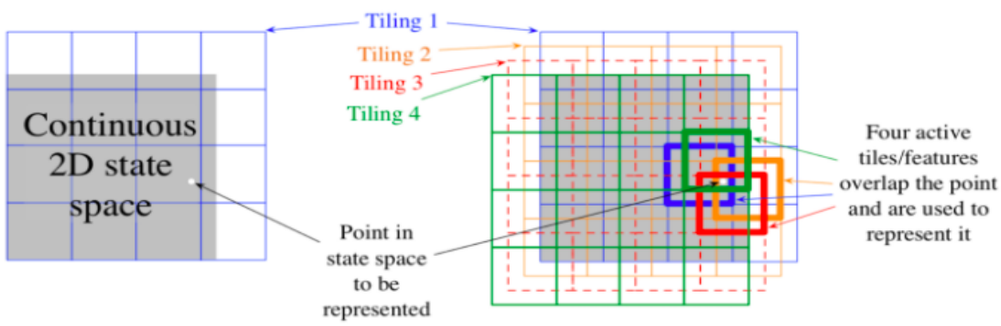

Figure 1: Tile Coding

Figure 1 [3] shows tile coding for 2D continuous state space. Left side of the figure has one tiling which represents the continuous state space in the form of a grid. On the right, the feature space is represented by the gray area. There are four tilings and the blue grid represents the first one. It encompasses the entire search space but isn't necessarily confined to it. Many other tilings are overlaid on the feature space. There's a tiny white dot which is marked by arrows. On the right, there are four active tilings that contain the white dot. All of these are offset by certain margins. Typically, the value of the grid cell is set to one in the tilings that contain the dot and it is zero for the rest of the tilings.

Tile coding is a powerful concept that has shown good results in the past for continuous control tasks [2], [1], [5]. It is essentially based on the concept of state aggregation. A state, i.e. a dot in the feature space, can belong to multiple tilings at the same time and one tiling can contain many such dots. This idea is the mainstay of the generalising capability of tile coding.

During implementation, each feature range is divided by a number of bins, which is a hyperparameter for this approach. An offset value is also defined which gives the amount by which each tiling grid is apart from the previous one. For a particular tiling, the value of the bin carrying the point of interest in the state space is set to 1 and the rest of the bins are set to a value of 0.

1. NumTilings = number of tilings
2. NumBins = number of bins
3. NumActions = number of actions
4. NumFeatures = number of features

Then, the new feature space created by tile coding can be thought of as a cube with dimensions equal to the $NumTilings \times (NumBins + 1)^{NumFeatures} \times NumActions$. For example, if there are

2 features and 3 actions, and we create 8 tilings of size 10 x 10 each, then the resulting shape of the tiling is (11, 11, 8, 3). The learning rate α is generally set to $\frac{c}{NumTilings}$.

2 Q Learning with Tile Coding

Consider an episodic Markov Decision Process (MDP) specified by the tuple $< \mathcal{S}, \mathcal{A}, \mathcal{R}, P, \gamma >$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{R}$ is the set of possible rewards, P is the transition function or dynamics and γ is the discount factor. We want to learn the optimal mapping π^* between state space $\mathcal{S}$ and action space $\mathcal{A}$, We use Q-Learning with linear function approximation to learn the value of a state-action pair. We have the following Bellman update equation for value of a state under a policy π:

$$V_\pi(s) = \sum_a \pi(a|s) \sum_{s' \in S} \sum_{r \in R} p(s', r|s, a)[r + \gamma V_\pi(s')]$$

Using Q learning, we can estimate the value of a state-action pair as follows:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha[R_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$$

For linear function approximation, we have the following:

$$V(s) = \theta^T \phi(s)$$

Let δ_t be the TD error at time-step t. Therefore, by replacing $V(s)$ in δ we have,

$$\delta_{t+1} = R_{t+1} + \theta^T \phi(s_{t+1}) - \theta^T \phi(s_t)$$

The action-value function is as follows,

$$\delta_{t+1} = R_{t+1} + \max_{a \in A} \theta^T \phi(s_{t+1}, a) - \theta^T \phi(s_t, a)$$

The aim is to compute θ to minimise the TD error δ. To achieve that, we minimise $J(\theta)$ as follows:

$$J(\theta) = \|\delta_{t+1}\| = \|R_{t+1} + \theta^T \phi(s_{t+1}, a) - \theta^T \phi(s_t, a)\|$$

The minimization of $J(\theta)$ can be done by using gradient descent method of optimization to update θ using the following equation:

$$\theta \leftarrow \theta + \alpha \delta \phi(s_t, a)$$

where, α is the learning rate and the rest of the second term is the approximate derivative of $J(\theta)$ w.r.t. θ.

3 Results

We tune the hyperparameters for all the environments using Weights and Biases tool [4]. Following are the sweep results for all the environments. Also, at the end, we show the results of the algorithm run locally.

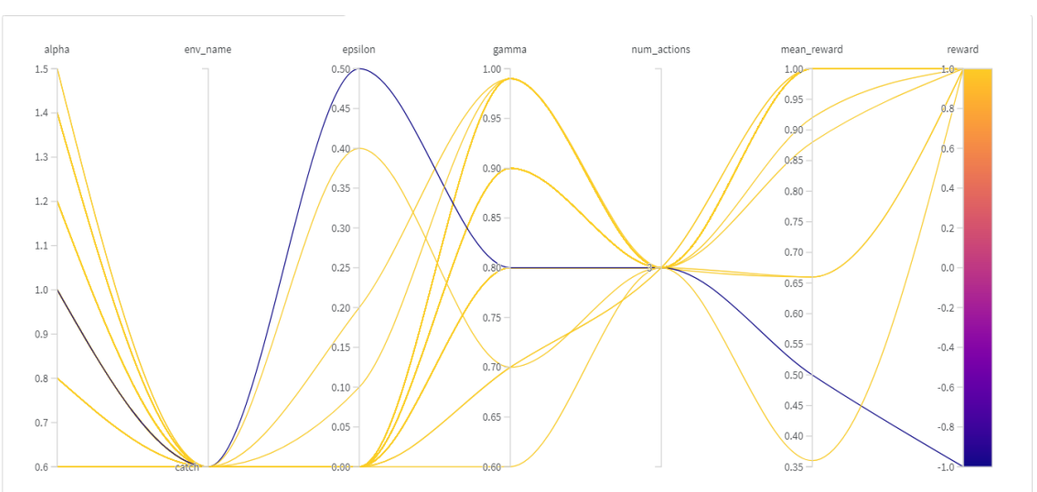

Figure 2: Catch environment

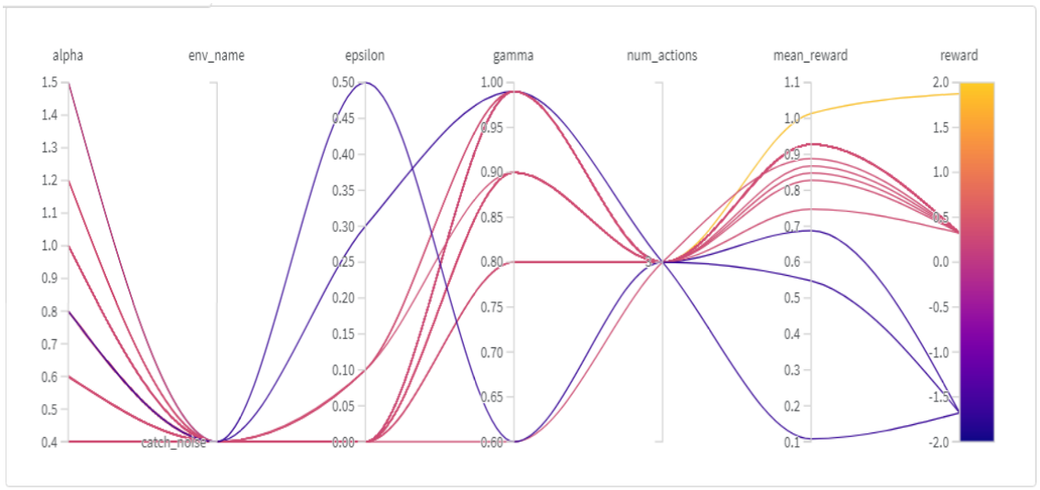

Figure 3: Catch (noise) environment

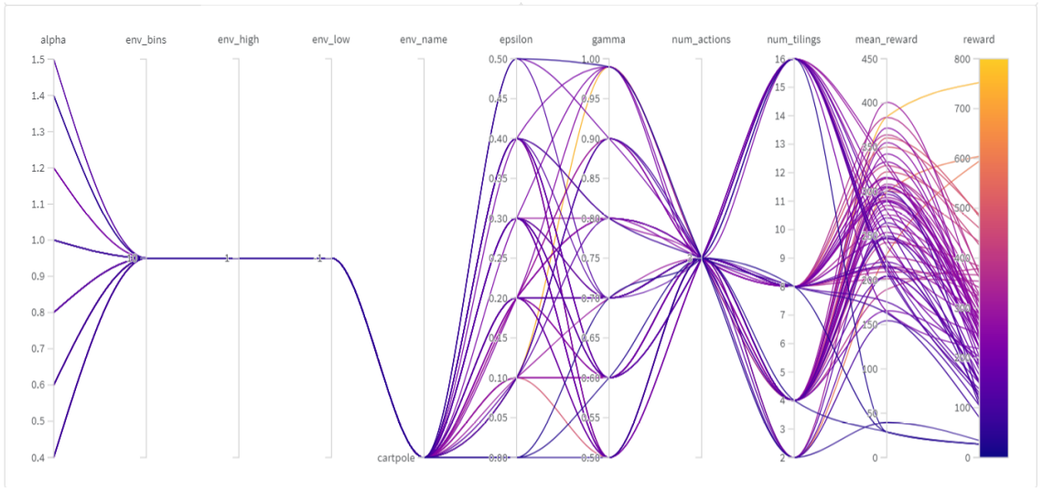

Figure 4: Cartpole environment

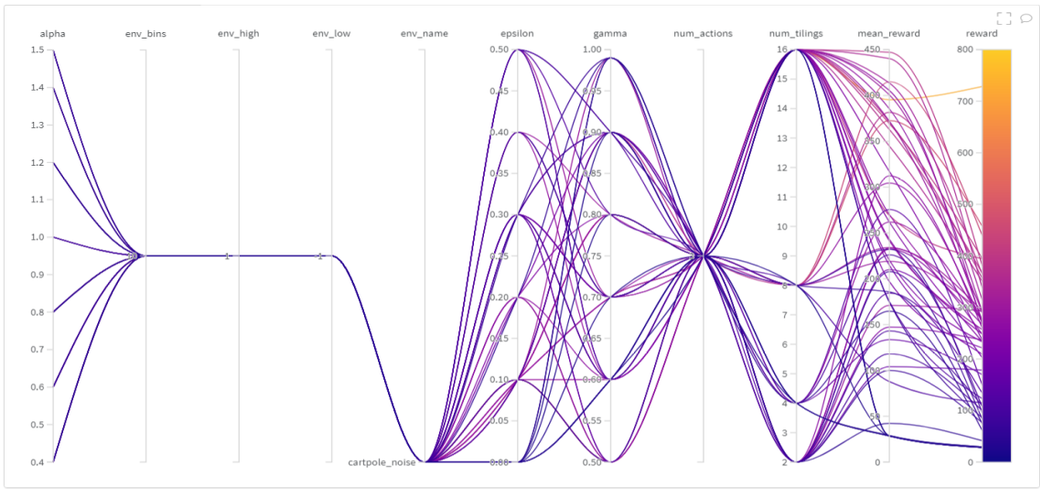

Figure 5: Cartpole (noise) environment

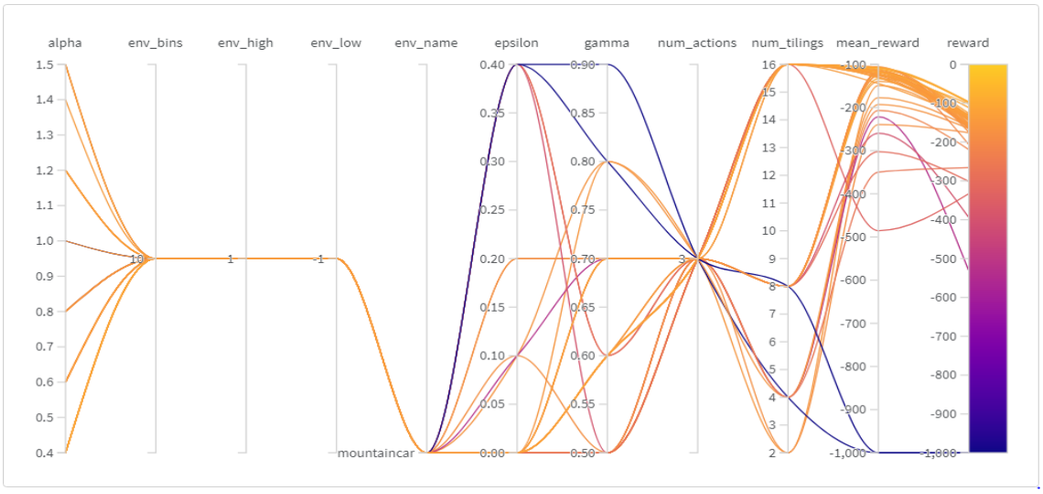

Figure 6: Mountain car environment

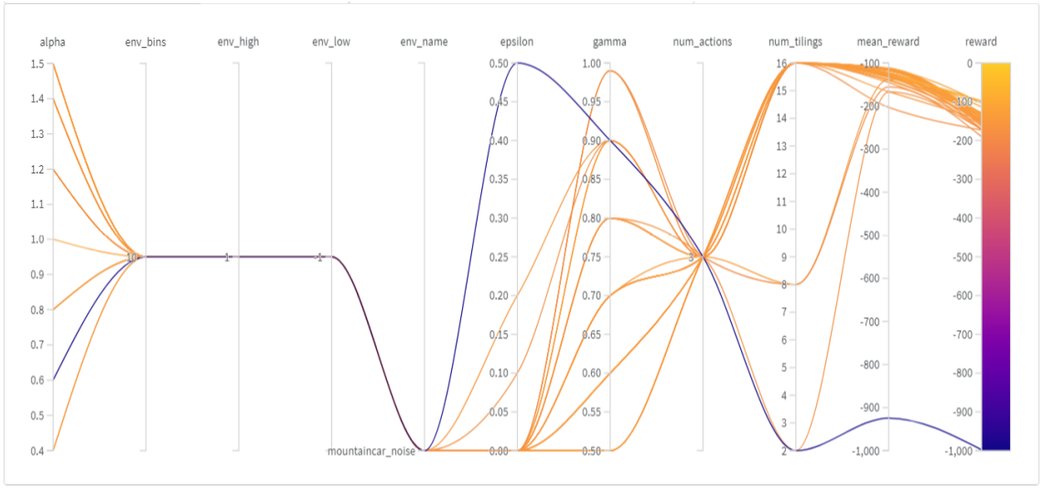

Figure 7: Mountain car (noise) environment

ENVIRONMENT	SCORE
catch	0.9485
catch_noise	0.952875
cartpole	0.64002
cartpole_noise	0.687795
mountain_car	0.918485
mountain_car_noise	0.917231

Figure 8: Local results

References

[1] Alexander A Sherstov and Peter Stone. "Function approximation via tile coding: Automating parameter choice". In: *International Symposium on Abstraction, Reformulation, and Approximation*. Springer. 2005, pp. 194–205.

[2] Alexander A Sherstov and Peter Stone. "On continuous-action Q-learning via tile coding function approximation". In: *Under Review* (2004).

[3] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: A Bradford Book, 2018. ISBN: 0262039249.

[4] *Weights and Biases – Developer tools for ML. [Online]*. URL: https://wandb.ai/site.

[5] Shimon Whiteson. *Adaptive tile coding for value function approximation*. Tech. rep. 2007.