TILE CODING IN THEANO WITH AN APPLICATION ON MOTION MODELING

Mohammad Pezeshki
Montreal Institute for Learning Algorithms
University of Montreal
mohammad.pezeshki@umontreal.ca

ABSTRACT

In this report, we have presented a framework for mapping a continuous feature space into a binary representation using Tile Coding. The presented framework for Tile Coding is created as a symbolic computation layer using the Theano library. We have provided experiments of function approximation using the presented framework. Of necessity, we only consider tilings that are uniformly distributed in a mesh shaped structure. Our main contribution is to ease the use of Tile Coding representations along with other symbolic layers such as neural networks. The code for the framework and experiments are available on authors' GitHub page[1].

1 INTRODUCTION

Reinforcement Learning (RL) has gained great deal of attention over the last decade as an area of powerful algorithms for sequential decision making. Both prediction and control algorithms in RL require evaluating or optimizing a value function. In most of real-world RL applications, the state and/or action spaces are enormous which results in in-effectiveness of conventional tabular methods (Kaelbling et al., 1996). Consequently, compelling function approximations is essential for triumph of a real-world RL agent.

Variety of different linear and non-linear function approximators have been proposed and widely used such as radial basis functions, Coarse Coding, or neural networks (Santamaría et al., 1997). Among them, Tile Coding (Sutton & Barto, 1998) forms a decent trade-off between the three factors of representational power, computational cost, and ease of use (Sherstov & Stone, 2005). Tile Coding is an extension of Coarse Coding generalization method which partitions the space into multiple overlapping partitioning as described in Section 2.1.

This paper aims to enable effective use of Tile Coding along with other function approximators such as neural networks which has been notably successful in the past decade. We present a framework on top of Theano that enables us to easily construct the computational graph with mixed blocks of Tile Coding and neural networks. In the Experiments Section, firstly, we provide a simple example of Sinusoidal function approximation. Moreover, to further illustrate the functionality of the proposed framework, we evaluate the model for the task of "*Nexting*" as explained later. We adopted the idea of Horde model (Sutton et al., 2011) in order to predict the future positions of all human body joints given current positions of all other joints inspired by Taylor et al. (2007).

2 FUNCTION APPROXIMATION

In Reinforcement Learning applications, an agent interacts with an environment following a behavior (i.e. policy) in order to do both or one of the following tasks: Prediction or Control. Prediction amounts to evaluation of a given policy while Control is to optimize a policy to gain the maximum amount of future rewards. A function from States and Action (or States alone) to the expected future reward is called the Value Function. Having the concept of value function is essential as optimizing it directly leads to the optimal policy which is desirable.

[1] https://github.com/mohammadpz/Theano_Tile_Coding

Formally, an agent given its policy can take an action from a set of actions A. Taking an action results in changing the state between a set of states S, and achieving an immediate reward R. As mentioned, value function is defined as $V : S \rightarrow R$. In many real-world cases, any of states or actions could be continuous variables leading to a huge multi-dimensional state and action spaces. Evaluating the value function over a large continuous space is not computationally feasible unless using function approximation.

Value function approximation tries to find a representation of states or actions that approximate the real unknown value function. In conventional tabular methods, experiences of the agent is stored in a sparse representation that could not easily be generalized (Kaelbling et al., 1996). Tile Coding is a function approximation method which markedly increases the generalization ability by setting multiple layers of overlapping partitioning.

2.1 Tile Coding

Tile Coding quantizes a continuous feature space into tiles. A quantization is called a tiling and each tile in a tiling is associate with a weight. By having multiple overlapping tilings, each point could have multiple weights. The approximate value for a given point is achieved by summing the weights of tiles in all tilings which contain the point. Figure 2.1 depicts the concept of tilings and tiles.

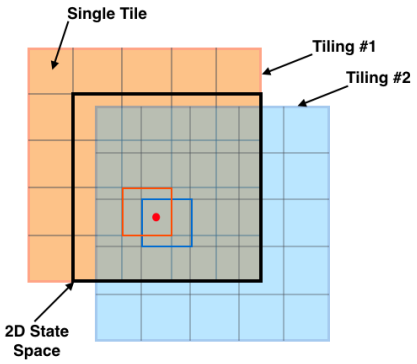

Figure 1: An example of two tilings on a 2D state space. The approximate value for the shown red point is the sum of two tiles containing the point.

During the coarse of training, the model updates the weights of involved tiles in order to decrease the total error. For a given training example (x, y) and the weight tensor w, the update rule is as following,

$$Q = \text{quantize}(x), \tag{1}$$

$$\hat{y} = \sum_{i=0}^{|Q|} w[Q_i], \tag{2}$$

$$\forall q \in Q \ : \ w[q] = w[q] + \alpha(y - \hat{y}), \tag{3}$$

in which α is the learning rate and quantize$(.)$ is function that partitions the continuous space into discrete tile indices (different tilings have different offsets).

3 The Framework

We build the Tile Coding layer using Theano library (Bergstra et al., 2010) (Al-Rfou et al., 2016). Theano is a Python library that facilitates defining, optimization, and evaluation of symbolic mathematical expressions efficiently. For a given input, we first apply a quantization layer in order to locate the given point into a tile in each tiling. Tilings are different from each other in a uniform offset resulting in Tilings larger that the feature space by one tile for each dimension (See Figure 3).

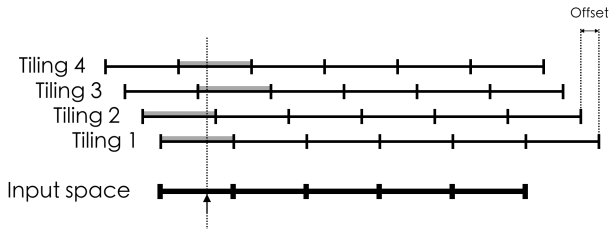

Figure 2: Offseting of tiling: in this example, since there are total of four tilings, each tiling is off by a quarter of a single tile.

In the case of T tilings and N features which any of them is quantized into t tiles, the weight tensor turns out to be a sparse 3D tensor with a shape of $T \times N \times t$. After a symbolic quantization, the involved weights are access by indexing and then sumed. Later in updating time, in order to located the involved wights, we take advantage of Theano auto differentiation. In fact, the derivative of the output with respect to the weight tensor results in a tensor of zeros and ones where the entry in the index corresponding to any of the involved tiles is one. Figure 3 illustrates this process.

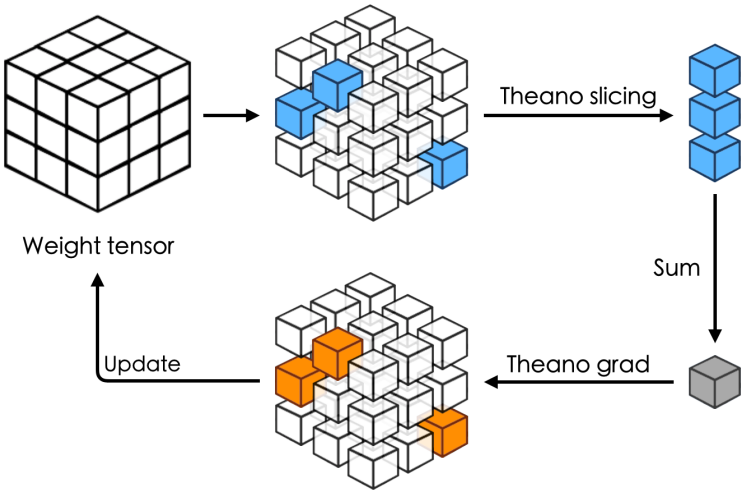

Figure 3: Positions of the involved weights are extracted from the computational graph by applying a gradient backward path from the output of the network to the weights.

4 Experiments

4.1 2D Surface Modeling

As a simple toy task, we try to approximate the following function over the range of $[0, 2\pi]$ for both dimensions,

$$z = \sin(x) + \cos(y). \tag{4}$$

As shown in Figure 4.1, the trained model is able to approximate the target function with a decent accuracy. In this example, we use ten tilings and each dimension is portioned into ten tiles. It is worth mentioning that as we increase the number of tilings, the model is more generalizable to new unseen examples and is also more robust to noisy input.

4.2 Motion Modeling using Horde

We test our framework on an adopted version of the Horde model. Horde is a multi-timescale future prediction model on an RL robot that uses thousands of parallel learners in order to approximate

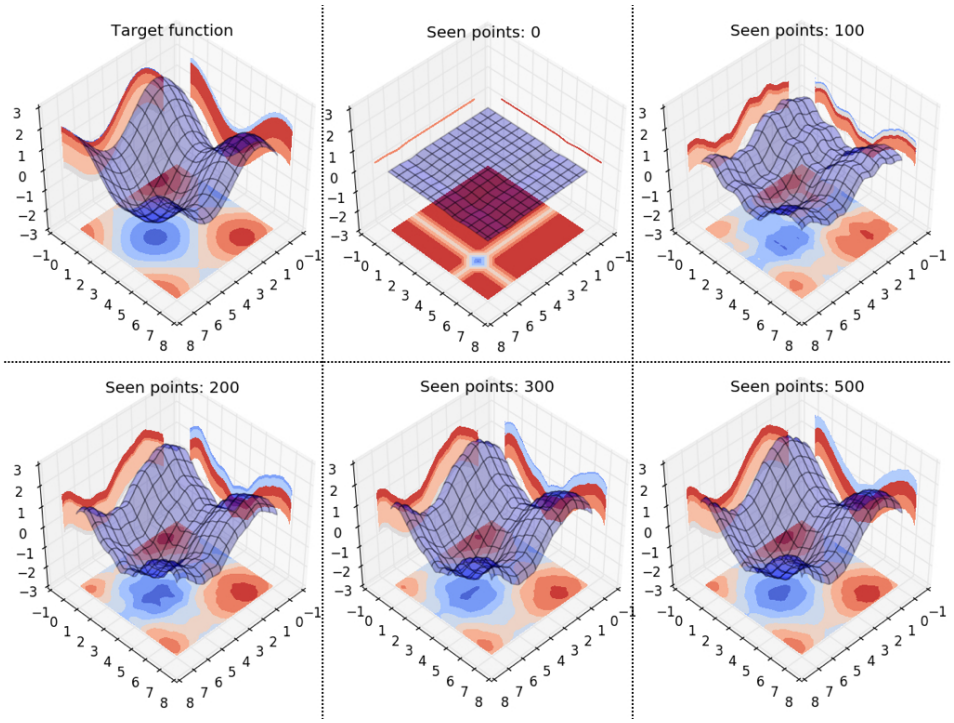

Figure 4: The Figure shows an example of approximation of the function $z = \sin(x) + \cos(y)$. Snapshots of the training procedure is shown as the number of seen data points grows.

many different outputs at the same time. The task of continually prediction future in different time-scales is called "Nexting". We adopted this model for modeling human motion. We used a single walking sequence from MIT MoCap dataset (Hsu et al., 2005) and trained our adopted version of Horde on it, armed with our Theano-based implementation of Tile Coding. The data consists of 3000 time-steps and in each time-step position of 17 body joint is presented. All body joints are represented in 49-dimensional space where each dimension is normalized to be in $[0, 1]$. The task is to predict future of each joint in the next 10 time-steps given the current position of all joints. The feature space is tile-coded into 30 tilings with 50 tiles each. An example of a trained model predictions for one of the joints is shown in Figure 4.2.

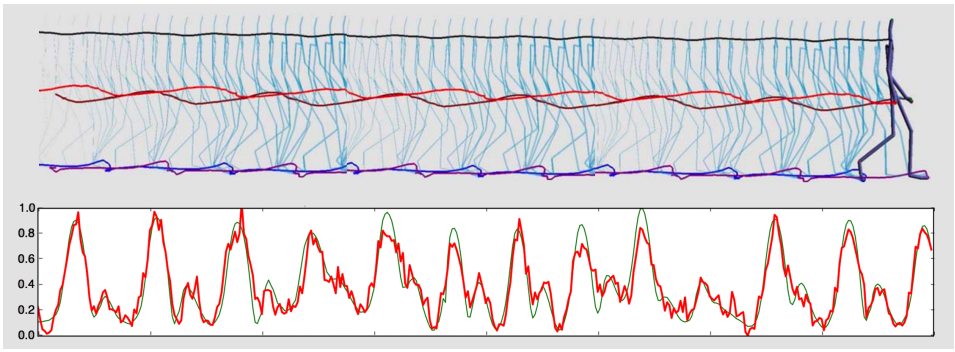

Figure 5: **(a)** The trace of each joint in our dataset. (The Figure is adopted from Taylor et al. (2007)). **(b)** The prediction the trained model (red) to be compared with the real future (green).

5 CONCLUSION

In this work, we provided a framework for function approximation using the well-known method of Tile Coding in Reinforcement Learning literature. This framework is coded on top of Theano library that provides a high-level programming environment for construction of computational graphs. We tested our implementation on a real-world task based on the Horde Model. We hope the provided framework and its ability to be used along with neural network will help researchers to build and improve function approximators.

REFERENCES

Al-Rfou, Rami, Alain, Guillaume, Almahairi, Amjad, Angermueller, Christof, Bahdanau, Dzmitry, Ballas, Nicolas, Bastien, Frédéric, Bayer, Justin, Belikov, Anatoly, Belopolsky, Alexander, et al. Theano: A python framework for fast computation of mathematical expressions. *arXiv preprint arXiv:1605.02688*, 2016.

Bergstra, James, Breuleux, Olivier, Bastien, Frédéric, Lamblin, Pascal, Pascanu, Razvan, Desjardins, Guillaume, Turian, Joseph, Warde-Farley, David, and Bengio, Yoshua. Theano: A cpu and gpu math compiler in python. In *Proc. 9th Python in Science Conf*, pp. 1–7, 2010.

Hsu, Eugene, Pulli, Kari, and Popović, Jovan. Style translation for human motion. In *ACM Transactions on Graphics (TOG)*, volume 24, pp. 1082–1089. ACM, 2005.

Kaelbling, Leslie Pack, Littman, Michael L, and Moore, Andrew W. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

Santamaría, Juan C, Sutton, Richard S, and Ram, Ashwin. Experiments with reinforcement learning in problems with continuous state and action spaces. *Adaptive behavior*, 6(2):163–217, 1997.

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

Sutton, Richard S and Barto, Andrew G. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

Sutton, Richard S, Modayil, Joseph, Delp, Michael, Degris, Thomas, Pilarski, Patrick M, White, Adam, and Precup, Doina. Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pp. 761–768. International Foundation for Autonomous Agents and Multiagent Systems, 2011.

Taylor, Graham W, Hinton, Geoffrey E, and Roweis, Sam T. Modeling human motion using binary latent variables. *Advances in neural information processing systems*, 19:1345, 2007.