An Online Kernel-Based Clustering Approach for Value Function Approximation

Nikolaos Tziortziotis and Konstantinos Blekas

Department of Computer Science, University of Ioannina
P.O. Box 1186, Ioannina 45110 - Greece
{ntziorzi,kblekas}@cs.uoi.gr

Abstract. Value function approximation is a critical task in solving Markov decision processes and accurately modeling reinforcement learning agents. A significant issue is how to construct efficient feature spaces from samples collected by the environment in order to obtain an optimal policy. The particular study addresses this challenge by proposing an on-line kernel-based clustering approach for building appropriate basis functions during the learning process. The method uses a kernel function capable of handling pairs of state-action as sequentially generated by the agent. At each time step, the procedure either adds a new cluster, or adjusts the winning cluster's parameters. By considering the value function as a linear combination of the constructed basis functions, the weights are optimized in a temporal-difference framework in order to minimize the Bellman approximation error. The proposed method is evaluated in numerous known simulated environments.

1 Introduction

The objective of Reinforcement Learning (RL) [1,2] is to control an autonomous agent in usually unknown environments. The agent interacts with the environment which is typically modelled as a Markov Decision Process (MDP), and at each time step receives a scalar reward signal that evaluates the quality of the selected transitions. The decision making procedure is designed so as to choose actions with the optimum expected returns. The quality of a policy is quantified by the so-called value function which associates to every state the expected discounted return which is received starting from the particular state and all decisions are made following this policy. However, in cases with large or continuous state spaces the value function cannot be calculated explicitly. In such domains a common strategy is to employ function approximation, by representing the value function as a linear combination of some predefined set of basis functions.

The Temporal Difference (TD) family of algorithms [3] provide a nice framework for policy evaluation. The parameters of the value function are usually learned from data, as in the case of typical TD and the Least-Squares TD (LSTD) methods [4,5]. Also, kernelized reinforcement learning methods have been paid a lot of attention by employing kernel techniques to standard RL methods [6] and Gaussian Processes for approximating the value function [7,8,9].

I. Maglogiannis, V. Plagianakos, and I. Vlahavas (Eds.): SETN 2012, LNAI 7297, pp. 182–189, 2012.

However, in most cases the basis functions used for estimating the value function remain fixed during the learning process, as for example in [10] where a predefined number of fixed Fourier basis functions are used for value approximation. Alternatively, a steady number of basis functions are tuned in a batch manner, as in the cases presented in [11, 12] that build a graph over the state space after selecting a large number of input data and then generates the k eigenvectors of the graph Laplacian matrix. In another work [13], a set of k RBF basis function are adjusted directly over the Bellman's equation of the value function. Finally, in [14] the probability density function and the reward model, which are assumed to be known, are used for creating basis function from Krylov space vectors (powers of the transition matrix used to systems of linear equations).

In this paper, a novel framework for value function approximation is proposed which addresses the issue of the on-line construction of basis functions. An online kernel-based clustering approach is used for separating the input space that contains pairs of state-action by appropriate considered a kernel function that encapsulates both kind of information. The clustering procedure selects iteratively a winning prototype and applies a learning procedure for the adaptation of its parameters based on the stochastic gradient descent. Additionally, it provides a mechanism for automatically adding clusters. The parameters of the clusters can be further used for building a dictionary of basis function which can be employed on the policy evaluation procedure for the adaptation of weights of the value function linear model. These two stages act simultaneously during the learning process aiming at the estimation of an optimal policy. The proposed method has been tested to several known simulated environments where we have made comparisons with a recent value function approximation approach that uses a fixed number of predefined Fourier basis functions.

The remaining of this paper is organized as follows. In Section 2, we briefly present some preliminaries and review the basic TD scheme for value function approximation. Section 3 contains the main contribution of this paper where we describe an efficient on-line kernel-based clustering algorithm for constructing basis functions and how it can be embedded to the basic TD learning scheme. In Section 4, experimental results are provided to illustrate the effectiveness of the proposed method. Finally, in Section 5 we give conclusions and suggestions for future research.

2 Background and Preliminaries

A *Markov Decision Process* (MDP) is a tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$, where $\mathcal{S}$ is the state space; $\mathcal{A}$ is the action space; $P : \mathcal{S} \times \mathcal{A} \times S \to [0, 1]$ is a Markovian transition model that specifies the probability $P(\boldsymbol{s}, a, \boldsymbol{s}')$ of transition to a state $\boldsymbol{s}'$ when taken an action a in state $\boldsymbol{s}$; $R : \mathcal{S} \to \mathbb{R}$ is the reward function; and $\gamma \in (0, 1)$ is the discount factor for future rewards. A *stationary policy* $\pi : \mathcal{S} \to \mathcal{A}$ for a MDP is a mapping from states to actions and denotes a mechanism for choosing actions. An *episode* is a sequence of state transitions: $< \boldsymbol{s}_1, a_1, r_1, \boldsymbol{s}_2, \ldots, >$. An agent repeatedly chooses actions until the current episode terminates, and then a new episode starts over again.

The notion of *value function* is of central interest in reinforcement learning tasks. Given a policy π, the value $V^\pi(\boldsymbol{s})$ of a state $\boldsymbol{s}$ is defined as the expected discounted return obtained when starting from this state and all decisions are made according to policy π until the current episode terminates:

$$V^\pi(\boldsymbol{s}) = E_\pi \left[\sum_{t=0}^{\infty} \gamma^t R(\boldsymbol{s}_t) | \boldsymbol{s}_0 = \boldsymbol{s} \right] . \tag{1}$$

As it is well-known the value function satisfy the following recursive equation:

$$V^\pi(\boldsymbol{s}) = E_\pi \left[R(\boldsymbol{s}_t) + \gamma V^\pi(\boldsymbol{s}_{t+1}) | \boldsymbol{s}_t = \boldsymbol{s} \right] . \tag{2}$$

which expresses a relationship between the values of successive states in the same episode and is known as *Bellman's equation*. In the same way, the state-action value function (Q-function) $Q(\boldsymbol{s}, a)$ denotes the expected cumulative reward as received by taking action a in state $\boldsymbol{s}$, and thereafter following the policy π:

$$Q^\pi(\boldsymbol{s}, a) = E_\pi \left[\sum_{t=0}^{\infty} \gamma^t R(\boldsymbol{s}_t) | \boldsymbol{s}_0 = \boldsymbol{s}, a_0 = a \right] . \tag{3}$$

In this study we will mainly focus on Q functions dealing with state-action pairs.

The objective of RL problems is to estimate an optimal policy π^* which is equivalent to finding the optimal state-action value function Q^*:

$$\pi^*(\boldsymbol{s}) = \arg\max_a Q^*(\boldsymbol{s}, a). \tag{4}$$

A common choice for representing the value function is through a linear function approximation using a set of k basis functions $\{\phi_j(\boldsymbol{s}, a)\}_{j=1}^k$:

$$Q(\boldsymbol{s}, a; \boldsymbol{w}) = \phi(\boldsymbol{s}, a)^\top \boldsymbol{w} = \sum_{j=1}^{k} \phi_j(\boldsymbol{s}, a) w_j \ , \tag{5}$$

where $\boldsymbol{w} = (w_1, \ldots, w_k)$ is a vector of weights which are unknown and must be estimated so as to minimize the approximation error. The selection of the basis functions is very important and must be chosen in such a way so as to encode properties of the state and action relevant to the proper determination of the Q values. As we will see later, our method provides an adaptive incremental procedure for discovering appropriate basis functions through on-line clustering.

One of the most popular on-policy TD algorithms is the SARSA [1] which is based on a *bootstrapping* technique. Assuming that an action a_t is taken and the agent moves from belief state $\boldsymbol{s}_t$ to a new state $\boldsymbol{s}_{t+1}$ receiving a reward r_t, a new action a_{t+1} is chosen according to the current policy. Then, the predicted Q value of this new state-action pair and the received reward are used to calculate an improved estimate for the Q value of the previous state-action pair:

$$\delta_t = r_t + \gamma Q(\boldsymbol{s}_{t+1}, a_{t+1}) - Q(\boldsymbol{s}_t, a_t) = r_t + \gamma(\phi(\boldsymbol{s}_{t+1}, a_{t+1}) - \phi(\boldsymbol{s}_t, a_t))^\top \boldsymbol{w}_t \ , \tag{6}$$

which is the one-step temporal-difference (TD) error. This is used for adjusting the weights of the policy by performing a stochastic gradient descent scheme:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha_t \delta_t \nabla_{\boldsymbol{w}} Q(\boldsymbol{s}_t, a_t) \ , \tag{7}$$

where α_t is the learning rate which set to some small value (e.g. 0.05) and can be decreased over time. Additionally, is useful to combine SARSA with the *eligibility traces*, SARSA(λ), allowing the update rule to propagate the TD error backward over the current trajectory of states. This is achieved by modifying the above equation (Eq. 7) as $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha_t \delta_t e_t$, where $e_t = \gamma \lambda e_{t-1} + \nabla_{\boldsymbol{w}} Q(\boldsymbol{s}_t, a_t)$ is a vector of eligibility traces and $\lambda \in [0, 1]$ is the trace-decay parameter.

3 The Proposed Method

The proposed methodology is based on a policy evaluation scheme that incrementally builds a dictionary of basis functions for modelling the value function. This is accomplished by using an on-line clustering scheme that decomposes appropriately an efficient kernel space of the inputs so as to achieve optimal exploration of the value function. To what follows and for simplicity we will assume that the input samples that are generated by the agent are state-action pairs, denoted as $x_i = (\boldsymbol{s}_i, a_i)$. We will also consider that the action space is discrete of size M.

Assuming a given data set of N samples $\{x_1, x_2, \ldots, x_N\}$ the task of clustering aims at partitioning the input set into k disjoint clusters, $c_1, c_2, \ldots, c_k$ containing samples with common properties. The kernel k-means [15,16], which is an extension of the standard k-means algorithm, is based on transforming data to a feature space through appropriate kernel functions and minimizing the clustering error in this space. In particular, the objective function is given by

$$J_k = \sum_{i}^{N} \min_{j=1}^{k} \{-K(x_i, m_j)\} \tag{8}$$

where m_j are some representatives for each cluster. In our study we have considered that every cluster c_j is characterized by the following features:

- $\boldsymbol{\mu}_j$: its centroid in state space $\mathcal{S}$,
- Σ_j: diagonal covariance matrix over the state space,
- $\boldsymbol{p}_j = (p_{j1}, \ldots, p_{jM})$: the density function over the M discrete actions, giving the probabilities of each action ($\sum_{m=1}^{M} p_{jm} = 1$).

These features constitute the representative vector $m_j = (\boldsymbol{\mu}_j, \Sigma_j, \boldsymbol{p}_j)$ of a cluster.

The kernel function $K(x_i, m_j)$ for an arbitrary sample, $x_i = (\boldsymbol{s}_i, a_i)$, with the c_j cluster is derived as a product of two kernels, one for each space:

$$K(x_i, m_j) = K_s(\boldsymbol{s}_i, \boldsymbol{\mu}_j, \Sigma_j) K_a(a_i, \boldsymbol{p}_j) \ . \tag{9}$$

For the state space a Gaussian kernel have been used:

$$K_s(\boldsymbol{s}_i, \boldsymbol{\mu}_j, \Sigma_j) = \exp(-\frac{1}{2}(\boldsymbol{s}_i - \boldsymbol{\mu}_j)^\top \Sigma_j^{-1}(\boldsymbol{s}_i - \boldsymbol{\mu}_j)) \ , \tag{10}$$

while for the action space the kernel function is derived from the probability for this action of the cluster action distribution, i.e.

$$K_a(a_i, \boldsymbol{p}_j) = p_{j,a_i} \ . \tag{11}$$

Note that in fact the action kernel is the cosine similarity between the probability vector for actions and an indicator vector of the input action a_i with zeros in all positions except in the position of a_i where has one.

In our case, the samples are non-stationary and are generated sequentially (i.e. time-varying). On-line clustering provides a framework for constructing recursive learning rules taking into account model evolutions over time. The proposed on-line kernel-based clustering method is performed iteratively as follows: For a random taken data point $x_i = (\boldsymbol{s}_i, a_i)$, the method first selects the winning cluster j^* according to the current kernel values, i.e.

$$j^* = \arg\max_{j=1}^{k} K(x_i, m_j) \ . \tag{12}$$

If the maximum kernel value is less than a predefined threshold K_{min}, then a new cluster is created ($k = k + 1$) by initializing it properly. This is done by setting the state s_i as the cluster centroid, $\boldsymbol{\mu}_k = \boldsymbol{s}_i$, while for the action density probability $\boldsymbol{p}_k$ we give a large value for the action probability of action a_i (e.g. $p_{k,a_i} = 0.8$) and normalize the others so as to hold the constraint, $\sum_m p_{km} = 1$.

The next step is the *adaptation phase* where the prototype m_{j^*} of the winning cluster must be adjusted. This is accomplished by using the next update rules:

$$\boldsymbol{\mu}_{j^*}^{(new)} = \boldsymbol{\mu}_{j^*} + \eta K(x_i, m_j)(\boldsymbol{s}_i - \boldsymbol{\mu}_{j^*}) \ , \tag{13}$$

$$\Sigma_{j^*}^{(new)} = \Sigma_{j^*} + \eta K(x_i, m_j) diag((\boldsymbol{s}_i - \boldsymbol{\mu}_{j^*}^{(new)})(\boldsymbol{s}_i - \boldsymbol{\mu}_{j^*}^{(new)})^\top) \ , \tag{14}$$

$$n_{jm}^{(new)} = \begin{cases} n_{jm} + 1, & \text{if } \ m = a_i \\ n_{jm}, & \text{otherwise} \end{cases} \ , \tag{15}$$

where the term η is the learning rate taking a small value (e.g. 0.05), which can be reduced over time. It must be noted that the density of actions, p_{jm}, is guided by the frequency distribution n_{jm} and thus it is more convenient to keep records of frequencies. Then, the probabilities p_{jm} are calculated by the relative frequencies. From the above rule, it is easily to show that the probability of action a_i will be increased by $(\nu_j - n_{jm})/(\nu_j(\nu_j + 1))$, while the probability of the other $M - 1$ actions will be decreased by $n_{jm}/(\nu_j(\nu_j + 1))$ ($\nu_j = \sum_{m=1}^{M} n_{jm}$ is the total frequency of cluster c_j).

The method starts with a single cluster $k = 1$, where it is initialized as described previously by the first data point collected by the agent $x_1 = (\boldsymbol{s}_1, a_1)$. At every time step the policy evaluation stage uses the k basis functions as

(currently) taken by the clustering procedure. Therefore, the on-line clustering approach provides not only the shape, but also the proper number of basis functions for estimating the value function. At a second level, the linear weights are re-estimated following the temporal difference (TD) learning process, as described previously. The above procedure is repeated until convergence, or the number of episodes reaches a prespecified value. The overall scheme of the proposed methodology is given in Algorithm 1.

Algorithm 1. The proposed method for value function approximation

Start with $k = 1$ and use first point $x_i = (\boldsymbol{s}_1, a_1)$ for initializing it. Set a random value to weight w_1. $t = 0$.
while convergence or maximum number of episodes not found **do**
  Suppose previous input $x_i = (\boldsymbol{s}_i, a_i)$. Observe new state $\boldsymbol{s}_{i+1}$.
  Select action according to the current policy $a_{i+1} = \arg\max_{l=1}^{M} Q(\boldsymbol{s}_{i+1}, l)$.
  Find the winning cluster $j^* = \arg\max_{j=1}^{k} K(x_{i+1}, m_j)$.
  if $K(x_{i+1}, m_{j^*}) < K_{min}$ **then**
    Create a new cluster ($k = k + 1$) and initialize its prototype m_k with x_{i+1}.
    Create a new weight w_k of linear model initialized randomly. $\boldsymbol{w}_t = \boldsymbol{w}_t \cup w_k$.
  else
    Update the prototype m_{j^*} of the winning cluster using Eqs. 13-15.
  end if
  Obtain the new k basis functions as: $\phi_j(\boldsymbol{s}, a) = K((\boldsymbol{s}, a), m_j)\ \forall\ j = 1, \ldots, k$.
  Update the weights $\boldsymbol{w}_t$ of Q function according to Eqs. 6, 7.
  $t = t + 1$
end while

4 Experimental Results

A number of experiments have been conducted using three well-known continuous benchmarks in order to assess the performance of the proposed methodology. These environments can be found on the RL-Glue software which are freely available at http://glue.rl-community.org/. Comparison has been made with a recent method presented in [10] that uses fixed Fourier basis functions of order 3, denoted as 'O(3)Fourier' [1]. In all experiments we have set the discount factor γ equal to 1, the parameter λ equal to 0.9, and the threshold for adding a new cluster as $K_{min} = 0.5$.

The first benchmark is the famous *cart pole* where the objective is to keep the pole balanced and the cart within its limits by applying a fixed magnitude force either to the left, or to the right (two actions). There are four continuous variables: the horizontal position and the velocity of the cart, as well as the angle and the angular velocity of the pole, while the reward received is +1.

[1] Open source code for this method can be found in the RL-Glue library.

The second environment is the *mountain car*, where the objective is to drive an under-powered car up a steep mountain road from a valley to the right tophill using three actions. The state consists of two continuous variables: the position and the current velocity of the car, while at each time step a negative reward $r = -1$ is received.

In the last domain the agent controls a simulated *acrobot* attached by the hands to a fixed location. The goal is to apply torque to the hips of the robot and swing the feet above a pre-specified threshold. Each state characterized by four continuous variables: the angle and the angular velocity of the two joints. The agent can select between three actions: positive torque, negative torque and zero torque on the second joint. A negative reward ($r = -1$) is received at each time step except for the case where the goal is reached ($r = 0$). An episode is terminated only when the goal is reached.

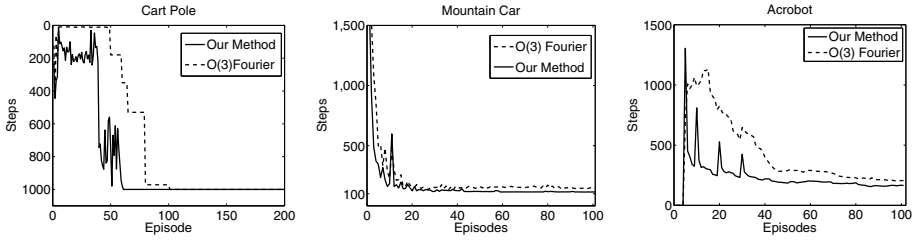

Fig. 1. Comparative results in three simulated environments

The depicted results on these three benchmarks are illustrated in Fig. 1, where each curve gives the number of steps that the agent makes per episode. Note these are the mean curves obtained by 10 runs per problem. As it is obvious our method has the tendency to converge to the optimum solution faster than the 'O(3)Fourier' method that employs (256) fixed Fourier basis function. It is interesting to note that in the case of the 'mountain car' and 'acrobot' environments the proposed method managed to discover better policies.

5 Conclusions and Future Directions

In this study we have presented a novel framework for learning representation of reinforcement learning agents and control in Markov decision processes. An on-line kernel-based clustering approach is used as a mechanism for creating and adjusting clusters over the input state-action pairs generated by the agent. At each step, the current cluster parameters are used for building an efficient kernel space that provides with the appropriate basis functions to the temporal-difference learning framework. Thus, the linear weights used for value function approximation are sequentially adjusted in a more optimal way. The initial results of our method obtained from the comparative study are very promising and promote directions for further research. Since the proposed scheme of constructing basis functions is general, it allows the possibility to study its impact

to other temporal difference algorithms for learning the weights of the value function, such as the Least-Squares Temporal Difference (LSTD) or Gaussian Process Temporal Difference (GPTD). Also, alternative schemes of on-line clustering can be examined, as well as to make an extensive comparison with other value function approximation approaches.

References

1. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press, Cambridge (1998)
2. Kaelbling, L.P., Littman, M.L., Moore, A.W.: Reinforcement learning: A survey. Journal of Artificial Inteligence Research 4, 237–285 (1996)
3. Sutton, R.: Learning to predict by the method of temporal differences. Machine Learning 3(1), 9–44 (1988)
4. Boyan, J.A.: Technical update: Least-squares temporal difference learning. Machine Learning, 233–246 (2002)
5. Lagoudakis, M.G., Parr, R.: Least-squares policy iteration. Journal of Machine Learning Research 4, 1107–1149 (2003)
6. Xu, X., Hu, D., Lu, X.: Kernel-based least squares policy iteration for reinforcement learning. IEEE Transactions on Neural Networks 18(4), 973–992 (2007)
7. Rasmussen, C.E., Kuss, M.: Gaussian processes in reinforcement learning. In: Advances in Neural Information Processing Systems 16, pp. 751–759 (2004)
8. Engel, Y., Mannor, S., Meir, R.: Reinforcement learning with gaussian process. In: International Conference on Machine Learning, pp. 201–208 (2005)
9. Farahmand, A.M., Ghavamzadeh, M., Szepesvári, C., Mannor, S.: Regularized policy iteration. In: NIPS, pp. 441–448 (2008)
10. Konidaris, G.D., Osentoski, S., Thomas, P.S.: Value function approximation in reinforcement learning using the fourier basis. In: AAAI Conf. on Artificial Intelligence, pp. 380–385 (2011)
11. Mahadevan, S.: Samuel meets amarel: Automating value function approximation using global state space analysis. In: AAAI (2005)
12. Mahadevan, S., Maggione, M.: Proto-value Functions: A Laplacian Framework for Learning Repersentation and Control in Markov Decision Porocesses. Journal of Machine Learning Research 8, 2169–2231 (2007)
13. Menache, I., Mannor, S., Shimkin, N.: Basis Function Adaptation in Temporal Difference Reinforcement Learning. Annals of Operations Research 134, 215–238 (2005)
14. Petrik, M.: An analysis of laplacian methods for value function approximation in mdps. In: International Joint Conference on Artificial Intelligence, pp. 2574–2579 (2007)
15. Scholkopf, B., Smola, A.J., Muller, K.-R.: Nonlinear component analysis as a kernel eigenvalue problem. Neural Computation 10(5), 1299–1319 (1998)
16. Tzortzis, G., Likas, A.: The Global Kernel k-Means Clustering Algorithm. IEEE Trans. on Neural Networks 20(7), 1181–1194 (2009)