Incremental Basis Function Expansion in Reinforcement Learning using Cascade-Correlation Networks

Sertan Girgin, Philippe Preux

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Incremental Basis Function Expansion in Reinforcement Learning using Cascade-Correlation Networks

Sertan Girgin — Philippe Preux

N° 6505

Avril 2008

Thème COG

ISSN 0249-6399 ISRN INRIA/RR--6505--FR+ENG

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

centre de recherche
LILLE - NORD EUROPE

Incremental Basis Function Expansion in Reinforcement Learning using Cascade-Correlation Networks

Sertan Girgin*, Philippe Preux†

Thème COG — Systèmes cognitifs
Équipes-Projets Sequel

Rapport de recherche n° 6505 — Avril 2008 — 19 pages

Abstract: In machine learning, in parallel to algorithms themselves, the representation of data is a point of utmost importance. Efforts on data pre-processing in general are a key ingredient to success. An algorithm that performs poorly on a particular form of given data may perform much better, both in terms of efficiency and the quality of the solution, when the same data is represented in another form. Despite the amount of literature on the subject, the issue of how to enrich a representation to suit the underlying mechanism is clearly still pending. In this paper, we approach this problem within the context of reinforcement learning, and in particular, interested in discovery of a "good" representation of data for the LSPI algorithm. To this end, we use the cascade-correlation learning architecture to automatically generate a set of basis functions which would lead to a better approximation of the value function, and consequently improve the performance of the resulting policies. We also show the effectiveness of the idea on some benchmark problems.

Key-words: reinforcement learning, policy iteration, cascade-correlation, basis functions

* sertan.girgin@inria.fr
† philippe.preux@inria.fr

Centre de recherche INRIA Lille – Nord Europe
Parc Scientifique de la Haute Borne
40, avenue Halley, 59650 Villeneuve d'Ascq
Téléphone : +33 3 59 57 78 00 — Télécopie : +33 3 59 57 78 50

Expansion incrémentale des fonctions de base en apprentissage par renforcement utilisant un réseau de neurones cascadés

Résumé : En apprentissage automatique, en plus des algorithmes eux-mêmes, la représentation des données est un point d'une très grande importance. Les efforts effectués au niveau du pré-traitement sont un ingrédient clé du succès. Sur une tâche donnée et un jeu de données fixé, un algorithme très peu performant peut produire d'excellents résultats si la représentation des données est changée, que ce soit en terme d'efficacité de l'algorithme, ou en termes de qualité de la solution. Malgré la littérature conséquente sur le sujet, le problème du choix de la représentation des données est toujours ouvert. Dans ce papier, nous proposons d'approcher le problème dans le contexte de l'apprentissage par renforcement. En particulier, nous nous intéressons à la découverte d'une "bonne" représentation des données. Nous mettons en oeuvre cette approche avec l'algorithme LSPI. Pour cela, nous utilisons un réseau de neurones cascadés (cascade-correlation network) qui génère automatiquement un ensemble de fonctions de base qui mène à une meilleure approximation de la fonction valeur, et, par voie de conséquence, à des politiques de meilleures qualité. Nous montrons que cette approche est praticable sur un ensemble de problèmes.

Mots-clés : apprentissage par renforcement, itération de la politique, réseau de neurones cascadés, fonctions de base

1 Introduction

Reinforcement learning (RL) is the problem faced by an agent that is situated in an environment and must learn a particular behavior through repeated trial-and-error interactions with it [15]; at each time step, the agent observes the state of the environment, chooses its action based on these observations and in return receives some kind of "reward", in other words a *reinforcement signal*, from the environment as feedback. The aim of the agent is to find a policy, a way of choosing actions, that maximizes its overall gain – a function of rewards, such as the (discounted) sum or average over a time period. RL has been and is being extensively studied under different settings (online / offline, discrete / continuous state-action spaces, discounted / average reward, perfect / imperfect state information; etc.) and various methods and algorithms (dynamic programming, Monte Carlo methods, temporal-difference learning, etc.) have been proposed. One point common to all such approaches is that, regardless of how they do it what is being produced as a solution is a mapping from *inputs*, observations, to *outputs*, actions. It is natural that different approaches may require or prefer the input data be in different forms, but still the same data can be represented in many different ways conforming to the specified form. This brings up the questions of what the best representation of input data is for a given method or algorithm, and how it can be found.

In particular, in this paper, we will focus on Least-Squares Policy Iteration (LSPI) algorithm [7] which uses a linear combination of basis functions to approximate a state-action value function and learn a policy from a given set of experience samples. The basis functions map state-action tuples into real numbers, and each basis function performs a mapping that is different from the others. From the point of view of LSPI, the real input becomes the values of basis functions evaluated for given state-action pairs. Therefore, the set of basis functions that is being employed directly affects the quality of the solution, and the question transforms into "what is the set of best basis functions?". We seek to provide a possible answer to this question by proposing a method that incorporates a cascade correlation learning architecture into LSPI and iteratively adds new basis functions to a set of initial basis functions. In Section 2, we first introduce policy iteration and the LSPI algorithm. Sections 3 describes cascade correlation learning architecture followed by the details of the proposed method for basis function expansion in Section 4. Section 5 presents empirical evaluations on some benchmark problems, and a review of related work can be found in Section 6. Finally, Section 7 concludes.

2 Least-Squares Policy Iteration

A *Markov decision process* (MDP) is defined as tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$ where $\mathcal{S}$ is a set of states, $\mathcal{A}$ is a set of actions, $\mathcal{P}(s, a, s')$ is the transition function which denotes the probability of making a transition from state s to state s' by taking action a, $\mathcal{R}(s, a)$ is the expected reward function, and γ is the discount factor that determines the importance of future rewards. A *policy* is a probability distribution over actions conditioned on the state; $\pi(s, a)$ denotes the probability that policy π selects action a at state s. An optimal policy maximizes the expected total discounted reward from any initial state if followed. *Policy*

iteration is a general framework for finding an optimal policy; starting from an initial policy, two basic steps, namely *policy evaluation* followed by *policy improvement*, are applied consecutively and iteratively until convergence to an optimal policy is achieved or certain stopping criteria is met. Let π_i be the policy at iteration i. Policy evaluation step consists of finding the state value function,

$$V^{\pi_i}(s) = E_{a_t \sim \pi_i}\big[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s\big]$$

or the state-action value function,

$$Q^{\pi_i}(s,a) = E_{a_t \sim \pi_i, t>0}\big[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s, a_0 = a\big]$$

of the current policy, where a_t is the action chosen by π_i and r_t is the reward received at time t. In some restricted cases[1], this can be accomplished by solving numerically or analytically the system of the Bellman equations

$$Q^{\pi_i}(s,a) = \mathcal{R}(s,a) + \gamma \int_{\mathcal{S}} \mathcal{P}(s,a,s') V^{\pi_i}(s') ds' \quad (1)$$

$$V^{\pi_i}(s) = \int_{\mathcal{A}} \pi_i(s,a) Q^{\pi_i}(s,a) da \quad (2)$$

The integrals over state and action spaces are replaced by finite sums in the discrete case. The right-hand side of the equations applied to a given state(-action) value function for a policy π define the *Bellman operators* denoted by T_π. Alternatively, one can employ *Monte Carlo methods* by sampling multiple trajectories (i.e. a sequence $s_0 a_0 r_0 s_1 \ldots$ of states, actions and rewards) and evaluate the expectations by taking the average value over such roll-outs, or use *temporal difference learning* in which the Bellman equations are considered as update rules and state(-action) value function estimate is successively updated based on the previous estimates. In the policy improvement step, the state(-action) value function obtained in the policy evaluation step is used to derive a new policy π_{i+1} which would perform at least as good as π_i, i.e. satisfies $V^{\pi_{i+1}}(s) \geq V^{\pi_i}(s)$ for all $s \in \mathcal{S}$. For a deterministic policy, this can be realized by defining π_{i+1} greedy with respect to π_i as

$$\pi_{i+1}(s) = \arg\max_{a \in \mathcal{A}} Q^{\pi_i}(s,a) \quad (3)$$

Policy iteration is guaranteed to converge to an optimal policy in a finite number of steps if both state and action spaces are finite, the value function and the policy are represented perfectly and the policy evaluation step is solved exactly. However, in most cases it may not be possible to fulfill these requirements (eg. in problems with large or infinite state and action spaces) and the value function and/or the policy need to be approximated, leading to the so-called *approximate policy iteration* approach. It has been shown that if the error in the approximations are bounded then approximate policy iteration generates policies such that their performance is also bounded with respect to the optimal

[1] Such as problems with finite and small state-action spaces, or linear quadratic optimal control problems in which the underlying MDP model is known.

policy, yet the performance bound can be arbitrarily large as the discount factor, γ, gets closer to 1 [2, 11].

Least-Squares policy iteration (LSPI) is an off-line and off-policy approximate policy iteration algorithm proposed by Lagoudakis and Parr (2003). Rather than relying on continual interactions with the environment or a generative model, it works on a fixed set of samples collected arbitrarily. Each sample is of the form (s, a, r, s') indicating that executing action a at state s resulted in a transition to state s' with an immediate reward of r. The state-action value function is approximated by a linear form

$$\widehat{Q}^{\pi}(s,a) = \sum_{j=0}^{m-1} w_j \phi_j(s,a)$$

where $\phi_j(s,a)$ denote the *basis functions* and w_j are the parameters. Basis functions, also called *features*, are arbitrary functions of state-action pairs, but are intended to capture the underlying structure of the target function and can be viewed as doing dimensionality reduction from a larger space to $\Re^m$. Typical examples of basis functions are polynomials of a given degree or radial basis functions, such as Gaussian and multiquadratics, each possibly associated with a different center and scale. Note that, in the case of discrete state-action spaces, this generic form also includes tabular representations as a special case, but in general the number of basis functions, m, is much smaller compared to $|\mathcal{S}|\,|\mathcal{A}|$.

Instead of representing the policy explicitly, LSPI opts to determine the action that is imposed by the current policy at a given state by directly evaluating Eq. 3 (hence the policy improvement step becomes inherent). This may not be a feasible operation when the number of actions is large or possibly infinite, except in certain cases where a closed form solution is achievable; however, in most problems this drawback may not pose a significant problem as the action space is generally less susceptible to discretization compared to the state space.

Given a policy π_{i+1}, greedy with respect to $\widehat{Q}^{\pi_i}$ which is defined by the set of parameters $w_j^{\pi_i}$, LSPI performs the policy evaluation step and determines $\widehat{Q}^{\pi_{i+1}}$, in other words the corresponding set of new parameters $w_j^{\pi_{i+1}}$, by invoking an algorithm called LSTDQ. One can easily observe that by definition the state-action value function Q^{π} of a policy π is necessarily a fixed point of the Bellman operator, i.e. $Q^{\pi} = T_{\pi} Q^{\pi}$ (Eq. 1), which also holds for $Q^{\pi_{i+1}}$. Due to the specific choice of linear function approximation, any $\widehat{Q}^{\pi}$ is confined to the subspace spanned by the basis functions $\{\phi_j\}$, and therefore, an approximation $\widehat{Q}^{\pi_{i+1}}$ to $Q^{\pi_{i+1}}$ which stays invariant under the Bellman operator may not exist. Instead, LSTDQ tries to find an approximation $\widehat{Q}^{\pi_{i+1}}$ which is equal to the orthogonal projection of its image under the Bellman operator. Such $\widehat{Q}^{\pi_{i+1}}$ also possesses the property that

$$\widehat{Q}^{\pi_{i+1}} = \arg\min_{\widehat{Q}^{\pi}} \| T_{\pi_{i+1}} \widehat{Q}^{\pi_{i+1}} - \widehat{Q}^{\pi} \|_2$$

A motivation for choosing this particular approximation is expressed as the expectation that the approximation should be close to the projection of $Q^{\pi_{i+1}}$ onto the subspace spanned by the basis functions if subsequent applications of the Bellman operator point in a similar direction. Without going into the details, given any N samples, LSTDQ finds $\widehat{Q}^{\pi_{i+1}}$ by solving the $m \times m$ system

$$\widetilde{A} w^{\pi_{i+1}} = \widetilde{b}$$

where $\phi(s,a) = [\phi_0(s,a)\phi_1(s,a)\dots\phi_{m-1}(s,a)]^\top$ is the row vector of basis functions evaluated at (s,a), and

$$\widetilde{A} = \frac{1}{N}\sum_{i=1}^{N}\left[\phi(s_i,a_i)\Big(\phi(s_i,a_i) - \gamma\phi(s_i',\pi_{i+1}(s_i'))\Big)^\top\right],$$

$$\widetilde{b} = \frac{1}{N}\sum_{i=1}^{N}\phi(s_i,a_i)r_i,$$

both of which in the limit converge to the matrices of the least-squares fixed-point approximation obtained by replacing $\widehat{Q}^{\pi_{i+1}} = \Phi w^{\pi_{i+1}}$ in the system

$$\widehat{Q}^{\pi_{i+1}} = \Phi(\Phi^\top\Phi)^{-1}\Phi^\top(T_{\pi_{i+1}}\widehat{Q}^{\pi_{i+1}})$$

Here, $\Phi(\Phi^\top\Phi)^{-1}\Phi^\top$ is the orthogonal projection and Φ denotes the matrix of the values of the basis functions evaluated for the state-action pairs. The details of the derivation can be found in the seminal paper [7].

Algorithm 1 The LSPI algorithm.

input Set of samples, D, number of basis functions, m, basis functions, $\vec{\phi}$, discount factor, γ, stopping criterion, ϵ.
output w^π
  Initialize weights w^π.
  $i \leftarrow 0$
  repeat
    $i \leftarrow i+1$
    $w^{\pi_i} \leftarrow w^\pi$
    $w^\pi \leftarrow \texttt{LSTDQ}(D, m, \vec{\phi}, \gamma, w^{\pi_i})$
  until $\|w^{\pi_i} - w^\pi\| < \epsilon$ {weights have converged}

The LSPI algorithm presented in Algorithm 1 has been demonstrated to provide "good" policies within relatively small number of iterations. Furthermore, as it is possible to use a single and common sample set for all policy evaluations, LSPI makes efficient use of the available data; as such, it is quite suitable for problems in which data gathering process is time consuming and costly. However, the quality of the resulting policies depends on two important factors: the basis functions, and the distribution of the samples.

In an offline setting, one may not have any control on the set of samples and too much bias in the samples would inevitably reduce the performance of the learned policy. On the other hand, in an on-line setting, LSPI allows different sample sets to be employed at each iteration; thus, it is possible to fine tune the trade-off between exploration and exploitation by collecting new samples using the current policy.

Regarding the basis functions, the user is free to choose any set of functions as long as they are linearly independent (a restriction which can be relaxed in most cases by applying singular value decomposition). As shown in [7], and in accordance with the generic performance bound on policy iteration, if the error between the approximate and the true state-action value functions at each iteration is bounded by a scalar ϵ, then in the limit the error between the optimal state-action value function and those corresponding to the policies generated by

LSPI is also bounded by a constant multiple of ϵ. Therefore, selecting a "good" set of basis functions has a significant and direct effect on the success of the method.

In general, the set of basis functions is defined by the user based on domain knowledge, and usually in a trial and error fashion. They can either be fixed, or one can start from an initial subset of predefined basis functions and iteratively introduce remaining functions based on the performance of the current set, so-called *feature iteration approach* [1]. However, as the complexity of the problem increases it also gets progressively more difficult to come up with a good set of basis functions. Generic approaches, such as regular grids or regular radial basis function networks, which are quite successful in small problems, become impractical due to the exponential growth of the state-action spaces with respect to their dimension. Therefore, given a problem, it is highly desirable to determine a compact set of such basis functions automatically. In the next section, we will first describe a particular class of function approximator called *cascade-correlation networks*, and then present how they can be utilized in the LSPI algorithm to iteratively expand the set of basis functions.

3 Cascade Correlation Networks

Cascade correlation is both an architecture and a supervised learning algorithm for artificial neural networks introduced by [4]. It aims to overcome *step-size* and *moving target* problems that negatively affect the performance of back-propagation learning algorithm. Similar to traditional neural networks, the neuron is the most basic unit in cascade correlation networks. However, instead of having a predefined topology with the weights of the fixed connections between neurons getting adjusted, a cascade correlation network starts with a minimal structure consisting only of an input and an output layer, without any hidden layer. All input neurons are directly connected to the output neurons (Figure 1a). Then, the following steps are taken:

1. All connections leading to output neurons are trained on a sample set and corresponding weights (i.e. only the input weights of output neurons) are determined by using an ordinary learning algorithm until the error of the network no longer decreases. This can be done by applying the regular "delta" rule, or using more efficient methods such as quick-prop or rprop. Note that, only the input weights of output neurons (or equivalently the output weights of input neurons) are being trained, therefore there is no back-propagation.

2. If the accuracy of the network is above a given threshold then the process terminates.

3. Otherwise, a set of *candidate units* is created. These units typically have non-linear activation functions, such as sigmoid or Gaussian. Every candidate unit is connected with all input neurons and with all existing hidden neurons (which is initially empty); the weights of these connections are initialized randomly. At this stage the candidate units are not connected to the output neurons, and therefore are not actually active in the network. Let s denote a training sample. The connections leading to a candidate

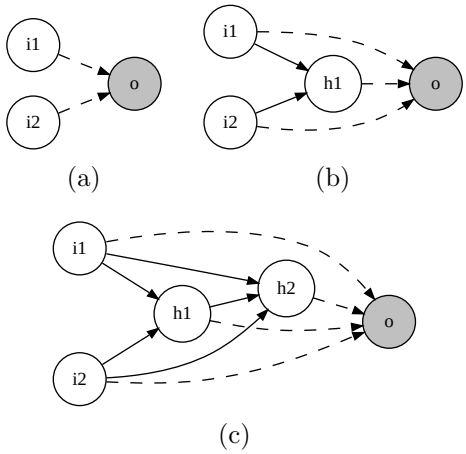

Figure 1: (a) Initial configuration of a simple cascade-correlation network with two inputs and a single output (in gray). (b) and (c) show the change in the structure of the network as two new hidden nodes are subsequently added. Solid edges indicate input weights that stay fixed after the candidate training phase.

unit are trained with the goal of maximizing the sum $\mathcal{S}$ over all output units o of the magnitude of the correlation between the candidate units value denoted by v_s, and the residual error observed at output neuron o denoted by $e_{s,o}$. S is defined as

$$S = \sum_o |\sum_s (v_s - v)(e_{s,o} - e_o)|$$

where v and e_o are the values of v_s and $e_{s,o}$ averaged over all samples, respectively. As in step 1, learning takes place with an ordinary learning algorithm by performing gradient ascent with respect to each of the candidate units incoming weights:

$$\frac{\partial S}{\partial w_i} = \sum_{s,o} \sigma_o (e_{s,o} - e_o) f'_s I_{i,s}$$

where σ_o is the sign of the correlation between the candidates value and output o, f'_s is the derivative for sample s of the candidate units activation function with respect to the sum of its inputs, and $I_{i,s}$ is the input the candidate unit received from neuron i for sample s. Note that, since only the input weights of candidate units are being trained there is again no need for back-propagation. Besides, it is also possible to train candidate units in parallel since they are not connected to each other. By training multiple candidate units instead of a single one, different parts of the weight space can be explored simultaneously. This consequently increases the probability of finding neurons that are highly correlated with the residual error. The learning of candidate unit connections stops when the correlation scores no longer improve or after a certain number of passes over the training set. Now, the candidate unit with the maximum correlation is chosen, its incoming weights are frozen (i.e. they are not updated

in the subsequent steps) and it is added permanently to the network by connecting it to all output neurons (Figure 1b and c). The initial weights of these connections are determined based on the value of correlation of the unit. All other candidate units are discarded.

4. Return back to step 1.

Until the desired accuracy is achieved at step 2, or the number of neurons reaches a given maximum limit, a cascade correlation network completely self-organizes itself and grows as necessary. One can easily observe that, by adding hidden neurons one at a time and freezing their input weights, training of both the input weights of output neurons (step 1) and the input weights of candidate units (step 3) reduce to one step learning problems. Since there is no error to back-propagate to previous layers the moving target problem is effectively eliminated. Also, by training candidate nodes with different activation functions and choosing the best among them, it is possible to build a more compact network that better fits the training data.

One observation here is that, unless any of the neurons has a stochastic activation function, the output of a neuron stays constant for a given sample input. This brings the possibility of storing the output values of neurons which in return reduces the number of calculations in the network and improve the efficiency drastically compared to traditional multi-layer back-propagation networks, especially for large data sets. But more importantly, *each hidden neuron effectively becomes a permanent feature detector*, or to put it another way, a basis function in the network; the successive addition of hidden neurons in a cascaded manner allows, and further, facilitates the creation of more complex feature detectors that helps to reduce the error and better represent the functional dependency between input and output values. We would like to point out that, this entire process does not require any user intervention and is well-matched to our primary goal of determining a set of good basis functions for function approximation is RL, in particular within the scope of LSPI algorithm. We will now describe our approach for realizing this.

4 Using Cascade Correlation Learning Architecture in LSPI

As described in Section 2, in LSPI the state-action value function of a given policy is approximated by a linear combination of basis functions. Our aim here is to employ cascade correlation networks as function approximators and at the same time use them to find useful basis functions in LSPI. Given a reinforcement learning problem, suppose that we have a set of state-action basis functions $\Phi = \{\phi_1(s,a), \phi_2(s,a), \ldots, \phi_m(s,a)\}$. Using this basis functions and applying LSPI algorithm on a set of collected samples of the form (s, a, r, s'), we can find a set of parameters w_i together with an approximate state-action value function

$$\widehat{Q}(s,a) = \sum_{i=1}^{m} w_i \phi_i(s,a)$$

and derive a policy $\widehat{\pi}$ which is greedy with respect to $\widehat{Q}$. Let $\mathcal{N}$ be a cascade correlation network with m inputs and a single output having linear activation

function (i.e. identity function). In this case, the output of the network is a linear combination of the activation values of input and hidden neurons of the network weighted by their connection weights. Initially, the network doesn't have any hidden neurons and all input neurons are directly connected to the output neuron. Therefore, by setting the activation function of the i^{th} input neuron to ϕ_i and the weight of its connection to the output neuron to w_i, $\mathcal{N}$ becomes functionally equivalent to $\widehat{Q}$ and outputs $\widehat{Q}(s,a)$ when all input neurons receive the (s,a) tuple as their input.

Now, the Bellman operator T_π is known to be a contraction in L_∞ norm, that is for any state-action value function Q, $T_\pi Q$ is closer to Q^π in the L_∞ norm, and in particular as mentioned in Section 2, Q^π is a fixed point of T_π. Ideally, a good approximation would be close to its image under the Bellman operator. As opposed to the Bellman residual minimizing approximation, least-squares fixed-point approximation, which is at the core of the LSPI algorithm, ignores the distance between $T_{\widehat{\pi}}\widehat{Q}$ and $\widehat{Q}$ but rather focuses on the direction of the change. Note that, if the true state-action value function Q^π lies in the subspace spanned by the basis functions, that is the set of basis functions is "rich" enough, fixed-point approximation would be solving the Bellman equation and the solution would also minimize the magnitude of the change. This hints that, within the scope of LSPI, one possible way to drive the search towards solutions that satisfy this property could be to expand the set of basis functions by adding new basis functions that are likely to reduce the distance between the found state-action value function $\widehat{Q}$ and $T_{\widehat{\pi}}\widehat{Q}$ over the sample set.

For this purpose, given a sample (s,a,r,s'), in the cascade correlation network we can set $r+\gamma\widehat{Q}(s',\widehat{\pi}(s'))$ as the target value for (s,a) tuple, and train candidate units that are highly correlated with the residual output error, i.e. $\widehat{Q}(s,a) - \Big(r+\gamma\widehat{Q}(s',\widehat{\pi}(s'))\Big)$. At the end of the training phase, the candidate unit having the maximum correlation is added to the network by transforming it into a hidden neuron, and becomes the new basis function ϕ_{m+1}; $\phi_{m+1}(s,a)$ can be calculated by feeding (s,a) as input to the network and determining the activation value of the hidden neuron. Through another round of LSPI learning, one can obtain a new least-squares fixed-point approximation to the state-action value function $\widehat{Q}'(s,a) = \sum_{i=1}^{m+1} w'_i\phi_i(s,a)$ which is more likely to be a better approximation also in the sense of Bellman residual minimization. The network is then updated by setting the weights of connections leading to the output neuron to w'_i for each basis function. This process can be repeated, introducing a new basis function at each iteration, until the error falls below a certain threshold, or a policy with adequate performance is obtained. We can regard this as a hybrid learning system, in which the weights of connections leading to the output neuron of the cascade correlation network are being regulated by the LSPI algorithm. Note that, the values of all basis functions for a given (s,a) tuple can be found with a feed-forward run over the network, and as stated before can be cached for efficiency reasons if desired. The complete algorithm that incorporates the cascade correlation network and basis function expansion to LSPI is presented in Algorithm 2.

One possible problem that may emerge with the proposed method is that, especially when the sample set is small, with increasingly complex basis functions there may be over-fitting, and the on-line performance of the resulting policy may degrade. This can be avoided by increasing the amount of samples,

Algorithm 2 The LSPI algorithm with basis function expansion using cascade correlation network.

input Set of samples, D, number of basis functions, m, basis functions, $\vec{\phi}$, discount factor, γ, stopping criterion for LSPI, ϵ, number of candidate units, n.
output w^π
Create a cascade correlation network $\mathcal{N}$ with m inputs and a single output, and set activation functions of input neurons to ϕ_i.
repeat
$w^\pi \leftarrow LSPI(D, m, \phi, \gamma, \epsilon)$
Set the weight of connections in $\mathcal{N}$ leading to the output neuron to w_i^π.
Calculate the residual output error, $\widehat{Q}(s,a) - (r + \gamma \widehat{Q}(s', \widehat{\pi}(s')))$
Train n candidate units on $\mathcal{N}$.
$\kappa \leftarrow$ Candidate unit having the maximum correlation with the residual output error.
Add κ to $\mathcal{N}$.
$m \leftarrow m + 1$
$\phi_m \leftarrow \kappa$ {Function represented by κ}
until termination condition is satisfied

or alternatively a cross-validation approach can be ensued. Suppose that for a particular reinforcement learning problem, we are given multiple sample sets. The intuition is that a set of "good" basis functions should give rise to "good" policies and similar value functions for all sample sets. By applying LSPI algorithm independently on each sample set but training a single set of candidate units over all sample sets, in other words having a common set of basis functions, one can obtain basis functions, and consequently policies, that are less biased to training data.

5 Experiments

We have evaluated the proposed method on three problems: chain walk [7], pendulum swing-up and multi-segment swimmer. Chain walk is an MDP consisting of a chain of n states. There are two actions, $left$ and $right$, which succeed with probability 0.9, or else fail, moving to the state in the opposite direction. The first and last states are dead-ends, i.e. going left at the first state, or right at the last state revert back to the same state. The reward is 1 if an action ends up in a predefined set of states, and 0 otherwise. The pendulum swing-up and multi-segment swimmer problems are dynamical systems where the state is defined by the position and velocity of the elements of the system, and actions (applied forces) define the next state. These are non-linear control tasks with continuous state spaces. In pendulum, the aim is to keep a simple pendulum actuated by a bounded torque in vertical upright position. Since the torque available is not sufficient, the controller has to swing the pendulum back and forth to reach the goal position. The state variables are the angle of the pendulum and its angular speed. We used two discrete actions, applying a torque of -5, and 5. The reward is defined as the cosine of the angle of the pole. In *swimmer*, a swimmer moving in a two dimensional pool is being simulated.

The swimmer is made of n ($n \geq 3$) segments connected to each other with $n-1$ joints. The goal is to swim as fast as possible to the right by applying torques to these joints and using the friction of the water. There are $2n+2$ state variables consisting of (i) horizontal and vertical velocities of the center of mass of the swimmer, (ii) n angles of its segments with respect to vertical axis and (iii) their derivatives with respect to time; the actions are the $n-1$ torques applied at segment joints. The reward is equal to the horizontal velocity of the swimmer. The system dynamics and more detailed information about Swimmer problem can be found in [3].

In all problems, we started from a set of basis functions consisting of the following:

1. a constant bias function (i.e. 1),
2. a basis function for each one of the state variables, which returns the normalized value of that variable, and
3. a basis function for each possible value of each control variable, which returns 1 if the corresponding control variable in the state-action tuple is equal to that value, and 0 otherwise.

Therefore, the number of the initial basis functions were 4 (1+1+2), 5 (1+2+2) and $3+4n$ for chain, pendulum and swimmer problems respectively, where n is the number of swimmer segments. In the LSPI algorithm, we set $\epsilon = 0.0001$ and limit the number of iterations to 20. The samples for each problem are collected by running a random policy, which uniformly selects one of possible actions, for certain number of time steps (or episodes). In cascade correlation network, we trained an equal number of candidate units having Gaussian and sigmoid activation functions using RPROP method [13]. In RPROP, instead of directly relying on the magnitude of the gradient for the updates (which may lead to slow convergence or oscillations depending on the learning rate), each parameter is updated in the direction of the corresponding partial derivative with an individual time-varying value. The update values are determined using an adaptive process that depends on the change in the sign of the partial derivatives. We allowed at most 100 passes over the sample set during the training of candidate units, and employed the following parameters: $\triangle_{min} = 0.0001, \triangle_{ini} = 0.01, \triangle_{max} = 0.5, \eta^- = 0.5, \eta^+ = 1.2$.

Figure 2 shows the results for the 50-state (numbered from 0 to 49) chain problem using 5000 samples from a single trajectory. Reward is given only in states 9 and 40, therefore the optimal policy is to go right in states 0-8 and 25-40, and left in states 9-24 and 41-49. The number of candidate units was 4. In [7], using 10000 samples LSPI fails to converge to the optimal policy with polynomial basis function of degree 4 for each action, due to the limited representational capability, but succeeds with a radial basis function approximator having 22 basis functions. Using cascade-correlation basis expansion, after 10 basis functions near-optimal policies can be obtained and after 20 basis functions it converges to the optimal policy. The basis functions start from simpler ones and get more complex in order to better fit the target function. A set of basis functions generated in a sample run are presented in Figure 3.

The results for the pendulum problem is presented in Figure 4. For this problem, we again collected 5000 samples restarting from a random configuration every 20 time steps, and trained 10 candidate units. As new basis functions

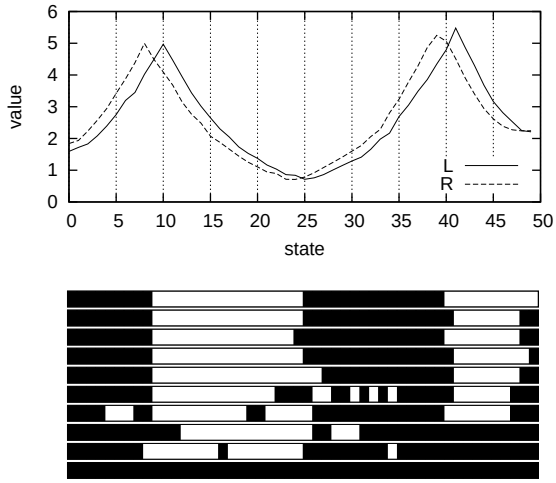

Figure 2: 50-state chain. (State-action value functions with 20 basis functions (top), and policy after every 2 basis functions (bottom).

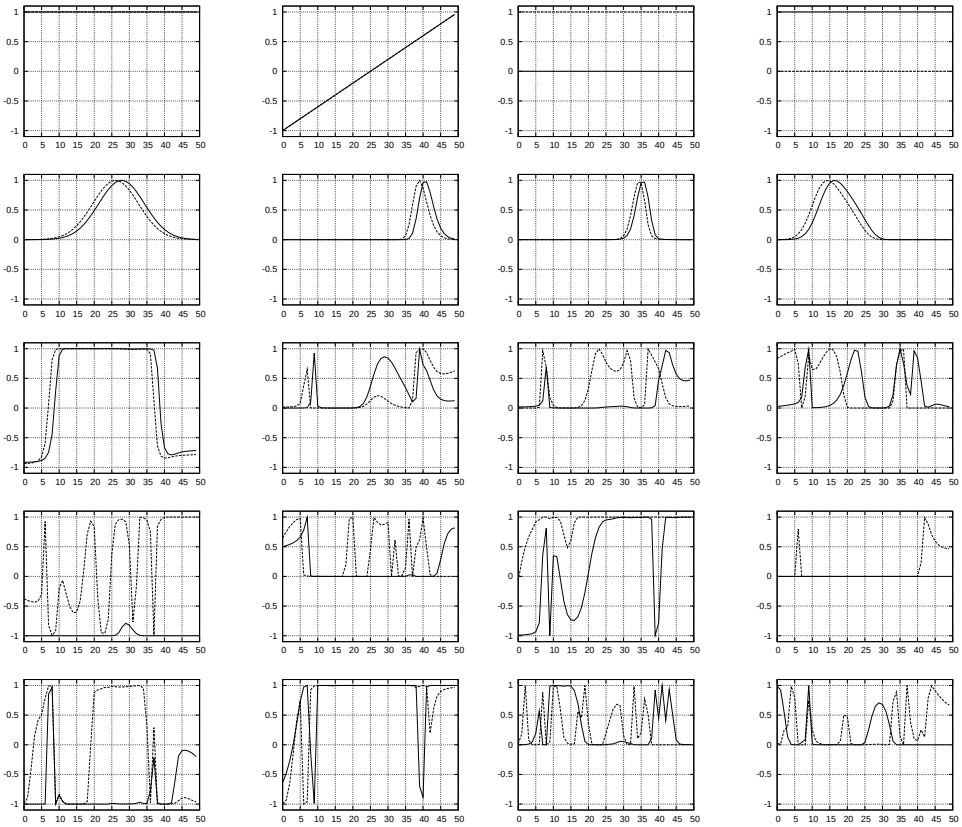

Figure 3: Examples of basis functions for the 50-state chain problem.

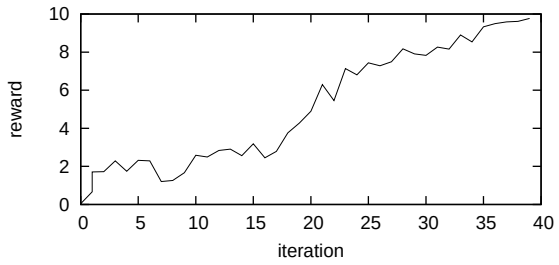

Figure 4: The progress of learned policies after each new basis function in the pendulum problem.

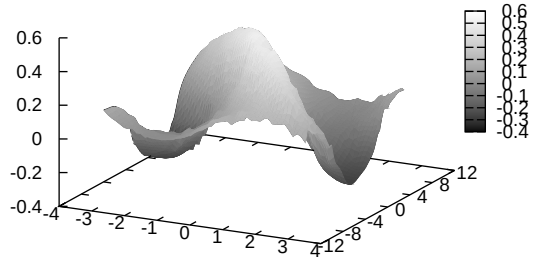

Figure 5: $\hat{Q}(s, \hat{\pi}(s))$ for the pendulum problem where $\hat{\pi}$ is the resulting policy.

are added, the performance of the policies found by LSPI algorithm improves consistently attaining near-optimal levels. Also, the value function obtained after 40 iterations is very close to the true one and successfully captures the shape of the function including the sharp edges around the ridge (Figure 5).

We observed a similar behavior on more complex 5-segment swimmer problem as presented in Figure 6. For this problem, we collected 100000 samples restarting from a random configuration every 50 time steps. The number of trained candidate units was 10 as in the pendulum problem, but we allowed RPROP to make more passes (a maximum of 200) over the sample set.

6 Related Work

Basis function, or feature, selection and generation is essentially an information transformation problem; the input data is converted into another form that "better" describes the underlying concept and relationships, and "easier" to process by the agent. As such, it can be applied as a preprocessing step to a wide range of problems and have been in the interest of the data-mining community, in particular for classification tasks. Following the positive results obtained using efficient methods that rely on basis functions (mostly, using

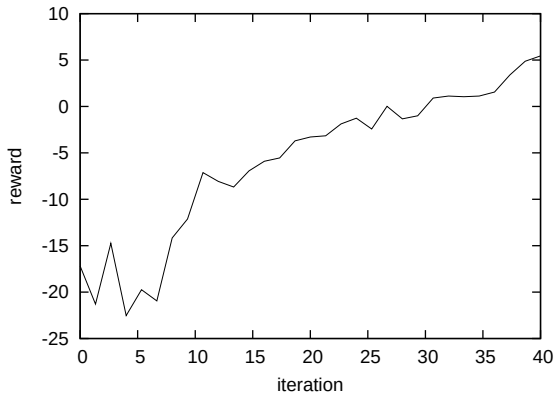

Figure 6: The results for the 5-segment swimmer.

linear approximation architectures) in various domains, it also recently attracted attention from the RL community.

In [10], Menache et al. examined adapting the parameters of a fixed set of basis functions (i.e, center and width of Gaussian radial basis functions) for estimating the value function of a fixed policy. In particular, for a given set of basis function parameters, they used $LSTD(\lambda)$ to determine the weights of basis functions that approximate the value function of a fixed control policy, and then applied either a local gradient based approach or global cross-entropy method to tune the parameters of basis functions in order to minimize the Bellman approximation error in a batch manner. The results of experiments on a grid world problem show that cross-entropy based method performs better compared to the gradient based approach.

In [6], Keller et al. studied automatic basis function construction for value function approximation within the context of LSTD. Given a set of trajectories and starting from an initial approximation, they iteratively use neighborhood component analysis to find a mapping from the state space to a low-dimensional space based on the estimation of the Bellman error, and then by discretizing this space aggregate states and use the resulting aggregation matrix to derive additional basis functions. This tends to aggregate states that are close to each other with respect to the Bellman error, leading to a better approximation by incorporating the corresponding basis functions.

In [12], Parr et al. showed that for linear fixed point methods, iteratively adding basis functions such that each new basis function is the Bellman error of the value function represented by the current set of basis functions forms an orthonormal basis with guaranteed improvement in the quality of the approximation. However, this requires that all computations are exact, in other words, are made with respect to the precise representation of the underlying MDP. They also provide conditions for the approximate case, where progress can be ensured for basis functions that are sufficiently close to the exact ones. Their application in the approximate case on LSPI is closely related to our work, but differs in the sense that a new basis function for each action is added

at each policy-evaluation phase by directly using locally weighted regression to approximate the Bellman error of the current solution.

In contrast to these approaches that make use of the approximation of the Bellman error, including ours, the work by Mahadevan et al. aims to find policy and reward function independent basis functions that captures the intrinsic domain structure that can be used to represent any value function [8, 5, 9]. Their approach originates from the idea of using manifolds to model the topology of the state space; a state space connectivity graph is built using the samples of state transitions, and then eigenvectors of the (directed) graph Laplacian with the smallest eigenvalues are used as basis functions. These eigenvectors possess the property of being the smoothest functions defined over the graph and also capture the nonlinearities in the domain, which makes them suitable for representing smooth value functions.

To the best of our knowledge, the use of cascade correlation networks in reinforcement learning has rarely been investigated before. One existing work that we would like to mention is by Rivest and Precup (2003), in which a cascade correlation network together with a lookup-table is used to approximate the value function in an on-line temporal difference learning setting [14]. It differs from our way of utilizing the cascade correlation learning architecture to build basis functions in the sense that in their case, cascade correlation network purely functions as a cache and an approximator of the value function, trained periodically at a slower scale using the state-value tuples stored in the lookup-table.

7 Conclusion

In this paper, we explored a new method that combines cascade correlation learning architecture with least-squares policy iteration algorithm to find a set of basis function that would lead to a better approximation of the state-action value function, and consequently results in policies with better performance. The experimental results indicate that it is effective in discovering such functions. An important property of the proposed method is that the basis function generation process requires little intervention and tuning from the user.

In the proposed method, LSPI is run to completion at each iteration, and then a new basis function is generated using the cascade correlation training (Algorithm 2). This benefits from a better approximation for the current set of basis functions. An alternative approach would be to add new basis functions within the LSPI loop after the policy evaluation step. This may lead to better intermediate value functions and steadier progress towards the optimal solution, but the resulting basis functions may not be useful at later iterations. It is also possible to combine both approaches by temporarily adding new basis functions within the LSPI loop and then discarding them. We are currently investigating these possibilities.

Although, our focus was on LSPI algorithm in this paper, the approach is in fact more general and can be applied to other reinforcement learning algorithms that approximate the state(-action) value function with a linear architecture. We pursue future work in this direction and also apply the method to more complex domains.

References

[1] D. Bertsekas and S. Ioffe. Temporal differences-based policy iteration and applications in neuro-dynamic programming. Technical Report LIDS-P-2349, MIT, 1996.

[2] D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, Belmont, MA, 1996.

[3] Rémi Coulom. *Reinforcement Learning Using Neural Networks with Applications to Motor Control*. PhD thesis, Institut National Polytechnique de Grenoble, 2002.

[4] S. E. Fahlman and C. Lebiere. The cascade-correlation learning architecture. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 2, pages 524–532, Denver 1989, 1990. Morgan Kaufmann, San Mateo.

[5] Jeff Johns and Sridhar Mahadevan. Constructing basis functions from directed graphs for value function approximation. In *ICML '07: Proceedings of the 24th international conference on Machine learning*, pages 385–392, New York, NY, USA, 2007. ACM.

[6] Philipp W. Keller, Shie Mannor, and Doina Precup. Automatic basis function construction for approximate dynamic programming and reinforcement learning. In *ICML '06: Proceedings of the 23rd international conference on Machine learning*, pages 449–456, New York, NY, USA, 2006. ACM.

[7] Michail G. Lagoudakis and Ronald Parr. Least-squares policy iteration. *Journal of Machine Learning Research*, 4:1107–1149, 2003.

[8] Sridhar Mahadevan. Representation policy iteration. In *UAI*, pages 372–379. AUAI Press, 2005.

[9] Sridhar Mahadevan and Mauro Maggioni. Proto-value functions: A laplacian framework for learning representation and control in markov decision processes. *Journal of Machine Learning Research*, 8:2169–2231, 2007.

[10] Ishai Menache, Shie Mannor, and Nahum Shimkin. Basis function adaptation in temporal difference reinforcement learning. *Annals of Operations Research*, 134:215–238(24), February 2005.

[11] Rémi Munos. Error bounds for approximate policy iteration. In Tom Fawcett and Nina Mishra, editors, *ICML '03: Proceedings of the 20th international conference on Machine learning*, pages 560–567. AAAI Press, 2003.

[12] Ronald Parr, Christopher Painter-Wakefield, Lihong Li, and Michael Littman. Analyzing feature generation for value-function approximation. In *ICML '07: Proceedings of the 24th international conference on Machine learning*, pages 737–744, New York, NY, USA, 2007. ACM.

[13] Martin Riedmiller and Heinrich Braun. A direct adaptive method for faster backpropagation learning: the rprop algorithm. pages 586–591 vol.1, 1993.

[14] François Rivest and Doina Precup. Combining td-learning with cascade-correlation networks. In Tom Fawcett and Nina Mishra, editors, *ICML '03: Proceedings of the 20th international conference on Machine learning*, pages 632–639. AAAI Press, 2003.

[15] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998. A Bradford Book.

Contents

1 Introduction **3**

2 Least-Squares Policy Iteration **3**

3 Cascade Correlation Networks **7**

4 Using Cascade Correlation Learning Architecture in LSPI **9**

5 Experiments **11**

6 Related Work **14**

7 Conclusion **16**

Centre de recherche INRIA Lille – Nord Europe
Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq (France)

Centre de recherche INRIA Bordeaux – Sud Ouest : Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex
Centre de recherche INRIA Grenoble – Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier
Centre de recherche INRIA Nancy – Grand Est : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Rennes – Bretagne Atlantique : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex
Centre de recherche INRIA Saclay – Île-de-France : Parc Orsay Université - ZAC des Vignes : 4, rue Jacques Monod - 91893 Orsay Cedex
Centre de recherche INRIA Sophia Antipolis – Méditerranée : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399