Basis Expansion in Natural Actor Critic Methods

Sertan Girgin[1] and Philippe Preux[1,2]

[1] Team-Project SequeL, INRIA Lille Nord-Europe
[2] LIFL (UMR CNRS), Université de Lille
{sertan.girgin,philippe.preux}@inria.fr

Abstract. In reinforcement learning, the aim of the agent is to find a policy that maximizes its expected return. Policy gradient methods try to accomplish this goal by directly approximating the policy using a parametric function approximator; the expected return of the current policy is estimated and its parameters are updated by steepest ascent in the direction of the gradient of the expected return with respect to the policy parameters. In general, the policy is defined in terms of a set of basis functions that capture important features of the problem. Since the quality of the resulting policies directly depend on the set of basis functions, and defining them gets harder as the complexity of the problem increases, it is important to be able to find them automatically. In this paper, we propose a new approach which uses cascade-correlation learning architecture for automatically constructing a set of basis functions within the context of Natural Actor-Critic (NAC) algorithms. Such basis functions allow more complex policies be represented, and consequently improve the performance of the resulting policies. We also present the effectiveness of the method empirically.

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 [1]; 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. Policy gradient methods try to accomplish this goal by directly approximating the policy using a parametric function approximator. Instead of estimating the value function and then deriving a greedy policy with respect to the value function, the expected return of the current policy is estimated and its parameters are updated by steepest ascent in the direction of the gradient of the expected return with respect to the policy parameters. Policy gradient methods benefit from strong convergence properties, and especially with the emergence of methods that utilize the natural gradient

estimations that are independent of the coordinate frame chosen for expressing the policy parameters, shown to be effective. In most of these approaches, the policy is represented in terms of a set of problem dependent basis functions. The basis functions map the state(-action) variables to real numbers; each basis function performs a mapping that is different from the others, and aims at capturing the important features of the problem domain and the relations within. Consequently, the quality of the resulting policies directly depend on these functions. However, as the complexity of the problem that is being solved increases, it also becomes harder to find a "good" set of such functions. This brings up the question of what the set of best basis functions is and how we can find them.

In particular, in this paper, we will focus on the Natural Actor Critic (NAC) methods together with a linear combination of basis functions to represent a parameterized policy, and a compatible advantage value function to learn an optimal policy from a given set of experience samples. We seek to provide a possible answer to the question posed above by proposing a method that incorporates a cascade correlation learning architecture into NAC and iteratively adds new basis functions to a set of initial basis functions. In Section 2, we first introduce policy gradient and natural actor critic methods. Sections 3 describes cascade correlation learning architecture followed by the details of the proposed method for basis function expansion. Section 4 presents empirical evaluations on some benchmark problems, and a review of related work can be found in Section 5. Finally, Section 6 concludes.

2 Policy Gradient and Natural Actor Critic Methods

A *Markov decision process* (MDP) is defined by a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$ 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 when taking action a in state s. A *policy* is a probability distribution over actions conditioned on the state; $\pi_\theta(s, a)$ denotes the probability that policy π selects action a at state s. We assume that the policy is parameterized with (a small number) of parameters θ, and is differentiable with respect to these parameters, i.e. that $\partial\pi_\theta(s, a)/\partial\theta$ exists. The state-value function $V^\pi(s)$ is a mapping from states to real numbers, where the value of a state represents the expected return starting from that state, and following policy π; it is given by

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

where $\gamma \in [0, 1]$ is the discount factor, r_t is the reward received at time t, and a_t denotes the action drawn from policy π at time t. Similarly, state-action value function $Q^\pi(s, a)$ is defined as the expected return when taking action a at state s, and following policy π thereafter:

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

Our aim is to find an optimal policy π^* that maximizes the expected return

$$\rho(\pi) = E\{\sum_{t=0}^{\infty} \gamma^t r_t | s_0, \pi\} = \int_{\mathcal{S}} d^\pi(s) \int_{\mathcal{A}} \pi(s,a)\mathcal{R}(s,a) dsda$$

where $d^\pi(s) = \sum_{t=0}^{\infty} \gamma^t Pr\{s_t = s | s_0, \pi\}$ is the discounted distribution of states under π[3].

In policy gradient methods, an optimal policy is sought by an iterative process; starting from an initial policy, at each iteration the gradient $\nabla_\theta \rho(\pi_{\theta_k})$ of the expected return for the current policy π_θ is estimated, and then the policy parameters θ_k are updated in the direction of the gradient by gradient ascent

$$\theta_{k+1} = \theta_k + \alpha_k \nabla_\theta \rho(\pi_{\theta_k}) \tag{1}$$

where α_k is the learning rate. The estimation step is called *policy evaluation*, and the update step is called *policy improvement*; the entire process falls into the general framework of policy iteration [2, 3]. Policy gradient methods possess two important properties, (i) small changes in θ results in small changes in the policy and in the distribution of states under that policy, and (ii) if the gradient estimate is unbiased and learning rates are square summable but not summable (i.e. $\sum_{k=0}^{\infty} \alpha_k > 0$ and $\sum_{k=0}^{\infty} \alpha_k^2 = c$) then it is guaranteed to converge to a (locally) optimal policy. The first property brings an important advantage over value-function based methods[4] as small changes in the value estimations may lead to arbitrary changes in the associated policy, hindering convergence.

The critical problem that needs to be solved in the policy evaluation step is to obtain a "good" estimate of the gradient. A simple way to do this is to use the method of finite differences. In this approach, the policy parameters are varied by small perturbations, and then roll-outs are performed for each perturbed policy to generate estimates of the change in the expected return; regression of perturbations onto these changes yields the gradient. The major drawback of this approach is that the amount of perturbations usually differ for each parameter and they must be chosen carefully for a successful estimation. Also, it is sensitive to noise. An alternative approach is to apply likelihood ratio principle, which leads to the policy gradient theorem stating that

$$\frac{\partial \rho}{\partial \theta} = \int_{\mathcal{S}} d^\pi(s) \int_{\mathcal{A}} \frac{\partial \pi(s,a)}{\partial \theta} (Q^\pi(s,a) - b^\pi(s)) dsda \tag{2}$$

where $b^\pi(s)$ is a baseline function [4]. As there are no terms regarding the gradient of the state distribution $d^\pi(s)$ with respect to θ, the gradient of the expected

[3] When one is interested in maximizing the average reward rather than the discounted case, we have $\rho(\pi) = \lim_{n\to\infty} \frac{1}{n} E\{r_t|\pi\}$ and $d^\pi(s)$ becomes the stationary distribution of the states under π.

[4] In value-function based methods, the focus is on parameterizing and approximating the value function rather than the policy, and the policy is represented implicitly as being greedy with respect to the estimated value function.

return does not depend on changes in the distribution of states, and hence an unbiased estimate can be obtained from states sampled from the state distribution obtained by following π. If one uses the actual returns calculated over sample trajectories to approximate $Q^\pi(s,a)$, this formula reduces to the (episodic) REINFORCE algorithm of Williams [5]. The baseline $b^\pi(s)$ can be an arbitrary function of s; it does not introduce any bias but can minimize the variance of the estimate if chosen accordingly. In [4] and [6], it has been shown that the $Q^\pi(s,a) - b^\pi(s)$ term in Equation 2 can be replaced by a compatible linear function approximation $f^\pi_\omega(s,a)$ of the form

$$f^\pi_\omega(s,a) = \nabla_\theta \log \pi(s,a)^T \omega \tag{3}$$

without affecting the bias of the estimation, giving

$$\frac{\partial \rho}{\partial \theta} = \int_{\mathcal{S}} d^\pi(s) \int_{\mathcal{A}} \frac{\partial \pi(s,a)}{\partial \theta} \nabla_\theta \log \pi(s,a)^T ds da \, \omega \tag{4}$$

$$= \int_{\mathcal{S}} d^\pi(s) \int_{\mathcal{A}} \pi(s,a) \nabla_\theta \log \pi(s,a) \nabla_\theta \log \pi(s,a)^T ds da \, \omega \tag{5}$$

$$= F(\theta)\omega \tag{6}$$

as $\nabla_\theta \pi(s,a) = \pi(s,a)\nabla_\theta \log \pi(s,a)$. $F(\theta)$ is called the all-action matrix. Note that, since

$$\int_{\mathcal{A}} \pi(s,a) f^\pi_\omega(s,a) = 0 \tag{7}$$

$f^\pi_\omega(s,a)$ actually approximates the advantage value function, i.e. in fact $b^\pi(s) = V^\pi(s)$ and we have

$$f^\pi_\omega \approx A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s) \tag{8}$$

An important result regarding Equation 5 as demonstrated by Peters and Schaal is that $F(\theta)$ corresponds to the Fisher information matrix [7]; this has an interesting and useful consequence as we will see shortly. In [8], Amari showed that for a function $L(\omega)$ defined on a parameter space $S = \{\omega \in \Re^n\}$, if ω is not an orthonormal coordinate system, i.e. S is a Riemannian space, then the steepest ascent direction of $L(\omega)$ in S is not equal to the gradient of $L(\omega)$ with respect to ω, $\nabla_\omega L(\omega)$, but is given by

$$\tilde{\nabla}_\omega L(\omega) = G^{-1}(\omega) \nabla_\omega L(\omega) \tag{9}$$

where $G(\omega)$ denotes the Fisher information matrix. $\tilde{\nabla}_\omega L(\omega)$ is called the *natural gradient*. It is guaranteed that the angle between $\nabla_\omega L(\omega)$ and $\tilde{\nabla}_\omega L(\omega)$ is never larger than ninety degrees and following the natural gradient convergences to a (local) optimum. In most cases, the steepest descent with respect to the natural gradient seems more efficient than normal gradients, especially when the gradients are small and do not point directly toward the optimal solution. Normally, one needs to construct the Fisher information matrix and find its inverse in order to calculate the natural gradient. However, in case of policy gradient with

compatible function approximation, combining Equations 1, 5 and 9, Fisher information matrix and its inverse cancel each other, and one ends up with a very simple update rule for the policy parameters:

$$\theta_{k+1} = \theta_k + \alpha_k \omega \tag{10}$$

where the update terms become the weights of $f_\theta^\pi(s,a)$.

In order to take advantage of this simple formulation and convert it into an effective reinforcement learning algorithm, we need to be able to approximate $f_\theta^\pi(s,a)$ from the samples generated from a model or collected from interactions with the environment. Unlike state-value and state-action value functions, it is not possible to learn $f_\theta^\pi(s,a)$ solely using temporal difference like bootstrapping methods, as the value of states, i.e. $V^\pi(s)$, that are required for comparison are subtracted in the advantage function. In order to remedy this situation, one approach might be to have a separate approximation for the value function. Note that, by definition $Q^\pi(s,a) = f_\theta^\pi(s,a) + V^\pi(s)$, and the Bellman equation can be written as

$$f_\theta^\pi(s,a) + V^\pi(s) = \mathcal{R}(s,a) + \gamma \int_{\mathcal{S}} P(s,a,s') V^\pi(s') ds \tag{11}$$

Given a set of samples (s_t, a_t, r_t, s_{t+1}), the Natural Actor-Critic (NAC) algorithm proposed by Peters et. al uses a linear function approximation of the value function in the form of

$$V^\pi(s) = \phi(s)^T v$$

with appropriate basis functions $\phi(s)$ to construct a set of simultaneous linear equations

$$\nabla_\theta \log \pi(s_t, a_t)^T \omega + \phi(s_t)^T v \approx r_t(s,a) + \gamma \phi(s_{t+1})^T v \tag{12}$$

and solves Equation 11 using the $LSTD(\lambda)$ policy evaluation algorithm [9, 7]. $LSTD(\lambda)$ is shown to converge with probability one for function approximation, hence with appropriate basis functions NAC also succeeds in finding the true natural gradient and converges to a (local) optima in the Riemannian space. In this approach, not only the parametrization of the policy but also the basis functions chosen to represent the value function play an important role, and have a direct effect on the quality of the solution. An alternative approach, which does not require this dependency is the episodic Natural Actor-Critic (eNAC) algorithm [9, 7]. Given a trajectory of samples $(s_0, a_0, r_0, s1) \ldots (s_n, a_n, r_n, s_{n+1})$, by summing Equation 12 over the trajectory one obtains:

$$\sum_{t=0}^{n} \gamma^t A^\pi(s_t, a_t) = V^\pi(s_0) + \sum_{t=0}^{n} \gamma^t r_t - \gamma^{n+1} V^\pi(s_{n+1})$$

Ignoring the last term, and assuming a single start state s_0 for all trajectories, from a set of trajectories we can define a set of linear equations

$$\sum_{t=0}^{n} \gamma^t \nabla_\theta \log \pi(s_t, a_t)^T \omega + \rho = \sum_{t=0}^{n} \gamma^t r_t$$

with $|\omega|+1$ unknowns, which can be solved in a least squares sense to find the natural gradient.

Natural policy gradient based methods have been shown to be quite effective and perform better than regular policy gradient methods in various settings under the condition that the learning process is started with a reasonable policy [10, 11, 7]. Nonetheless, the performance of the resulting policies, at the end, depend on their structure and expression power. In practice, policies are generally represented as parametric linear mixtures of basis functions; consequently, the set of basis functions and the initial weights emerge as the determining factors. 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 and 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 a function approximator and learning architecture called *cascade-correlation networks*, and then we will present how they can be utilized in order to iteratively construct new basis functions.

3 Cascade Correlation Networks and Basis Function Construction

Cascade correlation is both an architecture and a supervised learning algorithm for artificial neural networks introduced by [12]. 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

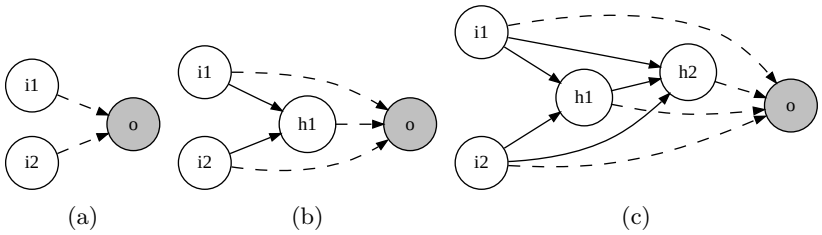

Fig. 1. (a) Initial configuration of a simple cascade-correlation network with two inputs and a single output (in gray). (b) and (c) show the change in the structure of the network as two new hidden nodes are subsequently added. Solid edges indicate input weights that stay fixed after the candidate training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

unit 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 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:

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

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 representing the policy and the corresponding compatible advantage value function in natural actor-critic methods.

As described in Section 2, in Natural Actor-Critic methods, we have a parameterized policy and a compatible approximation to the advantage value function, which is a linear combination of the partial derivatives of the logarithm of the policy with respect to the policy parameters. For the sake of simplicity, in the discussion below we will consider the case in which there is only one control variable; the extension to multiple control variables is trivial and easily follows. We will assume that the stochastic policy with policy parameters θ is defined by a set of basis functions $\boldsymbol{\varphi} = \{\varphi_1(s), \ldots, \varphi_n(s)\}$, each basis function being a function of state, and is in the form of a normal distribution with mean $\boldsymbol{\varphi}^T\theta$ and variance σ^2:

$$\pi_\theta(s, a) = \mathcal{N}(\boldsymbol{\varphi}^T\theta, \sigma^2) \tag{13}$$

For this specific case, we have

$$\nabla_\theta \log \pi(s, a) = \frac{2c}{\sigma^2}(a - \boldsymbol{\varphi}^T\theta)\boldsymbol{\varphi} \tag{14}$$

where c is a normalization constant, and the compatible advantage value function becomes

$$f_\omega^\pi(s,a) = \nabla_\theta \log \pi(s,a)^T \omega = \left[\frac{2c}{\sigma^2}(a - \boldsymbol{\varphi}^T\theta)\boldsymbol{\varphi}\right]^T \omega \tag{15}$$

Our aim here is to extend the set of basis functions $\boldsymbol{\varphi}$ by adding new basis functions, so that we would obtain better policies. In order to accomplish this goal, we need to be able to assess the effect of potential candidate basis functions on the system and choose the most promising one. One possible way to do this would be to take advantage of the inherent relationship between the policy and the compatible advantage value function. If the error in the estimation of the advantage value function can be determined, then useful basis functions such that their influence on advantage value function through the gradient of the logarithm of the policy would reduce this error, can be constructed and used to improve the policy.

Now, for a particular reinforcement learning problem, given a set of basis functions $\boldsymbol{\varphi} = \{\varphi_1(s), \ldots, \varphi_n(s)\}$ and an initial set of policy parameters θ_0, we can run episodic Natural Actor-Critic algorithm to find a sequence of natural gradient estimates and apply gradient ascent to obtain corresponding policies at each iteration. Let ω_t, θ_t, and π_{θ_t} denote the natural policy gradient, policy parameters and policy at iteration t, respectively. Note that, by definition the gradient of the logarithm of the policy for the action corresponding to the mean of the normal distribution, i.e. $\boldsymbol{\varphi}^T\theta$, in Equation 13 is the zero vector; this means that for policy π_{θ_t}, the state-action tuple $(s, \boldsymbol{\varphi}^T(s)\theta_t)$ has an advantage value of 0. Let ω be an estimation of natural policy gradient for π_{θ_t} over a set of trajectories, and $\theta = \theta_t + \epsilon\omega$ be an updated set of policy parameters, where ϵ denotes the step size. By definition, we have

$$\boldsymbol{\varphi}^T(s)\theta = \boldsymbol{\varphi}^T(s)\theta_t + \epsilon\boldsymbol{\varphi}^T\omega$$

that is the mean of the normal distribution moves by an amount of $\epsilon\boldsymbol{\varphi}^T\omega$ at state s. By putting $\boldsymbol{\varphi}^T(s)\theta$ in Equation 15, we obtain an advantage difference of

$$f_\omega^\pi(s, \boldsymbol{\varphi}^T(s)\theta) = \epsilon\boldsymbol{\varphi}^T(s)\omega\boldsymbol{\varphi}^T(s)\omega_t$$

which can be used as an estimate error in the advantage value of the state-action tuple $(s, \boldsymbol{\varphi}^T(s)\theta_t)$. Once we have the error estimates, we can then employ cascade correlation learning in order to construct the basis functions. Let $\mathcal{N}$ be a cascade correlation network with n inputs, where n is the number of initial basis functions, and a single output. We will not be using the output node, but only input and hidden nodes, as the parameters of the policy and the compatible advantage value function are determined by the eNAC algorithm. Initially, the network does not have any hidden neurons and all input neurons are directly connected to the output neuron; the activation function of the i^{th} input neuron is set φ_i, i.e. when fed with state s they output $\varphi_i(s)$. For each sample of the given set of trajectories, in the cascade correlation network we input s to the input nodes, set $\epsilon\boldsymbol{\varphi}^T(s)\omega\boldsymbol{\varphi}^T(s)\omega_t$ as the residual output error, and train candidate

units that are highly correlated with the residual output error. Note that, in Equation 15 ($a - \boldsymbol{\varphi}^T\theta$) term is common for all basis functions, and hence there is a linear dependency between the candidate units and the compatible advantage value function. 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_{n+1}; $\varphi_{m+1}(s, a)$ can be calculated by feeding s as input to the network and determining the activation value of the hidden neuron. The parameter vectors θ and ω are also expanded. By continuing eNAC training, one can calculate the natural policy gradient and update the current policy, which now is represented by one more basis functions and potentially more powerful. This process can be repeated at certain intervals, introducing a new basis function at time, until a policy with adequate performance is obtained or new functions do not improve the exiting policy. In this hybrid learning system, the basis functions are determined by the cascade correlation network, and the corresponding parameters of the policy and the advantage value function are regulated by the natural actor-critic 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.

4 Experiments

We have evaluated the proposed method on a new variant of the well known cart-pole problem, called spring cart-pole. Spring cart-pole is a dynamica system where the state is defined by the position and velocity of the elements of the system, and actions (applied forces) define the next state: it is a non-linear control task with continuous state spaces. In spring cart-pole, there are two carts, instead of one in the original version, that are connected to each other by a spring. The spring restricts the movement of carts with respect to each other, and makes the problem more complex. Both carts try to balance their own poles in upright position while staying on the track and staying close to each other. An episode ends if one of the carts move out of the boundary of track, move very close to or far away from the other cart. The eight state variables are the angles of the poles and their angular speed, and the position of the carts and their velocity. The reward is equal to sum of the cosine of the angle of the poles. Forces that can be applied to the carts are limited to [-2,2].

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: $\triangle_{min} = 0.0001, \triangle_{ini} = 0.01, \triangle_{max} = 0.5, \eta^- = 0.5, \eta^+ = 1.2$.

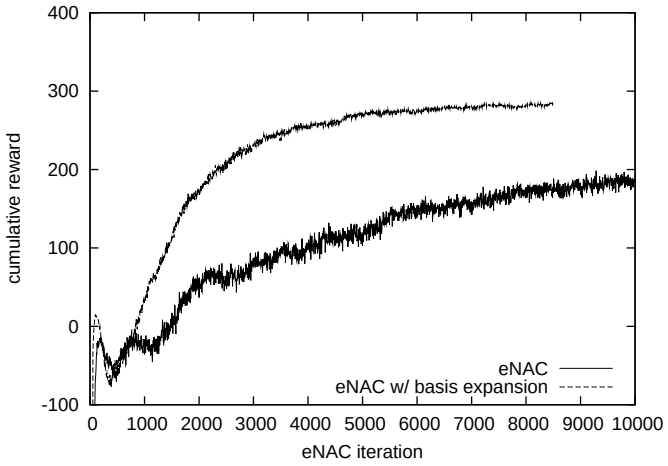

Fig. 2. eNAC using original state variables vs. eNAC with basis function expansion. A new basis function is added after every 500 iterations, up to a number of 16. After 10 basis functions, the policy converges to a near optimal policy.

Figure 2 shows the training results for the spring cart-pole problem averaged over 40 runs. The policy parameters are updated by calculating the natural policy gradient using 40 trajectories. For the simulations, we chose a rather complex initial state in which the poles are in downright position, the first cart is stationed in the middle of the track, and the second cart is half-way from it. Each trajectory starts from a slightly perturbed initial state, and is limited to a length of 200 steps. The initial policy parameters are set to 0. By employing the proposed method, a new basis function is trained every 500 policy update using the cascade correlation learning algorithm, and added to the set of basis functions of the policy. Every 5 policy update, we made a test run of 500 time steps to measure the performance of the current policy. During testing the variance of the policy distribution is set to 0, i.e. stochasticity is removed. As can be seen from the figure, policies represented by the automatically constructed basis functions outperform those that use the original basis functions that fell short of attaining near-optimal levels. Note that, the performance of the policies increase steadily, starting from as early as two additional basis functions, which indicates that the proposed method is successful in constructing useful basis functions from the existing ones.

5 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 (see [14]). Following the positive results obtained using efficient methods that rely on basis functions (mostly, using linear approximation architectures) in various domains, it also recently attracted attention from the RL community. However, the existing research is focused on constructing basis functions for approximating the value function, and, contrary to our work presented in this paper, do not consider the direct policy representation.

In [15], 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 [16], 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 [17], 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. 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 [18–20]. 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 [21]. 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.

6 Conclusion

In this paper, we explored a new method that combines cascade correlation learning architecture with episodic Natural-Actor Critic algorithm to find a set of basis function that would lead to a better and more expressive representation for the policy, and consequently results in improved 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.

We think that learning sparse representation for states in a very important issue to tackle large reinforcement learning problems. A lot of work is still due to get a proper, principled approach to achieve this, not mentioning the theoretical issues that are pending. We pursue future work in this direction and also apply the method to more complex domains.

References

1. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press, Cambridge, MA (1998) A Bradford Book.
2. Howard, R.: Dynamic programming and Markov processes. MIT Press (1960)
3. Puterman, M.: Markov Decision Processes — Discrete Stochastic Dynamic Programming. Probability and mathematical statistics. Wiley (1994)
4. Sutton, R.S., McAllester, D., Singh, S., Mansour, Y.: Policy gradient methods for reinforcement learning with function approximation. In: Neural Information Processing Systems (NIPS). (1999) 1057–1063
5. Williams, R.J.: Simple statistical gradient-following algorithms for connectionist reinforcement learning. Machine Learning **8** (1992) 229–256

6. Konda, V.R., Tsitsiklis, J.N.: On actor-critic algorithms. SIAM J. Control Optim. **42**(4) (2003) 1143–1166
7. Peters, J., Schaal, S.: Policy gradient methods for robotics. Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on (Oct. 2006) 2219–2225
8. Amari, S.i.: Natural gradient works efficiently in learning. Neural Computation **10**(2) (1998) 251–276
9. Peters, J., Schaal, S.: Natural actor-critic. Neurocomput. **71**(7-9) (2008) 1180–1190
10. Bhatnagar, S., Sutton, R., Ghavamzadeh, M., Lee, M.: Incremental natural actor-critic algorithms. In Platt, J., Koller, D., Singer, Y., Roweis, S., eds.: Advances in Neural Information Processing Systems 20. MIT Press, Cambridge, MA (2008) 105–112
11. Riedmiller, M., Peters, J., Schaal, S.: Evaluation of policy gradient methods and variants on the cart-pole benchmark. In: Approximate Dynamic Programming and Reinforcement Learning, 2007. ADPRL 2007. IEEE International Symposium on. (2007) 254–261
12. Fahlman, S.E., Lebiere, C.: The cascade-correlation learning architecture. In Touretzky, D.S., ed.: Advances in Neural Information Processing Systems. Volume 2., Denver 1989, Morgan Kaufmann, San Mateo (1990) 524–532
13. Riedmiller, M., Braun, H.: A direct adaptive method for faster backpropagation learning: the rprop algorithm. (1993) 586–591 vol.1
14. Guyon, I., Elisseff, A.: An introduction to variable and feature selection. Journal of Machine Learning Research **3** (2003) 1157–1182
15. Menache, I., Mannor, S., Shimkin, N.: Basis function adaptation in temporal difference reinforcement learning. Annals of Operations Research **134** (February 2005) 215–238(24)
16. Keller, P.W., Mannor, S., Precup, D.: Automatic basis function construction for approximate dynamic programming and reinforcement learning. In: ICML, NY, USA, ACM (2006) 449–456
17. Parr, R., Painter-Wakefield, C., Li, L., Littman, M.: Analyzing feature generation for value-function approximation. In: ICML, NY, USA, ACM (2007) 737–744
18. Mahadevan, S.: Representation policy iteration. In: UAI. (2005) 372–379
19. Johns, J., Mahadevan, S.: Constructing basis functions from directed graphs for value function approximation. In: ICML, NY, USA, ACM (2007) 385–392
20. Mahadevan, S., Maggioni, M.: Proto-value functions: A laplacian framework for learning representation and control in markov decision processes. Journal of Machine Learning Research **8** (2007) 2169–2231
21. Rivest, F., Precup, D.: Combining td-learning with cascade-correlation networks. In Fawcett, T., Mishra, N., eds.: ICML, AAAI Press (2003) 632–639