A Hierarchical Algorithm for Neural Training and Control

T. V. Theodosopoulos* and M. S. Branicky †
Laboratory for Information and Decision Systems
and
Dept. of Electrical Engineering and Computer Science
Massachusetts Institute of Technology

1 Introduction

Lately, there has been an interest in the possible uses of recurrent neural networks for nonlinear system identification and control. For instance, Matthews and Moschytz [4] suggest that just as multilayer perceptrons have been shown to be good approximators for nonlinear feedforward systems, so can recurrent neural networks for nonlinear feedback systems. In particular, they first suggested the use of the Extended Kalman Filter (EKF) algorithm for training neural networks [4]. Later Livstone, Farrell, and Baker showed how to localize the EKF algorithm to make the learning more computationally efficient [3]. The Spatially Localized Extended Kalman (SLEK) algorithm of [3] forms the basis of this paper.

We will first discuss the SLEK algorithm for training a neural network to approximate a nonlinear feedback system. We next give a dynamic programming-based method of deriving near optimal control inputs for the real plant based on this approximation and a measure of its error (covariance). Combining these methods leads to a hierarchical algorithm for identification and control of a class of uncertain, unknown systems.

2 Identification of Uncertain Nonlinear Systems

The algorithm in this paper handles the case where

$$\begin{aligned} x[k+1] &= f(x[k], u[k]) + \xi[k] \\ y[k] &= x[k] + v[k] \end{aligned}$$

where $x, \xi, v, y \in R^n$, $u \in R^m$, and $f \in C[R^{n+m}, R^n]$; f is **unknown**. Further, $\xi[k]$ and $v[k]$ are uncorrelated white Gaussian noise sequences with covariances

$$\begin{aligned} E\{\xi[k]\xi^T[n]\} &= Q[k]\delta_{kn} \\ E\{v[k]v^T[n]\} &= R[k]\delta_{kn} \\ E\{\xi[k]v^T[n]\} &= 0 \end{aligned}$$

and $\xi[k]$, $v[k]$ are uncorrelated with the initial state $x[0]$. We have the parameter vector

$$\theta = [A_{11}, A_{21}, \ldots, A_{n1}, A_{12}, A_{22}, \ldots, A_{n2}, \ldots, A_{1N}, A_{2N}, \ldots, A_{nN}]^T$$

where N is the number of neurons per dimension. For simplicity the of presentation, we will assume this number to be the same both in state and input space.

The nonlinear state equation will be approximated by the network

$$x[k+1] \approx f_{net}(q[k]; \theta[k])$$

where $\theta[k]$ are to be learned and $q[k] = [x^T[k], u^T[k]]^T$. We will refer to this R^{n+m} vector space as the *state-input space*. In this paper we consider Gaussian radial basis functions (RBFs) of the form

$$g_j(q[k]) = \exp\{-(q[k] - c_j)^T S^{-1}(q[k] - c_j)\}$$

*77 Massachusetts Ave., M.I.T., 35-415, Cambridge, MA 02139. Phone: 617-253-2832, FAX: 617-258-8553. Email: tsrsacac@athena.mit.edu
†branicky@lids.mit.edu

where the covariance matrix $S = \text{diag}(s_1^2, s_2^2, \ldots, s_{n+m}^2)$ and c_j is the jth mean or *center*.

Implicitly, we have assumed that the state-space is really $\mathcal{S} \subset R^n$ and the input space is $\mathcal{C} \subset R^m$, where $\mathcal{S}$ and $\mathcal{C}$ are compact sets surrounding the region of interest in state and input space, respectively. For many applications—*e.g.*, motor control or robotics—the state space is inherently compact (*e.g.*, angles, velocity saturation, kinematic constraints). In practice, one would use a safety net controller (usually based on that portion of dynamics that is known *a priori*) to insure that the state remains within $\mathcal{S}$.

One could now use the EKF algorithm to simultaneously approximate the state and learn the parameters (here, weights) [4]. However, this requires $O(n^3 + (nN)^3)$ operations per step (assuming no cross-correlation between the state and the parameters). Instead, we will use a localized version of the EKF to update at each step only those parameters which are significant to the model at that point. The so-called Spatially Localized Extended Kalman Filter (SLEK) algorithm of [3] implements such a strategy and can lead to an algorithm which is more computationally tractable.

In particular, we will consider the two closest centers in each dimension:

$$\theta_{\text{local}}(q) = \{A_{kj} : |q_i - c_{j,i}| < d_i\}$$

where $k = 1, \ldots, n$, $j = 1, \ldots, N$, $i = 1, \ldots, n+m$, d_i is the spacing of the centers of the RBFs in dimension i of the state-input space, and c_j is the center vector of the jth RBF.

With this notation, we have

$$f_{\text{net},i}(q[k]; \theta[k]) = \sum_{j : A_{ij} \in \theta_{\text{local}}(q[k])} A_{ij} g_j(q[k]) \tag{1}$$

Next, we construct an augmented state vector with the local parameters of θ:

$$\begin{aligned} z[k+1] &= \begin{bmatrix} x[k+1] \\ \theta_{\text{local}}[k+1](q[k]) \end{bmatrix} \\ &\approx \begin{bmatrix} f_{\text{net}}(x[k], u[k]) \\ \theta_{\text{local}}[k](q[k]) \end{bmatrix} + \begin{bmatrix} \xi[k] \\ \eta[k] \end{bmatrix} \end{aligned}$$

where

$$\begin{aligned} E\{\eta[k]\eta^T[n]\} &= \Theta[k]\delta_{kn} \\ E\{\eta[k]\xi^T[n]\} &= 0 \end{aligned}$$

Now, we linearize the above, augmented dynamics and proceed with the usual EKF algorithm [1]. The successive updates estimate the augmented state vector, part of which are the local parameters of the neural network. The covariance matrix $\Sigma[\cdot|\cdot] = \text{blockdiag}(\Sigma_1[\cdot|\cdot], \Sigma_2[\cdot|\cdot])$, where $\Sigma_i[\cdot|\cdot]$ are symmetric and have dimensions $n \times n$ and $2^n \times 2^n$, respectively.

3 Near Optimal Control with an Approximate Model

Now we proceed with the control problem. At every discrete point in time we have an estimate of where the system will go if we were to apply a specific input. This estimate is provided to us by the SLEK algorithm along with a measure of its uncertainty, the covariance ($\Sigma_1[k|k]$). Let

$$G : R^{n+m} \to R$$

be a (non-negative) cost function. For a discount factor $\alpha \in (0, 1)$ we define

$$J(x[0]) = E\left\{\sum_{i=0}^{\infty} \alpha^i G(x[i], u[i])\right\}$$

to be the penalty function for the infinite horizon problem. So we want to

$$\min_{u[0], u[1], u[2], \ldots} J(x[0])$$

with

$$\Pr(x[k+1] = y | q[k]) = \begin{cases} K(n, \delta)\, |\Sigma_1[k|k]|^{-1/2}\, \Gamma_k(y), & \|f_{\text{net}}(q[k]; \theta[k]) - y\|_\infty \le \delta \\ 0, & \text{otherwise} \end{cases}$$

$$\Gamma_k(y) = \exp\left\{-\frac{1}{2}\left[y - f_{\text{net}}(q[k];\theta[k])\right]^T (\Sigma_1[k|k])^{-1}\left[y - f_{\text{net}}(q[k];\theta[k])\right]\right\}$$

where $|A|$ is the determinant of matrix A and $K(n,\delta)$ is a constant that normalizes the truncated Gaussian pdf to integrate to unity and δ comes from discretizing the state space. In particular, if $S \subset R^n$ was the state space before, we now consider the δ-lattice that spans S, called S^δ.

We will use dynamic programming to solve the above control problem. In particular, we will choose:

$$u[k+1] = \arg\min_{u\in C}\left\{G(x[k],u) + \alpha\sum_{y\in S^\delta}\Pr(y|x[k],u)\hat{J}(y)\right\}$$

where $\hat{J}(y)$ is the estimated *cost-to-go*.

The above problem has two components: the estimation of the cost-to-go function and the solution of the corresponding minimization problem. The former is accomplished using the contraction mapping developed in [2]. Given an estimate of the cost-to-go function, we use localized gradient searches to approximate the desired input. Computationally, the estimate of the cost-to-go is calculated at an upper D. P. level; the nonlinear search is done in parallel at a number of lower-level units.

There are N^m lower level units, one for each neuron in the input space. The output of each lower level unit is an input vector $\hat{u}^*_x(x)$ corresponding to the point $x = \arg\min_{y\in S^\delta}\|y - \hat{x}[k-1|k-1]\|_\infty$. We keep track of these input vectors from the most recent time we visited every point in S^δ. In other words, for every $x \in S^\delta$ we have a set of N^m vectors $\hat{u}^*_x(x)$ that were the outputs of the N^m lower level units the last time our updated state estimate was closest to x.

The upper D. P. level updates the cost-to-go estimate for the point $x = \arg\min_{y\in S^\delta}\|y - \hat{x}[k-1|k-1]\|_\infty$. We also keep track of the most recent estimates of the cost-to-go for every point $x \in S^\delta$.

In particular, the algorithm proceeds as follows:

4 The Hierarchical Algorithm

We now describe the overall algorithm:

0. Initialize the algorithm (at $k = 0$) with some cost-to-go estimate at some $\hat{J}_0(x)$ for every point $x \in S^\delta$.

1. Given $y[k-1]$, $u[k-1]$, run SLEK at the top level, updating the network model, f_{net}, and $\Sigma_1[k-1|k-1]$. Send them to the lower level units.

2. Find $x = \arg\min_{y\in S^\delta}\|y - \hat{x}[k-1|k-1]\|_\infty$.

3. Run (for a prespecified length of time or number of steps, or until convergence) each lower-level unit for x, searching locally for the minimum input, initiating the xth unit at $\hat{u}^*_x(x)$, which is in memory, and using the existing estimates of the cost-to-go for every point in S^δ.

4. Send the new local solutions to the upper D. P. level which calculates:

$$u_{\min}(x) = \arg\min_{\hat{u}^*_x(x)}\left\{G(x,\hat{u}^*_x(x)) + \alpha\sum_{y\in S^\delta}\Pr(y|x,\hat{u}^*_x(x))\hat{J}(y)\right\}$$

5. Update the cost-to-go estimate for x:

$$\hat{J}(x) \longleftarrow G(x,u_{\min}(x)) + \alpha\sum_{y\in S^\delta}\Pr(y|x,u_{\min}(x))\hat{J}(y)$$

6. Set $u[k] = u_{\min}(x)$ and send it to the plant.

7. Increase k and Go back to Step 1.

5 Results

The simulations we have run with the above hierarchical algorithm have shown that:

- The Neural Network state converges fast to the true plant state (in the order of one thousand iterations).
- The cost-to-go estimate is difficult to converge. This is due to the fact that, for computational efficiency, we only update the cost-to-go estimate one point at a time.
- u_{min} seems to scan the whole input space for over ten thousand iterations, apparently attempting to "learn" the cost-to-go.
- Comparison of the convergence of the neural network to the true plant using random, persistently exciting inputs and the inputs chosen by our hierarchical algorithm doesn't show any significant differences.

6 Conclusions

To conclude, we have presented a hierarchical algorithm based on training recurrent networks for nonlinear control. The Extended Kalman Filter (EKF) algorithm and its localized variant (SLEK) play a central role, both in the training of the neural network and in the dynamic programming iterations to find the near optimal input at any time: The SLEK algorithm is used for training a neural network to approximate a nonlinear feedback system; we use a dynamic programming-based method of deriving near optimal control inputs for the real plant based on this approximation and a measure of its error (covariance). Finally, we combine these methods in a hierarchical algorithm for identification and control of a class of uncertain, unknown systems.

References

[1] B. D. O. Anderson and J. B. Moore. *Optimal Filtering*. Prentice-Hall, Inc., Englewood Cliffs, NJ, 1979.

[2] C-S. Chow and J. N. Tsitsiklis. Optimal multigrid algorithm for continuous-state, discrete-time stochastic control. Technical Report LIDS-P-1751, Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, February 1988.

[3] M. M. Livstone, J. A. Farrell, and W. L. Baker. Computationally efficient algorithm for training recurrent networks. In *Proc. American Control Conference*, Chicago, IL, June 1992.

[4] M. B. Matthews and G. S. Moschytz. Neural network nonlinear adaptive filtering using the extended Kalman filter algorithm. In *Proc. Int. Conf. Neural Networks*, pages 115–119, Paris, France, July 1990.