LIDS-P-2124-Revised
October 1992

A Hierarchical Algorithm for Neural Training and Control

T. V. Theodosopoulos, M. S. Branicky, and M. M. Livstone
Laboratory for Information and Decision Systems
and
Dept. of Electrical Engineering and Computer Science
Massachusetts Institute of Technology
Cambridge, MA

Accepted as a regular paper for
The Thirtieth Annual Allerton Conference
on Communication, Control, and Computing

Abstract

Lately, there has been an extensive interest in the possible uses of neural networks for nonlinear system identification and control. In this paper, we provide a framework for the simultaneous identification and control of a class of unknown, uncertain nonlinear systems. The identification portion relies on modeling the system by a neural network which is trained via a local variant of the Extended Kalman Filter. We will discuss this local algorithm for training a neural network to approximate a nonlinear feedback system. We also 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). Finally, we combine these methods in a hierarchical algorithm for identification and control of a class of uncertain, unknown systems. The complexity of the whole algorithm is analyzed.

1 Introduction

Lately, there has been an extensive interest in the possible uses of neural networks for nonlinear system identification and control [7, 3, 6]. Typically, these approaches use multilayer feedforward neural networks with various backpropagation learning rules.

Recurrent neural networks have thus far enjoyed a disproportionately low level of attention. That is mainly because the delays internal to such a model make traditional backpropagation learning ineffective. Matthews and Moschytz [5] 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, Matthews and Moschytz first suggested the use of the Extended Kalman Filter (EKF) algorithm for training neural networks [5]. Later Livstone, Farrell, and Baker showed how to localize the EKF algorithm to make the learning more computationally efficient [4]. The Spatially Localized Extended Kalman (SLEK) algorithm of [4] 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. Finally, the complexity of the whole algorithm is analyzed.

2 Identification and Control of Unknown, Uncertain Systems

2.1 Approximating the System

The algorithms in this paper are written to handle the case where

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

where $x, \xi \in R^n$, $u \in R^m$, $y, v \in R^p$ and f, f_{known}, g, h are continuous, possibly nonlinear, functions on the associated spaces. We assume f_{known}, g, and h are **known**; f is **unknown**. Further, $\xi[k]$ and $v[k]$ are uncorrelated white Gaussian noise sequences that are also uncorrelated with the initial condition $x[0]$.

However, for ease of presentation, we will treat the special case where the system dynamics are modeled as:

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

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 $x[0]$. Note that the measurement equation is linear. The nonlinear state equation will be approximated by the network

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

where $\theta[k]$ are network parameters to be learned. Let $q[k] = [x^T[k], u^T[k]]^T$. We will refer to this R^{n+m} vector space as the *state-input space*. With this notation we see

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

where the g_j are radial basis functions (RBFs) and A_{ij}, the magnitudes or *weights*, are the parameters in $\theta[k]$. In this paper, we will use Gaussian RBFs to simplify calculations:

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

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

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

Thus, θ is a vector in R^{nN}.

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 $\mathcal{S}$.

2.2 SLEK Algorithm

One could now use the EKF algorithm to simultaneously approximate the state and learn the parameters (here, weights) [5]. However, this requires $O(n^3 + (nN)^3)$ operations per step (assuming no cross-correlation between the state and the parameters).

Instead, we wish to use a localized version of the EKF to update at each step only those parameters which are significant to the model at that point. When the underlying basis functions have compact support (or die quickly) as in the case of Guassian RBFs, this strategy can lead to an algorithm which is more computationally tractable. The so-called Spatially Localized Extended Kalman Filter (SLEK) algorithm of [4] implements such a strategy.

To localize, we need to create a generalized nearest neighbor map:

$$\begin{aligned} T_\epsilon : R^{n+m} &\rightarrow \text{power set of } \{1, 2, \ldots, nN\} \\ T_\epsilon(q) &= \left\{ j : \left| \frac{\partial f_{\text{net},i}}{\partial \theta_j} \right|_q \right| > \epsilon \right\} \end{aligned}$$

That is, $T_\epsilon(q)$ is a map from a point in the state-input space to the indices of those parameters that are significant to the approximation at that point. Next, we will let $\phi_{\text{local}}(q)$ denote the

vector (matrix) obtained by ordering those elements of the vector (matrix) ϕ which are "picked off" by $T_\epsilon(q)$. For example, $\theta_{\text{local}}(q) \in R^p$ where p is the cardinality of $T_\epsilon(q)$ and $\theta_{\text{local},j}(q) = \theta_k$ if k is the jth smallest element of $T_\epsilon(q)$. Likewise, the matrix $\Theta_{\text{local}}(q)$ would be $p \times p$ with Θ_{ij} part of $\Theta_{\text{local}}(q)$ if $\{i,j\} \subset T_\epsilon(q)$ (with the obvious ordering). Finally, in a slight abuse of notation, we will say $\theta_i \in \theta_{\text{local}}(q)$ whenever $i \in T_\epsilon(q)$. Specifically, we will say $A_{ij} \in \theta_{\text{local}}(q)$ whenever $ij \in T_\epsilon(q)$.

To ease exposition and the complexity analysis, we will consider the nearest neighbor map that gives the two closest centers in each dimension. In other words, if d_i is the spacing of the centers of the RBFs in dimension i of the state-input space and if c_j is the center vector of the jth RBF,

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

where $k = 1, 2, \ldots, n$, $j = 1, 2, \ldots, N$, and $i = 1, 2, \ldots, n+m$. Note that this can be made consistent with our general $(T_\epsilon(q))$ notation by picking

$$\epsilon \leq \exp\left\{-\sum_{i=1}^{n+m} (d_i/s_i)^2\right\}$$

So now the sum in (1) is taken over j such that $A_{ij} \in \theta_{\text{local}}(q[k])$. Next, let's construct an augmented state vector with the local parameters of θ:

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

where

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

so that

$$E\{w[k]w^T[n]\} = \delta_{kn} \begin{bmatrix} Q[k] & 0 \\ 0 & \Theta_{\text{local}}[k] \end{bmatrix}$$

If we linearize the new system about the current conditions we see that

$$\delta z[k+1] \approx A[k]\delta z[k] + B[k]\delta u[k] + w[k]$$

where

$$\begin{aligned} A[k] &= \left.\frac{\partial F}{\partial z}\right|_{\hat{z}[k|k],u[k]} = \begin{bmatrix} \partial f_{\text{net}}/\partial x & \partial f_{\text{net}}/\partial \theta_{\text{local}} \\ 0 & I \end{bmatrix}_{\hat{z}[k|k],u[k]} \\ B[k] &= \left.\frac{\partial F}{\partial u}\right|_{\hat{z}[k|k],u[k]} = [\partial f_{\text{net}}/\partial u]_{\hat{z}[k|k],u[k]} \end{aligned}$$

where $\hat{z}[k|k]$ is our best current estimate of the state provided by the SLEK algorithm (see below). With some straightforward calculations, we see that

$$\left.\frac{\partial f_{\text{net},i}}{\partial q_l[k]}\right|_{q[k]} = \frac{2}{s_l^2}\left[h_{\text{net},(i,l)}(q[k]) - q_l[k] f_{\text{net},i}(q[k])\right]$$

where

$$h_{\text{net},(i,l)}(q[k]) = \sum_{j:A_{ij}\in\theta_{\text{local}}(q[k])} A_{ij}c_{j,l}g_j(q[k])$$

is a network with the same RBFs but with scaled weights. We also have:

$$\left.\frac{\partial f_{\text{net},i}}{\partial \theta_j[k]}\right|_{q[k]} = \begin{cases} 0, & \theta_j[k] \notin \theta_{\text{local}}(q[k]) \\ g_l(q[k]), & \theta_j[k] \in \theta_{\text{local}}(q[k]), l = \lceil j/n \rceil \end{cases}$$

The measurement equation is $y[k] = [I,\ 0]z[k] + v[k]$.

Now, we will use the SLEK algorithm to estimate the augmented state vector. The algorithm is as follows (see [4, 1]):

$$\begin{aligned}
\hat{z}[k|k-1] &= F(\hat{z}[k-1|k-1], u[k-1]) \\
\Sigma[k|k-1] &= A[k-1]\Sigma[k-1|k-1]A^T[k-1] + \begin{bmatrix} Q[k] & 0 \\ 0 & \Theta_{\text{local}}[k] \end{bmatrix} \\
H[k] &= \Sigma[k|k-1]C^T\left[C\Sigma[k|k-1]C^T + R[k]\right]^{-1} \\
\hat{z}[k|k] &= \hat{z}[k|k-1] + H[k]\left[y[k] - CF(\hat{z}[k|k-1], u[k])\right] \\
\Sigma[k|k] &= [I - H[k]C]\,\Sigma[k|k-1]
\end{aligned}$$

Note that the calculations above can be performed much more efficiently than the general $n + 2^n$ Kalman Filter algorithm because of the special form of C and $\Sigma[\cdot|\cdot]$: $C = [I_n,\ 0_{n,2^n}]$ and $\Sigma[\cdot|\cdot] = \text{blockdiag}(\Sigma_1[\cdot|\cdot], \Sigma_2[\cdot|\cdot])$, where $\Sigma_i[\cdot|\cdot]$ are symmetric and have dimensions $n \times n$ and $2^n \times 2^n$, respectively.

The successive updates of the SLEK algorithm attempt to estimate the augmented state vector, part of which are the local parameters of the neural network. Thus the SLEK algorithm trains the neural network by finding the weights necessary to express the dynamic nonlinearity as a local sum of Gaussians (assuming such a decomposition is possible).

2.3 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

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

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.

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

and δ comes from discretizing the state space. In particular, if $\mathcal{S} \subset R^n$ was the state space before, we now consider the δ-lattice that spans $\mathcal{S}$, called $\mathcal{S}^\delta$. Thus we substitute $\mathcal{S}$ with a grid of "fineness" δ. From the definition of the probability, we only allow nonzero probability of the two closest neighbors in each dimension of the state estimate (though, as with T_ϵ before, more general maps can be formulated). This is a convention for computational tractability.

From now on, the input space is also assumed to be discrete (for practical considerations). In particular the input space is $\mathcal{C} \subset R^m$. The discretization, $\mathcal{C}^{\delta'}$, need not have $\delta' = \delta$ because it arises from different considerations.

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

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

where $\hat{J}(y)$ is the estimated *cost-to-go*. So $u[k+1]$ is the solution of

$$\left.\frac{\partial G}{\partial u}\right|_{u[k+1]} = -\alpha \sum_{y\in\mathcal{S}^\delta} \left.\frac{\partial \Pr(y|q[k])}{\partial u}\right|_{u[k+1]} \hat{J}(y) \tag{2}$$

We now digress to discuss how to estimate the cost-to-go. Let $B(\mathcal{S}^\delta)$ be the set of bounded continuous functions on $\mathcal{S}^\delta$. Consider the following operator:

$$\begin{aligned} T : B(\mathcal{S}^\delta) &\rightarrow B(\mathcal{S}^\delta) \\ TJ(x) &= \min_{u\in\mathcal{C}} \left\{ G(x,u) + \alpha \sum_{y\in\mathcal{S}^\delta} \Pr(y|q[k])J(y) \right\} \end{aligned}$$

This operator can be proven to be a contraction [2]. So, since $B(\mathcal{S}^\delta)$ is complete, T has a unique fixed point, which is the solution we seek, *i.e.*, the real cost-to-go. So, if we start from some approximation $J_0(x)$, we can iterate:

$$J_{i+1}(x) = TJ_i(x)$$

One could even use fancier techniques to obtain tight bounds on the error at every iteration (*e.g.*, see [8]). We will not concern ourselves with the specifics of this approximation in this paper.

Going back to solving (2), we see that it is not a straightforward task. So we will iterate $\hat{u}$ to find $u[k+1]$. We will first introduce some notation:

$$\begin{aligned} c'_r &= [c_{r,n+1}, \ldots, c_{r,n+m}]^T \\ d' &= [d_{n+1}, \ldots, d_{n+m}]^T \\ B_r &= \left\{ y \in R^m \mid \|y - c'_r\|_\infty \leq \frac{1}{2} d' \right\} \end{aligned}$$

Also, define the function

$$\begin{aligned} \rho_r : R^m &\rightarrow B_r \\ \rho_r(x) &= \arg\min_{y \in B_r} \|x - y\|_\infty \end{aligned}$$

Then

$$\hat{u}_r^{(p+1)}(x) = \rho_r \left(\hat{u}_r^{(p)}(x) + \beta \left[\left. \frac{\partial G}{\partial u} \right|_{\hat{u}_r^{(p)}(x)} + \alpha \sum_{y \in \mathcal{S}^\delta} \left. \frac{\partial \Pr(y|x,u)}{\partial u} \right|_{\hat{u}_r^{(p)}(x)} \hat{J}_k(y) \right] \right)$$

with $\beta < 0$, where

$$\begin{aligned} \left. \frac{\partial \Pr(y|x,u)}{\partial u} \right|_{\hat{u}_r^{(p)}(x)} &= \Pr(y|x, \hat{u}_r^{(p)}(x)) \Delta_k(y) \\ \Delta_k(y) &= [y - f_{\text{net}}(x, \hat{u}_r^{(p)}(x); \theta[k])]^T (\Sigma_1[k|k])^{-1} \left. \frac{\partial f_{\text{net}}}{\partial u} \right|_{\hat{u}_r^{(p)}(x)} \end{aligned}$$

Above, p is the index of iteration and r shows that this gradient search is implemented in parallel N^m times, where N is the number of RBFs per input dimension. In particular, for each r we will start the corresponding search from c'_r and limit it to the $\|\cdot\|_\infty$ ball around it of radius d'.

2.4 The Hierarchical Algorithm

We now describe the overall algorithm (see Figure 1):

0. Initialize the algorithm with $k = 0$ and $u[-1] = 0$; $\hat{u}_r^{(0)}(x)$ of each D. P. r at c'_r; and the cost-to-go estimate at some $\hat{J}_0(x)$, for all $x \in \mathcal{S}^\delta$

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

2. Given f_{net}, h_{net}, and $\Sigma_1[k-1|k-1]$, run at the D. P. command level for each $x \in \mathcal{S}^\delta$

$$\hat{J}_{k-1}(x) = G(x, u_{\min}(x)) + \alpha \sum_{y \in \mathcal{S}^\delta} \Pr(y|x, u_{\min}(x)) \hat{J}_{k-2}(y)$$

3. Run (for a prespecified length of time or number of steps, or until convergence) for each $x \in \mathcal{S}^\delta$ each D. P. r starting at $\hat{u}_r^*(x)$ from the previous iteration.

4. Send the new $\hat{u}_r^*(x)$'s to the upper D. P. level which recalculates

$$u_{\min}(x) = \arg\min_{\hat{u}_r^*(x)} \left\{ G(x, \hat{u}_r^*(x)) + \alpha \sum_{y \in \mathcal{S}^\delta} \Pr(y|x, \hat{u}_r^*(x)) \hat{J}_{k-1}(y) \right\}$$

5. Set $u[k] = u_{\min}(x[k-1])$ and send to the plant.

6. Increase k and Go back to Step 1.

There are some points we wish to make about this algorithm. In practice, we suggest that the SLEK algorithm be run first, with random inputs, for a number of iterations in order for the network to begin converging. Then, the search for $u_{\min}(x)$ can be localized after a limited number of D. P. iterations. Thus, the number of required computations will decrease rapidly.

3 Computational Complexity

We will now discuss some computational complexity considerations. Let's assume we use N Gaussian RBFs in each dimension of the state-input space. Then each iteration of $\hat{u}_r^{(\cdot)}(x)$ at D. P. r takes $O(n^3 9^n)$ operations. At the upper level, each iteration of the SLEK algorithm takes $O(n^3 + 8^n)$ operations. Also, we need $O(N^m n^3 3^n)$ operations to calculate $u_{\min}(x)$.

4 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. We also provide computational complexities for the different calculations required for the algorithm.

5 Acknowledgements

During this work M. S. Branicky was supported by an Air Force Office of Scientific Research Graduate Fellowship (Contract F49620-86-C-0127/SBF861-0436, Subcontract S-789-000-056)., which is greatly appreciated.

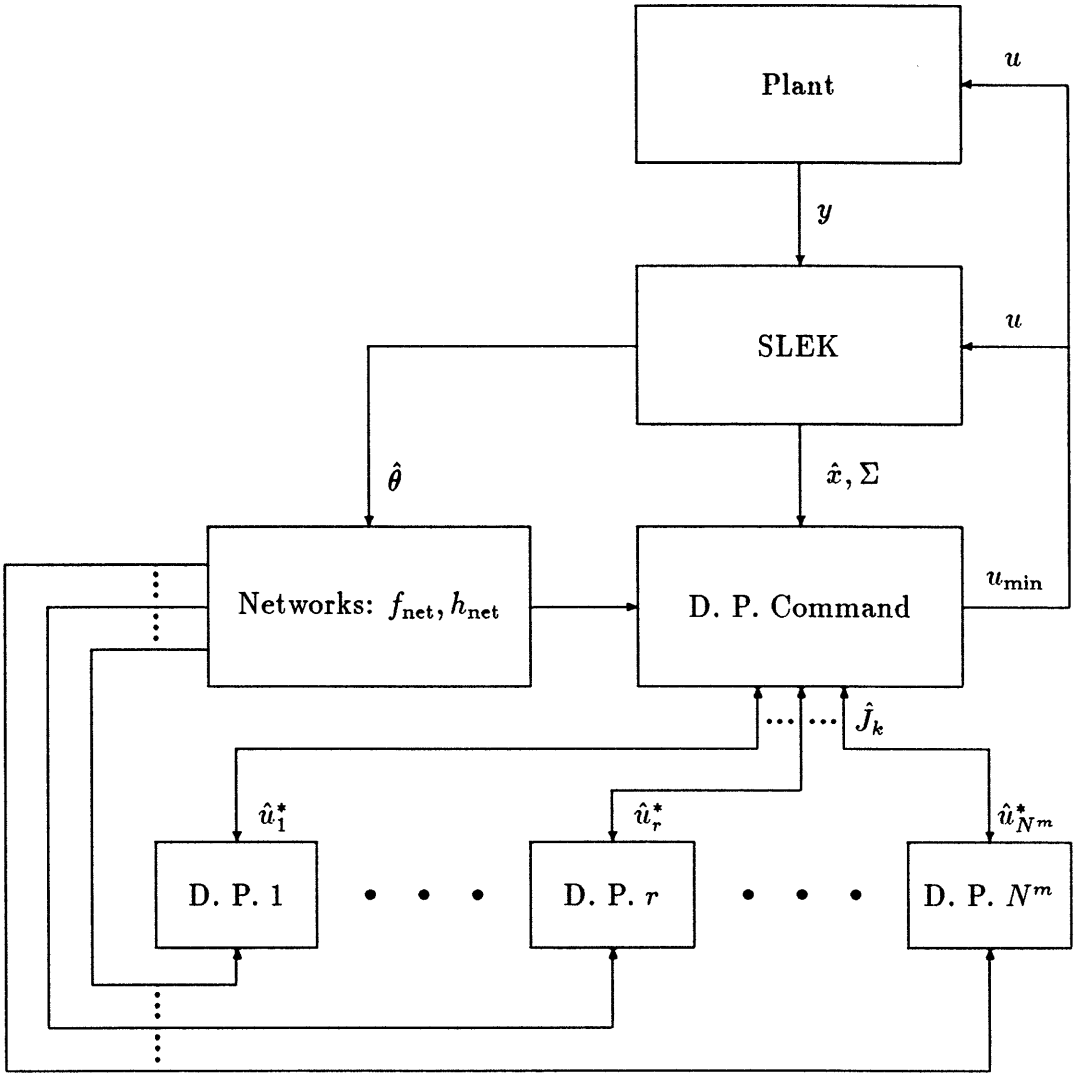

Figure 1: Hierarchical Controller

References

[1] Brian D. O. Anderson and John 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] S. R. Chu, R. Shoureshi, and M. Tenorio. Neural networks for system identification. *IEEE Control Systems Magazine*, 10(3):31–35, 1990.

[4] 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.

[5] 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.

[6] K. Narendra et al. Identification and control of dynamical systems using neural networks. *IEEE Trans. Neural Nets*, 1(1):4–26, March 1990.

[7] D. H. Nguyen and B. Widrow. Neural networks for self-learning control systems. *IEEE Control Systems Magazine*, 10(3):18–23, 1990.

[8] A. R. Odoni. On finding the maximal gain of Markov decision processes. *Operations Research*, 17:857–860, 1969.