Preventing Unlearning During On-Line Training of Feedforward Networks[1]

Scott Weaver[2,3], Leemon Baird[4], Marios Polycarpou[2]

[2]Department of Electrical and Computer Engineering
University of Cincinnati
Cincinnati, Ohio 45221-0030
Email: scott.weaver@uc.edu

[3]Wright-Patterson Air Force Base
AFRL/SNAT
2241 Avionics Circle
WPAFB, Ohio 45433-7318

[4]Computer Science Department
5000 Forbes Avenue
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213-3891

ABSTRACT

Interference in neural networks occurs when learning in one area of the input space causes unlearning in another area. These interference problems are especially prevalent in on-line applications where learning is directed by training data that is currently available rather than some optimal presentation schedule of the training data. We propose a procedure that enhances a learning algorithm by giving it the ability to make the network more local and hence, less likely to suffer from future interference. Through simulations using Radial Basis Function (RBF) networks and sigmoidal, multi-layer perceptron (MLP) networks it is shown that by optimizing a new cost function that penalizes non-locality, the approximation error is reduced more quickly than with standard back-propagation.

KEYWORDS: *interference, unlearning, spatially local networks*

[1]Partially supported under Task 2312 R1 by the United States Air Force Office of Scientific Research.

1. INTRODUCTION

An issue that arises when using neural networks is the learning problem, that is, how the network comes to sufficiently approximate some desired function. Although the learning process can become inefficient for various reasons, we focus on situations that cause unlearning of previously learned data. One such situation called passive learning [3], which occurs when the training samples cannot be preselected or reordered, often results in slower on-line learning because the incoming training data is not randomly (optimally) distributed. When using dynamical systems, the system state and training samples can remain in a local region of the state space, and hence, over a short period of time, will be unrepresentative of the entire domain being learned. These problems have been explored by Farrell [2] who demonstrates that even if the desired function is learned in a small region of the domain, which might result in a state estimation error converging to zero, the function approximator fails to learn the desired function. The longer the learning remains focused in a local region, the longer these weight changes accumulate, possibly causing other regions of the state space to unlearn. This unlearning side effect, reffered to as *interference*, occurs when learning in one area of the input space

causes unlearning in another area. Networks that are less susceptible to interference are typically referred to as *spatially local networks* [1].

Sofge and White [4] discuss characteristics of local networks and suggest that successful learning control of process dynamics requires the use of local neural network paradigms. A real-world problem, called Global Network Collapse [4], illustrates what can happen when using non-local networks. For example, consider a stable dynamical system after it settles into a desired trajectory (which traces only a small portion of the input space). In the absence of noise, a network function approximator that learns the system dynamics will stop adapting, due to a reduction in the approximation error. In the presence of noise, however, the learning algorithm remains active and continually adapts its weights (because the error never goes to zero). This attempt at learning the noise is not only unproductive locally, but more importantly may cause the learned input/output map in other areas of the state space (those areas not on the trajectory) to gradually "collapse" due to interference.

Frequently, solutions to these interference problems are sought by choosing local network architectures such as Radial Basis Function (RBF) networks. Finding a suitably-local network architecture, however, may be challenging since the amount of locality required, which may be affected by the shape of the basis functions or the complexity of the desired function, is unknown. If using an RBF, for example, one must decide how narrow the basis function widths should be to achieve sufficient locality. Also, if the required degree of locality is not uniform throughout the domain, prescribing a network with uniformly fixed locality throughout the domain, would not be acceptable. Another solution to the interference problem is to choose a network capable of becoming more local and an algorithm that adjusts the network weights in such a way that local properties emerge from the network during learning.

This paper introduces a procedure to transform an algorithm that only attempts to reduce approximation error, into an algorithm that also increases the network's locality. The resultant localizing algorithm allows the network to adapt on-line to the required degree of locality rather than requiring it to be specified before hand. Also, the localizing algorithm can be used to obtain a non-uniform degree of locality throughout the domain, allowing the network resources to focus on important areas where interference is a problem, without wasting resources by increasing locality over the entire domain. It should be noted that the ultimate goal of optimizing a network's locality is to speed learning by reducing unlearning.

To develop such a localizing algorithm we augment the standard approximation error cost function with a term that measures interference. The interference measure, developed in [5], characterizes the effect learning at one point in the domain has on another point. Reducing interference will tend to make a network less likely to suffer from future interference and hence make the network more local. Interference and approximation error are linearly combined to form a new cost function that penalizes non-locality and poor approximation. Performing gradient descent on this new cost function produces a localizing algorithm where each of the two objective functions is weighted more or less via a tuning parameter. The usefulness of the localizing algorithm is then illustrated using simulations.

The paper is organized as follows. In Section 2 we summarize definitions of interference and localization found in [5]. In Section 3, we develop the localizing algorithm by adding an interference term to the standard approximation error cost function. Section 4 provides simulations that illustrate network locality and the localizing algorithm's utility. Some concluding remarks are presented in Section 5.

2. MEASURES OF INTERFERENCE AND LOCALIZATION

In order to develop a localizing algorithm with the capability of making a network more immune to interference we begin with rigorous measures of interference and localization developed in [5] that are summarized below. Consider a network whose input/output map is $f(x,\theta)$, where $x \in \mathcal{X}$ is the input, $\theta \in \Theta$ is the weight vector, $f : \mathcal{X} \times \Theta \mapsto \mathbb{R}$ is a smooth map describing the network topology, $\mathcal{X} \subset \mathbb{R}^r$ is the input domain, and $\Theta \subset \mathbb{R}^m$ is the weight domain. During supervised learning, the objective is to adjust θ such that the network approximates a desired function $f^*(x)$. We assume the learning algorithm has the form

$$\Delta\theta = \alpha \mathbf{H}(x,\theta,e) \tag{1}$$

where α is a positive constant characterizing the step-size and $\mathbf{H}(x,\theta,e)$ is the direction for weight change, which depends on the input x, weights θ, and the approximation error $e = f(x,\theta) - f^*(x)$.

Because the weight changes in a learning algorithm affect the input/output map in various regions of the input space, any definition of interference (which measures side effects of the learning process) should incorporate the learning algorithm within the definition. To do so, consider what happens during one weight update. Given a training input/output sample $(x, f^*(x))$, the current weight θ is updated to a new weight $\theta + \alpha\mathbf{H}(x,\theta,e)$. At the point x, where the net-

work is trained, the network map changes from $f(x,\theta)$ to $f(x,\theta+\alpha\mathbf{H}(x,\theta,e))$. During this weight update the network I/O map is also affected at other points such as $x' \neq x$. We present a definition of interference followed by a motivation behind its development.

Definition 1 *Let f represent a network I/O map with weight vector θ which is updated according to a generic learning algorithm $\Delta\theta = \alpha\mathbf{H}(x,\theta,e)$. For randomized algorithms, $\alpha\mathbf{H}(x,\theta,e)$ is the expected value of the weight change. Then the interference at x' due to learning at x is defined as*

$$\mathcal{I}(x,x',\theta) = \mathcal{I}_{f,\mathbf{H}}(x,x',\theta) \stackrel{\triangle}{=} \begin{cases} \lim_{\alpha\to 0} \frac{f(x',\theta)-f(x',\theta+\alpha\mathbf{H}(x,\theta,1))}{f(x,\theta)-f(x,\theta+\alpha\mathbf{H}(x,\theta,1))} \\ \quad \textit{if limit exists} \\ 0 \\ \quad \textit{otherwise.} \end{cases} \tag{2}$$

This definition provides a measure of the degree to which training at an input point x influences the input/output function of the network at another point x'. Taking the limit of the ratio in (2) as α approaches zero gives the sensitivity of one point to an infinitesimal amount of training at another point, and normalizing e to one allows us to compute interference without specifying a desired function. The choice of e does not affect $\lim_{\alpha\to 0}$ in (2) for algorithms such as gradient descent because e can be subsumed into α, which approaches zero. If the limit does not exist, we define interference to be zero because (the attempt at) learning at x does not affect the output at x'.

For any network function approximator, f, that has a well defined gradient (with respect to θ) everywhere in $\mathcal{X}$, applying L'Hospital's rule to the ratio in (2) leads to an equivalent yet simpler form of the interference measure given by

$$\mathcal{I}(x,x',\theta) = \mathcal{I}_{f,\mathbf{H}}(x,x',\theta) \stackrel{\triangle}{=} \begin{cases} \lim_{\alpha\to 0} \frac{\nabla_\theta f(x',\theta)\cdot\mathbf{H}(x,\theta,1)}{\nabla_\theta f(x,\theta)\cdot\mathbf{H}(x,\theta,1)} \\ \quad \text{if } \nabla_\theta f(x,\theta)\cdot\mathbf{H}(x,\theta,1) \neq 0 \\ 0 \\ \quad \text{otherwise} \end{cases} \tag{3}$$

where $\nabla_\theta f(x,\theta)$ is the gradient column vector of $f(x,\theta)$ with respect to θ.

In the special case that the learning algorithm is gradient descent, equation (3) reduces to

$$\mathcal{I}(x,x',\theta) = \begin{cases} \frac{\nabla_\theta f(x,\theta)\cdot\nabla_\theta f(x',\theta)}{||\nabla_\theta f(x,\theta)||_2^2} & \text{if } \nabla_\theta f(x,\theta) \neq \mathbf{0} \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

Equation (3) provides the underlying framework for defining a measure of localization, which is done next. Specifically, interference is defined as a function of two points $x, x' \in \mathcal{X}$, while localization is defined over the entire input domain $\mathcal{X}$. A definition for network localization, given below, provides a measure of how immune a network is to interference.

Definition 2 *Let f represent a network I/O map with weight vector θ which is updated according to a generic learning algorithm $\Delta\theta = \alpha\mathbf{H}(x,\theta,e)$. For randomized algorithms, $\alpha\mathbf{H}(x,\theta,e)$ is the expected value of the weight change. Then the localization of the network over an input domain $\mathcal{X}$ is denoted by $L_\mathcal{X}(\theta)$ and is defined as*

$$L_\mathcal{X}(\theta) \stackrel{\triangle}{=} 1/\bar{\mathcal{I}}_\mathcal{X}(\theta) \tag{5}$$

where $\bar{\mathcal{I}}_\mathcal{X}(\theta) \stackrel{\triangle}{=} E[\mathcal{I}(x,x',\theta)^2]$ and $E[\cdot]$ is the expected value over all x and x' chosen from some probability density function (pdf) over the input domain $\mathcal{X}$.

If the pdf of both x and x' is uniformly distributed over $\mathcal{X}$, (5) becomes

$$L_\mathcal{X}(\theta) = \left[\int_\mathcal{X}\int_\mathcal{X} \mathcal{I}(x,x',\theta)^2 dx dx'\right]^{-1}. \tag{6}$$

In general, the localization measure $L_\mathcal{X}(\theta)$ can take any positive real value. Large values of $L_\mathcal{X}(\theta)$ indicate the network is more local over the domain $\mathcal{X}$. This definition transforms a measure of interference (between two points in the input domain) into a measure of localization (of a network over the entire input domain).

3. DEVELOPING A LOCALIZING ALGORITHM

The measure of interference introduced in the previous section is used to develop a localizing algorithm designed to reduce simultaneously both interference and approximation error. In the standard learning problem of minimizing the approximation error $e = e(x,\theta) = f(x,\theta) - f^*(x)$ for all x, a typical cost function is

$$E[J_1(x,\theta,e)] \tag{7}$$

where the expectation operator is over all $x \in \mathcal{X}$ and $J_1(x,\theta,e) = \frac{1}{2}e(x,\theta)^2$. In order to perform gradient descent on (7), an unbiased estimate of $-\alpha\nabla_\theta E[J_1]$ is given by the localizing algorithm

$$\Delta\theta(x,\theta,e) = \alpha\mathbf{H}(x,\theta,e) \tag{8}$$

which is in the form given in (1) and

$$\mathbf{H}(x,\theta,e) = -e\nabla_\theta f(x,\theta) \tag{9}$$

specifies gradient descent.

For applications where interference is a problem we reduce the cost function $E[J_2(x, x', x'', \theta, e)]$, where

$$J_2(x, x', x'', \theta, e) = \nu \frac{1}{2} e(x, \theta)^2 + (1 - \nu) \frac{1}{2} \mathcal{I}(x', x'', \theta)^2 \tag{10}$$

and the expected value is taken over all $x, x', x'' \in \mathcal{X}$. Reducing this cost function reduces the effects of interference as well the approximation error as determined by the relative weighting of the two competing goals of (10) using $\nu \in [0,\ 1]$. Using (3) we define $\mathcal{I}(x', x'', \theta)$ based upon learning algorithm (8)-(9). Throughout the rest of this paper the variable $x \in \mathcal{X}$ is the point at which training actually occurs, $x' \in \mathcal{X}$ represents a hypothetical training point, and $x'' \in \mathcal{X}$ represents the point at which one measures how much interference is present when learning at x'.

Taking the gradient of (10) produces an unbiased estimate of the gradient of $E[J_2]$ giving a localizing algorithm whose weight change is

$$\Delta\theta(x, x', x'', \theta, e) = \tag{11}$$
$$-\alpha \left[\nu e \nabla_\theta f(x, \theta) + (1 - \nu) \mathcal{I}(x', x'', \theta) \nabla_\theta \mathcal{I}(x', x'', \theta)\right].$$

The interference function is given by

$$\mathcal{I}(x', x'', \theta) = \frac{\nabla_\theta f(x'', \theta) \cdot \nabla_\theta f(x', \theta)}{\nabla_\theta f(x', \theta) \cdot \nabla_\theta f(x', \theta)} \tag{12}$$

and its gradient is

$$\nabla_\theta \mathcal{I}(x', x'', \theta) =$$
$$\frac{\nabla^2_\theta f(x', \theta) \cdot \nabla_\theta f(x'', \theta) + \nabla^2_\theta f(x'', \theta) \cdot \nabla_\theta f(x', \theta)}{\nabla_\theta f(x', \theta) \cdot \nabla_\theta f(x', \theta)}$$
$$- 2 \frac{(\nabla_\theta f(x'', \theta) \cdot \nabla_\theta f(x', \theta))(\nabla^2_\theta f(x', \theta) \cdot \nabla_\theta f(x', \theta))}{(\nabla_\theta f(x', \theta) \cdot \nabla_\theta f(x', \theta))^2}$$

where $\nabla^2_\theta f(x', \theta)$ is the Hessian matrix of $f(x', \theta)$. Each row of the matrix represents the gradient of each element of $\nabla_\theta f(x', \theta)$. Explicitly, the (i, j) element of $\nabla^2_\theta f(x', \theta)$ is $\frac{\partial}{\partial \theta_j} \frac{\partial}{\partial \theta_i} f(x', \theta)$.

Next, we consider how each term in (11) affects the learning process. To control the weighting between the two goals that compete for the network's resources, we choose ν from the range $[0,\ 1]$. When $\nu = 1.0$, (11) reduces to the standard back-propagation on the mean squared error (8) and hence does not attempt any localization of the network. In this case, if the training points are randomly distributed and independent from one another, learning may take place quickly. However, if the training points are provided by a system trajectory, for example, and are highly correlated with one another, which happens frequently in passive learning scenarios, interference may cause learning to be unacceptably slow due to the unlearning problem. It is cases like these that motivate us to use more local networks, which we can obtain by choosing ν to be less than one. This invokes use of the localizing term in the cost function. Optimization of the locality measure requires x' and x'', which unlike x, are not constrained to be chosen from a trajectory, since e does not appear in the localizing term of (11). As a result, "learning to be local" is not subject to the vagaries of passive learning and hence the network can be expected to learn to be local more quickly than it would otherwise. The main objective is an algorithm which is able to self-organize its localization properties such that leaning over the entire domain of interest occurs faster. Once the network becomes local, it can learn to approximate the function with less interference side effects that may occur due to a (possibly) poor distribution of the training data. The use of the localizing algorithm is illustrated in the following simulation section.

4. SIMULATIONS

Each simulation presented here is designed to help the reader better understand the localizing algorithm and local networks by comparing the localizing algorithm (11) and the original learning algorithm (8). The following example illustrates how interference and network locality are related in an RBF network.

Example 1 Consider the problem of learning a training set of two input/output pairs at r and s shown in Figure 1. We use a two-node Radial Basis Function (RBF) network given as

$$f(x, \theta) = \sum_{i=1}^{2} a_i e^{-((x - c_i) b_i)^2} \tag{13}$$

where the six weights a_i, b_i, and c_i, $i = \{1, 2\}$ compose θ. For Figures 1-3, the points represent the training data, the two dashed curves are the two weighted basis functions found at the output of the two nodes in the hidden layer. Their sum is the approximation rendered by the network, $f(x, \theta)$, indicated by the solid curve. Figure 1 shows the results after randomly initializing the six weights between -1.0 and 1.0, from a uniform distribution. Figure 2(a) shows results after training with standard back-propagation on the two training points. One can see that the network (solid curve) does indeed learn the two points, that is, $|f^*(r) - f(r, \theta)|$ and $|f^*(s) - f(s, \theta)|$ are minimized. Figure 2(b) shows the results after training with the localizing algorithm, with $\nu = 0.5$. The two approximation errors are minimized, as well as $|\mathcal{I}(r, s, \theta)|$ and $|\mathcal{I}(s, r, \theta)|$. Figure 2(b) illustrates how minimizing these four quantities affects the solution obtained by the localizing algorithm. Each ba-

sis of the RBF contributes significantly to the approximation at both points showing that a local network can be distributed. This network is still considered local, because training at either point does not cause unlearning at the other point.

Figure 3 shows the results of an additional phase of learning where the network is retrained at a new desired point at $x = r$ using the standard back-propagation algorithm. Figure 3(a) corresponds to retraining of the network weights from Figure 2(a), while Figure 3(b) corresponds to retraining the network weights of Figure 2(b). This training phase helps demonstrate the different levels of interference between $x' = r$ and $x'' = s$ in the two graphs of Figure 2. The results show the effect a localized network, shown in Figure 2(b), has on subsequent training in comparison to a network that is not local, as shown in Figure 2(a). In each phase of learning the learning rate is 0.1 and training is stopped when all terms being minimized are less than .0001. $\diamond$

Next, we provide a more complex example which simulates passive learning of a continuous function by causing the input to move slowly throughout its domain while performing on-line training. Again we compare standard back-propagation with the localizing algorithm. As the velocity with which the input moves throughout its domain, slows, learning with standard back-propagation also slows due to interference.

Example 2 Consider the problem of learning

$$f^*(x) = \sin(2\pi x) \tag{14}$$

in the domain $\mathcal{X} = [0,\ 1]$ using a single-input, single-output, two-hidden-layer multi-layer perceptron (MLP) network with 10 nodes in each of the two hidden layers and a bias combined into the input layer and each of the two hidden layers. Each of the network's 141 adjustable weights is initialized randomly from a uniform probability distribution in $[-2,\ 2]$.

To simulate the passive learning scenario our training data is obtained from sampling the sinusoid found in (14). On the i^{th} timestep of learning, we chose the input as

$$x_i = (1 + \sin(2\pi i/160))/2. \tag{15}$$

The pattern generated by the training points repeats every 160 timesteps. Learning in this environment is considerably more difficult as compared to the case of random inputs. For example learning to within an approximation error of 0.2 takes 165,000 timesteps as opposed to 2,500 timesteps when the values of x are chosen randomly from the domain $\mathcal{X} = [0,\ 1]$. Figure 4(a) plots the $\mathcal{L}_1$ norm of the approximation error, given as $\int_{\mathcal{X}} |e(x,\theta)|dx$, versus timestep which reaches a threshold of 0.2 considerably more quickly when $\nu = 0.97$ (solid curve) than when $\nu = 1.0$ (dashed curve). In Figure 4(b) we see the $\mathcal{L}_1$ norm of interference, given as $\int_{\mathcal{X}}\int_{\mathcal{X}} |\mathcal{I}(x', x'', \theta)|dx'dx''$, is reduced quickly when $\nu = 0.97$ (solid curve) and seems instrumental in helping to reduce the approximation error. It is interesting to note that when $\nu = 1.0$, the degree of network interference is reduced (dashed curve) even though such a reduction is not directly sought since $\nu = 1.0$. Also this reduction coincides temporally with the reduction in approximation error. This shows how adding an incentive to be local during training reduces the approximation error faster despite the extra burden on the network's "degrees of freedom" due to additional term to the cost function. $\diamond$

5. CONCLUSION

Moving directly towards a goal is sometimes not as efficient as expending the energy to circumvent an obstacle. This paper illustrates a method for circumventing interference, which is accentuated during passive learning scenarios, by borrowing from a network's approximation power (by adding a competing goal in the standard quadratic cost function). Local properties emerge from the network, in a self-organized manner, during optimization of this new cost function. This procedure mollifies the detrimental interference effects allowing the network to avoid unlearning, ultimately causing the overall approximation error to be reduced more quickly.

6. REFERENCES

[1] W. Baker and J. Farrell. An introduction to connectionist learning control systems. In D. White and D. Sofge, editors, *Handbook of Intelligent Control Neural, Fuzzy, and Adaptive Approaches*, pages 35–63, New York, NY, 1992. Van Nostrand Reinhold.

[2] J. Farrell. Approximators characteristics and their effect on training misbehavior in passive learning control. In *Proceedings of the 1996 IEEE International Symposium on Intelligent Control*, pages 181–187, 1996.

[3] J. Farrell and T. Berger. On the effects of the training sample density in passive learning control. In *Proceedings of the American Control Conference*, pages 872–876, 1995.

[4] D. Sofge and D. White. Applied learning: optimal control for manufacturing. In D. White and D. Sofge, editors, *Handbook of Intelligent Control Neural, Fuzzy, and Adaptive Approaches*, pages 259–281, New York, NY, 1992. Van Nostrand Reinhold.

[5] S. Weaver, L. Baird, and M. Polycarpou. An analytical framework for local feedforward networks. *IEEE Trans. on Neural Net.*, 9(3):473–482, 1998.

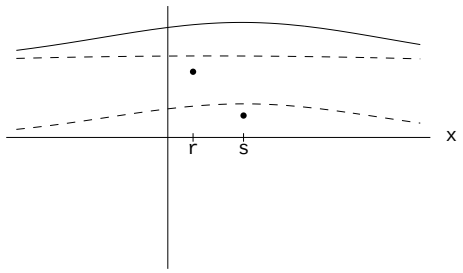

Figure 1: Initial network approximation, denoted by the solid curve, which is the sum of the two weighted hidden node outputs denoted by the dashed curves.

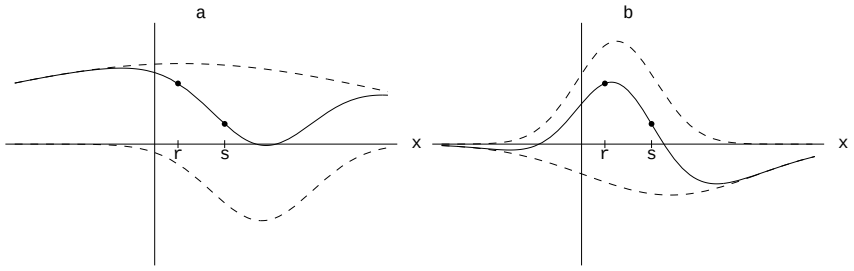

Figure 2: Network approximation after training, (a) with standard back-propagation, and (b) the localizing algorithm with $\nu = 0.5$.

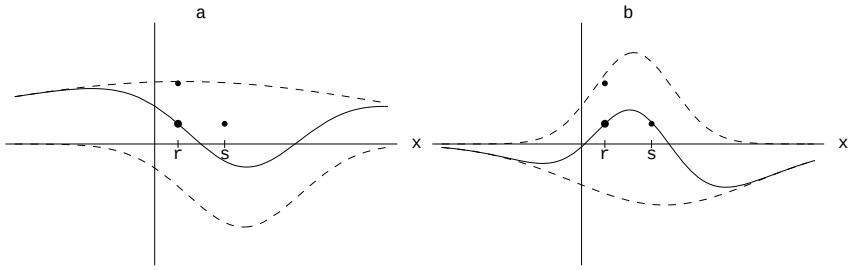

Figure 3: Network approximation after retraining exclusively on a single point at $x = r$ with standard back-propagation, starting with weights obtained from (a) Figure 2(a) and (b) Figure 2(b).

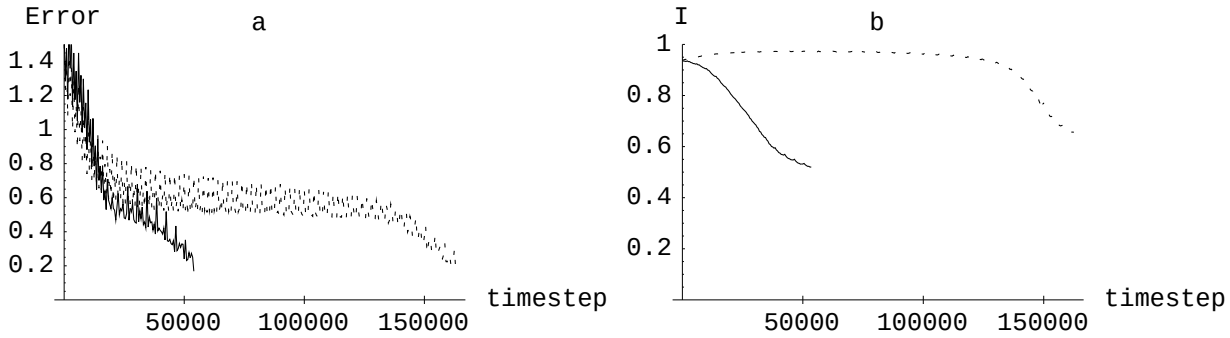

Figure 4: $\mathcal{L}_1$ norm of the approximation error (over the entire domain) is shown in (a), $\mathcal{L}_1$ norm of interference (over the entire domain) is shown in (b) with $\nu = 0.97$ (solid curve, supervised learning plus localization) and $\nu = 1.0$ (dashed curve, pure supervised learning, no localization).