Training Guidelines for Neural Networks to Estimate Stability Regions

Enrique D. Ferreira Bruce H.Krogh [1]

Department of Electrical and Computer Engineering
Carnegie Mellon University
5000 Forbes Av., Pittsburgh, PA, 15213-3890 USA
edf@cmu.edu / krogh@ece.cmu.edu

Abstract

This paper presents new results on the use of neural networks to estimate stability regions for autonomous nonlinear systems. In contrast to model-based analytical methods, this approach uses empirical data from the system to train the neural network. A method is developed to generate confidence intervals for the regions identified by the trained neural network. The neural network results are compared with estimates obtained by previously proposed methods for a standard two-dimensional example.

1 Introduction

The estimation of regions of stability for autonomous nonlinear systems has been studied for many years [3, 10]. The knowledge of regions of stability is essential to identify safe operating states in applications such as power systems and chemical reactors. In this paper we present further results on a method reported in [4] to estimate the region of stability of a stable equilibrium for an autonomous nonlinear system using a neural network. Notation is introduced and the problem is formulated in § 2. § 3 describes the neural network classifier and the main theoretical results are presented in § 4. § 5 deals with computational issues. The method is applied to a simulation example and compared with previously proposed methods in § 6.

2 Problem formulation

We consider a state-constrained nonlinear autonomous system described by the discrete-time state equations

$$\mathbf{x}_{k+1} \quad = \quad f(\mathbf{x}_k) \qquad (1)$$

with $\mathbf{x}_k \in D \subseteq \Re^n$, and $\mathbf{x} = 0$ being an asymptotically stable equilibrium point. The nonlinear function f is assumed smooth on the constraint set D. The trajectory of the system (1) with the initial state $\mathbf{x}_o$ at time step $k = 0$ is denoted by $\mathbf{x}_k(\mathbf{x}_o)$. The stability region R for $\mathbf{x} = 0$ is a connected, invariant set defined as:

$$R \quad = \quad \{\, \mathbf{x}_o : \mathbf{x}_k(\mathbf{x}_o) \in D,\ \forall k \geq 0,\ \lim_{k\to\infty} \mathbf{x}_k(\mathbf{x}_o) = 0 \,\}$$

The problem of interest is to design a classifier that determines whether a given state is in R. This classifier should be conservative, but as accurate as possible, and fast enough to be used on-line.

3 Stability region estimation

3.1 Neural network architecture

Fig. 1 shows the architecture of the proposed neural network to estimate stability regions. A multilayer feedforward network was selected for its capacity as an universal approximator [2]. Two hidden layers plus a linear input-output component is known to give robust approximations [7]. Let

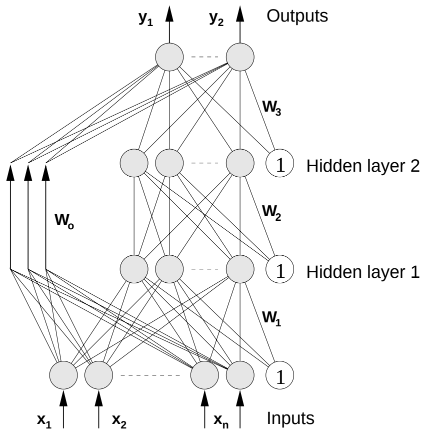

Figure 1: Multilayer neural network architecture.

N be the number of layers of units from input to output. Each layer k is composed of n_k units. We use n_o to note the number of inputs to the network. Let $\mathbf{x} \in \Re^{n_o}$ be the vector of inputs to the neural network and $\mathbf{y} \in \Re^{n_N}$ the

[1]This research has been supported by CONICYT-IBD, OAS and Carnegie-Mellon University.

output vector. The output of the k^{th} layer is given by

$$\begin{aligned}\mathbf{s}_k &= \Phi_k(\mathbf{x}) = \phi_k(W_k^T\mathbf{s}_{k-1}), \quad 1 \le k < N \\ \mathbf{y} &= \mathbf{s}_N = W_o^T\bar{\mathbf{x}} + W_N^T\mathbf{s}_{N-1}\end{aligned}$$

where $W_k \in \Re^{(n_{k-1}+1)\times n_k}$, $\mathbf{s}_k \in \Re^{n_k+1}$, and

$$\begin{aligned}\Phi_o(\mathbf{x}) &= \bar{\mathbf{x}} = [1 \quad \mathbf{x}^T]^T \\ \phi_k^T(\mathbf{x}) &= [1 \quad \phi_{k,1}(x_1)\ldots\phi_{k,n_k}(x_{n_k})].\end{aligned}$$

The activation functions $\phi_{k,i} : \Re \to S \subseteq \Re$ are bounded, monotonic, non-decreasing functions. All the units in the same layer usually have the same activation functions but they may differ with respect to other layers in the network.

Using this notation, the input-output relationship of the architecture in Fig. 1 is expressed as

$$\mathbf{y} = W_o^T\bar{\mathbf{x}} + W_3^T\Phi_2(\mathbf{x}).$$

The activation functions for the hidden units of the neural network are all chosen to be the hyperbolic tangent function. The inputs to the neural network are the components of the system state vector, **x**, and the output is a two-dimensional vector, **y**. The ideal output values are $(y_1, y_2) = (1, -1)$ when **x** is in R and $(y_1, y_2) = (-1, 1)$ when **x** is not in R. This two-variable codification of the classes follows a typical configuration which assigns as many output units as classes to separate and sets up the ideal case with a different unit active per class [2].

To make a conservative classification in the non-ideal case, two positive parameters, θ and δ, are selected to implement the following decision rules. Declare **x** belongs to R if:

$$\begin{array}{ll}1. & y_1(\mathbf{x}) - y_2(\mathbf{x}) > \theta \\ 2. & \|\nabla_{\mathbf{x}}(y_1(\mathbf{x}) - y_2(\mathbf{x}))\| < \delta\end{array} \qquad (2)$$

where $\nabla_{\mathbf{x}}$ denotes the gradient with respect to $\mathbf{x}$. The first condition is a threshold rule. The second condition discards $\mathbf{x}$ values in regions where the sensitivity of the outputs is large, as is the case near the boundary of R. The selection of θ and δ is explained in § 4.

3.2 Training procedure

Training of the neural network is based on supervised and reinforcement learning techniques. Three disjoint regions A, B and C are defined as follows, based on a priori knowledge of the autonomous system behavior.

1. *Inner region A.* A region which includes all the states from which convergence to the origin is certain. This region might be obtained from a local Lyapunov function derived from a linearized model.
2. *Study region B.* The region on which the training procedure is going to be conducted.
3. *Unsafe region C.* The bounding region in which the system is either known to be unstable or the state is outside the operating region of interest.

Data are selected from regions A and C and supervised training is applied to generate a first approximation to R. Then, experiments are performed with the system starting at states belonging to region B. For each experiment, reinforcement learning techniques are used because we do not know whether the current state belongs to R beforehand. A temporal differences method (TD_λ) [9] is applied to generate a prediction of the desired outputs for the network. The update equation for the network weights W at step k is

$$\Delta W_k = \eta(P_{k+1} - P_k).\sum_{i=1}^{k}\lambda^{k-i}\nabla_w P_i, \; k = 1, \ldots, N$$

$$P_k = \mathbf{y}_k, \quad k = 1, \ldots, N. \qquad P_{N+1} \equiv \mathbf{y}^*$$

where $\mathbf{y}_k$ is the network output at step k and $\mathbf{y}^*$ is the desired value. This algorithm is faster than waiting until the end of an experiment to get precise data to train the network [11]. At the end of each experiment, the new data points can be added to the supervised training set.

This procedure is carried out initially using off-line data, but training can continue on line as the system operates. The initial values of the weights are randomly distributed. Weight decay and early stopping with a test data set is used to prevent overtraining. A pruning algorithm is run to optimize the size of the network.

The next issue to consider is how to select the initial data points to be used in the training procedure described above. In a sense, we are trying to train the neural network to approximate the characteristic function for R. This approximation is defined on a compact set D rather than the whole state-space $\Re^n$. For this situation, Sanner and Slotine [8] showed that it is possible to uniformly approximate the function with known bounds on the number of required grid points and the error in the approximation. Their result is based on the sampling theorem and the smoothness of the nonlinear system function f in (1). As experiments continue, the network estimation of the characteristic function can be improved by selecting training patterns close to the current estimated boundary of R.

4 Estimation properties

4.1 Guidelines for grid design

Applying results on the sensitivity of discrete-time system trajectories we can use the data we can get from experiments more efficiently to guarantee a conservative estimation of R. First, we state a well known result on the sensitivity of a discrete-time system trajectory to initial conditions.

Theorem 4.1 [1] *Consider the dynamical system given by*

$$\mathbf{x}_{k+1} - \mathbf{x}_k = F(k, \mathbf{x}_k) \qquad (3)$$

with F continuous and satisfying

$$\|F(k, \mathbf{x}) - F(k, \mathbf{y})\| \le g(k, \|\mathbf{x} - \mathbf{y}\|) \qquad (4)$$

with $g : \mathcal{Z} \times \Re \to \Re$, such that g(k,r) is nondecreasing in r for all k. Let $\mathbf{x}_o^1, \mathbf{x}_o^2 \in \Re^n$ be two different state vectors such that $\|\mathbf{x}_o^1 - \mathbf{x}_o^2\| \le z_o$, then $\|\mathbf{x}_k(\mathbf{x}_o^1) - \mathbf{x}_k(\mathbf{x}_o^2)\| \le z_k$ where $z_{k+1} - z_k = g(k, z_k)$.

The system form used in Theorem 4.1 is slightly different from the form we have adopted. However, we can translate (3) to (1) by defining $f(\mathbf{x}_k) = F(k, \mathbf{x}_k) + \mathbf{x}_k$. The following proposition applies Theorem 4.1 rather directly.

Proposition 4.2 *Consider the dynamical system given by* (1) *satisfying the conditions of Theorem 4.1 with $f(\mathbf{x}_k) = F(k, \mathbf{x}_k) + \mathbf{x}_k$. Assume there exists $\epsilon > 0$ such that the ball $B(\mathbf{x}_f, \epsilon) \subset R$ for some $\mathbf{x}_f \in D$. If there exists a trajectory $\mathbf{x}_k(\mathbf{x}_o)$ of* (3) *with $\mathbf{x}_N = \mathbf{x}_f$ then the ball $B(\mathbf{x}_o, z_o) \subset R$, where z_o is computed from the equations*

$$z_{k+1} - z_k = g(k, z_k), \qquad z_N = \epsilon \tag{5}$$

If our initial state $\mathbf{x}_o$ lies in the region B and the system trajectory goes to region A as defined in § 3.2, by applying Prop. 4.2 we can compute the radius z_o of a ball around $\mathbf{x}_o$ of states which also belong to R. Therefore, z_o is an estimate for the separation between points around $\mathbf{x}_o$ in the training set leading to a non-uniform grid. To find z_o we have to solve (5) backwards in time. Another use is to estimate a local bound on the norm of the gradient of the neural network around $\mathbf{x}_o$. Since the neural network gives a continuous function, in order to reach the threshold right outside $B(\mathbf{x}_o, z_o)$ we should have

$$z_o \times \|\nabla_x(y_1 - y_2)\| \ge |(y_1 - y_2) - \theta|.$$

Similar conclusions can be applied if the experiments leads to region C. It is worth noting that it is not necessary to know the system (1) exactly. A bound on the function f is all that we need to apply the result for on-line experiments.

4.2 Confidence intervals

Since it is not possible to have noise-free data, we have to establish confidence intervals for the values obtained from the neural network and use them in our conservative decision.

We generalize the results on confidence intervals presented in [2] by introducing uncertainty in the measurements. We would like to approximate a function $h(\mathbf{x})$ using a known parametric estimation function $\phi(\mathbf{x}, W)$ that depends on a set of parameters W, and N_p input-output data values $(\mathbf{x}_i, y_i)$ perturbed by gaussian noise. We assume there exists a particular set W^* that perfectly models the unknown function $h(\mathbf{x})$. The problem can be formulated with the following expressions,

$$\begin{array}{rcl} y^* & = & h(\mathbf{x}^*) \;=\; \phi(\mathbf{x}^*, W^*) \\ y_i & = & \phi(\mathbf{x}_i^*, W^*) + e_i, \quad i = 1, \ldots, N_p \\ \mathbf{x}_i & = & \mathbf{x}_i^* + \mathbf{v}_i \\ \hat{y}_i & = & \phi(\mathbf{x}_i, \hat{W}) \end{array} \tag{6}$$

with the pair $(\mathbf{x}^*, y^*)$ representing the true input-output values and $\hat{W}$ the estimated parameters. The input noise $\mathbf{v}_i$ and output error e_i are assumed to be uncorrelated gaussian random vectors with

$$\mathbf{v}_i \sim \mathcal{N}(0, \sigma_v^2 I), \qquad e_i \sim \mathcal{N}(0, \sigma_e^2) \tag{7}$$

The output error has two components due to the modeling error between the function h and the family of functions ϕ and a measurement error. Under these assumptions we show that the posterior output distribution can be approximated by a gaussian distribution. Its variance can be used as a confidence interval.

Using (6) and (7), we can express the input-output conditional density function as

$$p(y, \mathbf{x}|\mathbf{x}^*, W) \;\sim\; \mathcal{N}(\phi(\mathbf{x}^*, W), \sigma_e^2)\, \mathcal{N}(\mathbf{x}^*, \sigma_v^2 I).$$

Assuming the input-output noisy patterns $T_i = (\mathbf{x}_i, y_i)$ are independent of each other, the conditional density function for the pattern set $T = \{T_1, \ldots\}$ is

$$p(T|X^*, W) \;=\; \prod_{i=1}^{N_p} p(y_i, \mathbf{x}_i|\mathbf{x}_i^*, W) \tag{8}$$

with $X^* = \{\mathbf{x}_1^*, \ldots\}$. Applying Bayes' formula we get the a posteriori probability distribution of the weights

$$p(W|X^*, T) \;=\; \tfrac{p(T|X^*, W)p(W|X^*)}{p(T|X^*)} \;\sim\; \exp(-S(W))$$
$$S(W) = \beta \textstyle\sum_{i=1}^{N_p} (\phi(\mathbf{x}_i^*, W) - y_i)^2 + \alpha \sum_{i=1}^{N_w} W_i^2$$

where $S(W)$ is the squared error over the target data. If the neural network is trained, $\hat{W}$ minimizes $S(W)$ and

$$\begin{array}{rcl} S(W) & \simeq & S(\hat{W}) + \dfrac{1}{2}\,(W - \hat{W})^T H_w (W - \hat{W}) \\ \phi(\mathbf{x}^*, W) & \simeq & \hat{y} + y_x^T \tilde{x} + y_w^T \tilde{W} \end{array}$$

in a neighborhood of the optimum value W^*. Using the a posteriori conditional density function for the weights we can find the output conditional density function for given input and training data:

$$\begin{array}{rcl} p(y|\mathbf{x}, T) & = & \displaystyle\int p(y, \mathbf{x}^*|\mathbf{x}, W) p(W|X^*, T)\, dW d\mathbf{x}^* \\ & \simeq & \displaystyle\int \exp\left[\frac{-1}{2\sigma_e^2}(y - \phi(\mathbf{x}^*, W))^2 \right. \\ & & \left. -\dfrac{1}{2\sigma_v^2}\tilde{\mathbf{x}}^T\tilde{\mathbf{x}} - \dfrac{1}{2}\,\tilde{W}^T H_w \tilde{W}\right] dW d\mathbf{x}^* \end{array}$$

Completing squares in W in the integrand and integrating first in the variable W we get

$$p(y|\mathbf{x}, T) \;\simeq\; \mathcal{N}(\phi(\mathbf{x}, \hat{W}), \sigma_y^2) \tag{9}$$

$$\sigma_y^2 \;=\; \sigma_w^2 + \sigma_v^2\, \nabla_x^T \phi\, \nabla_x \phi \tag{10}$$

$$\begin{array}{rcl} \sigma_w^2 & = & \sigma_e^2 + \nabla_w^T \phi\, H_w^{-1}\, \nabla_w \phi \\ H_w & = & \left.\dfrac{\partial^2 S(W)}{\partial W^2}\right|_{\hat{W}} \end{array}$$

Given the training set and a proposed threshold value θ, the probability of error in the decision rules (2) can be computed from (9). Therefore, for a given $\epsilon > 0$ we define

$$\begin{aligned}\theta &\in (\theta_1(\epsilon), \theta_2(\epsilon)) \\ \theta_1(\epsilon) &= \max_{\mathbf{x}\in T, \mathbf{x}\notin R} \theta_1(\mathbf{x}, \epsilon) \\ \theta_2(\epsilon) &= \min_{\mathbf{x}\in T, \mathbf{x}\in R} \theta_2(\mathbf{x}, \epsilon) \qquad (11) \\ \theta_1(\mathbf{x},\epsilon) &= \min_{\vartheta}\{\vartheta \mid p((y_1 - y_2) > \vartheta | \mathbf{x}, T) \le \epsilon\} \\ \theta_2(\mathbf{x},\epsilon) &= \max_{\vartheta}\{\vartheta \mid p((y_1 - y_2) < \vartheta | \mathbf{x}, T) \le \epsilon\}\end{aligned}$$

The result in (9) also shows the influence of the derivative of the output with respect to the input for the confidence interval, which confirms the usefulness of rule two. In fact, the expression of σ_y^2 in (10) can be used to set up an estimate for δ.

5 Computational issues

Computation costs in neural networks are measured in terms of the number of weights N_w in the network and the number of patterns N_p used for training. For the network architecture defined in § 3.1 we have

$$N_w = (n_o + n_2 + 1)(n_3 + n_1) + n_2.$$

The computational complexity of the supervised learning using the backpropagation algorithm to compute the derivatives is $\mathcal{O}(N_w N_p)$ [2].

The confidence intervals introduced in § 4.2 require the computation of the Hessian of the sum of the squared errors for the patterns with respect to the weights of the neural network. This computation has a complexity of $\mathcal{O}(N_w^2)$. This is not significant, however, since the computation is performed only after the network has been trained. We have used an optimized algorithm developed by Hassibi and Stork [5] and based on the outer product approximation [6] to compute the inverse of the Hessian matrix directly.

6 A two-dimensional example and comparison

In this section we apply the proposed method to the Van der Pol equation and compare the results to estimates obtained with the methods developed in [3, 10]. Since the Van der Pol equation is a continuous time system, an object-oriented tool was developed to simulate sampled-data systems and generate the neural network estimate for the stability region. Further analysis and display was done using MATLAB ™.

The state equations for this example are:

$$\begin{aligned}\dot{x}_1 &= -x_2 \\ \dot{x}_2 &= x_1 - x_2 + x_1^2 x_2\end{aligned}$$

The regions for training were taken as:

$$\begin{aligned}A &= \{(x_1, x_2) \in \Re^2 \mid \quad |x_1| \le 0.5 \text{ and } |x_2| \le 0.5\} \\ C &= \{(x_1, x_2) \in \Re^2 \mid \quad |x_1| \ge 2.5 \text{ or } |x_2| \ge 3.0\} \\ B &= A^c \cap C^c\end{aligned}$$

where the superscript c refers to the complement set. The sample time used was $T = 0.01$. The error values and parameters were averaged over five runs. Table 1 shows the results of the proposed method. Fig. 2 shows the estimation of that region according to the three methods. The network approximation is closer to R than the other methods.

The ideas developed in § 4.1 were not necessary for this small simulation example as satisfactory results were obtained with the use of a uniform grid. However, if we try to design the value of θ by applying the results of § 4.2 and use equations (11) directly, we come up with no solution for an error ϵ of five percent or less. Examining the results we note that the confidence intervals are greatly affected at some points in state space for which there is a large derivative of the output of the network with respect to the input. Therefore, the second condition in (2) was checked first and the remaining points were only tested to determine the value for θ. Furthermore, if we select

Training/Testing samples:	3721 / 961
Units in each layer:	2 - 10 - 2 - 2
Training/Test set MSE:	0.105 / 0.152

Table 1: Estimation results.

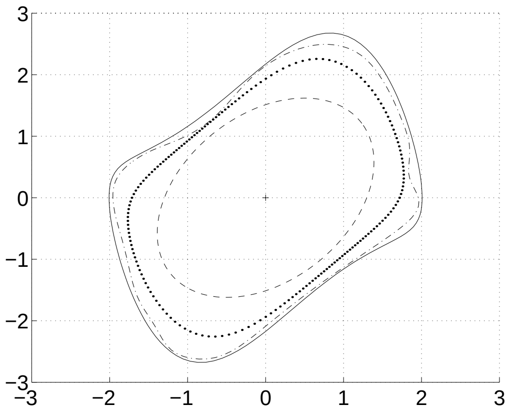

Figure 2: Van der Pol: Estimates comparison.

solid:	actual stability region.
dotted:	Vannelli and Vidyasagar method
dashed:	Davison and Kurak method
dash-dotted:	neural network estimation.

$$\epsilon = 0.01 \qquad \sigma_v = 0.05$$

$$\sigma_e^2 = \frac{1}{N_p - N_w} \sum_{i=1}^{N_p} (\mathbf{y}_i - \phi(\mathbf{x}_i, \hat{W}))^2$$

we find a good trade-off for $\theta = 1.0$ and $\delta = 4.2$. Fig. 3

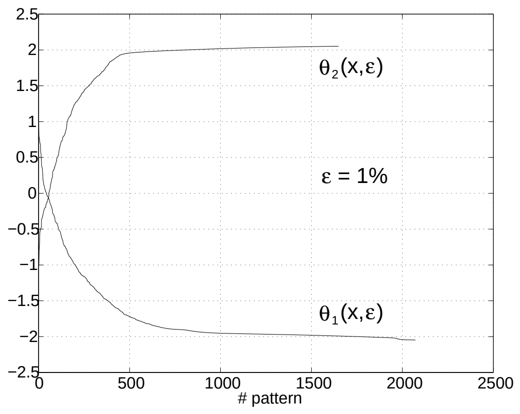

Figure 3: Confidence bounds for θ for the training set.

shows the values obtained for θ_1 and θ_2 for the training patterns. We observe that the maximum value of θ_1 is larger than the minimum of θ_2, implying a solution that correctly classifies all the patterns with a one percent accuracy is not possible. However, if we take $\theta = \theta_1$, to assure $\epsilon = 0.01$ accuracy on the classification for $\mathbf{x} \notin R$, the percentage of patterns wrongly classified as not in R is small. Finally, Fig. 4 displays the contour plots of both rules and relative to the real stability boundary. In this case the derivative con-

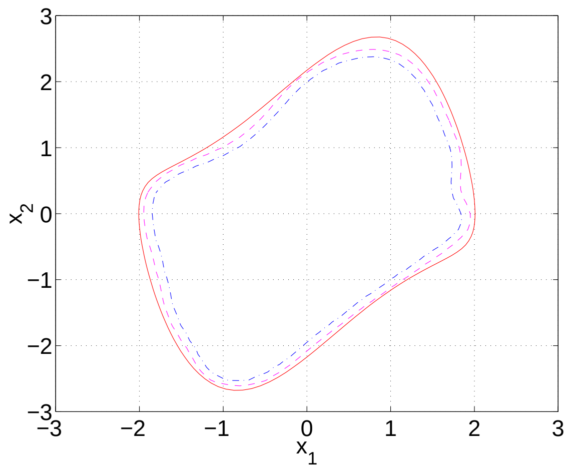

Figure 4: Confidence interval contours.

solid:	actual stability region.
dashed:	contour for rule one ($\theta = 1.0$).
dash-dotted:	contour for rule two ($\delta = 4.2$).

dition is the most conservative. Actually, the contour for the threshold condition and the real boundary lie inside the band set up by the derivative rule.

7 Discussion

A new method to estimate the stability region of an autonomous nonlinear system using neural networks is presented. Confidence intervals are computed to make a conservative classification. The model of the system is not used directly. However, any a priori knowledge should be used for training. For example, the knowledge of bounds on the dynamic equation function may lead to a better design of the grid to train the neural network. Experiments are necessary to accurately estimate the region of stability, but actual data from the system operation can be used as off-line training data for the network.

In applications, if we think of a nonlinear autonomous system as a closed-loop system with state and input constraints, a stability region estimation may allow us to make a decision to shutdown the system or change controllers if the trajectory goes out of the stable region.

References

[1] Ravi P. Agarwal. *Difference equations and inequalities : theory, methods, and applications*. M. Dekker, New York, 1992.

[2] Christopher M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, Oxford, Great Britain, 1995.

[3] E.J. Davison and E.M. Kurak. A computational method for determining quadratic lyapunov functions for nonlinear systems. *Automatica*, 7:627–636, 1971.

[4] E. Ferreira and B. Krogh. Using neural networks to estimate regions of stability. In *Proc. of 1997 American Control Conference*, volume 3, pages 1989–93, Albuquerque, NM, Jun. 1997.

[5] B. Hassibi and D.G. Stork. Second order derivatives for network pruning: optical brain surgeon. In S.J. Hanson, J.D. Cowan, and C.L. Giles, editors, *Advances in Neural Information Processing Systems*, volume 5, pages 164–71. Morgan Kaufmann, san Mateo, CA, 1993.

[6] D.W. Marquardt. An algorithm for least squares estimation of nonlinear parameters. *SIAM Journal*, 11(2):431–41, Feb 1963.

[7] P.M. Mills, A.Y. Zomaya, and M.O. Tade. *Neuro-Adaptive Process Control*. John Wiley & Sons Ltd., 1996.

[8] R.M. Sanner and J-J. E. Slotine. Gaussian networks for direct adaptive control. *IEEE Trans. on Neural Networks*, 3(6):837–863, 1992.

[9] R.S. Sutton. Learning to predict by the method of temporal differences. *Machine Learning*, 3:9–44, 1988.

[10] A. Vannelli and M. Vidyasagar. Maximal lyapunov functions and domains of attraction for autonomous nonlinear systems. *Automatica*, 21:69–80, Jan 1985.

[11] P. Werbos. A menu of designs in reinforcement learning over time. In W.T. Miller III, R.S. Sutton, and P. Werbos, editors, *Neural Networks for Control*. MIT Press, 1991.