# Time-Series Predictions Using Constrained Formulations for Neural-Network Training and Cross Validation*

*Benjamin W. Wah and Minglun Qian*
Department of Electrical and Computer Engineering
and the Coordinated Science Laboratory
University of Illinois, Urbana-Champaign
1308 West Main Street
Urbana, IL 61801, USA
E-mail: {wah, m-qian}@manip.crhc.uiuc.edu
URL: http://www.manip.crhc.uiuc.edu

## Abstract

In this paper, we formulate the training of artificial neural networks for time-series prediction as a constrained optimization problem, instead of the traditional formulation as an unconstrained optimization. A constrained formulation is better because violated constraints can provide additional guidance during a search than the traditional sum-of-squared errors in an unconstrained formulation. Using a constrained formulation, we propose to add constraints on validation errors in order to monitor the prediction quality during training of an ANN. We then solve the constrained problem using our newly developed *constrained simulated annealing algorithm* (CSA). Experimental results on the *sunspot* and *laser* time series show that constrained formulations on training and cross-validation, when solved by CSA, lead to better prediction quality and less number of weights than previous designs.

## 1 Introduction

Existing time-series predictions using *artificial neural networks* (ANNs) have generally been formulated as unconstrained optimization problems with a single objective on the sum-of-squared errors. These problems can be solved by local-search algorithms, such as back-propagation, or by global-search algorithms that bring a trajectory to new starting points when it is stuck in a local minimum. Although these algorithms work well for simple training problems, they have difficulties when the number of weights is relatively small as compared to the number of training patterns.

We have proposed to formulate ANN learning as a constrained optimization problem in which the objective is to minimize the sum-of-squared errors of all training patterns, while satisfying constraints that the error on each training pattern must be less than a prescribed threshold [7]. A constrained formulation is beneficial in difficult training scenarios because violated constraints provide additional guidance during a search, leading the trajectory towards a direction that reduces overall constraint violations.

By transforming the constrained problem into a discrete Lagrangian function, constraint satisfaction is achieved by associating dynamically changing penalties with violated constraints and by minimizing a weighted sum of constraint violations and penalties. By solving the problem using our recently developed *constrained simulated annealing algorithm* (CSA) [8], we were able to solve some difficult ANN training problems in reasonable time. For instance, we were the first to find a converged ANN for solving the two-spiral problem with four hidden units and 19 weights based on a constrained formulation [7].

In contrast, in a traditional unconstrained formulation using a single objective of minimizing the sum-of-squared errors of all training patterns, there is little guidance on search directions because errors accumulated in the objective do not pinpoint the patterns

*Proc. Int'l Conf. on Intelligent Information Processing, IFIP World Computer Congress, Aug. 2000.

that are violated. As a result, it may take longer to train a network of the same complexity. For instance, the best existing design to solve the two-spiral problem based on an unconstrained formulation has four hidden units and 25 weights [5].

In this paper we study constrained formulations in training and cross validation of fully recurrent ANNs for time-series prediction. These recurrent ANNs are single-input single-output networks, with a single hidden layer of multiple units, and the outputs of the hidden layer at time $t$ are fed back as inputs to the hidden layer at time $t+1$.

*Cross validation* entails the evaluation of errors between the actual and the predicted outputs on a validation set during training. There are two classes of cross-validation schemes. In *open-loop single-step cross validation* (or in short, *single-step validation*), the external input to the ANN is always true observed data from the validation set, and the ANN is used to predict the next output in the validation set. In contrast, in *closed-loop iterative cross validation* (or in short, *iterative validation*), the external input to the ANN is the predicted output obtained in the last iteration. Since the input in iterative validation is predicted, larger prediction errors are generally expected. The final network selected is one that minimizes errors in either single-step or iterative validation.

One of the major issues in cross validation is to identify the part of training patterns to reserve for cross validation, as a pattern cannot be used for both training and cross validation. Traditional approaches usually divide the set of training patterns into two disjoint sets and use the most recent set for cross validation because the recent set generally reflects better the behavior of the patterns. An important reason for using a single validation set is that the training problem is formulated as a single-objective unconstrained optimization problem and cannot handle errors from multiple validation sets. In general, the single validation set selected must be kept small, since patterns for training and validation are disjoint and it is important to use as many patterns as possible for training. However, validations using only one set of patterns are restricted because the single set cannot cover multiple regimes in multi-stationary time series.

We plan to address in this paper the two limitations on traditional cross-validation schemes for time-series prediction, namely, the *a priori* reservation of a fraction of training patterns for cross validation and the use of only one validation set. Our proposed approach defines multiple validation sets in training and includes errors from each set in constraints in a constrained formulation. The approach is illustrated in Figure 1 with three validation sets, two of which are overlapped. Our approach also uses the entire training set in training, without sacrificing any training patterns for cross validation. The test set in Figure 1 is used to test the performance of the ANN after training is completed.

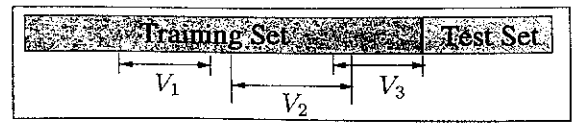



Figure 1: Multiple validation sets in a training set. $V_1$,$V_2$ and $V_3$ are three validation sets.

This paper is divided into five sections. We present in Section 2 our constrained formulation and in Section 3 algorithms for training and cross validation. Section 4 summarizes our experimental results on two time-series benchmarks, using in each case a fully-recurrent one-hidden-layer ANN with one external input and one output. Finally, conclusions are drawn in Section 5.

## 2 Constrained Formulations

Traditional ANN training for time-series prediction is formulated as an unconstrained optimization:

$$\min_{w} \quad E(w) = \sum_{t=1}^{n} (o_t(w) - d_t)^2, \tag{1}$$

where $o_t$ and $d_t$ are, respectively, the actual and desired outputs of the network at time $t$, $w$ is a vector of weights of the ANN trained, and the training data consist of patterns observed at $t = 1, \cdots, n$. There are many algorithms that can be used to solve (1). An example is the *epoch-wise back-propagation through time* (EWBPTT) [13].

We have formulated ANN training as a constrained optimization problem in which the objective is to minimize the sum-of-squared errors, while satisfying constraints that the error on each training pattern must be less than a prescribed threshold [7]. A similar constrained formulation on ANN training for time-series prediction is:

$$\begin{aligned} \min_w \quad & E(w) = \sum_{t=1}^{n} max^2\{|o_t(w) - d_t| - \tau, 0\} \\ \text{s.t.} \quad & h_t(w) = |o_t(w) - d_t| \leq \tau, \end{aligned} \tag{2}$$

where $t = 1, \cdots, n$, $h_t(w)$ is the constraint on the pattern observed at $t$, and $\tau$ is a small positive number that decreases towards 0 as looser constraints are satisfied.

It is easy to see that for $\tau = 0$, (1) and (2) are equivalent when training converges. There are two reasons for using $\tau$ larger than zero in training. First, in difficult training scenarios in which it may not be possible to reach convergence with $\tau = 0$, a non-zero $\tau$ may allow training to converge faster and provide a starting point for further training using a smaller $\tau$. Based on successive refinements of $\tau$, convergence can be achieved for small $\tau$ in reasonable time. Second, using a non-zero $\tau$ across all patterns allows training to be done to within a uniform error tolerance in case that smaller tolerances cannot be achieved. Our experience shows that ANNs with uneven errors in their training patterns usually perform poorly, especially in multi-stationary time-series.

A constrained formulation simplifies the inclusion of errors from multiple validation sets. We first define the error from the $i^{th}$, $i = 1, \cdots, v$, validation set as the *normalized mean squared error*:

$$\varepsilon_i = \frac{1}{\sigma_i^2 N_i} \sum_{t=t_{i,0}}^{t_{i,1}} (y(t) - d(t))^2, \tag{3}$$

where $\sigma_i^2$ is the variance of the true time series in $[t_{i,0}, t_{i,1}]$, $y(t)$ is the actual output, $d(t)$ is the desired output, and $N_i$ is number of patterns in the $i^{th}$ set.

The error from each validation set can now be incorporated easily as a constraint in (2):

$$\begin{array}{ll} \min_w & E(w) = \sum_{t=1}^{n} max^2\{|o_t(w) - d_t| - \tau, 0\} \\ \text{s.t.} & h_t(w) = |o_t(w) - d_t| \leq \tau \\ & h_i^I(w) = \varepsilon_i^I \leq \tau_i^I \\ & h_i^S(w) = \varepsilon_i^S \leq \tau_i^S, \end{array} \tag{4}$$

where $\varepsilon_i^I$ (resp. $\varepsilon_i^S$) is the normalized mean squared error of the iterative (resp. single-step) validation error on the $i^{th}$ validation set, and $\tau_i^I$ and $\tau_i^S$ are small positive constants. In a similar way, $\tau_i^I$ and $\tau_i^S$ can be refined successively as training progresses.

# 3 Constrained Simulated Annealing

In this section, we summarize our discrete Lagrange-multiplier theory [9], adaptation of *simulated annealing* (SA) to look for discrete saddle points [8], and application of the algorithm to train ANNs.

Consider a discrete equality-constrained *nonlinear programming problem* (NLP):

$$\begin{array}{ll} \text{minimize}_x & f(x) \\ \text{subject to} & h(x) = 0, \end{array} \tag{5}$$

where $x = (x_1, \ldots, x_n)$ is a vector of discrete variables, and $f(x)$ and $h(x)$ are nonlinear, and either analytic in closed forms or procedural.

To characterize the solutions sought in discrete space, we define for discrete problems, $\mathcal{N}(x)$, the *neighborhood* of point $x$ in discrete space $X$, as a user-defined set of points $x' \in X$ such that $x' \in \mathcal{N}(x) \Longleftrightarrow x \in \mathcal{N}(x')$ and that it is possible to reach every other $x''$ starting from any $x$ in one or more steps through neighboring points.

Point $x \in X$ is called a *constrained local minimum* (*CLM*) iff a) $x$ is a feasible point, and b) for every feasible point $x' \in \mathcal{N}(x)$, $f(x') \geq f(x)$.

Point $x \in X$ is called a *constrained global minimum* (CGM) iff a) $x$ is a feasible point, and b) for every feasible point $x' \in X$, $f(x') \geq f(x)$. The set of all CGM is $X_{opt}$.

A *generalized discrete Lagrangian function* of (5) is defined as follows:

$$L_d(x, \lambda) = f(x) + \lambda^T H\left(h(x)\right), \tag{6}$$

where $H$ is a continuous transformation function satisfying $H(y) = 0$ iff $y = 0$ and $H(y) \geq 0$.

We define a *discrete saddle point* $(x^*, \lambda^*)$ in discrete space with the following property:

$$L_d(x^*, \lambda) \leq L_d(x^*, \lambda^*) \leq L_d(x, \lambda^*) \tag{7}$$

for all $x \in \mathcal{N}(x^*)$ and all $\lambda \in R$. The concept of saddle points is very important in discrete problems because, starting from them, we can derive first-order necessary and sufficient conditions for CLM that lead to global minimization procedures. This is stated formally in the following theorem [9]:

**Theorem 1.** *First-order necessary and sufficient conditions for CLM.* A point in the variable space of (5) is a CLM if and only if it satisfies the saddle-point condition (7)

An inequality constraint like $g_j(x) \leq 0$ can be transformed into an equivalent equality constraint $\max(g_j(x), 0) = 0$.

Theorem 1 implies that any algorithm for finding a CGM of (5) can be reduced to finding a discrete saddle point with the minimum objective value. There are many algorithms that can achieve this goal. Figure 2 describes CSA [8], an adaptation of SA to find discrete saddle points. CSA carries out *probabilistic ascents* in the Lagrange-multiplier space $\Lambda$ and *probabilistic descents* in the original-variable space $X$, with probabilities of acceptance governed by a temperature. It has been proven to have asymptotic convergence to a CGM with probability one [8], using

```
1. procedure CSA
2.     set initial x = (x, λ) by randomly generating
           x and by setting λ ← 0;
3.     initialize starting temperature T to be large
           enough and the cooling rate 0 < α < 1
4.     set N_T (number of trials per temperature);
5.     while stopping condition is not satisfied do
6.         for n ← 1 to N_T do
7.             generate x' from N(x) using G(x, x');
8.             accept x' with probability A_T(x, x')
9.         end_for
10.        set T ← α × T;
11.    end_while
12. end_procedure
```

Figure 2: CSA: Constrained simulated annealing.

distribution $G(\mathbf{x}, \mathbf{x}')$ to generate a trial point $\mathbf{x}'$ in a user-defined neighborhood $\mathcal{N}(\mathbf{x})$, a Metropolis acceptance probability $A_T(\mathbf{x}, \mathbf{x}')$, and a geometric cooling schedule. Due to space limitations, we do not describe the details here.

In addition to finding CGM for discrete constrained NLPs, CSA can be applied to solve continuous and mixed-integer constrained NLPs because numerical evaluations of continuous variables using computers can be considered as discrete approximations of the original variables up to a computer's precision. Under certain general assumptions on the objective and constraint functions [14], we have proved upper bounds on the error between the CGM found in discretized space and the CGM in continuous space, independent of the search algorithm used. Hence, the CGM found by CSA in discretized space cannot be improved by any other numerical method limited by the precision of computers. We can, therefore, apply CSA to solve continuous NLP (4) represented as discrete Lagrangian function (8).

We can now apply the theory of discrete Lagrange multipliers to solve the continuous constrained NLP that models ANN learning for time-series prediction. Based on (6), the discrete Lagrangian function of the training and cross-validation problem in (4) is:

$$\mathcal{L}(w, \lambda) = E(w) + \sum_{i=1}^{n} \lambda_i max\{0, h_i\} + \sum_{j=1}^{v} (\lambda_j^I max\{0, h_j^I\} + \lambda_j^S max\{0, h_j^S\}) \quad (8)$$

where $\lambda_i$, $\lambda_j^I$ and $\lambda_j^S$ are Lagrangian multipliers.

A sampling procedure like CSA may be too expensive to apply in training large networks with a large number of patterns. In our implementation, we apply EWBPTT to generate a new point $(w + \delta w)$ by running one pass of the feedforward process through all training data, recording errors $e_t = o_t - d_t$ for all $t$, setting $e_t = \lambda_t e_t$, and evaluating EWBPTT to obtain $\delta w$. After computing $\mathcal{L}(w + \delta w, \lambda)$, CSA then decides whether to accept $(w + \delta w)$ using the Metropolis probability. To preserve the ergodicity property of CSA, we occasionally add noise to the direction generated by EWBPTT.

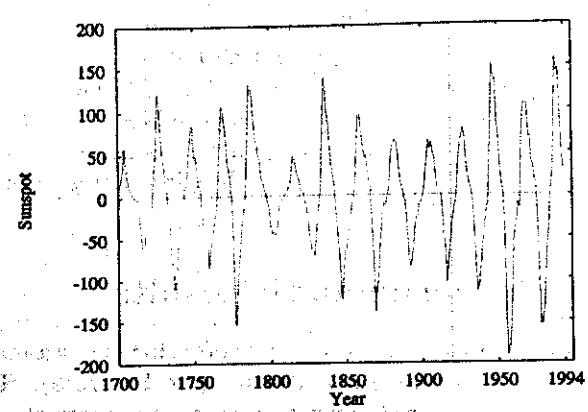
# 4 Experimental Results

We evaluate our proposed method using two standard benchmarks: *sunspot* and *laser* (see Figure 3). The *sunspot* time series consists of yearly numbers of sunspots collected from 1700 to 1994, whereas the *laser* time series is a set of chaotic intensity pulsation of an $NH_3$ laser. In our experiments, we used data from 1700 to 1920 in *sunspot* as training data and tested the ANN trained using data from 1921 to 1994 in a single-step fashion. In *laser*, we used the first 1000 patterns for training and tested the ANN trained using the next 100 patterns in both single-step and iterative fashion. Before training, we normalized all data to have 0 mean and to be within $[-1, 1]$ for *sunspot* and $[-3, 3]$ for *laser*.

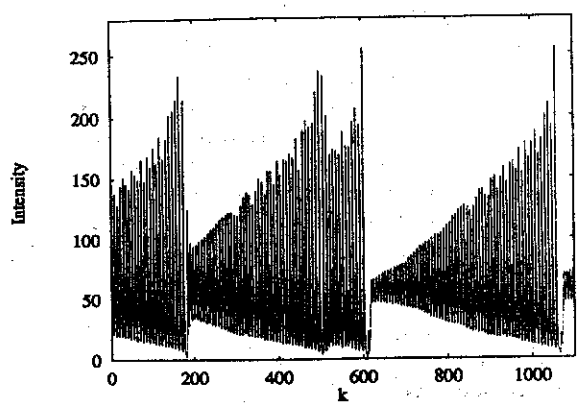
We used the $tanh(\alpha x)$ hyperbolic activation function in the hidden units and a linear activation function in the output units. We have tried to design ANNs that are as small as possible for each time series. After experimentation, we used 2-hidden-unit ANNs with 11 weights for *sunspot* and 20-hidden-unit ANNs with 461 weights for *laser*. We evaluated the performance in both validation and testing using (3).

To compare the performance of different formulations and cross-validation methods, we trained an ANN for an application using a fixed amount of time by each of the following five methods. Due to the exhaustive nature of evaluations and the need to collect statistically meaningful results, we have chosen *sunspot* that takes relatively short training time.

1. SA: Unconstrained formulation trained by SA (using the SIMANN [3] package with its sample generation replaced by EWBPTT) without cross-validation,
2. SA&V1: Unconstrained formulation trained by SIMANN in the previous part and with traditional cross-validation, using *sunspot* patterns from 1901 to 1920 as the validation set,
3. CSA: Constrained formulation trained by CSA without cross-validation,
4. CSA&V1: Constrained formulation trained by CSA with traditional cross-validation, using




a) Sunspot time series




b) Laser intensity time series

Figure 3: *Sunspot* and *laser* time series. Dashed vertical lines divide the training and testing sets.

Table 1: Average test performance by different formulations and cross-validation methods on *sunspot*. (Each entry in the second thru fifth columns represents a mean value with 95% confidence and ±10% of the value indicated. The last column shows the number of runs needed to achieve the required level of confidence. Bold numbers indicate the best results. Each run took approximately 24 to 26 seconds on a 450-MHz Pentium-III computer under Solaris 7. The ANN used has one input, two hidden units, one output, and 11 weights.)

| Method | 1921-1955 | 1956-1979 | 1980-1994 | 1921-1994 | Runs |
|---|---|---|---|---|---|
| SA | 0.052306 | 0.113958 | 0.055074 | 0.076735 | 28 |
| SA&V1 | 0.081943 | 0.138510 | 0.086798 | 0.102199 | 76 |
| CSA | 0.035385 | 0.061361 | 0.039559 | 0.045554 | 10 |
| CSA&V1 | 0.042079 | 0.086607 | 0.051244 | 0.060784 | 18 |
| CSA&V2 | **0.034288** | **0.053549** | **0.034236** | **0.040634** | 4 |

*sunspot* patterns from 1901 to 1920 as the validation set, and

5. CSA&V2: Constrained formulation trained by CSA with proposed cross-validation, using *sunspot* patterns from 1881 to 1920 as the validation set.

Table 1 compares the average test performance of the five methods. The much better results of CSA over SA on all four test durations (without cross validation) demonstrate the merits of using constraints in the problem formulation. The table also shows that traditional cross validation does not work well in both unconstrained and constrained formulations. This may be attributed to the facts that the test set (20 patterns) takes up patterns in the already small training set (220 patterns), and that the choice of the test set may not be the best. The last column of Table 1 illustrates the rather erratic training behavior of SA and SA&V1 in which each method needs a large number of runs in order to reach a 95% confidence of the mean behavior within 10% of the values presented. Finally, the table shows that CSA&V2 has the best test performance among the five methods over the four test durations, and that it requires the least number of runs to reach a given confidence interval. The stability of CSA&V2 has also been observed in 100 runs of the algorithm.

Tables 2 and 3 compare the test performance of previous approaches and that of CSA&V2. Note that two validation sets are used in training *laser*: one containing patterns 591 to 640 and another containing patterns 901 to 1000. These results show that CSA&V2 leads to smaller networks and achieves better short-term as well as long-term single-step and iterative predictions.

Figure 4 shows that CSA&V2 is very accurate in single-step predictions on *sunspot*. Figure 5a shows that CSA&V2 is very accurate in single-step predictions on *laser* before the corruption point, drift away from the actual data at the corruption point, recover quickly, and retain good accuracy afterwards. Finally, Figure 5b shows that CSA&V2 is very accurate in iterative predictions before the corruption point, predicts successfully the corruption point, and predicts with some phase shift afterwards.

Table 2: Single-step test performance in $nMSE$ on *sunspot*. (The test set consists of patterns from 1921 to 1994. As a comparison, we also compute $nMSE$ on single-step prediction for three segments in the test set: 1921-1955, 1956-1979 and 1980-1994. Bold numbers indicate the best results. N/A stands for data not available. The second column shows the number of weights/free variables in each method.)

| Method | No. of Free Variables | Training | Single-Step Testing | | | |
|---|---|---|---|---|---|---|
| | | 1700-1920 | 1921-55 | 1956-79 | 1980-94 | 1921-94 |
| AR(12): $12^{th}$-order linear auto-regression [10] | 14 | 0.128 | 0.126 | 0.36 | 0.306 | 0.238 |
| TAR: Threshold auto-regressive model [6] | 18 | 0.097 | 0.097 | 0.28 | 0.306 | 0.197 |
| WNet: Feedforward ANN with weight elimination [12] | 113 | 0.082 | 0.086 | 0.35 | 0.313 | 0.219 |
| SSNet: Soft-weight-sharing network [4] | N/A | - | 0.077 | N/A | N/A | N/A |
| DRNN: Dynamic recurrent neural network [1] | 30 | 0.105 | 0.091 | 0.273 | N/A | N/A |
| COMM: Committee prediction using FIR network [10] | N/A | 0.079 | 0.065 | 0.24 | 0.188 | 0.148 |
| ScaleNet: Multiscale ANN [2] | N/A | 0.086 | 0.057 | 0.13 | N/A | N/A |
| Proposed CSA&V2 | 11 | 0.0559 | **0.0337** | **0.0524** | **0.0332** | **0.0397** |

Table 3: Single-step and iterative test performance in $nMSE$ on *laser*. (The test set consists of patterns from 1001 to 1100. As a comparison, we also compute $nMSE$ on single-step as well as iterative predictions for the segment 1001-1050 in the test set. Bold numbers indicate the best results; N/A stands for data not available.)

| Method | Number of weights | Training | Single Step Prediction | | Iterative Prediction | |
|---|---|---|---|---|---|---|
| | | 100-1000 | 1001-1050 | 1001-1100 | 1001-1050 | 1001-1100 |
| FIR network [11] | 1105 | 0.00044 | 0.00061 | 0.023 | 0.0032 | 0.0434 |
| ScaleNet: Multiscale ANN [2] | N/A | 0.00074 | 0.00437 | 0.0035 | N/A | N/A |
| Proposed CSA&V2 (Run 1) | 461 | 0.00036 | **0.00043** | **0.0034** | 0.0054 | **0.0194** |
| Proposed CSA&V2 (Run 2) | 461 | 0.00107 | **0.00030** | **0.00276** | **0.0030** | **0.0294** |

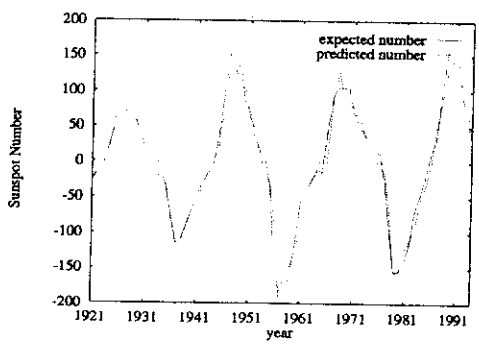


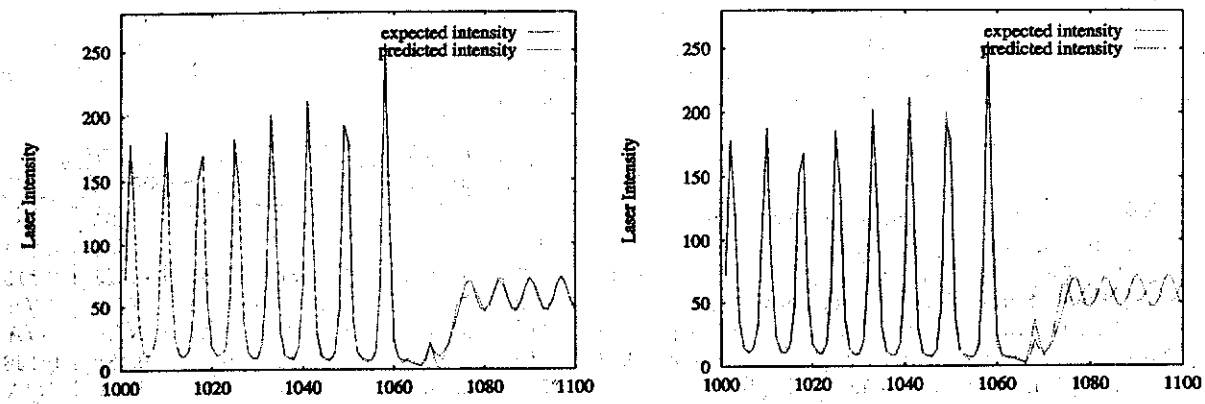

Figure 4: Single-step prediction performance of an ANN trained by CSA&V2 for *sunspot* time-series.

# 5 Conclusions

In this paper, we have presented a new constrained formulation for time-series prediction and have incorporated cross validation as new constraints. The use of constraints provide additional guidance during a search that allows difficult training scenarios to be completed in shorter time. Moreover, the use of constraints allows multiple validation sets to be defined during training and the use of the entire training set for training, without sacrificing any training patterns from cross validation. We also applied CSA to train ANNs formulated as constrained optimization problems. Experimental results on *sunspot* and *laser* show that our proposed CSA&V2 for time-series prediction works well and outperforms previous designs in terms of better prediction quality and less number of weights.

# References

[1] A. Aussem. Dynamical recurrent neural networks towards prediction and modeling of dy-




a) Single-step (*laser*) b) Iterative (*laser*)

Figure 5: Prediction performance of an ANN trained by CSA&V2 for the *laser* time-series.

namical systems. *Neurocomputing*, 28:207–232, 1999.

[2] A. B. Geva. ScaleNet – multiscale neural-network architecture for time series prediction. *IEEE Trans. on Neural Networks*, 9(5):1471–1482, September 1998.

[3] W. L. Goffe, G. D. Ferrier, and J. Rogers. Global optimization of statistical functions with simulated annealing. *Journal of Econometrics*, 60:65–99, 1994.

[4] S. Nowlan and G. Hinton. Simplifying neural networks by soft weight sharing. *Neural Computation*, 4(4):473–493, 1992.

[5] Y. Shang and B. W. Wah. Global optimization for neural network training. *IEEE Computer*, 29:45–54, March 1996.

[6] H. Tong and K. Lim. Threshold autoregression, limit cycles and cyclical data. *J. Royal Statistic. Soc.*, 42:245–292, 1980.

[7] B. W. Wah and M.-L. Qian. Constrained formulations for neural network training and their applications to solve the two-spiral problem. In *Proc. Fifth Int'l Conf. on Computer Science and Informatics*, volume 1, pages 598–601, February 2000.

[8] B. W. Wah and T. Wang. Simulated annealing with asymptotic convergence for nonlinear constrained global optimization. *Principles and Practice of Constraint Programming*, pages 461–475, October 1999.

[9] B. W. Wah and Z. Wu. The theory of discrete Lagrange multipliers for nonlinear discrete optimization. *Principles and Practice of Constraint Programming*, pages 28–42, October 1999.

[10] E. Wan. Combining fossils and sunspots: Committee predictions. In *IEEE Int'l Conf. on Nerual Networks*, volume 4, pages 2176–2180, Houston, USA, June 1997.

[11] E. A. Wan. *Finite Impulse Response Neural Networks with Applications in Time Series Prediction.* PhD thesis, Standford University, 1993.

[12] A. S. Weigend, B. A. Huberman, and D. E. Rumelhart. Predicting the future: A connectionist approach. *Int'l. J. of Neural Systems*, 1(3):209, 1990.

[13] R. J. Williams and J. Peng. An efficient gradient-based algorithm for on-line training of recurrent networks trajectories. *Neural Computation*, 2(4):490–501, 1990.

[14] Z. Wu. *The Theory and Applications of Discrete Lagrange Multipliers in Constrained Nonlinear Programming.* Ph.D. Thesis, Dept. of Computer Science, Univ. of Illinois, Urbana, IL, August 2000.