Grammatical Inference by Attentional Control of Synchronization in an Oscillating Elman Network

Bill Baird
Dept Mathematics,
U.C.Berkeley,
Berkeley, Ca. 94720,
baird@math.berkeley.edu

Todd Troyer
Dept of Phys.,
U.C.San Francisco,
513 Parnassus Ave.
San Francisco, Ca. 94143,
todd@phy.ucsf.edu

Frank Eeckman
Lawrence Livermore
National Laboratory,
P.O. Box 808 (L-270),
Livermore, Ca. 94550,
eeckman@.llnl.gov

Abstract

We show how an "Elman" network architecture, constructed from recurrently connected oscillatory associative memory network modules, can employ selective "attentional" control of synchronization to direct the flow of communication and computation within the architecture to solve a grammatical inference problem.

Previously we have shown how the discrete time "Elman" network algorithm can be implemented in a network completely described by continuous ordinary differential equations. The time steps (machine cycles) of the system are implemented by rhythmic variation (clocking) of a bifurcation parameter. In this architecture, oscillation amplitude codes the information content or activity of a module (unit), whereas phase and frequency are used to "softwire" the network. Only synchronized modules communicate by exchanging amplitude information; the activity of non-resonating modules contributes incoherent crosstalk noise.

Attentional control is modeled as a special subset of the hidden modules with ouputs which affect the resonant frequencies of other hidden modules. They control synchrony among the other modules and direct the flow of computation (attention) to effect transitions between two subgraphs of a thirteen state automaton which the system emulates to generate a Reber grammar. The internal crosstalk noise is used to drive the required random transitions of the automaton.

1 Introduction

Recordings of local field potentials have revealed 40 to 80 Hz oscillation in vertebrate cortex [Freeman and Baird, 1987, Gray and Singer, 1987]. The amplitude patterns of such oscillations have been shown to predict the olfactory and visual pattern recognition responses of a trained animal. There is further evidence that although the oscillatory activity appears to be roughly periodic, it is actually chaotic when examined in detail. This preliminary evidence suggests that oscillatory or chaotic network modules may form the cortical substrate for many of the sensory, motor, and cognitive functions now studied in static networks.

It remains be shown how networks with more complex dynamics can performs these operations and what possible advantages are to be gained by such complexity. We have therefore constructed a parallel distributed processing architecture that is inspired by the structure and dynamics of cerebral cortex, and applied it to the problem of grammatical inference. The construction views cortex as a set of coupled oscillatory associative memories, and is guided by the principle that attractors must be used by macroscopic systems for reliable computation in the presence of noise. This system must function reliably in the midst of noise generated by crosstalk from it's own activity. Present day digital computers are built of flip-flops which, at the level of their transistors, are continuous dissipative dynamical systems with different attractors underlying the symbols we call "0" and "1". In a similar manner, the network we have constructed is a symbol processing system, but with analog input and oscillatory subsymbolic representations.

The architecture operates as a thirteen state finite automaton that generates the symbol strings of a Reber grammar. It is designed to demonstrate and study the following issues and principles of neural computation: (1) Sequential computation with coupled associative memories. (2) Computation with attractors for reliable operation in the presence of noise. (3) Discrete time and state symbol processing arising from continuum dynamics by bifurcations of attractors. (4) Attention as selective synchronization controling communication and temporal program flow. (5) chaotic dynamics in some network modules driving randomn choice of attractors in other network modules. The first three issues have been fully addressed in a previous paper [Baird et al., 1993], and are only briefly reviewed. We focus here on the last two.

1.1 Attentional Processing

An important element of intra-cortical communication in the brain, and between modules in this architecture, is the ability of a module to detect and respond to the proper input signal from a particular module, when inputs from other modules irrelevant to the present computation are contributing crosstalk noise. This is smilar to the problem of coding messages in a computer architecture like the Connection Machine so that they can be picked up from the common communication buss line by the proper receiving module.

Periodic or nearly periodic (chaotic) variation of a signal introduces additional degrees of freedom that can be exploited in a computational architecture. We investigate the principle that selective control of synchronization, which we hypopthesize to be a model of "attention", can be used to solve this coding problem and control communication and program flow in an architecture with dynamic attractors.

The architecture illustrates the notion that synchronization not only "binds" sensory inputs into "objects" [Gray and Singer, 1987], but binds the activity of selected cortical areas into a functional whole that directs behavior. It is a model of "attended activity" as that subset which has been included in the processing of the moment by synchronization. This is both a spatial and temporal binding. Only the inputs which are synchronized to the internal oscillatory activity of a module can effect previously learned transitions of attractors within it. For example, consider two objects in the visual field separately bound in primary visual cortex by synchronization of their components at different phases or frequencies. One object may be selectively attended to by its entrainment to oscillatory processing at higher levels such as V4 or IT. These in turn are in synchrony with oscillatory activity in motor areas to select the attractors there which are directing motor output.

In the architecture presented here, we have constrained the network dynamics so that there exist well defined notions of amplitude, phase, and frequency. The network has been designed so that amplitude codes the information content or activity of a module, whereas phase and frequency are used to "softwire" the network. An oscillatory network module has a passband outside of which it will not synchronize with an oscillatory input. Modules can therefore easily be desynchronized by perturbing their resonant frequencies. Furthermore, only synchronized modules communicate by exchanging amplitude information; the activity of non-resonating modules contributes incoherant crosstalk or noise. The flow of communication between modules can thus be controled by controlling synchrony. By changing the intrinsic frequency of modules in a patterned way, the *effective* connectivity of the network is changed. The same hardware and connection matrix can thus subserve many different computations and patterns of interaction between modules without crosstalk problems.

The crosstalk noise is actually essential to the function of the system. It serves as the noise source for making random choices of output symbols and automaton state transitions in this architecture, as we discuss later. In cortex there is an issue as to what may constitute a source of randomness of sufficient magnitude to perturb the large ensemble behavior of neural activity at the cortical network level. It does not seem likely that the well known molecular fluctuations which are easily averaged within one or a few neurons can do the job. The architecture here models the hypothesis that deterministic chaos in the macroscopic dynamics of a network of neurons, which is the same order of magnitude as the coherant activity, can serve this purpose.

In a set of modules which is desynchronized by perturbing the resonant frequencies of the group, coherance is lost and "random" phase relations result. The character of the model time traces is irregular as seen in real neural ensemble activity. The behavior of the time traces in different modules of the architecture is similar to the temporary appearance and switching of synchronization between cortical areas seen in observations of cortical processing during sensory/motor tasks in monkeys and humans [Bressler and Nakamura, 1993]. The structure of this apparently chaotic signal and its use in network learning and operation are currently under investigation.

2 Normal Form Associative Memory Modules

The mathematical foundation for the construction of network modules is contained in the normal form projection algorithm [Baird and Eeckman, 1993]. This is a learning algorithm for recurrent analog neural networks which allows associative memory storage of analog patterns, continuous periodic sequences, and chaotic attractors in the same network. An N node module can be shown to function as an associative memory for up to $N/2$ oscillatory, or $N/3$ chaotic memory attractors [Baird and Eeckman, 1993]. A key feature of a net constructed by this algorithm is that the underlying dynamics is explicitly isomorphic to any of a class of standard, well understood nonlinear dynamical systems - a *normal form* [Guckenheimer and Holmes, 1983].

The network modules of this architecture were developed previously as models of olfactory cortex with distributed patterns of activity like those observed experimentally [Baird, 1990, Freeman and Baird, 1987]. Such a biological network is dynamically equivalent to a network in normal form and may easily be designed, simulated, and theoretically evaluated in these coordinates. When the intramodule competition is high, they are "memory" or winner-take-all cordinates where attractors have one oscillator at maximum amplitude, with the other amplitudes near zero. In figure two, the input and output modules are demonstrating a distributed amplitude pattern (the symbol "T"), and the hidden and context modules are two-attractor modules in normal form coordinates showing either a right or left side active.

In this paper all networks are discussed in normal form coordinates. By analyzing the network in these coordinates, the amplitude and phase dynamics have a particularly simple interaction. When the input to a module is synchronized with its intrinsic oscillation, the amplitude of the periodic activity may be considered separately from the phase rotation. The module may then be viewed as a static network with these amplitudes as its activity.

To illustrate the behavior of individual network modules, we examine a binary (two-attractor) module; the behavior of modules with more than two attractors is similar. Such a unit is defined in polar normal form coordinates by the following equations of the Hopf normal form:

$$\begin{aligned}
\dot{r}_{1i} &= u_i r_{1i} - c r_{1i}^3 + (d - b sin(\omega_{clock} t)) r_{1i} r_{0i}^2 + \sum_j w_{ij}^{+} I_j \cos(\theta_j - \theta_{1i}) \\
\dot{r}_{0i} &= u_i r_{0i} - c r_{0i}^3 + (d - b sin(\omega_{clock} t)) r_{0i} r_{1i}^2 + \sum_j w_{ij}^{-} I_j \cos(\theta_j - \theta_{0i}) \\
\dot{\theta}_{1i} &= \omega_i + \sum_j w_{ij}^{+} (I_j / r_{1i}) \sin(\theta_j - \theta_{1i}) \\
\dot{\theta}_{0i} &= \omega_i + \sum_j w_{ij}^{-} (I_j / r_{0i}) \sin(\theta_j - \theta_{0i})
\end{aligned}$$

The clocked parameter $bsin(\omega_{clock} t)$ is used to implement the discrete time machine cycle of the Elman architecture as discussed later. It has lower frequency (1/10) than the intrinsic frequency of the unit ω_i.

Examination of the phase equations shows that a unit has a strong tendency to synchronize with an input of similar frequency. Define the phase difference $\phi = \theta_0 - \theta_I = \theta_0 - \omega_I t$ between a unit θ_0 and it's input θ_I. For either side of a unit driven by an input of the same frequency, $\omega_I = \omega_0$, There is an attractor at zero phase difference $\phi = \theta_0 - \theta_I = 0$ and a repellor at $\phi = 180$ degrees. In simulations, the interconnected network of these units described below synchronizes robustly within a few cycles following a perturbation. If the frequencies of some modules of the architecture are randomly dispersed by a significant amount, $\omega_I - \omega_0 \neq 0$, phase-lags appear first, then synchronization is lost in those units. An oscillating module therefore acts as a band pass filter for oscillatory inputs.

When the oscillators are sychronized with the input, $\theta_j - \theta_{1i} = 0$, the phase terms $\cos(\theta_j - \theta_{1i}) = \cos(0) = 1$ dissappear. This leaves the amplitude equations $\dot{r}_{1i}$ and $\dot{r}_{0i}$ with static inputs $\sum_j w_{ij}^{+} I_j$ and $\sum_j w_{ij}^{-} I_j$. Thus we have network modules which emulate static network units in their amplitude activity when fully phase-locked to their input. Amplitude information is transmitted between modules, with an oscillatory carrier.

For fixed values of the competition, in a completely synchronized system, the internal amplitude dynamics define a gradient dynamical system for a fourth order energy function. External inputs that are phase-locked to the module's intrinsic oscillation simply add a linear tilt to the landscape.

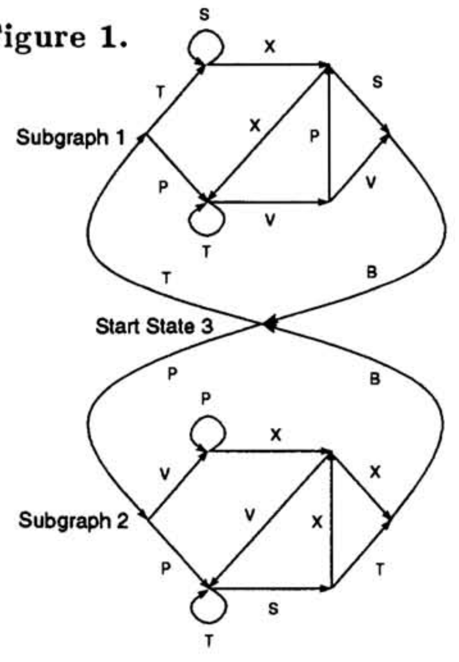
For low levels of competition, there is a broad circular valley. When tilted by external input, there is a unique equilibrium that is determined by the bias in tilt along one axis over the other. Thinking of r_{1i} as the "acitivity" of the unit, this acitivity becomes a monotonically increasing function of input. The module behaves as an analog connectionist unit whose transfer function can be approximated by a sigmoid. We refer to this as the "analog" mode of operation of the module.

With high levels of competition, the unit will behave as a binary (bistable) digital flip-flop element. There are two deep potential wells, one on each axis. Hence the module performs a winner-take-all choice on the coordinates of its initial state and maintains that choice "clamped" and independent of external input. This is the "digital" or "quantized" mode of operation of a module. We think of one attractor within the unit as representing "1" (the right side in figure two) and the other as representing "0".

3 Elman Network of Oscillating Associative Memories

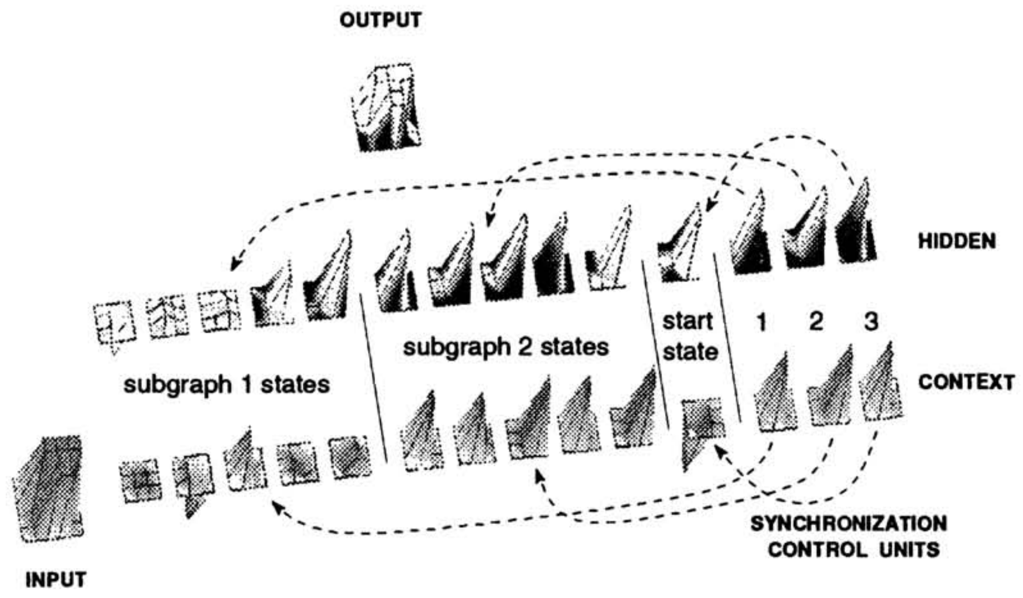
As a benchmark for the capabilities of the system, and to create a point of contact to standard network architectures, we have constructed a discrete-time recurrent "Elman" network [Elman, 1991] from oscillatory modules defined by ordinary differential equations. Previously we constructed a system which functions as the six state finite automaton that perfectly recognizes or generates the set of strings defined by the Reber grammar described in Cleeremans et. al. [Cleeremans et al., 1989]. We found the connections for this network by using the backpropagation algorithm in a static network that approximates the behavior of the amplitudes of oscillation in a fully synchronized dynamic network [Baird et al., 1993].
Here we construct a system that emulates the larger 13 state automata similar (less one state) to the one studied by Cleermans, et al in the second part of their paper. The graph of this automaton consists of two subgraph branches each of which has the graph structure of the automaton learned as above, but with different assignments of transition output symbols (see fig. 1).

Figure 1.



We use two types of modules in implementing the Elman network architecture shown in figure two below. The input and output layer each consist of a single associative memory module with six oscillatory attractors (six competing oscillatory modes), one for each of the six symbols in the grammar. The hidden and context layers consist of the binary "units" above composed of a two oscillatory attractors. The architecture consists of 14 binary modules in the hidden and context layers - three of which are special frequency control modules. The hidden and context layers are divided into four groups: the first three correspond to each of the two subgraphs plus the start state, and the fourth group consists of three special control modules, each of which has only a special control output that perturbs the resonant frequencies of the modules (by changing their values in the program) of a particular state coding group when it is at the zero attractor, as illustrated by the dotted control lines in figure two. This figure shows control unit two is at the one attractor (right side of the square active) and the hidden units coding for states of subgraph two are in synchrony with the input and output modules. Activity levels oscillate up and down through the plane of the paper. Here in midcycle, competition is high in all modules.

Figure 2. OSCILLATING ELMAN NETWORK



The discrete machine cycle of the Elman algorithm is implemented by the sinusoidal variation (clocking) of the bifurcation parameter in the normal form equations that determines the level of intramodule competition [Baird et al., 1993]. At the beginning of a machine cycle, when a network is generating strings, the input and context layers are at high competition and their activity is clamped at the bottom of deep basins of attraction. The hidden and output modules are at low competition and therefore behave as a traditional feedforward network free to take on analog values. In this analog mode, a real valued error can be defined for the hidden and output units and standard learning algorithms like backpropagation can be used to train the connections.

Then the situation reverses. For a Reber grammar there are always two equally possible next symbols being activated in the output layer, and we let the crosstalk noise break this symmetry so that the winner-take-all dynamics of the output module can chose one. High competition has now also "quantized" and clamped the activity in the hidden layer to a fixed binary vector. Meanwhile, competition is lowered in the input and context layers, freeing these modules from their attractors. An identity mapping from hidden to context loads the binarized activity of the hidden layer into the context layer for the next cycle, and an additional identity mapping from the output to input module places the chosen output symbol into the input layer to begin the next cycle.

4 Attentional control of Synchrony

We introduce a model of attention as control of program flow by selective synchronization. The attentional controler itself is modeled in this architecture as a special set of three hidden modules with ouputs that affect the resonant frequencies of the other corresponding three subsets of hidden modules. Varying levels of intramodule competition control the large scale *direction* of information flow *between* layers of the architecture. To direct information flow on a finer scale, the attention mechanism selects a subset of modules *within* each layer whose output is effective in driving the state transition behavior of the system.

By controling the patterns of synchronization within the network we are able to generate the grammar obtained from an automaton consisting of two subgraphs connected by a single transition state (figure 1). During training we enforce a segregation of the hidden layer code for the states of the separate subgraph branches of the automaton so that different sets of synchronized modules learn to code for each subgraph of the automaton. Then the entire automaton is hand constructed with an additional hidden module for the start state between the branches. Transitions in the system from states in one subgraph of the automaton to the other are made by "attending" to the corresponding set of nodes in the hidden and context layers. This switching of the focus of attention is accomplished by changing the patterns of synchronization within the network which changes the flow of communication between modules.

Each control module modulates the intrinsic frequency of the units coding for the states a single subgraph or the unit representing the start state. The control modules respond to a particular input symbol and context to set the intrinsic frequency of the proper subset of hidden units to be equal to the input layer frequency. As described earlier, modules can easily be desynchronized by perturbing their resonant frequencies. By perturbing the frequencies of the remaining modules away from the input frequency, these modules are no longer communicating with the rest of the network. Thus coherent information flows from input to output only through one of three channels. Viewing the automata as a behavioral program, the control of synchrony constitutes a control of the program flow into its subprograms (the subgraphs of the automaton).

When either exit state of a subgraph is reached, the "B" (begin) symbol is then emitted and fed back to the input where it is connected through the first to second layer weight matrix to the attention control modules. It turns off the synchrony of the hidden states of the subgraph and allows entrainment of the start state to begin a new string of symbols. This state in turn activates both a "T" and a "P' in the output module. The symbol selected by the crosstalk noise and fed back to the input module is now connected to the control modules through the weight matrix. It desynchronizes the start state module, synchronizes in the subset of hidden units coding for the states of the appropriate subgraph, and establishes there the start state pattern for that subgraph.

Future work will investigate the possibilities for self-organization of the patterns of synchrony and spatially segregated coding in the hidden layer during learning. The weights for entire automata, including the special attention control hidden units, should be learned at once.

4.1 Acknowledgments

Supported by AFOSR-91-0325, and a grant from LLNL. It is a pleasure to acknowledge the invaluable assistance of Morris Hirsch, and Walter Freeman.

References

[Baird, 1990] Baird, B. (1990). Bifurcation and learning in network models of oscillating cortex. In Forest, S., editor, *Emergent Computation*, pages 365–384. North Holland. also in Physica D, 42.

[Baird and Eeckman, 1993] Baird, B. and Eeckman, F. H. (1993). A normal form projection algorithm for associative memory. In Hassoun, M. H., editor, *Associative Neural Memories: Theory and Implementation*, New York, NY. Oxford University Press.

[Baird et al., 1993] Baird, B., Troyer, T., and Eeckman, F. H. (1993). Synchronization and gramatical inference in an oscillating elman network. In Hanson, S., Cowan, J., and Giles, C., editors, *Advances in Neural Information Processing Systems 5*, pages 236–244. Morgan Kaufman.

[Bressler and Nakamura, 1993] Bressler, S. and Nakamura. (1993). Interarea synchronization in Macaque neocortex during a visual discrimination task. In Eeckman,F. H., and Bower, J., editors, *Computation and Neural Systems*, page 515. Kluwer.

[Cleeremans et al., 1989] Cleeremans, A., Servan-Schreiber, D., and McClelland, J. (1989). Finite state automata and simple recurrent networks. *Neural Computation*, 1(3):372–381.

[Elman, 1991] Elman, J. (1991). Distributed representations, simple recurrent networks and grammatical structure. *Machine Learning*, 7(2/3):91.

[Freeman and Baird, 1987] Freeman, W. and Baird, B. (1987). Relation of olfactory EEG to behavior: Spatial analysis. *Behavioral Neuroscience*, 101:393–408.

[Gray and Singer, 1987] Gray, C. M. and Singer, W. (1987). Stimulus dependent neuronal oscillations in the cat visual cortex area 17. *Neuroscience [Suppl]*, 22:1301P.

[Guckenheimer and Holmes, 1983] Guckenheimer, J. and Holmes, D. (1983). *Nonlinear Oscillations, Dynamical Systems, and Bifurcations of Vector Fields.* Springer, New York.

ADAPTIVE KNOT PLACEMENT FOR NONPARAMETRIC REGRESSION

Hossein L. Najafi*
Department of Computer Science
University of Wisconsin
River Falls, WI 54022

Vladimir Cherkassky
Department of Electrical Engineering
University of Minnesota
Minneapolis, Minnesota 55455

Abstract

Performance of many nonparametric methods critically depends on the strategy for positioning knots along the regression surface. Constrained Topological Mapping algorithm is a novel method that achieves adaptive knot placement by using a neural network based on Kohonen's self-organizing maps. We present a modification to the original algorithm that provides knot placement according to the estimated second derivative of the regression surface.

1 INTRODUCTION

Here we consider regression problems. Using mathematical notation, we seek to find a function f of $N-1$ predictor variables (denoted by vector $\mathbf{X}$) from a given set of n data points, or measurements, $\mathbf{Z}_i = (\mathbf{X}_i\ , Y_i)$ $(i = 1, \ldots, n)$ in N - dimensional sample space:

$$Y = f(\mathbf{X}) + error \tag{1}$$

where error is unknown (but zero mean) and its distribution may depend on $\mathbf{X}$. The distribution of points in the training set can be arbitrary, but uniform distribution in the domain of $\mathbf{X}$ is often used.

*Responsible for correspondence, Telephone (715) 425-3769, e-mail hossein.najafi@uwrf.edu.

The goal of this paper is to show how statistical considerations can be used to improve the performance of a novel neural network algorithm for regression [CN91], in order to achieve adaptive positioning of knots along the regression surface. By estimating and employing the second derivative of the underlying function, the modified algorithm is made more flexible around the regions with large second derivative. Through empirical investigation, we show that this modified algorithm allocates more units around the regions where the second derivative is large. This increase in the local knot density introduces more flexibility into the model (around the regions with large second derivative) and makes the model less biased around these regions. However, no over-fitting is observed around these regions.

2 THE PROBLEM OF KNOT LOCATION

One of the most challenging problems in practical implementations of adaptive methods for regression is adaptive positioning of knots along the regression surface. Typically, knot positions in the domain of $\mathbf{X}$ are chosen as a subset of the training data set, or knots are uniformly distributed in $\mathbf{X}$. Once $\mathbf{X}$-locations are fixed, commonly used data-driven methods can be applied to determine the number of knots. However, de Boor [dB78] showed that a polynomial spline with unequally spaced knots can approximate an arbitrary function much better than a spline with equally spaced knots. Unfortunately, the minimization problem involved in determination of the optimal placement of knots is highly nonlinear and the solution space is not convex [FS89]. Hence, the performance of many recent algorithms that include adaptive knot placement (e.g. MARS) is difficult to evaluate analytically. In addition, it is well-known that when data points are uniform, more knots should be located where the second derivative of the function is large. However, it is difficult to extend these results for non-uniform data in conjunction with data-dependent noise. Also, estimating the second derivative of a true function is necessary for optimal knot placement. Yet, the function itself is unknown and its estimation depends on the good placement of knots. This suggests the need for some iterative procedure that alternates between function estimation(smoothing) and knot positioning steps.

Many ANN methods effectively try to solve the problem of adaptive knot location using ad hoc strategies that are not statistically optimal. For example, local adaptive methods [Che92] are generalization of kernel smoothers where the kernel functions and kernel centers are determined from the data by some adaptive algorithm. Examples of local adaptive methods include several recently proposed ANN models known as radial basis function (RBF) networks, regularization networks, networks with locally tuned units etc [BL88, MD89, PG90]. When applied to regression problems, all these methods seek to find regression estimate in the (most general) form $\sum_{i=1}^{k} b_i \mathbf{H}_i(\mathbf{X}, C_i)$ where $\mathbf{X}$ is the vector of predictor variable, C_i is the coordinates of the i-th 'center' or 'bump', $\mathbf{H}_i$ is the response function of the kernel type (the kernel width may be different for each center i), b_i are linear coefficients to be determined, and k is the total number of knots or 'centers'.

Whereas the general formulation above assumes global optimization of an error measure for the training set with respect to all parameters, i.e. center locations, kernel width and linear coefficients, this is not practically feasible because the error surface is generally non-convex and may have local minima [PG90, MD89]. Hence most practical approaches first solve the problem of center(knot) location and assume identical kernel functions. Then the remaining problem of finding linear coefficients b_i is solved by using familiar methods of Linear Algebra [PG90] or gradient-descent techniques [MD89]. It appears that the problem of center locations is the most critical one for the local neural network techniques. Unfortunately, heuristics used for center location are not based on any statistical considerations, and empirical results are too sketchy [PG90, MD89]. In statistical methods knot locations are typically viewed as free parameters of the model, and hence the number of knots directly controls the model complexity. Alternatively, one can impose local regularization constraints on adjacent knot locations, so that neighboring knots cannot move independently. Such an approach is effectively implemented in the model of self-organization known as Kohonen's Self-Organizing Maps (SOM) [Koh84]. This model uses a set of units ("knots") with neighborhood relations between units defined according to a fixed topological structure (typically 1D or 2D grid). During training or self-organization, data points are presented to the map iteratively, one at a time, and the unit closest to the data moves towards it, also pulling along its topological neighbors.

3 MODIFIED CTM ALGORITHM FOR ADAPTIVE KNOT PLACEMENT

The SOM model has been applied to nonparametric regression by Cherkassky and Najafi [CN91] in order to achieve adaptive positioning of knots along the regression surface. Their technique, called Constrained Topological Mapping (CTM), is a modification of Kohonen's self-organization suitable for regression problems. CTM interprets the units of the Kohonen map as movable knots of a regression surface. Correspondingly, the problem of finding regression estimate can be stated as the problem of forming an M - dimensional topological map using a set of samples from N - dimensional sample space (where $M \leq N-1$) . Unfortunately, straightforward application of the Kohonen Algorithm to regression problem does not work well [CN91]. Because, the presence of noise in the training data can fool the algorithm to produce a map that is a multiple-valued function of independent variables in the regression problem (1). This problem is overcome in the CTM algorithm, where the nearest neighbor is found in the subspace of predictor variables, rather than in the input(sample) space [CN91].

We present next a concise description of the CTM algorithm. Using standard formulation (1) for regression, the training data are N - dimensional vectors $\mathbf{Z}_i = (\mathbf{X}_i$, $Y_i)$, where Yi is a noisy observation of an unknown function of $N-1$ predictor variables given by vector $\mathbf{X}_i$. The CTM algorithm constructs an M - dimensional topological map in N - dimensional sample space ($M \leq N-1$) as follows:

0. Initialize the M - dimensional topological map in N - dimensional sample space.
1. Given an input vector $\mathbf{Z}$ in N - dimensional sample space, find the closest (best matching) unit i in the subspace of independent variables:

$$\| \mathbf{Z}^*(k) - \mathbf{W}_i^* \| = Min_j\{\|\mathbf{Z}^* - \mathbf{W}_j^* \|\} \qquad \forall j \in [1, \ldots, L]$$

where $\mathbf{Z}^*$ is the projection of the input vector onto the subspace of independent variables, $\mathbf{W}_j^*$ is the projection of the weight vector of unit j, and k is the discrete time step.

2. Adjust the units' weights according to the following and return to 1:

$$\mathbf{W}_j(k+1) = W_j(k) + \beta(k)C_j(k)(\mathbf{Z}(k) - \mathbf{W}_j(k)) \qquad \forall j \qquad (2)$$

where $\beta(k)$ is the learning rate and $C_j(k)$ is the neighborhood for unit j at iteration k and are given by:

$$\beta(k) = \beta_0 \times \left(\frac{\beta_f}{\beta_0}\right)^{\left(\frac{k}{k_{max}}\right)}, C_j(k) = \frac{1}{\exp^{0.5\left(\frac{\|i-j\|}{\beta(k)\times S_0}\right)^2}} \qquad (3)$$

where k_{max} is the final value of the time step (k_{max} is equal to the product of the training set size by the number of times it was recycled), β_0 is the initial learning rate, and β_f is the final learning rate ($\beta_0 = 1.0$ and $\beta_f = 0.05$ were used in all of our experiments), $\|i - j\|$ is the topological distance between the unit j and the best matched unit i and S_0 is the initial size of the map (i.e., the number of units per dimension) .

Note that CTM method achieves placement of units (knots) in X-space according to density of training data. This is due to the fact that X-coordinates of CTM units during training follow the standard Kohonen self-organization algorithm [Koh84], which is known to achieve faithful approximation of an unknown distribution. However, existing CTM method does not place more knots where the underlying function changes rapidly. The improved strategy for CTM knot placement in X-space takes into account estimated second derivative of a function as is described next.

The problem with estimating second derivative is that the function itself is unknown. This suggests using an iterative strategy for building a model, i.e., start with a crude model, estimate the second derivative based on this crude model, use the estimated second derivative to refine the model, etc. This strategy can be easily incorporated into the CTM algorithm due to its iterative nature. Specifically, in CTM method the map of knots(i.e., the model) becomes closer and closer to the final regression model as the training proceeds. Therefore, at each iteration, the modified algorithm estimates the second derivative at the best matching unit (closest to the presented data point in X-space), and allows additional movement of knots proportional to this estimate. Estimating the second derivative from the map (instead of using the training data) makes sense due to smoothing properties of CTM.

The modified CTM algorithm can be summarized as follows:

1. Present training sample $\mathbf{Z}_i = (\mathbf{X}_i\ , Y_i)$ to the map and find the closest (best matching) unit i in the subspace of independent variables to this data point. (same as in the original CTM)
2. Move the the map (i.e., the best matching unit and all its neighbors) toward the presented data point (same as in the original CTM)

3. Estimate average second derivative of the function at the best matching unit based on the current positions of the map units.
4. Normalize this average second derivative to an interval of [0,1].
5. Move the map toward the presented data point at a rate proportional to the estimated normalizes average second derivative and iterate.

For multivariate functions only gradients along directions given by the topological structure of the map can be estimated in step 4. For example, given a 2-dimensional mesh that approximates function $f(x_1, x_2)$, every unit of the map (except the border units for which there will be only one neighbor) has two neighboring units along each topological dimension. These neighboring units can be used to approximate the function's gradients along the corresponding topological dimension of the map. These values along each dimension can then be averaged to provide a local gradient estimate at a given knot.

In step 5, estimated average second derivative f'' is normalized to [0,1] range using $\psi_i = 1 - \exp^{(|f''|/\tan(\tau))}$ This is done because the value of second derivative is used as the learning rate.

In step 6, the map is modified according to the following equation:

$$\mathbf{W}_j(k+1) = W_j(k) + (1 - \beta(k))\psi_i(k)C_j(k)(\mathbf{X}(k) - \mathbf{W}_j(k)) \qquad \forall j \quad (4)$$

It is this second movement of the map that allows for more flexibility around the region of the map where the second derivative is large. The process described by equation (4) is equivalent to pulling all units towards the data, with the learning rate proportional to estimated second derivative at the best matched unit. Note that the influence of the second derivative is gradually increased during the process of self-organization by the factor $(1-\beta(k))$. This factor account for the fact that the map becomes closer and closer to the underlying function during self-organization; hence, providing a more reliable estimate of second derivative.

4 EMPIRICAL COMPARISON

Performance of the two algorithms (original and modified CTM) was compared for several low-dimensional problems. In all experiments the two algorithms used the same training set of 100 data points for the univariate problems and 400 data points for the 2-variable problems.

The training samples $(\mathbf{X}_i, Y_i)$ were generated according to (1), with $\mathbf{X}_i$ randomly drawn from a uniform distribution in the closed interval [-1,1], and the *error* drawn from the normal distribution $\mathbf{N}(0, (0.1)^2)$. Regression estimates produced by the self-organized maps were tested on a different set of $\mathbf{n} = 200$ samples (test set) generated in the same manner as the training set.

We used the Average Residual, $AR = \sqrt{\frac{1}{n} \sum_{i=1}^{n}[Y_i - f(\mathbf{X}_i)]^2}$, as the performance measure on the test set. Here, $f(X)$ is the piecewise linear estimate of the function with knot locations provided by coordinates of the units of trained CTM. The Average Residual gives an indication of standard deviation of the overall generalization error.

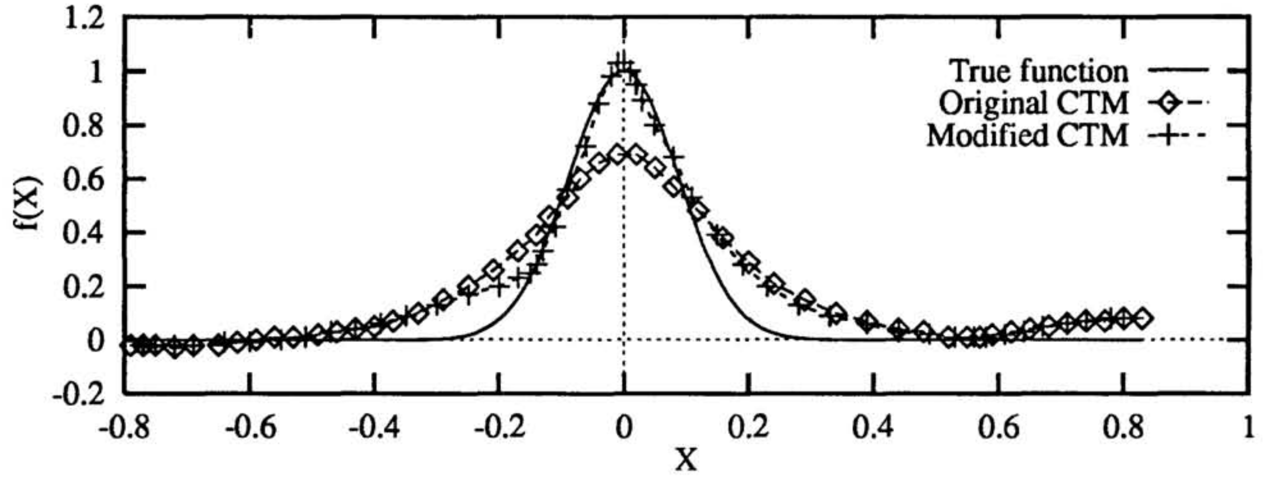


Figure 1: A 50 unit map formed by the original and modified algorithm for the Gaussian function.

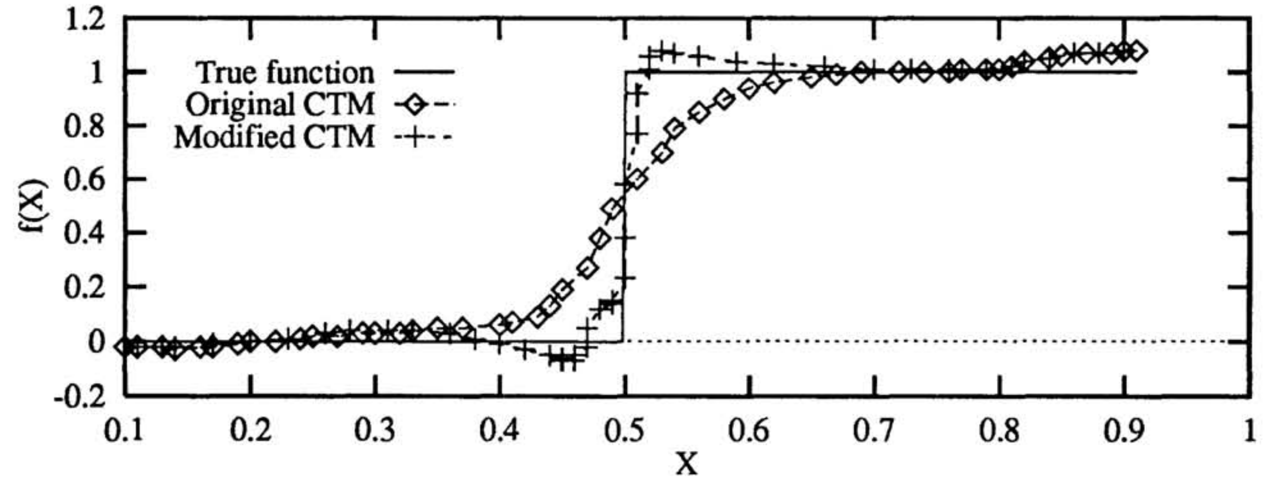


Figure 2: A 50 unit map formed by the original and modified algorithm for the step function.

We used a gaussian function ($f(x) = \exp^{-64x^2}$) and a step function for our first set of experiments. Figure 1 and 2 show the actual maps formed by the original and modified algorithm for these functions. It is clear from these figures that the modified algorithm allocates more units around the regions where the second derivative is large. This increase in the local knot density has introduced more flexibility into the model around the regions with large second derivatives. As a result of this the model is less biased around these regions. However, there is no over-fitting in the regions where the second derivative is large.

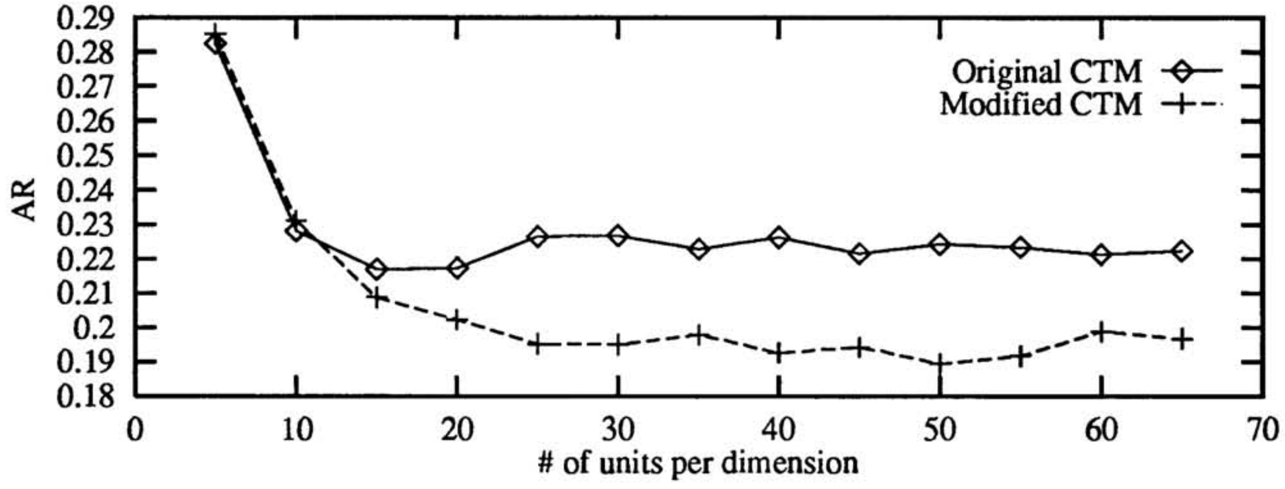


Figure 3: Average Residual error as a function of the size of the map for the 3-dimensional Step function

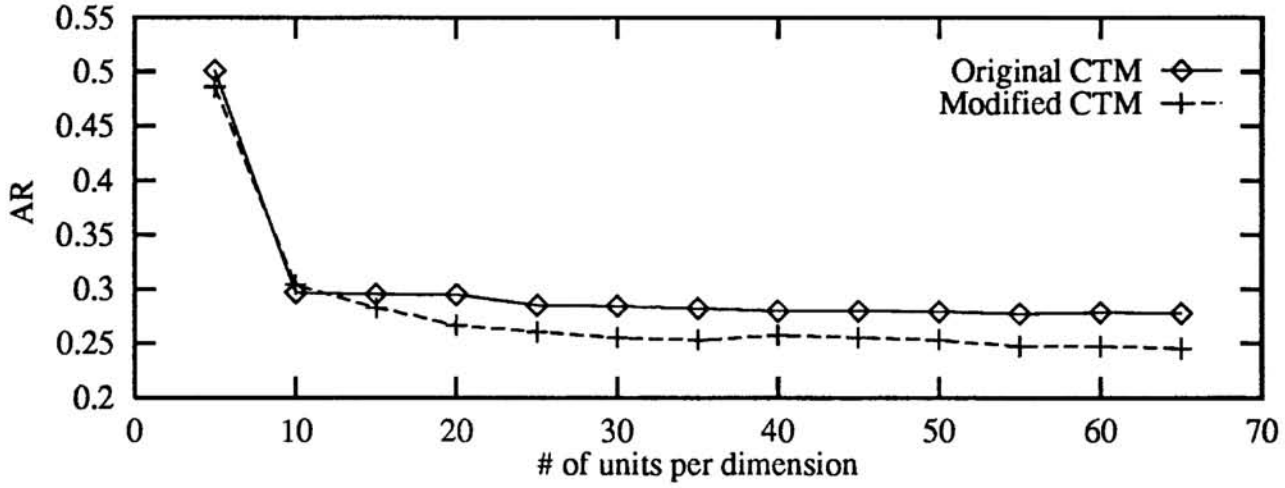


Figure 4: Average Residual error as a function of the size of the map for the 3-dimensional Sine function

To compare the behavior of the two algorithms in their predictability of structureless data, we trained them on a constant function $f(x) = 0$ with $error = \mathrm{N}(0, (0.1)^2)$. This problem is known as smoothing pure noise in regression analysis. It has been shown [CN91] that the original algorithm handles this problem well and quality of CTM smoothing is independent of the number of units in the map. Our experiments show that the modified algorithm performs as good as the original one in this respect.

Finally, we used the following two-variable functions (step, and sine) to see how well the modified algorithm performs in higher dimensional settings.

Step: $f(x_1, x_2) = \begin{cases} 1 & \text{for } ((x_1 < 0.5) \wedge (x_2 < 0.5)) \vee ((x_1 \geq 0.5) \wedge (x_2 \geq 0.5)) \\ 0 & \text{otherwise} \end{cases}$

Sine: $f(x_1, x_2) = \sin\left(2\pi\sqrt{(x_1)^2 + (x_2)^2}\right)$

The results of these experiments are summarized in Figure 3 and 4. Again we see that the modified algorithm outperforms the original algorithm. Note that the above example of a two-variable step function can be easily handled by recursive partitioning techniques such as CART [BFOS84]. However, recursive methods are sensitive to coordinate rotation. On the other hand, CTM is a coordinate-independent method, i.e. its performance is independent of any affine transformation in X-space.

References

[BFOS84] L. Breiman, J.H. Friedman, R.A. Olshen, and C.J. Stone. *Classification and Regression Trees.* Wadswordth, Belmont, CA, 1984.

[BL88] D.S. Broomhead and D. Lowe. Multivariable functional interpolation and adaptive networks. *Complex Systems*, 2:321–355, 1988.

[Che92] V. Cherkassky. Neural networks and nonparametric regression. In S.Y. Kung, F. Fallside, J.Aa. Sorenson, and C.A. Kamm, editors, *Neural Networks for Signal Processing*, volume II. IEEEE, Piscataway, NJ, 1992.

[CN91] V. Cherkassky and H.L. Najafi. Constrained topological mapping for nonparametric regression analysis. *Neural Networks*, 4:27–40, 1991.

[dB78] C. de Boor. *A Practical Guide to Splines.* Springer-Verlag, 1978.

[FS89] J.H. Friedman and B.W. Silverman. Flexible parsimonious smoothing and additive modeling. *Technometrics*, 31(1):3–21, 1989.

[Koh84] T. Kohonen. *Self-Organization and Associative Memory.* Springer-Verlag, third edition, 1984.

[MD89] J. Moody and C.J. Darken. Fast learning in networks of locally tuned processing units. *Neural Computation*, 1:281, 1989.

[PG90] T. Poggio and F. Girosi. Networks for approximation and learning. *Proceedings of the IEEE*, 78(9):1481–1497, 1990.