Generalization by Neural Networks

Shashi Shekhar
Minesh B. Amin

Department of Computer Science
University of Minnesota
Minneapolis, MN 55455.

ABSTRACT

Neural networks have traditionally been applied to **recognition** problems, and most learning algorithms are tailored to those problems. We discuss the requirements of learning for **generalization**, where the traditional methods based on gradient descent have limited success. We present a new stochastic learning algorithm based on simulated annealing in weight space. We verify the convergence properties and feasibility of the algorithm. We also describe an implementation of the algorithm and validation experiments.

1. Introduction

Neural networks are being applied to a wide variety of applications from speech generation[1], to handwriting recognition[2]. Last decade has seen great advances in design of neural networks for a class of problems called *recognition problems*, and in design of learning algorithms[3-5, 5-7]. The learning of weights for neural network for many recognition problem is no longer a difficult task. However, designing a neural network for generalization problem is not well understood.

Domains of neural network applications can be classified into two broad categories-- **recognition** and **generalization** [1, 8]. For both classes, we first train the neural network on a set of input-output pairs (I_1, O_1), (I_2, O_2),...,(I_n, O_n). In *recognition problems*, the trained network is tested with a previously seen input I_j ($1 \le j \le n$) corrupted by noise as shown in Fig.1. The trained network is expected to reproduce the output O_j corresponding to I_j, in spite of the noise. Shape recognition [9, 10], and handwriting recognition[2] are examples of recognition problems. On the other hand, in *generalization problems*, the trained neural network is tested with input I_{n+1}, which is distinct from the inputs I_1, I_2,...,I_n used for training the network, as shown in Fig.1. The network is expected to correctly predict the output O_{n+1} for the input I_{n+1} from the model it has learned through training. Typical examples of generalization problems are Bond Rating[11] and Robotics[12].

Neural networks for generalization problems are important, since there are many applications, of enormous importance in the real world[13-15] which would benefit from this work. In many of these applications it is difficult to successfully apply either conventional mathematical techniques (e.g., statistical regression) or standard AI approaches (e.g., rule based systems). A neural network with generalization ability will be useful for such domains[11], because it does not require an *a priori* specification of a functional domain model; rather it attempts to learn the underlying domain model from the training input-output examples.

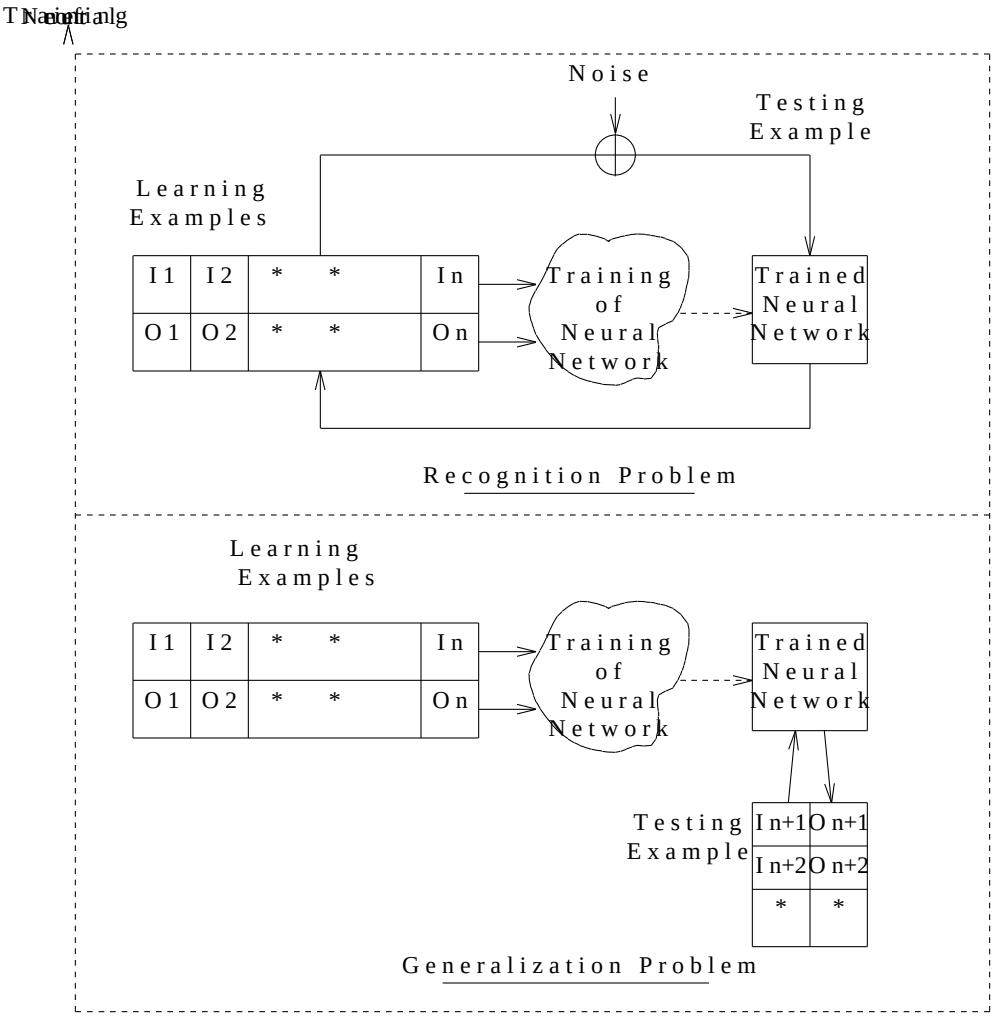

Figure 1. : Classes of Problems

The learning algorithm for generalization problems, should be different from the learning algorithm for recognition problems. In recognition problems, the network is expected to reproduce one of the previously seen outputs. The network may remember the outputs and inputs by fitting a curve through the (I_i, O_i) pairs used for training. To remember the outputs, one often uses large networks with many nodes and weights. However memorization of learning samples is not suited for generalization problems, since it can lead to worse performance during prediction of outputs on unseen inputs. Furthermore, generalization problems allow a small amount of error in the output predicted by the network and hence the fitted curve need not pass through any (I_i, O_i) pair used for training. Networks addressing generalization problem may instead fit a simple ‡ curve (e.g. a low degree polynomial, or basic analytical functions like log(x), sine(x), tangent(x) etc.) through the input-output pairs rather than fitting a crooked curve. The neural network used in generalization problems tend to be simpler with small number of hidden nodes, layers, and interconnection edges and weights, enabling one to use computationally sophisticated algorithms.

Most of the earlier work in neural networks [4, 9] is related to recognition problems. There has been little research towards developing neural network models for generalization problems [16, 17].

We present a new learning algorithm, stochastic backpropagation for generalization problems. We verify the convergence of the algorithm and provide theoretical arguments towards the capability of the proposed algorithm to discover optimal weights. We also describe an implementation of the algorithm and our experience with the algorithms in solving generalization problems.

2. Problem Formulation

Generalization problems for neural networks have been formulated in three different ways: (a) analytical formalism[18-24], (b) constructive function learning [25-27], and (c) symbolic semantic network[8]. The analytical formalism focuses on the existence of networks with a capability to generalize. It also provide worst case time complexity to discover such networks to solve arbitrary generalization problems. It does not provide a way to discover the networks. The constructive function learning formalism approaches generalization problems in a complementary fashion. It studies algorithms to discover networks which can solve a class of generalization problems. Its aims at discovering a function f mapping the input domain to the output domain from a set of learning examples. The function to be discovered may be defined over boolean numbers or over real numbers. The inputs and outputs are assumed to be numbers with no symbolic meaning. The function and network do not represent symbolic meaning beyond the numeric computation. The third approach of symbolic semantic network associates symbolic meaning to the network. Generalization occurs by attaching a new node to an appropriate parent node in the network to inherit the properties of the parent.

This classification of formulations of generalization problems does not include some special cases. For example, signal detection problem can be considered as a special case. The task in signal detection problem is to learn to recognize a parameter of the function f: I -> O, rather than learn the function. Recognizing the frequency of given sinusoidal function is an example of signal detection problem. Backpropagation neural networks has been applied successfully to this problem[28].

We focus on the constructive function learning formulation in terms of learning a numeric function. A simple neural network can be described as a directed graph G = (V, E). The vertex set V has three kinds of nodes: (a) input nodes at leaves, (b) hidden nodes as internal nodes and (c) output nodes at roots. Each edge e_i in E is associated with a weight $w_{i,j}$ as shown in Fig.2.

The network is used to compute an output from a set of inputs. Each node i computes a function of weighted sum of the input signals, $g(\sum_{j \text{ in } inset} w_{ij} s_{ij})$ as its output. The function g maps from [-∞, ∞] to [-1,1]. Given any input, the network would compute an output = f(input). The inverse problem of discovering the function f (i.e. the set of edge weights) from a given set of input-output pairs is referred to as the learning problem, which can be stated as follows. Given a set of example input output pairs $\{(I_1,O_1),..(I_n,O_n)\}$, find the weights on each edge $e_{i,j} \in$ E, of neural network, such that the network maps I_j to O_j for j=1,2,..n, as closely as possible.

Formulation:

Let I represent the possibly infinite domain of inputs, and O represent the possibly infinite range of outputs. Let F represent the k dimensional feature space, F_1, ..., F_k, describing each of the input. Each input I_j can be considered a k-tuple $(F_{1j},F_{2j}, \cdots ,F_{kj})$ in the Cartesian space $F_1 \times F_2 \times \cdots \times F_k$.

Given a learning sample S = (S_I, S_O) and per sample error function E: $I \times O \rightarrow Real$ with $S_I = \{ I_1,I_2, ..., I_n \}$, and $S_O = \{ O_1, O_2, ..., O_n \}$, generalization involves finding the mapping function $f : F_1 \times F_2 \times \cdots \times F_k \rightarrow O$

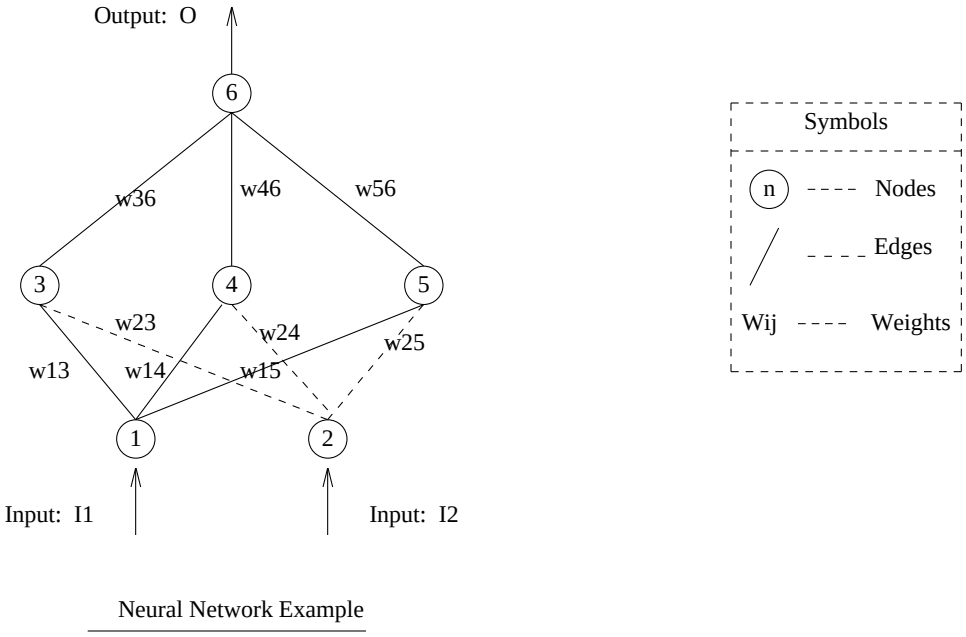

Figure 2.: Feed-forward neural network

to minimize error function E over the entire domain I. In particular $f(I_j) = O_j + <small\ error>$ for each learning sample.

Reducing the error function over entire domain I by looking at a small subset S_I is difficult for arbitrary learning samples S_I and arbitrary input domain I. The generalization problem is often simplified by making assumptions about the domain I and the learning sample S. We assume S_I represents entire domain S adequately so that the value of E over I can be estimated from the the value of E over S_I. The function f is assumed to be smooth, continuous and well behaved.

The domain and range of function f can be boolean sets or the set of real numbers. Usually more than one function can fit the given set of learning samples of input-output pairs. It makes the generalization problem harder. For example, learning a specific boolean function from a subset of domain is difficult [25] since several boolean functions over the domain can fit the learning samples. There is little consensus on a criteria to prefer one of the candidate boolean functions over the rest to break the tie. We restrict our attention to the functions over the set of real numbers. We draw upon the notion of simplicity of functions over real numbers to choose one function among the set of possible functions which fit the learning samples. Simplicity is intuitively defined in term of the number of maxima and minima of the function. Simplicity reduces to the notion of degree of polynomials for polynomial functions.

3. Stochastic Backpropagation

The general idea behind our algorithm is to use simulated annealing in the weight space. The weight space is defined by a collection of configurations $W_i = (w_{11}, \ldots w_{mn})$, where various w_{lm} represent the connection weights in the neural network. The simulated annealing procedure searches the weight space for the configuration W_{opt} to minimizes the error-to-fit function $E(W_i)$. The search procedure is based on the Monte Carlo method [29]. Given the current configuration W_i of the network, characterized by the values of its weights, a small, randomly generated, perturbation is applied by a small change in a randomly chosen weight. If the difference in error ΔE between the current configuration W_i and the slightly perturbed one is negative, i.e. if the perturbation results in a lower error of fit, then the process is continued with the new state. If $\Delta E \geq 0$ then the probability of acceptance of the new configuration is given by $exp(-\Delta E/k_B T)$. These occasional transitions to higher error configuration help the search process to get out of local minimas. This acceptance rule for the new configurations is referred to as the *Metropolis Criteria*. Following this criteria, the probability distribution of the configurations approaches the boltzman distribution given by Eq.A.1:

$$Pr[\mathbf{E}=\mathrm{E}] = \frac{1}{\mathrm{Z(T)}} \times \exp(-\frac{\mathrm{E}}{\mathrm{k_B T}}), \qquad \text{(A.1)}$$

where $Z(T)$ is a normalization factor, known as partition function, depending on temperature T and boltzman constant k_B. The factor $exp(-E/k_B T)$ is known as the boltzman factor.

T denotes a control parameter, which is called temperature due to historic reasons. Starting off at a high value, the temperature is decreased slowly during the execution of algorithm. As the temperature decreases, the boltzman distribution concentrates on the configurations with lower error and finally, when the temperature approaches zero, only the minimum error configurations have a non-zero probability of occurring. In this way we get the globally optimal weights for the network minimizing the error of fit with the training examples, provided the maximum temperature is sufficiently high and the cooling is carried out sufficient slowly.

Algorithm Description: One has to define configurations, a cost function and a generation mechanism(or equivalently, a neighborhood structure) before describing the algorithm. We assume that each weight takes discrete values from set $\psi = \{-s\delta,\ldots,-\delta,0,\delta,2\delta,\ldots,s\delta\}$. The restriction of the weights to discrete values does limit ‡ the learning ability of neural networks for most generalization problems. The configurations can now be defined as n-tuple of weights, where n is the number of weights in the network. The configuration space is constructed by allowing each weight to take values from ψ. The cost function is defined by the error between desired outputs and network outputs for the learning examples as shown below:

$$\mathrm{E} = \frac{1}{2}\sum_{j,c}(y_{j,c}-d_{j,c})^2 \qquad \text{(A.2)}$$

Here $y_{j,c}$ refers to the j-th network output for the input from c-th training example, and $d_{j,c}$ refers to the j-th (desired) output from the c-th training example. The indices c and y refers to different outputs of the network and various training examples respectively.

To generate the neighboring configurations, we change one randomly chosen weight element in the configuration by δ. We use uniform probability distribution to chose the weight to be changed, and thus the probability of generating any neighboring configuration from the current configuration, is uniformly distributed over the neighbors.

---

‡ One can always choose large value for *s* and scale the input output data value to a small range to achieve better accuracy.

```
Procedure STOCHASTIC_BACKPROP;
begin
    INITIALIZE;
    M := 0;
    while (not stop_criterion) {system is not ''frozen''}
        repeat
            accept := false;
            BACKPROP ;
            PERTURB(config.i→config.j) ;
            ΔE_ij := (∂E/∂w_lm) ×Δw_lm ; {where w_lm was changed by PERTURB}
            if ΔE_ij ≤ 0 then accept := true
            else if exp(−E(i)/c) > random[0,1] then accept:= true;
            if accept‡ then UPDATE(configuration j);
        until equilibrium_is_approached_sufficiently_closely;

        c_{M+1} := f(c_M);
        M := M + 1;
    end;
end;
```

Fig.3: Stochastic backpropagation algorithm in psuedo-Pascal

A psuedo-Pascal description of the stochastic backpropagation is shown in Fig.3. The INITIALIZE routine assigns default values to all the variables, in particular to current configuration, temperature, and outer_iteration_count M. The loops correspond to simulated annealing in the weight space. The inner loop represents simulated annealing at a fixed value of the control parameter T. It executes until the probability distribution of current configuration being any of the possible configuration, becomes stable. This helps us to achieve boltzman distribution. The outer loop changes the control parameter slowly to the final value near 0. This corresponds to slow cooling to achieve configurations with globally minimum error. The steps inside the innermost loop combine backpropagation with transitions for simulated annealing.. We use the BACKPROP, the backpropagation algorithm, [4] as a subroutine to compute the error derivatives $\partial E/\partial w_{lm}$ with respect to various weights. These derivatives help us to estimate the change in error function, when a particular weight is changed by δ. PERTURB produces a neighboring configuration by changing a randomly chosen weight by δ. The change in error function due to the perturbation is estimated using the error derivatives obtained from the backpropagation algorithm. The new configuration is accepted unconditionally, iff it has lower error than the current one. Otherwise the new configuration is accepted with probability = $\exp(-\Delta E/k_B T)$. Finally the program variables are updated by UPDATE to state of the newly chosen configuration.

---

‡ Note that the acceptance criterion is implemented by drawing random number from a uniform distribution on (0,1) and comparing these with $exp(-E(i)/T)$.

The basic algorithm shown in Fig.3. can be made more efficient, if one changes the function of BACKPROP procedure. We notice that the backpropagation produces the partial derivatives of E with respect to all the weights, whereas we use only one of the derivatives in subsequent computation. It is better to modify the backpropagation algorithm to compute only one derivative, which is required.

Comparison with Existing Algorithms: Some of the existing learning algorithms, e.g. in Hopfield networks[30, 31], are based on memorizing the learning examples accurately. These cannot be used for prediction in generalization problems. Two of the more flexible learning methods include backpropagation [9, 32] and boltzman machine learning [33]. Both are iterative algorithms based on gradient descent on the error surface.

In *backpropagation* the error of fit for a given set of weights is defined by Eq. *B.1* where $y_{j,c}$ is the actual state of unit j in input-output training example c and $d_{j,c}$ is the desired state. Backpropagation algorithm computes the gradient of error with respect to each weight. A hidden unit, j, in layer J affects the error via its effects on the units, k, in the next layer K. So the derivative of the error $\partial E/\partial y_j$ is given by Eq. *B.2* where the index c has been suppressed for clarity.

$$E = \frac{1}{2}\sum_{j,c}(y_{j,c} - d_{j,c})^2 \qquad \text{(B.1)}$$

$$\frac{\partial E}{\partial y_j} = \sum_k \frac{\partial E}{\partial y_k}\frac{dy_k}{dx_k}\frac{\partial x_k}{\partial y_j} \qquad \text{(B.2)}$$

The weights are then changed in a direction to reduce the error in the output. One may change the weights simultaneously to avoid conflicting local weight adjustments.

The boltzman machine is stochastic in nature and the aim of learning the weights is to achieve a probability distribution of input-output mapping. The boltzman machine learning [5] is based on a series of simulated annealing on the state space of network. State space for the network can be characterized by defining the state of the network. The state of a node is described by its output and the state of the network is a n-tuple vector with one component for each node. The learning algorithm aims to achieve a certain probability distribution over the states, and does a gradient descent to minimize the error in probability distribution. *Our use of simulated annealing in weight space is quite different from the boltzman machine [33], where simulated annealing is carried out in the state space of the network.*

These learning methods work only when the cost/error surface is concave. Since these algorithms are based on a simple heuristic of gradient descent, these can get stuck in local minima. Furthermore these can get stuck at plateaus, where the gradient is very small, as shown in Fig.4. These algorithms cannot guarantee the optimality of discovered weights. The learning problem is NP-complete in general [34] and remains NP-complete under several restrictions. it is not surprising that heuristic learning algorithms like backpropagation do not always work, and cannot be trusted to find the globally optimal weights for generalization problems.

There are two ways of approaching NP-complete problems: (a) approximation methods [35], and (b) stochastic enumeration method such as simulated annealing [36]. Since it is difficult to formulate a general approximation method for neural network learning, we use the *simulated annealing* method. We extend the backpropagation algorithm with stochastic weight changes for learning the weights. The algorithm has convergence properties and can achieve globally optimal weights for a simple network for generalization. The implementation and performance studies show that the algorithm performs well for many generalization problems.

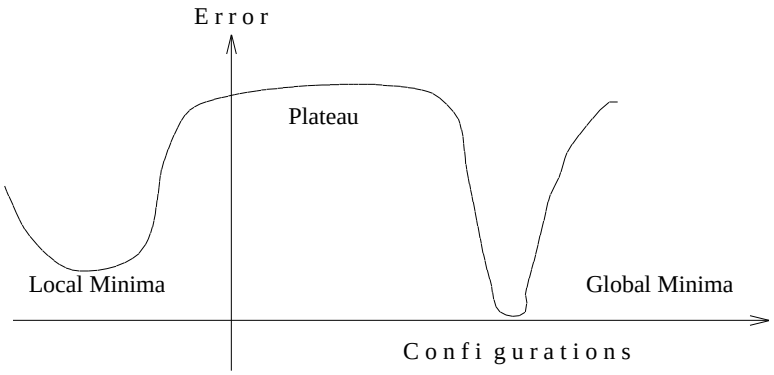

Figure 4: Two Bad Cases for Gradient Descent

4. Modeling and Analysis of Stochastic Backpropagation

Given a neighborhood structure, stochastic backpropagation can be viewed as an algorithm that continuously attempts to transform the current configuration into one of its neighbors. This mechanisms can mathematically be described by means of a *Markov Chain*: a sequence of trials, where the outcome of each trial depends only on the outcome of the previous one [37]. In the case of stochastic backpropagation, the trials correspond to transitions and it is clear that the outcome of a transition depends only on the outcome of the previous one (i.e. the current configuration).

A Markov chain is described by means of a set of conditional probabilities $P_{ij}(k-1,k)$ for each pair of outcomes (i,j); $P_{ij}(k-1,k)$ is the probability that the outcome of the k-th trial is j, given that the outcome of the k - 1-th trial is i. Let $a_i(k)$ denote the probability of outcome i at the k-th trial, then $a_i(k)$ is obtained by solving the recursion:

$$a_i(k) = \sum_l a_i(k-1).P_{li}(k-1,k),\ \mathrm{k}=1,2..,\quad \text{(C.1)}$$

where the sum is taken over all possible outcomes. Hereinafter, **X**(k) denotes the outcome of the k-th trial. Hence

$$P_{ij}(k-1,k) = Pr(\mathbf{X}(\mathrm{k}) = \mathrm{j} \,|\, \mathbf{X}(\mathrm{k}-1) = \mathrm{i}) \qquad \text{(C.2)}$$

$$\text{and} \quad a_i(k) = Pr(\mathbf{X}(\mathrm{k}) = \mathrm{i}) \qquad \text{(C.3)}$$

If the conditional probabilities do not depend on k, the corresponding Markov chain is called *homogeneous*, otherwise it is called *inhomogeneous*.

In case of stochastic backpropagation, the conditional probability $P_{ij}(k-1,k)$ denotes the probability that k-th transition is a transition from configuration i to configuration j. Thus **X**(k) is the configuration obtained after k transitions. In view of this, $P_{ij}(k-1,k)$ is called the transition probability and the $|R|\times|R|$ matrix P(k-1,k) the transition matrix. Here $|R|$ denotes the size of the configuration space.

The transition probabilities depend on the value of the control parameter T (temperature). Thus if T is kept constant, the corresponding Markov chain is homogeneous and its transition matrix P = P(T) can be defined as

$$P_{ij}(T) = G_{ij}(T)\, A_{ij}(T) \qquad forall j \neq i \qquad \text{(C.4)}$$

$$P_{ij}(T) = 1 - \sum_{l=1, l\neq i}^{|R|} G_{il}(T)\, A_{il}(T) \, forall j = i$$

i.e., each transition probability is defined as the product of the following two conditional probabilities: the generation probability $G_{ij}(T)$ of generating configuration j from configuration i, and the acceptance probability $A_{ij}(T)$ of

accepting configuration j, once it has been generated from configuration i. The corresponding matrices G(T) and A(T) are called the *generation and acceptance matrices*, respectively. As a result of the definition in Eq.C.4, P(T) is a stochastic matrix, i.e. $forall\ i: \sum_j P_{ij}(T) = 1$. G(T) is represented by a uniform distribution over the neighborhoods since transitions are implemented by choosing at random a neighboring configuration j from the current configuration i. A(T) is computed from the Metropolis criteria, i.e. $\min(1, \exp(-\Delta E/k_B T))$.

The stochastic backpropagation algorithm attains a global minimum, if after a (possibly large) number of transitions, say K, the following relation hold:

$$\Pr\{\mathbf{X}(k) \in R_{opt}\} = 1 \qquad (C.5)$$

where R_{opt} is the set of globally optimal configurations with minimum error of fit. It can be shown that Eq.C.5 holds asymptotically (i.e. $\lim_{k\to\infty} \Pr\{\mathbf{X}(k) \in R_{opt}\} = 1$), if

1. certain conditions on matrix $A(T_l)$ and $G(T_l)$ are satisfied
2. $\lim_{k\to\infty} T_k = 0$
3. under certain additional conditions on matrix $A(T_k)$, the rate of convergence of the sequence $\{T_k\}$ is not faster than $O(|log\ k|^{-1})$.

The proof is carried out in three steps: (a) showing the existence of stationary distribution for homogeneous Markov chains, (b) showing the convergence of inner-loop of stochastic backpropagation to the stationary distribution, and (c) showing that the stationary distribution for the final ''frozen system'', has non-zero probabilities on the optimal configurations only. The proof for last step is contingent on the cooling rate and provides us with a bound on the rate.

4.1. Existence of Stationary Distribution

The following theorem establishes the existence of the stationary distribution.

Theorem 1 (Feller, [37]): The stationary distribution **q** of a finite homogeneous Markov chain exists if the Markov chain is irreducible and aperiodic. Furthermore, the vector **q** is uniquely determined by the following equation:

$$forall\ i: q_i > 0, \sum_i q_i = 1, \quad (C.6)$$

$$forall\ i: q_i = \sum_j q_i P_{ij}. \qquad (C.7)$$

We note that **q** is the left eigenvector of the matrix P with eigenvalue 1.

A Markov chain is irreducible, if and only if for all pairs of configurations (i,j) there is a positive probability of reaching j from i in a finite number of transitions, i.e.

$$forall\ i,j \quad exists\ n: 1 \le n < \infty, \text{ and } P^n{}_{ij} > 0; \quad (C.8)$$

Markov chain is aperiodic, if and only if for all configurations $i \in R$, the greatest common divisor of all integers $n \ge 1$, such that

$$P_{ii}{}^n > 0 \qquad (C.9)$$

is equal to 1.

In the case of stochastic backpropagation, the matrix P is defined by Eq. C.4. Since the definition of A guarantee that forall i,j,c > 0 : $A_{ij}(T) > 0$, it is sufficient for irreducibility to check that the Markov chain induced by G(T) is irreducible [38],

i.e. $forall\ i,j \in R,\ exists\ p \ge 1,$ and $l_0, l_1, ..., l_p \in R$ ($l_0 = i$ and $l_p = j$):

$$G_{l_k l_{k+1}}(T) > 0,\ k=0,1,...,p-1. \qquad \text{(C.10)}$$

To establish aperiodicity, one uses the fact that an irreducible Markov chain is aperiodic if the following condition is satisfied [38]:

$$forall\ T>0,\ exists\ i_T \in R: P_{i_T i_T}(T)>0. \text{ (C.11)}$$

Thus for aperiodicity it is sufficient to assume that

$$forall\ T > 0,\ exists\ i_T, j_T \in R: A_{i_T j_T}(T) < 1, \qquad \text{(C.12)}$$

Using the inequality of Eq. C.12 and the fact that $forall\ i,j : A_{ij} \le 1$, we can prove the following:

$$\begin{aligned} & \sum_{l=1, l\neq i_T}^{|R|} A_{i_T l}(T)\, G_{i_T l}(T) \\ = & \sum_{l=1, l\neq i_T, j_T}^{|R|} A_{i_T l}(T)\, G_{i_T l}(T) + A_{i_T j_T}(T)\, G_{i_T j_T}(T) \\ < & \sum_{l=1, l\neq i_T, j_T}^{|R|} G_{i_T l}(T) + G_{i_T j_T}(T) \\ = & \sum_{l=1, l\neq i_T}^{|R|} G_{i_T l}(T) \le \sum_{l=1}^{|R|} G_{i_T l}(T) = 1. \end{aligned} \qquad \text{(C.13)}$$

Thus $P_{i_T i_T} = 1 - \sum_{l=1, l\neq i_T}^{|R|} A_{i_T l}(T)\, G_{i_T l}(T) > 0$ (C.14)

and thus, Eq. C.11 holds for i=i_T.

Summarizing we have the following result. The homogeneous Markov chain with conditional probabilities given by Eq. C.4. has a stationary distribution if the matrices A(T) and G(T) satisfy Eqs. C.10 and C.12, respectively. Note that in the stochastic backpropagation the acceptance probabilities are defined by

$$A_{ij}(T) = min(1, exp(-(E(j) - E(i))/T)) \text{ (C.15)}$$

and hence Eq. C.12 is always satisfied by setting, forall T > 0, $i_T \in R_{opt}$, $j_T \neg\in R_{opt}$.

4.2. Convergence of the Stationary Distribution

We now impose further conditions on the matrices A(T) and G(T) to ensure convergence of **q**(T) to the distribution π, as given by Eq. C.5.

Theorem 2 [38]: if the two argument function $\psi(E(i) - E_{opt}, T)$ is taken as $A_{i_0,i}(T)$ (for an arbitrary configuration $i_0 \in R_{opt}$ and if G(T) does not depend on T, then the stationary distribution **q**(T) is given by

$$forall\ i \in R : q_i(T) = \frac{A_{i_0,i}(T)}{\sum_{j\in R} A_{i_0,j}(T)}, \qquad \text{(C.16)}$$

provided the matrices A(T) and G satisfy following conditions:

(a1) $forall i,j \in R: G_{ij} = G_{ji}$; (C.17)

(a2) *forall* i,j,k ∈ R, E(i) ≤E(j) ≤E(k) → A_{ij}(T) = A_{ij}(T)A_{jk}(T); (C.18)

(a3) *forall* i,j ε R: E(i) ≥ E(j) → A_{ij}(T) = 1; (C.19)

(a4) *forall* i,j ε R, c > 0: E(i) < E(j) 0 < A_{ij}(T) < 1, (C.20)

The proof of this theorem is discussed elsewhere [39].

It is implicitly assumed that the acceptance probabilities depend only on the cost values of the configuration and not on the configurations itself. Hence, $A_{i_0,i}$(T) does not depend on the particular choice for i_0, since

$$\text{forall } i_0 \in R_{opt} : E(i_0) = E_{opt}.$$

To ensure that $\lim_{c \to 0} \mathbf{q}(T) = \pi$ of Eq. C.5, the following condition is sufficient [38]:

(a5) forall i,j $\in R$: $E(i) < E(j) \rightarrow \lim_{T \to 0} A_{ij}(T) = 0$, (C.21)

Thus the conditions (C.19)-(C.23) guarantee the convergence. It can be easily checked that the matrices G(T) and A(T) for stochastic backpropagation meet all these conditions.

4.3. Cooling rate

Under certain conditions on matrices A(T) and G(T), the stochastic backpropagation algorithm converges to a global minimum with probability 1 if for each value of T_l of the control parameter(l = 0,1,2...), the corresponding Markov chain is of infinite length and if the T_l eventually converges too 0 for $l \rightarrow \infty$, i.e. the validity of the following equation is shown: $\lim_{c \to 0} (\lim_{k \to \infty} Pr(X(k)=i)) = \lim_{c \to 0} q_i(T) = \begin{cases} |R_{opt}|^{-1} \text{ if } i \in R_{opt} \\ 0 \quad elsewhere \end{cases}$

However the cooling rate or the constraint on the sequence T_l of the control parameter(l = 0,1,2...), has to satisfy certain properties to assure convergence to the globally optimal configurations. In particular, if T_k is of the form

$$T_k = \frac{\Gamma}{\log k}$$

then one can guarantee † the convergence to the globally optimal configurations [40].

4.4. Convergence Results

We have listed the conditions under which simulated annealing converges to globally optimal configurations. Our formulation of stochastic backpropagation uses similar acceptance probabilities, generation probabilities, and cooling schedule, as the simulated annealing algorithm [38]. That is we use Metropolis criteria as acceptance criteria. We have a configuration generating mechanism, which has uniform distribution over neighbors. The generation mechanism across two neighboring configuration is symmetric. One can generate any arbitrary configuration from a given configuration in finite number of steps. Thus we satisfy the conditions of Theorem 1 and 2 and are guaranteed convergence to global minima. We follow a cooling schedule of $T_n \leq \frac{\Gamma}{\log n}$ to satisfy the conditions on cooling rate. This guarantees the convergence to global minima provided Γ is greater than the depth of any local minima.

5. Implementation

We have carried out a complete implementation of stochastic backpropagation and conducted validation studies. We implemented the algorithm on a unix platform on a sequential machine (e.g. SUN 3/60). Since generalization often takes large amounts of computation, we plan to reimplement the algorithm on a vector processor (e.g. Cray XMP) for speed-up.

The current prototype is based on the source code of a public domain software implementing backpropagation algorithm[41]. We studied the software to reuse pertinent modules to implement stochastic backpropagation. The implementation comprises of 5000 lines of C code. It required approximately seven man months to design, code and debug. A large fraction of the effort was directed towards reading and understanding of the backpropagation software for reuse. The effort was rewarded in reduced time of designing, coding and debugging. The code was

† Provided Γ >= D, the maximal depth of any local minima in the error surface.

verified by walkthrough method and by extensive usage. The prototype has been used in seminar courses and research.

The backpropagation package [41] has three modules: user interface, learning module and testing module. **User interface** implements the commands by associating the commands to internal functions via a table. The commands allow users to examine, and modify the state of the software, choose options to specify the type and speed of computing and displays. The **learning module** implements the backpropagation algorithm by computing error derivatives and weight adjustments to tune the weights iteratively for training. It repeats the weight adjustments for fixed number of times or till the total squared error reaches a value set by the user. The learning algorithm provides option of adjusting weights after examining each pattern or after examining all the patterns. The **testing module** computes outputs for given inputs to the neural network. It also computes the total squared error between the output produced by the network and the desired output specified for the input pattern.

We augmented the user interface module by adding commands to examine and modify the parameters of stochastic backpropagation. The routines to process the commands were installed in the table associating commands to the processing routines. A command to enable the user to choose between the alternative learning algorithms was also added.

We implemented the stochastic backpropagation in C with a simplified cooling schedule. The cooling schedule is based on expotential cooling $T_k = \lambda * T_{k-1}$ for simplicity and efficiency. This cooling schedule has been used in many applications [42]. The main routine in the learning module, namely trial(), was modified to adjust weight based weight by randomly choosing a neighbor and accepting it by metropolis criteria. Testing module was not altered for the implementation.

6. Validation

We evaluated stochastic backpropagation learning algorithm for generalization to two types of functions: (a) monotonic functions , and (b) non monotonic functions.

The experimental setup consisted of four modules: data set generator, neural network simulator, data collection and data analysis as shown in Fig 5. The data set generator module uses four functions : linear, quadratic, logarithmic and trigonometric as shown in table 1. The neural network simulator implements alternative learning algorithms of backpropagation and stochastic backpropagation. It takes the network configuration and data sets. The module simulates learning algorithm and produces the outputs as well as the weights. The data collection module comprises of a set of routines to sample the state of neural network simulator. It can periodically (say every 100 epochs of learning) sample the weights or collect them at the termination of learning. The data analysis module produces graphs, and statistics.

The algorithms are monitored during the learning phase as well as during the testing phase. The performance of algorithm during learning phase is measured by the total square error on the learning set of input-output pairs. The performance of algorithm during testing phase is measured by the per pattern error on a new set of input-output pairs, which is distinct from learning set. The behavior of alternative algorithms during learning are shown in Figures 6 and 7. Figure 6 shows the change in total square error along with learning steps for monotonic functions, i.e. linear, logarithmic and quadratic functions. Stochastic backpropagation and backpropagation yield comparable total square error. We tested the trained network with an independent set of samples. Stochastic backpropagation trained network yielded 1.7% error per pattern. Backpropagation trained network yielded 0.9% error per pattern. Both networks predict the outputs for all samples within 5% of desired output. Figure 7 shows the change in total square error with epochs of learning for non-monotonic function, i.e. trigonometric function. Stochastic backpropagation

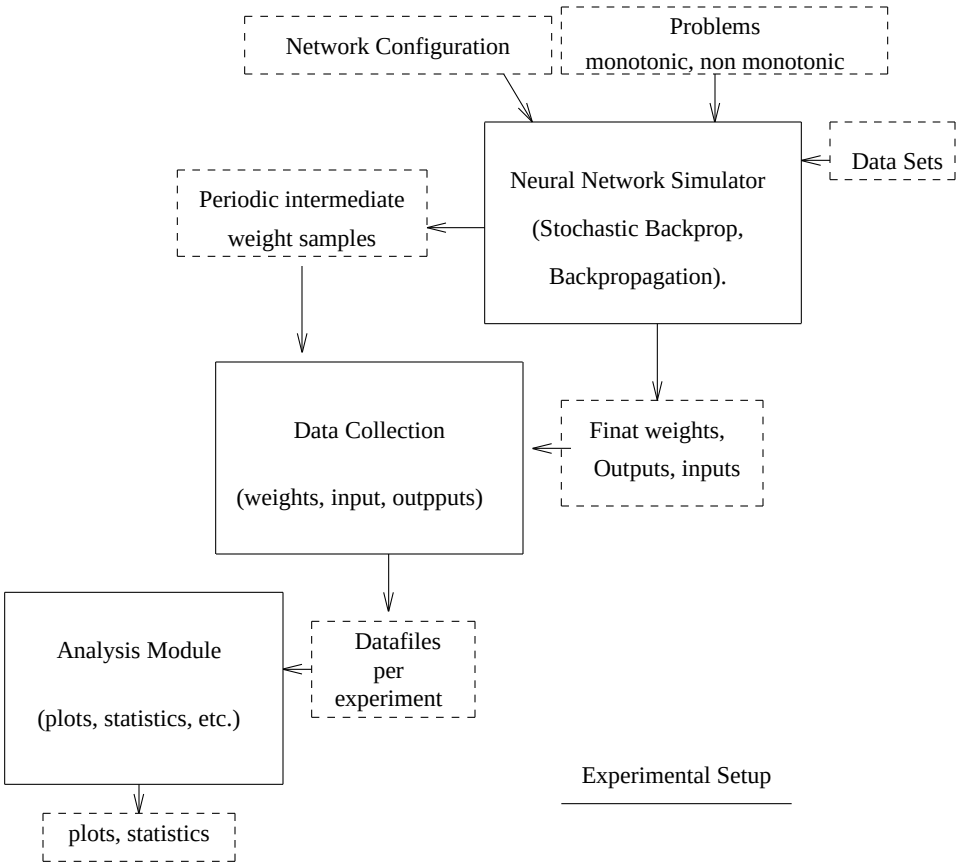

Fig. 5.: Experiment Design for the Performance Comparison Study

f1(x,y)	=	x + 2y + 3
f2(x,y)	=	0.5 Sqr(x) + 0.2 sqr(y) + 9.0
f3(x,y)	=	3 log(x) + log(y) + 2.0
f4(x,y)	=	2 sin(x) + cos(y) + 4.0

Table 1: Functions for controlled study

yields better total square error during learning as well as during testing. The network trained with backpropagation network yields per pattern error of 15%, predicting the output for 14 samples out of 50 within 5% of the desired output. The network trained with stochastic backpropagation yields per pattern error of 11%. It predicts the output for 21 samples out of 50 within 5% of the desired output.

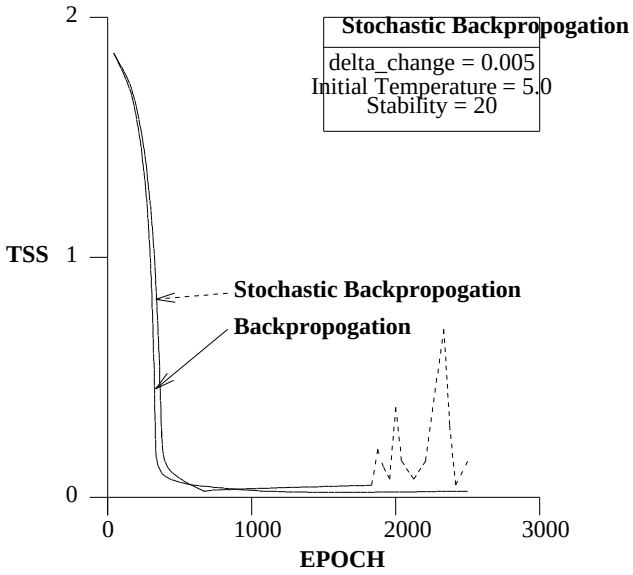

Figure 6: Learning monotonic functions

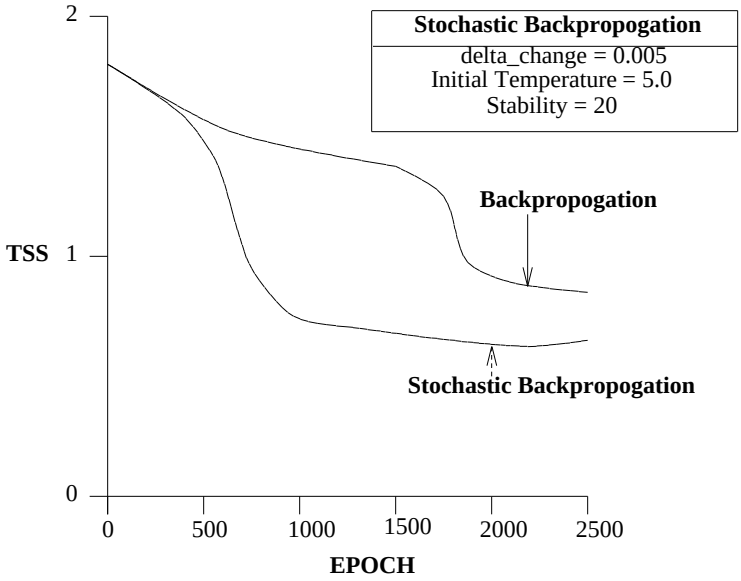

Figure 7: Learning non-monotonic functions

7. Conclusions

Stochastic backpropagation provides a feasible learning algorithm for generalization problems. It reduces the error of fit with the training examples, which is critical for generalization problem. Stochastic backpropagation performs well on monotonic functions using simple networks with fewer hidden nodes. It performs better than backpropagation algorithm on non-monotonic functions. The stochastic backpropagation learning algorithm has theoretical property of convergence. It also provides a stochastic guarantee of finding the optimal weights. However, our experiments did not confirm it. One needs to further tune the parameters of the implementation of stochastic backpropagation to get better results.

8. Acknowledgements

We acknowledge useful help from the CS 8199 class of spring 1991, UROP at the University of Minnesota, and the backpropagation package[41]

9. References

1. Sejnowski, T.J. and Wasserman,P.D., Neural Networks, Part 2, *IEEE Expert Magazine*, **3**(1)(spring 1988.). John Hopkins Univ., Baltimore, MD.
2. K. Yamada, H. Kami, J. Tsukumo, and T. Temma, Handwritten numeral recognition by multi-layered neural network with improved learning algorithm, *Proc. Intl Conf. on Neural Networks*, IEEE and ICNN, (June 1989).
3. M. A. Fanty, Learning in Neural Networks, *Ph.D. Thesis TR252*, C.S.Dept., Univ. of Rochester, (1988).
4. D.E.Rumelhart, G.E.Hinton, and R.J.Williams, Learning Internal Representations by Back Propagating Errors, *Nature* **323** pp. 533-536 (1986).
5. D. H.Ackley, G.E.Hinton, and T.J. Sejnowski, A Learning Algorithm for Boltzman Machine, *Cognitive Science* **9** pp. 147-169 (1985).
6. S. E. Hampsin and D. J. Volper, Linear Function Neurons: Structure and Training, *Biological Cybernetics* **53** pp. 203-217 (1986).
7. G. Carpenter and S. Grossberg, The ART of adaptive pattern recognition by a self-organizing neural network, *Computer* **21** pp. 77-88 (1988).
8. D. E. Rumelhart, Brain Style Computation: Learning and Generalization, *An introduction to neural and electronic networks*, Acadmemic Press, (1990).
9. G.E.Hinton, Learning to Recognize Shapes in a Parallel Network, *Proc. 1986 Fyssen Conference*, (1987). Oxford University Press, Oxford.
10. G.E. Hinton, Learning Translation Invariant Recognition in Massively Parallel Network, *Lecture Notes in Computer Science # 258*, pp. 1-13 Springer Verlag, (1987).
11. S. Shekhar and S. Dutta, Bond Rating: A Non-Conservative Application of Neural Network, *Proc. IEEE Intl. Conf. on Neural Networks*, (July, 1988). San Diego.
12. R. Eckmiller, The design of intelligent robot as a federation of geometric mchines, *An introduction to neural and electronic networks*, Acadmemic Press, (1990).
13. Hecht Nielsen, Neurocomputer Applications, *Neural Computers (Ed R. Eckmiller & Malsburg)* **F41**Springer-Verlag Berlin Heidelberg, (1988).
14. A. Irani, J. P. Matts, J.M.Long, and J. R. Slagle, *Using Artificial Neural Nets for Statistical Discovery: Observations after using Back-Propagation, Expert Systems, and Multiple-Linear Regression on Clinical Trial Data*, University of Minnesota Technical Report, (1989.).
15. P. Werbs, Beyond Regression: New Tools for Prediction and Analysis in the Behaviorial Sciences, *Ph.D. thesis*, (Nov. 1974). Applied math., Harvard University
16. Gallant,S.I., Connectionist Expert Systems, *Comm. of the ACM*, **31**(2)(Feb. 1988.).
17. Parunak,H.V., Material Handling: A Conservative Domain for Neural Connectivity and Propagation, *Proc. AAAI Conf.*, pp. 307-311. (1987).
18. M. Minsky and S. Papert, *Perceptrons*, MIT Press (1968). reprinted in 1988
19. A. K. Kolmogorov, On the representation of continuous functions of several variablesby superposition of continuous functions of one variable and addition, *Doklady Akademii Nauk* **114** pp. 369-373 SSSR, (1957).
20. S. Patarnello and P. Carnevali, Meaning of Generalization (ch. 4 of Learning Capabilities of Boolean Networks), *Neural Computing Architectures (Ed. I. Aleksander)*, The MIT Press, (1989).
21. M. Arai, Mapping Abilities of Three Layered Neural Networks, *Proc. Intl. Jt. Conf. on Neural Network*, IEEE and INNS, (June 1989).

22. A. Blum and R. L. Rivest, Training a 3-node neural net is NP-complete, *Advances in Neural Information Processing Systems*, pp. 494-501 Morgan Kaufmann, (1989).

23. S. Judd, On Complexity of Loading Shallow Networks, *Journal of Complexity* **4** pp. 177-182 Academic Press, (1988).

24. J. Stephen Judd, *Neural Network Design and the Complexity of Learning*, MIT Press (1990).

25. S. Ahmad and G. Tesauro, *Scaling and Generalization in Neural Networks: A case study*, Intl Conf. Neural Info. Processing Systems (1988).

26. C.V. Ramamoorthy and S. Shekhar, Stochastic Backproagation: A Learning Algorithm for Generalization Problems, *Proc. IEEE COMPSAC Conf.*, (1989.). Orlando, FL.

27. J. Stephen Judd, Memorization and Generalization (Ch. 7), *Neural Network Design and the Complexity of Learning*, MIT Press, (1990).

28. J. Sietsma and R. J. F. Dow, Creating Artificial Neural Networks That Generalize, *Neural Networks* **4** pp. 67-69 Pergamon Press, (1991).

29. N. A. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller, Equation of State Calculations by Fast Computing Machines, *J. of Chem. Physics* **21** pp. 1087-1092 (1953).

30. J. J. Hopfield, Neural networks and physical systems with emergent collective computing abilities, *Proc. of the Natl Academy of Sciences USA* **79**(8) pp. 2554-2558 (1982).

31. J. J. Hopfield and D. W. Tank, Neural computation of decisions in optimization problems, *Biological Cybernetics* **52**(3) pp. 141-152 (1985).

32. D.E.Rumelhart and J.LMcClelland, Parallel Distributed Processing: Explorations in the Microstructure of Cognition, *Bradford Books, Cambridge, MA.* **1**(1986).

33. G.E. Hinton, T.J. Sejnowski, and D.H. Ackley, Boltzman Machines: Constraint Stisfaction Machines That Learn, *Tech.Rep. CMU-CS-84-119*, Carnegie Mellon University, (1984).

34. S. Judd, Complexity of Connectionist Learning with Various Node Functions, *Tech. Rep. 87-60*, Univ. of Mass., Amherst, MA, (1987).

35. M.R.Garey and D.S.Johnson, Computers and Interactability: A Guide to the Theory of NP-Completeness, *Freeman*, (1979).

36. S. Kirkpatrick, Optimization by simulated annealing, *Science* **220** pp. 671-680 (May 13, 1983).

37. W. Feller, *An Introduction to Probability Theory and Applications*, Wiley, New York 1950.

38. F. Romeo and A.L Sangiovanni-Vincentelli, Probabilistic Hill Climbing Algorithms: Properties and Applications, *Proc. 1985 Chapel Hill Conf. on VLSI*, pp. 393-417 (May 1985).

39. P.J.M. van Laarhoven, *Simulated Annealing: Theory and Applications*, D. Reidel Publishing Co., Boston 1987.

40. B. Hajek, Cooling Schedules for Optimal Annealing, *Math. of Operations Research (submitted)*, (1986).

41. L. McClelland and D. E. Rumelhart, *Explorations in Parallel Distributed Processing -- A Handbook of Models, Programs, and Exercises*, Bradford MIT Press (1989).

42. E.H.L.Aarts and J.H.M. Korst, *Simulated Annealing and Boltzman Machines*, John Wiley & Sons (1989).