The Scheduling Problem in Learning From Hints

Thesis by
Zehra Cataltepe

In Partial Fulfillment of the Requirements
for the Degree of
Master of Science

California Institute of Technology
Pasadena, California

1994
(submitted 26 May 1994)

Caltech-CS-TR-94-09

Copyright © Zehra Cataltepe, 1994
All Rights Reserved

Abstract

Any information about the function to be learned is called a *hint*. *Learning from hints* is a generalization of learning from examples. In this paradigm, hints are expressed by their examples and then taught to a learning-from-examples system. In general, using other hints in addition to the examples of the function, improves the generalization performance.

The scheduling problem in learning from hints is deciding which hint to teach at which time during training. Over- or under-emphasizing a hint may render it useless, making scheduling very important. Fixed and adaptive schedules are two types of schedules that are discussed.

Adaptive minimization is a general adaptive schedule that uses an estimate of generalization error in terms of errors on hints. When such an estimate is available, it can also be optimized by means of directly descending on it. An estimate may be used to decide on when to stop training, too.

A method to find an estimate incorporating the errors on invariance hints, and simulation results on this estimate, are presented. Two computer programs that provide a learning-from-hints environment and improvements on them are discussed.

Acknowledgments

Many thanks to my academic and research advisor, Yaser S. Abu-Mostafa, for his guidance, support and advice.

Thanks to Eric T. Bax, Bahadir Erimli, and Ayhan Irfanoglu, for proof-reading and suggestions. Any overseen mistakes are solely mine.

Thanks to the members of the *Learning Systems Group* – Eric T. Bax, Joe Sill, and Xubo Song – for valuable discussions.

And finally, thanks to my mother and father, my beloved husband Tanju, and my brothers Ahmet and Rifat, to all my friends, and whomever I forgot to thank.

This research was supported by the AFOSR under grant number F49620-92-J-0398.

Contents

List of Figures

This page would be left blank intentionally.
But it is not blank now!

1 Introduction

1.1 Learning, Hints, and Scheduling

Solving problems like *"What is the inverse of this matrix?"*, or *"What is π computed to* 10^{-10} *accuracy?"* by means of direct programming is a reasonable and feasible approach. However, for unstructured problems, such as *"Is this a picture of a tree?"*, or *"Will the stocks go up or down tomorrow?"*, direct programming may not be a good solution, if not impossible. Let's formalize problems and questions like that as *functions f that we want to implement.* A method to implement f, other than direct programming, is to search for it among a set of candidate functions G. For example, for a neural network [17] with a fixed architecture, G would be the set of all functions that can be obtained by setting the weights of the network to different values. If there is a $g \in G$ such that $g = f$ then f is *implementable* in G. If f is implementable in G, one would like to find the function $g = f$, or otherwise a g which agrees with f the most. Learning from examples is the learning method which uses examples of f in order to estimate how close f and g are and searches for $g \in G_0 \subseteq G$, where G_0 contains the elements $g \in G$ closest to the examples of f.

For a fixed f and set of examples of it, the agreement between examples of f and $g \in G$ can be defined in terms of an error function $E_0(g)$. Then learning from examples becomes an optimization problem of the form $\min_{g \in G} E_0(g)$. Gradient descent and its implementation for feed-forward neural networks, backpropagation [20], is one of the approaches to solve this problem.

Sometimes the number of examples of f is not enough to pinpoint f among $g \in G_0$. If there are few functions in G_0, then the probability that $g \in G_0$ agrees with f at points beyond the training set (i.e. g *generalizes* well) is high. Hence, without eliminating f from G_0, one would like to restrict G_0 as much as possible. Any additional information to restrict G_0 would be useful in terms of probability of generalization of g. Fortunately, there is usually more known about f than only a set of examples. For instance, a picture that has a tree in it still has a tree even after it is rotated, scaled, or reflected [13, 19]. If these constraints (being invariant under rotation, scaling, or reflection) are imposed on G_0, then the set of functions obtained, $G_1 \subseteq G_0 \subseteq G$, is more likely to generalize than G_0. In figure 1, the positive effect of evenness and cyclic shift invariance hints on generalization is shown. The algorithm achieves a smaller generalization

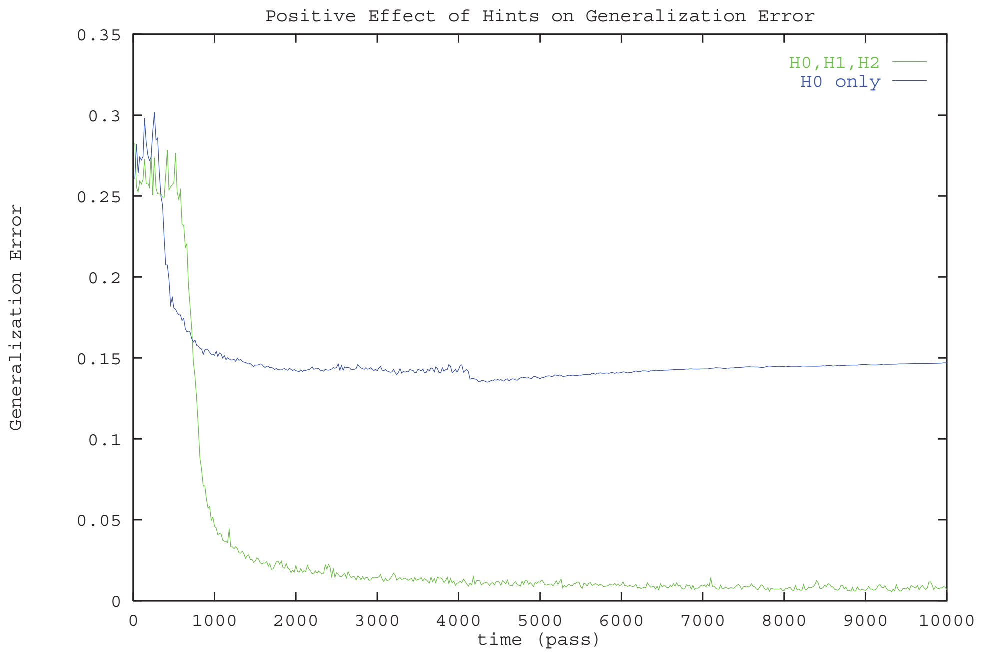

Figure 1: Generalization improves when in addition to H_0 (examples of f), H_1(cyclic shift) and H_2 (evenness) hints are taught.

error as the system learns f with the help of additional hints.

Any information about f (including examples of f) is called a *hint* [1]. *Learning from hints* [1], expresses each hint by examples, and restricts the search space in such a way that the agreement with the examples of all hints becomes better as the search for f proceeds.

If there are enough examples of f [5] or if the additional hints are directly implementable on the system G without excluding f [19], then expressing hints by their examples may not be a preferable option. However, if the number of examples of f is small, or if hints are not directly implementable on the system, or if it would be preferable to teach additional hints up to some degree instead of all the way at the expense of the examples of f, then learning from hints is a good option. The ability to use any learning-from-examples algorithm without any modifications either to the algorithm or to the system is one of the advantages of this method.

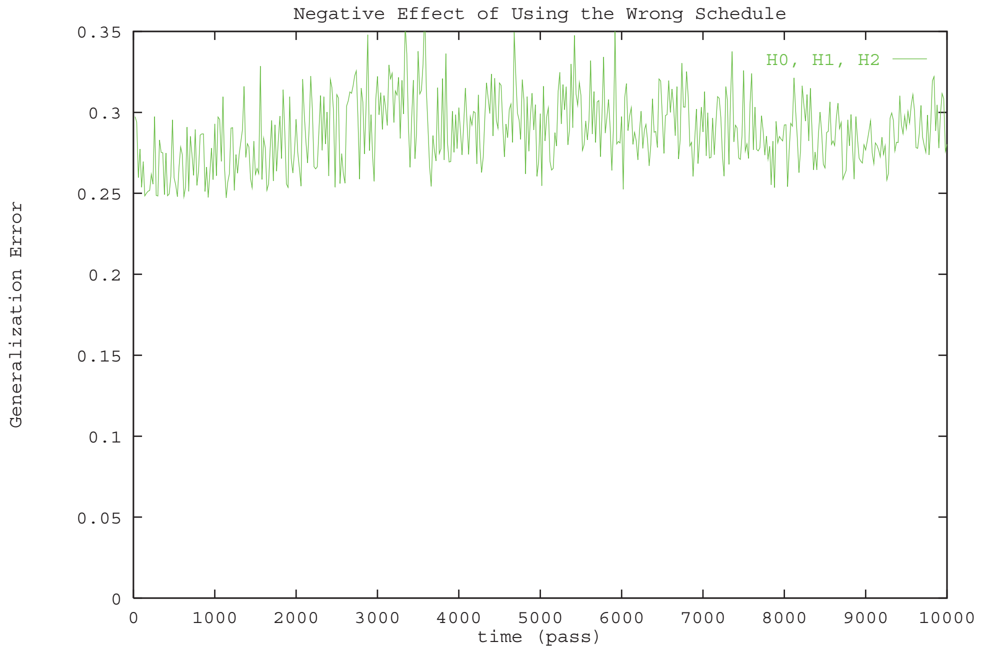

Figure 2: Oscillating generalization error when H1(cyclic shift) and H2 (evenness) hints are overemphasized.

When there is more than one hint, the question of *when to teach each hint* arises. We call this problem the *scheduling problem* in learning from hints. The scheduling problem is an important problem because overlearning a hint at the expense of other hints, or underlearning it, could make hints non-beneficial. For example, in figure 2, the shift invariance hint and the evenness invariance hint are overemphasized during training, and the system oscillates without being able to converge to a small generalization error.

We give two classes of schedules: **(i)** Fixed Schedules, and **(ii)** Adaptive Schedules. Fixed schedules determine which hint will be taught at a pass without using any knowledge about how training proceeds. Adaptive schedules use information during run time and decide on which hint to teach according to the current state of training. Rotation schedule is an example of the fixed schedules, and maximum error schedule is an example of the adaptive schedules that are discussed. A generalization of maximum error schedule is *adaptive minimization* [3]. Adaptive minimization requires an estimate of the generalization error, in terms of the errors on hints. Having this estimate is useful in two aspects: **(i)** descending on it would mean descending on generalization error according to the information given by the hints, and **(ii)** experimental evidence [7] suggests that the estimate may be used to determine when to stop training.

1.2 Definitions and Notation

Assume that the function to be implemented using a learning-from-examples algorithm is $f : X \rightarrow Y$, with a probability distribution P_X on the input space X. In this thesis $X = \mathcal{R}^n$ and $Y = [0, 1]$ are used. The function being implemented by a given learning-from-examples system is $g : X \rightarrow Y$, where X and Y are usually the same as in the definition of f. The set of all possible g's forms the search space, G. We use feed-forward neural networks with a given architecture as our model. In this case, G is the set of all possible functions, g, that can be obtained by setting the weights of the network.

The given examples of f are divided into two disjoint parts: training and validation sets [16]. The examples in the training set are used for teaching f to the network, and the examples in the validation set are used to approximate the generalization error, so as to determine when to stop training.

Training set examples form a special hint, called the *examples hint*, H_0. If there are N_0 examples in the training set, then H_0 is represented by the set of pairs $\{(x_i, f(x_i)) : 1 \leq i \leq N_0\}$ picked according to P_X. The error on the hint H_0 is defined as:

$$E_0(g) = \frac{1}{N_0} \sum_{i=1}^{N_0} (f(x_i) - g(x_i))^2. \tag{1}$$

For a fixed f and H_0, $E_0(.)$ is a function of g only.

The validation error is defined in the same manner as:

$$E_t(g) = \frac{1}{N_t} \sum_{i=1}^{N_t} (f(x_i) - g(x_i))^2 \tag{2}$$

where N_t is the number of validation set examples.

The generalizaton (or learning) error of g on f is defined as:

$$E(g) = \mathcal{E}_X((f(x) - g(x))^2) \tag{3}$$

where $\mathcal{E}_X(.)$ denotes the expected value with respect to the probability distribution P_X of the input space X. By the law of large numbers, E_t is very close in probability to E for large N_t. Throughout this thesis, we assume that N_t is large and treat E_t as the generalization error E.

In general, an example of a hint H_m is defined by an objective (error) function on some input vectors $x_0, x_1, \ldots, x_{K_m}$, and it is denoted as $e_m(g, x_0, x_1, \ldots, x_{K_m})$. The relationship between $x_0, x_1, \ldots, x_{K_m}$ is defined by the hint. An unbiased estimate [22] of expected value of e_m over the input probability distribution is given by:

$$E_m(g) = \frac{1}{N_m} \sum_{i=1}^{N_m} e_m(g, x_{i0}, x_{i1}, \ldots, x_{iK_m}). \tag{4}$$

According to this definition, for H_0 (**examples hint**):

$$e_0(g, x) = (f(x) - g(x))^2. \tag{5}$$

Invariance hints form a very widely used class of hints [13]. For an invariance hint H_m defined on an input vector x and its transformed version x' according to the invariance, the hint objective function is:

$$e_m(g, x, x') = (g(x) - g(x'))^2. \tag{6}$$

The invariance relationship determines x':

- **Evenness**: $x' = -x$.
- **Scale invariance**: $x' = a * x$ for a constant a.
- **Cyclic shift invariance**: if $x = [x^0, x^1, \ldots, x^{I-1}]$, then $x' = [x^1, x^2, \ldots, x^{I-1}, x^0]$.
- and so on.

In this thesis we use the cyclic shift and evenness invariance hints and refer to them as H_1 and H_2 respectively.

Binary hint asserts that f is a binary function. If $g(x) \in [0,1]$, $\forall x, g$, the objective function for an example of the binary hint is:

$$e_m(g,x) = g(x) * (1 - g(x)). \tag{7}$$

Monotonicity hint: asserts that $f(x)$ is an increasing function of x. (The ordering $>$ imposed on x can be defined in different ways):

$$e_m(g,x,x') = \begin{cases} (g(x) - g(x'))^2 & \text{if } x < x' \text{ and } g(x) > g(x') \\ & \text{or } x > x' \text{ and } g(x) < g(x'); \\ 0 & \text{otherwise.} \end{cases} \tag{8}$$

Approximation Hint: asserts that $g(x) \in [a,b]$ is the acceptable range for $f(x)$:

$$e_m(g,x,a,b) = \begin{cases} (a - g(x))^2 & \text{if } g(x) < a; \\ (g(x) - b)^2 & \text{if } g(x) > b; \\ 0 & \text{otherwise.} \end{cases} \tag{9}$$

Smoothness hint: asserts that f is a function with small curvature. If curvature is measured by function Q, then:

$$e_m(g,x) = Q(g(x))) \tag{10}$$

For the examples hint, N_0 examples of H_0 are created by picking N_0 pairs $(x, f(x))$ from the training set. For the other hints above, since e_m is defined for *any* input vector x, it is possible to produce as many examples of the hint as required to teach it.

A hint H_m is taught by means of an optimization step on the error estimate $E_m(g) = \frac{1}{N_m} \sum_{i=1}^{N_m} e_i(g, x_{i0}, x_{i1}, \ldots, x_{iK_m})$.

For gradient descent, this optimization step is modification of weights w of the network, as follows:

$$\begin{aligned} w^{t+1} &= w^t - \eta * \frac{\partial E_m(g)}{\partial w} |_{w^t} \\ &= w^t - \eta * \frac{1}{N} \sum_{j=1}^{N_m} \frac{\partial e_m(g, x_{j0}, \ldots, x_{jK_m})}{\partial w} |_{w^t} \end{aligned} \tag{11}$$

In this equation, w^t denotes a weight at time t of the optimization, and e_m should be differentiable with respect to each weight w.

2 The Scheduling Problem

When hints $H_0, \ldots, H_M$ are given about f, they are expressed by their error (objective) functions $E_0, \ldots, E_M$. In order to use all the available information, one would like to optimize all $E_0, \ldots, E_M$ in such a way that the generalization error, E, on f is as small as possible.

The function f should be *consistent* with all the hints given about it. This requires the following consistency relation between E (generalization error) and E_i (error on hint H_i) [3]:

$$E \rightarrow 0 \Longrightarrow (E_i \rightarrow 0 \quad \forall i : 0, \ldots, M). \tag{12}$$

Equation (12) can be interpreted as:

$$(\exists i : 0, \ldots, M \quad not(E_i \rightarrow 0)) \Longrightarrow not(E \rightarrow 0). \tag{13}$$

Therefore *all* E_i's should be made as small as possible in order to have a small E. This statement can be stated as the optimization problem:

$$\min_{g \in G} E_i(g) \quad \forall i : 0, \ldots, M \tag{14}$$

with the implicit assumption $E_i(g) \geq 0, \quad \forall i : 0, \ldots, M;\ g \in G$.

There are at least two possible ways of minimizing all E_i:

- **Constrained Optimization:**

  If the hints $H_1, \ldots, H_M$ can be implemented and kept as constraints on G, then $E_i(g)$ for $i : 1, \ldots, M$ can be imposed as constraints on $g \in G$. In this case equation (14) which is an unconstrained simultaneous optimization problem, is transformed into the constrained optimization problem:

$$\min_{g \in G} E_0(g) \quad \text{with} \quad E_i(g) = 0, \quad \forall i : 1, \ldots, M. \tag{15}$$

  In many cases implementing and maintaining additional hints as constraints is impossible. Besides that, constraints may be restricting the model so much that f can not be implemented by the learning system. In this case a smaller E_0 at the expense of nonzero $E_1, \ldots, E_M$ may lead to a smaller generalization error.

  These ideas suggest simultaneous minimization of all E_i's instead of keeping some of them as hard constraints.

- **Penalty Functions:**

  Using penalty functions [12, 18] is a popular method of simultaneous minimization. According to this method, an objective function of the form:

  $$\hat{E} = \sum_{i=0}^{M} \alpha_i E_i \tag{16}$$

  is minimized. In this function α_i's are nonnegative *penalty coefficients*.

  A problem with penalty functions is how to figure out the penalty coefficients.

  Another drawback of using penalty functions may be "Steep valleys and discontinuous derivatives are created at the constraint boundary and these features are often difficult to overcome with gradient methods. Values of function at nonfeasible points are required, which may not be possible in practice. Penalty functions are not recommended as a general method." [15] However, these problems may be resolved by using sophisticated optimization techniques.

2.1 Schedules

We define a **schedule** as: *A procedure that decides on which hint to teach at a given iteration during training.* It takes into consideration all hints H_i, for $i : 0, \ldots, M$ and all the information available about the G and the training algorithm.

The penalty function in equation (16) implies that the effect of E_i on E is proportional to α_i. E_i should have a value inversely proportional to α_i, i.e. E_i should be smaller if α_i is larger. The schedules below are attempts to have E_i inversely proportional to α_i and all $\alpha_i E_i$ small and close to each other:

- **Fixed Schedules:**
  - **Rotation**: This schedule is based on the idea that *the more descents done on E_i the smaller it would be, therefore the number of descents on E_i should be proportional to α_i during total training time.* Even if the total number of descents is proportional to α_i's, it is important *when* those descents are done. For example, if an E_i is minimized till it becomes almost zero at the beginning and

then another E_j is minimized, either it may not be possible to reduce E_j because the search is in a local minimum, or reducing E_j may increase E_i, contrary to the objective of minimizing all E_i's. One possible solution is: Descend on E_0 with an amount proportional to α_0, and then descend on E_1 with an amount proportional to α_1, and so on.

Descent on E_i in proportion to α_i can be accomplished in the following ways:

* Use different learning rates η_i, proportional to α_i for each E_i, and descend on E_0 once, then E_1 once, and so on.
* Teach a number of examples of H_i proportional to α_i when it is E_i's turn.
* Teach each hint using the same number of examples of the hint and the same learning rate, however, give E_i a number of turns proportional to α_i. (We used this method in our experiments.)

When a rotation schedule is used: α_i and α_j have implications on:

* **(a)** How difficult it is to descend on E_i compared with E_j (implementation issue).
* **(b)** What the effect of E_i and E_j on generalization error is (generalization issue).

– **Random Rotation**: Since there is no reason to teach the hints in any particular order, which hint to teach at any pass can be chosen randomly according to the probability $\frac{\alpha_i}{\sum_{j=0}^{M} \alpha_j}$ for each hint H_i.

• **Adaptive Schedules:**

– **Maximum Error**: In rotation schedules, α_i have implications both in terms of implementation and generalization. Adaptive schedules are an attempt to eliminate the implementation issue from consideration. The maximum error schedule can be described as follows: If $\alpha_i E_i$ is the maximum over all $\alpha_j E_j : j : 0, \ldots, M$ then teach H_i (ties broken randomly).

In this schedule, α_i's only make an assertion on the generalization effect of each hint. The implementation issue is automatically handled because if a hint H_i is difficult to learn, then E_i

will remain large, and hence H_i is taught more frequently until E_i becomes small. Especially if each hint is implementable or learnable in different degrees, fixed schedules may not be useful unless one can adjust α_i's so as to deal with both implementation and generalization issues.

Another advantage of maximum error schedule compared with the fixed schedules is that it takes care of dependencies between hints. For example, if E_i and E_j are positively correlated then reducing E_i would also reduce E_j. Since it became smaller, E_j will not be chosen to be taught. For negative correlation the opposite would occur.

– **Random Maximum Error**: Randomized version of maximum error schedule. Each hint H_i has a probability of $\frac{\alpha_i * E_i}{\sum_{j=0}^{M} \alpha_j * E_j}$ of being taught at any iteration.

Experimental results on some of these schedules are given in the following section.

2.2 Experiments on Schedules

We experimented with the following function:

$$f(x) = \begin{cases} 0 & \text{if } |\sum_{i=0}^{7} x^i| > LIM, \\ 1 & \text{otherwise.} \end{cases} \tag{17}$$

LIM is chosen such that when P_X (the probability distribution on the input space X) is uniform on $[-1, 1]$, the probability that $f(x)$ will be 0 and the probability that it will be 1 are equal.

f has the cyclic shift and evenness invariances. Hence, we experimented with 3 different hints: H_0: the examples hint; H_1: cyclic shift hint; H_2: evenness hint.

The following schedules were used in the experiments. (The first entry gives the name of the schedule, and the second entry gives the weight (α_i) given to each hint.)

- **Schedule 0:** Rotation, 1-0-0.
- **Schedule 1:** Rotation, 1-1-1.

- **Schedule 2:** Rotation, 2-1-1.
- **Schedule 3:** Random rotation, 1-1-1.
- **Schedule 4:** Random rotation, 2-1-1.
- **Schedule 5:** Maximum error, 2-1-1.

The experiments were performed using the `train` program described in section 5. Backpropagation, with a learning rate (η) of 0.3 and a momentum (α) of 0.6, was used as the learning algorithm. The sequential mode of training was used. In this mode, for each training example, the inputs of the example are forwarded, the training error is measured and then a descend is made on this error.

A feed-forward neural network with 3 layers of sigmoidal units: 8 input, 3 hidden, and 1 output (an 8-3-1 network) was used. Training sets used were of sizes N_0=10, 20, 50, and 80. 5 different training sets were generated for each N_0. and for each training set 4 experiments with different initial weights of the network were run. Hence, for each training set size and schedule pair, 20 experiments were performed. Each experiment was run for 2000 `pass`es; at each pass 20 examples were used for training.

In order to compare the performance of different schedules for different training set sizes, the minimum generalization achieved during 2000 passes were found and averaged over all 20 experiments for each different training set size ($N_0 : 10, 20, 50, 80$) and schedule (Schedule : $0, 1, 2, 3, 4, 5$) pair.

Schedule No	$N_0 = 10$	$N_0 = 20$	$N_0 = 50$	$N_0 = 80$
Schedule 0	0.256	0.252	0.210	0.106
Schedule 1	0.203	0.109	0.061	0.059
Schedule 2	0.255	0.137	0.050	0.047
Schedule 3	0.251	0.144	0.059	0.048
Schedule 4	0.253	0.157	0.060	0.036
Schedule 5	0.255	0.168	0.069	0.071

Table 1: Average minimum generalization error reached for each schedule and training set size.

Schedule No	$N_0 = 10$	$N_0 = 20$	$N_0 = 50$	$N_0 = 80$
Schedule 0	0.008	0.005	0.057	0.093
Schedule 1	0.091	0.107	0.094	0.093
Schedule 2	0.007	0.112	0.084	0.085
Schedule 3	0.006	0.106	0.089	0.084
Schedule 4	0.006	0.107	0.095	0.071
Schedule 5	0.007	0.103	0.099	0.104

Table 2: Standard deviations corresponding to the averages in table 1.

In table 1, the averages of minimum generalization error reached for each training set size and schedule pair are listed. Table 2 gives the standard deviations corresponding to the averages in the previous table.

In figure 3, the average minimum generalization error reached for each training set size is shown as a function of schedules. For a very small training set size (10), in general using additional hints was not helpful. This can be explained as follows: when the number of examples of the function is very small, the additional restriction obtained on the G space is not enough to reduce the generalization error. With larger training set sizes, using additional hints decreased the minimum generalization error achieved.

In figures 4 and 5, the average generalization error versus pass number is shown for each schedule. Each curve is an average of 20 runs. In figures 6 and 7, the same figures are shown with variances (error bars). The significance of the latter two figures is that they illustrate that the probability that using additional hints will improve generalization is high. In general, after the initial passes, using hints with all the schedules decreased the generalization error. In the presence of additional hints, all schedules used in this experiment gave comparable results. For that reason we have no conclusive evidence about how these schedules compare to one another.

Schedule 5 would be expected to perform better than the others, since it is an adaptive schedule, but it didn't. Two reasons that we can think of for this behavior are: First, the weights α_i given to each hint may be too far from their effect on generalization error. Second, in all the runs performed using schedule 5, the errors E_0, E_1, E_2 were about the same at the end of training. However, they came to a large value and stayed there for a while before the end of training. Our interpretation for this phenomenon is: the

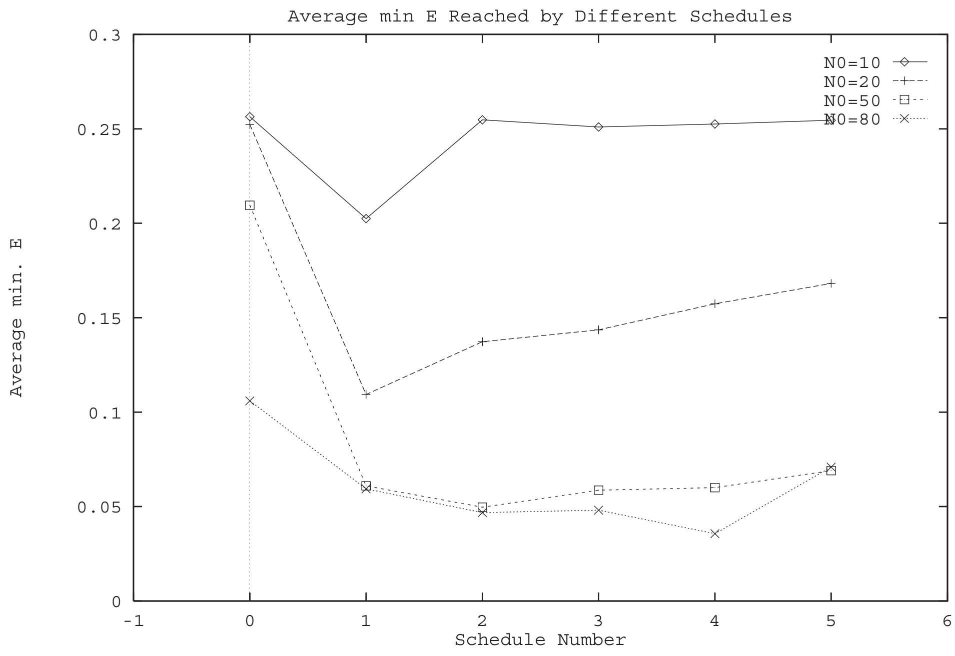

Figure 3: Using additional hints improves the average minimum generalization error reached during a run.

errors all went into local minimums and none of them could get out. An algorithm that avoids local minima (like TRUST [10]) could be helpful in that case.

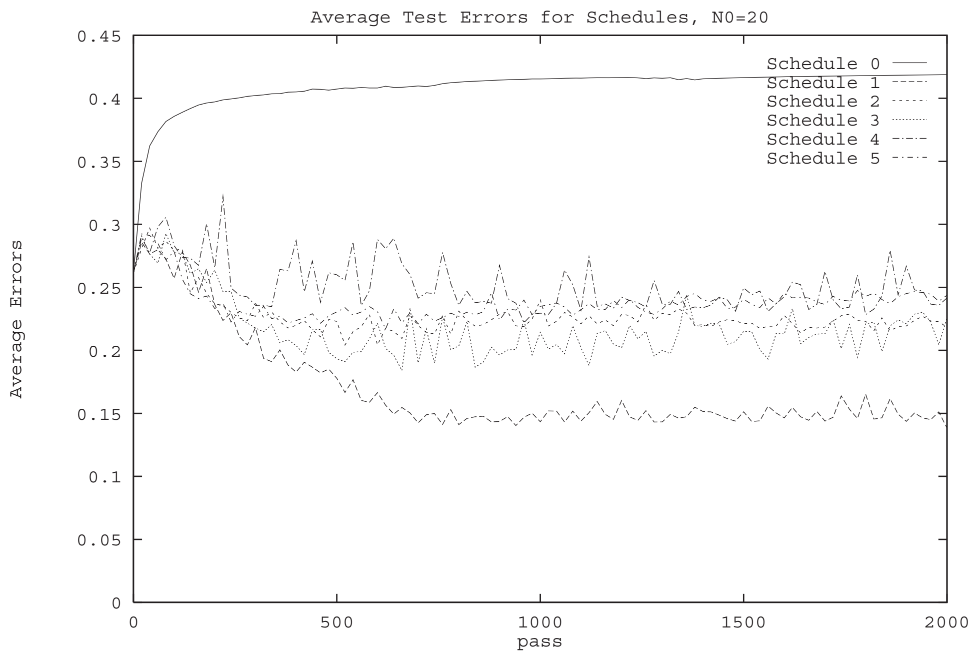

Figure 4: Average generalization error for different schedules with training sets of size 20.

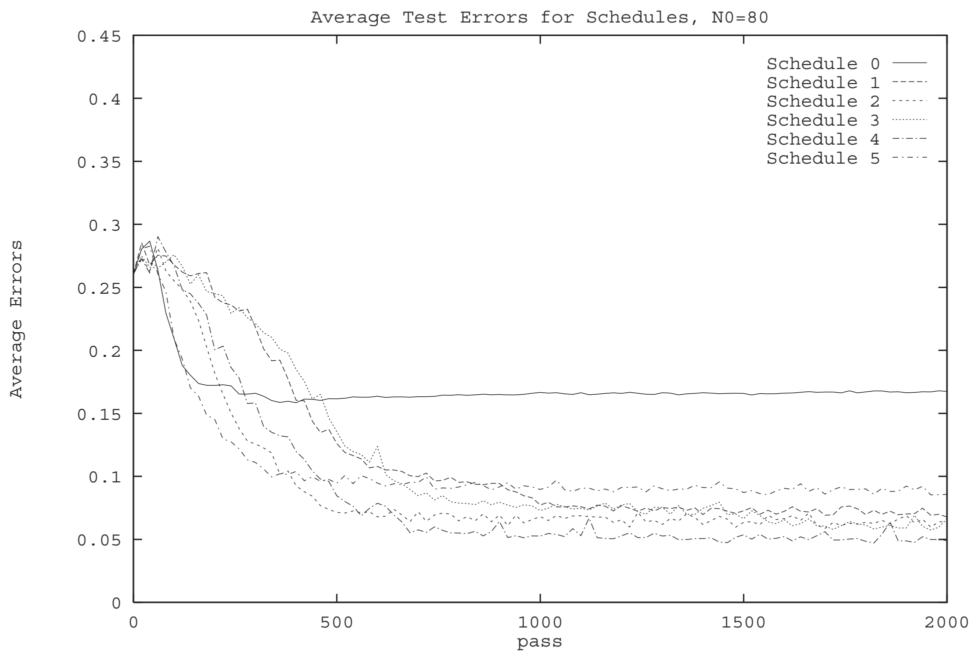

Figure 5: Average generalization error for different schedules with training sets of size 80.

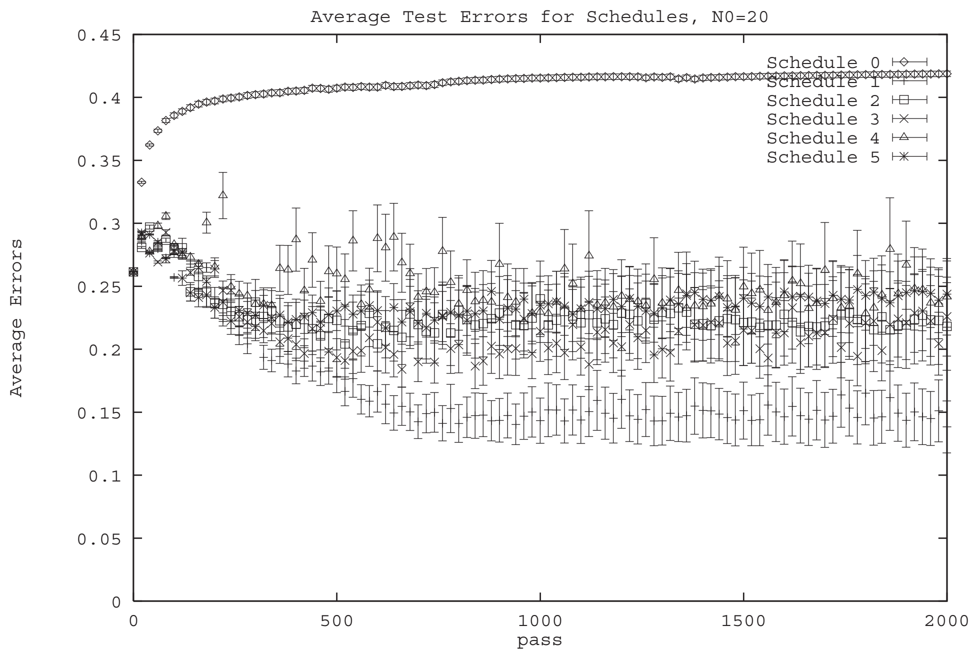

Figure 6: Average generalization error for different schedules with training sets of size 20 (with variances).

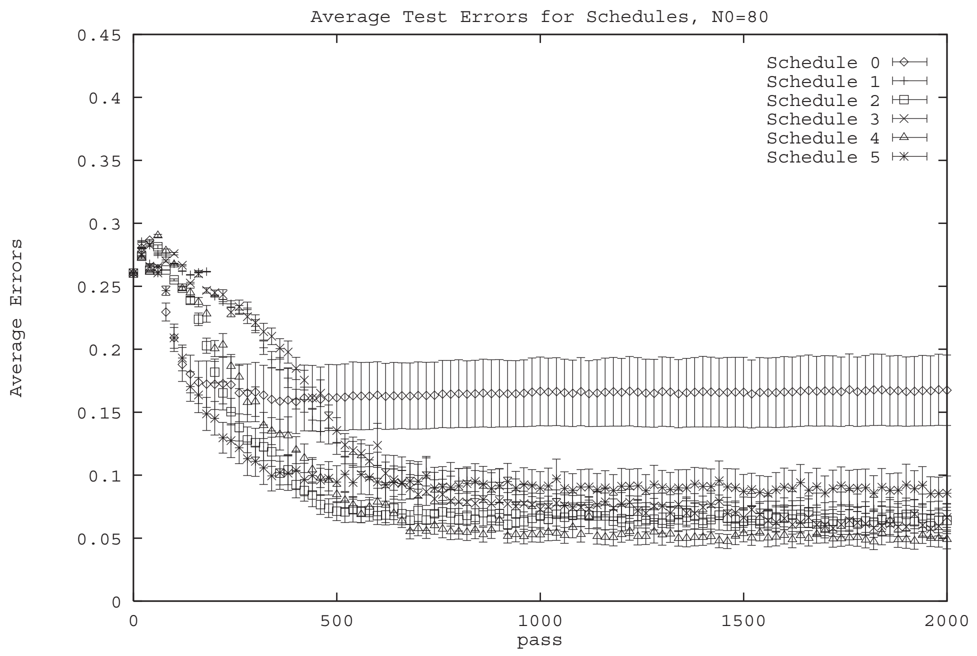

Figure 7: Average generalization error for different schedules with training sets of size 80 (with variances).

3 Adaptive Minimization

Consider the maximum error schedule given above. Assume $\alpha_i E_i = \max_j \alpha_j E_j$. Define $\hat{E}_k$, an estimate including all hints but H_k, as:

$$\hat{E}_k = \sum_{j=0, j\neq k}^{M} \alpha_j E_j. \tag{18}$$

Then clearly:

$$\min_{k:0,\ldots,M} \hat{E}_k = \hat{E}_i = \sum_{j=0, j\neq i}^{M} \alpha_j E_j. \tag{19}$$

Because $\alpha_i E_i$ was the term which contributed the most to

$$\hat{E}(E_0, E_1, \ldots, E_M) = \sum_{j=0}^{M} \alpha_j E_j. \tag{20}$$

Therefore the maximum error schedule can be articulated as:

For an estimate $\hat{E}(E_0, E_1, \ldots, E_M) = \sum_{j=0}^{M} \alpha_j E_j$, if $\min_k \hat{E}_k = \hat{E}_i$ then choose to teach H_i.

In general, assume that $\hat{E}(E_0, E_1, \ldots, E_M)$ is an estimate of E, and $\hat{E}_k$ is $\hat{E}$ including all hints but H_k. If $\min_{k=0,\ldots,M} \hat{E}_k = \hat{E}_i$, then according to the information obtained from all hints except H_k, H_i is the hint whose absence decreases $\hat{E}$ the most. Hence at the same time H_i is the hint whose presence increases $\hat{E}$ the most, therefore H_i should be taught. This process of deciding which hint to teach, taking into consideration the information provided by all hints together, is called *Adaptive Minimization* [2].

If $\hat{E}$ is a weighted sum of E_i's, then implementing it as a rotation schedule or a maximum error schedule is straightforward. However, if $\hat{E}$ is not in that form, then adaptive minimization can be used to implement $\hat{E}$ as a schedule. One remark is that adaptive minimization is also a schedule like rotation or maximum error, and it is a general adaptive schedule.

In general, given $\hat{E}(E_0, \ldots, E_M)$ as an estimate of E, there are at least two possible ways of descending on it. One of them is adaptive minimization, as described, and the other one is descending on $\hat{E}$ directly. Adaptive minimization requires $\hat{E}_k$ and direct optimization requires derivatives of $\hat{E}$ as explained below.

3.1 Calculating Derivatives of $\hat{E}$:

When direct optimization of $\hat{E}$ is chosen as the method to implement an estimate $\hat{E}$, various optimization techniques may require the value of $\hat{E}$, or its derivatives with respect to parameters of the learning-from-examples system G. If G is a feed-forward neural network with a fixed architecture, these parameters are the weights of the network. In this section, calculations of the first and second derivatives of $\hat{E}$ with respect to weights of a feed-forward neural network are given. For a more general discussion of this topic see [9].

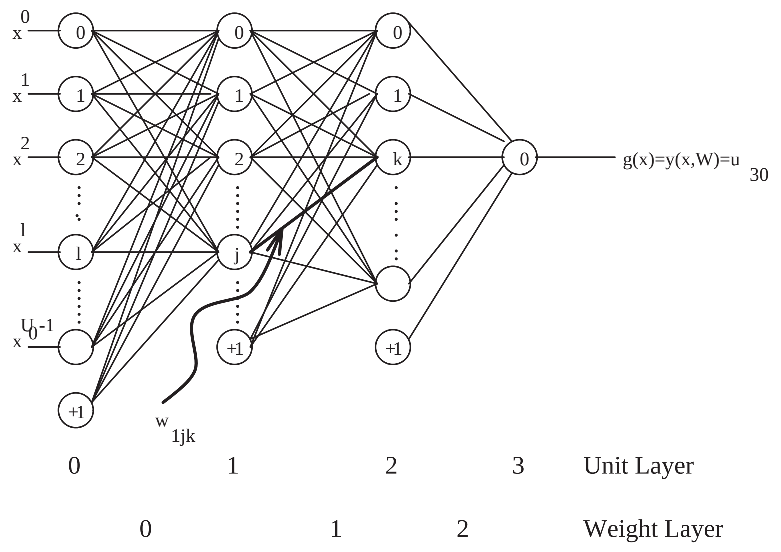

Figure 8: A feed-forward neural network with 4 layers of units.

Given $M+1$ hints $H_0, H_1, \ldots, H_M$ with respective error estimates E_0, $E_1, \ldots, E_M$ and error on an example of the hint $e_0, e_1, \ldots, e_M$, assume that:

- The elements of the learning-from-examples system G are feed-forward neural networks with a fixed architecture. Each unit (neuron) (except the units at the input layer) computes weighted sum of its inputs, adds a bias value to this sum, passes this sum through an activation function and sends the result to all the units in the next layer.

  Let's have the following notation for a neural network (see figure 8):

  - There are $L+1$ layers of units numbered $0, 1, \ldots, L$, $L > 2$, with layer 0 being the input layer and layer L being the output layer.
  - Each layer l, except the output layer, has $U_l + 1$ units numbered $0, 1, \ldots, U_l$. The output of jth unit of ith layer is called u_{ij} (when there is no confusion we use the same notation for the unit itself.) u_{iU_i} is $+1$ (bias unit) for each layer i, and u_{iU_i}, U_ith unit of ith layer, is not connected to any previous units, but all the units of the $i+1$st layer. The last layer L has only one output unit, connected to all units in layer $L-1$. When input vector x is fed from the input layer of the neural network, the output of the neural network for an input vector x is $g(x) = y(x, W) = u_{L0}$.
  - There are L layers of weights numbered $0, 1, \ldots, L-1$. The weight from jth unit of ith layer to kth unit of $i+1$st layer is called w_{ijk}. There are weights between consecutive layers of units only.
  - Each unit $u_{ij} : i : 1, \ldots, L; j : 0, \ldots, U_i - 1$ computes:

$$u_{ij} = t(net_{ij}) \tag{21}$$

    where net_{ij} is defined as:

$$net_{ij} = \sum_{k=0}^{U_{i-1}} w_{(i-1)kj} u_{(i-1)k} \tag{22}$$

    and t is a threshold function, which may be linear ($t(x) = ax + b$), sigmoid ($t(x) = \frac{1}{1+e^{-x}}$), tanh ($t(x) = \frac{e^{2x}-1}{e^{2x}+1}$), etc.. t should be first order differentiable with respect to its input if $\frac{\partial \hat{E}}{\partial w_{ijk}}$ is needed, and t should be second order differentiable with respect to its input if $\frac{\partial^2 \hat{E}}{\partial w_{ijk} \partial w_{nop}}$ is needed.

- $\hat{E}$ is a function of E_0, $E_1, \ldots, E_M$ and some constants only.
- If $\frac{\partial \hat{E}}{\partial w_{ijk}}$ is needed then $\frac{\partial \hat{E}}{\partial E_l}$ exists, and if $\frac{\partial^2 \hat{E}}{\partial w_{ijk} \partial w_{nop}}$ is needed then $\frac{\partial^2 \hat{E}}{\partial E_l \partial E_m}$ exists for all $l, m : 0, \ldots, M$.
- For notational convenience, let's denote the error on one example of hint H_m by $e_m(y_0(W), \ldots, y_{K_m}(W))$ instead of $e_m(g, x_0, \ldots, x_{K_m})$. In the new notation, W is the set of all weights of the network, and $y_j(W)$ stands for $g(x_j)$. Therefore y_j can be taken as a variable dependent on W only.

- E_m is estimated using N_m examples of H_m by:

$$E_m(g) = \frac{1}{N_m} \sum_{l=1}^{N_m} e_m(y_{l0}(W), \ldots, y_{lK_m}(W)) \tag{23}$$

- $\frac{\partial e_m(y_0(W),\ldots,y_{K_m}(W))}{\partial y_q}$ exists for each $q : 0, \ldots, K_m$.

3.1.1 The First Derivatives of $\hat{E}$

Based on these assumptions, the first derivatives of $\hat{E}$ with respect to each weight w_{ijk} can be computed as:

$$\begin{aligned}
\frac{\partial \hat{E}}{\partial w_{ijk}} &= \sum_{m=0}^{M} \frac{\partial \hat{E}}{\partial E_m} \frac{\partial E_m}{\partial w_{ijk}} \\
&= \sum_{m=0}^{M} \frac{\partial \hat{E}}{\partial E_m} \frac{1}{N_m} \sum_{l=1}^{N_m} \frac{\partial e_m(y_{l0}(W), \ldots, y_{lK_m}(W))}{\partial w_{ijk}} \\
&= \sum_{m=0}^{M} \frac{\partial \hat{E}}{\partial E_m} \frac{1}{N_m} \sum_{l=1}^{N_m} \sum_{q=0}^{K_m} \frac{\partial e_m(y_{l0}(W), \ldots, y_{lK_m}(W))}{\partial y_{lq}} \frac{\partial y_{lq}(W)}{\partial w_{ijk}} \qquad (24)
\end{aligned}$$

In equation (24), $\frac{\partial \hat{E}}{\partial E_m}$ is a function of $E_0, \ldots, E_M$ and some constants only, and so it can be computed directly. $\frac{\partial e_m(y_{l0}(W),\ldots,y_{lK_m}(W))}{\partial y_{lq}}$ is a function of the outputs $y_{lp} : p : 0, \ldots, K_m; l : 0, \ldots, N_m$ of the network for inputs x_{lp}, and hence it can also be computed directly. In order to compute $\frac{\partial y_{lq}(W)}{\partial w_{ijk}}$, backpropagation [20] can be used. For notational convenience, the calculation of $\frac{\partial y(W)}{\partial w_{ijk}}$ for a fixed input vector x is given:

$$\begin{aligned}
\frac{\partial y(W)}{\partial w_{ijk}} &= \frac{\partial y(W)}{\partial net_{(i+1)k}} \frac{\partial net_{(i+1)k}}{\partial w_{ijk}} \\
&= \frac{\partial y(W)}{\partial net_{(i+1)k}} u_{ij} \qquad (25)
\end{aligned}$$

The last step follows since $net_{(i+1)k} = \sum_{p=0}^{U_i} w_{ipk} u_{ip}$, and hence $\frac{\partial net_{(i+1)k}}{\partial w_{ijk}} = u_{ij}$.

Let's define $\delta_{(i+1)k}$ for unit u_{ik} as:

$$\delta_{(i+1)k} = \frac{\partial y(W)}{\partial net_{(i+1)k}} \tag{26}$$

If $i = L-1$, since we consider only one output unit $k = 0$, hence $u_{(i+1)0}$ is the output unit. Then δ_{L0} for the output unit u_{L0} is computed as:

$$\begin{aligned}\delta_{L0} &= \frac{\partial y(W)}{\partial net_{L0}} \\ &= \frac{\partial t(net_{L0})}{\partial net_{L0}} \\ &= t'(net_{L0}) \end{aligned} \tag{27}$$

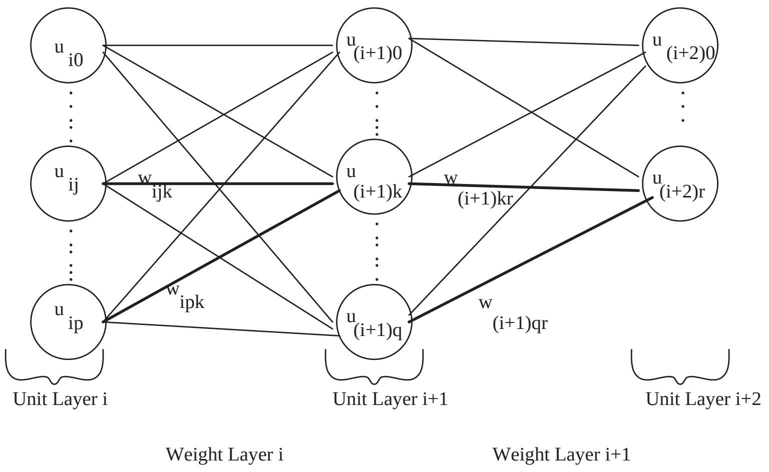

Figure 9: A unit $u_{(i+1)k}$ and units and weights around it.

If $i < L-1$ and hence $u_{(i+1)k}$ is not an output unit, then $\delta_{(i+1)k}$ is computed as (see figure 9 for indices):

$$\begin{aligned}\delta_{(i+1)k} &= \frac{\partial y(W)}{\partial net_{(i+1)k}} \\ &= \frac{\partial y(W)}{\partial u_{(i+1)k}} \frac{\partial u_{(i+1)k}}{\partial net_{(i+1)k}} \end{aligned}$$

$$
\begin{aligned}
&= \frac{\partial y(W)}{\partial u_{(i+1)k}} t'(net_{(i+1)k}) \\
&= t'(net_{(i+1)k}) \frac{\partial y(W)}{\partial u_{(i+1)k}} \\
&= t'(net_{(i+1)k}) \sum_{r=0}^{U_{i+2}-1} \frac{\partial y(W)}{\partial net_{(i+2)r}} \frac{\partial net_{(i+2)r}}{\partial u_{(i+1)k}} \\
&= t'(net_{(i+1)k}) \sum_{r=0}^{U_{i+2}-1} \frac{\partial y(W)}{\partial net_{(i+2)r}} \frac{\partial}{\partial u_{(i+1)k}} \sum_{q=0}^{U_{i+1}} w_{(i+1)qr} u_{(i+1)q} \\
&= t'(net_{(i+1)k}) \sum_{r=0}^{U_{i+2}-1} \frac{\partial y(W)}{\partial net_{(i+2)r}} w_{(i+1)kr} \\
&= t'(net_{(i+1)k}) \sum_{r=0}^{U_{i+2}-1} \delta_{(i+2)r} w_{(i+1)kr} \qquad (28)
\end{aligned}
$$

3.1.2 The Second Derivatives of $\hat{E}$

The second derivatives of $\hat{E}$ with respect to weights w_{ijk} and w_{nop} can be computed as follows:

$$
\frac{\partial^2 \hat{E}}{\partial w_{ijk} \partial w_{nop}} = \frac{\partial^2 \hat{E}}{\partial w_{nop} \partial w_{ijk}} = \frac{\partial}{\partial w_{nop}} \left(\frac{\partial \hat{E}}{\partial w_{ijk}} \right)
$$

$$
\begin{aligned}
&= \frac{\partial}{\partial w_{nop}} \left(\sum_{m=0}^{M} \frac{\partial \hat{E}}{\partial E_m} \frac{1}{N_m} \sum_{l=1}^{N_m} \frac{\partial e_m(y_{l0}(W), \ldots, y_{lK_m}(W))}{\partial w_{ijk}} \right) \\
&= \frac{1}{N_m} \sum_{m=0}^{M} \frac{\partial}{\partial w_{nop}} \left(\frac{\partial \hat{E}}{\partial E_m} \right) \sum_{l=1}^{N_m} \frac{\partial e_m(y_{l0}(W), \ldots, y_{lK_m}(W))}{\partial w_{ijk}} + \\
&\quad \frac{1}{N_m} \sum_{m=0}^{M} \frac{\partial \hat{E}}{\partial E_m} \sum_{l=1}^{N_m} \frac{\partial^2 e_m(y_{l0}(W), \ldots, y_{lK_m}(W))}{\partial w_{ijk} \partial w_{nop}} \\
&= \frac{1}{N_m} \sum_{m=0}^{M} \frac{\partial}{\partial w_{nop}} \left(\frac{\partial \hat{E}}{\partial E_m} \right) \sum_{l=1}^{N_m} \frac{\partial e_m(y_{l0}(W), \ldots, y_{lK_m}(W))}{\partial w_{ijk}} + \\
&\quad \frac{1}{N_m} \sum_{m=0}^{M} \frac{\partial \hat{E}}{\partial E_m} \sum_{l=1}^{N_m} \frac{\partial}{w_{nop}} \left(\frac{\partial e_m(y_{l0}(W), \ldots, y_{lK_m}(W))}{\partial w_{ijk}} \right)
\end{aligned}
$$

$$
\begin{aligned}
&= \frac{1}{N_m}\sum_{m=0}^{M}\frac{\partial}{\partial w_{nop}}\left(\frac{\partial \hat{E}}{\partial E_m}\right)\sum_{l=1}^{N_m}\frac{\partial e_m(y_{l0}(W),\ldots,y_{lK_m}(W))}{\partial w_{ijk}} + \\
&\quad \frac{1}{N_m}\sum_{m=0}^{M}\frac{\partial \hat{E}}{\partial E_m}\sum_{l=1}^{N_m}\frac{\partial}{w_{nop}}\left(\sum_{q=0}^{K_m}\frac{\partial e_m(y_{l0}(W),\ldots,y_{lK_m}(W))}{\partial y_{lq}}\frac{\partial y_{lq}}{\partial w_{ijk}}\right) \\
&= \frac{1}{N_m}\sum_{m=0}^{M}\frac{\partial}{\partial w_{nop}}\left(\frac{\partial \hat{E}}{\partial E_m}\right)\sum_{l=1}^{N_m}\frac{\partial e_m(y_{l0}(W),\ldots,y_{lK_m}(W))}{\partial w_{ijk}} + \\
&\quad \frac{1}{N_m}\sum_{m=0}^{M}\frac{\partial \hat{E}}{\partial E_m}\sum_{l=1}^{N_m}\sum_{q=0}^{K_m}\frac{\partial}{w_{nop}}\left(\frac{\partial e_m(y_{l0}(W),\ldots,y_{lK_m}(W))}{\partial y_{lq}}\right) + \\
&\quad \frac{1}{N_m}\sum_{m=0}^{M}\frac{\partial \hat{E}}{\partial E_m}\sum_{l=1}^{N_m}\sum_{q=0}^{K_m}\frac{\partial e_m(y_{l0}(W),\ldots,y_{lK_m}(W))}{\partial y_{lq}}\frac{\partial^2 y_{lq}(W)}{\partial w_{ijk}\partial w_{nop}} \qquad (29)
\end{aligned}
$$

In the final step, $\frac{\partial \hat{E}}{\partial E_m}$ is a function of $E_0, \ldots, E_M$ and some constants only, hence it can be regarded as a new estimate $\hat{E}'$ and its derivatives with respect to the weights can be calculated the same way $\frac{\partial \hat{E}}{\partial w_{ijk}}$ is calculated in equation (24). Similarly, $\frac{\partial e_m(y_{l0}(W),\ldots,y_{lK_m}(W))}{\partial y_{lq}}$ is a function of $y_{l0}(W)$, $\ldots, y_{lK_m}(W)$ only, hence it can be regarded as a new error function e'_m and its derivative can be calculated using the same methods used for $\frac{\partial e_m}{\partial w_{ijk}}$ in equation (24). The only remaining term which does not seem so explicit to compute is $\frac{\partial^2 y_{lq}(W)}{\partial w_{ijk}\partial w_{nop}}$.

A symbolic mathematical programming package such as [11] can be used to determine intermediate derivatives.

If the optimization technique requires higher order derivatives of $\hat{E}$ with respect to the weights, a derivation similar to the one used for the second derivatives should be enough.

4 Estimates

In the previous section, it was shown that given an estimate $\hat{E}(E_0, E_1, \ldots, E_M)$ of the generalization error E, there are methods for incorporating this estimate in the learning process, and hence getting the advantage of hints. In this section, *a* method for finding an estimate $\hat{E}$ for two invariance hints is given.

Before finding an estimate, let's state some of the desired properties of the estimate [3]:

- $\hat{E}$ should be computed easily in terms of time and memory.
- In general, minimizing $\hat{E}$ implies minimizing E. (monotonicity condition).

4.1 Noise Approach

In this section, derivation of an $\hat{E}$ for a binary function f is given. For a more detailed discussion of this method see [7].

Assume that the function $f : \mathcal{R}^n \rightarrow \{0,1\}$ is to be implemented by a learning-from-examples system. The error of $g : \mathcal{R}^n \rightarrow [0,1]$ on an example of f can be modeled by a noise function n:

$$n(x_i) = |f(x_i) - g(x_i)| = \begin{cases} 1 - g(x_i), & \text{if } f(x_i) = 1; \\ g(x_i), & \text{if } f(x_i) = 0. \end{cases} \tag{30}$$

Assume that the noise function n has a mean μ and a variance σ^2. Then the learning performance of the network can be measured by:

$$E = \mathcal{E}((f(x) - g(x))^2) = \mathcal{E}(n^2(x)) = \mu^2 + \sigma^2 \tag{31}$$

Similarly, the error on any invariance hint can be measured by:

$$E'_i = \mathcal{E}((g(x) - g(x'))^2) = \mathcal{E}((n(x) - n(x'))^2) = 2\sigma^2 \tag{32}$$

assuming that $n(x)$ and $n(x')$ are independent random variables.

Since only estimates E_0 for E, E_1 for E'_1, and E_2 for E'_2 are available, these estimates are used in equations (31) and (32).

In order to get an estimate of μ, the mean of n, the training set examples are used as follows:

$$[\mu] = \frac{1}{N_0} \sum_{i=1}^{N_0} |f(x_i) - g(x_i)| \tag{33}$$

Combining these formulas, an estimate of σ^2 is:

$$[\sigma^2] = \frac{2(E_0 - [\mu]^2) + E_1 + E_2}{6} \tag{34}$$

and finally, an estimate of E, using $E_i : (0 \leq i \leq 2)$ and $[\mu]$ is:

$$\hat{E} = [\sigma^2] + [\mu]^2 \tag{35}$$

4.1.1 Experiments on Noise Approach Estimate

The same experiments used in section 2 were used to evaluate the estimate $\hat{E}$ in equation (35).

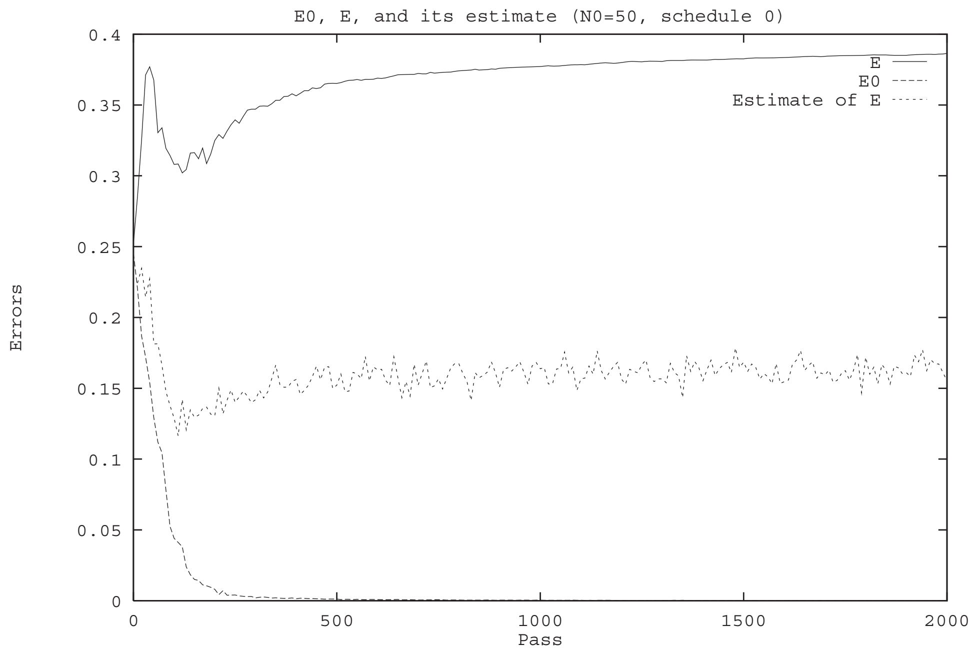

Figure 10: Behavior of $\hat{E}$ compared to E and E_0, with N_0=50 and schedule=0. $\hat{E}$ follows E as overtraining takes place.

In order to illustrate the relationship between E, E_0 and $\hat{E}$ as training takes place, example plots of them are shown in figures 10, 11 and 12. In figure 10, only H_0 was used for training. Notice that although E_0 continued to decrease, $\hat{E}$ did follow E. The reason for that is: $\hat{E}$ is a function of E_1 and E_2 too, and they did not decrease while E_0 did. In figure 11 all hints were used for training. In this case, both $\hat{E}$ and E_0 followed E, and they would both be good estimates of E. Figure 12 shows a case in which the estimate $\hat{E}$ is inadequate because the number of examples of f is too small (10). In this run, all hints were taught, and hence E_0, E_1 and E_2 decreased during training. However, information provided by H_0, H_1, and H_2 was not

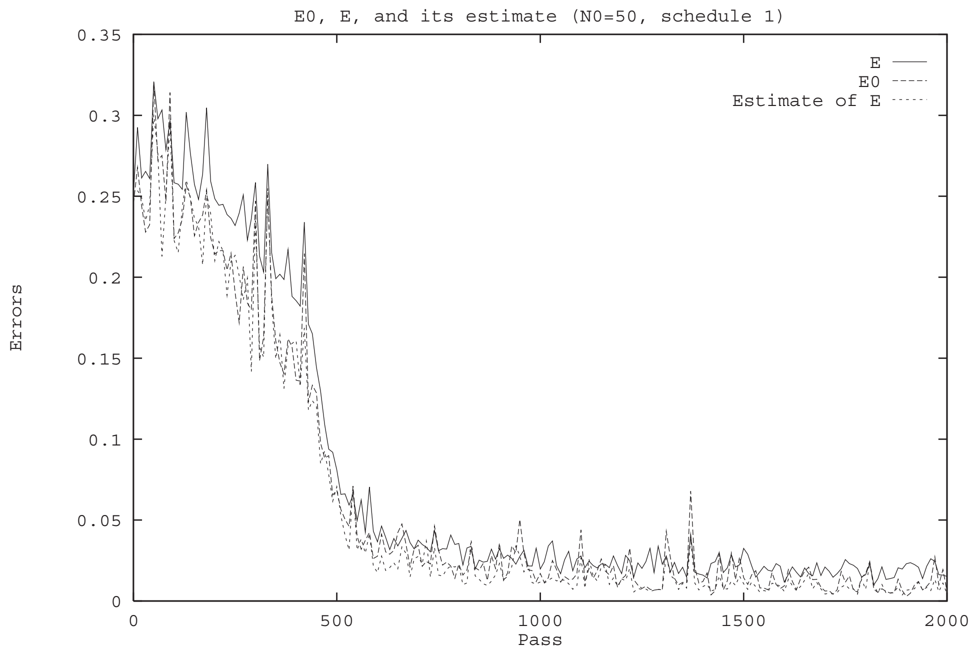

Figure 11: Behavior of $\hat{E}$ compared to E and E_0, with N_0=50 and schedule=1. Both $\hat{E}$ and E_0 follow E.

enough to generalize, hence E increased as training proceeded. However, $\hat{E}$ is a function of E_0, E_1, and E_2 (and $[\mu]$, but for small E_0, $[\mu]$ is small, too) only and hence could not follow the change in E.

$\hat{E}$ clearly satisfies the first property proposed for an estimate: it can be computed easily. In order to test the second property, whether minimizing $\hat{E}$ implies minimizing E or not, we use the following measure:

Define E_r as the generalization error when minimum $\hat{E}$ is reached, divided by the minimum generalization error during the run:

$$E_r = \frac{E|_{min\hat{E}}}{E_{min}} \tag{36}$$

The closer E_r is to 1, the higher is the probability that minimizing $\hat{E}$ implies minimizing E. In general small E_r would be in favor of $\hat{E}$.

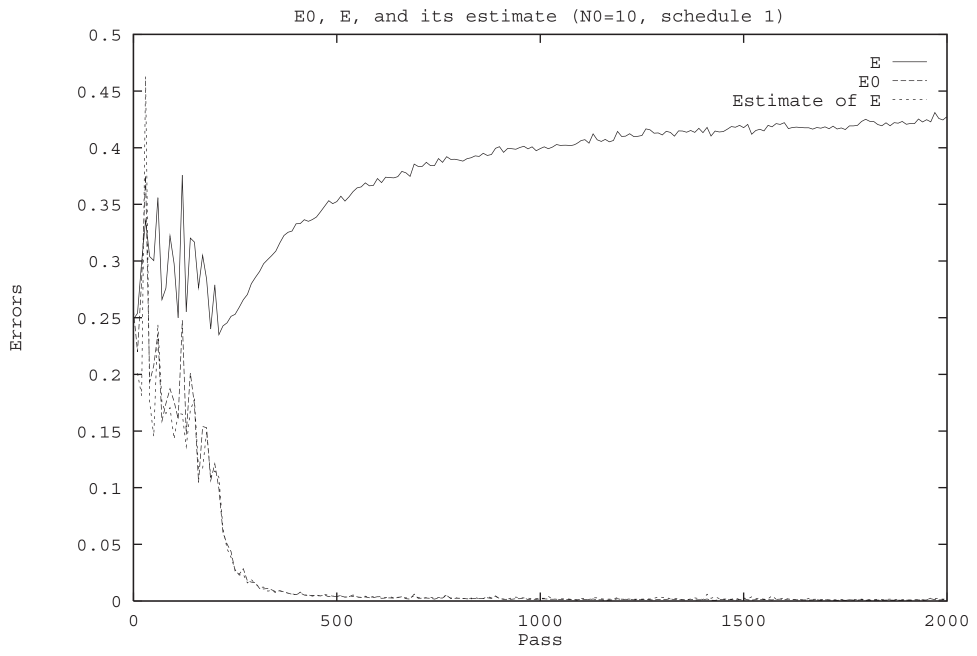

Figure 12: Behavior of $\hat{E}$ compared to E and E_0, with N_0=10 and schedule=1. Neither $\hat{E}$ nor E_0 can follow E.

An estimate $\hat{E}$ may be useful for another task at the same time: If all the examples of a function are used for training and it is not possible to have a validation set, then $\hat{E}$ can be used to determine when to stop training. If $\hat{E}$ was not known, then E_0 would be the only measure to determine when to stop. Hence, another ratio, E_{0r}, is defined as the ratio of generalization error when minimum $\hat{E}$ is reached, divided by the generalization error when the minimum E_0 is reached:

$$E_{0r} = \frac{E|_{min\hat{E}}}{E|_{minE_0}} \tag{37}$$

Again small E_{0r} would be in favor of $\hat{E}$.

In tables 3 and 4 the average values of E_r and the standard deviations for these averages are listed. The last line in table 3 is the average over all

Schedule No	$N_0 = 10$	$N_0 = 20$	$N_0 = 50$	$N_0 = 80$
Schedule 0	1.396	1.556	1.343	1.258
Schedule 1	1.561	1.405	1.348	1.312
Schedule 2	1.788	1.541	1.379	1.534
Schedule 3	1.726	1.436	1.269	1.543
Schedule 4	1.799	1.571	1.585	1.262
Schedule 5	1.702	1.434	1.354	1.400
Average	1.662	1.491	1.380	1.385

Table 3: Average E_r ratios for each schedule and N_0.

Schedule No	$N_0 = 10$	$N_0 = 20$	$N_0 = 50$	$N_0 = 80$
Schedule 0	0.083	0.155	0.208	0.216
Schedule 1	0.275	0.263	0.213	0.180
Schedule 2	0.162	0.271	0.218	0.311
Schedule 3	0.198	0.226	0.123	0.343
Schedule 4	0.169	0.307	0.368	0.090
Schedule 5	0.219	0.308	0.189	0.206

Table 4: Standard deviations corresponding to averages in table 3.

schedules. When averaged over all runs, E_r was 1.48, which means if $\hat{E}$ was used, instead of E, to decide on when to stop training, the trained networks would have 48% more generalization error. However, effectively, using $\hat{E}$ instead of the validation error may be more beneficial than indicated by this figure.

If the examples of f are not set aside as the validation set and instead used for training, and $\hat{E}$ is used as the criteria to stop training, using $\hat{E}$ may be better. The variance in the data is too much to arrive at a conclusion, but to give the reader an idea, the following calculation is carried out as an example: For $N_0 = 20$ the average minimum E achieved is 0.161 (average of the second column of table 1.) However, for $N_0 = 80$ the average generalization error achieved is 0.0612 (average of the fourth column of table 1), and for $N_0 = 50$ it is 0.0848 (average of the third column of table 1.) 48% more of minimum generalization error would be 0.091 for $N_0 = 80$, and it would be 0.126 for $N_0 = 50$. Both numbers are smaller than 0.161. Hence, if the validation set consisted of 30, or 60, or more examples, using them for training, and using $\hat{E}$ to stop training *could* cause smaller generalization error.

Schedule No	$N_0 = 10$	$N_0 = 20$	$N_0 = 50$	$N_0 = 80$
Schedule 0	0.822	0.958	0.908	0.872
Schedule 1	0.935	0.934	0.968	0.935
Schedule 2	1.012	0.999	0.934	0.896
Schedule 3	1.011	1.002	0.898	0.985
Schedule 4	1.014	0.961	0.863	0.722
Schedule 5	1.000	1.013	0.886	0.933
Average	0.966	0.978	0.910	0.891

Table 5: Average E_{0r} ratios for each schedule and N_0.

Schedule No	$N_0 = 10$	$N_0 = 20$	$N_0 = 50$	$N_0 = 80$
Schedule 0	0.051	0.035	0.105	0.163
Schedule 1	0.130	0.137	0.159	0.090
Schedule 2	0.016	0.123	0.152	0.169
Schedule 3	0.013	0.176	0.135	0.144
Schedule 4	0.023	0.188	0.176	0.187
Schedule 5	0.015	0.030	0.182	0.062

Table 6: Standard deviations corresponding to averages in table 5.

In tables 5 and 6, the average values of E_{0r} and the standard deviations for these averages are given. The last line in table 5 is the average over all schedules. When averaged over all runs, E_{0r} is 0.94, which means that as a criteria to stop training $\hat{E}$ is on the average 6% better than E_0.

5 `gentrain` and `train`: Programs to Simulate Learning from Hints on Neural Networks

In order to do experiments on learning from hints using neural networks, we have developed two programs: `gentrain` and `train`.

Both programs are written in C, and run on Unix, Mac, and DOS environments.

In the following descriptions of these two programs are given.

5.1 `gentrain` Program

This program produces a set of examples of a function $f : X \rightarrow Y$ where $X, Y \subset \mathcal{R}^D, D > 0, D \in \mathcal{Z}$. The set of examples can then be used either as training or validation set for the `train` program.

5.1.1 Inputs and Outputs

The `gentrain` program takes the following parameters as input:

- $|X|$ (`input_dimension`): Some functions can be defined for any input dimension, e.g. sum of elements of a vector. In order to generate examples of such a function for different input dimensions, `input_dimension` variable is used.
- `number_of_examples`: Number of examples of function f to be produced.
- `[input_lower_limit, input_upper_limit]`: The range of input vectors. Input vectors are chosen uniformly from real numbers in this range.
- `seed`: The seed for the random number generator. This parameter may used to repeat or not to repeat an experiment.

5.1.2 Implementation

The `gentrain` program outputs the input and output pairs generated for function f.

In pseudocode the `gentrain` program can be described as follows (After `Pre:` preconditions and after `Post:` the postconditions of the program is listed in terms of the input and output variables):

```
gentrain(seed, number_of_examples, input_dimension,
         input_lower_limit, input_upper_limit),
         f() )
int seed, number_of_examples, input_dimension ;
float input_lower_limit, input_upper_limit ;
/*Pre : number_of_examples>0, and input_dimension>0 and         */
/*      f: X^input_dimension -> Y^output_dimension  where       */
/*      float(X), float(Y) and                                  */
/*      input_lower_limit<= X <= input_upper_limit              */
/*      and output_dimension is determined by f().              */

/*Post: number_of_examples examples of f() generated and printed*/

begin
int output_dimension, i ;
float x_i[input_dimension], y_i[output_dimension] ;

   initialize_random_number_generator(seed) ;
   for (i=0; i<number_of_examples; i++)
   begin
      x_i = create_random_input_vector(input_dimension,
                 input_lower_limit, input_upper_limit) ;
      y_i = f(x_i) ;
      output(x_i, y_i) ;
   end
end
```

Figure 13 shows the interaction between the `gentrain` and `train` programs and their communications with their environment. It also shows how a typical run of generating training and validation sets by `gentrain` and then running `train` would proceed.

If the examples of the function are available, then there is no need to use the `gentrain` program.

5.2 `train` Program

The `train` program teaches a set of hints $H_0, \ldots, H_M$ to a randomly initialized feed-forward neural network with a fixed architecture, according to a

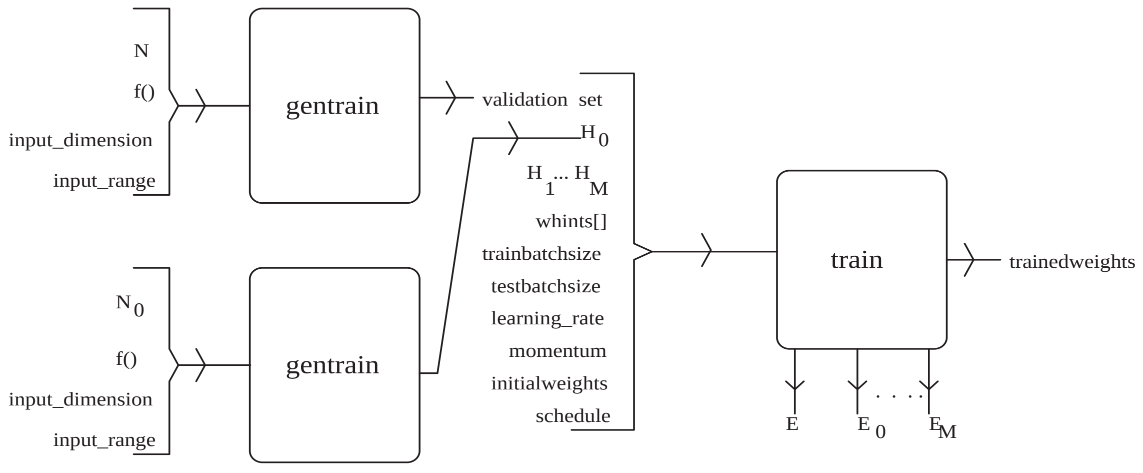

Figure 13: The interaction between the `train` and `gentrain` programs

schedule and given emphasis for each hint. The examples hint, cyclic shift, evenness, and binary hints, and rotation and maximum error schedules, are available with the program. Additional hints or schedules can be added easily. `train` uses the backpropagation [20] algorithm as the optimization technique.

The program starts with random initial weights and performs gradient descent on errors on hints; it outputs the generalization error and errors on each hint $E_0, E_1, \ldots, E_M$ as it does the descent and stops and outputs the weights of the network which has been trained after a certain number of descents (`pass`es) are performed.

5.2.1 Inputs and Outputs

- training set: Examples of f which will be used to teach it to the network. Training set is the examples hint $H_0 = \{i : 1, \ldots, N_0 : (x_i, f(x_i))\}$.
- validation set: Examples of f which will be used to calculate an estimate of the generalization error E. Validation error may be used to determine when to stop training.

- `learning rate` (η): The step size in the backpropagation algorithm. The weights are changed according to:

$$w_{ijk}^{t+1} = w_{ijk}^{t} - \eta \frac{\partial E_m}{\partial w_{ijk}} |_{w_{ijk}^t} \tag{38}$$

  In this formula, t denotes time (`pass`) and w_{ijk}^t denotes a weight w_{ijk} at time t. E_m is defined as the average error on N_m examples of the hint H_m:

$$E_m = \frac{1}{N_m} \sum_{k=1}^{N_m} e_m \tag{39}$$

  In this expression, e_m represents the error on a random example of H_m.

- `momentum`: A backpropagation parameter that may be used to speed up learning. The backpropagation learning rule with momentum is:

$$w_{ijk}^{t+1} = w_{ijk}^{t} - \eta \frac{\partial E_m}{\partial w_{ijk}} |_{w_{ijk}^t} + \alpha(w_{ijk}^t - w_{ijk}^{t-1}) \tag{40}$$

  where α is the momentum.

- `w[][][]`: Weights of the network. The network is a feed-forward multilayer neural network with sigmoid threshold functions and floating point number weights. `w[i][j][k]` denotes the weight from the `j`th unit in the `i`th layer (i : 0..`nlayer -1`) to the `k`th unit in the `i+1`st layer. Weights are randomly initialized to floating point numbers from [`w_lower_limit`, `w_upper_limit`].

- `nlayer, nunits[]`: Number of layers excluding the input layer and number of units at each layer in the network, respectively.

- `maxpass`: Number of `pass`es after which the program stops and outputs the weights of the network. A `pass` is defined as one descent on the objective function E_m.

  There are at least two possible modes of descent: **(i)** Batch Mode: In this mode the derivative $\frac{\partial E_m}{\partial w_{ijk}}$ is calculated and descent on E_m is done at once, and this is called a `pass`. **(ii)** Sequential Mode: In this mode, the derivative $\frac{\partial e_m}{\partial w_{ijk}}$ is calculated, and weights are modified according to this derivative. This descent, repeated N times makes up a `pass`.

For small learning rate sequential and batch modes give the same results. The program supports sequential mode only. However, batch mode may be necessary in the future, when different optimization algorithms are used.

- `trainbatchsize`: Number of examples of a hint to be taught at a `pass` (N_m in equation (39) above). An equal number of examples of each hint are taught.
- `testbatchsize`: Number of examples of e_m (or f to be picked from the validation set) used to determine an estimate of validation error for f or a hint H_m.
- `schedule`: The index of the schedule to be used during the run.
- `whints[M]`: Emphasis given to each hint. `whints[i]` corresponds to α_i in section 2. If `whints[i]` is zero, E_i has no effect on generalization error, and hence H_i is not taught.

  If the emphasis on each hint, i.e. `whints[i]` is known, a rotation schedule can be expressed in at least 3 different ways:

  - Descending on a number of examples N_i of H_i when H_i will be taught, and choosing N_i proportional to `whints[i]`.
  - Having different learning rates η_i for each H_i in such a way that η_i is proportional to `whints[i]`.
  - Having constant number N of examples to descend on and constant learning rate η for descend on each hint, but descending on H_i a number of times proportional to `whints[i]`.

  In the `train` program, for the implementation of rotation schedules the third strategy is used.

  If the schedule is the maximum error, then all `whints[i]`$*E_i$ are compared, and a pass is made on the hint having the maximum `whints[i]`$*E_i$.

All the input parameters above can be given as command line parameters. If an input is not given, then it is initialized to a consistent default value.

`train` program outputs `pass` number versus E_i for all hints as training continues. When `pass` exceeds `maxpass` it prints the final values of weights `w` and the execution stops.

5.2.2 Implementation

Implementation of Hints: Each hint H_i is implemented by a function `hinti()`. If it is called with argument `0`, the `hinti()` function outputs the value of e_i for one example without teaching it; otherwise it descends once on an example of H_i in the weight space, and then it returns the value of e_i before the descend.

For H_0 (the examples hint), an example of the hint is produced by chosing an element from the training set (H_0 input to the program) uniformly. For other hints, either a set of examples of the hint, like the training set, or a function which can generate an example of the hint must be provided. When the example of the hint is available, `hinti()` calculates the error on this example with the current weights of the network, and if, required, makes a backpropagation step on this error.

Implementation of Schedules: Each schedule is implemented by a function `schedulei()`. `schedulei()` has access to: `pass`, `w` (weights of the network), the `hinti()` functions (and hence information on all E_i's), and `whints[]`. `schedulei()` returns the hint it has chosen to be taught.

In pseudocode the `train` program can be described as follows:

```
train(learning_rate, momentum,
      maxpass, pass, trainbatchsize, testbatchsize,
      whints[M], schedule,
      w, w_lower_limit, w_upper_limit,
      nlayer, nunits, seed)
float learning_rate, momentum ;
int   maxpass, pass,
      trainbatchsize, testbatchsize,
      whints[M], schedule ;
float w[MAXLAYER][MAXUNIT][MAXUNIT],        /*Weights of network*/
      w_lower_limit, w_upper_limit;           /*range of weights*/
int nlayer,                      /*number of layers in the network*/
    nunits[MAXLAYER],               /*number of units at each unit*/
    seed;                            /*for random number generator*/

/* Pre: schedule>=0 and nlayer>1 and nunits[i]>=1:i:0..nlayer-1 */
/*      trainbatchsize>0 and testbatchsize>0 and                */
/*      w_lower_limit < w_upper_limit.                          */
/*      functions to implement schedules and hints and any input*/
/*      required by a schedule or hint must be provided.        */
```

```
/*Post: w[][][] initialized to random values, and according to  */
/*      hint and schedule parameters maxpass-pass descents done */
/*      on examples of hints. At each pass the errors on hints  */
/*      printed.   After all descents done weights printed      */

begin
   for (; pass<maxpass; pass++){

      turn = (*(fschedules[schedule])(
               pass, w, nunits, nlayer, H_0,H_1,...,H_M) ;
      /* Assert: turn=index of hint to be taught*/

      for (i=0; i<trainbatchsize; i++)
         (*(fhints[turn]))(1) ;
      /* Assert: turn'th hint taught on trainbatchsize examples*/

      err = 0.0 ;
      for (i=0; i<testbatchsize; i++)
         err += test_err() ;
      err = err / testbatchsize ;
      output(err);
      /* Assert: Generalization error measured and printed*/

      for (j=0; j<M; j++)
      begin
         err = 0.0;
         for (i=0; i<testbatchsize; i++)
            err += (*(fhints[j]))(0);
         err = err/trainbatchsize;
         output(err);
      end
     /* Assert: Errors on each hint measured and printed*/
   end

   output(w) ;
   /* Assert: Trained weights have been printed*/
end
```

In the Appendix the entire code for the `train` and `gentrain` programs are given.

Some improvements to the `train` program that will make it more flexible are given in the following section.

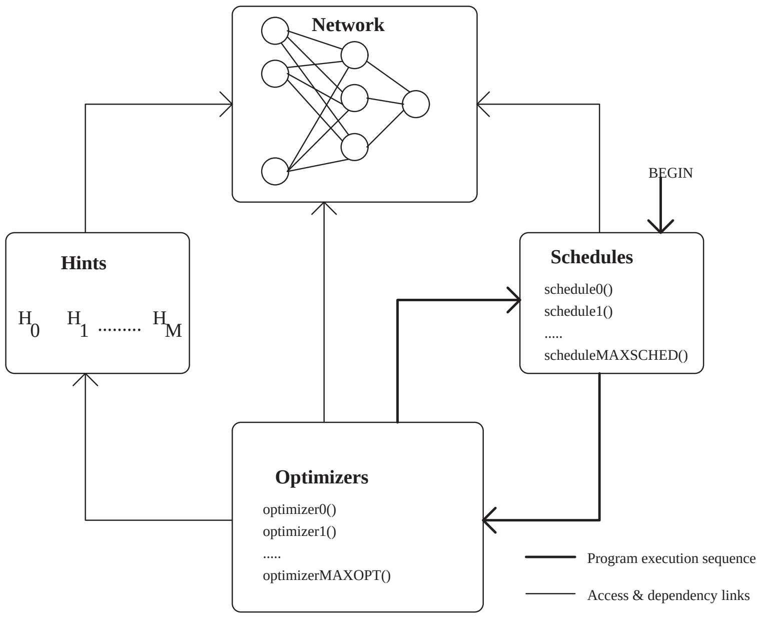

Figure 14: The `train` program execution.

6 Improved `train` program: `NNS`

The `train` program supports an adequate environment to simulate learning-from-hints in neural networks. However, in some aspects it needs to be improved. For example, it needs the the ability to handle:

- any estimate $\hat{E}$, in addition to the weighted sum formula that can be implemented through the maximum error schedule;
- different optimization techniques such as Adaptive Back Propagation (ABP) [4], QuickProp [14], TRUST [10]... etc. in addition to back-propagation (Backpropagation has problems of local minima and slow convergence. QuickProp and ABP may solve the speed problem and TRUST may solve the local minima problem.);
- feed-forward network architectures with different connections;
- different threshold (activation) functions for each unit (such as tanh or linear) in addition to sigmoid;

- binary, integer, and complex weights, in addition to floating point number weights;
- a graphical user interface in addition to the console;
- dynamic arrays (so that there will be no need to define constant maximum array sizes);
- reading weights from outside, in addition to randomly initializing them (this will enable to continue a run using the trained weights);

The NNS [1] (**N**eural **N**etwork **S**imulator) program whose description is given below is a step towards achieving these goals. It will be a general purpose program to simulate learning from hints, using different hints, estimates, schedules and optimization techniques.

6.1 Description of the NNS program

In this section the specifications (input/output arguments, global variables accessed, the job done) for functions in the NNS program, and the data types, the global variables, and the input/output file specifications are given. More detailed discussion of these can be found in [8].

The NNS program will have a defaults input and output file which are used to enter all the input parameters and to output the state of learning when it is terminated. Besides being more convenient than command line parameters, defaults input and output files can be used to continue a run.

6.2 Basic Data Types

The most important data type is the *Netw_type* which represents a neural network.

```
/*type definition for pointer to a function returning a float*/
typedef float (*PFF)();
typedef int   (*PFI)();
typedef void  (*PFV)();
```

[1] I would like to thank to Yaser S. Abu-Mostafa, Eric T. Bax, Mihail Iotov, Joe Sill, and Xubo Song for many discussions on the `train` and NNS programs.

```
typedef struct{
int numLayer, *numUnits ;
Weight_type ***w, ***wt_1 ;
/* w is the weight array at this time, wt_1 is the
   weight array at time t-1*/
byte ***connectivityMatrix ;

byte **thresholdFunctionIndex ;
} basic_Netw_type ;

typedef struct{
basic_Netw_type basic ;

float weightRange[2] ;

float **neuronOutputs ;

PFF **ptr_thresholdFunction, **ptr_dthresholdFunction,
    **ptr_d2thresholdFunction ;
/*Pointers to threshold functions and functions computing
  their 1st and 2nd derivatives with respect to their inputs*/

} Netw_type ;
```

basic_Netw_type is the structure that defines the hardware of the network, and *Netw_type* is the structure to be used as the network.

6.3 Global and Input Variables

In this section, we give the type declaration, default value and a brief description for global variables in the NNS program.

PFI $*$*ptr_schedules*; (Default: all nil):
Pointers to functions implementing schedules.

PFV $*$*ptr_optimizers*; (Default: all nil):
Pointers to functions implementing optimizers.

PFV $*$*ptr_hints*; (Default: all nil):
Pointers to functions implementing hints. Each of these functions provide the error(e_i) on an example of the hint and its derivatives with respect to weights.

int $schedulexx()$;:
xx'th schedule function.

void $optimizerxx()$;:
xx'th optimization function.

void $hintxx()$;:
xx'th hint function. Each of these functions provide the error(e_i) on an example of the hint and its derivatives with respect to weights. Any training or validation set examples provided as inputs to the program should be read by these functions.

float $***Hi, ***tHi$; (Default: all 0):
Arrays containing training and validation set examples for each hint.

int M; (Default: 3):
Maximum number of hints used excluding the examples hint. This variable must be input first because a many things depend on it.

int *optimization*; (Default: 0):
The index of the optimization technique to be used.

int *schedule*; (Default: 0):
The index of the schedule being used. (Different schedules can be used at one run by changing the value of this variable.)

int $*whints$; (Default: $[1, 0, 0...]$):
The weight given to each hint. If $whints[i] = 0$ then the hint is not taught. (*whints* may be modified to become a floating point array, instead of int.)

int $*hints\_teach$; (Default: $[0, -1]$):
A list of hints that will be taught during the run. The indices of all hints with nonzero $whints[i]$ are in *hints_teach* array.

int $*ohints$; (Default: $[1, 0, 0...]$):
If $ohints[i] = 1$ then the training (and if applicable validation) error for this hint is printed during training.

int $*hints\_print$; (Default: $[0, -1, ...]$):
A list of hints whose training (and the validation if applicable) errors will be printed during the run. If $hints\_print[i] = j$ then the i'th hint to be printed is j.

A convenient method of setting $whints, *hints\_teach, *ohints, *hints\_print$ arrays is: Request the user to enter *hint_index* *weight* pairs (either in defaults file or command line). According to these pairs set $whints$, $*hints\_teach$, $*ohints$, $*hints\_print$ as follows:

- If i is entered as $hint\_index$:
  - $whints[i] = |weight|$,
  - if $weight \geq 0$ then $ohints[i] = 1$ (print),
  - if $weight < 0$ then $ohints[i] = 0$ (do not print).
- If i is not entered as $hint\_index$:
  - If no other hint indices have been entered, either set $whints[0] = 1$ and $ohints[0] = 1$ and set all other entries of $whints$ and $ohints$ to 0;
  - else set $whints[i] = 0$ and $ohints[i] = 0$. Once $*whints$ and $*ohints$ are set $*hints\_teach$ and $*hints\_print$ can be set according to them.

int $pass$; (Default: 0):
i) One descent over an objective function could be counted as a pass, or epoch. `pass` indicates the amount of effort spent till a solution is reached. So instead of pass, some other criteria, such as **ii)** CPU time spent may be used. Especially when different optimization techniques are used, `pass` number may not be a good scale for comparison.

int $maxpass$; (Default: 1000):
The `pass` number after which the program terminates. By using the defaults output file from a run as the defaults input file for the next run and specifying a larger number for the maxpass one should be able to run the experiment for additional passes.

int $errorOutputTic$; (Default: 1):
Denotes after how many passes the program outputs errors on different hints.

float $**optimizer\_parameters$; (Default:[1.0,0.0][0.0...]):
The parameters needed by an optimization algorithm. The usual interpretation for backpropagation could be (assuming that usual backprop is the 0'th optimization algorithm):
$optimizer\_parameters[0][0]$ = learning rate,
$optimizer\_parameters[0][1]$ = momentum.

The meaning of $optimizer\_parameters[][]$ can be different for each optimizer, the related optimization function should keep track of them.

int $trainbatchsize, testbatchsize$; (Default: 20, 100):
Same as in the `train` program.

Netw_type $*netw$; (Default: to be specified):
Variable declaration for the networks. Since this describes an array of networks, any number of networks can be used in the program. The user should enter the number of networks (Num_Net default value 1). For parallel applications, or an application of genetic algorithms to neural networks, more than one network may be necessary.

char $*hintFname[2][20]$;
(Default: hintFname[0][0]="train00.inp", hintFname[0][1]="test00.inp"): The names of the files containing training and (if applicable) validation sets for each hint.

float $***H_i, ***tH_i$;
(Default: all 0): If there are training and validation input files for the jth hint, then the examples in these files are read into arrays $H_i[j]$ and $tH_i[j]$ respectively when $hint$**j**() function is called the first time. Each example (either of training or validation set) is an entry $H_i[j][k]$ which is a vector of floating point numbers. The function $hint$**j**() determines the meaning of these numbers.

char $defaultsInputFname[20]$;
(Default: ".NNSDefaults.I.**hintFname.time**")
char $defaultsOutputFname[20]$;
(Default: ".NNSDefaults.O.**hintFname.time**"):
The names of the defaults input and output file names.

The values of the variables in the program are set according to the following steps: i) Initialize to defaults within the program; ii) Read the defaults input file and initialize any variables specified there; iii) Read the command line options and initialize any variables specified.

If some command line parameters are entered before the name of the defaultsInputFile steps ii) and iii) may be interleaved.

The defaults input file contains a set of variables and their initial values which the user wants to use to run the program. The defaults output file records a set of variables, their values and the command line options (if any) entered, before the program terminates.

If neither input nor output defaults file name is specified, then the default output file name is .NNSDefaults.O.**hintFname.time** where *hintFname* is the variable defined above. If "dfname" is the defaults input file name specified, then "dfname.O.**time**" is the name of the defaults output file. (**time** denotes the system time written down in a suitable form.)

6.4 Main Program

```
/*Types for pointers to functions returning integer,float,void*/
typedef int (*PFI)();
typedef float (*PFF)();
typedef void (*PFV)();

/****************************************************************/
/*                                                              */
/*                     GLOBAL VARIABLES                         */
/*                                                              */
/****************************************************************/

int schedule00(), schedule01(),....schedule99() ;
PFI ptr_schedules[]={schedule00,....,schedule99} ;

void optimizer00(int hint, float params[]),
     optimizer01(int hint, float params[]),
     .....
     optimizer99(int hint, float params[]);
PFV ptr_optimizers[]={optimizer00,...,optimizer99};

void hint00(int mode, hint, float *e,
          float ***dei_dw, float ******d2ei_dw2),
     hint01(int mode, hint, float *e,
          float ***dei_dw, float ******d2ei_dw2),
     .....
     hint99(int mode, hint, float *e,,
          float ***dei_dw, float ******d2ei_dw2) ;
PFV ptr_hints[]={hint00,...,hint99} ;

/*Training and validation examples read from files for each hint*/
float ***Hi, ***tHi ;

int M;
int optimization, schedule,
    *whints, *hints_teach, *ohints, *hints_print,
    pass, maxPass, errorOutputTic;
float **optimizer_parameters ;

Netw_Type *netw ;

/*Some functions whose definitions will be given here*/

/*************************************************************/
/*                                                           */
/*                          MAIN                             */
/*                                                           */
/*************************************************************/
main(argc, argv)
int argc;
char *argv[];
{
char hintFname[20];
char defaultsInputFname[20], defaultsOutputFname[20] ;

FILE **fpo,**tfpo;/*pointers to training and valid'n out files*/

int seed ;
int hint ;

float *Ei, *tEi ;
/*arrays to store training and validation errors for hints    */

initialize(argc, argv, fpo, tfpo, hintFname, defaultsInputFname,
           defaultsOutputFname, &hint, &seed ) ;

/*Assert: All variables initialized to proper values           */

while (pass < maxPass){
   hint = (*ptr_schedules[schedule])() ;

   /*Assert:                                                   */
   /*hint is the index of objective function to be taught      */
   /*to the network when schedule is used to schedule          */

   (*ptr_optimizers[optimization])
      (hint, optimizer_parameters[optimization]) ;

   /*Assert:                                                   */
   /*one step of optimization of hint'th objective function,   */
   /*using optimizer has been made, training errors for hint'th*/
   /*objective function and the simple hints it uses are       */
   /*in the array Ei                                           */

   if (pass % errorOutputTic == 0){
     i = 0;
     while (print_hints[i] != -1){
      j = print_hints[i] ;
      estimate_err(j,1,0,&(tEi[j]),&(dEi_dw[j]),&(d2Ei_dw2[j])) ;
      estimate_err(j,0,0,&(Ei[j]),&(dEi_dw[j]),&(d2Ei_dw2[j])) ;

     /*Assert: Train&Test errors for jth hint in Ei[i], tEi[i]*/

      i++ ;
      }
      printerr(fpo, tfpo, Ei, tEi) ;
      /*Assert: Errors printed to proper output files          */
   }
}

print_defaults(defaultsOutputFname, hintFname,
               defaultsInputFname, hint, seed) ;
/*Assert: The values of all variables have been printed        */
/*        into file whose name is given by defaultsOutputFname */

closefiles(fpo, tfpo) ;
/*Assert: All opened files have been closed                    */
}
```

For each hint, schedule, and optimizer function, there should be empty functions which are to be filled in if necessary. In that way (up to 100 hint, schedule, and optimizer functions) there won't be a need to modify the array and pointer definitions regarding these functions.

6.5 Additional Functions

```
/************************************************************/
/*                                                          */
/*                        initialize()                      */
/*                                                          */
/************************************************************/
void
initialize(argc, argv, fpo, tfpo, hintFname, defaultsInputFname,
           defaultsOutputFname, hint, seed )
int argc ;
char *argv[] ;
FILE *fpo[], *tfpo[] ;
char hintFname[], defaultsInputFname[], defaultsOutputFname[] ;
int hint, seed ;
/*Pre : true                                                  */
/*Post: all global variables, and variables passed are        */
/*      initialized to their correct values,                  */
/*      either to their default values;                       */
/*      or default values read from defaultsInputFile;        */
/*      or values read from the command line.                 */
/*      The order in which the second or third type of        */
/*      initialization will take place depends on where the   */
/*      defaultsInputFileName has been given                  */
/*Global Accesses:                                            */
/*      M, optimization, schedule, hintsUsed, pass, maxPass,  */
/*      errorOutputTic, optimizer_parameters, netw, Hi, tHi   */

{
}

/*************************************************************/
/*                                                           */
/*                          schedulexx()                     */
/*                                                           */
/*************************************************************/
int
schedulexx()
/*Pre : 0<=xx<=99                                             */
/*Post: xxth scheduleing alg. is used to determine the index  */
/*      of the objective function to be taught. The scheduler */
/*      accesses the pass number, environment, and network    */
/*      (some or all of these variables) in order to determine */
/*      that objective function. The index of the objective   */
/*      function is returned                                  */
/*Global Access:                                              */
/*      M, hitsUsed, pass, netw, Hi, tHi                      */
{
}

/*************************************************************/
/*                                                           */
/*                          optimizerxx()                    */
/*                                                            */
/**************************************************************/
void
optimizerxx(hint, params)
int hint;
float params[] ;
/*Pre: 0<=xx<=99 and 0<=hint<=MaxHintIndex and               */
/*     params[xx] exist (if needed by the optimizerxx)        */
/*Post: One step optimization on hint'th objective function   */
/*     is done using the technique optimizexx, and the        */
/*     values of weights in w[][][] and/or wt_1[][][] have    */
/*     been modified.                                         */
/*Global Accesses:                                            */
/*     M, hitsUsed, pass, netw, Hi, tHi, hintxx() ;           */
{
If the the optimization technique needs error on hint'th hint,
or its 1st or second derivatives, using hint'th hintxx()
function determine the  value these errors or derivatives.

then combine them in the way optimization process requires.
(Using how many examples of the hint, batch or sequential
mode is decided and implemented by the optimizerxx())

Modify the weights in *netw.

}

/**************************************************************/
/*                                                            */
/*                          hintxx()                          */
/*                                                            */
/**************************************************************/
void
hintxx(mode, derivative, hint,  e,de_dw, d2e_dw2)
int mode, hint, derivative ;
float *e, ****de_dw, *******d2e_dw2 ;
/*Pre: 0<=xx<=99 and 0<=mode<=1 and 0<=derivative<=2 and      */
/*     e[i] exists if xx th hint uses info on ith hint.       */
/*Post:if (mode == 0)                                         */
/*      if (derivative == 0)                                  */
/*       the training error for xx th hint is found and       */
/*       written to e[xx].                                    */
/*      else if (derivative => 1) and (derivative<=2)         */
/*       the derivative'th and lesser order derivatives of    */
/*       training error on one hint example w.r.t. each weight */
/*       is found and returned in de arrays                    */
/*     else                                                    */
/*     if (mode == 1)                                          */
/*      if (derivative == 0)                                   */
/*       the teset    error for xx th hint is found and        */
/*       written to e[xx].                                     */
/*      else if (derivative => 1) and (derivative<=2)          */
/*       the derivative'th and lesser order derivatives of     */
/*       validation err on one hint example w.r.t. each weight */
/*       is found and returned in de arrays                    */
/*Global Access:                                               */
/*  M, hintsUsed, netw, Hi, tHi, forw(), back() ;              */
{
If xx is a simple hint then calculate e, or its derivatives
directly (using back(), forw, Hi, tHi), otherwise, use the
hintyy() where yy is a simple hint taking place in xx's
calculation.
}

/*************************************************************/
/*                                                           */
/*                    estimate_err                           */
/*                                                           */
/*************************************************************/
void
estimate_err(hintno,mode,derivative,tEi,dEi_dw,d2Ei_dw2)
int mode, hint, derivative ;
float *E, ****dE_dw, *******d2E_dw2 ;
/*Pre: hintno is  index of a hint whose training/validation err*/
/*     can be found. and 0<=mode<=1 and 0<=derivative<=2      */
/*Post: hintno'th hintxx() function called estbatchsize many   */
/*      times to get average estimate of training/validation er*/
/*      or their derivatives.                                  */
{
}

/*************************************************************/
/*                                                           */
/*                          forw()                           */
/*                                                           */
/*************************************************************/
void
forw(netw, inp)
```

```
Netw_Type *netw ;
float inp[] ;
/*Pre: netw exists, inp[] contains at least as many elements   */
/*     as the number of inputs to the network netw             */
/*Post:The values of neuron outputs for each neuron in the     */
/*     network are modified according to the inp[] vector      */
/*Global Access: None                                          */
{
}

/**************************************************************/
/*                                                            */
/*                          back()                            */
/*                                                            */
/**************************************************************/
void
back(netw, dy_dw)
Netw_Type *netw ;
float ****dy_dw[] ;
/*Pre : netw exists, dy_dw[][][][] has one location for each   */
/*      output unit and weight.                                */
/*Post:dy_dw[i][j][k][l] contains the derivative of the ith    */
/*      output unit wrto weight w[i][j][k] of the netw.        */
/*Global Access: None                                          */
{
}
```

7 Conclusions and Future Research

In this thesis, we have shown that using other hints in addition to the examples hint can improve the learning performance. Different schedules, and adaptive minimization as a general scheduling mechanism, have been presented. A method of finding an estimate $\hat{E}$ of the generalization error, and the experiments on this estimate have been discussed. When such an estimate is available, it can be optimized either using adaptive minimization or directly descending on the estimate. The specifications for programs that form an environment for learning-from-hints on neural networks, and improvements on these programs have also been discussed.

The future work on this subject can be grouped into two categories:

Theoretical:

- Having a complete set of desirable properties for an estimate of the generalization error.
- Finding other estimates, and testing them in terms of how well they reflect the generalization error and in terms of their effects when used as the objective function during training.
- Exploring the impact of different optimization algorithms on the performance of estimates.
- Exploring the impact of hints on learning speed and decrease in the VC-dimension [6] of G.

Applications:

- Rewriting the `train` program to produce the NNS (Neural Network Simulator) program in such a way that, together with the `gentrain` program, it makes up a complete neural network simulator that can use a library of optimization procedures, hints, and estimates.

  Some future improvements on NNS could be:

  - Given necessary inputs and objective functions $e_j : j : 0, \ldots, M$ for each hint, the ability to produce the programs for using these hints.
  - Given the all inputs, the ability to decide on which estimate to use.
  - Having a library of optimization routines, and the ability to decide on which one to use.
- Experimenting with real life applications, such as stock market prediction, pattern recognition, and medical applications.

A The C Code for the gentrain and train Programs.

In this appendix the C code for the train and gentrain programs that are described in section 5, is given. The purpose of having this appendix is to show an implementation example of including hints and schedules in learning from examples using feed-forward neural networks and backpropagation.

A.1 The gentrain Program

A.1.1 Main gentrain Program

```
/***************************************************************/
/*                                                             */
/*                         gentrain.c                          */
/*                                                             */
/***************************************************************/
/* This program generates input output vector pairs.          */
/* The function to compute output vectors from input vectors   */
/* should be in the same directory and in the file f.h.        */
/* The following are adjustable parameters of the program:     */
/* ----number of I/O pairs in the training set : N {100}       */
/* ----dimension of input vectors : I  {8}                     */
/* ----(output vector dimension (O) determined by function f)  */
/* ----output filename {train00.inp}                           */
/* ----range of input vectors :[downrange,uprange]{[-1.0, 1.0]}*/
/*                                                             */
/* max dimension for an input or output vector is MAXROWSIZE   */
/* which is used by the same name in backpropagation algorithm */
/* to denote the maximum number of units in any layer in net.  */
/*                                                             */
/* OUTPUT FILE FORMAT:                                         */
/* ------------------------------                              */
/* seed={random number seed used}                              */
/* {input vector dimension} {output vector dimension} {# pairs}*/
/* inp vector                                                  */
/* output vector                                               */
/* inp vector                                                  */
/* output vector                                               */
/* .....                                                       */
/* ------------------------------                              */
/* seed is just to see what was used to genrate the input and t*/
/* repeat things if necessary. 2nd and following lines are used*/
/* by the backpropagation algorithm which is in file "train.c".*/
/*                                                             */
/***************************************************************/

#include <stdio.h>
#include "math.h"
#include "f1.h"           /*contains f(inpvec,I,outvec,&O, LIM)*/
#define MAXROWSIZE 16        /*max dimension for I or O vectors*/

FILE *fp,                                 /*output file pointer*/
     *fopen();
float  drand48();        /*returns random numbers in [0.0, 1.0]*/
long now ;          /*used to initialize random number generator*/
float templim ;
/*-----------float  myrand(downrange, uprange)-----------------*/
/*Return a float  type random number in [downrange, uprange]   */
float
myrand(downrange, uprange)
float  downrange, uprange ;
{
float temp ;
while (fabs(temp =((drand48())*(uprange-downrange) + downrange))
         == templim) ;
return (temp) ;
}  /*myrand*/

/*---------------void geninp(inpvec, I, downrange,uprange)-----*/
/*generate and put into array inpvec an I dimensional vector
  whose components are in the range [downrange, uprange]*/
void
geninp(inpvec,I,downrange, uprange)
float  inpvec[] ;
int I ;
float  downrange, uprange ;
{
int i ;
   for (i=0; i<I; i++){
      inpvec[i]= myrand(downrange, uprange) ;
   }  /*for*/
}  /*geninp*/

/*---------------------void printvec(vec,dim)------------------*/
/*print the "dim" float  numbers in "vec" array into file pointed
```

```
by the file pointer "fp"*/
void
printvec(vec, dim)
float  vec[];
int dim ;
{
int i;
   for (i=0; i<dim; i++)
      fprintf(fp, " %f", vec[i]) ;
   fprintf(fp, "\n");
}  /*printvec*/

/*------------------------main(argc, argv)---------------------*/
main(argc,argv)
int argc;
char *argv[];
{
char outfile[256],                            /*the output file*/
     tempst[20], ch ;

int i,                               /*an ordinary loop counter*/
    N=100,                /*number of I/O pairs to be produced*/
    I=8,                               /*input vector dimension*/
    O,
    seed ,              /*seed for the random number generator*/
    erase_mode = 0 ;      /*Ask the user before deleting a file*/

float  inpvec[MAXROWSIZE],
       outvec[MAXROWSIZE],                          /*I/O vectors*/
       downrange = -1.0,
       uprange = 1.0 ,  /*upper and lower bounds for components*/
                                                   /* of I vector*/
       LIM ;            /*threshold limit. Inputted to funcn f*/
   strcpy(outfile,"train00.inp") ;
   /*37 because suggested so in UNIX man.*/
   seed = time(&now)%((time(&now)*37)%107+2);
   /*read in any arguments*/
   if (argc>1){
      sscanf(argv[1], "%s", tempst) ;
      if (tempst[0]=='q'){
    /*give info on options and exit*/
    printf("\nUSAGE:") ;
    printf("\n>>gentrain") ;
    printf("\n[-N #I/O pairs in training set{%d}]",N) ;
```

```
 printf("\n[-I input vector dimension{%d}]",I) ;
 printf("\n[-f output file name{%s}]",outfile) ;
 printf("\n[-s random number seed{now}]");
 printf("\n[-b lower bound of range of inp. vectors{%2.3f}]",
          downrange);
 printf("\n[-t upper bound of range of inp. vectors{%2.3f}]",
          uprange);
 printf("\n[-e ask to the user before erasing a file {%d}]\n",
          erase_mode) ;
     exit(0) ;
   }
   i=1 ;
   while (i<argc){
      sscanf(argv[i++],"%s",tempst) ;
      if (tempst[0] != '-'){
    printf("\nERROR IN INPUT FORMAT. Program exited with 1");
         exit(1) ;
      }
      switch (tempst[1]){
      case 'N':sscanf(argv[i++], "%d", &N) ; break ;
      case 'I':sscanf(argv[i++], "%d", &I) ;
               if (I>MAXROWSIZE) I = MAXROWSIZE ; break ;
      case 'f':sscanf(argv[i++], "%s", outfile) ; break ;
      case 's':sscanf(argv[i++], "%d", &seed) ; break ;
      case 'b':sscanf(argv[i++], "%f", &downrange) ; break ;
      case 't':sscanf(argv[i++], "%f", &uprange) ;
      case 'e':erase_mode = 1;
      }   /*switch*/
   }  /*while*/
}  /*if*/
/*open the output file*/
if ((fp = fopen(outfile, "r"))==NULL)
   fp = fopen(outfile, "w") ;
else{
   if (!erase_mode){
printf("\n Output file %s already exists. Overwrite(Y/N?):",
          outfile) ;
   scanf("%c",&ch) ;
   if ((ch == 'y') || (ch == 'Y'))
      fp = fopen(outfile, "w") ;
   else{
      printf("\n No computations done. Exited with 0 \n") ;
      exit(0) ;
   }
```

```
      }
      else
         fp = fopen(outfile, "w") ;
   }   /*else*/
   LIM = 0.67*sqrt((I*1.0)/3) ;
   templim = LIM ;
   /*initialize the random number generator*/
   srand48(seed);
   fprintf(fp,"/*seed: %d*/\n",seed) ;
   geninp(inpvec,I,downrange, uprange) ;
   f(inpvec, I, outvec, &O, LIM) ;
   fprintf(fp,"%d %d %d %f %f \n", I, O,N, downrange, uprange) ;
   printvec(inpvec, I) ;
   printvec(outvec, O) ;
   for (i=1; i<N; i++){
      /*Produce an input vector*/
      geninp(inpvec,I,downrange, uprange) ;
      /*Calculate the network output for this inp vector*/
      f(inpvec, I, outvec, &O, LIM) ;
      /*Print input and output vectors*/
      printvec(inpvec, I) ;
      printvec(outvec, O) ;
   }
   fclose(fp) ;
}  /*main*/
```

A.1.2 Included Function

```
/****************************************************************/
/*                                                              */
/*                             f1.h                             */
/*                                                              */
/****************************************************************/
/*                                                              */
/* This file contains the definition of the function f.         */
/* f(inpvec,I,outvec,&O) where                                  */
/* inpvec,outvec are  arrays of float                           */
/* I is input vector dimension (determined in gentrain.c file) */
/* O is the output vector dimension determined by f             */
/*                                                              */
/* see file gentrain.c for a complete picture.                  */
/*                                                              */
/****************************************************************/
```

```
void
f(inpvec,I,outvec,O, LIM)
float  inpvec[] ;
int I ;
float  outvec[] ;
int *O ;
float LIM ;  /*threshold limit*/
{
/* output vector is 1 dimensional and contains:                 */
/* 1.0 if sum of numbers in inpvec is in [-LIM, LIM]            */
/* 0.0 otherwise.                                               */
float  sum=0.0 ;
int i ;
   for (i=0; i<I; i++)
      sum += inpvec[i] ;
   if ((sum >= -LIM) && (sum <= LIM ))
      outvec[0]=1.0;
   else outvec[0]=0.0 ;
   (*O) = 1 ;     /*output vector is 1 dimensional*/
}   /*f*/
```

A.2 The `train` Program

A note: In the programs "validation error" and "test error" has been used interchangably to denote the validation error defined in section 1.2.

The program given here differs from the one used for experiments. The indices of schedules have been changed.

A.2.1 Main Functions

`train.c` contains the main file for the `train` program.

```
/**************************************************************/
/*                                                            */
/*                       train.c                              */
/*                                                            */
/**************************************************************/
/*This program reads input output pairs from an input file     */
/*and then trains a fully connected feedforward neural network */
/*using  backpropagation algorithm.                            */
```

```
/*"include.h" contains all #include's of files and also gives  */
/*which #include file contains which functions and their params*/
#include "include.h"

/*-------------------------------------------------------------*/
/*                                                             */
/*                     THE MAIN PROGRAM                        */
/*                                                             */
/*-------------------------------------------------------------*/

/*------------------main(argc, argv)-------------------------*/
main(argc,argv)
int argc;
char *argv[];
{
/*-------------------------------------------------------------*/
/*               INPUT OUTPUT FILE NAMES AND POINTERS          */
/*-------------------------------------------------------------*/
FILE *fpi[NUMINFILE],                    /*Input file pointers*/
                  /*0: training input file, 1: test input file*/

    *fpo[NUMOUTFILE] ;                  /*output file pointers*/
   /*0..MAXM-1     : error on the the ith hint               */
   /*MAXM          :  Error on the test set                  */
   /*1+MAXM..2*MAXM:a value computed for each hint by schedules*/

char fnamein[NUMINFILE][MAXLEN],            /*input file names*/
     fnameout[NUMOUTFILE][MAXLEN];           /*outputfilenames*/
                /*see fpi, fpo. File pointers&names correspond*/

float mine[2][MAXMINE];  /*mine[i][j]=E_j for 0<=j<=MAXM-1 and */
                         /*mine[i][MAXM]=E (Test err on f)     */
  /*  mine[0] contains errors when min. training err is reached*/
  /*mine[1] contains errors when a weighted min. err is reached*/
int disperr = DISPERR,/*# passes after which error is displayed*/
    seed ,                /*seed to the random number generator*/
    turn = 0,/*which hint(0..MAXM-1)'ll  be taught at this pass*/
    minpass[2] ;
   /* minpass[i]=pass number when errors in mine[i] are reached*/

   /*default initialization*/
   default_init(fpo, fnamein, fnameout, &seed) ;

   /*read any command line arguments*/
   read_params(argc,argv,fnamein,fnameout,
        &seed, &maxpass) ;

   /*open input and output files and initialize file pointers  */
   open_files(fnamein, fnameout, fpi, fpo) ;

   /*Fill in the inp, tinp, out, and tout arrays from inp.files*/
   read_inputs(fpi, fnamein) ;

   /*Init. weight vectors, random num gen, err, mine, minpass  */
   init_w_err_mine(seed, mine, minpass) ;

   while ((pass<(maxpass+1))){

      /*Fill in err. for all hint [0..MAXM-1] & validation err */
      test_all(mine, minpass) ;

      if (pass%disperr == 0)
     /*Print err array entries to stdout or output files*/
      print_err(fpo) ;

      /*Find who will be taught at this pass*/
      turn = find_turn(schedule) ;

      /*Teach the hint to be taught*/
      teach_hint(turn) ;  pass++ ;
   }   /*while*/


   /*Close files opened in the program*/
   close_files(fpi, fpo) ;

 /*Print the final weights, discrete errors and minimum errors*/
 /* and pass numbers they were reached on the stdout          */
   print_weights_err(mine, minpass) ;
}  /*main*/
```

`include.h` contains `#include`'s for all the files included in `train.c`.

All `schedulexx()` and `hintxx()` functions should be in directories `../hints` and `../schedules` respectively.

```
/*************************************************************/
/*                                                           */
/*                       include.h                          */
/*                                                          */
/************************************************************/
#include <stdio.h>
#include <math.h>
#include "declare.h"
/*-------------------------------------------------------------
Constants, variables and type definitions for the program
-------------------------------------------------------------*/


#include "utility.h"
/*-------------------------------------------------------------
float  fsqr(x)
float  x;
======================================
float
myrand(downrange, uprange)
float  downrange, uprange ;
======================================
float mean(ar, setsize)
float ar[];
int setsize ;
======================================
float variance(ar, setsize, m)
float ar[];
int setsize ;
float m ;
======================================
float
threshold(x)
float  x ;
-------------------------------------------------------------*/


#include "init.h"
/*-------------------------------------------------------------
void
default_init(fpo, fnamein, fnameout, seed)
FILE *fpo[NUMOUTFILE] ;
char fnamein[NUMINFILE][MAXLEN],
     fnameout[NUMOUTFILE][MAXLEN];
int *seed ;
======================================
void
initweights()
======================================
void
init_w_err_mine(seed, mine, minpass)
int seed;
float mine[2][MAXMINE];
int minpass[2] ;
======================================
void
read_params(argc,argv,fnamein, fnameout,
        seed, maxpass)
int argc;
char *argv[], fnamein[][MAXLEN], fnameout[][MAXLEN];
int *seed, *maxpass ;
--------------------------------------------------------------*/



#include "io.h"
/*--------------------------------------------------------------
void
read_inputs(fpi, fnamein)
FILE *fpi[NUMINFILE];
char fnamein[NUMINFILE][MAXLEN] ;
======================================
void
print_weights_err(mine, minpass)
float mine[2][MAXMINE];
int minpass[2] ;
======================================
void print_err(fp)
FILE *fp[NUMOUTFILE] ;
======================================
void
open_files(fnamein, fnameout, fpi, fpo)
FILE *fpi[NUMINFILE], *fpo[NUMOUTFILE] ;
char fnamein[NUMINFILE][MAXLEN],
     fnameout[NUMOUTFILE][MAXLEN];
======================================
void
close_files(fpi, fpo)
FILE *fpi[NUMINFILE], *fpo[NUMOUTFILE] ;
void
```

```
close_files(fpi, fpo)
FILE *fpi[NUMINFILE], *fpo[NUMOUTFILE] ;
---------------------------------------------------------------*/
#include "backprop.h"
/*---------------------------------------------------------------
void
forw(inpvect)
float inpvect[] ;
======================================
void
back(de_dy)
float de_dy[MAXROWSIZE];
=====================================
void
back2(de_dy, de_dy2)
float de_dy[MAXROWSIZE], de_dy2[MAXROWSIZE];
======================================
void
calc_de_dy(v1, v2, result)
float v1[MAXROWSIZE], v2[MAXROWSIZE], result[MAXROWSIZE] ;
======================================
int
choosepat()
======================================
float
calc_err(target, output, setsize)
float target[][MAXROWSIZE], output[][MAXROWSIZE] ;
int setsize ;
======================================
float
calc_one_err(target, output)
float target[MAXROWSIZE], output[MAXROWSIZE] ;
======================================
measure_t(inpa, outa, setsize)
float inpa[][MAXROWSIZE], outa[][MAXROWSIZE] ;
int setsize ;
---------------------------------------------------------------*/

#include "../hints/hint0.h"
#include "../hints/hint1.h"
#include "../hints/hint2.h"

/* Schedules indices have changed now
0: Rotation  (OLD: 0, 1, 3)
```

```
1: Random Rotation (OLD: 2,4)

Adaptive:
2: Maximum Error (OLD: 5,6) (also includes max. weighted err)
3: Random maximum error (New) (OLD? none)

*/
#include "../schedules/schedule0.h"  /* Rotation              */
#include "../schedules/schedule1.h"  /* Random Rotation       */
#include "../schedules/schedule2.h"  /* Maximum error         */
#include "../schedules/schedule3.h"  /* Random maximum error */

#include "main.h"
/*----------------------------------------------------------------
void
teach_hint(hintno)
int hintno ;
======================================
int
find_turn(schedule)
int schedule ;
=====================================
void
test_all(mine, minpass)
float mine[2][MAXMINE] ;
int minpass[2] ;
---------------------------------------------------------------*/
```

Files included in `include.h` : Now we give all the files included in `include.h` except the ones on schedules and hints.

```
/**************************************************************/
/*                                                            */
/*                        declare.h                           */
/*                                                            */
/**************************************************************/
/*------------------------------------------------------------*/
/*          DEFINE CONSTANTS USED IN THE PROGRAM              */
/*------------------------------------------------------------*/
#define MAXROWSIZE 128       /*max dimension for I or O vectors*/
#define MAXLAYER  3                  /*maximum # layers allowed*/
#define MAXSCHEDULE 4                 /*Max # schedules allowed*/
#define MAXPASS  1000      /*maximum # passes over training set*/
```

```
#define MAXSETSIZE 1000    /*Maximum #pairs in the training set*/
#define DISPERR  20   /*# passes after which error is displayed*/
#define MAXM 3     /*# hints that can be considered including f*/
#define MAXWHINT 100     /*Maximum value any whints[i] can take*/
#define NUMINFILE 2                     /*number of input files*/
#define NUMOUTFILE 1+2*MAXM           /*number of  output files*/
                      /*See main(), and fpo[] for a description*/
#define MAXLEN 80       /*Max length used for file name lengths*/
#define MAXMINE 1+MAXM /*# error measures in each row of mine  */
#define PRINT_SCHED    1   /*if schedule number is greater than*/
/*this number,err[MAXM+1..2*MAXM]'re printed on files or stdout*/
/*by setting PRINT_SCHED to <0, err[MAXM+1+i] (where ith hintis*/
/*used) can be printed, and by setting PRINT_SCHED to a value  */
/* >= MAXSCHEDULE none of err[MAXM+1..2*MAXM] canbe not printed*/
/* regardless of the schedule that is being used               */

typedef int (*PFI)();/*Defines pointer to a fn returning an int*/
typedef float (*PFF)();/*Define pointer to a fn returning float*/

/*Names of functions to be used for    ith schedule/hint should*/
/*be at    ith location of the arrays below,functions should be*/
/*declared as done here, and files contanining these functions */
/*should be included in file "include.h"                       */
int schedule0(), schedule1(), schedule2(), schedule3();

float hint0(), hint1(), hint2() ;

PFI fschedules[]={schedule0, schedule1, schedule2, schedule3};
PFF fhints[]={hint0, hint1, hint2} ;

/*-------------------------------------------------------------*/
/*      STANDARD FUNCTION DECLARATIONS                         */
/*-------------------------------------------------------------*/
float  drand48();        /*returns random numbers in [0.0, 1.0]*/
double pow() ;             /*To take x to the power y pow(x,y)*/
FILE *fopen();                                /*To open files*/
/*-------------------------------------------------------------*/
/*    DECLARATIONS FOR SOME OF THE FUNCTIONS USED IN PROGRAM   */
/*    used for reference wrto order of files only              */
/*-------------------------------------------------------------*/
void init_w_err_mine() ;
float measure_t() ;

/*-------------------------------------------------------------*/
```

```
/*                GLOBAL VARIABLES                             */
/*-------------------------------------------------------------*/
float  eta=0.5,                                 /*learning rate*/
     noweta= 0.5,               /*learning rate actually used*/
     alpha = 0.7,                                    /*momentum*/

     downrange = -1.0,    /*upper and lower bounds for initial */
     uprange = 1.0 ,                        /*values of weights*/

     inpdownrange = -1.0,
     inpuprange = 1.0,/*upper and lower bounds for inp. vectors*/

     inp[MAXSETSIZE][MAXROWSIZE],
     out[MAXSETSIZE][MAXROWSIZE] ,            /*inp/out vectors*/
     tinp[MAXSETSIZE][MAXROWSIZE],
     tout[MAXSETSIZE][MAXROWSIZE] ,      /*test inp/out vectors*/

  /*NETWORK ELEMENTS*/
     /*weights, current and old*/
     w[MAXLAYER-1][MAXROWSIZE][MAXROWSIZE],           /*weights*/
     wo[MAXLAYER-1][MAXROWSIZE][MAXROWSIZE],      /*old weights*/
     /*Bias values, current and old*/
     theta[MAXLAYER-1][MAXROWSIZE],/*threshold values for units*/
     thetao[MAXLAYER-1][MAXROWSIZE],/*old thres valuesfor units*/

     /*calculate delta for every unit except the input units*/
     delta[MAXLAYER][MAXROWSIZE],
     x[MAXLAYER][MAXROWSIZE] ,              /*outputs of units*/

     /*When backpropagationis used w/ an error function of two */
     /*variables (see back2() routine) weneed to store the netw*/
     /*outputs at x2[][] when the 1st input vector producing   */
     /*first output is presented to   netw (for each netw unit)*/
     /*delta2[]is used to keep track of delta valuesfor backp  */
     x2[MAXLAYER][MAXROWSIZE] ,
     delta2[MAXLAYER][MAXROWSIZE],          /*used like x2[][]*/

   /*BOOKKEEPING VARIABLES*/
   /*There is a 1-1 correspondence between err[] and fpo[] in  */
   /* main(), whatever is in err[i] is printed on the file     */
   /*pointed to by fpo[].  err is used for printing purposes   */
   /* only currently, and is modified at each pass to reflect  */
   /* E_0..E_MAXM-1, E, Q_0..Q_MAXM-1.  Q_is are variables set */
   /* in schedules and are used to monitor some variables      */
   /* attached with each hint (SEE ALSO: PRINT_SCHED constant  */
     err[NUMOUTFILE];

int nlayer=3,                        /* # layers in the network*/
    /*nlayer can't be <2, input and output layers are counted  */

    nunits[MAXLAYER],           /*number of units at each layer*/
    trainsetsize = MAXSETSIZE,     /*#items in the training set*/
    testsetsize = MAXSETSIZE,          /*#items in the test set*/
    trainbatchsize=20,    /*# examps of a hint taught at a pass*/
    testbatchsize= 50,       /*# examps of a hint to calcul.err*/
    estbatchsize=20,        /*a variable related with schedules*/

    schedule = 0,/*schedule to be used for training the network*/
    maxpass = MAXPASS,                        /*MAx no of passes*/
    pass = 0,                                   /*current # pass*/

    /*info in whints[] is used more efficiently by means of    */
    /* numhints, totwhint, phints[], bhints[]                  */
    whints[MAXM], /*each hintis assigned a weight(read_params()*/
    /*if whints[i]>0  teach and show error on phints[i]'th hint*/
    /*   whints[i]=0 don't teach, don't show                   */
    /*   whints[i]<0 don't teach, show                         */
    numhints= 1,         /*total # hints for which whints[i]>0 */
    totwhint =1 ,          /* \sum whhints[i] s.t. whints[i]>0 */
    thints[MAXM] ,  /*thint[0]=0, thints[i]=ith hint used      */
                    /*Example: if  hints 2 5 6 are used        */
                    /*thints={0,2,5,6,0,...}                   */

    ohints[MAXM] ;/*if ohints[i]=1 error on ith hint  outputted*/

/***********************************************************/
/*                                                         */
/*                      utility.h                          */
/*                                                         */
/***********************************************************/
/*contains the following functions:                        */
/*      fsqr(x)                                            */
/*      myrand(downrange, uprange)                         */
/*      mean(ar, setsize)                                  */
/*      variance(ar, setsize, m)                           */
/*      threshold(x)                                       */

/*---------------------------------------------------------*/
```

```
/*                                                              */
/*             GENERAL FUNCTIONS                                */
/*                                                              */
/*--------------------------------------------------------------*/
/*-------------float  fsqr(x)-----------------------------------*/
/*Pre: x is a float number                                      */
/*Post: "fsqr" is the square of x and is a float number         */
float  fsqr(x)
float  x;
{ return(x*x) ; }  /*fsqr*/

/*--------------float  myrand(downrange, uprange)--------------*/
/*Pre  : downrange<= uprange and  are float numbers            */
/*Post : "myrand" is a float rand number in [downrange,uprange]*/
float
myrand(downrange, uprange)
float  downrange, uprange ;
{
return ((drand48())*(uprange-downrange) + downrange) ;
}  /*myrand*/

/*---------float mean(ar, setsize)-----------------------------*/
/*Pre  : ar[0..setsize-1] exists, setsize>=1                   */
/*Post : "mean" is the average of ar[0..setsize-1]             */

float mean(ar, setsize)
float ar[];
int setsize ;
{
float sum = 0.0;
int i ;
   for (i=0; i<setsize; i++) sum += ar[i] ;
   sum = sum /setsize ;
   return(sum) ;
} /*mean*/


/*--------float variance(ar, setsize, m)-----------------------*/
/*Pre  :  ar[0..setsize-1] exists,m is the mean ar[0..setsize-1*/
/*        and setsize >= 1                                     */
/*Post : "variance" is the variance for ar[0..setsize-1] given */
/*       that "m" is the mean                                  */

float variance(ar, setsize, m)
float ar[];
int setsize ;
float m ;
{
float sum = 0.0;
int i ;
   for (i=0; i<setsize; i++) sum += fsqr(ar[i]-m);
   sum = sum / setsize ;
   return(sum) ;
} /*variance*/


/*-----------------float  threshold(x)------------------------*/
/*Pre  : x is a float number                                */
/*Post : "threshold"=1/(1+exp(-x))  (SIGMOID)               */
float
threshold(x)
float  x ;
{
#define nume 2.718281828
return(1.0/ (1.0+(float)(pow(nume, (double) (-1.0*x) )))) ;
}   /*threshold*/

/***********************************************************/
/*                                                         */
/*                       init.h                            */
/*                                                         */
/***********************************************************/
/*contains the following functions:                        */
/*    default_init(fpo, fnamein, fnameout, seed)           */
/*    initweights()                                        */
/*    init_w_err_mine(seed, mine, minpass)                 */
/*    read_params(argc,argv,fnamein, fnameout,seed, maxpass)   */
/*---------------------------------------------------------*/
/*                                                         */
/*           INITIALIZATION ROUTINES                       */
/*                                                         */
/*---------------------------------------------------------*/

/*---------void default_init(....)-----------------------------*/
/*Pre: fpo[0..NUMOUTFILE-1], fnamein[0..NUMINFILE-1],         */
/*     fnameout[0..NUMOUTFILE-1], nunits[0..nlayer-1] exists   */
/*Post: fpo[], nlayer, nunits[], fnamein[], fnameout[], *seed  */
/*      noweta initialized to default values                   */
```

```
/*GLOBAL REFERENCES: nlayer, nunits[],eta, noweta              */

void
default_init(fpo, fnamein, fnameout, seed)
FILE *fpo[NUMOUTFILE] ;                    /*Output file pointers*/
char fnamein[NUMINFILE][MAXLEN],               /*input file names*/
     fnameout[NUMOUTFILE][MAXLEN];              /*outputfilenames*/
int *seed ;
{
long now ;         /*used to initialize random number generator*/
int i ;

   nlayer = 3 ;
   nunits[0]=8; nunits[1]=3; nunits[2]=1;
   for (i=0; i<NUMOUTFILE; i++){
      fpo[i] =NULL ;
      strcpy(fnameout[i], "") ;
   }
   strcpy(fnamein[0],"train00.inp") ;
   strcpy(fnamein[1],"test00.inp") ;
   *seed = time(&now)%((time(&now)*37)%107+2);
   /*37 because suggested so in UNIX man.*/
   noweta = eta ;

   /*Set the default value of whints[i] */
   whints[0]=1 ;  /*Weight 1*/
   thints[0]=0 ;  /*Teach*/
   ohints[0]=1 ;  /*Output*/
   numhints = 1 ;
   totwhint = 1 ;
   for (i=1; i<MAXM; i++){
      whints[i]= 0 ;
      thints[i]= -1 ;
      ohints[i]= 0 ;
   }
}   /*default_init*/

/*---------------------void initweights()----------------------*/
/*Pre:downrange<=uprange w[][],wo[][],theta[][],thetao[][] exis*/
/*Post : w[]=wo[], theta[]=thetao[] initialized to random float*/
/*       numbers in the range [downrange, uprange]             */
/*GLOBAL REFERENCES: w[], wo[], theta[],thetao[],down/uprange  */
/*       nunits[], nlayer                                      */
void
```

```
initweights()
{
int i,j,k;
   for (i=0; i<nlayer-1; i++)
      for (k=0; k< nunits[i+1]; k++){
         for (j=0; j< nunits[i]; j++){
            w[i][j][k] = myrand(downrange, uprange) ;
            wo[i][j][k] = w[i][j][k] ;
         }
         theta[i][k] = 0.1 * myrand(downrange, uprange) ;
         thetao[i][k] = theta[i][k] ;
      }
}   /*initweights*/

/*------void init_w_err_mine(....)----------------------------*/
/*Pre: nunits[0..MAXLAYER-1], mine[][], minpass[] exist        */
/*Post: weight vectors, random num generator, err[], mine[][], */
/*      minpass[] are initialized                              */
/*GLOBAL REFERENCES: err,ohints[],testbatchsize,tinp[],tout[]  */
void
init_w_err_mine(seed, mine, minpass)
/*INP*/
int seed;
/*OUT*/
float mine[2][MAXMINE];
int minpass[2] ;
{
int i,j ;
   /*initialize the random number generator*/
   srand48(seed);
   /*Initialize the arrays w[], wo[], and theta[], thetao[]*/
   initweights();
   /*Initialize errors for hints 0..MAXM-1*/
   for (i=0; i<MAXM; i++){
      err[i] = 0.0 ;
      if (ohints[i]){
         for (j=0; j<testbatchsize; j++)
            err[i]+= (*(fhints[i]))(0) ;
         err[i] = err[i] / testbatchsize ;
      }
   }
   /*Initialize test error for f using the test data*/
   err[MAXM] = measure_t(tinp, tout, testbatchsize) ;
   /*FILLING IN THE mine and minpass arrays here              */
   /*see main() for a description of mine[] and minpass[]     */
   for (i=0; i<2; i++){
      for (j=0; j<MAXMINE; j++)
         mine[i][j] = err[j] ;
      minpass[i] = 1 ;
   }
}    /*init_w_err_mine*/

/*--------------------------------------------------------------*/
/*                                                              */
/*          COMMAND LINE INITIALIZATIONS                        */
/*                                                              */
/*--------------------------------------------------------------*/
/*The following are adjustable parameters of the program:

  NOTE:if nlayer is different than the default one, -d option
  must be entered before the -w option from the command line
-----d number of layers in the network 3
-----w number of units in each layer 8 3 1
-----r learning rate eta  0.5
-----m momentum alpha  0.6
-----n max # passes over training set 10000
-----i input file name train00.inp
-----o output file name "stdout"
-----x random number seed "now"
-----l lower bound of the range of weight vectors -1.0
-----u upper bound of the range of weight vectors 1.0
-----s schedule  0
-----v integer weights for hints 0..MAXM-1
-----e estbatchsize (used for estimation purposes)
-----t testbatchsize
*/


/*------void read_params(.....)--------------------------------*/
/*Pre: argv[0..argc-1], fnamein[], fnameout[] exists           */
/*Post:any parameters entered on the command line are read on  */
/*     appropriate variables                                   */
/*GLOBAL REFERENCES: nunits, nlayer, eta, alpha, downrange,    */
/*   uprange, schedule, thints, ohints,                        */
/*   whints, totwhint, numhints,noweta,eta                     */

void
```

```
read_params(argc,argv,fnamein, fnameout,
        seed, maxpass)
int argc;
char *argv[],
     fnamein[NUMINFILE][MAXLEN], fnameout[NUMOUTFILE][MAXLEN];
int *seed, *maxpass;
{
int i, j;
char tempst[80] ;

   /*read in any arguments*/
   if (argc>1){
      sscanf(argv[1], "%s", tempst) ;
      if (tempst[0]=='q'){
    /*give info on options and exit*/
    printf("\nUSAGE: ") ;
    printf("\n>>train");
    printf("\n[-d number of layers in the network{%d}]",nlayer) ;
    printf("\n[-w number of units in each layer{") ;
    for (i=0; i<nlayer; i++) printf("%d ", nunits[i]) ;
    printf("}]") ;
    printf("\n[-r learning rate eta {%2.3f}]",eta) ;
    printf("\n[-m momentum (alpha) {%2.3f}]", alpha) ;
    printf("\n[-n max # passes over training set{%d}]",*maxpass) ;
    printf("\n[-i input file name{%s}]",fnamein[0]) ;
    printf("\n[-o output file name{stdout}]") ;
    printf("\n[-x random number seed{now}]");
    printf("\n[-l lower bound of the range of weight vectors{%2.3f}]",
                    downrange);
    printf("\n[-u upper bound of the range of weight vectors{%2.3f}]",uprange) ;
    printf("\n[-s schedule an int in [0..%d], current:{%d}", MAXSCHEDULE-1, schedule) ;
    printf("\n        NEW SCHEDULES") ;
    printf("\n        Fixed:") ;
    printf("\n     0: Rotation  (OLD: 0, 1, 3)") ;
    printf("\n     1: Random Rotation (OLD: 2,4)") ;
    printf("\n    ") ;
    printf("\n        Adaptive:") ;
    printf("\n     2: Maximum Error (OLD: 5,6) (also includes max. weighted err)") ;
    printf("\n     3: Random maximum error (a new schedule) (OLD? none)") ;
    printf("\n      ") ;
    printf("\n        Estimate/Schedule") ;
    printf("\n     4: Mean Variance Check (7-8-9)") ;
    printf("\n     5: Sample Balanced Error (10, 11, 12)") ;
    printf("\n      ") ;
    printf("\n[-v int weight for each hint H_0...H_%d:  ", MAXM-1) ;
    for (i=0; i<MAXM; i++)
       printf("%d ", whints[i]);

    printf("\n[-e estbatchsize(# ex's of H_i used to estimate E_i){%d}",
                  estbatchsize);
    printf("\n[-t testbatchsize=#ex from test set used to find E{%d}\n",
           testbatchsize) ;
                 exit(0) ;
              }
              i=1 ;
              while (i<argc){
                 sscanf(argv[i++],"%s",tempst) ;
                 if (tempst[0] != '-'){
                    printf("\nERROR IN INPUT FORMAT.\n") ;
                    printf("Program exited with 1\n");
                    exit(1) ;
                 }
                 switch (tempst[1]){
                 case 'd':sscanf(argv[i++], "%d", &nlayer) ;
                          if (nlayer>MAXLAYER) nlayer=MAXLAYER;
                          else
                          if  (nlayer<2) nlayer = 2;
                          break ;
                 case 'w':j=0;
                          for (j=0; j<nlayer; j++){
                             sscanf(argv[i++], "%d", &(nunits[j])) ;
                             if (nunits[j]>MAXROWSIZE)
                                nunits[j]=MAXROWSIZE ;
                             else
                             if (nunits[j]<1) nunits[j] = 1 ;
                          }
                          break ;
                 case 'r':sscanf(argv[i++], "%f", &eta) ; break ;
                 case 'm':sscanf(argv[i++], "%f", &alpha) ; break ;
                 case 'n':sscanf(argv[i++], "%d", maxpass) ; break;
         case 'i':sscanf(argv[i], "%s", &(fnamein[0][0])) ;
                  sscanf(argv[i++], "%s", &(fnamein[1][1])) ;
                  fnamein[1][0] ='t';
                  break ;
         case 'o':sscanf(argv[i++], "%s", &(fnameout[MAXM][1])) ;
                  fnameout[MAXM][0] = 't' ;
                  break ;
         case 'x':sscanf(argv[i++], "%d", seed) ; break ;
```

```
      case 'l':sscanf(argv[i++], "%f", &downrange) ; break ;
      case 'u':sscanf(argv[i++], "%f", &uprange) ; break ;
      case 's':sscanf(argv[i++], "%d",&schedule) ; break ;
      case 'v':for (j=0; j<MAXM; j++){
                  sscanf(argv[i++], "%d",&(whints[j])) ;
                  fprintf(stderr, "%d \n", whints[j]) ;
               }
               break ;
      case 'e':sscanf(argv[i++], "%d",&j);
               if (j<=0) fprintf(stderr,
          "estbatchsize=%d<=0, changed to %d", j, estbatchsize) ;
               else estbatchsize = j;
               break;
      case 't':sscanf(argv[i++], "%d",&j);
               if (j<=0) fprintf(stderr,
        "testbatchsize=%d<=0, changed to %d", j, testbatchsize) ;
               else testbatchsize = j;
               break;
      }   /*switch*/
   }  /*while*/
}  /*if*/

/*whints[] are used by schedules and also in calculation of */
/* mine[1], to find the pass where a min. of a weighted err */
/* is reached            */

numhints = 0; totwhint = 0 ; j = 0;
for (i=0; i<MAXM; i++)
/*Teach and show*/
if (whints[i]>0){
   numhints++ ;
   totwhint += whints[i] ;
   thints[j++] = i ;
   ohints[i] = 1 ;
}
else
if (whints[i] < 0 ){
   ohints[i] = 1 ;
}
else{
   ohints[i] = 0 ;
}

noweta = eta ;
```

```
}   /*read_params*/

/************************************************************/
/*                                                          */
/*                       io.h                               */
/*                                                          */
/************************************************************/
/* input/output functions                                   */
/*                                                          */
/*      read_inputs(fpi, fnamein)                           */
/*      print_weights_err(mine, minpass)                    */
/*      print_err(fp)                                       */
/*      open_files(fnamein, fnameout, fpi, fpo)             */
/*      close_files(fpi, fpo)                               */
/*                                                          */
/* INPUT FILE FORMAT                                        */
/* ------------------------------                           */
/* seed={random number seed used}                           */
/* {input vector dimension} {output vector dimension} {# pairs}*/
/* inp vector                                               */
/* output vector                                            */
/* inp vector                                               */
/* output vector                                            */
/* .....                                                    */
/* ------------------------------                           */
/* seed is to see what was used to generate the input and to   */
/* repeat things if necessary. 2nd and following lines are used*/
/* by the backpropagation algorithm.                        */



/*------------void read_inputs(.....)-------------------------*/
/*Pre:fpi[0.NUMINFILE-1],inp,out,tinp,tout[0.MAXSETSIZE-1]exist*/
/*Post:inp[], out[] (training) are filled from fpi[0] and     */
/*     tinp[],tout[] (testing)  "    "     "   fpi[1]         */
/*     trainsetsize, testsetsize, nunits[0], nunits[nlayer-1]  */
/*     testbatchsize,inpdownrange,inpuprange are modified     */
/*     according to data read from input files,               */
/*GLOBAL REFERENCES: nlayer,nunits[],inp,out,tinp,tout        */
/*     trainsetsize, testsetsize, testbatchsize,inpdown/uprange*/

void
read_inputs(fpi, fnamein)
FILE *fpi[NUMINFILE];                    /*Input file pointers*/
```

```
char fnamein[NUMINFILE][MAXLEN] ;
{
int I, O, tI, tO,            /*#I/O units read from input files*/
    i, j ;
char ch ;

   if (fpi[0] != NULL){
      /*Reading the comment about random number seed here*/
      fscanf(fpi[0],"%c",&ch); fscanf(fpi[0],"%c",&ch);
      while (ch!='/') fscanf(fpi[0],"%c",&ch);
      fscanf(fpi[0],"%d %d %d %f %f\n",
             &I, &O, &trainsetsize, &inpdownrange, &inpuprange) ;
     /*if there is a mismatch between input file and arguments
     inputted, take the info in the input file as being correct*/
      if (I != nunits[0]) nunits[0] = I ;
      if (O != nunits[nlayer-1]) nunits[nlayer-1] = O ;
      if (trainsetsize > MAXSETSIZE) trainsetsize = MAXSETSIZE ;
   }

   if (fpi[1] != NULL){
      /*Reading the comment about random number seed here*/
      fscanf(fpi[1],"%c",&ch); fscanf(fpi[1],"%c",&ch);
      while (ch!='/') fscanf(fpi[1],"%c",&ch);
      fscanf(fpi[1],"%d %d %d %f %f\n",
             &tI, &tO, &testsetsize, &inpdownrange, &inpuprange);
      if ((fpi[0]!=NULL) & ((I != tI) || (O != tO))){
         fprintf(stderr, "ERROR!!!\n") ;
         fprintf(stderr, "training set in %s and\n", fnamein[0]);
         fprintf(stderr, "test set in %s and\n", fnamein[1]) ;
         fprintf(stderr, "contain incompatible data") ;
         fprintf(stderr, "program exited with 1") ;
         exit(1) ;
      }
      if (testsetsize > MAXSETSIZE) testsetsize = MAXSETSIZE ;
      if (testbatchsize > testsetsize)
         testbatchsize = testsetsize ;
   }

   if (fpi[0] != NULL){
  /*Read the I/O pairs from training input file into the arrays*/
      for (i=0; i<trainsetsize; i++){
         for (j=0; j<nunits[0]; j++)
            fscanf(fpi[0], "%f",&(inp[i][j])) ;
         for (j=0; j<nunits[nlayer-1]; j++)
```

```
            fscanf(fpi[0], "%f", &(out[i][j])) ;
      }
   }

   if (fpi[1] != NULL){
  /*Read the I/O pairs from the test input file into the arrays*/
      for (i=0; i<testsetsize; i++){
         for (j=0; j<nunits[0]; j++)
            fscanf(fpi[1], "%f",&(tinp[i][j])) ;
         for (j=0; j<nunits[nlayer-1]; j++)
            fscanf(fpi[1], "%f", &(tout[i][j])) ;
      }
   }
}   /*read_inputs*/


/*--------void print_weights_err(mine,minpass)-----------------*/
/*Pre : mine[0..1][0..MAXMINE-1]minpass[0,1],w[],theta[] exist */
/*Post: Contents of mine[][], minpass[] (min error and pass nos*/
/*      they were reached (mine[0]:min training err,           */
/*      mine[1]:min (training err+0.5*hint errors))            */
/* and the final values of weights&thresholds printed on stdout*/
/*GLOBAL REFERENCES: w, theta,nunits[], nlayer                 */

void
print_weights_err(mine, minpass)
float mine[2][MAXMINE];
int minpass[2] ;
{

int i, j, k ;

   printf("Min acc. to :passno") ;
   printf("%9s%9s", "E_0", "E") ;
   for (j =1; j<MAXM; j++)
      printf("%8s%d","E",j) ;
   printf("\n") ;
   for (i=0; i<2; i++){
      if (i==0) printf("Train ERR   : ") ;
      else      printf("Weighted ERR: ") ;
      printf("%5d ", minpass[i]) ;
      /*Output training and test errors (E_0 and E) firstly*/
      printf("%f %f ", mine[i][0], mine[i][MAXM]) ;
      for (j=1; j<MAXM; j++)
```

```
         printf("%f ", mine[i][j]) ;
      printf("\n") ;
   }
   /*Print the final values of weights and biases(thresholds)*/
   printf("Weights:\n") ;
   for (i=0; i<nlayer-1; i++){
      for (j=0; j<nunits[i]; j++){
         for (k=0; k<nunits[i+1]; k++)
            printf("  %f",w[i][j][k]) ;
         printf("\n") ;
      }
      printf("\n") ;
   }
   printf("Thresholds:\n") ;
   for (i=0; i<nlayer-1; i++){
      for (j=0; j<nunits[i+1]; j++)
         printf("  %f",theta[i][j]) ;
      printf("\n") ;
   }
}   /*print_weights_err*/

/*-------------void print_err(fp)------------------------------*/
/*Pre  : fp[0..NUMOUTFILE-1] exists                            */
/*Post : contents of err[] printed on stdout or files pointed  */
/*       by fp[] according to fp[] & ohints[]                  */
/*GLOBAL REFERENCES: schedule,ohints[],pass, err               */

void print_err(fp)
FILE *fp[NUMOUTFILE] ;
{
int i ;
   /*Print everything to be printed to stdout*/
   if (fp[MAXM] == NULL){
      printf("%d ", pass) ;
      /*Print training error and test error*/
      if (ohints[0])  printf("%f ", err[0]) ;
      printf("%f ", err[MAXM]) ;
      /*Errors on hints 1..MAXM-1*/
      for (i=1; i<MAXM; i++)
      if (ohints[i]) printf("%f ", err[i]) ;
      if (schedule>PRINT_SCHED)
         for (i=1+MAXM; i<=2*MAXM; i++)
            if (ohints[i-1-MAXM])
               printf("%f ", err[i]) ;
      printf("\n") ;
   }
   else{
      for (i=0; i<NUMOUTFILE; i++)
         if (fp[i] != NULL)
            fprintf(fp[i],"%d %f\n",pass,err[i]) ;
   }
}  /*print_err*/
/*---------------------------------------------------------------*/
/*                                                               */
/*         OPENING AND CLOSING FILES                             */
/*                                                               */
/*---------------------------------------------------------------*/

/*----------void open_files(fnamein, fnameout, fpi, fpo)-------*/
/*Pre:  fpi[0..NUMINFILE-1],fpo[0..NUMOUTFILE-1],              */
/*      fnamein[0..NUMINFILE-1], fnameout[0..NUMUTFILE-1],     */
/*      nunits[0..nlayer-1] exists                             */
/*Post:fpi[0..NUMINFILE-1],fpo[0..NUMOUTFILE-1] are initialized*/
/*     to NULL or to point to files depending on fname's and   */
/*     ohints[](if ohints[i] i'th hint error is outputted      */
/*GLOBAL REFERENCES: schedule, ohints[]                        */

void
open_files(fnamein, fnameout, fpi, fpo)
FILE *fpi[NUMINFILE], *fpo[NUMOUTFILE] ;
char fnamein[NUMINFILE][MAXLEN],            /*input file names*/
     fnameout[NUMOUTFILE][MAXLEN];           /*outputfilenames*/
{
int i;
   /*open the test input file*/
   if ((fpi[1] = fopen(fnamein[1], "r")) == NULL){
      fprintf(stderr, "CAN'T OPEN TEST INPUT FILE %s. ",
         fnamein[1]) ;
      fprintf(stderr, "Program exited with 1") ;
      exit(1) ;
   }
   /* Schedules 4 and 5 need E_0 and hence training set*/
   if (ohints[0]||schedule>=4) {
   /*open the input file*/
   if ((fpi[0] = fopen(fnamein[0],"r")) == NULL){
      if (ohints[0]==0)
     fprintf(stderr,"Schedule %d needs E_0 and training set\n") ;
      fprintf(stderr,"CAN'T OPEN INPUT FILE %s. ",fnamein[0]) ;
      fprintf(stderr,"Program exited with 1") ;
      exit(1) ;
   }   /*if*/
   }
   else fpi[0] = NULL ;

   /*Treating H_0 as yet another hint*/
   /*open the output file*/
   if (fnameout[MAXM][0]!=0) {
      /*Test file, for E*/
      fpo[MAXM] = fopen(fnameout[MAXM], "w") ;
      /*file pointer allocations for hint files if necessary*/
      for (i=0; i<MAXM; i++)
         if (ohints[i]){
         strcpy(fnameout[i], "h1") ;
         fnameout[i][1]='0'+i ;
         sscanf(fnameout[MAXM]+1, "%s", &(fnameout[i][2]));
         fpo[i] = fopen(fnameout[i], "w") ;
      }
      if (schedule>PRINT_SCHED){
         for (i=0; i<MAXM; i++)
         if (ohints[i]){
            strcpy(fnameout[1+MAXM+i], "Q1") ;
            fnameout[1+MAXM+i][1]='0'+i ;
            sscanf(fnameout[0], "%s", &(fnameout[1+MAXM+i][2]));
            fpo[1+MAXM+i] = fopen(fnameout[1+MAXM+i], "w") ;
         }
      }
   }
}   /*open_files*/


/*----------void close_files(fpi, fpo)------------------------*/
/*Pre : fpi[0..NUMINFILE-1],fpo[0..NUMOUFILE-1] exists        */
/*Post: input files and any opened output file i.e.           */
/*      (file pointer!=NULL) are closed.                      */
void
close_files(fpi, fpo)
FILE *fpi[NUMINFILE], *fpo[NUMOUTFILE] ;
{
int i ;
   for (i=0; i<NUMINFILE; i++)
   if (fpi[i] != NULL)
      fclose(fpi[i]) ;
   for (i=0; i<NUMOUTFILE; i++)
   if (fpo[i] != NULL)
         fclose(fpo[i]) ;
}   /*close_files*/

/***********************************************************/
/*                                                         */
/*                      backprop.h                         */
/*                                                         */
/***********************************************************/
/*   backprop main routines, and related functions:        */
/*   forw(inpvect)                                         */
/*   back(de_dy)                                           */
/*   back2(de_dy, de_dy2)                                  */
/*   calc_de_dy(v1,v2,result)                              */
/*   choosepat()                                           */
/*   calc_err(target, output, setsize)                     */
/*   calc_one_err(target, output)                          */
/*   measure_t(inpa, outa, setsize)                        */

/*---------------------------------------------------------*/
/*                                                         */
/*         FORWARD AND BACKWARD PROPAGATION MAIN ROUTINES  */
/*                                                         */
/*---------------------------------------------------------*/
/*----------------void forw(inpvect)-------------------------*/
/*Pre:  inpvect[0..nunits[0]-1],x[][], w[][][] exist        */
/*Post: Outputs of network when feeded with inpvec  calculated*/
/*      and are at x[nlayer-1][0...nunits[nlayer-1]-1]       */
/*GLOBAL REFERENCES: x, w, theta, nunits, nlayer            */
void
forw(inpvect)
float inpvect[] ;                            /*input vector*/
{
int i, j, k;
float  sum ;
   for (i=0; i<nunits[0]; i++)
      x[0][i] = inpvect[i] ;

   for (i=1; i<nlayer; i++)
      for (k=0; k<nunits[i]; k++){
         sum = 0.0 ;
         for (j=0; j<nunits[i-1]; j++)
            sum += x[i-1][j]*w[i-1][j][k] ;
         /*N.B.: Unlike the convention, I assume that bias     */
         /* weights are connected to +1 (convention is -1)     */
         sum += theta[i-1][k] ;
         x[i][k] = threshold(sum) ;
      }
} /*forw*/

/*----------------void back(de_dy)-----------------------------*/
/* BAckpropagation using an error function of one variable only*/
/*Pre  : Netwk outputs are at x[nlayer-1][0..nunits[nlayer-1]-1*/
/*       and (delta E)/ (delta x) i.e. the derivative of the   */
/*       error function E(x) with respect to x (where x is the */
/*       network output is in de_dy[]                          */
/*       x[][], w[][][], delta[][],theta[][] exist             */
/*Post : Accordingto delta_e_y values w, theta, wo, thetao are */
/*       modified,using backpropagation.                       */
/*GLOBAL REFERENCES: nunits[], nlayer, delta, noweta, alpha    */
/*       x, w, wo, theta, thetao                               */
void
back(de_dy)
float de_dy[MAXROWSIZE];    /*Derivative of E w.r.t. output var*/
{
int i, j, k ;
float  temp ;
   for (i=0; i<nunits[nlayer-1]; i++){
      delta[nlayer-1][i]=x[nlayer-1][i]*(x[nlayer-1][i]-1)*
                         de_dy[i] ;
   }
   for (i=nlayer-1; i>0; i--){
      for (k=0; k<nunits[i]; k++){
         for (j=0; j<nunits[i-1]; j++){
            temp = w[i-1][j][k] ;
            w[i-1][j][k]+=noweta*delta[i][k]*x[i-1][j]+
                          alpha*(w[i-1][j][k]-wo[i-1][j][k]) ;
            wo[i-1][j][k] = temp ;
         }   /*for k*/
         temp = theta[i-1][k] ;
         theta[i-1][k]+=noweta*delta[i][k]+
                        alpha*(theta[i-1][k]-thetao[i-1][k]) ;
         thetao[i-1][k] = temp ;
      } /*for j*/
      for (j=0; j<nunits[i-1]; j++){
         temp = 0.0 ;
         for (k=0; k<nunits[i]; k++)
            temp += delta[i][k]*wo[i-1][j][k] ;
         delta[i-1][j]=temp*x[i-1][j]*(1-x[i-1][j]) ;
      } /*for j*/
   }  /*for i*/
} /*back*/


/*---------------void back2(de_dy, de_dy2)---------------------*/
/*Backpropagation using an error function of two variables     */
/*Pre  : Netwk outputs are at x[nlayer-1][0..nunits[nlayer-1]-1*/
/*       and the outputs of the netw. corresponding to 2nd     */
/*       variable x2 are at x2[0...nunits[nlayer-1]-1]  with   */
/*       x[] and x2[] arrays containing outputs of each unit in*/
/*       the  network, and                                     */
/*       de_dy[], de_dy2[]                                     */
/*       contain the derivatives of the error func             */
/*       with respect to x and w.r.t. x2 resp.(x,x2:outputs )  */
/*       EXAMPLE: if E=(x-x2)^2 then  de_dy, and de_dy2        */
/*                2*(x-x2)=2*(x[nlayer-1]-x2[nlayer-1])        */
/*                2*(x2-x)=2*(x2[nlayer-1]-x[nlayer-1]) resp.  */
/*      N.B.:de_dy/y2 arrays have    one entry per output unit */
/*    x2[][],x[][],w[][][],delta[][],delta2[][],theta[][] exist*/
/*Post : According to de_dy[] and de_dy2[] values w,theta, wo, */
/*       thetao are modified.                                  */
/*GLOBAL REFERENCES: nunits[], nlayer, delta,noweta, alpha     */
/*       x, x2, w, wo, theta, thetao, delta2                   */
void
back2(de_dy, de_dy2)
float de_dy[MAXROWSIZE], de_dy2[MAXROWSIZE];
{
int i, j, k ;
float  temp ;

   for (i=0; i<nunits[nlayer-1]; i++){
      delta[nlayer-1][i]=x[nlayer-1][i]*(x[nlayer-1][i]-1)*
                         de_dy[i] ;
      delta2[nlayer-1][i]=x2[nlayer-1][i]*(x2[nlayer-1][i]-1)*
                         de_dy2[i] ;
   }
   for (i=nlayer-1; i>0; i--){
      for (k=0; k<nunits[i]; k++){
         for (j=0; j<nunits[i-1]; j++){
            temp = w[i-1][j][k] ;
            w[i-1][j][k]+=noweta*delta[i][k]*x[i-1][j];
            w[i-1][j][k]+=noweta*delta2[i][k]*x2[i-1][j]+
                          alpha*(temp-wo[i-1][j][k]) ;
            wo[i-1][j][k] = temp ;
         }   /*for k*/
         temp = theta[i-1][k] ;
         theta[i-1][k]+=noweta*delta[i][k];
         theta[i-1][k]+=noweta*delta2[i][k]+
                        alpha*(temp-thetao[i-1][k]) ;
         thetao[i-1][k] = temp ;
      } /*for j*/
      for (j=0; j<nunits[i-1]; j++){
         temp = 0.0 ;
         for (k=0; k<nunits[i]; k++)
            temp += delta[i][k]*wo[i-1][j][k] ;
         delta[i-1][j]=temp*x[i-1][j]*(1-x[i-1][j]) ;
         temp = 0.0 ;
         for (k=0; k<nunits[i]; k++)
            temp += delta2[i][k]*wo[i-1][j][k] ;
         delta2[i-1][j]=temp*x2[i-1][j]*(1-x2[i-1][j]) ;
      } /*for j*/
   }  /*for i*/
} /*back2*/

/*---------------------------------------------------------------*/
/*                                                               */
/*         BACKPROP DE / DY  CALCULATING ROUTINE                 */
/*                                                               */
/*---------------------------------------------------------------*/

/*---------void calc_de_dy(v1, v2, result)---------------------*/
/*Pre: v1[i], v2[i], result[i] where i in 0..MAXROWSIZE-1 exist*/
/*Post: result[i] = (v1[i] - v2[i])*2 which is derivative of   */
/*      error function w.r.t. v1 when E = (v1-v2)^2            */
void
calc_de_dy(v1, v2, result)
float v1[MAXROWSIZE], v2[MAXROWSIZE], result[MAXROWSIZE] ;
{
int i ;
   for (i=0; i<nunits[nlayer-1]; i++)
      result[i] = 2*(v1[i] - v2[i]) ;
}    /*calc_de_dy*/
```

```
/*-----------------------------------------------------------------*/
/*                                                                 */
/*          BACKPROP DETAIL ROUTINES                               */
/*                                                                 */
/*-----------------------------------------------------------------*/

/*---------int choosepat()-----------------------------------------*/
/*Pre  : trainsetsize>=1                                           */
/*Post :      0<= "choosepat" < trainsetsize and                   */
/*            "choosepat" is the index of a vector in inp array*/
/*GLOBAL REFERENCES: trainsetsize                                  */
int
choosepat()
{
   return (lrand48() % trainsetsize) ;
}   /*choosepat*/

/*-----------------------------------------------------------------*/
/*                                                                 */
/*          MEASUREMENT OF ERROR ROUTINES                          */
/*                                                                 */
/*-----------------------------------------------------------------*/

/*-----------float calc_err(target, output, setsize)-----------*/
/*Pre : target[0..setsize-1], output[0..setsize-1] exists      */
/*       setsize>=1, nunits[nlayer-1]>=1                       */
/*Post: "calc_err"=sum_i(sqr(target[i]-output[i])/setsize      */
/*GLOBAL REFERENCES: nunits, nlayer                            */
float
calc_err(target, output, setsize)
float target[][MAXROWSIZE], output[][MAXROWSIZE] ;
int setsize ;
{
int   i, j;
float sum = 0.0 ;
   for (i=0; i<setsize; i++)
      for (j=0; j<nunits[nlayer-1]; j++)
         sum += fsqr(target[i][j]-output[i][j]) ;
   sum = sum/(nunits[nlayer-1]*setsize) ;
   return sum ;
}

/*----------float calc_one_err(target, output)-----------------*/
/*Like calc_err but for one item only                          */
```

```
/*Pre  : nunits[nlayer-1]>=1                                  */
/*Post : "calc_err"=sum_i(sqr(target-output) (squared err)     */
/*GLOBAL REFERENCES: nunits[], nlayer                          */

float
calc_one_err(target, output)
float target[MAXROWSIZE], output[MAXROWSIZE] ;
{
int i ;
float sum = 0.0 ;
   for (i=0; i<nunits[nlayer-1]; i++)
      sum += fsqr(target[i]-output[i]) ;
   sum = sum / nunits[nlayer-1] ;
   return sum ;
}

/*----------measure_t(inpa, outa, setsize)---------------------*/
/*Pre : inpa[0..setsize-1], outa[0..setsize-1] exists          */
/*Post:sum squared error when elemets of inpa feeded to the net*/
/*      compared with desired outputs at outa array="measure_t"*/
/* GLOBAL REFERENCES: x, nunits, nlayer                        */
float measure_t(inpa, outa, setsize)
float inpa[][MAXROWSIZE], outa[][MAXROWSIZE] ;
int setsize ;
{
float output[MAXSETSIZE][MAXROWSIZE];    /*to keep netw outputs*/
int i,j ;
   for (i=0; i<setsize; i++){
      /*Present the pattern and see the outputs*/
      forw(inpa[i]) ;
      for (j=0; j<nunits[nlayer-1]; j++)
         output[i][j] = x[nlayer-1][j] ;
   }
   return(calc_err(outa, output, setsize))  ;
}

/*************************************************************/
/*                                                           */
/*                        main.h                             */
/*                                                           */
/*************************************************************/
/*High level functions called from main directly             */
/*         teach_hint(hintno)                                */
/*         find_turn(schedule)                               */
/*         test_all(mine, minpass)                               */

/*-------- void teach_hint(hintno)-----------------------------*/
/*Pre : 0<=hintno<MAXM                                         */
/*Post: trainbatchsize many examples of hintnoth hint is taught*/
/*      to the network                                         */
/*GLOBAL REFERENCES: trainbatchsize                            */
void
teach_hint(hintno)
int hintno ;
{
int i;
float eh ;
/*COMMENTARY  fprintf(stderr, " %d\n", hintno) ;*/

   eh = 0.0 ;
   for (i=0; i<trainbatchsize; i++)
      eh += (*(fhints[hintno ]))(1) ;
   eh = eh / trainbatchsize ;
}    /*teach_hint*/


/*----------int find_turn(schedule)----------------------------*/
/*Pre: true                                                    */
/*Post:"find_turn"=which hint [0..MAXM-1] ll be taught accordin*/
/*      to schedule, thints[] and state of the network         */
/* MAXSCHEDULE (a constant) determines the # schedules used.   */
int
find_turn(schedule)
int schedule ;
{
   if ((schedule<0) || (schedule>=MAXSCHEDULE)){
      fprintf(stderr, "ERROR!!!\n") ;
      fprintf(stderr,
      "Unknown schedule no: %d!!! Exiting...\n", schedule) ;
      exit(1) ;
   }
   return( (*(fschedules[schedule]))()) ;
}  /*find_turn*/

/*----------void test_all(mine, minpass) ----------------------*/
/*Pre: mine[][], minpass exist                                 */
/*Post: Errors on each hint are in err[0..MAXM-1], test error  */
/*      on f (calculated using test set data) is in err[MAXM]  */
/*      mine[][], minpass[] are modified if min E_0 (mine[0])  */
/*      or min weighted error (mine[1]) have been reached      */
/*GLOBAL REFERCES: numhints,thints[],whints[],testbatchsize,err*/
void
test_all(mine, minpass)
/*OUT*/
float mine[2][MAXMINE] ;
int minpass[2] ;
{
int i, j ;
float prev_minwe, minwe; /*min weighted(acc.to whints[]) errors*/
   for (i=0; i<MAXM; i++)
   if (ohints[i]){
      err[i] = 0.0 ;
      for (j=0; j<testbatchsize; j++)
         err[i]+= (*(fhints[i]))(0) ;
      err[i] = err[i]/testbatchsize ;
   }
   err[MAXM] = measure_t(tinp, tout, testbatchsize) ;

   /*FILLING IN THE mine and minpass arrays here*/
   minwe = 0.0 ;
   for (i=0; i<numhints; i++)
      minwe +=
     ((float)whints[thints[i]]/whints[0])*err[thints[i]];
   prev_minwe = 0.0 ;
   for (i=0; i<numhints; i++)
      prev_minwe +=
     ((float)whints[thints[i]]/whints[0])*mine[1][thints[i]];

   if (mine[0][0] > err[0]){
      for (j=0; j<MAXMINE; j++)
         mine[0][j] = err[j] ;
      minpass[0] = pass ;
   }
   if (prev_minwe>minwe){
      for (j=0; j<MAXMINE; j++)
         mine[1][j] = err[j] ;
      minpass[1] = pass ;
   }
}   /*test_all*/
```

A.2.2 `schedulexx()` Functions

```
/************************************************************/
/*                                                          */
/*                       schedule0.h                        */
/*                                                          */
/************************************************************/
/* Fixed Schedule (0)                                       */
/* Rotation (i'th active hint has whints[thints[i]](an integer)*/
/*        weight, and is taught proportional # times to whints*/
/*        Default: whints[]={1,1,1}                         */
/************************************************************/
int
schedule0()

/************************************************************/
/* Pre: thints[i] is e index(0..MAXM-1) of the ith active hint */
/*   &whints[thints[i]] is integer weight of thints[i]'th hint */
/*   &pass is the current pass(epoch) number incremented      */
/*                by one after each training batch          */
/*   &totwhint=\sum_j {whints[j]} for all active H_j        */
/*                                                          */
/*Post: schedule0 = max_j (\sum_k^j whints[thints[k]])<i      */
/*                                                          */
/*                  where                                   */
/*             i = (pass%totwhint) +1 ;                     */
/************************************************************/

{
int i, j, k=0 ;
static first=1, whints_hints[MAXWHINT*MAXM] ;

   if (first){
      for (i=0; i<numhints; i++){
         for (j=0; j<whints[thints[i]]; j++)
            whints_hints[k++]=thints[i] ;
      }
      first = 0;
   }
   return(whints_hints[pass%totwhint])  ;
}

/************************************************************/
/*                                                          */
/*                        schedule1.h                          */
/*                                                             */
/***************************************************************/
/* Fixed Schedule (1)                                          */
/* Random rotation (ith active hint has whints[thints[i]] (an  */
/*                  integer) weight, and has a probability of  */
/*                being taught proportional # times to whints  */
/*        Default: whints[]={1,0,0}                            */
/***************************************************************/
int schedule1()
/***************************************************************/
/*Pre:thints[i] is the index(0..MAXM-1) of the ith active hint */
/*   &whints[thints[i]] is integer weight of thints[i]'th hint */
/*   &totwhint=\sum_j {whints[j]} for all active H_j           */
/*                                                             */
/*Post: schedule1=max_j (\sum_k^j whints[thints[k]])<i         */
/*                                                             */
/*                where                                        */
/*           i = (lrand48()%totwhint) +1 ;                     */
/***************************************************************/

{
int i, j, k=0 ;
static first=1, whints_hints[MAXWHINT*MAXM] ;
static int taught[MAXM] ;

   if (first){
      for (i=0; i<numhints; i++){
         for (j=0; j<whints[thints[i]]; j++)
            whints_hints[k++]=thints[i] ;
         for (j=0; j<numhints; j++)
            taught[thints[j]] = 0;
      }
      first = 0 ;
   }
   return(whints_hints[lrand48()%totwhint]) ;
}

/***************************************************************/
/*                                                             */
/*                        schedule2.h                          */
/*                                                             */
/***************************************************************/
/*Adaptive Schedule (0)                                        */
```

```
/* Maximum Error                                              */
/*  The hint H_i which has maximum whints[i]*E_i is taught     */
/*  whints[i] is an integer weight that enables comparison of  */
/*  E_i's with different ranges, importance etc.              */
/*                                                            */
/*************************************************************/
int schedule2()
/*Pre:thints[i] is the index(0..MAXM-1) of the ith active hint */
/*************************************************************/
/*    whints[thints[i]] is integer weight of thints[i]'th hint */
/*    (*fhints[thints[i]]))(0) returns the value of error on   */
/*    hint thints[i] for one example.                         */
/*Post:schedule2=max_j (E[thints[j]])                         */
/*      where E[thints[j]] (sample error on hint thints[i]) is */
/*      determined by:                                        */
/*     (1/estbatchsize)*\sum_i=1^estbatchsize                 */
/*                {(*(fhints[thints[i]]))(0) ;                */
/*************************************************************/

{
int i, j, val ;
float max, ear, E[MAXM], Q[MAXM] ;

   for (i=0; i<numhints; i++)
     {
       ear=0.0;
       for (j=0; j<estbatchsize; j++)
       ear += (*(fhints[thints[i]]))(0) ;
       E[thints[i]] = ear/(estbatchsize*1.0) ;
       Q[thints[i]] = whints[thints[i]] * E[thints[i]] ;
     }

   /*Find the max Q[i] between the calculated Q[i]*/
   max = Q[thints[0]] ; val = thints[0] ;
   for (i=1; i<numhints; i++)
      if (max<Q[thints[i]]){
         max = Q[thints[i]]; val = thints[i] ;
       }
   return val ;
}

/*************************************************************/
/*                                                            */
/*                         schedule3.h                        */
/*                                                              */
/****************************************************************/
/*Adaptive Schedule (1)                                         */
/* Random Maximum Error                                         */
/*  Each hint H_i has a respective chance of whints[i]*E_i      */
/*  of being taught. whints[i] is an integer weight that        */
/*  enables comparison of E_i's with different ranges,          */
/*  importance etc.                                             */
/****************************************************************/
int schedule3()
/****************************************************************/
/*Pre:thints[i] is the index(0..MAXM-1) of the ith active hint */
/*    whints[thints[i]] is integer weight of thints[i]'th hint */
/*    (*fhints[thints[i]]))(0) returns the value of error on   */
/*    hint thints[i] for one example.                          */
/*Post:schedule2=max_j (E[thints[j]])                          */
/*      where E[thints[j]] (sample error on hint thints[i]) is */
/*      determined by:                                         */
/*     (1/estbatchsize)*\sum_i=1^estbatchsize                  */
/*                 {(*(fhints[thints[i]]))(0) ;                */
/****************************************************************/

{
int i,j ;
float ear, E[MAXM], Q[MAXM], sum_Q_i=0.0, rand_err ;


   for (i=0; i<numhints; i++){
      ear=0.0;
      for (j=0; j<estbatchsize; j++)
      ear += (*(fhints[thints[i]]))(0) ;
      E[thints[i]] = ear/(estbatchsize*1.0) ;
      Q[thints[i]] = whints[thints[i]] * E[thints[i]] ;
      sum_Q_i += Q[thints[i]] ;
   }
   /* Normalize wrto sum_Q_i*/
   for (i=0; i<numhints; i++)
      Q[thints[i]] = Q[thints[i]]/sum_Q_i ;

   /* Generate a random number in [0,1] uniformly*/
   rand_err = drand48() ;

   sum_Q_i = 0.0 ;
   for (i=0; i<numhints; i++){
      sum_Q_i += Q[thints[i]] ;
      if (sum_Q_i >= rand_err) {
         return(thints[i]) ;
      }
   }
}
```

A.2.3 hintxx() Functions

```
/**************************************************************/
/*                                                            */
/*                      hint0.h                               */
/*                                                            */
/**************************************************************/
/*Function itself as a hint                                   */
float
hint0(mode)
/* if mode == 0 then just test using hint on random examples   */
/*    mode == 2 then test using hard threshold                 */
/*    mode == 1 teach random hint examples                     */
/*    mode == 3 teach transformed (acc to hint)examples of f   */
int mode ;
{
int i, patno ;
float one_err,  hardx[MAXROWSIZE], de_dy[MAXROWSIZE] ;
   patno = choosepat() ;
   forw(inp[patno]) ;

   if (mode != 2)
      one_err = calc_one_err(out[patno], x[nlayer-1]) ;
   if ((mode==1)||(mode==3)){
   /*Backpropagate errors and modify network*/
      calc_de_dy(x[nlayer-1],out[patno], de_dy)  ;
      back(de_dy) ;
   }
   else
   /*Hard threshold*/
   if (mode == 2){
      for(i=0;i<nunits[nlayer-1];i++){
         if(x[nlayer-1][i]>0.5) hardx[i]=1.0 ;
         else         hardx[i]=0.0 ;
      }
      one_err = calc_one_err(out[patno], hardx) ;
   }
   return(one_err) ;
}

/**********************************************************/
/*                                                        */
/*                      hint1.h                           */
/*                                                        */
/**********************************************************/
/*                  cyclic shift hint                     */
float hint1(mode)
/* if mode == 0 then just test using hint on random examples   */
/*    mode == 2 then test using hard threshold                 */
/*    mode == 1 teach random hint examples                     */
/*    mode == 3 teach transformed (acc to hint)examples of f   */
int mode ;
{
float one_err, xvec[MAXROWSIZE], hxvec[MAXROWSIZE],
      hardx[MAXROWSIZE], hardx2[MAXROWSIZE],
      de_dy[MAXROWSIZE], de_dy2[MAXROWSIZE] ;
int i, j, k, patno ;
   if (mode != 3){
   /*Produce an input vector randomly*/
   for (i=0; i<nunits[0]; i++)
      xvec[i] = myrand(inpdownrange, inpuprange) ;
   }
   else{
      patno = choosepat() ;
      for (i=0; i<nunits[0]; i++)
         xvec[i] = inp[patno][i] ;

   }
   /*Definition of the hint is here*/
   for (i=0; i<nunits[0]; i++)
      hxvec[i] = xvec[(i+1)%nunits[0]] ;
   forw(xvec) ;
   /*copy outputs of units to temporary x2 vector*/
   for (j=0; j<nlayer; j++)
      for (k=0; k<nunits[j]; k++)
         x2[j][k] = x[j][k] ;
   /*forward the H(x) vector*/
   forw(hxvec) ;
   if (mode != 2)
```

```
      one_err = calc_one_err(x[nlayer-1], x2[nlayer-1]) ;
   if ((mode==1) || (mode == 3)){
      calc_de_dy(x[nlayer-1], x2[nlayer-1], de_dy) ;

     /*Using the advantage of having E=(x-x2)^2 and hence      */
     /* having derivative of E w.r.t. one var. being negative  */
     /* of the other derivative here. If Error function changes*/
     /* there may be need to write a new calc_de_dy() routine  */
      calc_de_dy(x2[nlayer-1], x[nlayer-1], de_dy2) ;

      /*Backpropagate errors and calculate modify network*/
      back2(de_dy, de_dy2) ;
   }
   else
   /*Hard threshold*/
   if(mode==2) {
      for(i=0;i<nunits[nlayer-1];i++) {
         if(x[nlayer-1][i]>0.5) hardx[i]=1.0 ;
         else       hardx[i]=0.0 ;
         if(x2[nlayer-1][i]>0.5) hardx2[i]=1.0 ;
         else       hardx2[i]=0.0 ;
       }
      one_err = calc_one_err(hardx, hardx2) ;
   }
   return(one_err) ;
} /*hints(mode)*/
/****************************************************************/
/*                                                              */
/*                       hint2.h                                */
/*                                                              */
/****************************************************************/
/*                     evenness hint                            */
/*                     evenness hint                            */
float hint2(mode)
/* if mode == 0 then just test using hint on random examples   */
/*    mode == 2 then test using hard threshold                 */
/*    mode == 1 teach random hint examples                     */
/*    mode == 3 teach transformed (acc to hint)examples of f   */

int mode ;
{
float one_err, xvec[MAXROWSIZE], hxvec[MAXROWSIZE],
      hardx[MAXROWSIZE], hardx2[MAXROWSIZE],
      de_dy[MAXROWSIZE], de_dy2[MAXROWSIZE] ;
```

```
int i, j, k, patno ;
   if (mode != 3){
   /*Produce an input vector randomly*/
   for (i=0; i<nunits[0]; i++)
      xvec[i] = myrand(inpdownrange, inpuprange) ;
   }
   else{
      patno = choosepat() ;
      for (i=0; i<nunits[0]; i++)
         xvec[i] = inp[patno][i] ;
/*printf("pass = %d, hint2 trainsformed", pass) ;*/

   }
   /*Definition of the hint is here*/
   for (i=0; i<nunits[0]; i++)
      hxvec[i] = -1.0 * xvec[i] ;
   forw(xvec) ;
   /*copy outputs of units to temporary x2 vector*/
   for (j=0; j<nlayer; j++)
      for (k=0; k<nunits[j]; k++)
         x2[j][k] = x[j][k] ;
   /*forward the H(x) vector*/
   forw(hxvec) ;
   if (mode != 2)
      one_err = calc_one_err(x[nlayer-1], x2[nlayer-1]) ;
   if ((mode==1)||(mode==3)){
      calc_de_dy(x[nlayer-1], x2[nlayer-1], de_dy) ;

     /*Using the advantage of having E=(x-x2)^2 and hence      */
     /* having derivative of E w.r.t. one var. being negative  */
     /* of the other derivative here. If Error function changes*/
     /* there may be need to write a new calc_de_dy() routine  */
      calc_de_dy(x2[nlayer-1], x[nlayer-1], de_dy2) ;

      /*Backpropagate errors and calculate modify network*/
      back2(de_dy, de_dy2) ;
   }
   else
   /*Hard threshold*/
   if(mode==2) {
      for(i=0;i<nunits[nlayer-1];i++) {
         if(x[nlayer-1][i]>0.5) hardx[i]=1.0 ;
         else       hardx[i]=0.0 ;
         if(x2[nlayer-1][i]>0.5) hardx2[i]=1.0 ;
         else       hardx2[i]=0.0 ;
        }
       one_err = calc_one_err(hardx, hardx2) ;
    }
    return(one_err) ;
}   /*hint2(mode)*/
```

References

[1] Y.S. Abu-Mostafa (1990), "Learning from Hints in Neural Networks," in *Journal of Complexity,* vol. **6**, pp. 192–198.

[2] Y.S. Abu-Mostafa (1993), "A Method for Learning from Hints," in *Advances in Neural and Information Processing Systems,* vol. **5**, pp. 73–80.

[3] Y.S. Abu-Mostafa (1994), "Learning from Hints," *Journal of Complexity,* vol. **10**, pp. 165-178.

[4] A. Atiya (1991), "Learning Algorithms for Neural Networks," Ph.D Thesis, California Institute of Technology, CA.

[5] E.B. Baum, D. Haussler (1989), "What Size Net Gives Valid Generalization," in *Neural Computation,* vol. **1**, pp. 151–160.

[6] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. Warmuth (1989), "Learnability and the Vapnik-Chervonenkis Dimension," in *Journal of the ACM,* **36**, pp. 929-965.

[7] Z. Cataltepe, Y.S. Abu-Mostafa (1993), "Estimating Learning Performance Using Hints," *Proceedings of the 1993 Connectionist Models Summer School,* M. Mozer *et. al.* (Eds.), Lawrence Erlbaum Associates, Publishers, Hillsdale, NJ. pp. 380–386. also to be presented at TAINN-III (1994), The Third Turkish Symposium on Artificial Intelligence and Neural Networks, METU, Ankara, Turkey.

[8] Z. Cataltepe (1994), "Neural Network Simulator Program (NNS): Specifications of Functions, (Global) Variables, and Input/Outputs," Research Report (January 10 1994).

[9] Z. Cataltepe (1994), "Neural Network Simulator Program (NNS): Objective Functions and Their Derivatives," Research Report (January 10 1994).

[10] B.C. Cetin (1993), *TRUST A New Global Optimization Methodology, Application to Artificial Neural Networks and Analog VLSI Implementation,* Ph.D Thesis, California Institute of Technology, CA.

[11] B.W. Char, *et. al.* (1991), *Maple V Library Reference Manual,* Springer Verlag, NY.

[12] M.M. Denn (1969), *Optimization by Variational Methods,* McGraw-Hill Inc. NY.

[13] R.O. Duda, P.E. Hart (1973), *Pattern Classification and Scene Analysis,* John Wiley & Sons, Inc., NY.

[14] S.E. Fahlman (1988), *An Empirical Study of Learning Speed in Back-Propagation Networks,* Technical Report, CMU-CS-88-162.

[15] R. Fletcher (1969), *Optimization Symposium of the Institute of Mathematics and Its Applications,* University of Reele, England, 1968, Academic Press, London.

[16] R. Hecht-Nielsen (1990), *Neurocomputing,* Addison-Wesley Publishing Co.

[17] K.P. Hertz, A. Krough, R. G. Palmer (1991), *Introduction to the Theory of Neural Computation,* Lecture Notes, vol. **1**, Santa Fe Institute Studies in The Sciences of Complexity.

[18] S.L.S. Jacoby, J.S. Kowalik, J.T. Pizzo (1972), *Iterative Methods for Nonlinear Optimization Problems,* Prentice-Hall Inc., Englewood Cliffs, NJ.

[19] M.L. Minsky, S.A. Papert (1969), *Perceptrons,* MIT Press, Cambridge, MA.

[20] D.E. Rumelhart, J.L. McClelland, R.J. Williams, (1986) “Learning Internal Representations by Error Propagation,” in *Parallel Distributed Processing,* (D.E. Rumellhart *et al.,* Eds.) MIT Press, Cambridge, MA., vol. **1**, pp. 318–362.

[21] G.A.F. Seber, C.J. Wild (1989), *Nonlinear Regression,* John Wiley and Sons, Inc. NY.

[22] S, M. Ross (1987), *Introduction to Probability and Statistics for Engineers and Scientists,* John Wiley & Sons Inc., NY.