Distributed Fault Tolerance in Optimal Interpolative Nets

Dan Simon, d.simon@ieee.org
Cleveland State University
1960 East 24th Street
Department of Electrical Engineering
Stilwell Hall Room 332
Cleveland, OH 44115

ABSTRACT

The recursive training algorithm for the Optimal Interpolative (OI) classification network is extended to include distributed fault tolerance. The conventional OI Net learning algorithm leads to network weights that are nonoptimally distributed (in the sense of fault tolerance). Fault tolerance is becoming an increasingly important factor in hardware implementations of neural networks. But fault tolerance is often taken for granted in neural networks rather than being explicitly accounted for in the architecture or learning algorithm. In addition, when fault tolerance is considered, it is often accounted for using an unrealistic fault model (e.g., neurons that are stuck on or off rather than small weight perturbations). Realistic fault tolerance can be achieved through a smooth distribution of weights, resulting in low weight salience and distributed computation. Results of trained OI Nets on the Iris classification problem show that fault tolerance can be increased with the algorithm presented in this paper.

Keywords: Optimal interpolative net, fault tolerance, constrained optimization, regularization.

I INTRODUCTION

One of the difficulties that a neural net trainer often faces is deciding how many neurons to use in the network. If too many neurons are used, training time may be much longer than necessary, and the resultant network may have poor generalization properties [1]. If too few neurons are used, the learning algorithm may not converge to a suitable configuration. It is clearly desirable to use a training method which intelligently and automatically generates the optimal number of neurons [2, 3].

One solution to this difficulty is the Optimal Interpolative (OI) Net [4]. The OI Net is a three layer classification network that grows only as many middle layer neurons as necessary to correctly classify the training set. The efficient recursive learning procedure presented in [5, 6] makes the OI Net an attractive architecture.

In the present paper we extend the OI Net learning algorithm to include distributed fault tolerance. Biological systems are inherently fault tolerant due to the distributed nature of computation and information representation [7]. Fault tolerance has also been touted as an inherent property of artifical neural systems. But this has often been taken for granted rather than being explicitly provided for in the learning method. Rather than being an inherent property of all neural networks, fault tolerance is a feature that will generally result only if explicitly accounted for in the architecture or learning algorithm. Various schemes have been proposed to increase fault tolerance in neural networks [8]. The more popular approach has been to train with neurons stuck on or stuck off during learning to give the network a greater ability to withstand such faults during operation [9]. This is the approach taken in [10] to incorporate fault tolerance into the OI Net learning algorithm. However, neurons that are stuck on or stuck off do not represent a realistic model of neural network faults. A more realistic model is small perturbations of the network weights from their trained values [11]. It has been suggested that for analog VLSI implementations of neural networks, the network weights will be inherently limited to a relative precision of up to 1% [12]. This will be the result of such things as component mismatch and changes in component threshold voltages. For digital implementations the relative precision of network weights will be limited by bit resolution. Many implemented hardware functions are approximations of desired mathematical functions, and tradeoffs need to be made between the need for functional accuracy and other factors (e.g., silicon area and manufacturing costs). Errors in hardware implementations of neural networks inevitably occur and are generally in the form of small signal perturbations, including weight errors.

It is therefore on this type of error that we concentrate in this paper—small weight perturbations rather than stuck neurons. This is the difference between the results presented in [10] and the results presented in this paper. The former paper gives tolerance to "hard" faults (i.e., faults that consist of neurons that are stuck on or stuck off). The present paper gives tolerance to "soft" faults (i.e., faults that consist of small signal perturbations at the neurons). A training algorithm that spreads the weights evenly throughout the network would lead to a useful and realistic fault tolerance. In this paper we explicitly attempt to distribute the weights evenly throughout the network and thus account for fault tolerance in the OI Net training algorithm. Section II reviews the architecture of the OI Net, and Section III provides a theoretical basis for distributed fault tolerance in the OI Net. Section IV presents a recursive learning algorithm for a fault-tolerant OI Net. Section V presents some simulation results, and Section VI presents concluding remarks.

II THE OPTIMAL INTERPOLATIVE NET

Suppose we are given a training set with q sets of input/output pairs. Each of the q training inputs $x^i \in R^n$ maps into one of m classes C_j. Let $y^i \in R^m$ be the desired output corresponding to x^i. The output y^i is defined as $x^i \in C_j \Longrightarrow y^i = \delta_j$, where δ_j is the m-dimensional vector containing all zeros except for the jth element, which is one.

The OI Net consists of three layers of neurons. The first layer has n neurons, one for each component of the input. The second layer has p neurons, where p is a number which is chosen during training. The third layer has m neurons, one for each component of the output. The weight from the ith input neuron to the jth middle layer neuron is given by the element in the ith row and jth column of the $n \times p$ matrix V. The vectors $v^i \in R^n$, which comprise the columns of V, are called *prototypes* and are chosen from the training set inputs during the learning procedure. The activation function at each middle layer neuron is given by $\phi(s) = \exp(s/\rho)$ where ρ is a learning constant chosen by the user. The weight from the jth middle layer neuron to the kth output layer neuron is given by W_{jk}, where W is the weight matrix to be chosen during training by solving the following minimization problem.

$$\min_W \parallel Y - W^T G \parallel \Longrightarrow W = (GG^T)^{-1} G Y^T \in R^{p \times m} \tag{1}$$

where $\parallel \cdot \parallel$ refers to the Frobenius norm of a matrix (which is the square root of the sum of the squares of each matrix element), and $(GG^T)^{-1}G$ is the classical pseudo-inverse of G^T [13]. y^i is the ith column of the $m \times q$ matrix Y. $\phi(v^i, x^j)$ (where $(\cdot, \cdot)$ denotes the dot product of two vectors) is the element in the ith row and jth column of the $p \times q$ matrix G. A training input is included as a prototype (one of the v^i vectors) only if it does not induce ill conditioning in GG^T. This reduces the number of prototypes, and hence limits the number of middle layer neurons in the network.

A given exemplar is included in the minimization problem of (1) only if it cannot be correctly classified by the network which has been trained up to that point. Those exemplars which are included as an (x^i, y^i) pair in Y and G are referred to as *subprototypes*. So Y becomes an $m \times l$ matrix, and G becomes a $p \times l$ matrix, where l is the number of subprototypes chosen from the training inputs. The training algorithm is such that $p \leq l \leq q$. Figure 1 shows a graphical representation of the OI Net. See [5, 10] for further details.

III FAULT TOLERANCE

As discussed in Section I, much of the literature on the subject of fault tolerance deals with neurons that are stuck on or stuck off. A more realistic picture of neural network faults, however, is imprecision in the network's weights. It has been suggested that analog VLSI implementations of neural networks are inherently limited to a relative precision of about 1% [12]. In order to protect against hardware imprecision in the operation of a neural network, we can spread the computation throughout the network in such a way that the effect of an error in any given weight is minimized. This can be done by attempting to give each weight approximately the same magnitude. This idea is captured in the investor's mantra of diversification and in the maxim "Don't put all your eggs in one basket."

There is currently no standard method of implementing neural networks in hardware. A neural net may be implemented with digital electronics, analog electronics, or optics. One way that a neural network can be implemented in hardware results in the weights being stored in the processing elements [14, 15]. An OI Net has $n + p + m$ processing elements, where n is the number of inputs, p is the number of middle layer neurons, and m is the number of outputs (see Fig. 1). So if there is some hardware imprecision at one of the processing elements, that would be equivalent to imprecision in all of the weights connected to that processing element. Looking more closely at Fig. 1, for example, it can be seen that hardware imprecision at the y_1 processing element would be equivalent to imprecision in all of the W_{i1} weights ($i = 1, \ldots, p$). In other words, imprecision in the first output processing element is equivalent to imprecision in the first column of the W weight matrix. Similarly, imprecision in the jth output processing element is equivalent to imprecision in the jth column of the W weight matrix. Likewise, imprecision in the kth middle layer processing element is equivalent to imprecision in the kth row of the W weight matrix.

These equivalences can be seen from Fig. 1 to be exact. For example, if the hardware of the jth output processing element exceeds its nominal value by 1%, that is exactly equivalent to each element in the jth column of the W weight matrix exceeding its nominal value by 1%. Similarly, if the hardware of the kth middle layer processing element is 1%

below its nominal value, that is exactly equivalent to each element in the kth row of the W weight matrix being 1% below its nominal value.

It can therefore be seen that in order to achieve fault tolerance in a hardware implementation of an OI Net we can attempt to train the network in such a way that the sum of each row of the weight matrix W is equal, and the sum of each column of W is also equal. Then if some imprecision occurs in one of the middle layer processing elements, its effect will be minimized on the operation of the network because the other rows of the weight matrix will be able to take up the slack more easily. Likewise, if some imprecision occurs in one of the output layer processing elements, its effect will be minimized on the operation of the network because the other columns of the weight matrix will be able to take up the slack more easily. If we train the OI Net without taking processing element errors into account, then if imprecision occurs at a processing element that corresponds to a "large" row or column of W, the operation of the network will be strongly affected.

This sounds something like classical regularization techniques, such as weight decay, pruning, input perturbation, and weight smoothing [16]. However, in this paper we are neither trying to reduce the complexity of a network nor are we trying to improve the generalization properties of the network. Those objectives are already satisfied by the standard OI Net. In this paper we are specifically trying to improve network performance in the presence of the particular types of hardware imprecisions described above. Regularization techniques (in general) do not improve this type of fault-tolerance, as will be shown in Section V.

A Output Layer Fault Tolerance

As discussed above, we can attempt to achieve output layer fault tolerance by keeping the sum of all of the columns of W equal. This can be viewed as a constraint on the OI Net optimization problem. In Section II we said that W was determined as the solution to the optimization problem min $\| Y - W^T G \|$. In order to introduce output layer fault tolerance into W we can constrain the problem as follows.

$$\min_W \parallel Y - W^T G \parallel \text{ subject to } \sum_{i=1}^{p} W_{ij} = K_c \ (j = 1, \ldots, m) \tag{2}$$

where K_c is a constant to be determined. In other words, we want to perform the original optimization problem except with the constraint that the solution W is such that the sum of each column is the same. This constrained optimization problem can be written as

$$\min_W \parallel Y - W^T G \parallel \text{ subject to } LW = C \equiv [K_c \ \cdots K_c] \tag{3}$$

where L is a $1 \times p$ matrix where each element is a 1, and C is a $1 \times m$ matrix. This constrained optimization problem has been solved in [17]. If $\hat{W}$ is the solution of the unconstrained optimization problem (1), then the solution $\tilde{W}$ of the constrained optimization problem (3) is given as

$$\tilde{W} = \hat{W} - (GG^T)^{-1} L^T [L(GG^T)^{-1} L^T]^{-1} (L\hat{W} - C) \tag{4}$$

The solution of the constrained optimization problem consists of the solution of the unconstrained problem plus a correction term. L is a $1 \times p$ matrix, where p is the number of prototypes in the OI Net. Therefore $L(GG^T)^{-1}L^T$ is a scalar and its inversion in (4) is not a computational issue. (Although the inversion of the $p \times p$ matrix GG^T is a computational issue, it will be dealt with later in this paper.) K_c in (2) is the constraint that we are enforcing on the sum of the columns of the W matrix. The following theorem guarantees that appropriate values of K_c will improve the output layer fault tolerance of the OI Net.

Theorem 1 *Denote by $\hat{e}_c$ the change of the unconstrained OI Net solution (1) due to an error in one of the output layer nodes. Similary, denote by $\tilde{e}_c$ the change of the constrained OI Net solution (4) due to the same error. Then, for sufficiently large values of K_c, the upper bound of $\tilde{e}_c$ is smaller than $\hat{e}_c$.*

The proof is provided in the appendix. Note that this theorem does not address the fault-free performance of the OI Net. Neither does it state that the constrained OI Net solution changes less (due to output layer errors) than the unconstrained OI Net solution; it states only that this relationship holds between the upper bounds of the changes. However, the theorem does give us some confidence that the constrained OI Net will indeed be more tolerant to output layer faults than the unconstrained OI net. This is similar to H_∞ control, where the control objective is to minimize the upper bound of the effect of noise on a performance criterion [18].

B Middle Layer Fault Tolerance

As discussed above, we can attempt to achieve middle layer fault tolerance by keeping the sum of all of the rows of W equal. This can be viewed as a constraint on the OI Net optimization problem. In Section II we said that W was determined as the solution to the optimization problem min $\| Y - W^T G \|$. In order to introduce middle layer fault tolerance into W we can constrain the problem as follows.

$$\min_W \| Y - W^T G \| \text{ subject to } \sum_{j=1}^{m} W_{ij} = K_r \ (i = 1, \ldots, p) \tag{5}$$

where K_r is a constant to be determined. In other words, we want to perform the original optimization problem except with the constraint that the solution W is such that the sum of each row is the same. This constrained optimization problem can be written as

$$\min_W \|Y - W^T G\| \text{ subject to } MW^T = N \equiv [K_r \ \cdots K_r] \tag{6}$$

where M is a $1 \times m$ matrix where each element is a 1, and N is a $1 \times p$ matrix. This constrained optimization problem has been solved in [17]. If $\hat{W}$ is the solution of the unconstrained optimization problem (1), then the solution $\tilde{W}$ of the constrained optimization problem (6) is given as

$$\tilde{W} = \hat{W} - (\hat{W}M^T - N^T)(MM^T)^{-1}M \tag{7}$$

The solution of the constrained optimization problem consists of the solution of the unconstrained problem plus a correction term. (Note that in [17] a weighting matrix V is part of the constrained optimization solution. In our implementation we choose V to be equal to the identity matrix; that is, we attach equal important to each element of $Y - W^T G$ in (6)). M is a $1 \times m$ matrix, where m is the dimension of the output vector of the OI Net. Therefore MM^T is a scalar equal to m and its inverse in (7) is simply equal to $1/m$. K_r in (5) is the constraint that we are enforcing on the sum of the rows of the W matrix. The following theorem guarantees that appropriate values of K_r will improve the middle layer fault tolerance of the OI Net.

Theorem 2 *Denote by $\hat{e}_r$ the change of the unconstrained OI Net solution (1) due to an error in one of the middle layer nodes. Similary, denote by $\tilde{e}_r$ the change of the constrained OI Net solution (7) due to the same error. Then, for values of K_r with sufficiently small magnitude, the upper bound of $\tilde{e}_r$ is smaller than $\hat{e}_r$.*

The proof is provided in the appendix. Note that (as in Theorem 1) this theorem does not address the fault-free performance of the OI Net. Neither does it state that the constrained OI Net solution changes less (due to middle layer errors) than the unconstrained OI Net solution; it states only that this relationship holds between the upper bounds of the changes. However, the theorem does give us some confidence that the constrained OI Net will indeed be more tolerant to middle layer faults than the unconstrained OI net.

C <u>General Fault Tolerance</u>

Attempting to both keep the sum of all of the rows of W equal and the sum of all of the columns of W equal can be viewed as a generalization of the two preceding sections. We can constrain the problem of Section II as follows.

$$\min_W \parallel Y - W^T G \parallel \text{ subject to} \tag{8}$$
$$\sum_{i=1}^{p} W_{ij} = K_c \ (j = 1, \ldots, m) \text{ and } \sum_{j=1}^{m} W_{ij} = K_r \ (i = 1, \ldots, p)$$

where K_c and K_r are constants that satisfy the conditions of the two preceding subsections. In other words, we want to perform the original optimization problem except with the constraints that the solution W is such that the sum of each column is the same and the sum of each row is the same. The constrained optimization problem can be written as

$$\begin{aligned} &\min_{W} \parallel Y - W^T G \parallel \ \text{ subject to} \\ &\quad LW = C \equiv [K_c \ \cdots K_c] \text{ and } MW^T = N \equiv [K_r \ \cdots K_r] \end{aligned} \tag{9}$$

where L is a $1 \times p$ matrix where each element is a 1, M is a $1 \times m$ matrix where each element is a 1, C is a $1 \times m$ matrix, and N is a $1 \times p$ matrix. This constrained optimization problem has been solved in [17]. If $\hat{W}$ is the solution of the unconstrained optimization problem (1), then the solution $\tilde{W}$ of the constrained optimization problem (9) is given as

$$\begin{aligned} \tilde{W} = \hat{W} - (GG^T)^{-1}L^T[L(GG^T)^{-1}L^T]^{-1}(L\hat{W} - C) - (\hat{W}M^T - N^T)(MM^T)^{-1}M + \\ (GG^T)^{-1}L^T[L(GG^T)^{-1}L^T]^{-1}(L\hat{W} - C)M^T(MM^T)^{-1}M \end{aligned} \tag{10}$$

The solution of the constrained optimization problem consists of the solution of the unconstrained problem plus some correction terms. (As in the preceding subsection, we note that in [17] a weighting matrix V is part of the constrained optimization solution. In our implementation we choose V to be equal to the identity matrix; that is, we attach equal importance to each element of $Y - W^T G$ in (9)).

The preceding two subsections indicate that K_c should be a large positive number and K_r should be a number with small magnitude (like zero). At this point we do not have analytical proof that the constrained OI Net solution is more fault tolerant than the unconstrained solution. But the reasonableness of the approach, along with the theorems of the preceding two subsections, give us a high degree of confidence that the use of (10) will improve the fault tolerance of the OI Net.

IV THE FAULT-TOLERANT LEARNING ALGORITHM

In this section we extend the recursive OI Net learning algorithm [5, 6] to include the distributed fault tolerance described in the previous section. The algorithm presented here is based on [5], so this paper presents only the differences between the algorithm in [5] and the fault-tolerant algorithm. The notation used in the learning algorithm is summarized in Table 1.

1. Initialization.

   This step is just as described in [5], with the addtional equation

   $$\tilde{W}_p^l = W_p^l + (W_p^l - C)(1_{m,m} - I_m)$$

   where C is given as indicated in Section III and I_m denotes the $m \times m$ identity matrix. $\tilde{W}$ is derived from (10), but it has a very simple form here because of the small initial dimension of the problem ($l = p = 1$). The initial training error due to W_p^l should be initialized as $\mathcal{E}_p^l = 0$.

2. Main Recursion.

   After W_p^{l+1} is computed as described in Equation (20) in [5], we solve the constrained minimization problem

   $$\min_W \parallel Y_{l+1} - W^T G_p^{l+1} \parallel \text{ subject to } LW = C \text{ and } MW^T = N \tag{11}$$

   where C and N are given as indicated in Section III. The discussion following (9) shows that this problem can be solved as

   $$\begin{aligned}\tilde{W}_p^{l+1} = W_p^{l+1} - (R_p^{l+1})^{-1}1_{p,1}[1_{1,p}(R_p^{l+1})^{-1}1_{p,1}]^{-1}(1_{1,p}W_p^{l+1} - C) - \\ (W_p^{l+1}1_{m,1} - N^T)1_{1,m}/m + \\ (R_p^{l+1})^{-1}1_{p,1}[1_{1,p}(R_p^{l+1})^{-1}1_{p,1}]^{-1}(1_{1,p}W_p^{l+1} - C)1_{m,m}/m\end{aligned} \tag{12}$$

   We next consider including z^{l+1} as a prototype. Two constants chosen by the user, γ_1 and γ_2, determine whether z^{l+1} is used as a prototype. γ_1 is chosen to prevent ill conditioning in the weight determination procedure; it should be chosen on the basis of the numerical precision of the computer on which this algorithm runs. γ_2 is chosen as a network complexity parameter since it determines the number of prototypes (and hence the number of middle layer neurons). A large γ_2 will reduce the complexity of the network. That is, a large γ_2 will improve the generalization of the network by decreasing the network variance and increasing the bias. A small γ_2 will improve the performance of the network on the training data at the probable expense of generalization capability. That is, a small γ_2 will increase the network variance and decrease the bias [1]. If the two conditions involving γ_1 and γ_2 given by Equations (36) and (41) in [5] are satisfied, we compute W_{p+1}^{l+1} as described in [5]. We then solve the constrained minimization problem

$$\min_{W} \parallel Y_{l+1} - W^T G_{p+1}^{l+1} \parallel \text{ subject to } LW = C \text{ and } MW^T = N. \tag{13}$$

where C and N are given as described in Section III. The discussion following (9) shows that this problem can be solved as

$$\begin{aligned}\tilde{W}_{p+1}^{l+1} = W_{p+1}^{l+1} - (R_{p+1}^{l+1})^{-1} 1_{p+1,1} [1_{1,p+1} (R_{p+1}^{l+1})^{-1} 1_{p+1,1}]^{-1} (1_{1,p+1} W_{p+1}^{l+1} - C) - \\ (W_{p+1}^{l+1} 1_{m,1} - N^T) 1_{1,m}/m + \\ (R_{p+1}^{l+1})^{-1} 1_{p+1,1} [1_{1,p+1} (R_{p+1}^{l+1})^{-1} 1_{p+1,1}]^{-1} (1_{1,p+1} W_{p+1}^{l+1} - C) 1_{m,m}/m\end{aligned} \tag{14}$$

We then augment the subprototype z^{l+1} to the prototype matrix V and the subprototype matrix Z, and increment p and l by one to reflect the addition of a new prototype and subprototype.

If the two conditions involving γ_1 and γ_2 given by Equations (36) and (41) in [5] are not both satisfied then z^{l+1} cannot be included as a prototype. We augment z^{l+1} to the subprototype matrix Z and increment l by one to reflect the addition of a new subprototype.

3. Reiterate.

   This step is the same as described in [5].

Step (2) executes $q - 1$ times at the most [5].

V SIMULATION RESULTS

We tested the algorithm of the previous section on the classic Iris classification problem [20]. Each Iris input has four features and is classified into one of three categories. The Iris data contains 50 examplars from each category for a total of 150 patterns. We normalized the features to values between 0 and 0.5. The test environment was MATLAB code on a Pentium III 550 MHz PC.

The 150 patterns were randomly divided into two equal parts such that 25 patterns from each class formed a 75-pattern training set, and the remaining 25 patterns from each class formed a 75-pattern test set. The OI Net was run as described earlier in this paper with parameters $\gamma_1 = 10^{-8}$, $\gamma_2 = 0.35$, and $\rho = 0.5$. γ_1 was chosen as an arbitarily small number to prevent ill conditioning in the training algorithm. In this example it was found that the choice of γ_2 and ρ did not significantly affect the performance of the network. For instance, any value of γ_2 between 0.2 and 0.8, and any value of ρ between 0.2 and 0.8 yielded virtually identical results.

The fault-tolerance parameters K_c and K_r in Section III were chosen as 1000 and 0 respectively. The choice of K_r is straightforward because Theorem 2 says that we want to use a K_r with a small magnitude. The choice of K_c is more ambiguous because, according to Theorem 1, we want to use a large value. But if we use too large of a value then we may encounter numerical problems, or the weights of the fault-tolerant network may change so much relative to the traditional network that fault-free performance will suffer. A value of 1000 was chosen for K_c as a large value that is not "too" large.

The advantage of the OI Net as compared to the backpropagation and nearest neighbor classification methods has been documented in previous work [5]. This paper compares the traditional OI Net with the fault-tolerant OI Net. The OI Nets were simulated with weight perturbations at the middle layer neurons and output neurons. The classification rate was obtained by perturbing the weights at each of these $p + m$ neurons by a given percentage, one neuron at a time, and then averaging the resultant $p + m$ classification rates. This was done for 10 successive runs, where the 75-element training and test sets were randomly generated each time. The average classification rates were then averaged over the 10 simulations. The performance of the traditional OI Net as compared with the fault-tolerant OI Net for various levels of weight perturbations is depicted in Fig. 2. It should be noted that these performance results are robust across a wide range of sampling schemes. The results presented here are essentially the same regardless of the sample selection (as long as an equal number of samples are selected from each of the three categories). The results are also the same whether 10 successive runs or 100 successive runs are used.

A close look at Fig. 2 shows that the fault-tolerant OI Net performs slightly better than the traditional OI Net even in the case where the weight perturbations are zero. The regularization introduced by fault tolerance results in slightly better generalization even though the fault-tolerant weights are theoretically "less optimal" than the traditional weights. But more importantly, Fig. 2 shows that the fault-tolerant network performs considerably better than the traditional network in the presence of the type of weight perturbations that could be expected in a hardware implementation. The performance of the fault-tolerant network does drop off as the magnitude of the faults increases, but the drop-off is less severe than for the traditional network. This is in accordance with Theorems 1 and 2.

Fault-tolerant training results in a network with fewer middle layer neurons than regular training (an average of 5.1 prototypes versus 7.4 prototypes). Building fault tolerance into the network can be viewed as protecting the network from noisy data, which leads to improved generalization properties. It has been suggested that smaller networks result in improved generalization [2, 3], so the reduction in network size due to fault-tolerant training is not unexpected.

The standard deviation of the weights between the middle layer neurons and output neurons was 12 for the fault-tolerant OI Net, while the standard deviation was 312 for the traditional OI Net. These numbers show how successful the fault-tolerant training algorithm is at evenly distributing the weights throught the network.

The computational efforts of the two OI Net training algorithms are comparable. The fault-tolerant algorithm consists of the standard algorithm plus some correction terms, but the correction terms add a negligible amount of computational effort. However, the fault-tolerant algorithm is slightly more expensive computationally because of the increased number of training iterations. In the Iris example discussed in this section, the standard algorithm required an average of 0.15 s of CPU time and 4.3 learning steps (i.e., Step (2) in Section IV executed an average of 4.3 times). The fault-tolerant algorithm required an average of 0.18 s of CPU time and 5.3 learning steps. This increased number of iterations (and hence CPU time) can be inferred from a careful examination of the training algorithm. The addition of fault-tolerance reduces (on average) the fault-free performance of the network relative to the training data. This results in more iterations of Step (2) of the training algorithm before all of the training exemplars can be correctly classified by the network. Another way of stating the same thing is that the fault-tolerant algorithm requires more subprototypes in the optimization problem in order to correctly classify all of the training data. The average number of subprototypes was 15.9 for fault-tolerant training and 14.6 for traditional training.

As indicated in Section III, this sounds something like classical regularization techniques for improving the generalization of a neural network. We explored this possible equivalence by simulating the standard OI Net training algorithm with various levels of training noise [21]. Figure 3 shows the performance of the network trained with various levels of input perturbations. (The percentages show the standard deviation of the randomly generated input noise as a percentage of the ranges of each network input parameter.) Figure 3 shows that input perturbation does not improve the fault tolerance of the network. So general regularization techniques do not help the OI net for the specific types of faults considered in this paper.

However, the fault-tolerant OI net in this paper can be viewed as a special purpose regularization algorithm. It is a regularization algorithm because it imposes some structure on the network weights. It is a special purpose regularization algorithm because the imposed structure is specifically designed to minimize the effect of particular types of faults. As such, it can be expected that the fault-tolerant OI net would exhibit more bias in capturing the desired data structure, but less variance. This has been found to be true via simulation results. The integrated bias, variance, and total error were approximated by a Monte Carlo procedure as described in [1].

$$\begin{aligned}
\text{Bias} &\approx \frac{1}{N}\sum_{l=1}^{N}|\bar{f}(x_l)-y_l|^2 & (15)\\
\text{Variance} &\approx \frac{1}{N}\sum_{l=1}^{N}\frac{1}{M}\sum_{k=1}^{M}|f(x_l,D^k)-\bar{f}(x_l)|^2 & (16)\\
\text{Total Error} &\approx \frac{1}{N}\sum_{l=1}^{N}\frac{1}{M}\sum_{k=1}^{M}|f(x_l,D^k)-y_l|^2 & (17)
\end{aligned}$$

In the above equations, N is the number of test exemplars (75 in our example) and M is the number of training sets that were used to approximate the above quantities (50 in our example – i.e., we executed 50 training/test cycles). For each training/test cycle, we trained with a different 45-element subset of the 75 training exemplars. In the above equations, $f(x_l, D^k)$ is the response to the lth test input of the OI Net that was trained with the kth training set. $\bar{f}(x_l)$ is the average response to the lth test input. y_l is the desired response to the lth test input. The total error is equal to the sum of the bias and the variance. As the form of a neural network becomes more constrained (e.g., by having less parameters), the bias increases and the variance decreases. In our Iris example, the regular OI net had a bias, variance, and total error equal to 0.16, 0.22, and 0.38. The fault-tolerant OI net had values equal to 0.60, 0.09, and 0.69. So the introduction of fault-tolerance increased the bias and the total error, but decreased the variance of the network. The bias increase and variance decrease are expected because we are essentially adding more constraints to the network. The surprising part of these results is that the fault-tolerant network performs better even though the total error is larger. This underscores the fact that a network with a larger error might perform better than a network with a smaller error. It may be that variance is more important for network performance than total error. This could be true if, as in the results of this section, the underlying problem is a classification problem rather than an interpolation problem. For a classification problem we may get better results if the solution surface is more smooth, even if we have more total error. This analysis, along with the simulation results of this section, indicates that fault-tolerant training may be more appropriate for classification problems than interpolation problems. In fact, the OI Net was originally proposed as a solution to classification problems [5].

The MATLAB m-files that were used to generate the results presented in this paper can be downloaded from the world-wide web at `http://academic.csuohio.edu/simond/oinet/`. A user who re-runs the experiments presented here can expect to get similar results, although the results will not be identical because of the random generation of training sets and test sets.

VI CONCLUSION

A recursive learning algorithm for a fault-tolerant Optimal Interpolative Net has been presented. The inclusion of fault tolerance makes the network more robust to small perturbations in the weights, such as those that might occur in a hardware implementation. The resultant network contains fewer hidden layer neurons and hence decreases the complexity of the network. The fault tolerance discussed in this paper applies to small perturbations of the weights between the middle layer and the output layer.

The fault-tolerant OI Net has been applied to the classic Iris data. The results show that not only is fault tolerance greatly increased but nominal performance slightly improves also. This is because the introduction of fault tolerance can be viewed as protecting the network against noisy data and hence improving the generalization properties of the network. The MATLAB code used in this research can be downloaded from the world wide web at `http://academic.csuohio.edu/simond/oinet/`.

This research is limited to tolerance to imprecision in the middle layer and output layer neurons. Further research along these lines is focusing on tolerance to imprecision in the input layer neurons, and the generalization of Theorems 1 and 2.

ACKNOWLEDGEMENTS

The comments and suggestions of the Associate Editor and three anonymous referees were instrumental in significantly strengthening this paper from its original version.

APPENDIX

Theorem 1 Proof: Theorem 1 states that for sufficiently large values of K_c, the upper bound of the change of the constrained OI Net solution (4) due to an error in one of the output layer nodes is smaller than the upper bound of the change of the unconstrained OI Net solution (1) due to the same error. Recall from (4) that

$$\tilde{W} = \hat{W} - (GG^T)^{-1}L^T[L(GG^T)^{-1}L^T]^{-1}(L\hat{W} - C) \tag{18}$$

We will use the shorthand notation $\mathcal{G} = (GG^T)^{-1}$ and $\lambda = L\mathcal{G}L^T$ (note that λ is a scalar). By carrying out the multiplications of (18) we can show that the element in the ith row and jth column of $\tilde{W}$ is given by

$$\tilde{W}_{ij} = \hat{W}_{ij} - \sum_k \mathcal{G}_{ik}\left(\sum_k \hat{W}_{kj} - K_c\right)/\lambda \tag{19}$$

Figure 1 and Section II indicate that the nominal output of the OI Net can be written as $y = W^T g$. If there is a relative precision error of δ in the rth output processing element, the output of the OI Net changes from y to y_r. This precision error is equivalent to perturbing the rth column of W. Denote the perturbed weight matrix W as W_r.

$$W_r = W(I - \Delta_r) \tag{20}$$

where Δ_r is the symmetric $m \times m$ matrix that contains all zeros except for the element in the rth row and rth column, which contains δ. So the change in the OI Net solution due to an output layer node error is given by

$$\begin{aligned} y - y_r &= W^T g - W_r^T g && (21)\\ &= [W^T - (I - \Delta_r)^T W^T]g && (22)\\ &= \Delta_r W^T g && (23) \end{aligned}$$

One reasonable way to measure the size of the change of the OI Net solution is to take the vector two-norm, which gives

$$\begin{aligned} \|y - y_r\| &= \|\Delta_r W^T g\| && (24) \\ &\leq \|\Delta_r W^T\| \cdot \|g\| && (25) \end{aligned}$$

where $\| \cdot \|$ refers to the two-norm of a vector and the Froebenius norm of a matrix [22, p. 291]. We know that (in general) $\|A\| = \|A^T\|$. So the above equation becomes

$$\|y - y_r\| \leq \|W\Delta_r\| \cdot \|g\| \tag{26}$$

So $\|W\Delta_r\|$ provides an upper bound for the change of the OI Net solution due to a relative precision error in the rth output processing element. From (19) we can derive the element in the ith row and jth column of $\tilde{W}\Delta_r$ as

$$(\tilde{W}\Delta_r)_{ij} = \begin{cases} \delta[\hat{W}_{ij} - \sum_k \mathcal{G}_{ik}(\sum_k \hat{W}_{kj} - K_c)/\lambda] & \text{if} \quad j = r \\ 0 & \text{if} \quad j \neq r \end{cases} \tag{27}$$

From this we can obtain

$$\begin{aligned} \|\tilde{W}\Delta_r\|^2 &= \delta^2 \left[\sum_i \hat{W}_{ir}^2 - 2 \sum_i \hat{W}_{ir} \sum_k \mathcal{G}_{ik} \left(\sum_k \hat{W}_{kr} - K_c \right) / \lambda + \right. && (28) \\ & \left. \left(\sum_k \hat{W}_{kr} - K_c \right)^2 \sum_i \left(\sum_k \mathcal{G}_{ik} \right)^2 / \lambda^2 \right] \end{aligned}$$

From this we can see that $\|\tilde{W}\Delta_r\| < \|\hat{W}\Delta_r\|$ if the following condition holds.

$$2 \sum_i \hat{W}_{ir} \sum_k \mathcal{G}_{ik} \left(\sum_k \hat{W}_{kr} - K_c \right) / \lambda > \left(\sum_k \hat{W}_{kr} - K_c \right)^2 \sum_i \left(\sum_k \mathcal{G}_{ik} \right)^2 / \lambda^2 \tag{29}$$

The above condition holds if

$$K_c > \sum_k \hat{W}_{kr} - \frac{2\lambda \sum_i \hat{W}_{ir} \sum_k \mathcal{G}_{ik}}{\sum_i (\sum_k \mathcal{G}_{ik})^2} \tag{30}$$

So if K_c in (3) satisfies (30) for all values of r ($r = 1, \cdots, m$) then the upper bound of the change of the constrained OI Net solution (4) due to a relative precision error in one of the output processing elements is smaller than the upper bound of the change of the unconstrained OI Net solution (1) due to the same error. **QED**

Theorem 2 Proof: Theorem 2 states that for sufficiently small values of K_r, the upper bound of the change of the constrained OI Net solution (7) due to an error in one of the middle layer nodes is smaller than the upper bound of the change of the unconstrained OI Net solution (1) due to the same error. Recall from (7) that

$$\tilde{W} = \hat{W} - (\hat{W}M^T - N^T)(MM^T)^{-1}M \tag{31}$$

By carrying out the multiplications of (31) we can show that the element in the ith row and jth column of $\tilde{W}$ is given by

$$\tilde{W}_{ij} = \hat{W}_{ij} - \left(\sum_k \hat{W}_{ik} - N \right) / m \tag{32}$$

Figure 1 and Section II indicate that the nominal output of the OI Net can be written as $y = W^T g$. If there is a relative precision error of δ in the rth middle processing element, the output of the OI Net changes from y to y_r. This precision error is equivalent to perturbing the rth row of W. Denote the perturbed weight matrix W as W_r.

$$W_r = (I - \Delta_r)W \tag{33}$$

where Δ_r is the symmetric $p \times p$ matrix that contains all zeros except for the element in the rth row and rth column, which contains δ. So the change in the OI Net solution due to a middle layer node error is given by

$$\begin{aligned} y - y_r &= W^T g - W_r^T g && (34) \\ &= [W^T - W^T(I - \Delta_r)^T]g && (35) \\ &= W^T \Delta_r g && (36) \end{aligned}$$

One reasonable way to measure the size of the change of the OI Net solution is to take the vector two-norm, which gives

$$\begin{aligned} \|y - y_r\| &= \|W^T \Delta_r g\| && (37) \\ &\leq \|W^T \Delta_r\| \cdot \|g\| && (38) \end{aligned}$$

where $\| \cdot \|$ refers to the two-norm of a vector and the Froebenius norm of a matrix [22, p. 291]. We know that (in general) $\|A\| = \|A^T\|$. So the above equation becomes

$$\|y - y_r\| \leq \|\Delta_r W\| \cdot \|g\| \tag{39}$$

So $\|\Delta_r W\|$ provides an upper bound for the change of the OI Net solution due to a relative precision error in the rth middle processing element. From (32) we can derive the element in the ith row and jth column of $\Delta_r \tilde{W}$ as

$$(\Delta_r \tilde{W})_{ij} = \begin{cases} \delta[\hat{W}_{ij} - (\sum_k \hat{W}_{ik} - N)/m] & \text{if} \quad i = r \\ 0 & \text{if} \quad i \neq r \end{cases} \tag{40}$$

From this we can obtain

$$\|\Delta_r \tilde{W}\|^2 = \delta^2 \left[\sum_j \hat{W}_{rj}^2 - 2 \left(\sum_j \hat{W}_{rj} - N \right) \sum_j \hat{W}_{rj}/m + \left(\sum_j \hat{W}_{rj} - N \right) /m \right] \tag{41}$$

From this we can see that $\|\Delta_r \tilde{W}\| < \|\Delta_r \hat{W}\|$ if the following condition holds.

$$\left(\sum_j \hat{W}_{rj} - N \right) \left[2 \sum_j \hat{W}_{rj} - \left(\sum_j \hat{W}_{rj} - N \right) \right] > 0 \tag{42}$$

The above condition holds if

$$-\left| \sum_j \hat{W}_{rj} \right| < N < \left| \sum_j \hat{W}_{rj} \right| \tag{43}$$

So if K_r in (6) satisfies (43) for all values of r ($r = 1, \cdots, p$) then the upper bound of the change of the constrained OI Net solution (7) due to a relative precision error in one of the middle processing elements is smaller than the upper bound of the change of the unconstrained OI Net solution (1) due to the same error. **QED**

References

[1] S. Geman, E. Bienenstock, and R. Doursat, "Neural Networks and the Bias/Variance Dilemma," *Neural Computation*, vol. 4, pp. 1-58, 1992.

[2] E. Karnin, "A Simple Procedure for Pruning Back-Propagation Trained Neural Networks," *IEEE Transactions on Neural Networks*, vol. 1, pp. 239-242, 1990.

[3] B. Hassibi, D. Stork, and G. Wolff, "Optimal brain surgeon and general network pruning," *Neural Networks Theory, Technology, and Applications*, P. Simpson, Ed., New York: IEEE, 1996, pp. 56-68.

[4] R. deFigueiredo, "An Optimal Matching-Score Net for Pattern Classification," *International Joint Conference on Neural Networks*, San Diego, California, pp. 909-916, 1990.

[5] S. Sin and R. deFigueiredo, "An Evolution-Oriented Learning Algorithm for the Optimal Interpolative Net," *IEEE Transactions on Neural Networks*, vol. 3, pp. 315-323, 1992.

[6] S. Sin and R. deFigueiredo, "Efficient Learning Procedures for Optimal Interpolative Nets," *Neural Networks*, vol. 6, pp. 99-113, 1993.

[7] C. Neti, M. Schneider, and E. Young, "Maximally Fault Tolerant Neural Networks," *IEEE Transactions on Neural Networks*, vol. 3, pp. 14-23, 1992.

[8] P. Edwards and A. Murray (eds.), *Analogue Imprecision in MLP Training*. Singapore: World Scientific, 1996.

[9] C. Sequin and R. Clay, "Fault tolerance in artificial neural networks," *International Joint Conference on Neural Networks*, pp. 703-708, San Diego, California, 1990.

[10] D. Simon and H. El-Sherief, "Fault-tolerant training for optimal interpolative nets," *IEEE Transactions on Neural Networks*, vol. 6, pp. 1531-1535, 1995.

[11] P. Edwards and A. Murray, "Can deterministic penalty terms model the effects of synaptic weight noise on network fault tolerance?" *International Journal of Neural Systems*, vol. 6, pp. 401-416, 1995.

[12] P. Edwards and A. Murray, "Penalty terms for fault tolerance," *International Conference on Neural Networks*, pp. 943-947, Houston, Texas, 1997.

[13] G. Golub and C. Van Loan, *Matrix Computations.* Baltimore, Maryland: The Johns Hopkins University Press, 1990.

[14] P. Ienne and M. Viredaz, "GENES IV: A bit-serial processing element for a multi-model neural-network accelerator," *Neural Networks Theory, Technology, and Applications*, P. Simpson, Ed., New York: IEEE, pp. 797-808, 1996.

[15] R. Sridhar and Y. Shin, "VLSI neural network architectures," *Neural Networks Theory, Technology, and Applications*, P. Simpson, Ed., New York: IEEE, pp. 864-873, 1996.

[16] R. Reed, R. Marks, and S. Oh, "Similarities of error regularization, sigmoid gain scaling, target smoothing, and training with jitter," *IEEE Transactions on Neural Networks*, vol. 6, pp. 529-538, May 1995.

[17] T. Chia, P. Chow, and H. Chizek, "Recursive parameter identification of constrained systems: an application to electrically stimulated muscle," *IEEE Transactions on Biomedical Engineering*, vol. 38, pp. 429-442, May 1991.

[18] J. Burl, *Linear Optimal Control.* Menlo Park, California: Addison-Wesley, 1999.

[19] L. Ljung and T. Soderstrom, *Theory and Practice of Recursive Identification.* Cambridge, Massachusetts: The MIT Press, 1985.

[20] J. Bezdek, J. Keller, R. Krishnapuram, L. Kuncheva, and H. Pal, "Will the *real* Iris data please stand up?" *IEEE Transactions on Fuzzy Systems*, vol. 7, pp. 368-369, 1999.

[21] K. Matsuoka, "Noise injection into inputs in back-propagation learning," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 22, pp. 436-440, 1992.

[22] R. Horn and C. Johnson, *Matrix Analysis*. New York, NY: Cambridge University Press, 1990.

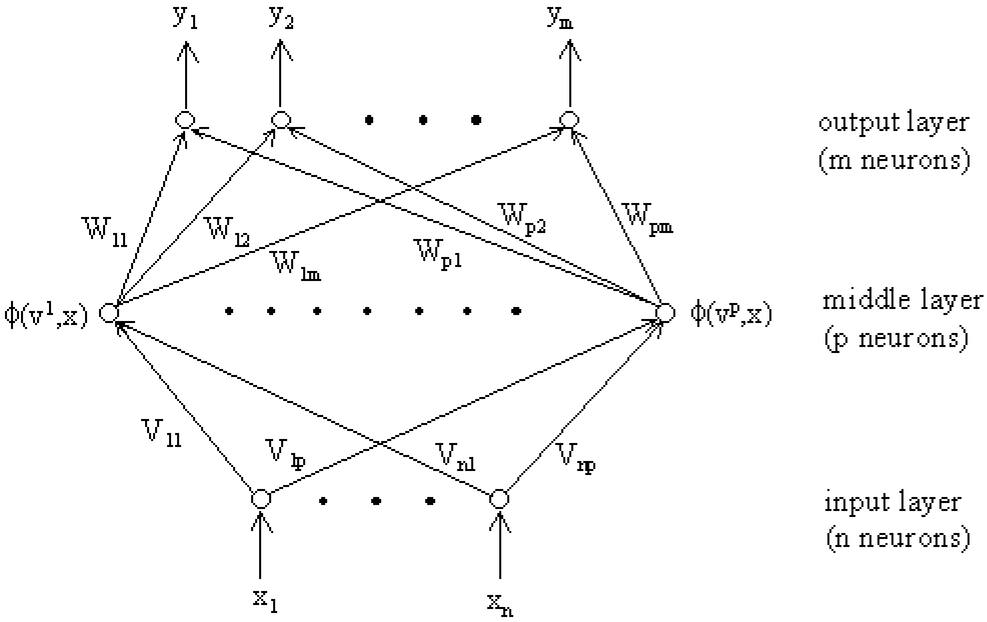

Figure 1: Optimal Interpolative Net Architecture.

Symbol	Meaning
x^i	training input vector
n	dimension of each input vector
A	set of all training input vectors
y^i	training output vector
m	dimension of each output vector
q	number of training exemplars
v^i	prototype vector (taken from A)
V	matrix containing prototypes
$\hat{W}$	unconstrained weight matrix between hidden layer and output layer
$\tilde{W}$	constrained weight matrix between hidden layer and output layer
p	number of prototypes
z^i	subprototype vector (taken from V)
Z	matrix containing subprototypes
l	number of subprototypes
f_p^l	the neural map based on the p prototypes
	in V and the l subprototypes in Z
γ_1	ill conditioning threshold
γ_2	error reduction threshold
ρ	learning parameter
$1_{r,q}$	the $r \times q$ matrix where each element is a 1

Table 1: Optimal Interpolative Net Notation.

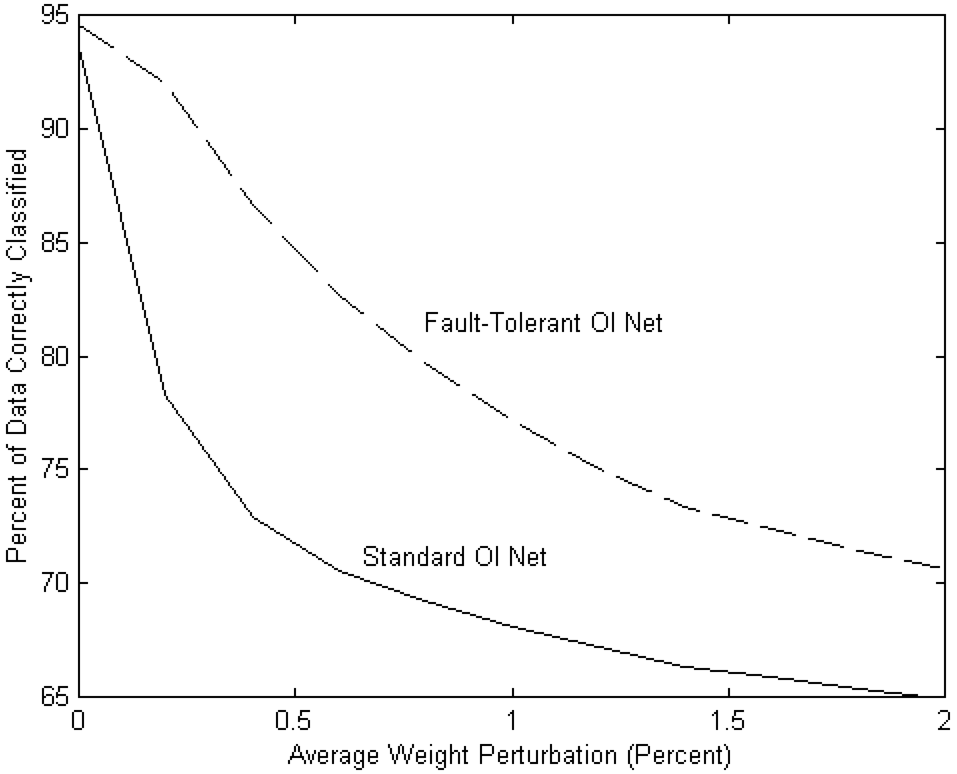

Figure 2: Average Optimal Interpolative Net Performance. The plot shows the average percentage of test data correctly classified. The test data consisted of 75 patterns and the average was taken over 10 simulations.

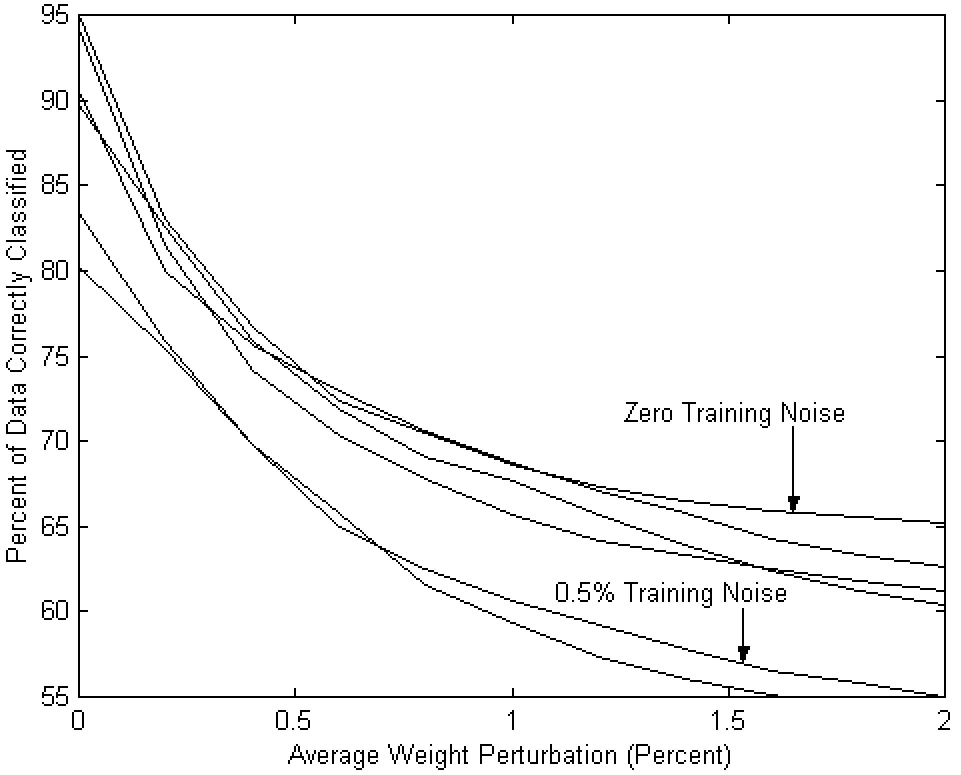

Figure 3: Average Optimal Interpolative Net Performance for six diffent noise levels (0%, 1%, 2%, 3%, 4%, and 5%) used during training. The plot shows the average percentage of test data correctly classified. The test data consisted of 75 patterns and the average was taken over 10 simulations.