Unsupervised Adaptation to Improve Fault Tolerance of Neural Network Classifiers

Alex Nugent, Garret Kenyon[1] and Reid Porter
Space Data Systems Group, Bio-Physics Group[1]
Los Alamos, NM 87545
Email contact: rporter@lanl.gov

Abstract

We investigate how to exploit the dynamics of unsupervised online learning rules for fault tolerance in neural network classifiers. We first design an adaptation mechanism that keeps neural network weights at a useful fixed point for classification problems. We then demonstrate the robustness of the system when the network inputs are subjected to faults.

1 Introduction

There are at least two scenarios where traditional fault detection and mitigation schemes appear insufficient: extreme environment electronics and computing with next-generation nano-scale devices. These have been significant application drivers for the evolvable hardare community. Haddow and Remortel ask the question [1]: can adaptive systems that maintain a chemical state of equilibrium provide the basis for robust, fault tolerant evolvable hardware? In this paper we investigate this concept within the framework of artificial neural networks. We will design an adaptive algorithm that maintains equilibrium for a neural network classifier and then demonstrate how it can help increase fault tolerance.

For applications like image and signal processing and real-time control, artificial neural networks, due to their distributed cellular architecture, can provide natural solutions for computing with fault prone devices. Some neural network architectures use system dynamics to solve computational problems e.g. associative memories and self-organizing maps. Solutions are represented as minima and the convergence of the system effectively provides self-organized fault tolerance [4].

In this paper we suggest how self-organized fault tolerance might be introduced into a wider class of application. Instead of requiring the network dynamics to directly encode the desired computation, we observe that the dynamics need to only include the desired computation as a local minimum. We use the initial conditions to bootstrap the system to the desired minimum. The basin of attraction for the minimum then provides self-organized fault tolerance.

We develop the approach for two-class classification problems. There are two modes of operation. First there is a design stage, where we use training data and supervised learning to find a good set of network weights for solving classification problems. Second there is a online stage where we imagine the network is deployed and classifies incoming data, one sample at a time. In the online stage we replace the static network weights of the supervised classifier with a dynamic update mechanism. During normal operation this mechanism maintains a stable equilibrium around the static network weights. When the neural network is subject to faults the mechanism can adapt weights to maintain performance.

2 Neural network classifiers

We use a Radial Basis Function (RBF) neural network illustrated in Figure 1 to solve classification problems. There are d first layer nodes. The i^{th} first layer node implements a Gaussian radial basis function with a fixed center $\boldsymbol{c_i}$ and fixed width l :

$$x_i = e^{-l\|\boldsymbol{c_i}-\boldsymbol{u}\|} \tag{1}$$

In the second layer there is a single node, which implements a weighted sum of first layer outputs:

$$y = \sum_{i=1}^{d} w_i x_i - b \tag{2}$$

The sign of the network output determines which class the sample will be assigned:

$$z = sign(y) \tag{3}$$

In the *online stage* faults are introduced into first layer nodes and the output node is assumed fault free. There are two types of faults that occur in 1st layer nodes:

- Static faults cause a node to output a constant value regardless of its input. This value is chosen randomly from a uniform distribution between 0 and 1.
- Dynamic faults cause the node to output a random value between 0 and 1 at each time step.

The assumption that the output node is fault free is based on the two layer network being used as part of a much larger multi-layered system. If this is the case, faults in the output node could be dealt with by applying our strategy to subsequent layers i.e., the output node becomes a first-layer node.

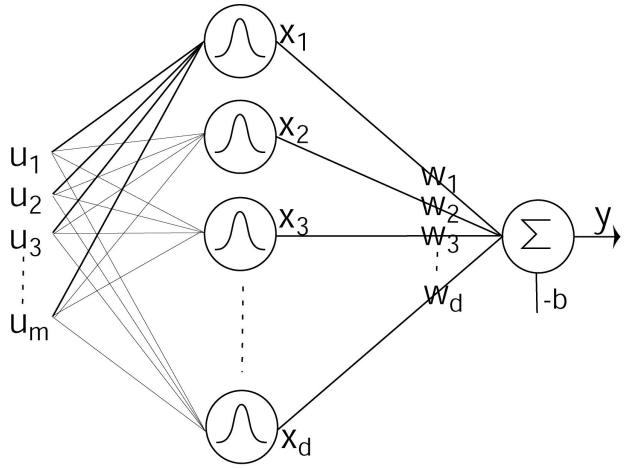

Figure 1. Network classifier used in experiments

3 Dynamics of unsupervised online learning

Our goal is to find an adaptation mechanism for updating the output node weights that can help account for the first layer faults. We investigate adaptation mechanisms with the general form:

$$\Delta w_i = a\, f\left(\boldsymbol{w}, \boldsymbol{x}, y\right) \tag{4}$$

where Δw_i is the change in the i^{th} weight from one time step to the next, a is a small constant and $f(\bullet)$ is some function that depends on quantities that are local (physically) to the i^{th} weight. A useful framework for analyzing this type of update is stochastic gradient ascent of an objective function $J(y)$, in which case (4) can be expressed as:

$$\Delta w_i = a\, \frac{d}{dw_i} J(y) \tag{5}$$

We denote the derivative of $J(y)$ with respect to y as $g(y)$, and the update is:

$$\Delta w_i = a\, x_i g\left(\boldsymbol{w}^T \boldsymbol{x}\right) \tag{6}$$

Oja [2] was one of the first to suggest this type of rule for finding interesting statistical structure in data. Recently higher-order objective functions have been suggested, motivated from a number of fields including statistics [3], information-theory [4], and biology [5]. These objective functions have multiple local maxima and learning rules converge to one of many fixed points. We suggest that the multiple fixed points of higher-order learning rules can provide a novel mechanism for fault tolerance. The basic assumption is that an objective function can be found that is related to, at some level, the desired behavior for the output node. If this is the case then it follows:

- The adaptation mechanism will be in a local fixed point close to the desired value of $\boldsymbol{w}$ and hence will maintain a stable equilibrium during normal operation.
- With faults present, the input statistics will shift the local fixed point and in some cases the adaptation mechanism will track this shift appropriately.

A classifier aims to separate inputs into one of two classes. Therefore we suggest a reasonable objective function would measure multi-modality. Several objective functions have been suggested that measure multimodality [6]. We choose to minimize the fourth-order cumulant, or kurtosis*:*

$$J(y) = E\left\{y^4\right\} - 3E\left\{y^2\right\}^2 \tag{7}$$

The weight vector $\boldsymbol{w}$ must be bound to produce a stable learning rule. One way to do this to place a constraint on the variance:

$$E\left\{y^2\right\} = 1 \tag{8}$$

This constraint can be included in the stochastic ascent learning rule by introducing a penalty term:

$$J(y) = \left(E\left\{y^4\right\} - 3\right) - b\, E\left\{y^2\right\} \tag{9}$$

Take the derivative of (9) with respect to y, substituting into (6) and multiplying by -1 (since we want the learning rule to minimize kurtosis) we arrive at

$$\Delta w_i = a\, x_i y\left(B - Ay^2\right), \tag{10}$$

where A and B are constants. The learning rule (10) is a generalized Hebbian learning rule, which we call Anti-Hebbian and Hebbian (Ahah!).

4 Detailed Experimental Design

4.1 Design Stage

In the *design stage* we use supervised learning and a training set to estimate the parameters for the RBF

network in Figure 1. The training set consists of N samples. For each sample there is an input vector $\boldsymbol{u}$ and a class label $l \in \{-1,1\}$. The training data is divided into 3 sets. Half of the samples are used during the online stage as a test set. The remaining samples are divided into two sets: training and validation. We use a Support Vector Machine (SVM) for the supervised learner [7]. This provides the number and location of centers $\boldsymbol{c_i}$ in equation (1), as well as the weights $\boldsymbol{w}$ and b of equation (2). Using cross-validation we also choose the SVM free parameters including the Gaussian width l in (1) and the level of regularization.

In the design stage we also set the Ahah rule constants A and B. These constants control the fixed point scale and therefore must be initialized close to the scale found by the supervised learner. We set $A = 1$ and average B over all inputs, where the expectation is taken over the training data:

$$B = \frac{1}{d}\sum_{i=1}^{d}\frac{\langle x_i y^3 \rangle}{\langle x_i y \rangle} \tag{11}$$

The update for the output node bias in (2) also uses (10) but has a constant input, $x = -1$.

$$\Delta b = a\, y(B - y^2) \tag{12}$$

4.2 Online Stage

During the *online stage* we randomly select one sample at a time from the test set and provide it as input to the RBF network. The network predicts a class label for the sample and we compare it to the desired label to estimate network performance over time. At a certain point some of the RBF nodes become faulty. In this paper we report on incremental failures where only one node becomes faulty at any one time but faulty nodes accumulate until the desired percentage of faulty nodes is attained. We compare three scenarios as faults are introduced:

1. Normal: For each sample the RBF network simply predicts the class label for the input. The network makes no attempt to account for newly occurring faults.
2. Ahah: For each sample the RBF network predicts the class label for the input and then updates each weight in equation (2) according to equation (10).
3. Retrain: After faults occur we retrain the weights in equation (2) and then predict future samples with the re-optimized classifier.

4.3 Retraining

The third scenario is used as an estimate for the best we can hope to achieve. To retrain, the original training and test sets are supplied as input to the faulty first layer. For each sample $\boldsymbol{u}$ we record the (possibly faulty) RBF outputs $\boldsymbol{x}$ producing a new training and test set. There are a number of supervised learning methods that could be used to re-optimize $\boldsymbol{w}$ and b based on the new training set. We use a linear program that uses the sum of absolute values as a weight regularizer instead of the sum of squares used by the SVM. This is preferred since the faults we introduce are specific to particular nodes. When using the sum of absolute values for regularization the weights corresponding to faulty input nodes will tend to be set to zero more easily than if we used the sum of squares for regularization [7].

5 Experimental Results

We generated a synthetic two-class, two-dimensional exclusive-or problem for initial experiments. Gaussian centers for each class are located along the diagonals of the input space: $[x_1, x_2] = (\pm 1, \pm 1)$. All Gaussians have a variance of 0.5. We use 200 samples for both training and validation sets. For the test set we continue to draw new samples as required (typically 8000).

Incremental dynamic faults were introduced according to a linear schedule between time steps 2000 and 6000. We take the average classification error (a running average over 300 time steps) between 7000 and 8000 and then averaged the result over 10 trials. For the retraining scenario we re-optimized the network at time step 6500. Note that this is a best-case scenario, particularly when faults are introduced incrementally

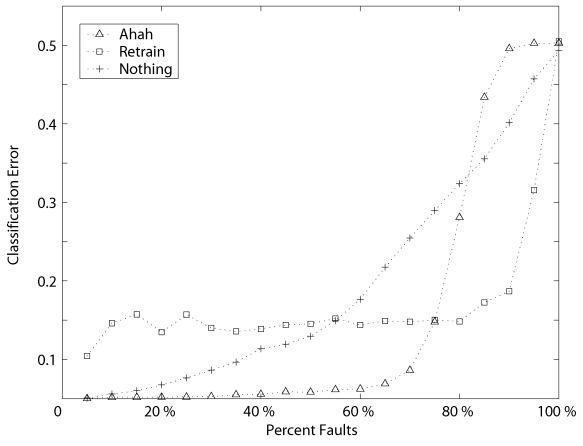

Figure 2. Gradual fault introduction of dynamic faults in x-or classification network

In Figure 2 we observe that retraining had poorer performance than doing nothing for low levels of faults. It

is likely a more thorough search for the optimal level of regularization during supervised learning would lead to more similar performance. For Ahah, the result in figure 2 is typical and the adaptation mechanism provides a substantial degree of fault tolerance. This was found for both static and dynamic faults.

We repeated the experimental setup described for real-world data obtained from the UCI Machine Learning repository [8]. We used the Ionosphere data set containing 34 attributes and 351 samples. Of the 351 samples approximately 70% belong to one class and therefore in Figure 3 the maximum error is approximately 0.3. We resample the 175 test samples with replacement to obtain 8000 online training points. In Figure 3 we observe Ahah again provides a substantial degree of fault tolerance.

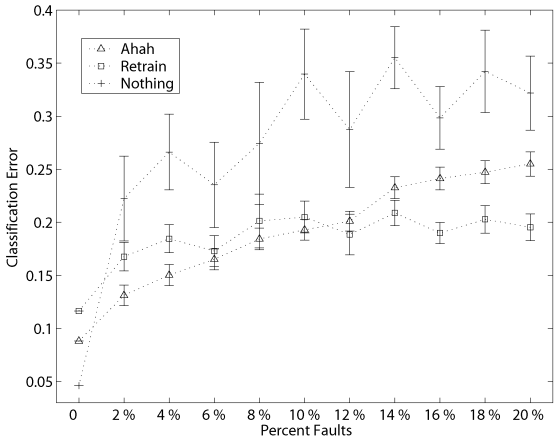

Figure 3. Gradual introduction of static faults in Ionosphere classification network

6 Discussion

In comparison to retraining, our approach has several advantages. Retraining must take the system offline in order to pass the training data through the faulty processing nodes. In addition, the learning rule itself is often complicated (such as a Linear Program) and may require a microprocessor and additional memory. In contrast, our unsupervised approach can be applied without taking the system off-line and has far fewer fault prone control mechanisms. In addition, it may be possible that the update term $b - ay^2$ can be implicitly calculated by using a specific nonlinearity (3) [3].

Another advantage of our approach is rapid response to newly occurring faults. The performance of retraining will depend on how often the faults occur and how often the retraining can occur. If retraining is seldom then the system may suffer long periods of poor performance. If retraining is over-scheduled then computational resources will be wasted

For a practical system, it is likely that coupling the unsupervised approach with retraining and/or more traditional fault detection strategies will be desirable. The unsupervised approach can provide fast response to new faults with little computational overhead. A supervised fault detection system could then detect operating conditions outside of the systems specification and provide mechanisms to reinitialize the unsupervised system as required.

7 Conclusion

We have suggested using the dynamics of an online learning rule to implement a form of self-organized fault tolerance. We showed that local minima of a multimodal objective function based on kurtosis can coincide closely with classifiers trained with supervised learning. We showed that the basins of attraction for these local minima can improve fault tolerance. Future research aims to develop and apply this concept to larger multi-layered neural network architectures.

Acknowledgments

This research was supported by the Los Alamos National Laboratory's Directed Research project: Scalable Reconfigurable Computing.

References

1. Haddow, P.C. and P.v. Remortel. *From Here to There: Future Robust EHW Technologies for Large Digital Designs.* in *The Third NASA/DoD Workshop on Evolvable Hardware*. 2001. Long Beach, California.
2. Oja, E., *A simplified neuron model as a principal component analyzer.* Journal of Math. Biology, 1982. **15**: p. 267-273.
3. Hyvarinen, A., J. Karhunen, and E. Oja, *Independent Component Analysis*. Adaptive and Learning Systems for Signal Processing, Communications and Control, ed. S. Haykin. 2001, New York: John Wiley & Sons, Inc.
4. Bell, A.J. and T.J. Sejnowski, *The `Independent Components' of Natural Scenes are Edge Filters.* Vision Research, 1997. **37**(23): p. 3327-3338.
5. Intrator, N. and L.N. Cooper, *Objective function formulation of the BCM theory of visual cortical plasticity,* Neural Networks, 1992. **5**(1): p. 3-17.
6. Dotan, Y. and N. Intrator, *Multimodality Exploration by an Unsupervised Projection Pursuit Neural Network.* IEEE Trans. Neural Networks,, 1998. **9**(3).
7. Hastie, T., R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*. Springer Series in Statistics, ed. Springer. 2001: Springer-Verlag
8. V. Sigillito. UCI Machine Learning Repository, http://www.ics.uci.edu/~mlearn/MLRepository.htm.