Neural Network Training for Varying Output Node Dimension

Jae-Byung Jung [1] , M.A. El-Sharkawi[1], R.J. Marks II[1], Robert Miyamoto[2], Warren L.J. Fox[12], G.M. Anderson[2], C.J. Eggen[2]

Abstract

Considered is the problem of neural network supervised learning when the number of output nodes can vary for differing training data. This paper proposes irregular weight updates and learning rate adjustment to compensate for this variation. In order to compensate for possible over training, an *a posteriori* probability that shows how often the weights associated with each output neuron are updated is obtained from the training data set and is used to evenly distribute the opportunity for weight update to each output neuron. The weight space becomes smoother and the generalization performance is significantly improved.

Key Words: layered perceptron, training, neural networks, learning rate.

1 Introduction

Multilayer perceptrons (MLP's) typically use a fixed network topology for all training patterns. We consider the case where the dimension of the output can vary from training pattern to training pattern. Let the input-target output training data be $\{\vec{i}[n], \vec{t}[n] | 1 \leq n \leq N\}$. By different output dimensionality, we mean the dimensions of the output vector, $\vec{t}[n]$, can vary as a function of n. We assume the dimension, $M[n]$, of each output is known.

A modular neural networks structure [1][2][3] containing local experts for different dimension-specific training patterns can be applied to this problem. Here, each component neural network is trained with the subset of data corresponding to a fixed number of outputs. It becomes increasingly difficult, however, to implement a large number of neural networks for a large number of experts in the absence of ample training data.

If a single neural network is to be used for the problem the output must be made sufficiciently large to handle the longest of target vectors. Let $M = \max_n M[n]$. We define a new output vector set of length M with elements, $\vec{\tau}[n]$, as the vector concatination

$$\vec{\tau}[n] = \begin{bmatrix} \vec{t}[n] \\ \vec{t}_{DC}[n] \end{bmatrix} \tag{1}$$

where the vector $\vec{t}_{DC}[n]$ (the DC is for *don't care*), of length $M - M[n]$, corresponds to output values not in the active region. If the nth output is partitioned into ACTIVE_n and DON'T CARE_n, then (1) can equivalently be written as

$$\begin{aligned} \tau_m[n] &= (\vec{\tau}[n])_m \\ &= \begin{cases} t_m[n] & ; m \in \text{ACTIVE}_n \\ (\vec{t}_{DC}[n])_m & ; m \in \text{DON'T CARE}_n \end{cases} \end{aligned} \tag{2}$$

One approach when using a single neural network is filling $\vec{t}_{DC}[n]$ with arbitrary numbers and using conventional training. The problem, however, is that the vectors $\vec{\tau}[n]$, may contain components $\vec{t}_{DC}[n]$ that significantly and undesirously affect the generalization of the trained neural network. An alternate approach is therefore required.

2 *Don't Care* Training Data

For *don't care* training, the partition of the output, varying for each training data pair, is that in (1). The input representation is augmented. The first component, $\vec{i}[n]$, of the augmentation contains the conventional input training data. The second component, $\vec{i}_{DC}[n]$, dubbed the *don't care* input, contains characterization and statistics of those output values with a "don't care" status. The assignment of each output neuron to either $\vec{t}_{DC}[n]$ or $\vec{t}[n]$, for example, is determined by $\vec{i}_{DC}[n]$. The dimensions of $\vec{t}[n]$ and $\vec{t}_{DC}[n]$ vary with n as dictated by $\vec{i}_{DC}[n]$. The dimensions of $\vec{i}_{DC}[n]$ and $\vec{i}[n]$ are static. The *don't care* input is not used as conventional neural network input data but, rather, is used in

[1]University of Washington, Computational Intelligence Applications Laboratory, Department of Electrical Engineering, Seattle, WA. [2]Applied Physics Laboratory, University of Washington, Seattle, WA.

0-7803-7044-9/01/$10.00 ©2001 IEEE

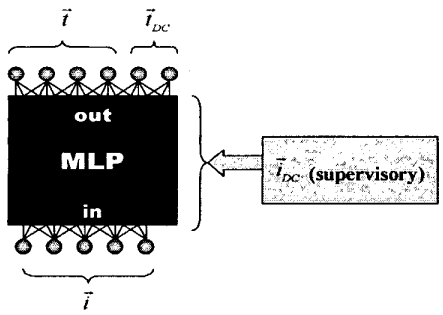

Figure 1: The neural network training architecture.

- **training** to alter learning parameters as a funtion of n, and
- **testing** to specify which of the output neurons have *don't care* values and therefore should be ignored.

3 Step Size Modification

During error backpropagation training, the weights connected to the *don't care* output neurons are not updated while other weights are updated with a modified step size. An empirical *a posteriori* probability showing how often the weights associated with each output neuron are updated is obtained from the training data set and is used to give even amount of opportunity for weight update to every output neuron. The empirical probability of weight update associated with the mth output neuron is defined as the ratio of the frequency of weight update associated with the output neuron m, denoted by $f_o[m]$, to the total number of training patterns, N.

$$p_o[m] = \frac{f_o[m]}{N}.$$

It is reasonable to give a large gain to the weight that has less opportunity of correction. Thus, the step size modification (SSM) is defined by

$$\begin{cases} \eta_m^o = \frac{\eta}{p_o[m]} & \text{; for active output neurons} \\ \eta_m^o = 0 & \text{; for } \textit{don't care} \text{ output neurons} \\ \eta^h = \eta & \text{; for all other neurons} \end{cases} \tag{3}$$

where η is a global learning rate used for the weight update of ordinary error back propagation, η_m^o is the modified learning rate for the weights of output neuron m and η^h is the step size for all other weights. The difference between the target output value, $t[n]$, and the actual output, $o[n]$, of neuron n included in $o[n]$ at the nth training pattern vector is used to take the instantaneous sum of squared errors of the network.

A commonly used variation is batch-mode learning. The nth pattern is evaluated to obtain the derivative terms, $\frac{\partial E(n)}{\partial w}$, which are summed to obtain the total derivative, $\frac{\partial E}{\partial w} = \sum_{n=1}^{N} \frac{\partial E(n)}{\partial w}$, used in batch mode training. For *don't care* training, the sum is $\frac{\partial E}{\partial w} = \sum_{t_m[n] \in \text{Active}_n} \frac{\partial E(n)}{\partial w}$. The step size is modified to

$$\begin{cases} \hat{\eta}_m^o = \frac{\hat{\eta}}{p_o[m]} & \text{for every output neuron } m \\ \hat{\eta}_m^h = \hat{\eta} & \text{otherwise} \end{cases}$$

where $\hat{\eta}$ is a global learning rate used for the batch-mode weight update of ordinary error back propagation, $\hat{\eta}_m^o$ is the modified learning rate for the weights of output neuron m and $\hat{\eta}^h$ is the step size for all other weights.

The mean squared error (MSE) is obtained by summing $E[n]$ over all n and then normalized with respect to the set size N [5],[6]. Specifically

$$E(n) = \frac{1}{2} \sum_{m \in \text{ ACTIVE}} (t_m[n] - o_m[n])^2,$$

and

$$E_{\text{MSE}} = \frac{1}{N} \sum_{n=1}^{N} E(n).$$

The MSE, however, is an inappropriate represention of training error for variable output dimensionality. For *don't care* training, rather, the average mean squared error (AMSE), representing the mean squared error per each output element, is more appropriate.

$$E_{\text{AMSE}} = \frac{1}{N} \sum_{n=1}^{N} \frac{E(n)}{f_o[n]}. \tag{4}$$

An output neuron, even when sparsely used, is represented equally in the composite error totally.

4 Performance Contrast: A Sonar Example

Sonar data corresponding to various environmental and sonar control parameters was generated from an computationally intensive acousic model. The ensonification[1] map is arranged in a 75 (range) × 20 (depth) pixel image.[2] There are therefore (a maximum of) 1500 = 75 × 20 values for each training data pair

[1] As measured by the acoustic transmission loss. More details about the acoustic emulation are in Jensen *et.al.*[7].

[2] The range is from zero to 75 km. The depth is from zero at the surface to 0.5 km. Sampling is uniform.

output. The *bathymetry* (*e.g.* the shape of the ocean's floor) is one of the environmental parameters varied. When a neural network is trained with the ensonification map as output, neurons asigned to pixels lying below the ocean bottom cannot be ensonified and are therefore classified as *don't care* output neurons. Twenty eight environmental and control parameters, detailed in Table 1, were used as inputs. A total of 5000 input-output profiles were used in the training of the neural network and 3000 were used to test.

The comparison of typical training performance with the algorithms using fixed arbitrary numbers and smearing method as stated earlier is shown in Figure 2. *Don't care* training invariably outperforms the other algorithms in terms of training error as well as convergence time. Figures 4 and 5 illustrate testing examples where the neural networks and their absolute errors are compared with the desired values. Points below the bathymetry correspond to unspecified output nodes ascribed a *don't care* status for this specific pattern. The plot labeled **Target** is desired ensonification map, NNA is the neural network trained using arbitrary fixed numbers, NNB is the neural network trained smearing, and NNC is the neural network trained by the SSM training technique and the ASME error in (4). The output nodes adjoining unspecified region have big errors in NNA, whereas NNB improved this problem significantly. However, NNB still has a big error on the rest of the region. If we take a look at maps on the right hand side column illustrating absolute differences, it is obvious that NNC, in terms of final accuracy, outperforms the other two techniques.

5 Conclusion

A novel neural network learning algorithm for data sets with varying output dimension is proposed in this paper. The possible memorization problem caused by irregular weight correction is avoided by employing step size modification.

Acknowledgments

This work is supported by the Office of Naval Research.

References

[1] R.A. Jacob and M.I. Jordan, "Hierarchical Mixtures of Experts and EM Algorithm", *Neural Computaion*, Vol.6 pp. 181-214, 1994.

[2] R.A. Jacob and M.I. Jordan, "Learning Piecewise Control Strategies in a Modular Neural Network 'architecture", *IEEE Transactions on System, Man, and Cybernetics*, Vol.23, No. 2, pp. 337-345, 1993. Addison-Wesley, 1989.

[3] Bart L.M. Happel and Jacob M.J. Murre, "Design and Evolution of Modular Neural Network Architecture", *Neural Networks*, Vol.7, Nos. 6/7, pp. 985-1004, 1994.

[4] Sherif Hashem, "Optimal Linear Combinations of Neural Networks", *Neural Networks*, Vol.10, No. 4, pp. 599-614, 1997

[5] S. Haykin, **Neural Networks**, IEEE Press, 1994.

[6] R.D. Reed and R.J. Marks II, **Neural Smithing : Supervised Learning in Feedforward Artificial Neural Networks**, MIT Press, 1999.

[7] C.A Jensen, R.D. Reed, R.J. Marks II, M.A. El-Sharkawi, Jae-Byung Jung, R.T. Miyamoto, G.M. Anderson, C.J. Eggen, "Inversion of feedforward neural networks: algorithms and applications", *Proceedings of the IEEE*, Volume 87, Sept. 1999, Pages: 1536 -1549

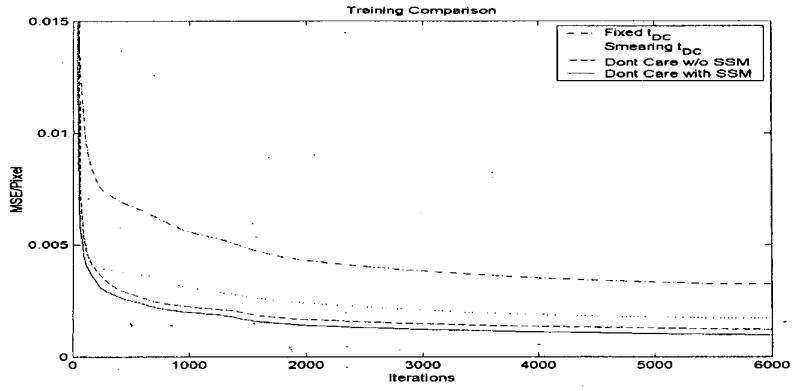

Figure 2: Training performance comparisons with other training algorithms where pixels below the bathymetry are treated differently. The top curve corresponds to fixing the *don't care* pixels to arbitrary values. The next plot, corresponding to a sequence of dots, results from replacing column of pixels under the bathymetry equal to the deepest pixel value in the column. We dub this procedure as *smearing*. The second plot from the bottom, shown as a broken line, is for *don't care* training without SSM. The bottom solid plot is *don't care* training with SSM.

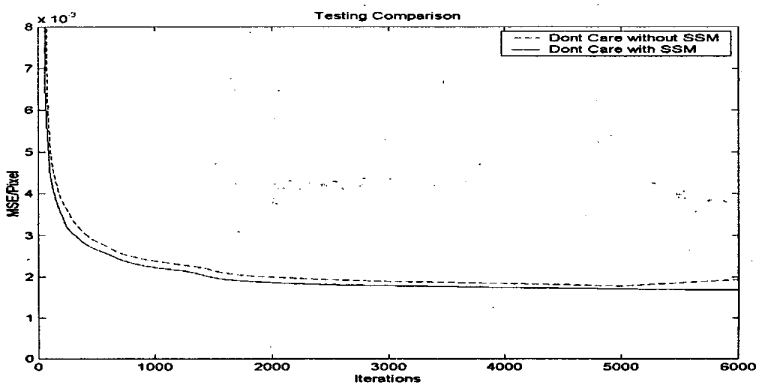

Figure 3: Test error for *don't care* training with and without SSM. Both cases used E_{AMSE} in (4). The results of 5%, shown here, are typical for this problem.

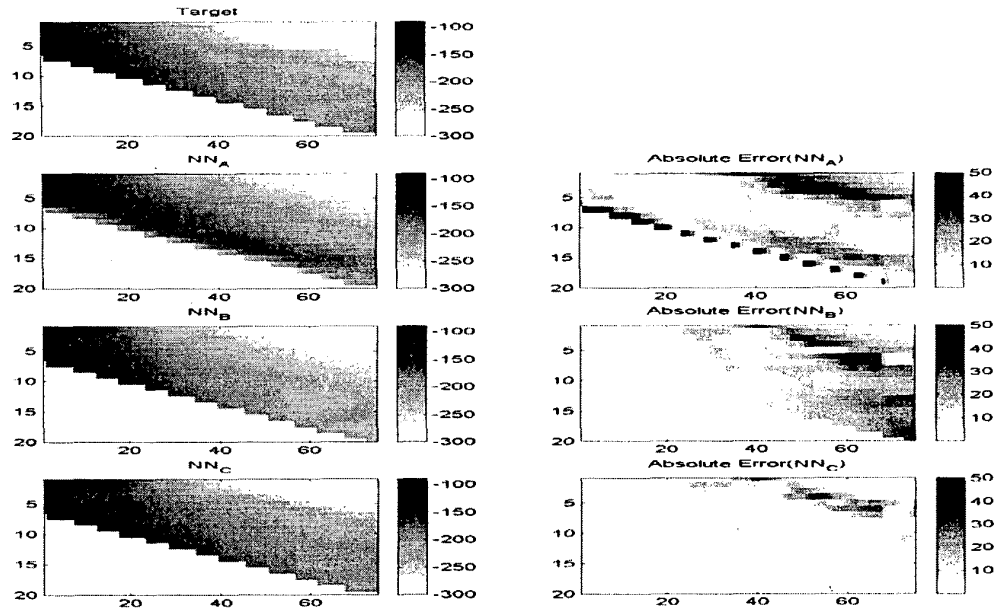

Figure 4: Neural network performance comparison using a testing pattern.

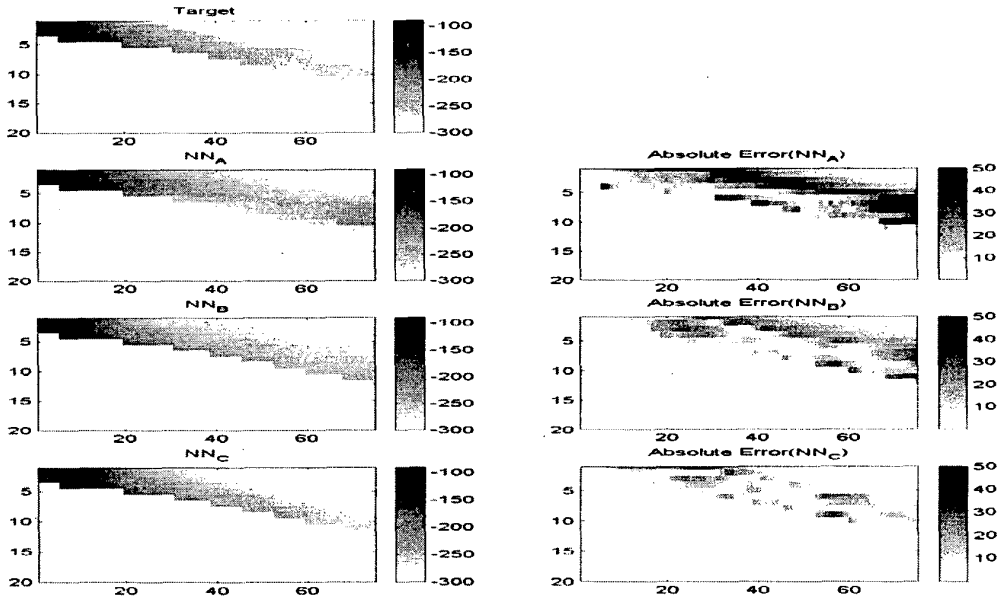

Figure 5: Neural network performance comparison using a testing pattern.

Table 1: Environmental and control parameters used as inputs to train the neural network.

No	Input parameter
1	Transmitter depth [m]
2	Receiver depth [m]
3	Pulse Length [s]
4	Bandwidth [Hz]
5	Center Frequency [Hz]
6	Total Noise [dB]
7	Target echo duration [s]
8	Wind speed [m/s]
9	Volume scattering strength [dB/m]
10	Bottom type, grain size[mm]
11	Bathymetry 1 (bottom depth at sonar) [m]
12	Bathymetry 2 (bottom range in the middle) [m]
13	Bathymetry 3 (bottom depth in the middle) [m]
14	Bathymetry 4 (bottom depth at 15 km) [m]
15	Sound speed at surface [m/s]
16	Sound speed at 10 m [m/s]
17	Sound speed at 20 m [m/s]
18	Sound speed at 30 m [m/s]
19	Sound speed at 50 m [m/s]
20	Sound speed at 75 m [m/s]
21	Sound speed at 100 m [m/s]
22	Sound speed at 125 m [m/s]
23	Sound speed at 150 m [m/s]
24	Sound speed at 200 m [m/s]
25	Sound speed at 250 m [m/s]
26	Sound speed at 300 m [m/s]
27	Sound speed at 400 m [m/s]
28	Sound speed at 500 m [m/s]