# Expanding the Training Set for Better Generalization in MLP

Debrup Chakraborty and Nikhil R. Pal
Electronics and Communication Sciences Unit
Indian Statistical Institute
Calcutta 700108, India
Email: {debrup_r,nikhil}@isical.ac.in

***Abstract*— A method to improve the generalization ability of a multilayered perceptron (MLP) network is proposed here. The method expands a given training set and trains an MLP with a new data set in each epoch. The method of data generation maintains the spatial density of the original training sample. Experiments show that the method can yield excellent generalization.**

***Index Terms*— MLP, Generalization, Mountain Potential, k-nearest neighbors**

## I. Introduction

A training set $\mathcal{L}$ contains data $\{(\mathbf{x}_n, y_n), n = 1, ..., N\}$, where $\mathbf{x}$ is a feature vector and $y$ is its corresponding numerical response or a class label. There are plenty of procedures available in literature which use this training set $\mathcal{L}$ to form a predictor function $\phi(\mathbf{W}, \mathbf{x})$. Where $\mathbf{W}$ is a parameter vector which is decided using $\mathcal{L}$. A very popular procedure of obtaining the predictor $\phi$ is by training a feed-forward neural network (like a multilayered perceptron (MLP)) with $\mathcal{L}$. In this case the parameter vector $\mathbf{W}$ becomes the parameters (weights and biases) of the neural network, which are learned with the aid of the training set $\mathcal{L}$ and the optimization procedure selects a $\mathbf{W}$ which minimizes the error on the training set. The operational performance measure for the trained network is the error on future data outside the training set, also known as the generalization error. This error may be undesireably large, when, for example the training set size is too small compared to the network parameter set size. Practice has shown that a direct minimization of the training error for a given fixed training set by backpropagation or similar type of training algorithm does not necessarily imply a minimization of the generalization error. A common means to bypass this difficulty is to use a validation set to judge the generalization ability and decide on the stopping time for training. Researchers have also followed a variety of different approaches to improve generalization like pruning [8], weight sharing [2], and complexity regularization [6], [7].

The problem of bad generalization and overfitting stems from the fact that neural networks are trained with a finite training sample. The available training data are reused in every epoch and as a result, the neural network "concentrates" more and more on these points and often results in a bad generalization. A probable solution to this problem would be to have an infinitely large training set which is seldom realizable in practice. Here we propose a method to expand the training set to any required size, for better generalization. A related concept, suggested in [3] adds noise to the training set. In [3] the authors considered adding additive white noise independently to the input and output vectors to generate new training samples. In [4], Karistinos and Pados gave an algorithmic procedure for random expansion of a given training set. They proposed a locally most entropic estimate of the true joint input-output probability density function of the training sample to generate new training samples. They argued that the method in [3] is an extreme special case of that in [4]. Here we present a scheme which generates new input vectors following the same density of the training sample and uses a k-nearest neighbor regression heuristic to generate the output of the corresponding input vector. Our experiments demonstrate that our method can produce nice generalizations. Our method is different from that in [4] as our method does not require an explicit density estimation, which is known to be unstable [10].

## II. Expanding the training set

It is assumed that the training data are obtained from an unknown time invariant probability distribution. Thus expansion of the training data can be best done if we attempt to learn the unknown probability distribution from which the data were generated and generate additional samples from the obtained distribution. Estimating probability distributions from data is an ill posed problem whose solutions are again unstable and tends to become inaccurate with the increase in dimensionality of the data [10], [11]. Here we do not attempt to learn the probability distribution of the data but we exploit its spatial distribution to generate additional data points which maintains the spatial distribution of the given training set.

Let us denote the the input vector $\mathbf{x}_i$ augmented with its output $y_i$ (which may be single valued or a vector, without loss of generality we consider a single valued output) by $\tilde{\mathbf{x}}_i$. For each data point $(\mathbf{x}_i, y_i)$ in $\mathcal{L}$, we assign a probability $p_i$ which is a function of the density of input data points in the neighborhood of $\mathbf{x}_i$. We model $p_i$ using the mountain potential which has been used successfully for numerous clustering applications [5], [13]. Thus

$$p_i = \frac{1}{Z} \sum_{j=1}^{N} exp(-\alpha ||\mathbf{x}_i - \mathbf{x}_j||), \tag{1}$$

where $Z$ is a normalizing constant which is so chosen that $\sum_{i=1}^{N} p_i = 1$. Our algorithm samples a point $\mathbf{x}_i$ from $\mathcal{L}$ with probability $p_i$. Then it finds the $k$ nearest neighbors of $\tilde{\mathbf{x}}_i$, we call them $\tilde{\mathbf{x}}_i^1, \tilde{\mathbf{x}}_i^2, ..., \tilde{\mathbf{x}}_i^k$. A new point $\tilde{\mathbf{x}}_{new}$ is generated as a convex combination of the points $\tilde{\mathbf{x}}_i^1, \tilde{\mathbf{x}}_i^2, ..., \tilde{\mathbf{x}}_i^k$, i.e,

$$\tilde{\mathbf{x}}_{new} = \sum_{j=1}^{k} \lambda_j \tilde{\mathbf{x}}_i^j, \tag{2}$$

where

$$\sum_{j=1}^{k} \lambda_j = 1, \quad 0 \leq \lambda_j \leq 1.$$

The $\lambda_j$'s are randomly generated. Note that $\tilde{\mathbf{x}}$ represents a new input output pair but the probability distribution assumed for sampling, depends on the input vectors only. In each step our algorithm samples a point according to the density, and generates a point in the neighborhood of it. So, more points will be generated in dense regions and less points in the sparse regions. This algorithm has two user defined constants the window size $\alpha$ in eq. (1), and $k$ which denotes the number of nearest neighbors considered. Here we use $\alpha = 0.5$ and $k = 5$ for simulations, which works quite well.

An MLP is trained with these generated data points. The MLP gets trained with the original training set $\mathcal{L}$ in the first epoch and in the subsequent epochs it faces points generated by the above described algorithm. Thus in each iteration, the MLP faces a new set of points.

## III. Results

Here we present results on two function approximation problems.

### A. Sine data

We consider the problem of learning a noisy sine curve [4]

$$f(x) = 0.4sin\ x + 0.5, \quad x \in R. \tag{3}$$

As in [4], we generate 36 input-output pairs $\tilde{\mathbf{x}}_i = (\mathbf{x}_i, y_i)$, $i = 1, .., 36$, with $\mathbf{x}_i$ generated uniformly on $[-\pi, \pi]$ and $y_i = f(x_i) + r_i$, where $r_i \sim \mathcal{N}(0, \sigma^2)$ accounts for the noise with variance $\sigma^2 = 0.01$.

We trained MLPs with this data set and also MLP's with the generated data points. In each case we used sigmoidal activation functions, used 7 nodes in the hidden layer and used the Levenberg-Marquardt algorithm to update the weights. Figure 1 shows the generalization produced for 8 MLPs trained with only the 36 points repeated in each epoch. Figure 2 shows the generalization for the generated data. The figures reveal that generating additional data in each epoch gives better and consistent generalization. In Fig. 1 out of 8 MLPs, 4 cases ((a), (c), (f) and (h)) exhibit very bad generalization, while no such case arises in Fig. 2. Even the smoothness of the output when the nets are trained with the original data is much less than in the cases where training is done using data generated by our method. The additional data act as constrains (regularizing force) on the network and prevents it from making poor generalization.

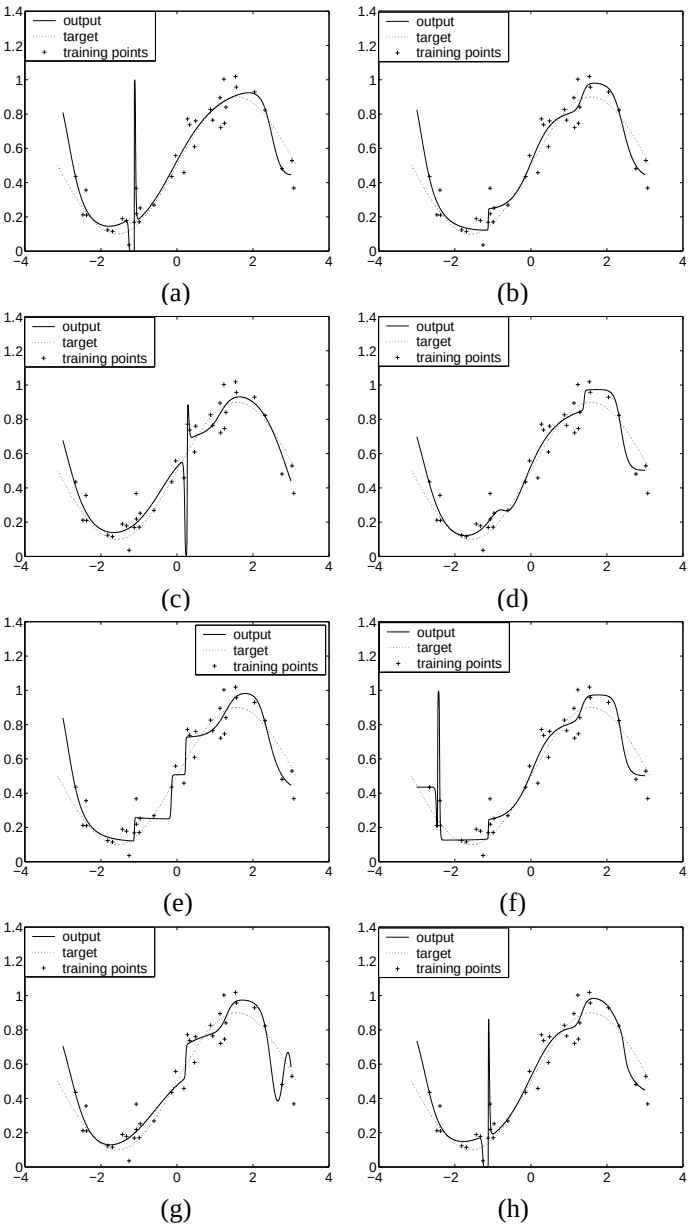


Fig. 1. Generalization produced by ordinary MLP

### B. Chemical Plant data

Chemical Plant data contains data for operator's control of a chemical plant for producing a polymer by polymerization of some monomers. There are five inputs and one output. The input variables are monomer concentration ($u_1$), change of monomer concentration ($u_2$), monomer flow rate ($u_3$) and the two local temperatures inside the plant ($u_4$ and $u_5$). The only output ($y$) is the set point for monomer flow rate. In [12] there is a set of 70 data points obtained from an actual plant operation. For our convenience we have normalized $u_3$ and $y$ such that they lie between [0,1], as they have greater magnitudes compared to others.

The results obtained on Sine data could be easily plotted to show that our method produces nice generalizations. But as the Chemical Plant data is a high dimensional data such visual comparison is not possible. Also, this is a comparatively small

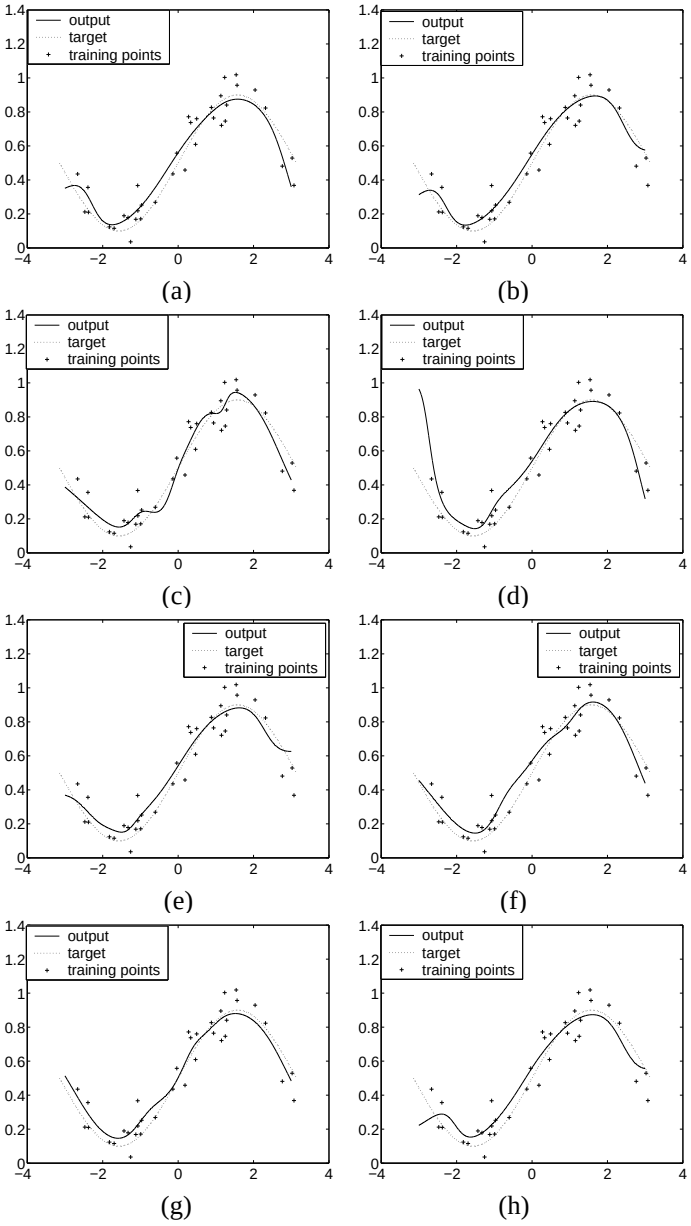


Fig. 2. Generalization produced by MLP trained with generated data

sized data set, hence we use all the data points for training and test the performance on an artificially generated data set. We generate 100 points within the approximate multidimensional convex hull of the 70 input vectors. We use the method used by Smith and Jain in [9] to generate points inside the approximate convex hull.

A method which produces consistent and smooth generalization must be "stable" [1], i.e., the output must not change much with a small change in input. We give a measure here to test the stability of a network. Let us denote the set of test input vectors as $\mathcal{T} = \{\mathbf{t_i} : i = 1, ..., m\}$ and the training set as $\mathcal{L} = \{(\mathbf{x}_i, y_i\} : i = 1...n\}$. Let $o_i$ be the output of the trained MLP for the test point $\mathbf{t}_i$. We calculate the measure of stability (S) as

$$S = \sum_{i=1}^{m} |o_i - y_k|\,,\ where,\quad k = argmin_j ||\mathbf{t}_i - \mathbf{x}_j||. \quad (4)$$

Hence $S$ measures the sum deviations of the output of a test point $\mathbf{t}_i$ from the output of the training point nearest to $\mathbf{t}_i$. Thus a more stable method should have a lower value of $S$. Also for a stable method the value of $S$ should be consistent over several runs.

We trained 100 ordinary MLPs and another 100 MLP's using our method of data generation. For both cases we used the Levenberg-Marquardt algorithm for training and used 10 nodes in a single hidden layer and each node having sigmoidal activation function. We calculated the values of $S$ for all runs. The average value of $S$ over 100 MLP runs using conventional training was 12.9691 and that for MLP's trained by our method was 12.5106. Thus on average our method produces a lesser value of $S$. Also for our method the value of $S$ is more consistent over runs. Fig. 3 shows the histogram for the values of $S$ for 100 runs of both ordinary MLP and the MLPs trained with the generated data. Fig. 3 shows that with usual MLP training for some runs $S$ can take a value as high as 21, but for our method, no run produces a $S$ greater than 15.5.

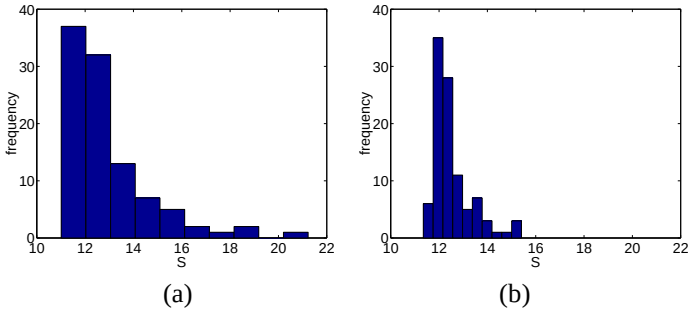


Fig. 3. Histogram for values of $S$ for 100 runs on Chemical Plant data : (a) Ordinary MLP (b) MLP trained with generated data

## IV. Conclusions and Discussion

We presented a simple method to improve the generalization ability of an MLP. Our method involves expanding the available training set and using new data points in each epoch to avoid overfitting.

The method developed here is for function approximation problems, but it may be extended for classification problems with minor modifications. Our results demonstrated that our method can yield better generalization compared to usual MLP training. There are a few issues that we like to address in near future : We shall compare our method with other methods which improve generalization. Our method requires two user defined parameters. So far we have experimentally determined suitable values for them. We shall try to provide some general guidlines on this. We also plan to extend our method to train networks with "small" training sets.

## References

[1] L. Breiman, "Bagging Predictors", *Machine Learning*, vol. 24, no. 2, pp. 123-140, 1996.

[2] Y. le Cun, B. Boser, J.S. Denker, D. Henderson, R.E. Howard, W. Hubbard, and L.D. Jackel, "Backpropagation applied to handwritten zip code recognition", *Neural Computation*, vol 1, no 4, pp. 541-551, 1989.
[3] L. Holmstrom and P. Koistinen, "Using additive noise in backpropagation training", *IEEE Trans. Neural Networks*, vol 3, pp. 24-38, 1992.
[4] G.N. Karystinos and D.A. Pados, "On overfitting, generalization, and randomly expanded training sets", *IEEE Trans Neural Networks* vol 11, no 5, pp. 1050-1057, 2000.
[5] N.R. Pal and D. Chakraborty," Mountain and Subtractive Clustering Method: Improvements and Generalizations", International Journal of Intelligent Systems, vol 15, pp.329–341, 2000.
[6] T. Poggio and F. Girosi, "Networks for approximation and learning", *Proceedings of the IEEE*, vol 78, pp. 1481-1497.
[7] T. Poggio and F. Girosi, "Regularization algorithms for learning that are equivalent to multilayered networks", *Science*, vol. 247, pp. 978-982.
[8] R. Reed, "Pruning algorithms - a survey", *IEEE Trans. Neural Networks*, vol 1, vol 4, pp. 740-747, 1993.
[9] S.P. Smith, A.K Jain, "Testing of uniformity in multidimensional data", *IEEE Trans. on Pattern Analysis and Machine Learning*, Vol 6, no 1, pp.73-81, 1984.
[10] A.N. Tikhonov and V.Y. Arsenin, *Solution of ill posed problems*, Washington D.C.: W.H. Winston, 1977.
[11] J. Weston, A. Gammerman, M. Stitson, V. Vapnik, V. Vovk and C. Watkins, "Support vector density estimation", *Advances in Kernel Methods — Support Vector Learning*, MIT Press, Cambridge, MA, pp. 293-306, 1999.
[12] M. Sugeno and T.Yasukawa, " A Fuzzy-Logic based approach to qualitative modeling", *IEEE transactions Fuzzy Systems*, vol 1, no 1, pp. 7-31, 1993.
[13] R. R. Yagar and D.P. Filev, "Approximate Clustering via the Mountain Method", *IEEE Trans. Systems man Cybernetics*, vol 24, no 8, pp. 1279-1284, 1994.