Instant Learning for Supervised Learning Neural Networks: A Rank-Expansion Algorithm

C. L. Philip Chen and Jiyang Luo†

Department of Computer Science and Engineering
Wright State University, Dayton, OH 45435
†Department of Computer Science and Technology
Tsinghua University, Beijing, PR China

ABSTRACT

An one-hidden layer neural network architecture is presented. An instant learning algorithm is given to decide the weights of a supervised learning neural network. For an n dimensional, N-pattern training set, a maximum of $N-r$ hidden nodes are required to learn all the patterns within a given precision (where r is the rank, usually the dimension, of the input patterns). Using the inverse of activation function, the algorithms transfer the output to the hidden layer, add bias nodes to the input, expand the rank of input dimension. The proposed architecture and algorithm can obtain either exact solution or minimum least square error of the inverse activation of the output. The learning error only occurs when applying the inverse of activation function. Usually, this can be controlled by the given precision. Several examples show the very promising result.

Keywords:

Neural networks, Supervised Learning, Instant Learning, Bias nodes, Rank-Expansion.

Notations:

N :	Number of patterns in the training set.
n :	Dimension of an input pattern, $n \leq N$.
m :	Dimension of an output pattern.
A :	Matrix of input patterns, with N rows and n columns.
r :	Rank of matrix A, input patterns of the training set listed in matrix form, $n \leq r \leq N$.
$\underline{A}$:	An N by $(N-r)$ matrix, where the columns of this matrix linearly independent with the A matrix.
$\hat{A}$:	A new matrix combining A and $\underline{A}$, $\hat{A}=[A \mid \underline{A}]$.
T :	Output patterns listed in matrix form, N rows and m columns
σ :	Nonlinear activation function, $\sigma(x) = \dfrac{e^x-e^{-x}}{e^x+e^{-x}}$.
σ^{-1} :	Inverse of nonlinear activation function. $\sigma^{-1}(x) = \dfrac{1}{2}\ln\dfrac{1+x}{1-x}$.
B :	A matrix obtained by applying σ^{-1} to each element of T. When the element of T is -1 or 1, a smaller value (e.g., -0.999 or 0.999) is given depending upon precision requirement, before applying σ^{-1}.
$[A \mid B]$:	The matrix combining A and B.
W :	Weight matrix of the network.

1. Introduction:

The function of a node in a neural network can be described as: $o = \sigma\left(\Sigma_i w_i x_i\right)$, where o is the output of the node, x_i is the *ith* input, w_i is the synaptic connection, or weight, for the *ith* input, and σ is the nonlinear activation function,

$$\sigma(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}.$$

Consider a multi-layer neural network with n input nodes and m output nodes. For a given training set of input patterns and corresponding output patterns, the supervised learning aims to learn all the patterns. That is to decide all the synaptic connections or to find out all the weights, such that the network will map the input in R^n to the output in R^m. Let V_i be the output of the *ith* layer; the weight matrix W^i be the synaptic connections between layer i and layer $i+1$; The output function of the *ith* layer, V_i, can be represented by a linear matrix operation and nonlinear σ operation, i.e., $V_i = \sigma\ (\ V_{i-1} \bullet W^{\ i-1})$,

Recently, effort has been made on design a network architecture for fast learning, adjust initial weight matrix for learning, and design a fast learning algorithm [1-4]. The existing back-propagation algorithm decides the weight matrices using gradient descent methods. Based on back-propagation training algorithm, the networks converge to the smallest error after repeatedly changing the weights on the gradient direction. However, the network architecture must be given before the training can be started, and the long-lasting training time has been criticized recently [1-4].

Least Squares Back-propagation (LSB) is a learning algorithm that bases on linear least squares computation [5]. In this algorithm, the training of a multi-layer neural network is performed by separating the linear and nonlinear parts of each layer. The output of each layer is converted by the inverse of the nonlinear activation function, and the network is trained by a linear least squares computation. In many cases, however, one pass of this algorithm can not train the network to obtain the least error. In order to reduce the error, the authors suggested to iterately apply linear least squares computation back and forth through the layers. But unfortunately, for many problems, though the error could be reduced to a smaller number after the first pass, it would not continue to decrease further on. Another disadvantage is that this algorithm does not suggest a good network architecture.

In order to reduce training time, Cascade-Correlation aims to learn the training set through a dynamic procedure, starting from the simplest architecture without hidden nodes [6]. If the error remains large, then the algorithm dynamically adds a node in a new hidden layer, adjusts the weights for this hidden node, and again trains the output layer of the new network to achieve less error. In this way it is possible to achieve least hidden nodes for the neural network, but it takes much time to train the weights of both the hidden nodes and the output layer using gradient descent method. Also, since the training may involve many hidden nodes each in a different layer, this also increases the responding time from receiving the input to turning out the output.

In this paper, we propose an one-hidden layer architecture and design a fast learning algorithm that obtains the weight matrices with only a few matrix operations. With at most $N-r$ hidden nodes (where r is the rank of the input patterns forming as a matrix form, usually the dimension of the input), the designed neural network maps the input patterns in the training set to the desired output patterns within required precision. In our method, the linear and nonlinear parts of the output of the output layer are separated. We then apply the inverse of the nonlinear activation function to the outputs. This way the optimization of the weight matrices can be done without the nonlinear part. The weight matrices can be found by solving linear matrix computation. Next section, the supporting theorem and architecture are given.

2. Theorem and Algorithm

2.1. The Theorem

Theorem 1: Consider a systems of equations $AW = B$, and denote the augmented matrix of the system $A' = [\ A\ \mid B\ ]$. Assume that W is an $n*m$ matrix. (1) The system $AW = B$ has a solution if and only if $rank(A) = rank(A')$. (2) The system $AW = B$ has a unique solution if and only if $rank(A) = n = rank(A')$.

Proof: See [7].

In the supervised learning problem, given the A and T matrix, the problem is to find the weight matrix, W. In order to eliminate the non-linear computation on the output, the B is found by taking the inverse of the σ function, i.e., $B = \sigma^{-1}\ (\ T\ )$. In order to have a solution or a unique solution, the above conditions must be met. However, in supervised learning, the A matrix (the training pattern), can not always meet the above requirement for most problems. For the training matrix, A, with dimension $N * n$, the output matrix, T (thus B), with dimension $N * m$, the W matrix must be well defined.

In order to solve the above equation, the back-propagation algorithm and LSB algorithm try to solve the W (either one hidden-layer or multiple layers) using gradient decent or pseudo-inverse of A matrix, respectively. Functional link neural network, however, expands the input dimension to a higher dimension by adding the higher order nodes to satisfy, hopefully, the full rank condition. If the rank of A can be increased from r to $rank(A') = n$, then the unique solution, W, can be solved easily. In other words, A matrix will have a full rank if $n-r$ additional linearly independent columns are added to it. Denote the new formed matrix, $\hat{A} = [\, A \mid \underline{A} \,]$, where $\underline{A}$ is the matrix representation of the additional linearly independent columns. In most case, if $n = r$ and $rank(A) \neq rank(\,[A \mid B\,]\,)$, then the rank of A can be increased to $rank(\hat{A}) = q = rank(\,[\,\hat{A} \mid B\,]\,)$ by adding $q-r$ additional linearly independent columns, then a unique solution, $\hat{W}$, can be found. If $r = n$ and $N-r$ additional linearly independent columns are added to the A matrix then $\hat{A}$ is the square matrix. As a result, $rank(\hat{A})$ equals to $rank([\,A \mid \underline{A} \mid B\,])$. The weight matrix W is simply $A^{-1} \bullet B$.

2.2 The Algorithm

As we mentioned previously, the maximum number of hidden nodes to be added is $N - r$. Obviously, we can expand the A matrix to $\hat{A}$ directly by adding $N - r$ linear independent columns and solve $W = A^{-1} \bullet B$ instantly. The rank of matrix A can be easily increased by one if a constant bias node with strength 1 or -1 is always accompanied with the input. In this case, there are $N - r - 1$ hidden nodes.

Based on the above discussion, the algorithm is below.

```
Algorithm Rank-Expansion with Instant Learning (REIL).
Input:  The input pattern matrix, $A$, the output matrix, $T$, precision, and the error tolerance.
Output: The weight matrix, $W$, and the neural network.

Step 1. Chop output elements in $T$ to be within (-1,1) according to the precision.
Step 2. Calculate $B = \sigma^{-1}(T)$.
Step 3. Add a constant bias node "1" ( or "-1") to the input node.
Step 4. Add $N-r-1$ hidden nodes and assign random weights.
Step 5. Solve weight, $W = \hat{A}^{-1} \bullet B$.
End of Algorithm REIL
```

3. Testing Examples

Example 1: Find the network weight for the given problem [8] (a constant bias "-1" is added to the A matrix):

$$\hat{A} = \begin{bmatrix} \frac{1}{4} & \frac{1}{2} & \frac{3}{4} & \frac{3}{4} & \frac{1}{4} & \frac{1}{2} & \frac{3}{4} & \frac{1}{4} & \frac{1}{2} \\ \frac{1}{4} & \frac{1}{8} & \frac{1}{8} & \frac{1}{4} & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \frac{3}{4} & \frac{3}{4} \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \end{bmatrix}^T, \quad T = \begin{bmatrix} 1 & -1 & -1 \\ -1 & 1 & -1 \\ 1 & -1 & -1 \\ -1 & 1 & -1 \\ -1 & -1 & 1 \\ -1 & -1 & 1 \\ -1 & -1 & 1 \\ -1 & 1 & -1 \\ 1 & -1 & -1 \end{bmatrix}$$

We use the algorithm REIL to solve this problem. We add hidden bias nodes one by one. Figures 1 and 2 show the convergeness of the algorithm. From Fig. 1 and Fig. 2, we can see that after adding the fifth hidden node, the error is already ignorably small (maximum error = 0.0639242, Mean-Squared Error = 0.000004). If we add one additional node to the hidden layer, The result is shown in Fig. 3 and Fig. 4. Now the maximum error is 0.001, the amount we chopped off, and The Mean-Squared Error is 0. The training only involves only a few matrix operations, and we are able to obtain the solution instantly.

For this problem, we also applied Least Squaies Back-propagation [5] to train a one-hidden-layer network with 6 hidden nodes, the result varies each time we run the algorithm. Fig. 5 (Maximum error) and Fig. 6 (Mean-Squared-Error) show the best result of many runs. Fig. 7 (Maximum error) and Fig. 8 (Mean-Squared-Error) show the result of the worst case in many runs. In our simulation, the maximum error remains larger than 1.0 in most cases. Similarly, we also applied Back-propagation to solve this problem, using a neural network with one hidden layer of 6 nodes, after 2000 epochs, the training error still remained large. Each epoch takes more time than we add a bias node using Algorithm REIL. The training result is shown in Fig. 9 (Maximum error) and Fig. 10 (Mean-Squared-Error).

Example 2: Consider a nonlinear mapping of eight inputs, x_i, and three outputs, y_i, defined by [5]

$$y_1 = (x_1 * x_2 + x_3 * x_4 + x_5 * x_6 + x_7 * x_8)/ 4$$
$$y_2 = (x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8)/ 8$$
$$y_3 = (1 - y_1)^{0.5}$$

The 50 sets of input patterns, x_i, $i = 1 \cdots 8$, are generated randomly within the range of (-1, 1); the outputs y_i, $i = 1, 2, 3$, are computed from the above three equations. Using Algorithm REIL, the learning result is significant. As shown in Fig. 11, the Mean-Squared Error reduces to 0.000746 at first run (with only constant bias -1). It decreases monotonely to 0.000014 after adding 40 bias nodes. Fig. 12 shows the maximum error decreasing as bias nodes added. With only a constant bias -1, the maximum error was 0.107, it decreased to 0.02 after adding 40 bias nodes. Table 1 lists the the error.

4. Conclusion

We have proposed a neural network architecture and algorithm for finding the weights of the supervised learning problem. We also provide the upper bound of the number of the hidden nodes to be able to solve the weight matrix exactly. The weight matrix, W, relys on the weight between the input and hidden layer. Once $rank(\hat{A}) = rank([\ \hat{A} \mid B\ ])$ a global minimum of the weight space with corresponding hidder-layer random weights can be obtained. Several tested examples show the efficiency the proposed architecture and algorithms.

5. References

[1] V. Nedeljkovic, "A Novel Multilayer Neural Networks Training Algorithms that Minimizes the Probability of Classification Error," IEEE Trans. on Neural Networks, Vol. 4, No. 4, pp. 650-659, 1993.

[2] Yoh-Han Pao, "Adaptive Pattern Recognition and Neural Networks," Addison-Wesley, N.Y., 1989.

[3] L. F. Wessels and E. Barnard, "Avoiding False Local Minima by Proper Initialization of Connection," IEEE Trans. on Neural Networks, Vol. 3, No. 6, pp. 899-905, 1992.

[4] P. L. Bartlett and T. Downs, "Using Random Weights to Train Multilayer Networks of Hard-Limiting Units," IEEE Trans. on Neural Networks, Vol. 3, No. 2, pp. 202-210, 1992.

[5] Friedrich Biegler-Konig and Frank Barmann, "A Learning Algorithm for Multilayered Neural Networks Based On Linear Least Squares Problems," *Neural Networks*, Volume 6, Number 1, pp.127-131, 1993.

[6] Scott E. Fahlman and Christian Lebiere, "The Cascade-Correlation Learning Architecture" Neural Information Processing Systems, Vol 2, 1990.

[7] Bill Jacob, Linear Algebra, W. H. Freeman and Company, NY, 1990.

[8] Jacek M. Zurada, "Introduction to Artificial Neural Systems" West Publishing Company, 1992

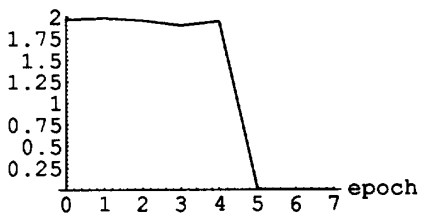
[9] R. W. Farebrother, "Linear Least Squares Computations" Marcel Dekker, Inc. New York, 1988.

[10] William W. Hager ,"Applied Numerical Linear Algebra" Prentice Hall, Eaglewood Cliffs, New Jersey.

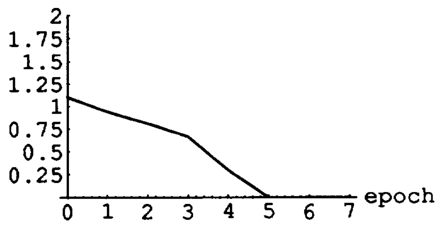
Table 1. Learning Error for Example 2 using REELS Algorithm

Added Nodes	Max error	Mean-Square error
1	0.1075	0.000746
5	0.0871	0.000612
10	0.0835	0.000507
12	0.0835	0.000446
24	0.0498	0.000217
36	0.0327	0.000051
40	0.0200	0.000014

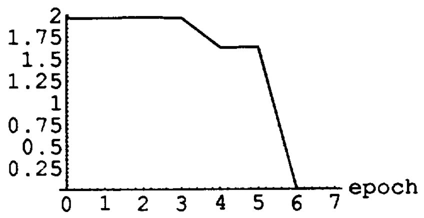
maxer:Fig. 1



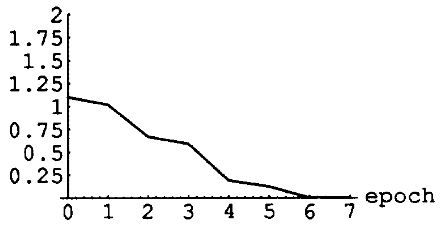
MSE:Fig. 2



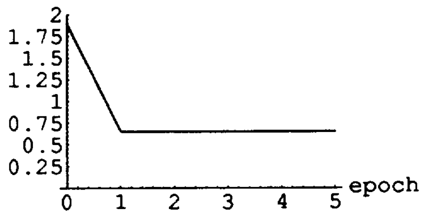
maxer:Fig. 3



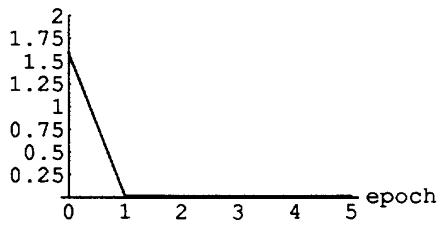
MSE:Fig. 4



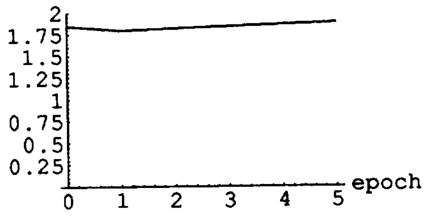
maxer:Fig. 5



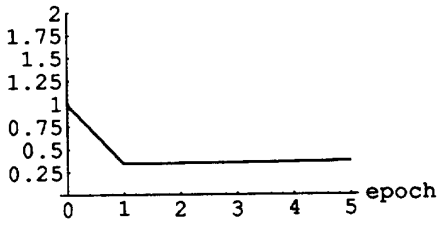
MSE:Fig. 6



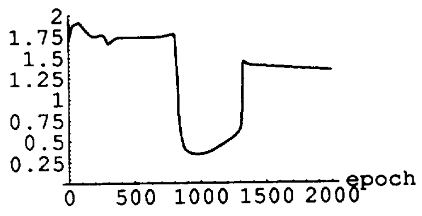
maxer:Fig. 7



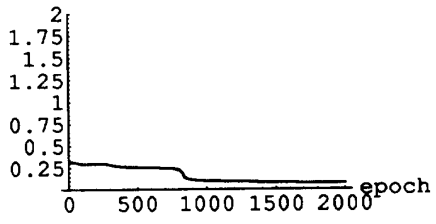
MSE:Fig. 8



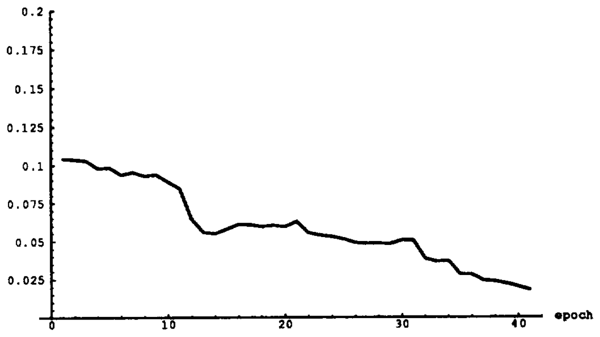
maxer:Fig. 9



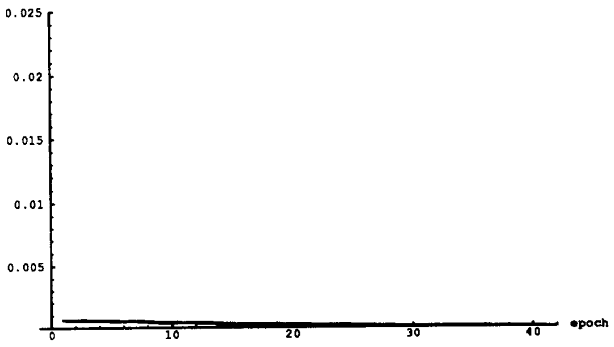
MSE:Fig. 10



ERR MAX: Fig. 11



MSE: Fig. 12


epoch