

Image Classification Using Elman Neural Network on Master-Slave Architecture

¹J. V. S. Srinivas, ²K. Priyadarsini, ³Dr. P. Premchand

¹DVRCET, Hyderabad, AP, India

²DRK IST, Hyderabad, AP, India

³OU, UCE, Hyderabad, AP, India

Abstract

Objective of this work is the classification of images using Elman Neural Network (ENN). ENN is one of the simplest supervised multi layer neural networks. We train the network with parallel algorithm on Master – Slave architecture to improve the performance. The performance of both sequential implementation and parallel implementations are evaluated. The parameters include speed-up, optimal number of processors and processing time.

Keywords

Image Set, ENN, Backpropagation(BP) algorithm, Parallel Implementation, Master-Slave architecture, Learning Rate

I. Introduction

Digital image processing is a collection of techniques to manipulate digital images by computers. Classification generally comprises four steps: 1. Pre-processing 2. Training 3. Decision and 4 Assessing. The objective of image classification is to identify the features occurring in an image in terms of the object these features actually represent. The studies were conducted on image Data Set consisting of 115 images of five classes (rockets, missiles, jets, aero planes and helicopters). Images are of size 400x200 in a scale of 256 values and each pixel of each image was scaled in the range 0-1.

Several classification algorithms have been developed from maximum likelihood classifier to neural network classifiers. This study emphasizes on the analysis and usage of a neural network classifier based on Elman neural network.

Artificial Neural Network (ANN) can provide suitable solutions for problems, which are characterized by non-linear data, high dimensionality noisy, complex, imprecise, and imperfect or error prone sensor data, and lack of a clearly stated mathematical solution or algorithm. The advantage with ANN is that we can design a model of the system from the available data. In this work image classification is done by applying the back propagation algorithm on Elman neural network.

Elman neural network is feed forward network with an input layer, a hidden layer, an output layer and a special layer called context layer. The output of each hidden neuron is copied into a specific neuron in the context layer. The value of the context neuron is used as an extra input signal for all the neurons in the hidden layer one time step later [6]. In an Elman network, the weights from the hidden layer to the context layer are set to one and are fixed because the values of the context neurons have to be copied exactly. Furthermore, the initial output weights of the context neurons are equal to half the output range of the other neurons in the network. The Elman network can be trained with gradient descent back propagation and optimization methods [7-8].

ENN network trained using BP algorithm can be parallelized by partitioning the number of patterns, or by partitioning the network or by combination. In general, parallel processing can be achieved in two different ways – data set parallelism and algorithmic parallelism [3-5]. Algorithmic parallelism offers

two possible schemes for the MLP: (i) mapping each layer of the network to a processor and (ii) mapping a block of neurons to a processor. A special case of second scheme is to map each neuron of the network to a processor so that the parallel computer becomes a physical model of the network. The proposed parallel implementation avoids recomputation of weights and requires less communication cycle per pattern. The communication of data among the processors in the computing network is also less. We obtain the performance parameters like speed-up, optimal number of processors and processing time for both sequential implementation and parallel implementation on Master – Slave architecture. The analytical and experimental results show that the proposed parallel implementation performs better than the sequential implementation.

II. Elman Neural Network

Let us consider a fully connected ENN trained using BP algorithm. Let N_0 be the number of neurons in the input layer. Similarly N_l , $l=1,2,3$ be the number of neurons in hidden, output and context layers respectively. The different phases in the learning algorithm and corresponding processing time are discussed in the following sections [10].

A. Sequential Implementation

The BP algorithm is a supervised learning algorithm, and is used to find suitable weights, such that for a given input pattern (X_0), the network output (Y_{2i}) should match with the desired output (t_i). The details of algorithm and time taken to process are discussed below.

1. Elman Sequential Algorithm

Step 0 : Initialize weights (set to small random values)

Step 1 : While stopping condition is false do steps 2-9

Is_First_Time = TRUE.

First iteration is a special case.

Step 2 : For each training pair do steps 3-8

Feed forward

Step 3 : Each input unit $x_k = y_k^0$ ($k = 1, \dots, N_0$) receives input signal x_k

and broadcasts this signal to all units in the layer above

(the hidden units)

Step 4: Each hidden unit y_j^1 ($j = 1, \dots, N_1$) sums its weighted input

signals and context signals. Then

applies the activation function to

compute its output signal.

If (Is_First_Time) then

$$y_j^1 = f(\sum_{k=0}^{N_0} w_{jk}^1 y_k^0)$$

else

$$y_j^1 = f(\sum_{k=0}^{N_0} w_{jk}^1 y_k^0 + \sum_{h=1}^{N_1} w_{jh}^3 y_h^3)$$

and send this signal to all units in the layer above.

Step 5: Each output unit y_i^2 ($i = 1, 2, \dots, N_2$) sums its weighted input signals and applies the activation function to compute its output signal.

$$y_i^2(t) = f(\sum_{j=0}^{N_1} w_{ij}^2 y_j^1)$$

$i = 1, 2, \dots, N_2$; $y_0^1 = 1$ and

w_{i0}^2 is the bias.

Back propagation

Step 6: Each output unit y_i^2 ($i = 1, 2, \dots, N_2$) receives a target pattern corresponding to the input training pattern and computes error e_i and error information term δ_i^2

$$e_i = (y_i^2 - t_i) \quad i = 1, 2, \dots, N_2$$

$$\text{and } \delta_i^2 = e_i f'(y_i^2) \quad i = 1, 2, \dots, N_2$$

calculates its weight correction terms

$$\Delta w_{ij}^2 = \eta \delta_i^2 y_j^1 \quad i = 1, 2, \dots, N_2$$

and sends δ_i^2 $i = 1, 2, \dots, N_2$ to the units in the layer below.

Step 7: Each hidden unit y_j^1 ($j = 1, 2, \dots, N_1$) sums its delta inputs from units in the layer above to find corresponding error information term

$$\delta_j^1 = f'(y_j^1) \sum_{i=1}^{N_2} w_{ij}^2 \delta_i^2 \quad j = 1, 2, \dots, N_1$$

calculates its weight correction terms

$$\Delta w_{jk}^1 = \eta \delta_j^1 y_k^0 \quad j = 1, 2, \dots, N_1$$

where $j = 1, 2, \dots, N_1$ and

$$k = 1, 2, \dots, N_0$$

$$\Delta w_{jh}^3 = \eta \delta_j^1 y_h^3 \quad \text{where } y_h^3 = y_h^1(t-1) \quad \text{and } j, h = 1, 2, \dots, N_1$$

Step 8: Each output unit y_i^2 ($i = 1, 2, \dots, N_2$) updates its weights $w_{ij}^2 = w_{ij}^2 + \Delta w_{ij}^2$ $j = 1, 2, \dots, N_1$

Similarly each hidden unit y_j^1 ($j = 1, 2, \dots, N_1$) updates its weights related to both input layer and

context layer.

$$w_{jk}^1 = w_{jk}^1 + \eta \delta_j^1 y_k^0 \quad k = 1, 2, \dots, N_0$$

$$w_{jh}^3 = w_{jh}^3 + \eta \delta_j^1 y_h^3 \quad h = 1, 2, \dots, N_1$$

Step 9: Error matrix $E = [E_1, E_2, \dots, E_p]$

$$\text{where } E_i = [e_{i1}, e_{i2}, \dots, e_{iN_2}]$$

$$\text{Calculate } E_2 = e_{ij}^2$$

If $E_2 < \text{Threshold value}$,

store w_{ij}^0 and w_{ij}^2 and stop

else go to step 2.

Let t_m , t_a , and t_{ac} be time taken for one floating point multiplication, addition, and calculation of activation value respectively. The time taken to complete the forward phase (T_1) is given by

$$T_1 = N_1 m_a (N_0 + N_1 + N_2) + t_{ac} (N_1 + N_2) + 2t_a \quad \text{where } m_a = t_m + t_{ac}$$

The time taken to complete the error back propagation phase is represented by T_2 and is calculated as

$$T_2 = N_2 (1 + N_1) m_a + N_1 N_2 t_m + (N_1 + N_2) t_{ac}$$

The time taken to update the weight matrix between the three layers is represented by T_3 and it is equal to

$$T_3 = 2N_1 (N_0 + N_1 + N_2) m_a + N_0 (N_0 + N_1 + N_2) t_m + N_1 N_2 t_m + 2t_a + 2t_{ac} (N_1 + N_2) + N_2 m_a (1 + N_1)$$

The total processing time (T_{seq}) for training a single pattern in one iteration is the sum of the time taken to process the three phases and is given as

$$T_{seq} = T_1 + T_2 + T_3 = (N_0 + N_1 + N_2) (2N_1 m_a + N_0 t_m) + N_1 N_2 t_m + 2t_a + (N_1 + N_2) t_{ac} + (N_1 + 1) m_a N_2$$

B. Parallel Implementation

In this implementation, the hidden layer is partitioned using vertical parallelism and weight connections are partitioned on the basis of synaptic level parallelism. The Master Slave architecture of the MLP network used in the proposed scheme is shown in the Fig. 1. In this architecture there is one front-end processor (FEP) and 'm' slave processors. The output layer is placed on front-end processor, hidden layer is partitioned into N_1/m neurons and the partitions are placed on slave processors. The input neurons are placed on all the slave processors.

In this architecture the communication is only between the FEP and m Slave processors. Each of the Slave processor with FEP executes three phases of BP training algorithm. Parallel execution of the three phases and the corresponding processing time for each phases are calculated.

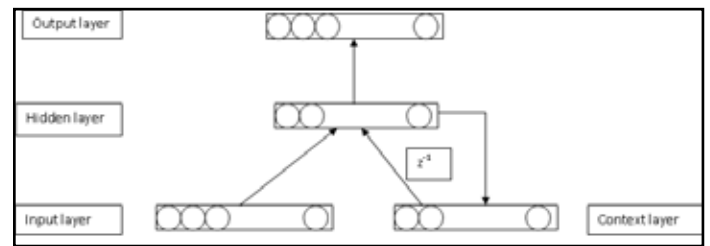


Fig. 1:

Elman Parallel with Batch updates and Adaptive learning rate: Algorithm for Front End Processor (FEP)

Step 0: Download all the input vectors onto all the processors.

$$E_{old} = \text{LARGE_VALUE.}$$

Step 1: while stopping condition is false do steps 1- 7

Step 2: For each training pair with old weights do steps 3-5

Step 3: Receive input partial sums calculated at processors.

Store the hidden layer outputs in the context layer.

Step 4: Compute the output vector O of neurons in the output layer, error E_{new} and error information terms δ_i^1 and δ_i^2

Step 5: Send this information to nodes.

Step 6: If $(E_{new}^2 < E_{old}^2)$

Send the command

“Realize weights” to nodes.

$$E_{old} = E_{new}$$

else

Send “Discard new weights”

command to nodes.

Step 7: If $(E_{new}^2 < \text{Threshold value})$

Send terminate command to

the nodes.

else

Send iterate command and go to

Step 2

Algorithm for nodes:

Step 0: Initialize the weights (set to small random values).

Receive all input vectors from FEP.

Step 1: While stop signal is not received from FEP do steps 2-9

Step 2: For each training pair do steps 3-8

Feedforward:

Step 3: Each input unit ($x_i = y_i^0, i = 1, 2, \dots, N_0$) receives input signal x_i and broad casts this signal to all units in the layer above.

Step 4: Each hidden unit ($y_j^1, j = 1, \dots, N_1$) present on this processor sums its weighted input signals from input & context layers and applies its activation function.

$$y_j^1 = f(\sum_{i=0}^{N_0} w_{ij}^1 y_i^0 + \sum_{h=1}^{N_1} w_{jh}^3 y_h^3)$$

and sends this signal to all units in the layer above.

Step 5: Each output unit y_i^2 ($i=1 \dots N_2$) sums its partial weighted input signals

$$y_{i-in}^2 = \sum_{j=1}^{N_1/m} w_{ij}^2 y_j^1$$

Where $i=1,2,\dots,N_2$ and m is the number of nodes. N_1/m neurons of the hidden layer are distributed onto m processors.

Send the partial sums to FEP

Error backpropagation:

Step 6: Receive Eold, δ_i^1 and δ_i^2 from FEP and compute Δw^1 , Δw^2 and Δw^3 .

Step 7: $w_{ij}^2 \text{ old} = w_{ij}^2$; $w_{jk}^1 \text{ old} = w_{jk}^1$;

$$w_{ij}^2 = w_{ij}^2 + \Delta w_{ij}^2(m)$$

$$i=1,2,\dots,N_2, j=1,2,\dots,N_1$$

$$w_{jk}^1 = w_{jk}^1 + \Delta w_{jk}^1(m) \quad j=1,2,\dots,N_1, \quad k=1,2,\dots,N_0$$

$$w_{jh}^3 = w_{jh}^3 + \Delta w_{jh}^3(m) \quad j=1,2,\dots,N_1, \quad h=1,2,\dots,N_1$$

Step 8: Receive commands from FEP

If (cmd= Realize weights)

$$\eta = \eta * 1.2$$

If (cmd= discard weights)

$$w_{jk}^1 = w_{jk}^1 \text{ old}; \quad w_{ij}^2 = w_{ij}^2 \text{ old};$$

$$w_{jh}^3 = w_{jh}^3 \text{ old};$$

Set η =initial value }

Step 9: If(cmd=Terminate)

Save the weights w_{old}^1 , w_{old}^2 and w_{old}^3
elseif (cmd=iterate) go to Step 2;

The approximate time taken to process the forward phase (T_1) is given by.

$$T_1 = [N_0 \frac{N_1}{m} + \frac{N_1}{m} N_2 + \frac{N_1}{m} N_1] M + \frac{N_1}{m} t_{ac} + \frac{N_2}{m} t_{ac} + \frac{N_1}{m} 2t_a + t_{com}^1$$

where $M = t_a + t_m$

The approximate time taken for processing error back-propagation phase (T_2) includes the error calculation at the FEP and at the slave processors and is

$$T_2 = \frac{N_1 N_2}{m} M + N_2 m_a + \frac{N_1 N_2}{m} t_m + t_{ac} [\frac{N_1}{m} + N_2] + t_{com}^2$$

The processing time to update the weight connections (T_3) is given by

$$T_3 = 2 \frac{N_1}{m} (N_0 + N_1 + N_2) M + 3 \frac{N_1}{m} (N_0 + N_1 + N_2) t_m + t_{com}^3$$

Let T be the time taken to train in the proposed HP scheme, and it is equal to the algebraic sum of time taken to process all the three phases.

$$T_p = [3N_0 N_1 + 4N_1 N_2 + 3N_1 N_1] [M/m + t_m / m] + [2 \frac{N_1}{m} + \frac{N_2}{m} + N_2] t_{ac} + 2N_1 t_a / m + M \log m \{2\sigma + \mu [(N_1/m) + N_2]\}$$

Here $t_{com} = \log m [T_{ini} + T_c D]$ where m is the number of processors, D is data size(number of words), T_{ini} is startup time and T_c is per word transfer time.[11]

III. Analytical Performance Comparison

A. Speed up analysis

Speed-up for m-processor system is the ratio between the time taken by uniprocessor to the time taken by parallel algorithm in m-processor network.

The speed-up ratio for parallel implementation can be formulated as

$$S = T_{seq} / T_p = (N_0 + N_1 + N_2)(2N_1 m_a + N_0 t_m) + N_1 N_2 t_m + 2t_a + (N_1 + N_2) t_{ac} + (N_1 + 1) m_a N_2 + \{ [3N_0 N_1 + 4N_1 N_2 + 3N_1 N_1] [M/m + t_m / m] + [2 \frac{N_1}{m} + \frac{N_2}{m} + N_2] t_{ac} + 2N_1 t_a / m + M \log m \{2\sigma + \mu [(N_1/m)]\} \}^{-1}$$

If the network size is extremely larger than the number of processors m , then the speed up ratio will approach m . This is due to extra computation required in weight update phase and extra communication in exchanging the hidden neurons activation values.

B. Difference Between Processing Times

From the expressions of T_{seq} and T_p the difference in processing time is calculated as follows.

$$T_{seq} - T_p = (N_0 + N_1 + N_2)(2N_1 m_a + N_0 t_m) + N_1 N_2 t_m + 2t_a + (N_1 + N_2) t_{ac} + (N_1 + 1) m_a N_2 - \{ [3N_0 N_1 + 4N_1 N_2 + 3N_1 N_1] [m_a/m + t_m / m] + [2 \frac{N_1}{m} + \frac{N_2}{m} + N_2] t_{ac} + 2N_1 t_a / m + m_a \log m \{2\sigma + \mu [(N_1/m) + N_2]\} \}$$

Assume that $N_1 = N_2 = N_3 = m = N$. Then

$$T_{seq} - T_p \approx [6N^3 - 4N^2 - 5N - 2 - \log N \{2\sigma + \mu [(1+N)]\}] > 0$$

From the above equation it is clear that the time taken by parallel implementation is less than the time taken by the sequential algorithm.

IV. Experimental Study and Results

In this paper, we implemented a parallel BP algorithm on a cluster of computers connected over Ethernet LAN to recognize the vowel dataset presented. The neural network is mapped onto Master-Slave architecture. The analytical performance of the proposed algorithm is compared with the sequential counterpart and results Are shown in fig. 2 & Table 1. Using the hybrid-partitioning scheme, recomputation of weights is avoided and the communication time is reduced. As one hidden layer is adequate for a large number of applications, in the present project work the algorithm is developed for neural nets with one hidden layer.

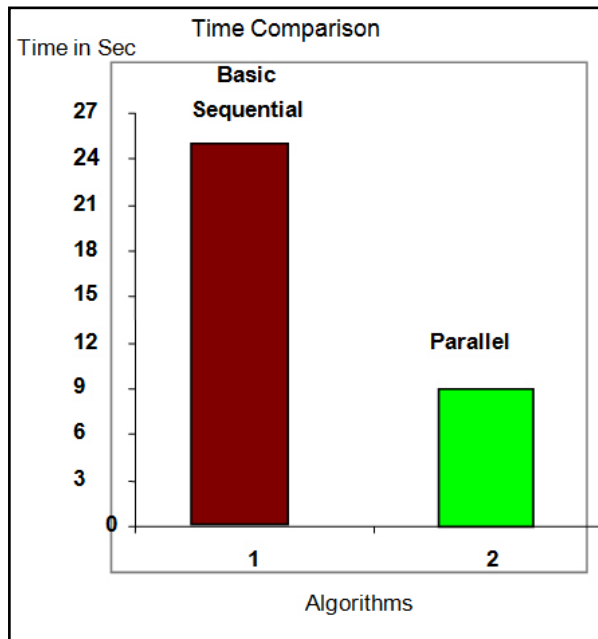


Fig. 2: Learning Rate: 0.1

Comparison of modified Serial and Parallel ENN

Table 1:

Algorithm	Metric	Value(Sec)
Serial	Time	25
Parallel algorithm with batch update weights	Time	9

V. Conclusions

In this paper, an implementation of distributed BP algorithm on Elman network on a cluster of computers connected over Ethernet LAN to classify the given image into one of the five classes is presented. Hybrid partitioning technique is proposed to partition the network. The partitioned network is mapped onto Master-Slave architecture. In fig. 2 & 3, we have shown the time comparison to train the ENN with both sequential and parallel algorithms. The analytical performance of the proposed algorithm is compared with the vertical partitioning. Using the hybrid-partitioning scheme, recomputation of weights is avoided and the communication time is reduced.

References

- [1] Laurence Fausett, "Fundamentals of neural networks; Architectures", algorithms and applications, PEA, 2005
- [2] Annanth Grama, Anshul Gupta, George Karypis, Vipin Kumar, "Introduction to parallel computing", PEA, 2004.
- [3] N. Sundararajan, P. Saratchandran, "Parallel Architecture for Artificial Neural Networks", IEEE CS Press, 1998.
- [4] Fedorova N.N, Terkhoff, "Parallel MPI implementation of Training algorithms for mid size feed forward neural networks", IJCNN, 1999.
- [5] AStrey, "Acomparision of OpenMP and MPI for neural network simulations on a Sunfire 6800", Parallel computing: Software Technology, Algorithms, Architectures & Applications, Elsevier 2004.

- [6] Tarik Rashid, "Shape Recognition through an Alternative Recurrent Network Architecture", Journal of Computer Science, Informatics & Electrical Engineering.
- [7] Qing Song, "On the Weight Convergence of Elman Networks", IEEE Transactions on Neural networks, Vol. 21, No. 3, March 2010.
- [8] Elif Derya Übeyli, Mustafa Übeyli, "Case Studies for Applications of Elman Recurrent Neural Networks", Recurrent Neural Networks, Book edited by: Xiaolin Hu and P. Balasubramaniam, pp. 400, September 2008, I-Tech, Vienna, Austria.
- [9] David Samek, "Elman neural networks in model predictive control", Proceedings 23rd European Conference on Modelling and Simulation ©ECMS Javier Otamendi, Andrzej Bargiela, José Luis Montes, Luis Miguel Doncel Pedrera (Editors).
- [10] William H. Wilson, "Stability of Learning in Classes of Recurrent and Feedforward Networks", Proceedings of the Sixth Australian Conference on Neural Networks (ACNN'95) 142-145.



Prof. JVS Srinivas received his M.Sc(Applied Mathematics) from University of Hyderabad, India, ALCCS from IETE, India. He is currently professor in CSE Department, DVR College of Engineering & Technology, Hyderabad and pursuing part time Ph. D program at OU College of Engineering, Hyderabad.



K. Priyadarsini received her B. Tech in CSE from Kakatiya University College in 2006 and currently pursuing M.Tech at DRK IST, Hyderabad.



Prof. P. Premchand received his B.Sc(Electrical Engineering) from R.I.T., Jamshedpur, India M.E(Computer Engineering) from Andhra University and doctorate Computer Science & Systems Engineering from Andhra University, Visakhapatnam, India. He is currently professor in CSE Department, OU College of Engineering, Hyderabad.