Parallel Processing in Neural Network for Pattern Recognition

Chairisni Lubis[1], Lely Hiryanto[2], Kurniawan Sulianto[3]
[1]*Faculty of Information Technology, Tarumanagara University, Jakarta, Indonesia*
chairisni@yahoo.com
[2]*Faculty of Information Technology, Tarumanagara University, Jakarta, Indonesia*
lely@fti.utara.org
[3]*Faculty of Information Technology, Tarumanagara University, Jakarta, Indonesia*
kslian86@gmail.com

Abstract

Perceptron is one of the fundamental algorithms in neural network for pattern recognition The common problem faced in recognizing pattern using neural network is in the learning computation which is time consuming. To reduce the learning time, we develop a parallel algorithm for the perceptron learning process. To support the parallel processing on the cluster, we use MPICH for Windows as message passing tool and Visual C++ 6.0 as the editor and compiler of the parallel pattern recognition program. Our experimental result of running the program in the cluster shows that the parallel process gives speedup 3 times of its sequential process.

1. Introduction

Perceptron is one of the fundamental learning algorithms in neural network. The perceptron architecture is a single layer and its learning process is perfomed by finding connection weight (w) of each pattern target. If an error occurred for a particular traning pattern the weight would be changed according to the formula

$$w_{ij}(new) = w_{ij}(old) + \alpha t_j x_i \qquad (1)$$

where :

w_{ij} (new) : new connection weight
w_{ij} (old) : old connection weight
α : learning rate
t_j : pattern target
x_i : pattern input

If an error did not occur, the weights would not be changed. Learning process is done when connection weights are correct for each pattern target.

The training algorithm for several output categories with $\alpha = 1$ are [1]

```
Step 0. Initialize weights and biases
        (0 or small random values).
Step 1. While stopping condition is false, do Steps 2-6.
Step 2. For each bipolar training pairs s:t, do Steps 3-5.
Step 3. Set activation of each input unit,
            $i = 1,...,n;\ x_i = s_i$
Step 4. Compute activation of each output unit,
            $j = 1,...m;\ y\_in_j = b_j + \sum_i x_i w_{ij}$
            $y_j = \begin{cases} 1 & if\ y\_in_j > \theta \\ 0 & if\ -\theta \le y\_in_j \le \theta \\ -1 & if\ y\_in_j < -\theta \end{cases}$
Step 5. Update biases and weight,
            $j = 1,...m;\ i = 1,...,n;$
        If $t_j \ne y_j$, then
            $b_j(new) = b_j(old) + t_j$
            $w_{ij}(new) = w_{ij}(old) + t_j x_i$
        Else,biases and weights remain unchanged.
Step 6. Test for stopping condition :
        If no weight changes occurred in Step 2, stop;
        otherwise, continue.
```

Fig 1. Perceptron learning algorithm

The common problem faced in recognizing pattern using perceptron is in the learning computation which is time consuming. To reduce the learning time, we develop a parallel algorithm for the perceptron learning process.

2. Perceptron in parallel processing

From learning algorithm perceptron in fig. 1 we seen that the longest computation time in learning process for computing y_in. Computation proses for n inputs patterns with m outputs patterns give m linear equations,

$$y\_in_1 = b_1 + x_1 w_{11} + x_2 w_{21} + x_3 w_{31} + ... + x_n w_{n1}$$
$$y\_in_2 = b_2 + x_1 w_{12} + x_2 w_{22} + x_3 w_{32} + ... + x_n w_{n2}$$
$$y\_in_3 = b_3 + x_1 w_{13} + x_2 w_{23} + x_3 w_{33} + ... + x_n w_{n3}$$
.
.
.
$$y\_in_m = b_m + x_1 w_{1m} + x_2 w_{2m} + x_3 w_{3m} + ... + x_n w_{nm}$$

When connection weights are correct for one pattern target, the learning process still compute that connection weight in the next iteration. We try to reduce the wasted time in learning process by modifying the algorithm. We use two iteration. If we found the correct connection weights for one pattern target then the first iteration is stoped. The second iteration continue the learning process with another pattern target. The iteration will stop when connection weights are correct for each pattern target. Perceptron learning algorithm modification is as follows

```
Step 0.  Initialize weights and biases
         (0 or small random values).
Step 1.  While stopping condition is false, do Steps 2-7.
Step 2.  For $j = 1,...,m$, do Steps 3-6.
Step 3.  Set activation of each input unit and output unit j,
             $i = 1,...,n;\quad x_i = s_i$
Step 4.  Compute activation of each output unit,
             $y\_in_j = b_j + \sum_i x_i w_{ij}$
             $y_j = \begin{cases} 1 & if \quad y\_in_j > \theta \\ 0 & if \quad -\theta \le y\_in_j \le \theta \\ -1 & if \quad y\_in_j < -\theta \end{cases}$
Step 5.  Update biases and weight,
             $j = 1,...m;\; i = 1,...,n;$
         If $t_j \ne y_j$, then
             $b_j(new) = b_j(old) + t_j$
             $w_{ij}(new) = w_{ij}(old) + t_j x_i$
         Else,biases and weights remain unchanged.
Step 6.  If no weight changes occurred in Step 2,
         stop; otherwise, continue.
Step 7.  Test for stopping condition :
         If no weight changes occurred in Step 2,
         stop; otherwise, continue.
```

Fig 2. Perceptron learning algorithm modification

From m linear equations on above, there are no dependencies for every weight and bias from one output to another. Every equation could be done without any result from computation of another output. So, every equation could be done with parallel processing. With parallel processing, every computer will compute weight and bias of every target on j.

3. Experiment

The sample data that we use for pattern recognition in experiment are :

- Data pattern : numerical character (0, 1, 2, …9) of abstract font.

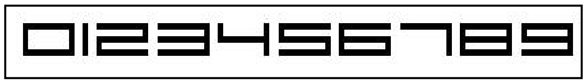

Fig 3. Numerical character of Abstract font
Reference: Dafont.com, Abstract, http://www.dafont.com/abstract.font?text=0123456789 , August 28, 2008.

- The sample character size : 25 pixel x 35 pixel
- Number of sample : 100
- Every character consist of :
  - 1 normal pattern
  - 4 dirty pattern (changes randomly several white colour pixel with black pixel as noises)
  - 5 impaired pattern(changes randomly several black colour pixel with white pixel as noises)
- Noise : 1%, 5%, 10%, 15% and 20%

Fig. 4 was illustrated sequential processing that we use for computing execution time in sequensial process. One node in input layer represent one pixel of input data and nodes in output layer represent output data (character). So , we have 975 node in input layer and 10 node in otput layer. Learning rate (α) for parallel processing determined from lowest computation time of sequential learning processing with α = 0.1 to α = 1.0.

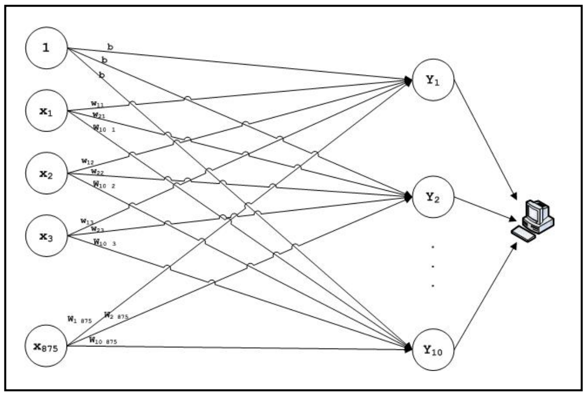

Fig 4. Sequential processing illustration

The number of computers could be used for parallel processing are 2 to 10 computers. This is llustration for parallel processing (fig. 5) and Clustering on parallel processing adopting Beowulf cluster (fig. 6).

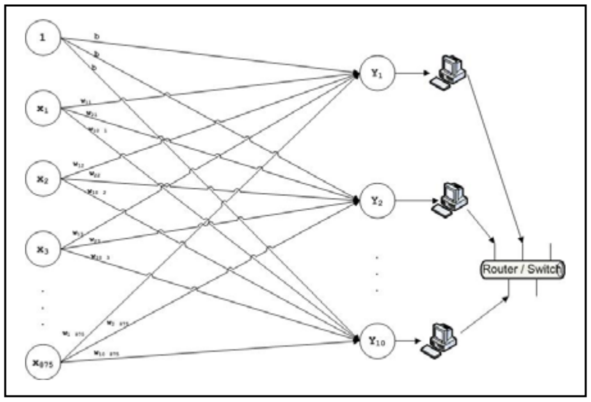

Fig 5. Parallel processing illustration

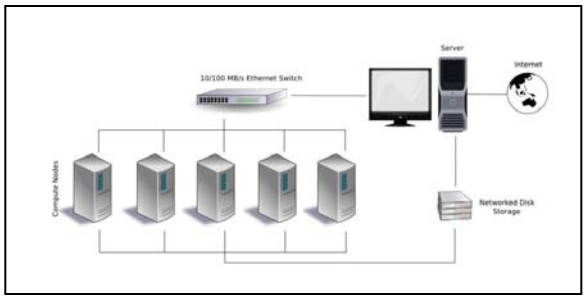

Fig 6. Beowulf Cluster
Reference : Wikipedia, Beowulf (computing), http://en.wikipedia.org/wiki/Image:Beowulf.png, August 23, 2008.

Some software that used for making this application are:

- Operation System for Server: Microsoft Windows Server 2003
- Operation System for Client: Microsoft Windows XP Professional
- Paraller Software : MPICH
- Programming Language : C and VB.Net
- Image Editing : Microsoft Paint

and configuration of hardware and software for experiment are :

Software (testing):

- Operation System for Server: Microsoft Windows Server 2003
- Operation System for Client: Microsoft Windows XP Professional
- Parallel Software : MPICH
- Framework : .Net Framework 2.0 (on Server)

Hardware (testing):

- Processor : Intel® Pentium®4 2.66 GHz
- RAM : 1 GB
- NIC : Broad Com Nextreme Giga bit Ethernet for HP
- Harddisk : 40 GB

Experiment result for sequential processing and parallel processing shown on table 1, 2a, and 2b.

Table 1
Execution time in sequential processing

Noise (%)	α	t (second)
1	0.2	0.1209
5	1.0	0.1604
10	0.2	0.2430
15	0.7	0.2344
20	0.5	0.2722

Note ;
α : learning rate
t : execution time

Table 2a.
Parallel processing for noise 1%, 5%, 10%

n	t (1%) α=0.2	t(5%) α=1.0	t(10%) α=0.2
2	0.0690	0.0934	0.1292
3	0.0569	0.0703	0.1262
4	0.0553	0.0741	0.1013
5	0.0337	0.0502	0.0918
6	0.0303	0.0545	0.1005

7	0.0342	0.0613	0.0876
8	0.0303	0.0458	0.0828
9	0.0296	0.0484	0.0858
10	0.0284	0.0431	0.0803

Table 2b
Parallel processing for noise 15 % and 20%

n	t(15%) α=0.7	t(20%) α=0.5
2	0.1217	0.1325
3	0.1102	0.1293
4	0.0945	0.1056
5	0.0990	0.1159
6	0.0964	0.1009
7	0.0846	0.0954
8	0.0899	0.1007
9	0.0824	0.0930
10	0.0823	0.0928

Note :
n : number of computer

From table 1, 2a and 2b, we can calculate the speed up time with:

$$S(p) = \frac{t_s}{t_p} \tag{2}$$

where ;
S(p) : Speed Up
t_s : execution time in sequential process
t_p : execution time in parallel process

Table 3
Speed Up in parallel process

n	Speed Up
2	1.8660
3	2.1132
4	2.3620
5	2.8299
6	2.8956
7	2.9102
8	3.1486
9	3.2000
10	3.3579

4. Conclusion

1. The learning time in perceptron depend on percentage of noise of every pattern. The higher noise take more computation time than lowest noise.
2. Parallel process that using 10 computers gives speedup up to 3 times of its sequential process.

5. References

[1] Fauset, Laurene. Fundamentals of Neural Networks: Architectures, Algorithms, and Applications. Englewood Cliffs: Prentice Hall, 1994.

[2] Wilkinson, Barry and Allen, Michael. Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers Second Edition. Upper Saddle River: Prentice Hall, 2005.