A Kind of Parallel Evolutionary Algorithms and Its Application in E-Business

Yan Zhu Jian Chen
School of Economics and Management
Tsinghua University
Beijing 100084, China
zhuyan@em.tsinghua.edu.cn, chenj@em.tsinghua.edu.cn

Abstract

Parallel Genetic Algorithms (PGAs) were used to simultaneously optimize the structure and weights for feedforward neural networks. Aiming at its large-scale application in common distributed network system, a concentrative coarse-grained model for parallel evolutionary neural networks is designed and realized in a laboratorial distributed computation environment, and the initial results of experiments indicate that the parallel model can quicken the searching process and improve the evolutionary efficiency. For the parallel characteristics, this method can analyze massive information during the process of e-business. Several application scopes of this method in e-business are discussed also in this paper .

Key words Parallel Evolutionary Neural Networks; Distributed computation; E-business

1. Introduction

The application of Artificial Neural Networks (ANNs) has been extended to many areas in recent years. Common practices are usually involved with using back propagation (BP) algorithms on structure-predefined full-connected feed-forward neural networks for connection weights adjustment. Unfortunately such training is often time-consuming, especially for large-scale problem, and user-dependent for ANNs' architecture definition, which is crucial to the training process, while still a difficult decision, without optimum rules to follow.

Evolutionary Neural Networks (ENNs) provide a prominent alternative neural-network training method. They take advantage of evolutionary algorithms' (EAs) global searching ability, not only automatically optimize ANNs' architecture according to particular tasks, but also dynamically improve ANNs' generalization performance. However, when encountered with large-scale problem, ENNs also suffer from considerable computation expense due to complex network architecture.

Parallel Evolutionary Neural Networks (PENNs) have the potential to find fittest ANNs in significantly less time than serial ENNs. Aiming at PENNs' extensive application in common distributed computer network, we have been devoted to the analysis, design and implementation of parallel algorithms in distributed computation environments, compared to numerous studies in multi-processor workstations.

In this paper, we introduce a kind of parallel model specially designed for PENN in distributed system, on the basis of which a parallel evolutionary algorithm is realized to optimize the architecture and weights of feed-forward neural networks simultaneously.

As we know, the most popular evolutionary algorithm used in ENNs is Genetic Algorithms (GAs). There are generally three kinds of evolution in GA-based ENN, i.e., the evolution of connection weights, of architectures and of learning rules, according to the level to which evolutionary search procedures come into ANNs[1]. If the learning rules are pre-defined and fixed, the main procedures of the two kinds of evolution are so similar that we can combine them into one evolutionary algorithm, which will be described later.

A successful design of ENN requires careful consideration about several crucial aspects[2]:

1) Encoding Scheme

How to map the ENNs' structure (connection weights and architecture) into one-dimension chromosomes? Using binary strings or real numbers to represent the connection weights? How to arrange the sequence of connection weights to reflect the network architecture?

2) Fitness Function

How to construct the fitness function? Need additional items besides training error? If necessary, how to decide their weights?

3) Genetic Operators

Which types of genetic operators are involved in the ENN? What specific operations do they execute respectively?

All these designing details have great influence on evolutionary process and final result, and need paying enough attention to. When discussing about our PENN later, we will describe these aspects thoroughly.

The parallelization of ENN is based on the parallelism of GA, and has been performed in four following ways:

1) The parallelization of single individual fitness evaluation

The work of fitness evaluation is most time-consuming in ENN, which is involved with decoding, training iteration and encoding. Using parallel computation technology on single individual fitness

evaluation will hopefully improve evaluation efficiency.

2) The parallelization of individual fitness evaluation in population

In a population, different individual's fitness evaluation is independent and can be performed on different processors simultaneously and independently.

3) The parallelization of individual-generating

The genetic operations involved in child-generation, i.e., reproduce, crossover, mutation etc, can be implemented on different individuals simultaneously and independently.

4) The parallelization based on population grouping

The evolutionary process of one or a group of individuals in the population is independent of each other, so this provides another way of parallel ENN based on population grouping: divide the population into several groups, assign each group a processor, evolve these population groups simultaneously, and exchange some individual information among groups at proper times to speed up global evolution progress.

The former three parallel methods haven't change ENN's overall frame, while the fourth, by breaking the population into groups and exchanging information periodically, has potential influence on ENN's general behavior and is most popular due to its simple realization in multiple-processor machine and distributed computer network system.

2. PENN based on CCG parallel model in distributed environments

2.1 CCG parallel model in distributed computation environment

Although distributed computer network is more convenient, abundant and economical parallel environment compared to multiple-processor machine, the much lower coupling degree make it much difficult to coordinate and communicate among different processors, which needs global schedule and control by software programming. Meanwhile, the communication efficiency in distributed environment is subject to many external impacts, such as connecting material, flow rate, flow amount etc., which is much less steady and reliable than multi-processor machine. Designing and realizing parallel models for distributed system must take these differences under careful consideration, and thus more difficult to deal with, which may be the reason why there is so little related work now.

The centralized coarse-grained model we present here is spirited by Cantu-paz E[3] about the master-slave parallel algorithm(We still use the master-slave notation in our model)and other previous work on coarse-grained parallel model. Our parallel model takes advantage of both master-salve and coarse-grained parallel model and we consider it fit well for the distributed system. Taking three distributed paralleling processors for example, the CCG model is illustrated in Figure1.

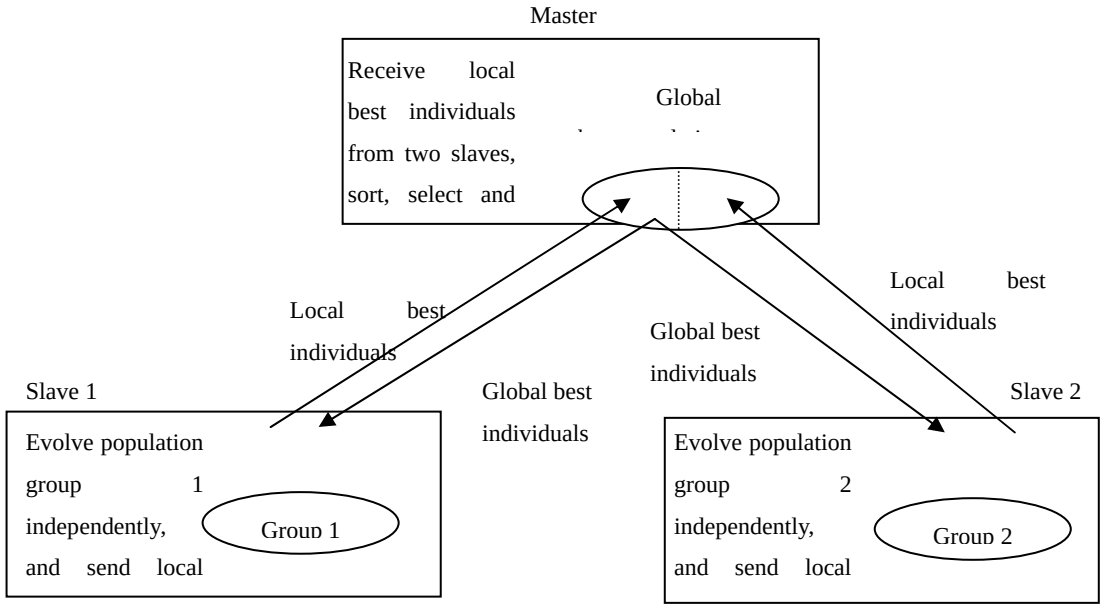

Figure 1 CCG model for three distributed paralleling processors

2.1.1 CCG model

According to different functions, divide distributed paralleling processors into one master and several slaves. On the other hand, following the idea of parallelization based on population grouping, divide the evolution population into several groups, one group for one slave. Each slave evolves its population group independently, and sends the master its local best individuals after a pre-specified number of generations. After the master receives all slaves' local best individuals, it incorporate them into one group, which is called global best population group, sorts the group according to individuals' fitness, selects global best individuals and broadcasts them back to slaves. Slaves keep waiting until they receive the feedback from master and continue their evolution process before next transfer.

In the model, the evolution processes on different slaves are paralleling, while the works on master and slaves are actually been carried on sequentially.

1) Function specification

The main responsibility of master is receiving local best individuals from all slaves, sorting them by fitness, selecting the global best ones and broadcasts them back to slaves, also keeping record of the best individual and storing global best population group. In contrast, the slaves are in charge of ENN on its own population group, sending local best, receiving global best, replacing local worst.

2) Communication and information exchange

In CCG model, inter-processor communication only takes place between master and slaves. This reflects the design intention for realization simplicity. When one slave finishes its current evolution period, it sends the master a signal as transfer request, begins data transferring if master isn't busy(if busy, blocks and waits for master's ready message), and then wait until receiving global best individuals from master.

For the master, it keeps sleeping until being woken by a slave's transfer request, and begins the uninterruptible data transferring(if another slave ask for service during this time, send a busy signal and put it to the end of the waiting line). After the transfer is finished, it arouses the first slave in the waiting line by sending a ready message and starts the next data transferring until it gets all slaves' local best.

3) CCG versus traditional parallel models

The CCG model has inherited the population-grouping and information-exchanging ideas from traditional coarse-grained model, while introduces centralizing control to overcome the unstableness in distributed system. On the other hand, CCG is quite similar to traditional centralizing parallel model at the first glimpse, while compared to global evolution and selection in centralizing model, CCG performs local evolution and selection, which is the most distinct advantage of coarse-grained towards centralizing.

2.1.2 Key parameters

There are several key paralleling parameters in the design and realization of CCG, whose different values have great influence on CCG model's behavior and characteristics.

2.1.2.1 Transferring Scale

The transferring scale of inter-processor individual exchanging is determined by two parameters: transferring rate and transferring period. Large-scale transferring will make CCG acting like global centralizing model, while small scale will enforce the independence of each ENN.

(1) Transferring rate

Transferring rate is the amount of the individuals transferred, which is often defined by absolute number or percent of the population group. A typical transferring rate is ten to twenty percent of the population group.

(2) Transferring period

Transferring period determines the time interval between two transfers, which is usually one or several evolutionary generations on slaves. Generally, higher transferring rate is companied by longer transferring period.

2.1.2.2 Transferring Strategy

Transferring strategy includes two aspects, transferring selection and transferring replacement. Other than the strategy we mentioned in CCG description, there are still different choices.

(1) Transferring selection

Transferring selection is responsible for selecting individuals transferred. Common method is selecting one or several best individuals, while selection based on fitness proportion is also optional.

(2) Transferring replacement

In most cases, one or several worst individuals are replaced. While similar with transferring selection, the replacement can also be based on fitness-proportion selection, which can bring some selection pressure on better individuals in the population group.

2.2 Parallel Evolutionary Neural Networks

Just as we have mentioned, our PENN aims at evolving neural networks' architecture and connection weights simultaneously. Focusing on the three fundamental problems in ENN designing, below we will present the details for our evolutionary algorithm, which mainly deals with feedforward neural networks with linear threshold unit and sigmoid transfer function.

2.2.1 Encoding Scheme

We use a Java class to describe the neural networks, which is the encoding representation of the individual.

In the Java class, the first parameter represents the number of neurons in the hidden layer, and then we use an array to store all the weights and connection

information of the network. The final two parameters denote the individual's fitness and relative fitness.

In order to encode the weights and connections exactly and convenient for genetic operations, we adopt real number representation instead of binary code, which also makes the length of gene flexible and shorter. Using real number representation, we are able to take following additional advantages:

- Crossover can be made only between weights;
- Mutation is just adding a random number to the weight existing, instead of replacing it;
- Smaller population size because of the gene-length flexibility.

2.2.2 Fitness Function

Assume that use a training-set on individual i and get the total network error $E(net_i)$, which is the sum of the errors between the object output vectors and the simulative output vectors. Obviously, the higher the total error is, the lower the individual's fitness.

In addition to the error, we expect the structure of the neural network is as simple as possible, given the same network performance. The simpleness means the optimized architecture should have minimum number of neurons and connections. In order to control the complexity of the neural network, one control item is added to the error function: CH. C is a positive control constant, which we call the complexity coefficients of the network; H is the number of neuron of the hidden layer. Certainly, we expect a CH value as low as possible, given a pre-specified C value.

As a conclusion, we define our fitness function as this:

$$pop[i].fitness = f(net_i, t) = \frac{1}{1 + E(net_i) + CH} \tag{1}$$

The value of coefficient C is suggested to be in interval (0,0.5).

2.2.3 Genetic Operators

The usual genetic operators include selection, breeding, crossover and mutation, which are separately implemented to a certain individual in pre-determined probabilities.

Selection strategy: the evolutionary algorithm adopts standard roulette selection operator. The selection probability of individual net_i, according to the roulette selection strategy, is:

$$p_i = pop[i].relativefit = \frac{pop[i].fitness}{\sum_{j=1}^{N} pop[j].fitness} \tag{2}$$

N is the size of the population group.

Breeding operator: select parents from the population according to the selection strategy and the individual selection probability (its relative fitness), and then copy these parents to current population without any change.

Crossover operator: the same number of weights is selected from the two parents stochastically and independently. Exchange the weights, and thus get two offsprings.

Mutation operator: as we are evolving neural network's architecture and connection weights simultaneously, the mutation operators should include architecture mutation and connection-weights mutation correspondingly.

1) Architecture mutation
   Four architecture mutation operations are carried out in certain probabilities, that is:
   - Delete some neurons in hidden layer and (or) the connections and their weights. When only delete the connections, we set the corresponding weights zero.
   - Insert some neurons into each hidden layer and (or) the connections and their weights, and generate the corresponding weights randomly.
   - For those deleted connections, repair them with special probability.
   - Mutate the connection weights with adaptive mutation rate.
2) Connection-weights mutation

The mutation operation we present here is based on adaptive mutation rate, satisfying the evolutionary requirement that when the individual has lower fitness value, the mutation should be higher. So we introduce the simulated annealing algorithms, in which the mutation temperature can be defined as:

$$T = 1 - f(net_i) \tag{3}$$

Assuming the weight v will be mutated, the definition scope is (a,b), then

$$v' = \begin{cases} v + \Delta(T, b - v), & rnd(2) = 0 \\ v - \Delta(T, v - a), & rnd(2) = 1 \end{cases} \tag{4}$$

Here the value field of $\Delta(T, y)$ is (0,y), and the probability that $\Delta(T, y)$ gets close to zero increases when T reduces.

$$\Delta(T, y) = y(1 - r^{T^{\lambda}}) \tag{5}$$

Here r is a random real number from 0 to 1, λ is a parameter that determine the adaptive degree, the value is from 2 to 5 commonly.

3. Results and Discussion

PENN based on CCG is realized in a one-master-two-slave scope. The parallel experimental platform is constructed in a three-PC distributed LAN, using Java programming and data-package transferring

technology, which is convenient to build and easy to expand to larger-scale application.

The values of some key experimental parameters are listed in Table 1.

Table 1 Values of key experimental parameters

Population Size (Slave group)	100
Transferring Rate	5%
Transferring Period	10
EA Mutation Rate	0.001
EA Crossover Rate	0.2
Network complexity coefficient C	0.1
Adaptive Mutation parameter λ	2
Transferring Selection Strategy	Best
Transferring Replacement Strategy	Worst

3.1 Experimental Results

The initial experiment results demonstrate that the paralleling evolutionary algorithm based on CCG has remarkably quickened the search process and improve the solution quality at some extent. A typical experimental result is shown in Table 2, which is the comparison between ENN and PENN on XOR Problem.

Table 2 Comparison results of ENN and PENN

	Number of Generations	Searching Time (ms)	Best Individual Fitness	Population Error
ENN	200	883090	0.707	0.481
PENN	100 for each slave	471480	0.751	0.068

The improvement of solution quality illustrates that the PENN based on CCG model is not the simple sum-up of two independent paralleling processes, but has changed the fundamental behavior of ENN by best individual transferring and exchanging. That is similar to different offspring quality in marriages between close relatives or far relatives, the latter of which is often better because of gene mutation and exchange.

3.2 Efficiency Analysis

In order to measure the paralleling efficiency, we also examined the relative loading degree of the three processors and total communication expense. A typical result is shown in Table 3.

Table 3 Run-Time Data of PENN

	Run Time	Block Time	Package Time*	Compu. Time	Com. rate**
Master	341140	267970	15250	37930	88.9%
Slave 1	341140	25609	15630	299901	12.1%
Slave 2	341140	23555	15720	301865	11.5%

**Including data packing and unpacking time, and time unit is ms*

***Communication Expense rate = (Block time + Data package time) / Run time*

From the table, we can find that much of master's run time is spent on blocking, that is, waiting for salves' evolutionary process. It is partly due to CCG's centralizing structure and is a big waste of master processor's resource. A further improvement on this problem is adding an evolutionary process to Master processor, which is to say, having the master assuming the responsibilities of both a master and a slave. It will hopefully reduce the master's idling time and increase global paralleling efficiency.

4. PENN in e-business

In the information era, more and more data are stored in enterprise database systems. It is still a big problem for enterprises to utilize these data well, especially for decision. The barriers are not the amount of data, but how to cope with mass data, that is, the efficiency of analyzing data is the bottleneck.

PENN provides an efficiency way for enterprise data analyzing, especially in the distributed environment. It can be used as data mining tools to classify or recognize the special pattern hidden in the data.

As a classifying tool, PENN can process the data in a distributed way, such as the customers' data in a commercial bank; it can be used to find the most promising customers. It also can be used as predicting tool, and time serial analyzing tool.

At present, a new method for portfolio selection based on PENN is studied, and this work will optimize the portfolio in a distributed environment. A lot of this kind of works can be done, if we find a way to process data efficiently and effectively. With the rapid development of e-business, more and more data crowd in Internet. PENN can be used to treat this data traffic problem also, and we will improve PENN methods to fill this demand in the future.

5. Conclusion

In this paper, we introduce a kind of paralleling model CCG specially designed for distributed system, which takes the advantage of both classical centralizing and coarse-grained paralleling model and demonstrated to fit for parallel algorithms in distributed computation environments.

On the basis of CCG, a parallel evolutionary algorithm is designed and realized to optimize the architecture and weights of feed-forward neural networks simultaneously. It adopts the idea of simulated annealing and has been demonstrated better performance than sequential algorithms both in searching time and solution quality.

Further work is expected on the convergence and behavior analysis of PENN. For its practical application, the parameters also need further careful analysis and adjustment according to different tasks.

Acknowledgement

This work is partly supported by National Science Foundation of China (grant No. 70101008).

Reference

[1] X. Yao, “A review of Evolutionary Artificial Neural Networks,” International Journal of Intelligent Systems, Vol.8, No.4, 1993, pp.539-567

[2] V. Maniezzo, “Genetic Evolution of the Topology and Weight Distribution of Neural Networks,” IEEE Trans. On Neural Networks, Vol.5, No.1, 1994, pp.39-53

[3] Cantu-paz E, ”Designing efficient master-slave parallel genetic algorithms”, ITR 97004. Urbana, IL: University of Illinois Urbana- Champaign, 1997

[4] Y. Zhu, J. Chen, “Studies on genetic multiplayer feedforward neural networks and the development of GMNN. IEEE International Conference on Systems, Man, and Cybernetics [C]. Tokyo: SMC'99 Publication Committee, 1999

[5] X,T. Guo, Y. Zhu, “Evolutionary Neural Networks based on Genetic Algorithms”, Tsinghua Science and Technology, Vol.40, No.10, 2000, pp.116-119