Function Approximation Using the Cooperative PSO Neural Network

Meiping Song Guochang Gu Xingce Wang Rubo Zhang

Computer Science and Technology, Harbin Engineering University, 150001
smping@163.com

Abstract

Among the methods of training weights of neural network, BP is the most popular for its reasonable basis. But there still are some unsolved problems, such as the training result being influenced greatly by the order of samples, local optimization and the slow learning speed etc. PSO (Particle Swarm Optimizer) is often used to replace the BP method to speed up the learning and reduce the probability of falling into local optima. According to the topologic structure of network, a pattern of cooperation called FLsplit is prompted, which will be convenient for parallel computation. And a particular individual named BParticle is involved in the swarm, to stabilize the learning result and decrease the computation cost. The performances of BP, standard PSO and FLsplit are analyzed through the experiments on two functions, and the efficiency of FLsplit is addressed.

Keywords: Neural Network, function approximation, cooperative PSO

1. Introduction

BP is one of the most popular training methods for multi-layer neural network. But there still are some unsolved problems, such as the training result being influenced greatly by the order of samples, local optimization and the slow learning speed etc.

PSO is an evolutionary method prompted in 1995 and has been developed rapidly. Its advantages are parallelizability and capability of searching globally. Once prompted, PSO was believed to be an alternative for BP, and there have been several papers addressing its perspective.

F. van den Bergh presented cooperative PSO firstly. He introduced three patterns and compared their performance on several benchmark problems. Based on the former works, a new pattern FLsplit is bought about, where the topology of neural network is considered and parallel computation can be realized more easily. In addition, a BParticle is involved in the swarm to stabilize the learning result and decrease the cost of computation. At the last, BP, PSO and Flsplit are tested on two benchmark functions, and their performances are analyzed at the same time.

2. PSO Algorithm

PSO was firstly presented by Kennedy and Eberhart [2]. The searching space of $\boldsymbol{D}$-dimension is defined as $S \subset R^D$. At the beginning, there are many particles distributed randomly in the space flying with arbitrary velocities. They keep modifying velocities according to their prior best position and the global best position until the optimal solution is found. For particle i, there are position vector $X_i = (x_{i1}, x_{i2}, ..., x_{iD})^T \in S$, velocity vector $V_i = (v_{i1}, v_{i2}, ..., v_{iD})^T \in S$, and prior best position $P_i = (p_{i1}, p_{i2}, ..., p_{iD})^T \in S$. If $\boldsymbol{g}$ is the index of the particle containing global best position, the updating rule can be described as following:

$$v_{id} = w * v_{id} + c_1 * rand(\,) * (p_{id} - x_{id}) + c_2 * Rand(\,) * (p_{gd} - x_{id}) \quad (1)$$

$$x_{id} = x_{id} + v_{id} \quad (2)$$

in which $\boldsymbol{d}$=1,2,…,$\boldsymbol{D}$. $\boldsymbol{w}$ is the inertia weight [12] used for balancing the global and local searching. $\boldsymbol{c_1}$ and $\boldsymbol{c_2}$ are positive, representing the cognitive factor and social factor respectively. They determine particle's bias of own best and global best. *rand*( ) and *Rand*( ) are random values within [0,1], aiming to alleviate local optimization. It was drawn out that the preferring values for $\boldsymbol{c_1}$ and $\boldsymbol{c_2}$ are 0.25 and 0.75, and $\boldsymbol{w}$ should be decreased with learning times [13].

3. PSO Neural Network

3.1. Single Swarm Case

By now, most of the researches on PSO neural network have been focused on single swarm [4-11]. Each particle is considered as a network. Take the topology of neural network shown in figure 1 for example. The goal of PSO is to find out the solution minimizing the error function on $\boldsymbol{W}$-dimension space, where $W=N*M+M*C$.

Firstly, the searching space is defined as $S \subset R^W$, and the properties of particle are defined as ***W***-dimensional vectors as well. Then, the output and error of each particle can be computed out through propagating forward with all the samples. At last, the particles are chosen and modified using the error as fitness. The training process is listed below [14].

Step1. For each network, iterate over the training data set and keep a running sum of the network error.

Step2. Compare all of the network errors to find the best network in the neighborhood.

Step3. If one of the networks has achieved the minimum error required, record its weights and exit the program.

Step4. Otherwise, for each network, execute the PSO algorithm to update its position and velocity vectors.

Step5. Loop to Step 1.

The difference between BP and PSO is that BP modifies network with every sample, but PSO does this with the whole sample space because it needs not propagating the error back. Therefore, PSO learns without the influence of sample order, and it would perform better on approximating wholly.

3.2. Cooperative PSO Neural Network

3.2.1 Motivations

It is not the best when using a single swarm. Next will present an argument to help explain why splitting the vector results in improved performance [3].

Consider an ***n***-dimensional vector ***v***, constrained to the error surface by the function to be minimized. Somewhere in this vector three components, $\boldsymbol{v}_a,\boldsymbol{v}_b,\boldsymbol{v}_c$, $1 \le a < b < c \le n$, will now be examined. Assume that the optimal solution to the problem requires that all three components must have a value of say 20. Now consider a particle swarm containing, in particular, two vectors $\boldsymbol{v}_j$ and $\boldsymbol{v}_k$. Let's assume that $\boldsymbol{v}_j$ is currently the global best solution in the swarm, so that $\boldsymbol{v}_k$ will be drawn towards it in the next epoch. Assume that the three components $\boldsymbol{v}_{ja},\boldsymbol{v}_{jb},\boldsymbol{v}_{jc}$ have the values (17, 2,17), respectively. The equivalent components in vector $\boldsymbol{v}_k$, namely $\boldsymbol{v}_{ka},\boldsymbol{v}_{kb},\boldsymbol{v}_{kc}$, have the values (5, 20, 5) respectively. In the next epoch, the vector $\boldsymbol{v}_k$ might be updated so that it now contains the sub-vector (15, 5, 15) at positions $\boldsymbol{v}_{ka},\boldsymbol{v}_{kb},\boldsymbol{v}_{kc}$. If it is assumed that the function at the new $\boldsymbol{v}_k$ yields a smaller error value, this will be considered an improvement. However, the valuable information contained in the middle component ($\boldsymbol{v}_{kb}$) has been lost, as the optimal solution requires a sub-vector of (20,20,20).

The reason for this behavior is that the error function is computed only after all the components in the vector have been updated to their new values. This means an improvement in two components will overrule a potentially good value for a single component. From this viewpoint, F. van den Bergh brought out the cooperative PSO neural network.

3.2.2 Existing methods

The main idea of cooperative PSO is to split the vector into several parts and optimize independently. But it is not the better the more parts. Increasing the number of swarms leads to a directly proportional increase in the number of error function evaluations, as one function evaluation is required for each particle in each swarm during each epoch. Therefore, an efficient splitting method is necessary for both improving the performance and saving the cost.

F. van den Bergh introduced three patterns according to the structure of neural network, Lsplit, Esplit and Nsplit. Lsplit splits the network weights between two swarms. The first swarm contains the $N*M$ weights from the first layer, while the second swarm optimizes the $M*C$ second-layer weights. Esplit takes the serialized $(N*M+M*C)$-dimensional vector and split it in half. In Nsplit, for each hidden and output unit, a swarm is treated containing all the weights entering that unit. This results in M N-dimensional vectors plus C M-dimensional vectors, for a total of $M+C$ swarms. The performances of PSO, Lsplit, Esplit and Nsplit were compared through several experiments, among which the Nsplit was the best.

3.3. FLsplit

Nsplit performs well, but it's not convenient to parallel. The number of input and output units varies in different applications, which will result in modification of each particle in Nsplit, and then the unit of parallel system. In this process, extra programming work and resource waste are inevitable. So, a novel pattern FLsplit is prompted in this paper suiting parallel computation better. The idea is that weights connected with each input or output neuron result in a M-dimensional vector, as shown in figure 1.

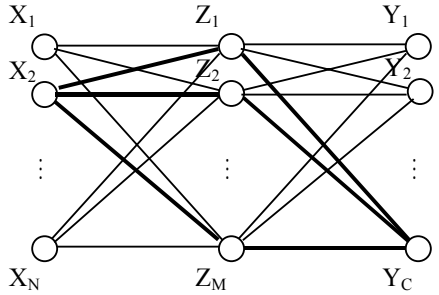

Fig. 1 Splitting in FLsplit

Here, all the swarms and particles have the same properties, which simplify the parallelization. It needs only add or remove unit when the number of input and

output neurons varies, and this will cost much less than Nsplit. On the other hand, FLsplit will optimize as well as Nsplit for their similar splitting, which will be argued later.

3.4. BParticle

After being split, the weights can be optimized independently. But there still is a problem necessary to address. If there is more than one optimal solution the finally returned position will be unstable. In that the search is terminated once there is some particle satisfying the predefined criteria. In the case of two optimal solutions, the learning result could be the positions of particles closest to either one. They will vary greatly if these two solutions are distant. This can be solved through adding a BParticle.

The distinctness of BParticle is the initial position and velocity. Instead of random initialization to promise global optimization, Bparticle's position is assigned to the best of the others'. And the velocity is 0 vector. That is, if particle ***i*** has the highest fitness with position $X_i(0) = (x_{i1}(0), x_{i2}(0), ..., x_{id}(0))^T$ in the initial state of swarm, there are $X_B(0) = (x_{i1}(0), x_{i2}(0), ..., x_{id}(0))^T$ and $V_B(0) = (0,0,...,0)^T$. With the best initial position and 0 velocity, BParticle would move to the optimal point more conveniently. And the max velocity of BParticle ensures to make the solution more stable.

Another permit of BParticle is the less cost for selecting the best particle. The solution vector is split into ***SN*** parts, each optimized by a swarm with NP_i particles. This leaves possible $\prod_{i=1}^{SN} NP_i$ vectors to choose from. But it is reduced greatly with BParticle. While selecting, the factors of other swarms could be represented by their Bparticles, and all best particles could be selected from $\sum_{i=1}^{SN} NP_i$ vectors.

4. Settings and Results of Experiment

In this part, the performances of BP, PSO and FLsplit will be analyzed through two experiments of function approximation. The non-linear functions are $f(x) = \cos(2\pi x)$ and $f(x) = 1/(x + 0.01)$, $0 \le x \le 1$, and the sample space is defined as $\{(0,f(0)),(0.1,f(0.1)),\ldots,(1.0,f(1.0))\}$.

4.1. Constructing Neural Network

A three-layer neural network is established as figure 2. There are 8 neurons in the hidden layer, and one neuron in input layer and output layer respectively. The activation function of hidden neuron is *S* function $f(x) = \frac{1}{1+e^{-x}}$, and that of outputting neuron is $f(x)=x$.

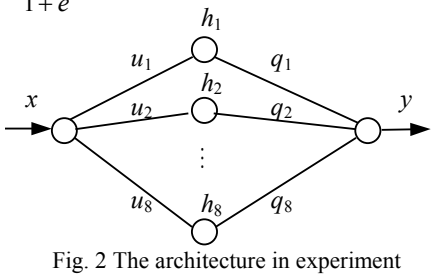

Fig. 2 The architecture in experiment

For sample ***i***, the output of network is o_i, and the value of function is y_i. Then the error is $E_i = \frac{1}{2}(o_i - y_i)^2$, i=1,2,…,11, and the evaluating function for network is $E = \frac{1}{11}\sum_{i=1}^{11} E_i$.

4.2. Settings of BP

The weights of network are initialized within [-0.2,0.2], in order to keep the inputs of hidden neurons in active zone of *S* function. α=0.02. But there is $\alpha' = 0.01$ for updating q_i, because the output is more sensitive to q_i.

4.3. Settings of PSO

The searching space is defined as $S \subset R^{16}$. There are 10 particles and one BParticle in the swarm. The vectors of particles are $X_i = (x_{i1},...,x_{i16})^T$, $x_{ij} \in [-0.2,0.2]$, and $V_i = (v_{i1},...,v_{i16})^T$, $v_{ij} \in [-0.0375,0.0375]$, $i = 1,2,...,10$, $j = 1,2,...,16$. The vectors of Bparticle are X_B=X_k, and V_B=$(0,0,\ldots 0)^T$, where k is the particle makes the least E, k=1,2,…,10. The max velocity of all particles is V_{max}=(0.1,…,0.1). $\boldsymbol{c_1}$=0.25, $\boldsymbol{c_2}$=0.75.

4.4. Settings of FLsplit

The weights are split into two parts and the searching space of each swarm is $S' \subset R^8$. The settings are similar to those of PSO NN, except that the position and velocity vectors are of 8-dimension, BParticle is used when selecting the best particles so as to reduce the computation cost, and the positions of Bparticles are used to evaluate the networks' error so as to stabilize the learning result.

4.5. Performance Comparison

BP, PSO and FLsplit approximate the benchmark functions respectively. The success rate and training epoch of all methods are analyzed within 10000 epochs with E=0.01. And each method is performed 20 times.

(1) $f(x) = \cos(2\pi x)$, $0 \le x \le 1$

Table 1. Performance comparison in $f(x) = \cos(2\pi x)$

Type	Success rate	Even epochs	Least epochs	Most epochs
BP	80%	8066	7226	9984
PSO	100%	563	200	1900
FLsplit	100%	94	64	126

(2) $f(x) = 1/(x + 0.01)$, $0 \le x \le 1$

Table 2. Performance comparison in $f(x) = 1/(x + 0.01)$

Type	Success rate	Even epochs	Least epochs	Most epochs
BP	100%	3776	3551	4062
PSO	90%	1089	533	2783
FLsplit	100%	603	461	741

5. Conclusions

Compared with BP algorithm, FLsplit achieves the predefined error in less training epochs with higher success rate. The influence of sample order on learning result is eliminated by training on the whole sample space.

The advantage is also tenable when compared with PSO. But the requirement for equipment will grow with the increase of swarms. Compared with Esplit, Lsplit and Nsplit, the FLsplit method is more suitable for parallel computation.

It can be concluded from the above description that the FLsplit and BParticle in this paper will reduce the computation cost and stabilize the learning result. But the stability is still not as good as BP. It is also related with the learning time, which is not the better the longer. To select the terminating criteria properly is also important, which is one of the future works to be continued.

6. References

[1] Jiang Zongli, "Introduction to Neural Network," *High Education Press*. pp. 39-54, 2001.

[2] Eberhart, R., and Kennedy, J., "Particle Swarm Optimization," *1995 Proceedings of the IEEE Conference on Neural Networks*. Vol.4. pp. 1942-1948, 1995.

[3] van den Bergh F. and Engelbrecht A. P., "Cooperative Learning in Neural Networks Using Particle Swarm Optimizers," *Computer Journal*. Vol.26, pp. 84-90, 2000.

[4] Alex Conradie, Risto Miikkulainen and Christiaan Aldrich, "Adaptive Control Utilising Neural Swarming," *GECCO 2002*. pp: 60-67

[5] Eberhart R. C. and Hu X., "Human tremor analysis using particle swarm optimization," *Proceedings of the IEEE Congress on evolutionary computation*. Washington D.C., pp. 1927-1930, 1999.

[6] AP Engelbrecht and A Ismail, "Training product unit neural networks," *Stability and Control: Theory and Applications*. Vol.2(1–2), pp. 59–74, 1999.

[7] Zhiwei Wang et al., "Particle Swarm Optimization and Neural Network Application for QSAR," *Proceedings of the 18th International Parallel and Distributed Processing Symposium*. Santa Fe New Mexico, 2004.

[8] Tsou, D. and MacNish, C., "Adaptive particle swarm optimisation for high-dimensional highly convex search spaces," *Proceedings of IEEE Congress on Evolutionary Computation 2003*. Canbella Australia, pp. 783-789, 2003.

[9] Conradie, A. v. E., Miikkulainen, R., and Aldrich, C., "Adaptive Control utilizing Neural Swarming," *Proceeedings of the Genetic and Evolutionary Computation Conference*. San Francisco, 2002.

[10] Lu WZ, Fan HY, Lo SM., "Application of evolutionary neural network method in predicting pollutant levels in downtown area of Hong Kong," *NeuroComputing*, Vol.51, pp.387-400, 2003.

[11] V.L. Georgiou et al., "Optimizing the Performance of Probabilistic Neural Networks in a Bioinformatics Task," *EUropean Network of Intelligent Technologies for Smart Adaptive System*. 2004.

[12] Shi Y. and Eberhart RC., "A modified particle swarm optimizer," *Proceedings of the IEEE International Conference on Evolutionary Computation*. Piscataway, pp.69-73, 1998.

[13] Gail DePuy, Tim Hardin and Jeff Greenwell, "Statistical evaluation of the particle swarm optimization algorithm," *Experimental Design*. Industrial Engineering 563, University of Louisville, 2003.

[14] Paulo F. Ribeiro, W. Kyle Schlansker, "A Hybrid Particle Swarm and Neural Network Approach for Reactive Power Control," http://engr.calvin.edu/PRibeiro_WEBPAGE/courses/engr302/Samples/ReactivePower-PSO-wks.pdf, 2004.

[15] Hou Yuanbin, "The study of an initializing algorithm for increasing the convergent speed of neural network," *Patten Recognition and Artificial Intelligence*. Vol.4, pp.385-389, 2001.