Evolutionary Extreme Learning Machine – Based on Particle Swarm Optimization

You Xu and Yang Shu

Department of Mathematics, Nanjing University,
Nanjing 210093, P.R. China
mathena@163.com

Abstract. A new off-line learning method of single-hidden layer feed-forward neural networks (SLFN) called Extreme Learning Machine (ELM) was introduced by Huang et al. [1, 2, 3, 4]. ELM is not the same as traditional BP methods as it can achieve good generalization performance at extremely fast learning speed. In ELM, the hidden neuron parameters (the input weights and hidden biases or the RBF centers and impact factors) were pre-assigned randomly so there may be a set of non-optimized parameters that avoid ELM achieving global minimum in some applications. Adopting the ideas in [5] that a single layer feed-forward neural network can be trained using a hybrid approach which takes advantages of both ELM and the evolutionary algorithm, this paper introduces a new kind of evolutionary algorithm called particle swarm optimization (PSO) which can train the network more suitable for some prediction problems using the ideas of ELM.

1 Introduction

ELM algorithm [1, 2, 3, 4] is a novel learning algorithm for single layer feed-forward neural networks (SLFNs). In ELM, the hidden neuron parameters (the input weights (linking the input layer to the hidden layer) and hidden biases or the RBF centers and impact factors) are randomly chosen, and the output weights (linking the hidden layer to the output layer) are determined by calculating Moore-Penrose (MP) generalized inverse.

ELM divides the learning procedure in a SLFN into two parts, the first part is to decide the nonlinear parameters, and the other is to decide the linear parameters. For instance, the model of a standard SLFN can be described as follows:

$$y_k = f_k(x) = \sum_{j=1}^{q} \theta_{kj} \phi(a_j), \tag{1}$$

$$a_j = \sum_{i=1}^{d} \omega_{ij} x_i - \omega_{0j} \qquad j = 1, 2, \cdots, q; \tag{2}$$

where a_j is the input activation of the jth neuron in the hidden layer; $\phi(\cdot)$ is the activation function and usually is sigmoid; q is the number of neurons

J. Wang et al. (Eds.): ISNN 2006, LNCS 3971, pp. 644–652, 2006.

in the hidden layer (NNHL); $\omega_{ij}, \omega_{0j} (i = 0, 1, \cdots, d; j = 1, 2, \cdots, q)$ are the weights and biases; $y \in R^m$ is the output and $x \in R^d$ is the input. Here (1) is linear.

In fact we can rewrite the input-output relationship as follows:

$$y = f(x, \omega, \theta) = \sum_{j=1}^{q} \phi(x, \omega_j)\theta_j, \tag{3}$$

thus, for N distinct samples $\{x_i, y_{di}, i = 1, 2, \cdots, N\}$, the learning model of these networks can be represented in an unified form as follows:

$$\begin{bmatrix} y_{d1} \\ \vdots \\ y_{dN} \end{bmatrix} = \begin{bmatrix} \phi(x_1, \omega_1) & \cdots & \phi(x_1, \omega_q) \\ \vdots & & \vdots \\ \phi(x_N, \omega_1) & \cdots & \phi(x_N, \omega_q) \end{bmatrix} \begin{bmatrix} \theta_1 \\ \vdots \\ \theta_q \end{bmatrix} + \varepsilon \tag{4}$$

If all the ω_js are determined, in order to make the training error, that is, the norm of ε minimal, we need to find the least square solution to (6). In fact ELM tries to find the its smallest norm least-square solution. In short, ELM assigns the parameters ω_js arbitrarily in the input layer—in the BP network these are input weights and biases while in the RBF case are the kernel centers and impact widths. Then it calculates the hidden layer output matrix H. The output vector $\theta = H^{\dagger} \cdot T$ where $H^{\dagger}$ represents the Moore-Penrose (MP) generalized inverse of matrix H. As θ is the smallest norm least-square solution of the linear problem $H\theta = T$, ELM claims that θ makes the network have both the smallest training error and the good generalization performance [1].

ELM is not the same as traditional gradient-based method because it can achieve a good generalization performance at an extremely fast learning speed. However, the solution of ELM highly depends on a certain set of input weights and hidden biases. In fact, ELM tries to find the global minimal point of a localized linear problem, so different sets of input weights and hidden biases may have different performances. Therefore, although ELM achieves the global minimal point of the linear system, maybe it is not the global minimal point in the problem space [6]. This kind of issue can be solved in two different ways: one is to increase the hidden units that make some randomly assigned parameters be very close to the best set of parameters [6, 3]; the other is to find the best set of parameters without increasing the hidden units. Xu [6] adopts a kind of gradient-based algorithm to find the best set but it cannot avoid the local minima problem. Huang, et al. [5] introduces a hybrid form of differential evolutionary algorithm and ELM method called E-ELM to train a SLFN.

It is well known that evolutionary algorithms can avoid the local minima problem using a global searching strategy. In E-ELM, users need to decide the mutation factor (the constant F) and the crossover factor (the constant CR).[1] In other words, if you want use E-ELM algorithm in some cases like pattern

[1] The selection factor in ELM is 0 which is not defined explicitly.

recognition, you need to decide and turn three parameters manually. Is there a kind of evolutionary algorithm that has the advantages of "global searching" but less or no parameter needs turned? The answer to this question is the particle swarm optimization (PSO) algorithm.

PSO is a population based stochastic optimization technique developed by Dr. Eberhart and Dr. Kennedy in 1995, inspired by social behavior of bird flocking or fish schooling. The concept of particle swarm originated from the simulation of a simplified social system. The original intent was to graphically simulate the choreography of bird flock or fish school. However, it was found that particle swarm model could be used as an optimizer. In this algorithm, a swarm consists of many particles, where each individual particle keeps track of its position, velocity and best position. The global best position in the problem space can be found via iterations. In iteration, a particle updates its own position by keeping track of two values: the best position *pbest* this particle have arrived; the best position *gbest* this particle swarm has arrived. Usually, whether the position is better or worse is judged by the fitness function as its value might be considered as the distance from the best position. The particles then update their velocities and positions according to the formulae as follows:

$$v_{t+1} = \omega \times v_t + c_1 rand_1(pbest_t - present_t) + c_2 rand_2(gbest_t - present_t) \quad (5)$$

$$present_{t+1} = present_t + v_{t+1} \quad (6)$$

where v is the velocity of a particle and *present* is the current position of a particle. *rand*s are random numbers in the range of $(0, 1)$, ω is called the inertial factor. It usually in the interval of $[0.4, 0.9]$, c_1 and c_2 are the learning factors and usually equal to constant 2. So the only parameter needs turning manually is ω.

During the iteration, the particle learns from the society and moves towards the potential best position using both its own and social knowledge. In fact, every iteration is like an evolution and every position changing is like mutation or crossover. Therefore, from this viewpoint, we can classify the PSO algorithm to a kind of evolutionary computing.

However, PSO has a few differences with traditional generic algorithm. In the first place, the sorting procedure of the fitness values and the selecting procedure of the population are not necessary any more in this algorithm. Besides these, it needs neither crossover nor mutation operation, which makes the algorithm much easier. In the second place, no factor in this algorithm needs to turn manually. In fact, the inertial factor usually equals to constant 0.5 and works well in most applications. This is unlike the traditional generic algorithm that requires at least three parameters[2] that would make the algorithm much more difficult to implement in some applications.

[2] The mutation factor, the crossover factor and the selection factor.

2 The PSO-ELM Algorithm

The essential issue in optimizing the performance of ELM is 1) to optimize the hidden neuron parameters and 2) to reduce the number of hidden neurons to make the network have a better and quicker performance [1, 5, 6].

In order to achieve these two goals, many algorithms originated from ELM were proposed. The Evolutionary Extreme Learning Machine (E-ELM) [5] is a good example that uses differential evolutionary algorithm and ELM to train the network suitable for the pattern recognition problems. Here we just replace the differential evolutionary algorithm with PSO algorithm. Obviously, the restriction that Φ should be differentiable to ω will no longer be a necessary condition to apply the algorithm. Moreover, less factors need turning, which will make the final implementation much easier. However, PSO has some detailed problems that need to be considered carefully. The first one is the issue that particles might fly out of bound in position or velocity.

1. The position is out of bound on a certain dimension

It is easy to find that when $rand_1$ is quite close to 1 and $rand_2$ is quite close 0, the particle will fly over the global best position, if the position *gbest* is quite close to the boundary in a certain dimension, occasionally the particle will go out of the bound. Since the out-bounded particles that are not the suitable solutions to the original problem any longer. This issue should be considered carefully. Unfortunately, hardly any researchers who applied the PSO algorithm in the training of neural networks treat the issue seriously; as they adopted the BP algorithm to train the network, which needs unbounded values to avoid local minima. However, in the ELM algorithm, the input weights and hidden biases should be bounded. For instance, if we choose all the hidden biases smaller than -10, the elements in the hidden layer output matrix will be very close to 1 (assuming that Φ is sigmoid), thus the MP generalized inverse of matrix H becomes trivial. In fact the rank of H here can be considered as 1 rather than K that we assumed, where K is the number of hidden neurons.

In fact, all the weights should be in the range of $[-1, 1]$ and the hidden biases should be in the range of $[-1, 1]$. [3]

We adopt the author's constraints directly and there are three different strategies to deal with the position out-of-bound issue:

(1) the boundary is absorptive.
The out-bounded particles will stay at the boundary in this dimension, It is like that the boundary is absorptive such that the out-bounded particles will adhere to the boundary.

(2) the boundary is reflective.
The out-bounded particles will inverse their directions in this dimension and leave the boundary keeping the same velocities.

[3] Such normalization was not stated explicitly in [1], but we can get them from the MATLAB source code of ELM which is available in the ELM portal http://www.ntu.edu.sg/home/egbhuang.

(3) the boundary is abysmal.
The out-bounded particles will disappear, which means they will not be taken into account anymore.

In this algorithm, in order to make an easier implementation, we adopt strategy (2). In the algorithm, we will simply inverse the directions of the out-bounded particles while keeping their velocities unchanged.

2. The particle's velocity is out of bound in a certain dimension

To some extent, this constraint is a necessary condition that avoids the PSO algorithm being trapped in local minima and ensures a global even search in the solution space. If a maximal velocity v^i_{max} were given in the ith dimension, and $v^i_{new} > v^i_{max}$, then the velocity in this dimension would be normalized to

$$v^i_{new} = \frac{v^i_{max}}{\|v^i_{new}\|} v^i_{new}. \tag{7}$$

In order to train a neural network quickly, according to [5], an algorithm should optimize the input weights and hidden biases using some strategies and adopting ELM to calculate the output layer weights. Actually, in most cases, these two steps are implemented iteratively. It is true that PSO algorithm can train a neural network accompanying with BP algorithm. However, BP is much slower than ELM. ELM also ensures the global minima, which means ELM can achieve a good result in the output layer. Therefore, we adopt PSO in the procedure of searching a best set of input parameters and employ ELM to train the output layer directly. The fitness function or the root means standard error (RMSE) of PSO is defined as follows:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N} \|\varepsilon_i\|_2^2}{m \times N}} \tag{8}$$

In the first step, the algorithm should generate a swarm of input values, then make an evolutionary computing until the maximum iterations or minimum criteria attained. In fact, it is difficult to define the minimum criteria here, as the RMSE or other factors may not have the limit of 0; and usually it is impossible to pre-evaluate this limit before calculation. However, as the PSO algorithm is stochastic, if the RMSE keeps unchanging during evolution or δ_{RMSE} is smaller than a pre-defined ε, we have sufficient reason that best point has been found.

Based on the careful analysis stated above, the algorithm using PSO and ELM is given below, we named it PSO-ELM:[4]

Algorithm PSO-ELM. *Given a training set* $\aleph = \{(x_i, t_i) | x_i \in R^N, t \in R, i = 1, \cdots, N\}$, *maximum iteration and* ε.

Step 1: Generate the initial swarm and set the boundary;
Step 2: Calculate the output weights using ELM algorithm;

[4] The MATLAB source code can be obtained from authors via e-mail.

Step 3: Update particle position, calculate RMSE and δ_{RMSE};
Step 4: *If* $\delta < \varepsilon$ or maximal iteration attained, Goto **Step 5**;
Else Goto **Step 2**;
Step 5: Output the best position, **EXIT**.

3 Prediction Problem

According to the conversational viewpoints, a single layer forward neural network can deal with some kinds of static problems (without the time factor). For instance, we can find their applications in regressing a multi-variable function, especially in regressing the nonlinear orbit of a robot in the assembly line; or in classifying the samples out of a raw data set like pattern recognition or disease diagnoses; or in compressing the data like encoding and extracting the characteristics from a digital image. However, the single layer feed-forward neural network might not be a suitable model for some dynamic problems with dynamic behaviors, e.g. prediction problems, real-time adaptive control systems. In other words, there are neither time factors nor feedback mechanisms in BP networks which make it not adaptive to dynamic models.

In fact, there are two different kinds of predictions: to predict the results using the information provided by last state; to predict the results using all the information before. The first one is essentially not dynamic. In fact, it has no relationship with time factor, as every time you have the last state, you will get current state just the same. Analog to the dynamic programming in the operational research [7] that current state only depends on the last state and has no relationship with earlier ones; the prediction of the future only relates to the factors currently. It fact it is called Markov attribute. Because it is hard to find a formula to describe or summarize the relationship between these two states and it is even harder to find which factors will actually affect the results, we employ the neural networks to deal with such kinds of problems. Additionally, the relationship is usually nonlinear that neural network is the only suitable model dealing with such problems. Researches have already applied the neural networks in the prediction cases such as the water-quality pollution prediction [8].

We rewrote the program according to the algorithm provided in [8] using MATLAB neural network toolbox. This algorithm uses PSO algorithm in global search and accelerates the convergence of a swarm using Levenberg-Marquardt (LM) algorithm. Unfortunately, we could not gain sufficient data of this problem. Instead, we got the statistical data about the farm production from 1978 to 2000 from the annals published by National Statistic Bureau of China in 2003 [9]. Seven factors affected the production of farm goods: irrigation area, amount of fertilizer, electric power, total mechanical power, labor force, planting acreage and area hit by natural calamity. In Table 1 we use $x_1, \cdots, x_7$ to denote these seven factors. In order to eliminate the errors caused by different dimensions of different factors, a Z-Score transformation is performed on the raw data. Z-Score is a measure of the distance in standard deviations of a sample from the mean. Calculated as $(X - \overline{X})/\sigma$ where $\overline{X}$ is the mean and σ

Table 1. Standard values of corn production and its influencing factors in China

Year	Production	x_1	x_2	x_3	x_4	x_5	x_6	x_7
1978	-1.7992	-0.7009	-1.5436	-1.1178	-1.4246	-2.2198	2.5280	0.6992
1979	-1.3592	-0.6891	-1.3545	-1.0748	-1.2842	-1.8086	2.1035	-0.9361
1980	-1.5453	-0.7248	-1.1834	-1.0193	-1.1665	-1.3167	1.4531	-0.1978
1981	-1.4734	-0.8223	-1.0549	-0.9481	-1.0860	-0.7364	0.7234	-0.8765
1982	-0.9993	-0.9456	-0.9553	-0.9087	-1.0055	-0.4193	0.2438	-1.8292
1983	-0.4721	-0.8005	-0.8185	-0.8530	-0.8841	-0.0905	0.4314	-1.6029
1984	-0.1499	-0.8598	-0.7437	-0.8111	-0.7570	-0.0639	0.0585	-2.0075
1985	-0.6035	-0.9893	-0.7101	-0.7458	-0.6351	-0.9541	-1.2363	-0.2208
1986	-0.4040	-0.9304	-0.5654	-0.6326	-0.4595	-0.8764	-0.5669	0.1758
1987	-0.1914	-0.8754	-0.5012	-0.5278	-0.2970	-0.6080	-0.4595	-0.5471
1988	-0.3627	-0.8838	-0.3683	-0.4504	-0.1472	-0.2170	-0.8266	0.7112
1989	-0.1460	-0.7158	-0.1667	-0.3362	-0.0186	0.4405	-0.1591	0.1552
1990	0.4762	0.0559	0.0512	-0.2577	0.0366	1.0386	0.2451	-1.0644
1991	0.3001	0.1860	0.2520	-0.0851	0.0953	1.6060	-0.1242	1.3680
1992	0.4186	0.4244	0.3689	0.1239	0.1745	1.5063	-0.6865	0.7770
1993	0.6411	0.4671	0.5761	0.3245	0.3045	0.9864	-0.7028	0.4184
1994	0.4579	0.4768	0.7313	0.6577	0.4756	0.6073	-1.3328	1.3082
1995	0.8040	0.6389	0.9891	0.9221	0.6751	0.3697	-0.8468	-0.0123
1996	1.4139	0.9804	1.2080	1.1504	0.8844	0.3203	-0.0492	0.1549
1997	1.2472	1.2465	1.3506	1.3938	1.1833	0.4367	0.0675	1.0771
1998	1.5387	1.5746	1.4471	1.4840	1.4583	0.5646	0.3480	0.6069
1999	1.4758	1.8424	1.4850	1.6750	1.7848	0.7551	0.1473	0.5834
2000	0.7326	2.0479	1.5057	2.0355	2.0930	0.6788	-1.3589	1.2574

Table 2. Performance comparison

Algorithms	**Training Time** (seconds)	**Output Value** 1999	2000	**Relative Error** 1999	2000
PSO-LM	10.921	1.4416	0.7018	2.32%	4.20%
BP(MATLAB)	0.732	1.4573	0.7737	1.25%	5.61%
PSO-ELM	4.478	1.4407	0.7010	2.33%	4.31%

is the standard deviation. In the experiment, we train the network using data from 1978 to 1998 and try to predict the production in the years of 1999 and 2000. The network has 7 inputs, 15 hidden neurons, and 1 output. We make a comparison between PSO-ELM and the algorithm provided in [8]. Dramatically the results of these two algorithms are quite close to each other. Moreover, the BP algorithm also provides the similar prefect results. The relative prediction error of year 2.33% and 4.31% in the year of 1999 and 2000 respectively. Thanks to the quick learning speed of ELM, the total learning time of PSO-ELM is less than 5 seconds, and the PSO-LM algorithm takes more than 10 seconds in average.

As we cannot get the original data from [8] and make exact comparison between these two algorithms, this comparison here might be a very particular

case that does not reveal the full characteristics of them. However, as PSO-ELM adopts the quick learning speed from ELM, it is always true that PSO-ELM algorithm is much quicker than the hybrid PSO and BP algorithm.

4 Conclusions

In this paper, in order to simplify Evolutionary ELM algorithm, a hybrid PSO and ELM algorithm was introduced. Although many researchers applied similar PSO algorithms in the training of neural networks, rarely did they consider the boundary conditions. In fact, the boundary condition is quite a significant part in the PSO algorithm. Without these constraints, a PSO algorithm will be entrapped in local minima in all probability. Additionally, the ELM algorithm has this restriction although it is not claimed explicitly in [1]. After the careful analysis of these issues, a PSO-ELM algorithm was given. Moreover, by adopting the ELM in the training step, the complete training procedure took less time than other PSO based algorithms. Additionally, we discussed the applicability of feed-forward neural networks in prediction problems, and found the right situation in which could we apply this model. Finally the experimental results originated from the real world problems stated that this algorithm could achieve as the same precision as traditional BP algorithms. There are three main advantages of PSO-ELM. In the first place, it needs only one parameter compared to three parameters in the traditional generic method. In the second place, the constraint that the activation function should be differentiable to x is no longer required. It is true that ELM can be used in the cases such as the threshold network where activation function is not differentiable [10]. However, in [5], this kind of constraints is needed. The PSO-ELM removes the constraint that makes the algorithm more applicable. In fact, we would like to do some future work on this field and plan to implement such algorithm to the embedding systems. In the third place, the learning speed of ELM makes this algorithm much quicker than the PSO-based algorithms such as PSO-LM.

References

1. Huang, G.B., Zhu, Q.Y., Siew, C.K.: Extreme Learning Machine: A New Learning Scheme of Feedforward Neural Networks. In: Proceedings of International Joint Conference on Neural Networks (IJCNN2004), Budapest, Hungary (25-29 July, 2004)
2. Huang, G.B., Zhu, Q.Y., Siew, C.K.: Extreme Learning Machine: Theory and Applications. Neurocomputing (in press) (2006)
3. Huang, G.B., Siew, C.K.: Extreme Learning Machine with Randomly Assigned RBF Kernels. International Journal of Information Technology. **11** (2005)
4. Huang, G.B., Siew, C.K.: Extreme Learning Machine: RBF Network Case. In: Proceedings of the Eighth International Conference on Control, Automation, Robotics and Vision (ICARCV 2004), Kunming, China (6-9 Dec, 2004)
5. Zhu, Q.Y., Qin, A., Suganthan, P., Huang, G.B.: Evolutionary Extreme Learning Machine. Pattern Recognition **38** (2005) 1739–1763

6. Xu, Y.: A Gradient-Based ELM Algorithm in Regressing Multivariable Functions. In: Proceedings of International Symposium on Neural Networks (ISNN2006), Chendu, China (29-31 May, 2006)
7. Bellman, R.E.: Dynamic Programming. Princeton University Press, Princeton, N.J. (1957)
8. Chau, K.: A Split-Step PSO Algorithm in Prediction of Water Quality Pollution. In: Proceedings of the International Symposium on Neural Networks (2005), Chongqing, China (30 May - 1 June, 2005) 1034–1039
9. National Bureau of Statistics of China: The Annals (2003) of Statistical Data in China. China Statistics Press, Beijing (2004)
10. Huang, G.B., Zhu, Q.Y., Mao, K.Z., Siew, C.K., Saratchandran, P., Sundararajan, N.: Can Threshold Networks Be Trained Directly? IEEE Transactions on Circuits and Systems - II **53** (2006)