Artificial Bee Colony Training of Neural Networks

John A. Bullinaria and Khulood AlYahya

School of Computer Science, University of Birmingham
Birmingham, B15 2TT, UK
[j.a.bullinaria,kya020]@cs.bham.ac.uk

Abstract. The Artificial Bee Colony (ABC) is a recently introduced swarm intelligence algorithm for optimization, that has previously been applied successfully to the training of neural networks. This paper explores more carefully the performance of the ABC algorithm for optimizing the connection weights of feed-forward neural networks for classification tasks, and presents a more rigorous comparison with the traditional Back-Propagation (BP) training algorithm. The empirical results show that using the standard "stopping early" approach with optimized learning parameters leads to improved BP performance over the previous comparative study, and that a simple variation of the ABC approach provides improved ABC performance too. With both improvements applied, we conclude that the ABC approach does perform very well on small problems, but the generalization performances achieved are only significantly better than standard BP on one out of six datasets, and the training times increase rapidly as the size of the problem grows.

1 Introduction

Recently, the study of different insect behaviours, animal colonies and swarms has led to the introduction of many nature inspired optimization algorithms [6]. Such swarm intelligence algorithms typically involve a group of simple agents that cooperate with each other locally, either directly or indirectly, and these simple interactions lead to the emergence of complex intelligent global behaviour for solving problems. The best known examples are Particle Swarm Optimization (PSO), inspired by the social behaviour of flocks of birds, and Ant Colony Optimization (ACO), inspired by the foraging behaviour of ants.

A more recent, and less well studied, swarm intelligence algorithm is the Artificial Bee Colony (ABC) originally proposed by Karaboga [7], and inspired by the foraging behaviour of honeybees. There are many possible applications of ABC, but this paper will concentrate on their use in optimizing the weights of artificial Neural Networks (NNs). Of course, there already exist many hybrid neural network learning algorithms that aim to improve upon standard gradient descent algorithms such as Back-Propagation (BP), but the advantages of those approaches are debatable. In particular, Cantu-Paz and Kamath [4] have shown that most combinations of Evolutionary Algorithms (EAs) and neural networks performed no better than simple BP on the classification tasks they tested. Karaboga and colleagues [9, 11], however, have previously applied ABC to neural network learning and claimed some success. The aim of this paper is to explore more carefully how effective ABC really is for training feed-forward neural networks to perform classification tasks.

In the following sections, we first describe the ABC algorithm and how it can be applied to neural network training. Then we describe and present results from a series of computational experiments that explore the power of standard and improved ABC for neural network applications in comparison with appropriately optimized BP. The paper ends with our conclusions and a discussion of the implications.

2 The Standard Artificial Bee Colony Algorithm

The ABC algorithm is a stochastic optimization algorithm inspired by the foraging behaviour of honeybees [7]. The algorithm represents solutions in the given multi-dimensional search space as food sources (nectar), and maintains a population of three types of bee (employed, onlooker, and scout) to search for the best food source. Comparative studies [8, 10] have indicated that ABC performance is competitive with other population-based algorithms such as PSO, Genetic Algorithms (GA) and Differential Evolution (DE).

The general idea of the ABC is that it starts with random solutions and repeatedly attempts to find better solutions by searching the neighbourhoods of the current best solutions and abandoning the poor solutions. The current problem solutions are represented as food sources that are each associated with an employed bee. An equal number of onlooker bees each choose one of those food sources to be exploited based on their quality or fitness, using standard roulette wheel selection [6]. Both onlooker and employed bees continuously try to locate better food sources in the neighbourhood of their current food source by changing a randomly chosen dimension of their food source position (i.e., a randomly chosen parameter of their solution) by a random amount in the direction of another randomly chosen food source. Specifically, at each stage, a randomly chosen parameter x_i of food source i is updated by $r.(x_i - x_j)$ where r is a random number drawn uniformly from the range $[-1, 1]$, and x_j is the corresponding parameter of a different randomly chosen food source j [11]. If that update results in a better solution, the existing food source is replaced by the one at the updated position. Meanwhile, scout bees carry out global exploration of the search space by randomly choosing new food sources to initialize the algorithm, and to replace food sources that have been deemed exhausted because they have failed too many times to lead to improvements.

It is clear from the above description that the standard ABC algorithm has only three control parameters that need to be set appropriately for the given problem. First, the bee colony size, equal to twice the number of food sources, and effectively equivalent to an EA population size. Second, the local search abandoning limit. Third, the maximum number of search cycles, that is equivalent to an EA number of generations, and can be defined indirectly by a suitably chosen fitness-based termination criterion.

3 Training Neural Networks using ABC

Applying the ABC algorithm to training neural networks is relatively straightforward. The multi-dimensional search space is the space of network connection weights and neuron thresholds, and the fitness is a standard measure of network output performance (such as sum-squared error or cross entropy) on the training data. However, the main objective here is for the trained network to generalize to perform well on previously unseen testing data, and it is well known that learning the training data too precisely can lead to "over-fitting" and unnecessarily poor generalization performance [1]. With gradient descent training, such as BP, that is typically avoided by "stopping training early", or by adding a regularization term to the cost function (such as "weight decay"), and optimizing those with reference to an independent validation dataset [1]. In principle, similar approaches can be applied to optimize the ABC training, though that does not appear to have been done in the previous studies.

Karaboga and Ozturk [11] have already tested the ABC approach on nine PROBEN1 benchmark classification problems [13], and compared their results with those they obtained using two traditional neural network learning algorithms (BP and Levenberg-Marquardt) and three population based algorithms (PSO, GA and DE). Overall, the ABC achieved good results. More recently, further improved results have been obtained with hybrid learning algorithms involving ABC combined with traditional neural network training algorithms [12, 14]. The key question to be addressed in this paper is: how can these good ABC results be reconciled with the earlier negative results that Cantu-Paz and Kamath obtained for the closely related population-based EAs [4]?

For the purpose of fair comparison, we shall follow as closely as possible the approaches used in the previous studies in this area. As with the earlier comparative study of using EAs for NN training [4], the ABC algorithm will be compared here with standard BP. Following the earlier study of using ABC for NN training [11], we shall concentrate on standard fully connected feed-forward classification neural networks with one hidden layer and use sigmoidal hidden and output activation functions. Sum squared error will again be used as the training cost function, a simple winner-take-all approach will be used to determine the predicted output classes during testing, and performance will be given as percentage correct scores.

An important issue when comparing learning algorithms is that many of the standard benchmark datasets in the UCI Machine Learning Repository [2] are actually trivial in the sense that even the simplest low complexity $O(nd)$ algorithms do not perform significantly worse on them than more sophisticated algorithms [5]. In fact, four of the nine datasets used in the Karaboga and Ozturk study [11] are trivial in that sense (Cancer, Card, Diabetes and Glass) [5], so we shall not consider them any further. They will be replaced by the more challeng-

Table 1. Neural network architectures, numbers of weights, and training, validation and testing dataset sizes.

Dataset	Architecture	Weights	Train	Valid.	Test
Thyroid	21-6-3	153	3600	1800	1800
Heart	35-6-2	230	460	230	230
Horse	58-6-3	375	182	91	91
Soybean	82-6-9	631	342	171	170
Gene	120-6-3	747	1588	794	793
Digits	64-40-10	3010	3058	765	1797

ing Optical Digits dataset that has 64 inputs representing pixelated images and 10 output classes for the digits 0 to 9, with 3823 training patterns and 1797 for testing [2]. The same network architectures were used as in the Karaboga and Ozturk study [11] for their five remaining datasets. For the new Optical Digits set, 6 hidden units was nowhere near enough, so 40 were used. Table 1 summarizes the properties of the six datasets studied, showing the corresponding network architectures, numbers of weights, and dataset sizes.

Throughout this study we shall use standard unpaired two-tailed *t tests* to determine the statistical significances of any performance differences found. Such tests on the Karaboga and Ozturk [11] results (repeated in Table 2) for each of their five datasets indicate that BP is significantly better ($p < 0.001$) than ABC on one (Gene), significantly worse ($p < 0.001$) on three (Heart, Soybean, Thyroid), and not significantly different ($p > 0.1$) on one (Horse). A potential problem with these results, however, is that the performance of both algorithms appear surprisingly poor, particularly for the Thyroid and Soybean datasets, so the following sections will attempt to optimize the performance of each algorithm, and repeat the comparisons using the improved results.

4 Training Neural Networks using Optimized BP

A common problem with all comparisons against BP is that it is very easy for BP to perform poorly on the chosen datasets if its learning parameters are not optimized well, and that can be difficult to do by hand, because the parameters are not independent, and the best values depend on the properties of the given dataset. The study of Karaboga and Ozturk [11] simply used the same learning parameters for all nine datasets, and it is likely that they were far from optimal for at least some of them. One solution would be to use an evolutionary algorithm to optimize the key BP learning parameters, such as the random initial weight range $[-\rho, \rho]$ and the learning rate η. With a fixed, sufficiently large, number of training epochs for each problem, the evolved learning rate is then able to implement a form of early stopping and avoid over-fitting, and that consistently leads to improved performances [3]. However, this evolutionary approach tends to be rather computationally intensive, and might be regarded as giving BP an unfair advantage over ABC. Instead, we can abstract a consistent emergent property

Table 2. Mean neural network Classification Error Percentages (CEP) and standard deviations (s.d.) for the six datasets using: BP from [11], ABC from [11], Optimized BP, Optimized ABC, and Optimized Unconstrained ABC.

Dataset		BP [11]	ABC [11]	Opt. BP	Opt. ABC	Opt. UABC
Thyroid	CEP	7.26	6.95	2.06	6.14	1.87
	s.d.	0.00	0.01	0.21	0.07	0.14
Heart	CEP	21.44	19.48	19.43	19.13	19.49
	s.d.	0.55	1.41	0.54	1.34	0.57
Horse	CEP	27.84	28.63	28.43	27.69	27.14
	s.d.	2.12	2.61	2.70	1.23	1.69
Soybean	CEP	61.16	38.63	10.08	13.93	9.91
	s.d.	19.18	3.18	1.98	1.13	1.04
Gene	CEP	11.37	29.50	13.23	19.55	12.22
	s.d.	1.15	1.88	0.57	0.71	0.52
Digits	CEP	-	-	4.32	6.29	4.27
	s.d.	-	-	0.27	0.18	0.34

of the evolutionary approach, namely that very small initial weight ranges and very slow learning rates tend to work best, and use a standard stopping early approach to set the number of epochs. The details of the experimental set-up and analysis were then chosen to provide the closest possible match with the ABC approach discussed in the next section.

The datasets were each split into standard training, validation and testing sub-sets (as indicated in Table 1), with the validation set performance used to determine the optimal stopping point for the training on the training set. For each training run, for each dataset, the initial network weights were drawn uniformly from the range [-0.03,0.03] and a maximum of one million epochs of BP training were applied. Clearly, a learning rate for each training dataset was required that consistently resulted in achieving the maximum validation set performance in the allowed number of epochs. These were found by initially trying a learning rate of 0.000001 in each case, and then increasing that by factors of ten till it was large enough, giving 0.000001 for Gene, 0.00001 for Heart and Digits, 0.0001 for Horse, 0.001 for Soybean, and 0.01 for Thyroid. These large differences serve to emphasise again how important it is to set the learning parameters differently and appropriately for each dataset. It is quite likely that the learning could be speeded up in some cases (by using fewer epochs and larger learning rates), but determining by how much would potentially require more computational effort overall for no improvement in performance.

As always, the random factors result in fluctuating performances within and across runs, so there are often no clear optimal stopping points for the training, and it is not obvious that all runs should be selected for use in computing the average test set performances. A number of valid model selection approaches were possible, but it made best sense to choose an approach to averaging that most closely matched the natural averaging approach for the ABC. An average test set performance was therefore determined using the network weights corresponding to the top ten validation set performances from five BP runs. This was then repeated ten times to give an indication of the variance of the results. These results are presented in the "Opt. BP" column of Table 2 for comparison with the corresponding results from the earlier study [11]. With the optimized parameter values, BP is now significantly better ($p < 0.001$) than ABC on three of the datasets (Thyroid, Soybean, Gene), and not significantly different ($p > 0.1$) on the other two (Heart, Horse), despite the fact that BP has been trained on less data (i.e., not on the subset of the full training data set that was kept aside to be the validation set). So, at this stage, the empirical results show that ABC is significantly worse than BP for training neural networks.

5 Training Neural Networks using Optimized ABC

In the same way that non-optimized learning parameter values resulted in misleadingly poor BP results, it may be that better optimization of the ABC parameters can bring that approach back up to, or even beyond, the performance levels of BP. This is the issue that we address next.

We proceed by investigating how the ABC performance depends on its parameters, and thus determine the best values that will enable a fair comparison against BP. A preliminary investigation revealed that the bee colony size and abandoning limit had very little effect on the results achieved, but the number of search cycles was extremely important. This is not surprising, since the ABC will obviously be prone to under- and over-fitting in exactly the same way as gradient descent algorithms such as BP, and stopping training early (at an optimum point determined by performance on a validation set) can be expected to lead to improved generalization performance on the test set. The way to get the best generalization results is therefore to apply the ABC algorithm for enough cycles that over-fitting has clearly begun, and then go back and take the solutions (i.e. network weights) corresponding to the best validation set performances to be the ones to represent the Optimized ABC. As with the above averaging approach for BP, we take the average test set performance over the ten sets of weights corresponding to the ten best validation performances from each ABC run, and repeat that ten times to estimate the variances. The use of five BP runs to give the ten best sets of BP weights can now be seen as providing a reasonable approximation to picking the best weights from whole bee colonies.

For neural network training using ABC, there is another crucial parameter that can have a big effect on the results, namely the size of the search space, which here corresponds to the limit on the network weights. It is known that optimizing the initial random weight range for BP can have a big effect on the generalization performance [3], so it is not surprising that it also has a big effect for ABC too. The obvious way to proceed is to start with the default ABC colony size of 30 and abandoning limit of 1000 used by Karaboga and Ozturk [11], but to train for a range of search space limits to find the best for each dataset.

Figure 1 shows how the performance varies with search space size, i.e. the weight range $[-\rho, \rho]$ used to generate the initial solutions and to limit the weights

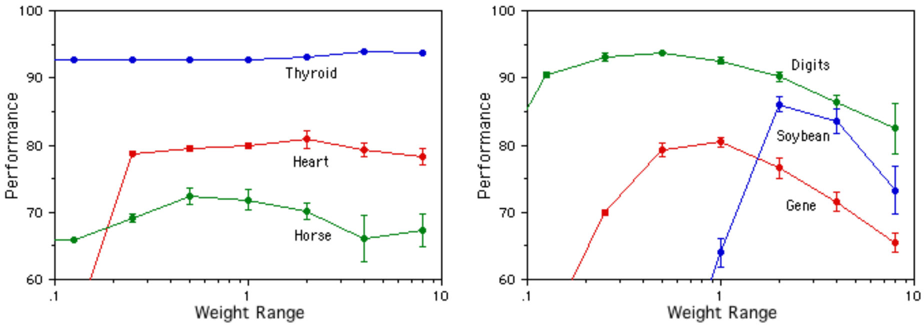

Fig. 1. Generalization performance as a function of weight range for the ABC training algorithm with limited random initial weight range, and the same limited weight range throughout training.

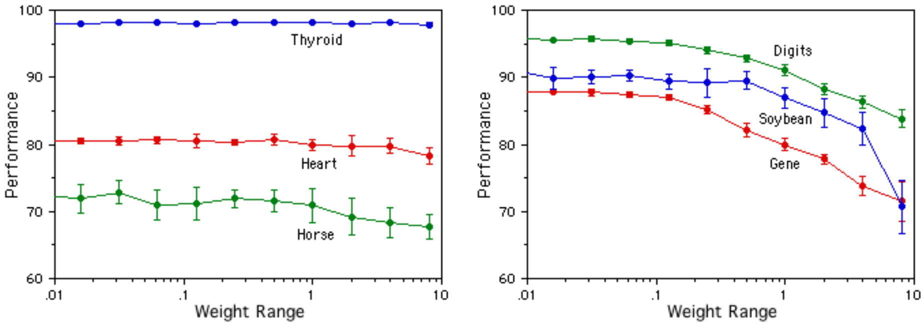

Fig. 2. Generalization performance as a function of initial weight range for the ABC training algorithm with limited random initial weight range, but unconstrained weights at later stages of training.

throughout training. There is inevitable problem dependence, but if the range is too small or too large, the generalization performance deteriorates in each case. The study of Karaboga and Ozturk [11] simply used the same range of $[-2, 2]$ for all the datsets, but that is significantly worse than optimal for four of the six datasets (Thyroid, Horse, Gene, Digits), and not significantly different for the other two (Heart, Soybean). The performances of the optimal data points from Figure 1 are shown in the “Opt. ABC” column of Table 2, and despite the reduced amount of training data caused by excluding the validation set, no datasets have reduced performance compared with the original study. However, even with the optimized weight ranges, ABC is still significantly worse ($p < 0.01$) than BP on four data sets (Thyroid, Soybean, Gene, Digits), and not significantly different ($p > 0.1$) on the other two (Heart, Horse).

The general pattern found for BP initial weight ranges is that smaller values tend to result in better generalization until a point is reached when any further reductions make little difference. The problem the ABC approach has is that

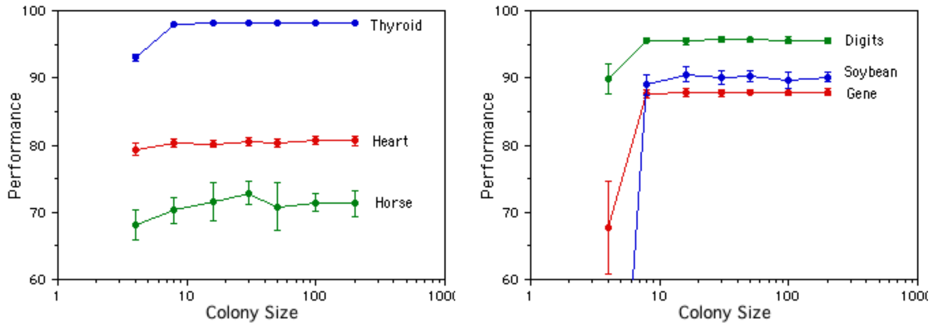

Fig. 3. Generalization performance as a function of the bee colony size for the UABC training algorithm with optimal initial weight range and abandoning limit of 1000.

smaller values will lead to an over-restricted search space if the weights are constrained to stay within that range throughout training. However, it is a simple variation of the standard ABC (that we shall call Unconstrained ABC, or UABC) to define an initial weight range, but allow the ABC algorithm to take the weights outside that range. Doing that leads to the improved pattern of performances shown in Figure 2. Now the performance is fairly level for small weight ranges, and the range $[-0.03, 0.03]$, that we used for the BP runs, is small enough to work well for all the datasets. Smaller values tend to increase the number of training cycles without significant performance improvement, so there is no point in using a smaller range. The optimized performances using this approach and initial weight range are given in the "Opt. UABC" column of Table 2. This shows significant performance improvement ($p < 0.01$) over the restricted weight range approach (in the "Opt. ABC" column) for four of the datasets (Thyroid, Soybean, Gene, Digits), and no significant difference ($p > 0.1$) for the other two (Heart, Horse). Comparing the optimized UABC results with the optimized BP results shows no significant difference ($p > 0.1$) for five of the six datasets (Thyroid, Heart, Horse, Soybean, Digits), but UABC is now significantly better ($p < 0.01$) than BP for the Gene dataset.

It was stated above that the bee colony size and abandoning limit had little effect on the results obtained by ABC for neural network training, but we now need to check that claim more carefully, in case their optimization can lead to further improvements in performance. First, Figure 3 confirms that, as long as the colony size does not fall below about 10, it makes no significant difference to the final performance what the colony size is. Obviously, larger colonies will inevitably result in longer compute times per cycle, and that tends to not be fully compensated by a reduction in the number of cycles required, so there is an overall advantage in keeping the colony size reasonably low. The default size of 30 used above is well within the range of good values, but not so high as to have serious adverse computational resource implications.

The effect of varying the abandoning limit is shown in Figure 4. As long as it is not below about 30, it makes no significant difference what the limit is. In

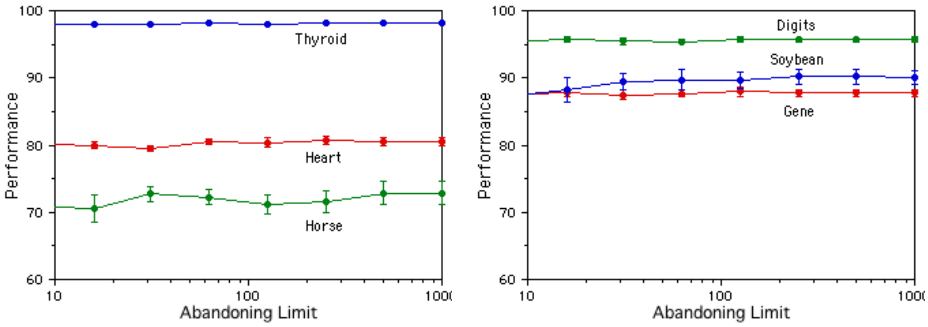

Fig. 4. Generalization performance as a function of the abandoning limit for the UABC training algorithm with optimal initial weight range and bee colony size of 30.

fact, for the default limit of 1000, or more, the scout bees are virtually never employed, and that has no adverse effect on performance.

Thus, we have now fully optimized the ABC algorithm parameters, and the results shown in Table 2 are the best possible without further modification of the algorithm itself. The ABC has achieved neural network generalization performance significantly better than BP on the Gene dataset, but the results for the other five datasets studied are not significantly different to those obtained using standard BP with appropriate learning parameter values.

6 Conclusions and Discussion

This paper has investigated the use of the ABC algorithm for training neural networks, and shown how it can be optimized to give better results than those found in previous studies. However, in most cases, the best ABC generalization performance levels obtained are not significantly different to standard BP that has been properly optimized for the given problems.

One could argue that ABC algorithms are relatively minor extensions of standard EAs: they both involve populations of solutions, the generation of new solutions based on existing solutions, and the discovery of better solutions by iteratively using fitness based selection to determine which "offspring" should replace which existing solutions. The obvious question to ask, then, is whether the offspring generation and selection inspired by bees perform any better on the application of interest (i.e. neural network training) than those inspired by evolution by natural selection. We have seen that the scout bee component of the ABC algorithm is redundant in this case, in that no decrease in performance results from setting the abandoning limit to values so high that the scout bees never become involved after the initial solution set generation. Thus there is effectively no further wide-scale random exploration of the search space during training. This means that all the offspring are generated by changing the value of a single randomly chosen parameter (i.e. network weight) by an amount that depends on the difference between that value and the corresponding value of another individual. That is exactly how a basic EA cross-over and mutation would optimize its genotype [3], so it is not surprising that we have come to a similar conclusion to that of the earlier study of Cantu-Paz and Kamath [4] which showed that weight optimization using EAs gave results that were not significantly better than standard BP.

This paper has shown that the optimized ABC and BP results are not significantly different for five of the six datasets studied, but we are still left with the question of how the ABC performs significantly better than BP on the Gene dataset. With BP learning, the weight update sizes depend on the back-propagated output errors and the chosen value of the learning rate parameter. With ABC optimization, the potential weight update sizes depend on the weight differences between individuals, and that means the algorithm can effectively generate its own learning rates for each weight during training [11]. For example, about half way through training on the Gene dataset, the mean standard deviation across individuals of the input to hidden unit weights is around 0.05, while that of the hidden to output weights is around 3.3. This means that there will be something like a factor of 66 difference in the average effective learning rates for the two sets of weights. Similar large differences in BP learning rates across network components have been found in evolutionary neural network studies to lead to significant improvements in performance for some datasets [3], so it is a reasonable hypothesis that this is why ABC is performing better here than BP with a single learning rate throughout the network.

Another important issue is the increased computational cost of using ABC rather than BP. With ABC updating random network weights one at a time, by amounts involving a random factor, it will inevitably become less computationally efficient as the network sizes increase. Of course, BP also becomes more computationally costly as the network size grows, but to a much lesser extent than ABC. This differing dependence on network size makes fair comparisons of the two approaches difficult, because past empirical results have shown that the generalization performance usually improves with more hidden units, as long as appropriate regularization (such as stopping early) is used (e.g., [3]), and using much larger networks than the current study will not only pose problems with getting the experiments completed in a reasonable time, but will also put ABC at a considerable compute time disadvantage compared with BP. If equal fixed maximum compute times were enforced for both algorithms, it is quite likely that BP would end up being able to use significantly more hidden units, and thus achieve better generalization performances than the ABC in that way.

There clearly remains considerable scope for future work in this area, but, unfortunately, most of it will be extremely computationally expensive. First, of course, the investigation of a wider range of datasets, with many more runs per dataset, will provide a more reliable indication of the patterns of results that can be expected more generally. Then, the application of an evolutionary approach to the optimization of the BP learning parameters, including the evolution of different learning rates for different layers of weights, should allow closer to optimal BP performance than is feasible with parameters set “by hand” [3], and also allow testing of the above hypothesis concerning the superior ABC performance on the Gene dataset. Finally, testing how the generalization performances and run times depend on the number of neural network hidden units will address the computational cost issue noted above. Ultimately, it will be the results of this future work that will determine whether the ABC is a worthwhile algorithm for training neural networks.

References

1. Bishop, C.M.: Neural Networks for Pattern Recognition. Oxford, UK: Oxford University Press (1995)
2. Blake, C.L., Merz, C.J.: UCI Repository of Machine Learning Databases. University of California, `http://www.ics.uci.edu/~mlearn/MLRepository.html` (1998)
3. Bullinaria, J.A.: Using evolution to improve neural network learning: Pitfalls and solutions. Neural Computing and Applications **16** 209-226 (2007)
4. Cantu-Paz, E., Kamath, C.: An empirical comparison of combinations of evolutionary algorithms and neural networks for classification problems. IEEE Transactions on Systems, Man, and Cybernetics-Part B: Cybernetics **35** 915-927 (2005)
5. Duch W., Maszczyk T., Jankowski N.: Make it cheap: learning with O(nd) complexity. Proceedings of the World Congress on Computational Intelligence, 132-135 (2012)
6. Engelbrecht, A.P.: Computational Intelligence: An Introduction. Sussex, UK: Wiley. (2007)
7. Karaboga, D.: An idea based on honey bee swarm for numerical optimization. Technical Report TR06, Computer Engineering Department, Erciyes University, Turkey (2005)
8. Karaboga, D., Akay, B.: A comparative study of Artificial Bee Colony algorithm. Applied Mathematics and Computation **214** 108-132 (2009)
9. Karaboga, D., Akay, B., Ozturk, C.: Artificial Bee Colony (ABC) optimization algorithm for training feed-forward neural networks. Proceedings of the 4th International Conference on Modeling Decisions for Artificial Intelligence, 318-329 (2007)
10. Karaboga, D., Basturk, B.: On the performance of Artificial Bee Colony (ABC) algorithm. Applied Soft Computing **8** 687-697 (2008)
11. Karaboga, D., Ozturk, C.: Neural networks training by Artificial Bee Colony algorithm on pattern classification. Neural Network World **19** 279-292 (2009)
12. Ozturk, C., Karaboga, D.: Hybrid Artificial Bee Colony algorithm for neural network training. Proceedings of the IEEE Congress on Evolutionary Computation, 84-88 (2011)
13. Prechelt, L.: PROBEN1 – A set of benchmarks and benchmarking rules for neural network training algorithms. Technical Report 21/94, Universitat Karlsruhe, Fakult at fur Informatik, Germany (1994)
14. Qiongshuai, L., Shiqing , W.: A hybrid model of neural network and classification in wine. Proceedings of the 3rd International Conference on Computer Research and Development, 58-61 (2011)