A copy-edited version of this paper has been published in the journal Memetic Computing, 2014, DOI: 10.1007/s12293-014-0137-7

Artificial Bee Colony Training of Neural Networks: Comparison with Back-Propagation

John A. Bullinaria and Khulood AlYahya

School of Computer Science
University of Birmingham
Birmingham, B15 2TT, UK

j.a.bullinaria@cs.bham.ac.uk

Phone: +44 (0) 121 414 2590

Abstract: The Artificial Bee Colony (ABC) is a swarm intelligence algorithm for optimization that has previously been applied to the training of neural networks. This paper examines 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 for benchmark problems demonstrate 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, 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. If different, evolutionary optimized, BP learning rates are allowed for the two layers of the neural network, BP is significantly better than the ABC on two of the six datasets, and not significantly different on the other four.

Keywords: Artificial Bee Colony, Neural Networks, Learning, Evolution.

1. Introduction

The study of different insect behaviours, animal colonies and swarms has led to the development 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 [10] and inspired by the foraging behaviour of honeybees [14]. There are many potential applications of the 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 [12, 15], however, have previously applied the ABC to neural network learning and claimed some success. The aim of this paper is to explore more carefully how effective the ABC really is for training feed-forward neural networks to perform classification tasks.

The remainder of this paper is organized as follows: The next two sections describe the ABC algorithm and how it can be applied to neural network training. Then a series of computational experiments are presented that explore the power of the standard and improved ABC for neural network applications in comparison with standard optimized BP. Further experiments with evolutionary optimized BP then demonstrate that the best ABC results are worse than can be achieved with BP. The paper ends with some conclusions and discussion.

2. The Standard Artificial Bee Colony Algorithm

The ABC algorithm is a stochastic optimization algorithm inspired by the foraging behaviour of honeybees [10, 14]. 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 (solution). Comparative studies [11, 13] have indicated that the 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 begins with random solutions and repeatedly attempts to find better solutions by searching the neighbourhoods of the current best solutions and abandoning unpromising solutions. The problem solutions at each stage 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 [15]. 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 an improvement.

It follows from the above specification that the standard ABC algorithm has only three control parameters that need to be set appropriately for each 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, which can be defined indirectly by a suitably chosen fitness-based termination criterion.

3. Neural Network Training using the 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 comes from 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 the 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 [15] have carried out the most comprehensive study so far, testing the ABC approach to neural network training on nine PROBEN1 benchmark classification problems [20], and comparing the 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, their ABC training achieved good results. Similar success with ABC trained neural networks

has also been claimed by numerous other authors [16, 17, 18, 22, 23], and further improved results have been obtained with hybrid learning algorithms involving the ABC combined with more traditional neural network training algorithms [9, 19, 21]. 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]?

To facilitate fair comparisons, the approaches used by the previous studies in this area will be followed as closely as possible. 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 the ABC for NN training [15], standard fully connected feed-forward classification neural networks will be used with one hidden layer and sigmoidal hidden and output activation functions. The training cost function will again be sum squared error, a simple winner-take-all approach will be used to determine the predicted output classes during testing, and performance will be computed as simple 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 [15] are trivial in that sense (Cancer, Card, Diabetes and Glass) [5], so those will not be considered any further. They will be replaced by the more challenging Optical Recognition of Handwritten 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 neural network architectures, with 6 hidden units, were used as in the Karaboga and Ozturk study [15] for their five remaining datasets. However, 6 hidden units was nowhere near enough for the new Digits dataset, so 40 were used. The crucial details for the six datasets studied are summarized in Table 1, showing the corresponding network architectures, numbers of weights, and dataset sizes.

Throughout this study, standard unpaired two-tailed *t tests* will be used to determine the statistical significances of any performance differences found. Using this test on the Karaboga and Ozturk [15] results (repeated in Table 2) for each of their five datasets indicate that BP is significantly better ($p < 0.001$) than the 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 reported 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. Neural Network Training 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 [15] 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 is to use an evolutionary algorithm to optimize the key BP learning parameters, such as the random initial weight range $[-\rho, \rho]$ and 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, such evolutionary approaches tend to be rather computationally intensive, and might be regarded as giving BP an unfair advantage over the ABC. A standard non-evolutionary approach will therefore be studied first, but using information that consistently emerges from evolutionary investigations [3], namely that very small initial weight ranges and very slow learning rates tend to work best, with a standard stopping early approach to set the number of epochs. The details of the experimental set-up and analysis were 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 emphasize 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 lead to 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. The generalization performance was therefore taken to be the average of ten individual neural network test set performances, where the ten sets of network weights were those that produced the top ten validation set performances from five BP training runs sampled every 100 epochs of training. This computation was repeated ten times to give

an indication of the mean and variance of the performance across independent sets of runs. These results are presented in the "Opt. BP" column of Table 2 for comparison with the corresponding results from the earlier study [15]. With the optimized parameter values, BP is now significantly better ($p < 0.001$) than the 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 the ABC is significantly worse than BP for training neural networks.

5. Neural Network Training using the 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. It is this possibility that will be addressed next.

The obvious way to proceed is by investigating how the ABC performance depends on its parameters, and thereby determining the best parameter values to enable a fair comparison against BP. A preliminary investigation indicated 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, given that 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 the 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. In line with the averaging approach for BP used earlier, the generalization performance here was taken to be the average test set performance over ten networks using the sets of weights corresponding to the ten best validation performances from each ABC run, sampled every 100 search cycles, and that was repeated ten times to provide an estimate of the mean and variance of these results across ABC runs. The earlier 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 the 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 the ABC too. This can easily be tested by starting with the default ABC colony size of 30 and abandoning limit of 1000 used by Karaboga and Ozturk [15], and applying ABC training with a wide

range of different search space limits to find the best for each dataset.

Figure 1 shows how the performance varies with the search space size, i.e. the weight range $[-\rho, \rho]$ used to generate the initial solutions and to limit the weights 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 [15] simply used the same range of [–2, 2] for all the datasets, 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 and early stopping points, the 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 pattern of results found quite generally for BP initial weight ranges is that using smaller values tends to result in better generalization, until a point is reached when any further reductions make little difference. The problem that the ABC approach has here is that smaller values can also lead to an over-restricted search space if the weights are constrained to stay within that range throughout training. However, there is an alternative version of the ABC (that will be referred to here as Unconstrained ABC, or UABC) that still defines an initial weight range, but allows 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 generalization is fairly level for small weight ranges, as with BP, and the range [–0.03, 0.03], that was 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 nothing to be gained by 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 the UABC is now significantly better ($p < 0.01$) than BP for the Gene dataset.

It was noted above that the bee colony size and abandoning limit had little effect on the results obtained by the ABC for neural network training, but this now needs to be checked 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 be able to explore more

solutions per cycle, inevitably resulting in longer compute times per cycle, but that tends to not be fully compensated by a reduction in the number of cycles required to reach the best solutions, 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. Even for very low abandoning limits, there is little degradation in performance. For neural network training, finding better solutions appears to be quite common compared to other ABC applications, so reasonably high abandoning limits are rarely reached. In fact, the default limit of 1000 is only ever reached for one of the six datasets (Soybean). For lower limits, what tends to happen is that as the training approaches the local maximum of the fitness function, it becomes harder to find an improved solution, and so the current solution is abandoned and the search effectively starts again from scratch. That may happen before the peak in validation set performance, so to be sure of not loosing the best solutions by abandoning the search too soon, a form of elitism needs to be employed, whereby the best solution on the validation set at each cycle is immune from abandoning. It is that elitism which hides the underlying differences in Figure 4.

If one looks at the details of the ABC training runs, the value of the abandoning limit is actually seen to make a big difference. Figure 5 shows the individual validation set performances at each stage of a typical ABC run for the Soybean dataset. The upper two graphs are for the default abandoning limit of 1000, with elitism (left graph) and without elitism (right graph). Initially there is no difference between the two cases, because the abandoning limit is never reached, and later on the abandoned solutions are sufficiently few and late that good results are achieved whether or not elitism is used. For abandoning limits that are much lower, such as the case of 16 shown in the lower two graphs, a very different pattern emerges. Without elitism (right graph), abandoning solutions too early prevents any good solutions from emerging at all. With elitism, a single good solution emerges early on and is refined throughout, with the others never managing to compete with it. Thus, despite the apparent lack of differences seen in Figure 4, keeping the abandoning limit high (at or above the default value of 1000), and the scout bees virtually never employed, is the way to make best use of the whole bee colony. This analysis also clarifies why the bee colony size makes so little difference in Figure 3. All that is required is one good solution and a few others to drive essentially random potential weight updates, so maintaining large colonies offers little advantage.

Thus, the ABC algorithm parameters are now fully understood and optimized, and the results shown in Table 2 are confirmed as 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.

This leads to the obvious next question: what is it that allows the ABC to perform significantly better than BP on the Gene dataset, but not on the other datasets? For 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 across the current set of solutions, which is determined at each stage of training by the weight distributions for each layer of the network. This means the ABC algorithm will effectively generate its own learning rates for each network layer. Standard BP has a single fixed learning rate throughout the whole network, but there is nothing to prevent having different BP learning rates for the distinct network components (i.e. layers of weights and thresholds). The problem, in practice, with trying to make good use of such differences is that finding optimal values for a whole set of BP learning rates, that interact with all the other network details, is extremely difficult to do "by hand". When automatic optimization of such a framework has been carried out using evolutionary computation techniques, large differences in BP learning rates have emerged across the network components, leading to significant improvements in performance for some datasets [3]. It is therefore a reasonable hypothesis that this is the factor that allows the ABC to perform better than BP on the Gene dataset. The next section will test that hypothesis.

6. Neural Network Training using Evolved BP

As noted above, simulated evolution provides a reliable approach for optimizing BP learning parameters [3]. This section will use that technique to search for further improvements beyond the BP parameter values used earlier. The evolutionary approach will first be applied to optimize the single learning rate case, and the results from that will be compared with those from the above "optimization by hand" approach. If the two approaches produce neural networks with optimal performances that do not differ significantly for any of the datasets studied, that will provide further confidence that the non-evolutionary BP and ABC optimization processes already employed have been appropriate, and also that the adopted evolutionary approach is likely to result in optimal solutions when it is then applied to the more difficult-to-optimize two-learning-rate case.

A simple EA was initially set up to optimize the two key neural network BP learning parameters, namely the random initial weight range $[-\rho, \rho]$, and the single learning rate η [3]. With a fixed, sufficiently large, number of training epochs specified for each dataset, under-fitting and over-fitting of the training data can be avoided by the learning rate η evolving so that the training ends near the optimal point. In this case, the validation set performance is not needed to determine an early stopping point, but is instead used to provide the fitness to drive the evolutionary selection processes. To maintain fairness of the comparisons, no additional learning factors (such as momentum or weight decay) were included, that might also be evolved and lead to further improved performance, but it will

be straightforward to introduce them into this framework in the future [3]. The number of hidden units was not evolved because that invariably results in using the maximum allowed number [3], and, again to ensure fair comparisons, it was appropriate to keep that parameter the same as for the ABC algorithm anyway. The same training, validation and testing datasets were used as in the earlier non-evolutionary approaches.

The EA follows a fairly standard approach that has proved successful in the past [3]. It maintains a population of individual neural networks, each with a genotype representing their evolvable parameters. In the initial case, just ρ and η are evolved, but there will generally be many more evolving parameters than that. The evolution starts from an initial population with random genotype parameter values, and for each new generation, each neural network has random initial weights drawn from its innate range $[-\rho, \rho]$, learns from the training data using its innate learning rate η, and has its fitness determined using the validation set. The fittest half of the population are then copied into the next generation, and also randomly select a partner to produce one child, thus maintaining the population size. The offspring inherit innate characteristics (i.e., genotype parameter values) drawn randomly from within the corresponding ranges spanned by their two parents, with random Gaussian mutations added to allow values outside the parental ranges. For each new generation, all the networks, both copies and offspring, start their learning from newly drawn random initial weights, so there is no learned information carried from one generation to the next. The evolutionary process continues for sufficient generations that no further improvements are evident.

Clearly, having to train whole populations of neural networks over many generations involves a massive computational cost. Consequently, it was not feasible to train each network for one million BP epochs as in the earlier non-evolutionary approach, nor have massive population sizes. Fortunately, just 1000 epochs proved to be sufficient for training in all cases, except the relatively small Thyroid networks which were previously found to be slower to learn and required 100,000 epochs. That still led to slow progress for the relatively large Digits networks, so only 200 epochs were used for those, which proved to be enough. Moreover, relatively small population sizes of only 50 proved sufficient to maintain a reasonable population diversity in all cases.

With only two parameters to evolve in the initial case, the evolutionary runs were very consistent, and all settled within 1000 generations. Simply taking the ten best individuals on the validation set from the final generations of four evolutionary runs for each dataset was sufficient to provide ten independent trained Evolved BP networks, which were evaluated on the previously unseen testing data to give the final generalization performance for that dataset. The results are presented in the "Evo. BP" column of Table 3, along with the corresponding non-evolutionary optimized UABC and BP results from Table 2.

For all six datasets, the performances obtained using the Evolved BP learning parameters are not significantly different ($p > 0.08$) to those arising from the non-evolutionary optimized BP. This

complete lack of differences suggests that the earlier BP runs have been successfully optimized "by hand", and that the evolved performances have not been compromised by using many fewer epochs of training. It is also consistent with the absence of improvements found in the study of Cantu-Paz and Kamath [4].

The key question here is whether allowing and evolving more than one BP learning rate can lead to improved performance. Using the previously described evolutionary approach to optimize the initial weight range ρ and *two* BP learning rates η, one for each layer of weights, leads to the generalization performances shown in the "Evo. 2lr BP" column of Table 3. Compared with evolving a standard single learning rate for the whole network, these results are significantly better ($p < 0.01$) for two datasets (Gene and Digits), marginally better ($0.01 < p < 0.05$) for two datasets (Thyroid and Horse), and not significantly different ($p > 0.4$) for the remaining two datasets (Heart and Soybean). This finding is consistent with earlier evolutionary studies which have shown that evolving two or more BP learning rates can sometimes lead to improved performances over standard BP learning [3].

The crucial comparison is between these improved BP results and the optimized ABC results. Now BP is significantly better ($p < 0.01$) than the ABC for two datasets (Thyroid and Digits), and not significantly different ($p > 0.06$) for the other four (Heart, Horse, Soybean and Gene). This is consistent with the above hypothesis that the ABC's advantage over standard BP for the Gene dataset comes from its ability to have different learning rates for the two weight layers. The fact that evolving optimally different BP learning rates for the two weight layers not only removes that advantage, but also gives BP an advantage over the ABC for two other datasets, suggests that the ABC is unable to control its learning rates as effectively as the evolution can for BP.

Obviously, the evolutionary process is computationally expensive, and it would be useful to know if any general patterns emerge that could be used for future datasets without the need for full evolutionary runs. Unfortunately, Table 4 shows that there is a rather large variation in the evolved parameters across the six datasets, with no obvious emergent pattern.

7. 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. The BP parameters were first optimized "by hand" in the same way as the ABC, and then by simulated evolution by natural selection. First, with one standard BP learning rate for the whole network, the by-hand and evolutionary optimization results were not significantly different, with the ABC performing significantly better than BP for one dataset, and not significantly different for the other five. Then with two evolved BP learning rates, one for each network layer, it was found that the results for some

datasets were significantly better than with only one learning rate, and that the BP performances were significantly better than the ABC for two datasets, and not significantly different for the other four.

One could argue that the ABC algorithms are relatively minor extensions of standard EAs in that 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. It has been established in this paper that the scout bee component of the ABC algorithm is redundant in this case, in that no degradation in performance arises 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 a similar conclusion has emerged 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.

A crucial feature of using the ABC for neural network training is that the bee colony is representing a set of solutions that are not converging to a single point like in many other optimization problems, nor even a small number of points. This is likely to be the main reason why the ABC is not performing any better than BP. The individual solutions could, for example, be equivalent neural networks that simply have the order of their hidden units permuted, and this could potentially lead to something like the permutation problem for evolving neural networks, which is known to exist but not pose serious problems in practice [7, 8]. Contrary to previous suggestions [15], the weight changes do not get smaller as the solutions converge, but rather they reflect the distribution of weights across the relevant network components. This is clearly not preventing the ABC algorithm from finding good solutions, but, together with the finding that the scout bees are not making any useful contribution, it does mean that the ABC is actually performing little more than stochastic hill climbing, which one would expect to end up with similar results to an informed hill climbing algorithm like BP, albeit more slowly. It also means that previous claims that the ABC can avoid becoming stuck in local optima better than BP [15, 16, 17, 22] could well prove unfounded too.

This also leads to an important related issue concerning the increased computational cost of using the ABC compared with BP. With the 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 the ABC. This differing dependence on the network size makes fair comparisons of the two approaches difficult, because past empirical studies have shown that the generalization performance usually improves with more hidden units, as long as appropriate regularization (such as stopping early) is used [3]. Getting better results by 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 the ABC at a considerable compute time disadvantage compared with BP. If equal fixed maximum compute times were to be enforced for both algorithms, BP would end up being able to use significantly more hidden units, and thus achieve significantly better generalization performances than the ABC in that way.

For cases requiring evolutionary parameter optimization, which is likely to be necessary whenever two or more BP learning rates are used, the computational cost of training many generations of many individual networks is clearly going to be considerable compared to standard hand-optimized BP and even the ABC. However, the evolution will generally be able to result in faster individual training runs [3], and as long as the total compute times required remain feasible, this may not be a problem in practice. It is also often possible to deduce general patterns concerning the optimal parameter values from earlier evolutionary runs, that can be applied directly to new problems without the need for full evolutionary optimization, but it is clear from the results in Table 4 that much more work will be required before that approach can be effective here.

The overall conclusion of this paper is that there is currently no evidence that the ABC algorithm offers a reliable advantage over BP for training neural networks. Using the optimized ABC to train neural network classifiers leads to generalization performance levels that are significantly better than standard single learning rate BP for only one of the six datasets tested, and not significantly different on the others, and if BP is allowed a different learning rate for each network layer, then it is significantly better than the ABC for two of the six datasets, and not significantly different on the others. All the indications are that the best ABC approach is doing little more than performing stochastic hill climbing, and that will generally be much slower than BP.

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, would provide a more reliable indication of the patterns of results that can be expected more generally. Then, the application of the evolutionary approach to the optimization of even more BP learning parameters may allow even better BP results [3]. Finally, testing how the generalization performances and run times depend on the number of neural network hidden units will allow investigation of the computational cost issues 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, but the current results suggest that it is not.

References

1. Bishop CM (1995) Neural Networks for Pattern Recognition. Oxford, UK: Oxford University Press
2. Blake CL, Merz CJ (1998) UCI Repository of Machine Learning Databases. University of California, http://www.ics.uci.edu/~mlearn/MLRepository.html
3. Bullinaria JA (2007) Using evolution to improve neural network learning: Pitfalls and solutions. Neural Computing and Applications 16:209-226
4. Cantu-Paz E, Kamath C (2005) 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
5. Duch W, Maszczyk T, Jankowski N (2012) Make it cheap: Learning with O(nd) complexity. Proceedings of the World Congress on Computational Intelligence, 132-135
6. Engelbrecht AP (2007) Computational Intelligence: An Introduction. Sussex, UK: Wiley
7. Haidason S, Neville R (2010) Quantifying the severity of the permutation problem in neuro-evolution. Proceedings of 4th International Workshop on Natural Computing (IWNC), 149-156
8. Hancock P (1992) Genetic algorithms and permutation problems: A comparison of recombination operators for neural net structure specification. Proceedings of the International Workshop on Combinations of Genetic Algorithms and Neural Networks, 108-122
9. Irani R, Nasimi R (2011) Application of Artificial Bee Colony-based neural network in bottom hole pressure prediction in underbalanced drilling. Journal of Petroleum Science and Engineering 78:6-12
10. Karaboga D (2005) An idea based on honey bee swarm for numerical optimization. Technical Report TR06, Computer Engineering Department, Erciyes University, Turkey
11. Karaboga D, Akay B (2009) A comparative study of Artificial Bee Colony algorithm. Applied Mathematics and Computation 214:108-132
12. Karaboga D, Akay B, Ozturk C (2007) 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
13. Karaboga D, Basturk B (2008) On the performance of Artificial Bee Colony (ABC) algorithm. Applied Soft Computing 8:687-697
14. Karaboga D, Gorkemli B, Ozturk C, Karaboga N (2012) A comprehensive survey: Artificial Bee Colony (ABC) algorithm and applications. Artificial Intelligence Review 1-37
15. Karaboga D, Ozturk C (2009) Neural networks training by Artificial Bee Colony algorithm on pattern classification. Neural Network World 19:279-292
16. Kurban T, Besdok E (2009) A comparison of RBF neural network training algorithms for inertial sensor based terrain classification. Sensors 9:6312-6329

17. Omkar SN, Senthilnath J (2009) Artificial Bee Colony for classification of acoustic emission signal source. International Journal of Aerospace Innovations 1:129-143
18. Ozkan C, Kisi O, Akay B (2011) Neural networks with Artificial Bee Colony algorithm for modeling daily reference evapotranspiration. Irrigation Science 29:431-441
19. Ozturk C, Karaboga D (2011) Hybrid Artificial Bee Colony algorithm for neural network training. Proceedings of the IEEE Congress on Evolutionary Computation, 84-88
20. Prechelt L (1994) PROBEN1 – A set of benchmarks and benchmarking rules for neural network training algorithms. Technical Report 21/94, Universitat Karlsruhe, Fakult at fur Informatik, Germany
21. Qiongshuai L, Shiqing W (2011) A hybrid model of neural network and classification in wine. Proceedings of the 3rd International Conference on Computer Research and Development, 58-61
22. Shah H, Ghazali R, Nawi NM (2011) Using Artificial Bee Colony algorithm for MLP training on earthquake time series data prediction. Journal of Computing 3:135-142
23. Yeh WC, Hsieh TJ (2012) Artificial Bee Colony algorithm-neural networks for S-system models of biochemical networks approximation. Neural Computing and Applications 21:365-375

Table 1 Neural network architectures (inputs – hidden units – outputs), numbers of weights, and training, validation and testing dataset sizes for each of the six datasets.

Dataset	Architecture	Weights	Training	Validation	Testing
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 – 19	631	342	171	170
Gene	120 – 6 – 3	747	1588	794	793
Digits	64 – 40 – 10	3010	3058	765	1797

Table 2 Mean neural network Classification Error Percentages (CEP) and standard deviations (s.d.) for each of the six datasets using: BP from [15], ABC from [15], Optimized BP, Optimized ABC, and Optimized Unconstrained ABC. All results that are significantly different ($p < 0.01$) to the one on their immediate left are indicated by a *.

Dataset		BP [15]	ABC [15]	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

Table 3 Mean neural network Classification Error Percentages (CEP) and standard deviations (s.d.) for the six datasets using: Optimized Unconstrained ABC, Optimized BP, Evolved BP, and Evolved two learning rates BP. All BP results that are significantly different ($p < 0.01$) to the corresponding UABC result are indicated by a *.

Dataset		Opt. UABC	Opt. BP	Evo. BP	Evo. 2lr BP
Thyroid	CEP	1.87	2.06	1.89	1.62 *
	s.d.	0.14	0.21	0.26	0.16
Heart	CEP	19.49	19.43	20.17	19.74
	s.d.	0.57	0.54	1.14	1.18
Horse	CEP	27.14	28.43	28.35	26.15
	s.d.	1.69	2.70	2.63	1.35
Soybean	CEP	9.91	10.08	9.18	9.18
	s.d.	1.04	1.98	0.84	0.57
Gene	CEP	12.22	13.23 *	13.10 *	11.85
	s.d.	0.52	0.57	0.77	0.80
Digits	CEP	4.27	4.32	4.15	3.55 *
	s.d.	0.34	0.27	0.18	0.39

Table 4 Means (mean) and standard deviations (s.d.) of the evolved initial weight ranges ρ and learning rates η, for each of the six datasets, for standard BP with one learning rate, and enhanced BP with two learning rates.

Dataset		Standard BP		Two learning rates BP		
		ρ	η	ρ	η_{IH}	η_{HO}
Thyroid	mean	0.054	0.22	0.087	0.71	0.038
	s.d.	0.051	0.04	0.053	0.12	0.019
Heart	mean	0.34	1.07	0.23	1.93	0.0061
	s.d.	0.18	0.16	0.11	1.04	0.0036
Horse	mean	0.27	0.12	0.74	0.018	0.00019
	s.d.	0.59	0.24	0.30	0.008	0.00005
Soybean	mean	0.27	0.0092	0.014	0.00056	0.076
	s.d.	0.14	0.0015	0.003	0.00021	0.148
Gene	mean	0.11	0.00015	0.55	0.12	0.000063
	s.d.	0.02	0.00003	0.09	0.05	0.000013
Digits	mean	0.097	0.0079	0.019	0.00018	0.14
	s.d.	0.030	0.0027	0.005	0.00006	0.03

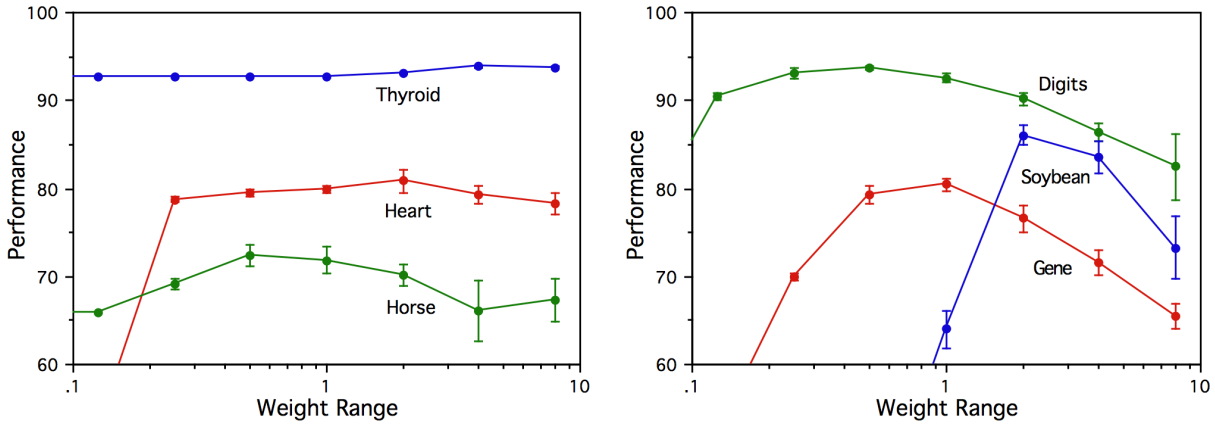

Fig. 1 Generalization performance as a function of weight range for the ABC trained neural networks with limited random initial weight range, and the same limited weight range throughout training.

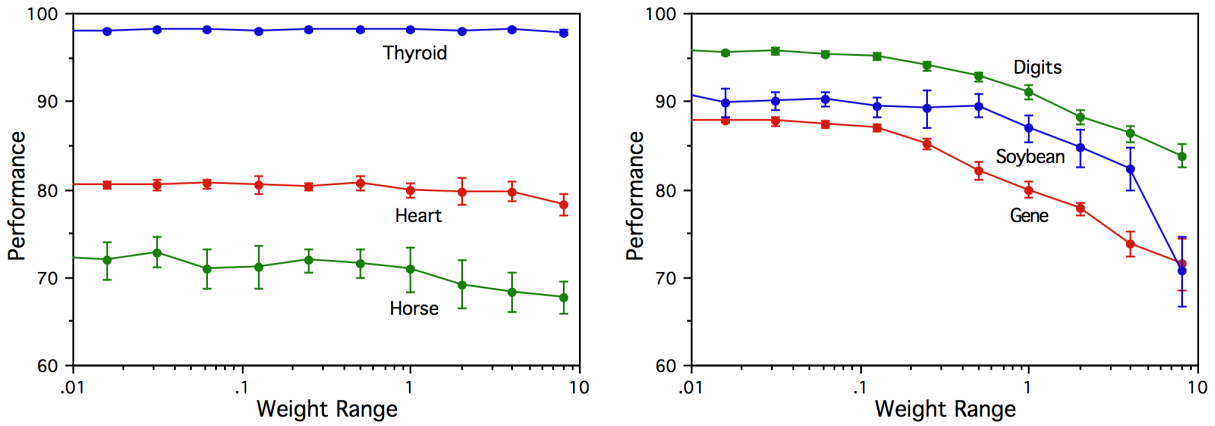

Fig. 2 Generalization performance as a function of initial weight range for the ABC trained neural networks with limited random initial weight range, but unconstrained weights at later stages of training.

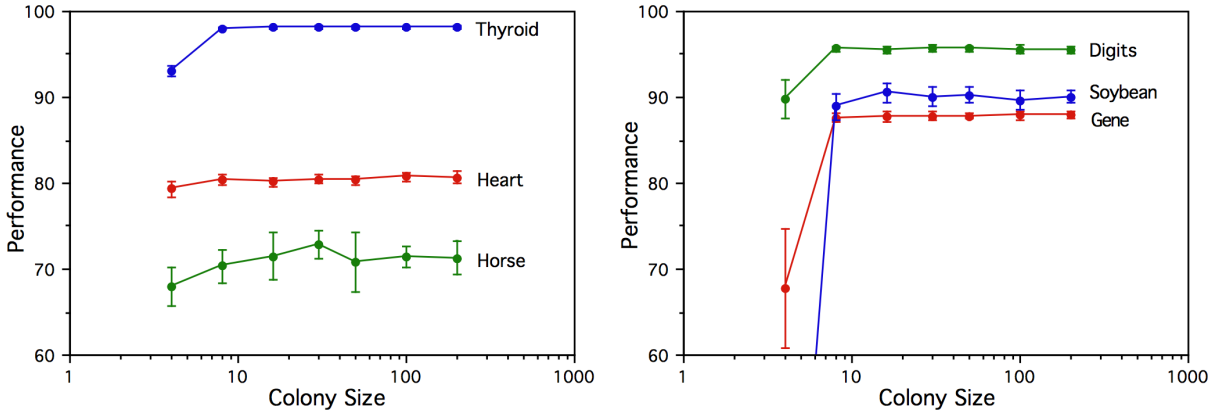

Fig. 3 Generalization performance as a function of the bee colony size for the UABC trained neural networks with optimal initial weight range and abandoning limit of 1000.

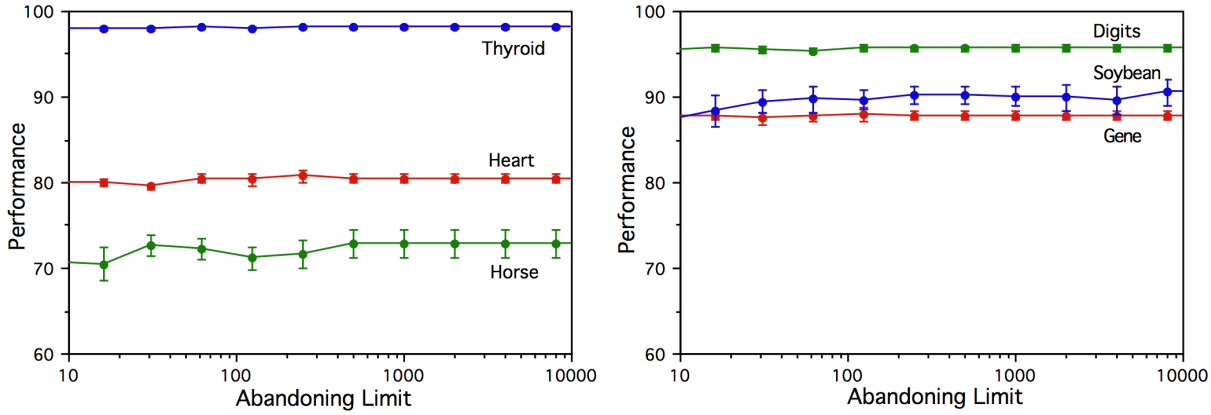

Fig. 4 Generalization performance as a function of the abandoning limit for the UABC trained neural networks with optimal initial weight range and bee colony size of 30.

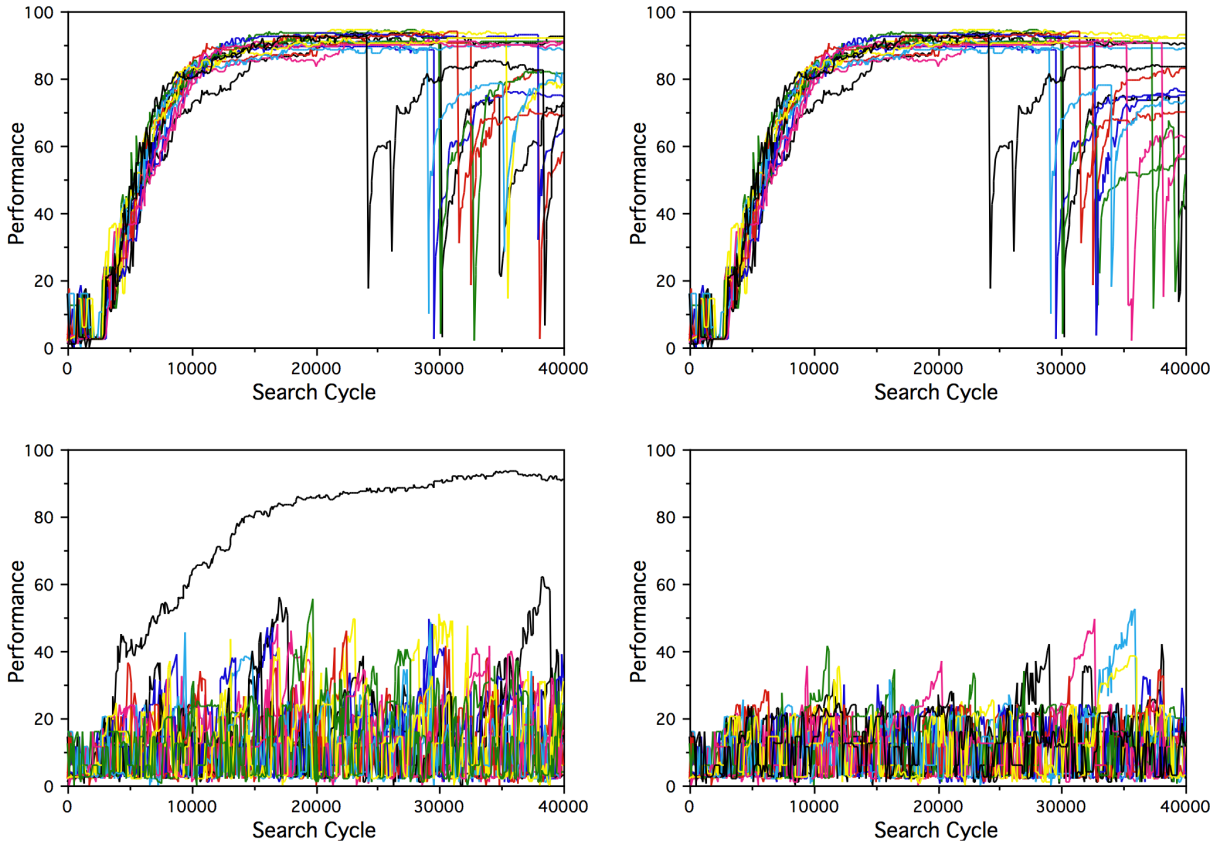

Fig. 5 Individual solution performances during a typical ABC training run for the Soybean dataset with abandoning limit of 1000 (top) and 16 (bottom), with elitism (left) and without elitism (right).