Evolving Improved Incremental Learning Schemes for Neural Network Systems

Tebogo Seipone & John A. Bullinaria
School of Computer Science
The University of Birmingham
Birmingham, B15 2TT, UK
{t.seipone, j.a.bullinaria}@cs.bham.ac.uk

Abstract- It is well known that incremental learning can often be difficult for traditional neural network systems, due to newly learned information interfering with previously learned information. In this paper we present simulation results which demonstrate how evolutionary computation techniques can be used to generate neural network incremental learners that exhibit improved performance over existing systems.

1 Introduction

Most neural network learning algorithms (Bishop, 1995) involve the network being trained with all the available data during a single training session, learning all the data concurrently. Once that training is finished, the network acquires no further information. Such concurrent training can make it difficult to update the network if additional information becomes available later, and needs to be incorporated into the neural network's performance.

An incremental learning algorithm gives a system the ability to learn from new information as it becomes available. Incremental learning is particularly important and relevant since in many real world applications the complete set of data is not all available at once, and learning really does need to be an ongoing process (Giraud-Carrier, 2000). A neural network should be able to use any new training data to improve its performance, without requiring access to the previous data. This could involve the network having to accommodate new classes of data that are introduced with the new data.

The problem we have with neural networks is that they are not naturally very good at incremental learning (Polikar, et al., 2001). They do not handle the *stability-plasticity dilemma* very well, and the learning of new data tends to interfere with the previously learned information in a manner that for memory systems is known as catastrophic forgetting (McCloskey & Cohen, 1989; Ratcliff, 1990). In a previous study we have shown how simulated evolution can be used to minimize such forgetting in neural memory systems (Seipone & Bullinaria, 2005). The aim of this paper is to extend that work to generalization tasks, and demonstrate how the application of evolutionary computation techniques can generate neural network incremental learners with better performance than traditionally built networks and other recent approaches to improving incremental learning. For concreteness, we shall restrict ourselves to classification problems, but the applicability to regression problems should be apparent.

In the next section we review the basic problem of incremental learning and the principal previous approaches to dealing with it. We then give an overview of how evolutionary computation can be usefully applied to optimizing neural network performance. In Section 4 we detail our series of simulations designed to explore the relevant issues, and in the following two sections present our main results. We end with some conclusions.

2 Incremental Learning Systems

Following Polikar et al. (2001), we define an incremental learning system as one that has the following properties:

1. Learning from new data does not cause large scale forgetting of previously acquired information.
2. It is able to acquire additional information from the new data, and hence improve its performance.
3. It does not require access to the previous data from which it learned its current state.
4. It has no problem accommodating any new classes that are introduced in the new data.

Humans have these properties, yet achieving them in artificial neural networks can pose serious problems.

When a neural network that has been trained on one set of patterns is then trained on a new set, its performance on the original set is affected. For memory tasks, this often results in catastrophic forgetting, where the new patterns seriously disrupt the patterns that were previously learned (French, 2003). This is a direct result of the stability-plasticity dilemma, and a major problem for artificial neural network models. The problem exists because of the very property that gives connectionist networks their generalization and graceful degradation abilities: the single set of shared weights representing the old information that gets modified as the new information is learnt.

For classification tasks, the interference may be less catastrophic, particularly if the new patterns only represent minor changes to the classification boundaries, but we still need to ensure that the new data improve the overall generalization performance rather than make it worse. In

particular, we need to make sure that the additional training doesn't cause over-fitting, and it is not obvious how best to incorporate standard regularisation techniques such as stopping early and weight decay (Bishop, 1995) into an incremental regime.

Several algorithms have already been developed with view to creating systems that allow the learning of new information without disrupting old information. Perhaps the simplest procedure is called *interleaved learning*. Here, the entire original training data set is mixed together with the new patterns, and the network retrained on this new expanded set. We can either continue training the existing classifier, or discard the previous classifier altogether and start again. This way, the new patterns will not interfere with the old information, satisfying properties 1 and 2 above, but property 3 is clearly violated. This method is also unrealistic in terms of human learning, and might be impractical in that it requires permanent storage of all the old data so that it is available for retraining. This can be a major problem because of the extra storage required, and in practice one regularly finds that the original data has been lost or corrupted anyway. Moreover, this approach might be too computationally expensive because larger training sets often result in longer training times.

Some other algorithms have been developed that try to reduce the size of the full training set by only using a subset of the new patterns, instead of the whole set. For example, Engelbrecht & Britis (2001) have devised a system whereby the candidate set of new training patterns is divided into clusters, and only the most informative pattern is selected from each of the clusters for adding to the training set. Of course, this approach still needs access to all the old data, and as the number of clusters increases, so does the computational complexity.

A related partial solution, that avoids having to use the entire original training set, is to employ *rehearsal*, where only a subset of the original data is used. Better still, *pseudorehearsal* (Robins, 1995) provides the advantages of rehearsal without requiring any access to the original data. After the neural network has been trained with the original data, the network generates pseudo-items by feeding randomly generated input vectors through it, producing a set of output vectors that correspond to the set of inputs. These pseudo-items approximate the earlier learning of the network, and a number of these are learnt together with the new data. This approach has been shown to substantially reduce the interference between sequential training items whilst still allowing new information to be learned, but there remains the problem of where to store these pseudo-patterns and how many of them to produce, and we still have to decide on appropriate values for all the traditional neural network parameters. Moreover, even though pseudorehearsal is guaranteed to succeed in high dimensions under fairly general conditions, it has been shown to fail in low dimensions. In fact, in some cases, the method is more likely to increase forgetting than to alleviate it, and if the number of pseudo-items approaches the dimensionality of the inputs, the system fails to learn the new items, let alone preserve old items (Frean & Robins, 1999).

Other researchers have developed somewhat different approaches to eliminate the need to access the old training data. Particularly successful is the *Learn++* algorithm of Polikar et al. (2001) which uses an ensemble of weak classifiers to generate multiple hypotheses using training data sampled according to some tailored distributions. Simulation results on a range of benchmark classification data-sets indicate that this algorithm works rather well in practice. Moreover, the Learn++ algorithm has also been shown to cope well with new classes, and hence satisfy property 4 above (Polikar et al. 2002).

Our own approach is to return to traditional neural network systems and use evolutionary techniques to optimize them to the extent that the usual problems of incremental learning are sufficiently minimised that we simply don't need to worry about them. We have previously shown how this works for memory tasks (Seipone & Bullinaria, 2005), and here we extend this approach to classification tasks.

3 Evolving Neural Networks

The idea of applying the basic principles and ideas of natural evolution to optimize the performance of neural network systems is now widely used (e.g. Yao, 1999; Bullinaria, 2003). A population of individual networks, each specified by an appropriate set of innate parameters, is maintained. The 'fittest' individuals from each generation, i.e. those exhibiting the best performance on their given task, are selected as parents. Suitable crossover and mutation operators are then applied to those parents to produce offspring for populating the next generation. This process is repeated, creating increasingly fit populations. Such an approach has been used to select optimal network topologies, learning algorithms, transfer functions, and other network parameters.

One attractive feature of evolving neural networks is the fact that any parameter of the neural network can be subjected to the evolutionary process, and it is possible for parameters that interact in complex manners to be evolved simultaneously. This means that the crucial and extremely difficult task of setting the neural network parameters can be left to the evolutionary process, rather than having to be set by hand by the network designer. Another distinctive feature of evolving neural networks is their adaptability to a dynamic environment, their ability to change their architecture and learning rules appropriately with limited or no human intervention. Results obtained from evolving neural networks have been reported to be significantly better than traditional hand built neural networks (Yao 1999, Bullinaria 2003).

The standard approach is to start the evolutionary process with an initial population of randomly created artificial genotypes, each encoding some or all of the free parameters of a neural network, or the initial values of any

adaptable parameters (such as connection weights). Each network is then trained and evaluated to determine its performance on the task at hand, and the fittest networks are allowed to reproduce by generating copies of their genotypes, with changes introduced by genetic operators such as crossover and mutations. This process is repeated for a number of generations until a network, or group of networks, that best satisfy the performance criteria is obtained. The major difficulties are to determine: which innate parameters to include in the genotype and how to represent them, how exactly to specify the fitness and choose the parents, and what are the most appropriate cross-over and mutation operators.

4 Simulation Details

To see what evolution can do to generate good neural network incremental learners, we now need to specify a concrete learning task, the details of our neural networks, and the properties we wish to evolve.

For ease of comparison, we used the main training data set studied by Polikar et al. (2001), namely the optical digits database from the UCI machine learning repository (Blake & Merz, 1998). This database contains hand-written digits 0 to 9 digitised on an 8x8 grid to create 64 input attributes for the ten classes. The full training file contains 3823 patterns. At each stage, these were divided into six distinct data sub-sets of 200 patterns (each with 20 patterns from each digit class) to be used for training, plus a further distinct sub-set of 1797 patterns to be used for validation purposes. The idea of course, is to maximize the generalization performance after training.

We specified the architecture of our networks and their learning algorithm to be fixed as standard sigmoidal Multi-Layer Perceptrons with one hidden layer, trained by gradient descent weight updating (back-propagation) with the Cross Entropy error measure (Bullinaria 2003). The nature of the training data then set the number of input units to be 64, and the number of output units to be 10, one for each class. The number of hidden units N_H is something that can be freely evolved, but preliminary simulations and related work (Seipone & Bullinaria, 2005) showed that the number tends to grow to near the maximum number allowed, which inflicts a considerable strain on the computational resources, so we fixed a maximum number of hidden units to be 100, which is many times that needed to learn the given training data.

The most obvious network parameters to evolve are the learning rates and initial weight distributions. It has been established in earlier studies (Bullinaria, 2003) that the best performance is obtained by allowing separate gradient descent learning rates η_L for each of the four network components L (hidden unit biases, output unit biases, input to hidden units, hidden to output units), and corresponding separate uniform initial weight distributions specified by the lower and upper ends of the range $[-l_L, +u_L]$. The networks are trained until all the output units are within a particular tolerance t of the target outputs for all training patterns. That tolerance t, a weight decay regularization parameter λ to limit over-fitting, a sigmoid prime offset s to prevent saturation of the hidden units, and connectivity proportions c_{IH}, c_{HO} between layers, are also evolved. In all, each network is specified by 18 innate parameters $\{N_H, \eta_L, l_L, u_L, t, \lambda, s, c_{IH}, c_{HO}\}$ whose values we aim to evolve to minimize over-fitting of the training data and hence maximize the generalization performance.

The initial population is started with random innate parameters, and for each generation, each individual has new random initial weights drawn from its own innately specified range. The aim is to evolve the various network parameters to produce networks with better incremental learning abilities.

There are two natural ways to use the training data. We can, for each generation, randomly select new training and validation pattern sets as specified above, or we can use the same sets for each generation. Having new training data for each generation proved to result in better general purpose learners, so we shall present results for that approach. Training takes place over series of six training sessions for each neural network. During each training session, only one of the six training sets is used to train each neural network until the error on each output unit is less than the innate tolerance t, or until a maximum number of training epochs is reached. The maximum number of epochs is set large enough that it only comes into play in the first few generations when the learning abilities are very poor. Each network is then tested on all the training sets used in the previous training sessions to see how much interference is taking place.

After the networks have been trained on each data-set, they are also tested on the validation set as a measure of their generalization ability. The fittest individuals are taken to be those with the lowest error on the validation set after training on all six data-sets. The least fit half of the population is then discarded, and each of the remaining individuals select a random partner and produce one child, thus restoring the population size. The children each inherit innate parameter values drawn randomly from the range spanned by both parents, plus random mutations from a Gaussian distribution that are added to allow values outside that range (Bullinaria, 2003; Seipone & Bullinaria 2005).

The UCI database also contains a separate testing data file of 1797 further patterns, none of which are used during any of the training sessions. This whole test set is reserved for a final testing of the evolved networks' incremental learning and generalization ability. As more and more of the six data sets are used to train an evolved network, its generalization ability is expected to gradually increase to demonstrate its incremental learning capability, but at the same time its performance on the previous datasets should not be seriously reduced.

For all our simulations, the populations consist of 100 neural networks. As explained above, each network is

	T1	T2	T3	T4	T5	T6
S_1	100	94.5	93.0	92.4	90.5	88.7
S_2	–	100	93.8	92.7	90.8	88.0
S_3	–	–	100	92.2	90.3	88.7
S_4	–	–	–	100	91.9	87.9
S_5	–	–	–	–	99.8	88.2
S_6	–	–	–	–	–	98.3
Val.	90.5	91.8	91.6	91.4	90.1	87.7
Test	88.0	89.1	89.0	89.0	87.6	85.4

Table 1: Average performances for neural networks trained using traditional back-propagation learning parameters.

	T1	T2	T3	T4	T5	T6
S_1	94	94	94	93	93	93
S_2	–	93.5	94	94	94	93
S_3	–	–	95	94	94	94
S_4	–	–	–	93.5	94	94
S_5	–	–	–	–	95	95
S_6	–	–	–	–	–	95
Val.	–	–	–	–	–	–
Test	82.0	84.7	89.7	91.7	92.2	92.7

Table 2: The Learn++ results presented by Polikar et al. (2001).

initialised with random initial weights from its own innate distribution, and is trained sequentially on six different data-sets using its own innately specified parameters. The networks' classification outputs are given by the highest activated output unit for each input pattern. Fitness was measured by generalisation performance on the validation set, and children were created to replace the least fit fifty networks by applying crossover and mutations to the fittest 50 networks.

5 Simulation Results

Before starting the evolutionary simulations, the baseline performance levels were established for standard neural network training parameters. One hundred fully connected networks with 100 hidden units were initialized with all their random weights drawn uniformly from the range [-1, 1], and trained for 5000 epochs using the back propagation algorithm with all the learning rates fixed at 0.02, and no weight decay or sigmoid prime offsets. The average performances of these standard networks are shown, as percentages, in Table 1. Each row shows the classification performance of the networks on the current data-set, and the performance on the same dataset after training on subsequent datasets. The last two rows show the generalization performance as measured on the validation and test sets. We see a general fall off in performance on each training data set as later sets are learned. The generalization performance does increase with the first two sets, but then starts falling again as the later sets are learned. This is a sure sign of poor incremental learning ability.

Polikar et al. (2001) developed their Learn++ system to obtain better incremental learning. Table 2 shows the results they achieved with the same data. Although the initial training performance starts lower, it is much more steady as more training data sets are introduced. We also see a steady increase in generalization performance as more data is made available. The question we now set out to explore is: can we do even better by evolving our neural networks to be good at incremental learning?

To get an idea of what is actually possible with the given data, we began by evolving networks to generalize as well as possible from only one set of 200 patterns, and from all six sets together (i.e. 1200 patterns) in a single training session. Exactly the same evolutionary regime was used as for the incremental learners, except that here we only have one training data set instead of six. Further details about evolved non-incremental learners such as these can be found in Bullinaria (2003). The ten fittest evolved individuals were then re-initialized and trained on new random training data-sets. This process resulted in final test set performances of 91.7% from 200 training patterns, and 95.8% from 1200 patterns. These set the best levels of performance that we can reasonably expect from our evolved incremental learners.

We now move on to the full evolutionary simulations described in the previous section. We started each initial population with innate connectivities, learning rates, and initial weight distribution parameters drawn randomly from the range [0, 1], tolerances from [0, 0.5], sigmoid prime offsets from [0, 0.2], weight decay parameters from [0, 0.001], and numbers of hidden units from [0, 100]. The precise starting ranges actually have very little effect on the final results, but poor values can lead to an extremely slow start to the evolutionary process.

The final results are fairly consistent across evolutionary runs starting from different random initial configurations. Figure 1 presents the population average results from a typical run. The top two graphs show how the learning rates and initial weight distributions evolve. The precise values don't tell us much, but note the large variation in learning rates that emerge for the different components. These would be very difficult to get right 'by hand'. The bottom left graph shows how the number of hidden units quickly drifts towards the maximum number allowed. Not shown (due to lack of space) are the sigmoid prime offset which quickly reduces to negligible values showing that it is of little help, the connectivity parameters that quickly establish full connectivity, the weight decay parameter that settles down to around 0.0000005, and the tolerance that settles down around 0.05. The bottom right graph shows how the generalization performance measures improve little after the first few hundred generations, despite many of the other parameters continuing to change. The persistent fluctuations in performance reflect the random nature of

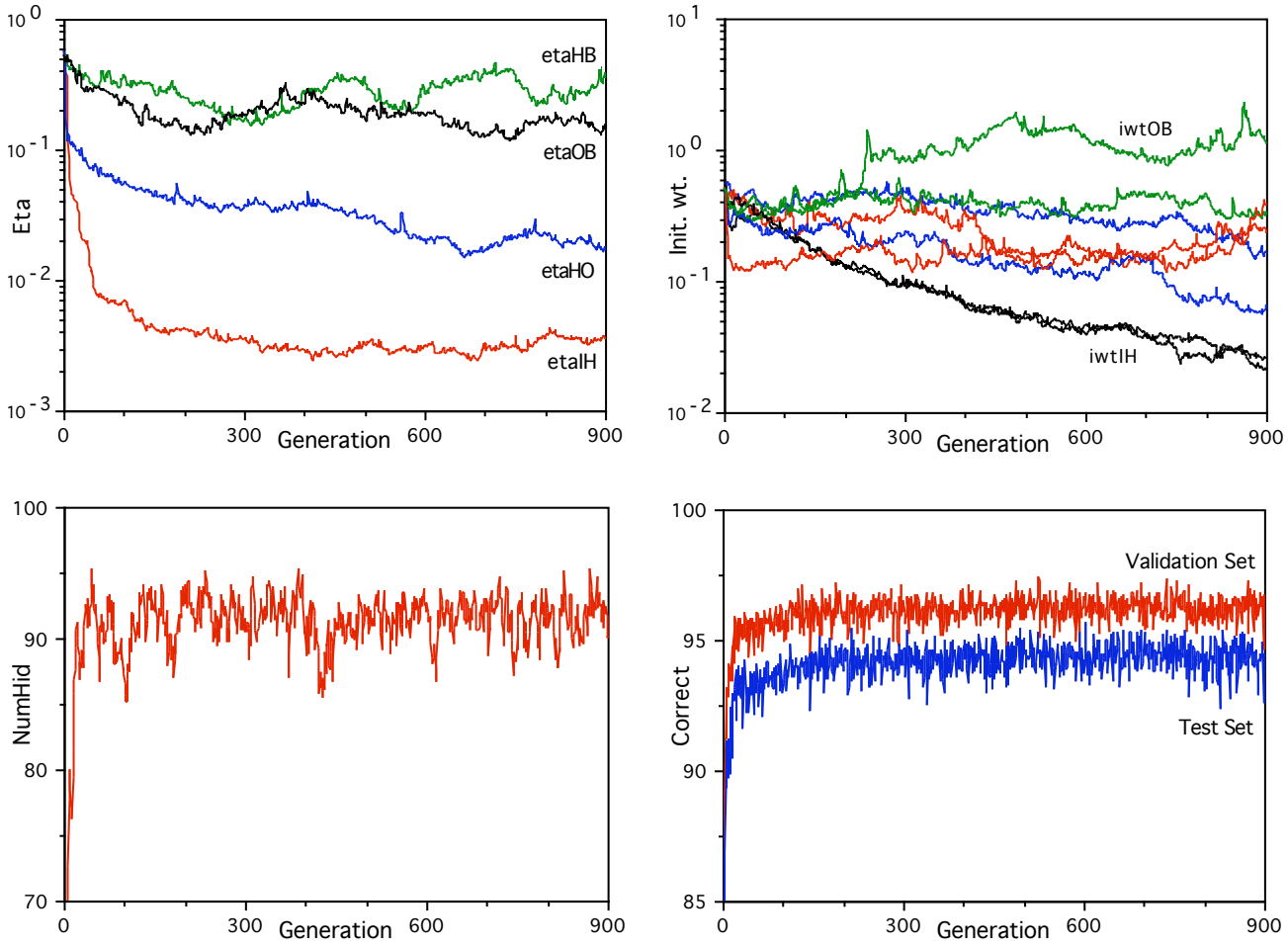

Figure 1: Evolution of the population average learning rates (top left), initial weight distributions (top right), number of hidden units (bottom left), and generalization performance measures after six sets of training data (bottom right).

the training data sets and initial weight distributions. The fluctuations in the parameters during evolution reflect how crucial each one is to the final fitness.

Table 3 shows the performance averages over ten runs of the fittest ten evolved networks. As one would expect, all aspects show an improvement over the standard network results of Table 1. More importantly, we also see an improvement over the Learn++ results of Table 2. The performance levels on the training data sets still fall slightly as the later training data sets are processed, but those performance levels remain well above those for Learn++. The generalization performance is also better than Learn++ at each stage, and shows a gradual improvement as more data sets are used, indicating a good incremental learner. The final test set performance of 94.4% appears a modest improvement over the Learn++ value of 92.7%, but it more than halves the gap between the incremental learning performance and the 95.8% obtainable by training on all the data at once.

It is clear that our evolved networks are satisfying the first three of the four properties of incremental learning systems discussed in Section 2. The fourth property will be discussed in the next section. First, we need to look at an unfortunate side effect of the evolutionary process, namely the number of epochs of training that is required to get the good results. This is a direct consequence of the evolved parameters, particularly the tolerance. During evolution, the tolerance falls, and that inevitably increases the number of epochs required to reach those lower error levels, as can be seen clearly in Figure 2. A similar requirement of slow training for good performance was also found in our related work on avoiding catastrophic forgetting in neural memory systems (Seipone & Bullinaria, 2005). However, it appears from the generalization results in Figure 1 that the final stages of evolution, including the massive increase in training time, actually have relatively little effect on the performance. One might therefore be tempted to impose a maximum number of epochs of training for each data-set, but this is problematic in that the first data-set will naturally require more epochs than later sets, and interfering with that could unbalance the whole incremental process. It makes better sense to see how well we can do if we simply stop the evolution after only 300 generations, when the training times are still relatively low, but the generalization performance appears to not be improving much further.

	T1	T2	T3	T4	T5	T6
S_1	100	98.8	98.4	98.1	97.9	98.0
S_2	–	100	98.9	98.3	98.0	97.8
S_3	–	–	100	99.0	98.5	98.3
S_4	–	–	–	100	99.1	98.5
S_5	–	–	–	–	100	98.9
S_6	–	–	–	–	–	100
Val.	93.4	95.0	95.5	95.9	96.0	96.3
Test	91.4	92.9	93.6	93.9	94.2	94.4

Table 3: Average individual performances after the neural network parameters have been evolved to a stable state.

	T1	T2	T3	T4	T5	T6
S_1	100	98.7	98.2	97.8	97.9	97.7
S_2	–	100	98.7	98.1	97.9	97.8
S_3	–	–	100	98.9	98.2	98.0
S_4	–	–	–	100	98.8	98.1
S_5	–	–	–	–	100	98.8
S_6	–	–	–	–	–	100
Val.	93.4	94.7	95.3	95.7	95.9	96.1
Test	91.2	92.7	93.3	93.7	94.1	94.3

Table 4: Average individual performances after evolving the neural network parameters for only 300 generations.

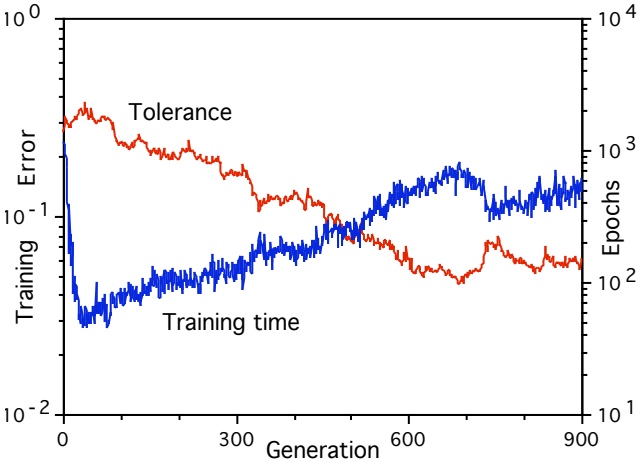

Figure 2: The relation during evolution between the training tolerance and resultant training times.

Class	Set 1	Set 2	Set 3	Set 4
0	100	50	50	25
1	0	150	50	0
2	100	50	50	25
3	0	150	50	25
4	100	50	50	0
5	0	150	50	25
6	100	50	0	100
7	0	0	150	50
8	100	0	0	150
9	0	50	100	50

Table 5: The differing distribution of classes across the "new classes" training sets, as used by Polikar et al. (2002).

Table 4 shows the performance levels this achieves. They are only slightly worse than after the full evolutionary process, and still show a big improvement over the non-evolved neural networks and the Learn++ results.

In our work on memory systems (Seipone & Bullinaria, 2005) we found that further improvements could be achieved by allowing even more hidden units. Preliminary results suggest that the same is true here, but the increased training times make getting statistically reliable performance results difficult.

6 Incremental Learning with New Classes

Having established that our networks exhibit the first three properties of good incremental learning systems described in Section 2, we now move on to explore property four, namely the ability of our networks to accommodate new classes of data that may be introduced with the new data. The same optical digits database was used as in the earlier simulations, but now, instead of using six data-sets containing equal proportions of all 10 classes, only four data-sets were used, with each data-set introducing some new classes or removing some previously seen classes.

A total of 2200 patterns randomly selected from the 3823 patterns in the training file were used for training, and 1270 of the remaining patterns used for validation. The distribution of the classes in the four sets matched that studied by Polikar et al. (2002) and is shown in Table 5. Each row shows the number of patterns from that class in each of the four sets. The validation and test sets, as before, both contain equal numbers of each class. The incremental learning results achieved by baseline neural networks and Learn++ are shown in Tables 6 and 7.

The simulations proceeded in the same way as before, except that instead of randomly selecting new training and validation sets for every generation, these were selected only once, with the same sets used for all generations. Figure 3 presents the main results from a typical run. We see a surprisingly different pattern of results to Figure 1. Dealing with new classes is obviously more difficult than the uniform case considered before, and new strategies are necessary. The most striking change is the high training tolerance of around 0.8, which means that many training patterns are nowhere near fully learned. Also of interest is the evolution of the performance results, which consists of a slow drift in average performance, plus increasingly large variations in performance between individuals. Despite those relatively large individual differences, there remains a strong correlation between the validation and test set

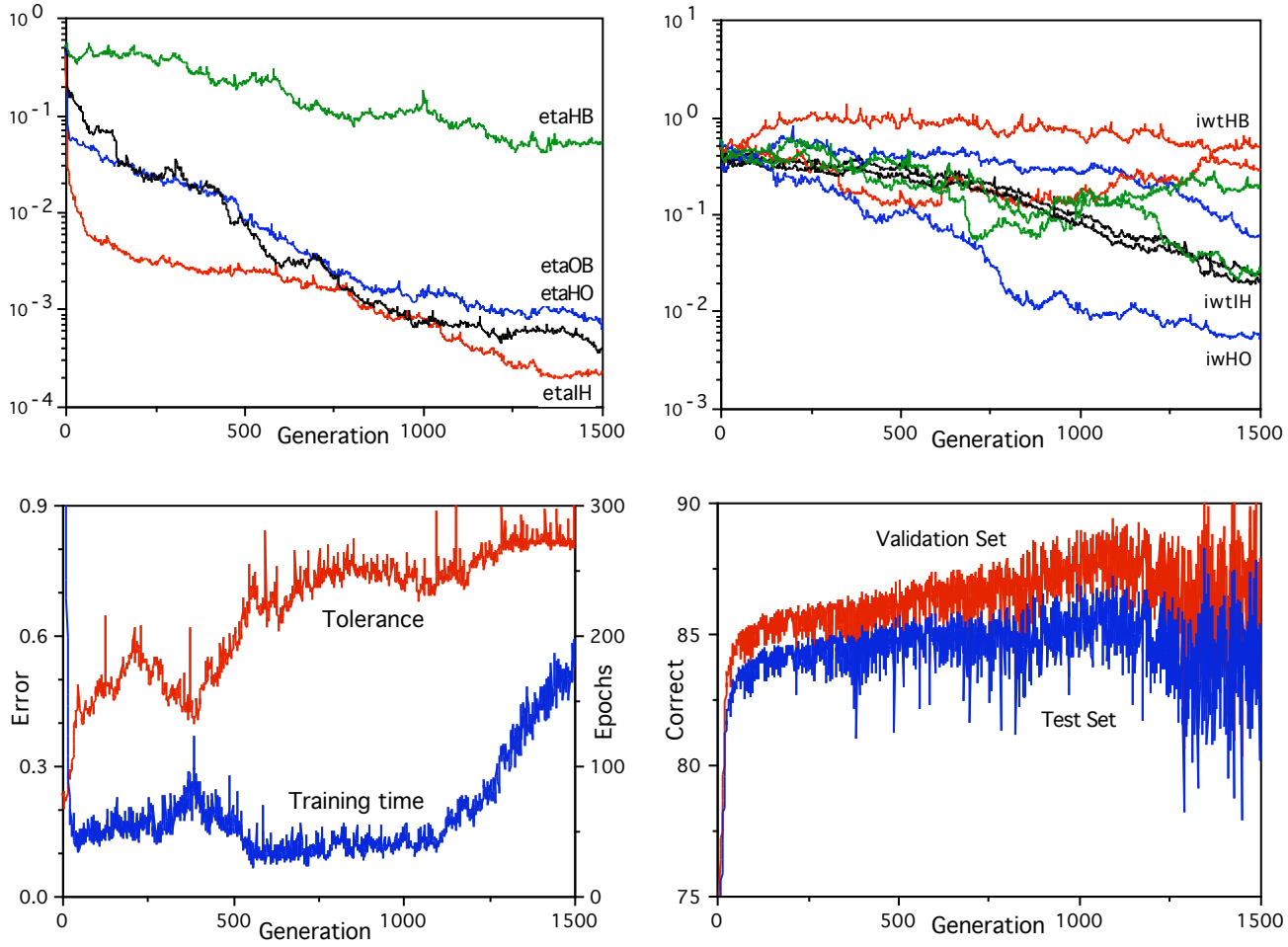

Figure 3: Evolution of the learning rates (top left), initial weight distributions (top right), training tolerance and time (bottom left), and generalization performance measures after four sets of training data (bottom right) for the "new classes" data-sets.

results, and measuring only the fittest ten individuals in each generation shows that evolution is still resulting in increasingly good test set results. By 1200 generations the mean fitness has stopped improving, with performance results shown in Table 8. We see clear improvement over the non-evolved neural networks and Learn++ results.

Interestingly, if we allow the evolution to continue further, we find that the variance across individuals increases even more, with significant numbers of very poor performers in each generation. However, the fittest ten individuals in each generation continue to show increasingly good final generalization. At generation 1500 we find the pattern of performance shown in Table 9. The final generalization performance has risen to 92.0%, but at the expense of much poorer generalization earlier on, only 32.8% after the first training set. This is clearly going to be problematic if we wish to have the best performance possible after each stage of training. Taking the final validation set performance as our measure of fitness worked well when all the classes were represented in all the training sets, but in the "new classes" situations, one may prefer to base the fitness on some function of the performance during the whole learning process. This is a straight-forward modification to the evolutionary process that will need to be tailored to specific applications.

7 Conclusions

We have reviewed the problems that traditional neural networks have in dealing with incremental learning tasks, and some of the previous approaches that have been developed to improve the situation. The main result of this paper has been to show that evolutionary computation techniques can alleviate these problems by optimizing standard neural networks to the extent that they perform much better than traditional neural networks, and also exhibit improved performance over more complex solutions such as Learn++.

We clearly have much further work to do, in particular, confirming the increase in performance for a wider range of data sets. It will also be important to evolve good general purpose neural networks, that can cope with any training data regime, rather than the fixed patterns of classes in each training set studied here. This will simply require switching between the relevant different training data set

	T1	T2	T3	T4
S_1	100	77.3	68.1	85.9
S_2	–	100	90.5	69.0
S_3	–	–	100	81.6
S_4	–	–	–	100
Val.	48.5	76.7	80.5	79.4
Test	48.3	75.7	80.1	77.9

Table 6: Performances for the 'new classes' data-sets for neural networks trained using traditional learning parameters.

	T1	T2	T3	T4
S_1	96.6	89.8	86.0	94.8
S_2	–	87.1	89.4	87.9
S_3	–	–	92.0	92.2
S_4	–	–	–	87.3
Val.	–	–	–	–
Test	46.6	68.9	82.0	87.0

Table 7: The Learn++ results of Polikar et al. (2002) for the 'new classes' data-sets.

	T1	T2	T3	T4
S_1	97.0	75.5	75.1	93.9
S_2	–	97.8	95.1	89.3
S_3	–	–	97.3	92.0
S_4	–	–	–	97.2
Val.	47.2	75.2	84.9	92.6
Test	47.5	73.2	84.4	90.4

Table 8: Average individual performances after 1200 generations of evolution with the "new classes" data-sets.

	T1	T2	T3	T4
S_1	67.3	75.4	74.9	93.3
S_2	–	98.1	95.3	93.7
S_3	–	–	97.7	95.9
S_4	–	–	–	95.4
Val.	32.7	76.1	85.0	94.4
Test	32.8	74.5	84.3	92.0

Table 9: Average individual performances after 1500 generations of evolution with the "new classes" data-sets.

regimes with each generation, so that populations evolve that can cope well with them all.

There may also be further improvements to be had by incorporating dual weight architectures, with two sets of weights, one of which learns and decays faster than the other (Hinton & Plaut, 1987). We have already shown that evolving such dual weight systems can reduce the catastrophic interference in neural memory systems beyond that achieved by evolving standard networks (Seipone & Bullinaria, 2005), and it is possible that it will produce similar improvements in classification networks too.

Bibliography

Bishop, C.M. (1995). *Neural Networks for Pattern Recognition*. Oxford, UK: Oxford University Press.

Blake, C.L. & Merz, C.J. (1998). *UCI Repository of machine learning databases*. University of California, Irvine, Dept. of Information and Computer Science. http://www.ics.uci.edu/~mlearn/MLRepository.html.

Bullinaria, J.A. (2003). Evolving Efficient Learning Algorithms for Binary Mappings. *Neural Networks*, **16**, 793-800.

Engelbrecht, A.P. & Brits, R. (2001). A Clustering Approach to Incremental Learning for Feedforward Neural Networks. *Proceedings of the International Joint Conference in Neural Networks*, **3**, 2019-2024

Frean, M. & Robins, A. (1999). Catastrophic Forgetting in Simple Neural Networks: An Analysis of the Pseudorehearsal Solution. *Network: Computation in Neural Systems*, **10**, 227-236.

French, R.M. (2003). Catastrophic Interference in Connectionist Networks. In: L. Nadel (Ed), *Encyclopedia of Cognitive Science,* **1**, 431-435.

Giraud-Carrier, C. (2000). A Note on the Utility of Incremental Learning. *AI Communications*, **13**, 215-223.

Hinton, G.E. & Plaut, D.C. (1987). Using Fast Weights to Deblur Old Memories. *Proceedings of the Ninth Annual Conference of the Cognitive Science Society*. NJ:Erlbaum. 177-186.

McCloskey, M. & Cohen, N.J. (1989). Catastrophic Interference in Connectionist Networks: The Sequential Learning Problem. *The Psychology of Learning and Motivation*, **24**, 109-165.

Polikar, R., Byorick, J., Krause, S., Marino, A., & Moreton, M. (2002). Learn++: A Classifier Independent Incremental Learning Algorithm for Supervised Neural Networks. *Proceedings of the 2002 International Joint Conference on Neural Networks*. **2**, 1742-1747.

Polikar, R., Udpa, L., Udpa, S.S., & Honavar, V. (2001). Learn++: An Incremental Learning Algorithm for Multi-Layer Perceptron Networks. *IEEE Transactions on Systems, Man, and Cybernetics*. **31**, 497-508.

Ratcliff, R. (1990). Connectionist Models of Recognition and Memory: Constraints Imposed by Learning and Forgetting Functions. *Psychological Review,* **97**, 205-308.

Robins, A. (1995). Catastrophic Forgetting, Rehearsal and Pseudorehearsal. *Connection Science,* **7**, 123-146.

Seipone, T. & Bullinaria, J.A. (2005). The Evolution of Minimal Catastrophic Forgetting in Neural Systems. *Proceedings of the Twenty-Seventh Annual Conference of the Cognitive Science Society*. Mahwah, NJ: LEA.

Yao, X. (1999). Evolving Artificial Neural Networks. *Proceedings of the IEEE,* **87**, 1423-1447.