ISSN: 1803-3814

AN EXPERIMENTAL STUDY ON COMPETITIVE COEVOLUTION OF MLP CLASSIFIERS

Marco Castellani, Rahul Lalchandani

University of Birmingham
Department of Mechanical Engineering
Birmingham B15 2TT
United Kingdom
m.castellani@bham.ac.uk

Abstract: *This paper investigates the effectiveness and efficiency of two competitive (predator-prey) evolutionary procedures for training multi-layer perceptron classifiers: Co-Adaptive Neural Network Training, and a modified version of Co-Evolutionary Neural Network Training. The study focused on how the performance of the two procedures varies as the size of the training set increases, and their ability to redress class imbalance problems of increasing severity. Compared to the customary backpropagation algorithm and a standard evolutionary algorithm, the two competitive procedures excelled in terms of quality of the solutions and execution speed. Co-Adaptive Neural Network Training excelled on class imbalance problems, and on classification problems of moderately large training sets. Co-Evolutionary Neural Network Training performed best on the largest data sets. The size of the training set was the most problematic issue for the backpropagation algorithm and the standard evolutionary algorithm, respectively in terms of accuracy of the solutions and execution speed. Backpropagation and the evolutionary algorithm were also not competitive on the class imbalance problems, where data oversampling could only partially remedy their shortcomings.*

Keywords: *evolutionary algorithms; coevolution; predator-prey systems; multi-layer perceptron; pattern classification.*

1 Introduction

Training a multi-layer perceptron (MLP) [1] classifier is essentially an optimisation problem in the high-dimensional space of the neural network (ANN) weights. The optimal solution is the one that attains the smallest classification error, or, equivalently, the highest classification accuracy. The most popular MLP training method is error backpropagation (BP) [2], a greedy optimisation algorithm based on gradient descent of the error surface. Due to the local search approach, BP is liable to get trapped into sub-optimal error minima, or be deceived by noise in the training patterns. For these reasons, global optimisation techniques like evolutionary algorithms (EAs) have been employed as alternative training methods [3] [4]. The standard EA procedure is to evolve a population of MLP solutions using the classification accuracy as fitness measure. Unfortunately, EAs imply a high computational cost due to the need to evolve a population of candidate solutions. If large ANNS or training sets are used, this cost implies prohibitively long execution times.

Typical evolutionary MLP learning curves are characterised by an initial phase of rapid improvement, where the population quickly locates one or more regions of high fitness in the parameter space, and a second phase of slow progress where the population gradually converges to a fitness peak. In the first phase, the candidate solutions learn a broad-brush description of the decision regions, which allows them to classify correctly the majority of the data patterns, whilst in the second phase they refine the borders of such regions. That is, in the second phase the individuals learn to discriminate those patterns that lie at the boundary between two or more classes [5]. To speed up the learning process, BP is often used as an additional genetic operator (Larmarckian evolution, [4]). However, the addition of BP increases the computational complexity of the EA, and the whole procedure may become impractically long.

One of the most time-consuming procedures in EA optimisation is the evaluation of the population fitness. For MLP classifiers, the accuracy of the candidate solutions is typically calculated on the whole training set of examples. This procedure implies that each training pattern needs to be forward processed by every individual. If Lamarckian evolution is used, two further (one forward and one backward) data passes need to be performed for each training pattern by each solution undergoing BP. Random sampling of the training set [4] helps to reduce the cost of evaluating and training the solutions, but may reduce the learning accuracy and speed. Ideally, as the learning process progresses, the sampling procedure should adaptively focus on the not yet learned examples.

Paredis introduced CGA [5][6], a coevolutionary genetic algorithm inspired by biological predator-prey interactions. In this scheme, one standard EA (GENITOR [7] in CGA) is used to evolve a population of MLP classifiers, and another evolutionary procedure is used to select from the training set the patterns for MLP fitness evaluation. The classifiers (predators) are evaluated on their ability to categorise (capture) randomly selected subsets of data samples (prey), whilst the data samples are evaluated on the number of times they are misclassified (ability to escape predation). The samples are selected for the evaluation subsets with a likelihood that is proportional to their fitness. According to this scheme, an evolutionary tug-of-war is created between classifiers and training data, with the aim of increasing the effectiveness and efficiency of the training procedure. Paredis experimentally showed that, after the initial phase of rapid improvement of the accuracy, the fittest prey lied at the boundaries of the data categories. By selecting mainly these difficult examples, CGA could outperform the GENITOR algorithm in terms of quality of the solutions and execution speed.

In a recent experimental study, Castellani [8] proved the robustness of two predator-prey MLP training algorithms to noisy and overlapping data categories. That is, when the nature of the data does not allow 100% classification accuracy, the coevolutionary procedure is not lead astray by the unsolvable cases. Castellani also indicated the promise of predator-prey algorithms for classification problems involving imbalanced data classes. In this latter case, once the categorisation of the most common classes is learned, the predator-prey procedure focuses on the examples of the least represented class. One question that has not yet been addressed concerns how the performance of the competitive approach scales up to large data sets. That is, how the ability of predator-prey mechanisms to identify challenging training patterns varies as the number of data instances is increased.

2 Aims and Objectives

This study focused on the competitive coevolutionary approach to MLP training. Competitive coevolution is intended in this paper as a multi-population procedure where interactions are modelled on predator-prey interactions [10]. Single-population schemes where members of the same population are pitched against each other (e.g. backgammon players [11]) are outside the scope of this paper. Hereafter, unless explicitly stated, the terms 'competitive' and 'predator-prey' will be used interchangeably. The aim of this work was to investigate the following two questions:

- how the effectiveness and efficiency of competitive coevolution vary as the size of the training set is increased.
- how competitive coevolution fares as the imbalance amongst data classes is increased.

Both research questions concern the ability of competitive coevolution to select critical information amongst increasingly large data sets. Two different predator-prey algorithms were tested, and their performance compared to the performance of a standard EA and the BP rule. In the class imbalance tests, oversampling [9] was used as a term of comparison to evaluate the effectiveness of the two predator-prey algorithms.

It is important to underline that this work aimed at shedding further light on the strengths, shortcomings, and overall viability of the competitive approach to MLP evolution. Ascertaining which of the many EA implementations [3] is most effective for MLP training was outside the scope of the research. For this reason, the same EA was used to train the MLP population in the two predator-prey algorithms and the standard EA. The experimental results are thus expected to reveal the effect of the data evolution approaches tested, and be fairly independent of the kind of evolutionary MLP optimiser used.

3 Methods

This section describes the methods used to investigate the scientific questions detailed in Section 2.

3.1 Data Sets

For ease of comparison with the literature, Paredis' four class artificial classification problem [4][5] was used in the tests. Paredis' problem consisted of 200 data patterns defined in the R^2 domain $D = [-1,1] \times [-1,1]$. The data classes were delimited as follows:

$$class = \begin{cases} 1 & if\ (x,y) \in D\ \&\ x^2 + y^2 < 0.25 \\ 2 & if\ (x,y) \in D\ \&\ (y > 1 - x\ ||\ y < -1 - x) \\ 3 & if\ (x,y) \in D\ \&\ (x > 0.5\ \&\ y < 0\ ||\ x > 0\ \&\ y < -0.5) \\ 4 & else \end{cases} \tag{1}$$

In this study, a number of data sets were randomly generated in D, labelling the data as in (1). A sample set is shown in Fig. 1. All sets were divided into a training set of variable size (M) and composition, and a test set (the same for all tests) of 1000 data patterns, equally partitioned into 250 patterns per class. The first set of experiments was carried out on five training sets of size varying from 1000 to 20000 elements, equally distributed among the four classes. The second set of experiments was carried out on four training sets of variable size, where class '1' ('minority' class) was respectively 5, 10, 15, and 20 times smaller than the other three classes. The main features of the data sets are summarised in Table 1.

3.2 Algorithms

This study investigated the performance of two predator-prey algorithms: Co-Adaptive Neural Network Training (*CANNT*) [8], and Co-Evolutionary Neural Network Training (*CENNT*) [8]. Both algorithms feature two coevolving populations (classifiers and training patterns), and are composed of three modules: *Predator*, *Prey*, and *Interaction*. The *Predator* module is the same for *CANNT* and *CENNT*. It evolves the classifiers, and can be thought of as a standard EA for MLP evolution. The *Prey* module adapts (*CANNT*) or evolves (*CENNT*) the training data, and can be thought of as

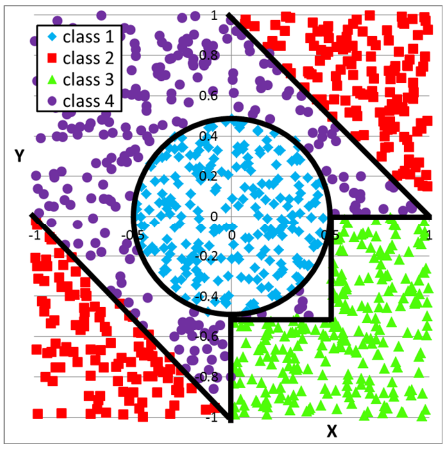


Figure 1: Data set 1K. For clarity, the boundaries between classes have been marked by a line

Table 1: Main features of data sets

Name	Overall Training patterns	Breakdown Class 1	Class 2	Class 3	Class 4	Test patterns
1K	1000	250	250	250	250	1000 patterns, 250 per class
5K	5000	1250	1250	1250	1250	
10K	10000	2500	2500	2500	2500	
15K	15000	3750	3750	3750	3750	
20K	20000	5000	5000	5000	5000	
UB5	800	50	250	250	250	
UB10	1550	50	500	500	500	
UB15	2300	50	750	750	750	
UB20	3050	50	1000	1000	1000	

another evolutionary procedure running in parallel to the *Predator* module. The candidate solutions of the *Predator* and *Prey* modules meet in the *Interaction* module, where they are evaluated and Lamarckian evolution takes place. One generation of the *CANNT* and *CENNT* algorithms includes the execution of one cycle of each of the three modules. After a set number of generations have elapsed, the fittest MLP classifier is taken as the final solution. The performance of the two coevolutionary algorithms was compared to the performance of the standard BP rule [2] (henceforth simply referred to as *BP*) and a traditional EA [4] (henceforth simply referred to as *EA*).

3.2.1 Co-Adaptive Neural Network Training (CANNT)

The algorithm follows the same scheme of CGA [5], where a population of N MLPs is coevolved with a population of $M = size(training\ set)$ training patterns. Like CGA, the *Predator* module concatenates the MLP weights into one long string of real numbers (chromosome). Differently from CGA, the *Predator* module uses generational replacement with elitism [12] (steady-state in CGA), fitness ranking [12] (proportional selection in CGA), and no crossover (two-point crossover [12] in CGA). The mutation operator of the *Predator* module changes all the elements (genes) of a selected chromosome. For each gene, the perturbation is randomly sampled with uniform probability from a small interval of fixed width. Lamarckian evolution occurs in the *Interaction* module. The structure of the MLP is pre-set and does not undergo evolution, whilst the weights of the initial population are randomly set to a small value.

The population of training patterns (*Prey* module) does not undergo true evolution, since only the fitness of the individuals is adapted. The fitness $f(\cdot)$ of the training patterns is normalised in the interval $[0,1]$, where $f(v_j) = 0$ indicates that pattern v_j ($j \in [1, M]$) was successfully categorised by all classifiers encountered, and $f(v_j) = 1$ that was misclassified by all classifiers encountered. At initialisation, all M training patterns are assigned fitness $f(v_j) = 0.5$.

In the *Interaction* module, each of the p_i ($i \in [1, N]$) predators (classifiers) is paired up with a sample s_i of $K_i \leq M$ prey (patterns) for Lamarckian evolution and evaluation. A different subset of training patterns is randomly picked for each candidate MLP solution. Different subsets may have a different number of data patterns, and each pattern may be included in different subsets. Each pattern v_j has a probability $f(v_j)$ of being selected in each of the N training subsets. That is, the fittest (most difficult) examples are the most likely to be picked. Each MLP has a chance of undergoing one cycle of BP training, and then is evaluated. For each classifier p_i, Lamarckian evolution and evaluation are carried out using the paired up subset s_i. The fitness of p_i is equal to its classification accuracy:

$$f(p_i) = \frac{c(s_i)}{K_i} \qquad (2)$$

where $c(s_i)$ is the number of patterns in s_i correctly classified by p_i, and K_i is the size of s_i. The fitness $f(v_j, t)$ of training pattern v_j at generation t is calculated as:

$$f(v_j, t) = 0.8 \cdot f(v_j, t-1) + 0.2 \cdot \frac{m_j(t)}{P_j(t)} \tag{3}$$

where $m_j(t)$ is the number of times v_j was misclassified at iteration t, and $P_j(t)$ is the number of training subsets v_j belongs to. That is, the fitness of a predator corresponds to its success in capturing (classify) prey, and the fitness of a prey corresponds to how successful is to evade predation (be misclassified). Note that the fitness of both predators and prey is normalised in $[0,1]$. The first term at the right hand side of Eq. (3) carries over a memory of the past evaluations of the training pattern, mimicking Paredis' life-time fitness evaluation mechanism [5]. Further information on the *CANNT* algorithm can be found in [8].

3.2.2 Co-Evolutionary Neural Network Training (CENNT)

CENNT uses the same *Predator* module used by *CANNT*. However, differently from *CANNT* that works at the level of individual patterns, *CENNT* evolves entire subsets of the training set. That is, the candidate solutions of the *Prey* module of *CENNT* are sets of patterns. The prey population is half the size of the predator population, namely $N/2$ individuals. Each solution is encoded as a binary string of size M; if the j^{th} element of the string is set to '1' the corresponding pattern v_j is included in the training subset.

The *Prey* module of *CENNT* is akin to a standard Genetic Algorithm [12]. It uses generational replacement, fitness ranking, two-point crossover [12], and bit flip mutation [12]. If an individual is selected for mutation, all the genes of the chromosome have a pre-set probability of being changed. The population of the *Prey* module is randomly initialised, giving each gene an equal probability of being set to '1' or '0'. If during the evolution process an offspring is created with all genes equal to '0' (empty set), the individual is randomly re-initialised.

In the *Interaction* module, each training data subset is paired to two MLP classifiers. Each MLP p_i undergoes one cycle of Lamarckian evolution on the associated data subset s_j. The fitness of p_i is then calculated as the average of its classification accuracy on s_j.

$$f(p_i) = \frac{c_i(s_j)}{K_j} \tag{4}$$

where K_j is the size of s_j, and $c_i(s_j)$ is the number of patterns of s_j correctly classified by p_i. The fitness of each data subset s_j is equal to:

$$f(s_j) = 1 - \frac{1}{2} \cdot \left(\frac{c_i(s_j)}{K_j} + \frac{c_k(s_j)}{K_j} \right) \tag{5}$$

where $c_i(s_j)$ and $c_k(s_j)$ are the two classifiers paired up to s_j.

In the original formulation, *CENNT* was slower than *CANNT* at discarding commonly learned patterns, and consequently required longer execution times [8]. To address this shortcoming, a procedure was added to the *Interaction* module. For each subset of the *Prey* module, each training pattern that is correctly classified by both MLPs is discarded. That is, its associated gene is changed to '0'. This new procedure can be thought of as an instance of Lamarckian evolution of the prey population. Further information on the *CENNT* algorithm can be found in [8].

3.2.3 Backpropagation and standard EA

The backpropagation rule with momentum term and a standard EA were used as terms of comparison for the performance of *CANNT* and *CENNT*. *BP* was implemented using a stochastic learning scheme, that is, the ANN weights were updated immediately after the presentation of each training pattern. The EA is the same evolutionary procedure employed in the *Predator* module of the two coevolutionary algorithms. However, in *EA* the whole training set is used by the Lamarckian learning and MLP fitness evaluation procedures.

3.2.4 Oversampling

Oversampling [9] was employed as an alternative method to redress the data imbalance problem. The procedure duplicates the members of the least numerous (minority) class until all data categories have the same number of training patterns. The procedure is implemented according to a deterministic scheme, where the members of the minority class are duplicated one by one until class distribution has been balanced.

3.3 Tests

The first series of tests aimed to verify how the performance of the predator-prey algorithms is affected by the size of the training set. The performance was analysed in terms of classification accuracy of the MLP solutions and speed of execution. The speed of execution was evaluated using the number of data passes (forward or backward) performed by the candidate solution(s). That is, a data pattern is processed by an MLP by forward processing the input information from the input layer to the output layer. In the BP rule, the error information is processed backwards through the ANN weights. Each of such passes was used as unit of measurement of speed. Based on experimental evidence, Castellani [8] indicated that this measure can be regarded as a good proxy for the execution time of the algorithms. The efficiency and

Table 2: Experimental tests

Test	Objective	Data Set	Algorithm
1-5	Training set size	1K, 5K, 10K, 15K, 20K	CANNT, CENNT, BP
6-9	Class imbalance size	UB5, UB10, UB15,UB20	CANNT, CENNT, BP, EA, BP_B, EA_B

Table 3: Parameterization of the algorithms

Algorithm Module	*Test*	*BP*	*CANNT & CENNT PREDATOR*	*CENNT PREY*	*CANNT & CENNT INTERACTION*
Learning cycles	1-5	50000	20000	20000	20000
	6-9	50000	50000	50000	50000
Population Size	all	n.a.	50	25	n.a.
BP learning rate	1-5	0.005	n.a.	n.a.	0.01
	6-9	0.01	n.a.	n.a.	0.01
Momentum term	all	0.1	n.a.	n.a.	n.a.
Initial MLP weights range	1-5	[-0.5, 0.5]	n.a.	n.a.	[-0.05, 0.05]
	6-9	[-0.05, 0.05]	n.a.	n.a.	[-0.05, 0.05]
MLP weights mutation rate	all	n.a.	0.2	n.a.	n.a.
MLP weights mutation width	all	n.a.	0.1	n.a.	n.a.
BP operator rate	all	n.a.	n.a.	n.a.	0.8
BP operator cycles	all	n.a.	n.a.	n.a.	1
Crossover rate	all	n.a.	n.a.	1.0	n.a.
Prey mutation rate	all	n.a.	n.a.	0.3	n.a.

time of the predator-prey algorithms was investigated using data sets 1K-20K. Due to the excessively long execution times, the *EA* algorithm was not used in this first series of tests.

The second series of tests examined the ability of the predator-prey algorithms to learn increasingly imbalanced data distributions. The performance of *CANNT* and *CENNT* was compared to the performance of *BP* and *EA* on the UB5-UB20 data sets, and the *BP* and *EA* after oversampling was applied. Henceforth, the *BP* and *EA* algorithms applied in conjunction with oversampling will be called *BP_B* and *EA_B*, where the suffix '*B*' stands for 'balanced'.

The parameterization of the MLP and training algorithms was performed experimentally. The MLP structure was set to 2 input nodes, one hidden layer of 30 nodes, and 4 output nodes (one per class). The hidden nodes used the hyper-tangent transfer function, whilst the output nodes used the sigmoidal transfer function. Tables 2 and 3 summarise respectively the experimental set up and the parameterization of the algorithms.

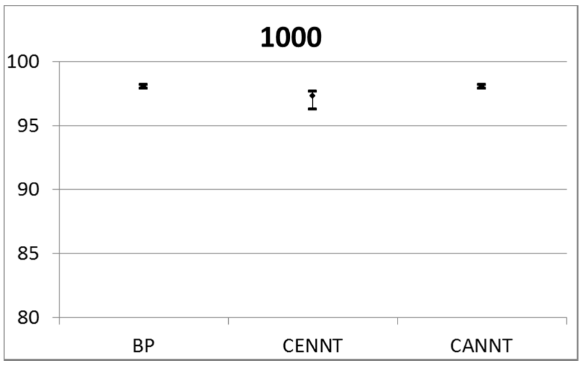
4 Experimental results

The results of the first series of experiments (tests 1-5) are given in Table 4 and visualised in Fig. 2. Table 4 reports the median of the classification accuracy attained by each algorithm in 10 independent learning trials (left hand side) and the median of the number of forward and backward passes performed by each algorithm per trial. For reference, the rightmost column of Table 4 reports the number of passes that would be performed by a standard EA. For each data set, the best accuracy result and those that are not significantly different from the best are highlighted in bold. The statistical significance of the differences between the results was analysed using pairwise Mann-Whitney U tests. The tests were run for a 5% level of significance. Fig. 2a-e visualise for each data set the first, second (median), and third quartile of the accuracy attained by each algorithm on the 10 learning trials. Fig. 2f shows the variation of the median accuracy.
The results obtained in this first series of experiments show that the performance of the *CANNT* algorithm is largely unaffected by the size of the data sets, whilst the *CENNT* algorithm becomes increasingly competitive as the size of the training set increases. Conversely, the training accuracy of the BP algorithm deteriorates severely as the size of the training set increases. In general, BP training became increasingly difficult as the training set became larger. The BP learning rate had to be halved to prevent the algorithm to get stuck in regions of the parameter space where the gradient saturated. For this reason, the duration of the BP procedure had to be increased to 50000 cycles. Further improvement of the performance would have been possible increasing further the number of cycles, even though the learning curve was extremely slow and the gains would have been modest in relation to the extra training time.

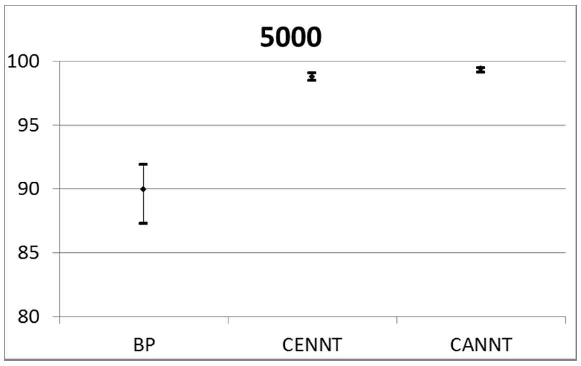
The two predator-prey algorithms required a number of data passes comparable to BP, and between 5 and 20 times less the data passes that would have been required by a standard EA. The execution speed varied accordingly, ranging between about 10 minutes on the smallest data set and over 5 hours on the largest data set. The *CANNT* algorithm gave the best accuracy results, although *CENNT* became increasingly accurate and efficient as the data set size increased. Indeed, on the two largest data sets the performance of the two competitive algorithms is statistically indistinguishable.

Table 4: Classification accuracy and computations vs. data set size

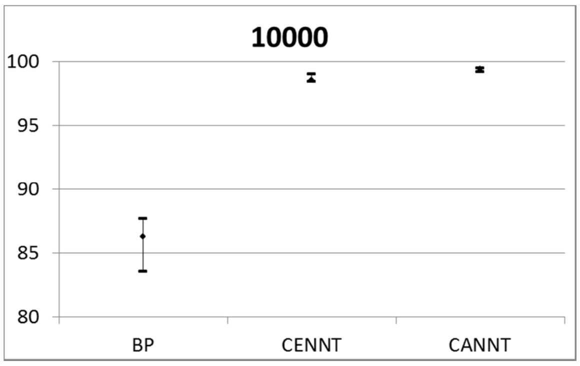
Data set	Accuracy (%)			Passes ($\times 10^6$)			
	BP	*CENNT*	*CANNT*	*BP*	*CENNT*	*CANNT*	*EA*
1K	**98.05**	97.30	**98.05**	100	104	234	2000
5K	89.95	98.75	**99.40**	500	559	1143	10000
10K	86.25	98.55	**99.40**	1000	1232	2279	20000
15K	85.70	**99.00**	**99.10**	1500	1920	4383	30000
20K	83.60	**99.20**	**99.30**	2000	2725	7017	40000



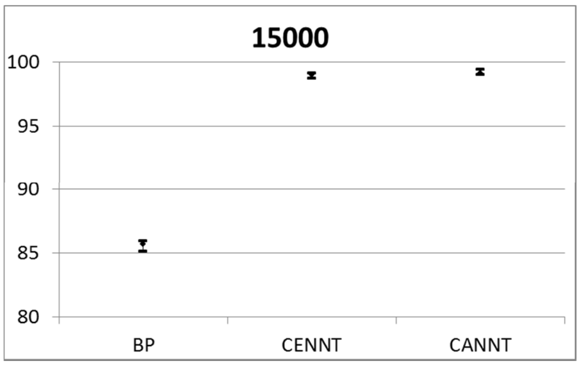
a) 1K data set



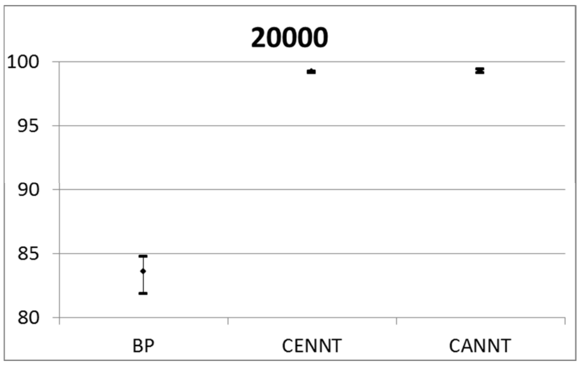
b) 5K data set



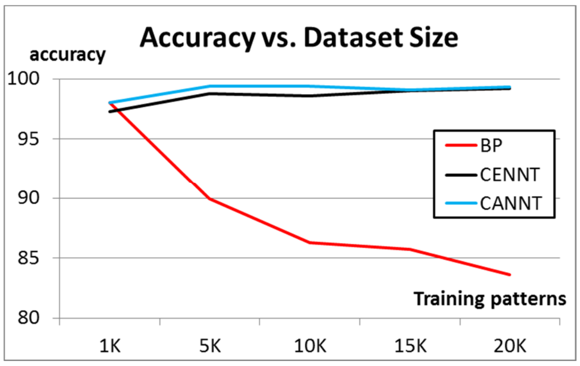
c) 10K data set



d) 15K data set



e) 1K data set



f) median accuracy vs. number of training patterns

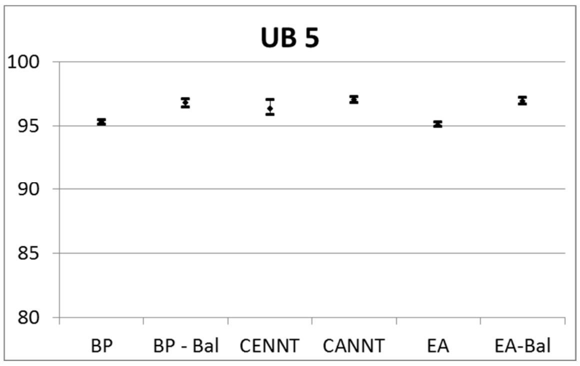
Figure 2: classification accuracy vs. data set size

Paredis [5] and Castellani [8] reported top classification accuracies of respectively 96.45% and 96.53% on the same classification problem. The accuracy results obtained in this study are slightly superior. This improvement in the quality of the solutions is probably due to the fact that the training process was extended to 20000 generations, instead of the 10000 generations used by Castellani [8].

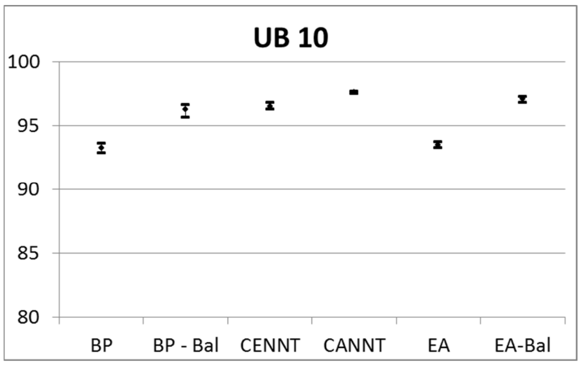
Table 5 reports the results of the second series of experiments following the same conventions used in Table 4. Likewise, Fig. 3 visualises the results using the same format of Fig. 2a-f. The *CANNT* algorithm is always amongst the

Table 5: Classification accuracy vs. class imbalance

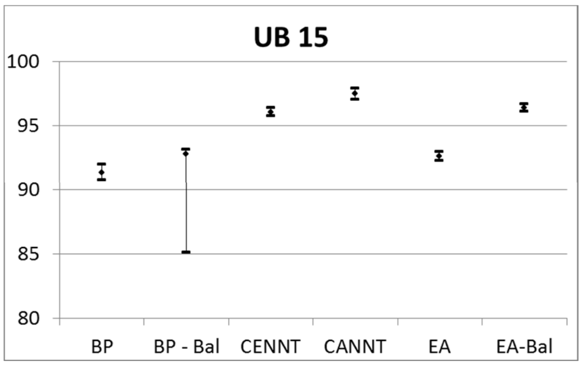
	Overall					
	BP	*BP_B*	*CENNT*	*CANNT*	*EA*	*EA_B*
UB5	95.25	**96.80**	**96.30**	**97.00**	95.05	**96.90**
UB10	93.25	96.25	96.50	**97.60**	93.45	97.05
UB15	91.35	92.75	96.05	**97.50**	92.60	96.40
UB20	89.15	87.90	96.35	**97.15**	91.50	96.90
	Minority Class (class 1)					
	BP	*BP_B*	*CENNT*	*CANNT*	*EA*	*EA_B*
UB5	86.80	**93.80**	**91.80**	**94.40**	87.20	**94.80**
UB10	78.20	**92.80**	91.80	**94.20**	77.40	**92.80**
UB15	72.60	88.00	88.80	**92.60**	73.60	90.00
UB20	67.00	83.80	**89.40**	**91.20**	69.60	**92.00**
	Passes ($\times 10^6$)					
	BP	*BP_B*	*CENNT*	*CANNT*	*EA*	*EA_B*
UB5	80	100	231	431	5200	6500
UB10	155	200	308	802	10075	13000
UB15	230	300	399	1172	14950	19500
UB20	305	400	501	1516	19825	26000



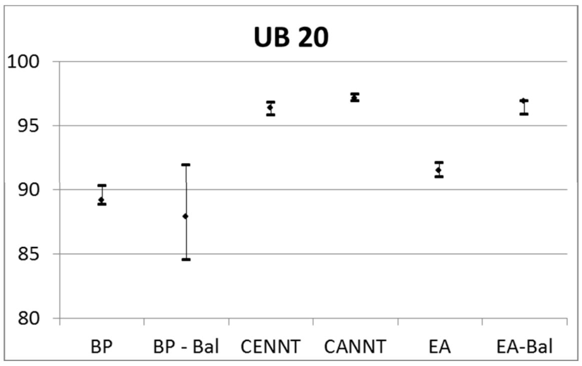
a) UB 5 data set



b) UB 10 data set



c) UB 15 data set



d) UB 20 data set

Figure 3: classification accuracy vs. class imbalance

best performers. *CENNT* evolved less accurate classifiers than *CANNT*, although its performance appeared to improve slightly with the data set size. In terms of computational overheads, the number of data samples required by *CANNT* increased proportionally to the class imbalance, whilst the *CENNT* algorithm was more economical. Compared to *BP* and *BP_B*, the predator-prey procedures required between 2 (*CENNT*) and 5 (*CANNT*) times more training data passes, whilst the standard evolutionary procedures required between 65 (*CENNT*) and 80 (*CANNT*) times more passes. The reduced computational needs of *CANNT* and *CENNT* translated into much reduced execution times compared to *EA* and *EA_B*. That is, whilst the predator-prey procedures required a couple of hours to complete one learning trial on the largest (UB20) data set, the standard EA procedures required nearly one day.

The ability of *BP* and *EA* to satisfactorily categorise the minority class decreases as the class imbalance increases. Oversampling sensibly improves the classification accuracy of *BP_B* and *EA_B* on the less numerous class. However, oversampling creates multiple copies of the instances of the minority class, and thus increases the size of the training set. Unfortunately, the learning accuracy of BP deteriorates on large training sets (see Table 4), and the gains in accuracy on the minority class are offset by an overall loss in accuracy. Indeed, on the set with the largest imbalance (UB20), the overall accuracy of *BP_B* is inferior to the accuracy of *BP*. The performance of the evolutionary procedures is not affected by the size of the training set, and *EA_B* attains competitive accuracy results on nearly all data sets. Unfortunately, oversampling increases the computational complexity of *EA_B*, and the number of MLP training data passes performed by *EA_B* on the UB20 set is about 50 times the number of passes performed by *CENNT*.

5 Conclusions

This paper analysed the performance of the *CANNT* and *CENNT* predator-prey algorithms on data sets of varying size and class imbalance. Overall, the *CANNT* algorithm produced the most accurate classifiers. However, on the largest data sets *CENNT* was competitive in terms of quality of the solutions, and excelled in terms of economy of training set samples. The above results suggest that *CANNT* should be used on data sets of moderate size, and *CENNT* on large sets.

The two predator-prey algorithms confirmed their effectiveness on classification problems characterised by class imbalance. Large training sets were problematic for the BP and standard EA algorithms. In the first case, the algorithm was unable to train highly accurate solutions on the largest data sets. In the second case, the algorithm execution time was unacceptably long. The BP and standard EA algorithms failed also to train satisfactorily the classifier in presence of class imbalance. In particular, the solutions produced by BP and the standard EA performed poorly on the minority class. Oversampling was effective in compensating the class imbalance problem, but created large data sets where the accuracy of BP and the execution speed of the standard EA were impaired.

This study on the effects of data set size and class imbalance on the performance of competitive coevolution is an original contribution of this paper. This paper also introduced a new Lamarckian learning procedure in *CENNT*, which removes already learned patterns from the candidate training subset. Thanks to this new procedure, *CENNT* excelled in the economy of training data samples, and hence execution speed. Future work should investigate further the ability of competitive coevolution to address class imbalance problems. The effectiveness and efficiency of CANNT and CENNT should be compared with that of other popular methods to redress class imbalance, such as undersampling, boosting [9], and cost-sensitive learning [9].

References

[1] Haykin, S.: *Neural Networks and Learning Machines*, 3rd edn. Prentice Hall, New York, USA (2009).

[2] Rumelhart, D., McClelland, J.: *Parallel distributed processing: explorations in the microstructure of cognition*, vol. 1-2, MIT Press, Cambridge, USA (1986).

[3] Yao, X.: Evolving Artificial Neural Networks. *Proceedings IEEE* 87(9), pp. 1423-1447 (1999).

[4] Castellani, M.: Evolutionary Generation of Neural Network Classifiers - An Empirical Comparison. *Neurocomputing* 99, pp. 214-229 (2013)

[5] Paredis, J.: Coevolutionary life-time learning. In: I. Rechenberg and H.P. Schwefel (ed.) *Parallel Problem Solving from Nature - PPSN IV*, pp. 72-80. Springer, Berlin Heidelberg, (1996).

[6] Paredis, J.: Coevolutionary computation. *Artificial life* 2(4), pp. 355-375 (1995).

[7] Whitley, D.: The Genitor algorithm and selection pressure: why rank-based allocation of reproductive trails is best. In: JD Schaffer (ed.) *Proceedings of the Third International Conference on Genetic Algorithms*, pp. 116-123. San Mateo, CA, Morgan Kaufmann Publishers, San Francisco, USA (1989).

[8] Castellani, M.: Competitive co-evolution of multi-layer perceptron classifiers. *Soft Computing*, published online (2017). DOI 10.1007/s00500-017-2587-6

[9] Weiss, G.M.: Mining with rarity: a unifying framework. *ACM SIGKDD Explorations Newsletter* 6(1), pp. 7-19 (2004).

[10] Popovici, E., Bucci, A., Wiegand, R.P., De Jong, E.D.: Coevolutionary principles. In: G. Rozenberg et al. (ed.) *Handbook of Natural Computing*, pp. 987-1033. Springer, Berlin Heidelberg (2012).

[11] Pollack, J.B., Blair, A.D.: Co-evolution in the successful learning of backgammon strategy. *Machine Learning* 32(3), pp. 225-240 (1998).

[12] Fogel, D.B.: *Evolutionary computation: toward a new philosophy of machine intelligence*, 2nd edn. IEEE Press, New York, USA (2000).