Training Neural Networks with PSO in Dynamic Environments

Anna Rakitianskaia and Andries P. Engelbrecht, *Senior Member, IEEE*

Abstract— **Supervised neural networks (NNs) have been successfully applied to solve classification problems. Various NN training algorithms were developed, including the particle swarm optimiser (PSO), which was proved to outperform the standard back propagation training algorithm on a selection of problems. It was, however, usually assumed that the decision boundaries do not change over time. Such assumption is often not valid for real life problems, and training algorithms have to be adapted to track the changing decision boundaries and detect new boundaries as they appear. Various dynamic versions of the PSO have already been developed, and this paper investigates the applicability of dynamic PSO to NN training in changing environments.**

I. Introduction

Most algorithms from the Computational Intelligence field assume that the search landscape is static. However, this assumption is not valid for many real-world problems. In case of classification problems, the decision boundaries may change over time due to the changes in the underlying context. This phenomena is also known as concept drift [1]. NNs are widely applied to classification problems due to their excellent pattern recognition ability, and it is crucial to ensure that the training algorithms used can track the decision boundaries as they change over time.

Recent developments have shown PSO to be a very effective NN training algorithm [2], [3], [4], and it was of particular interest to the authors to investigate the applicability of this algorithm to NN training in dynamic environments. Due to the success of the PSO in static environments, a number of variations have been developed such that PSO can be applied to dynamic environments [5], [6], [7], [8], [9]. Some of the most successful PSO algorithms for dynamic environments are the charged PSO and the quantum PSO, developed by Blackwell and Bentley [5], [7]. The aim of this paper was to apply these algorithms to NN training, and to investigate their performance on problems that incorporate concept drift of varied severity. The dynamic PSO algorithms were benchmarked against standard back propagation.

Recently, dimension-wise cooperative search has been shown to improve the performance of the charged PSO [10]. The applicability of this dynamic PSO variation to NN training is also investigated in this paper.

The rest of the paper is structured as follows: Section II provides background on NNs, back propagation, dynamic PSO, and concept drift. Section III discusses the experimental setup, problems used in the experiments, and experimental results. Section IV sums up the findings presented in this paper and concludes the paper with final remarks.

Anna Rakitianskaia is with the Department of Computer Science, School of Information Technology, University of Pretoria, Pretoria 0002, South Africa (email: annar@cs.up.ac.za).

A. P. Engelbrecht is with the Department of Computer Science, School of Information Technology, University of Pretoria, Pretoria 0002, South Africa (email: engel@cs.up.ac.za).

II. Background

This section briefly discusses various algorithms used in the paper and properties of dynamic environments.

A. Neural Networks

A neural network is a simple mathematical model inspired by the learning mechanisms of the human brain. NNs are able to carry out such tasks as pattern recognition and function approximation. A NN is essentially a collection of interconnected neurons aligned in layers. It can represent any non-linear function, provided the hidden layer has enough neurons [11], [12]. The NN itself is just a structure capable of representing a function, requiring to be trained on a problem in order to learn a functional mapping between input space and output space. Training can be supervised, unsupervised or reinforced. This paper deals with supervised NNs only. Such NNs work on a set of data patterns, where each pattern is a vector of problem inputs and the corresponding target values.

Many different supervised training algorithms exist. The most commonly used and popular algorithm is back propagation with gradient descent, which iteratively adjusts the NN weights to decrease the resulting NN output error. It is essentially a hill-climbing algorithm, and its major disadvantage is susceptibility to premature convergence on local minima. As a solution to the problem of premature convergence, alternative training algorithms such as PSO were suggested [4].

B. Training NNs with PSO

Particle Swarm Optimisation (PSO) was first introduced in [13]. It is a nature-inspired population-based optimisation technique. As the name suggests, PSO operates on a swarm of particles, where each particle represents a candidate solution to the problem. The swarm is flown through the search space, and each particle is attracted to both the best position encountered by itself so far, as well as the overall best position found by its neighbourhood. As a result of such social dynamics, a standard swarm normally converges on a small area around the optimum solution [14], [15]. For a detailed description of the PSO algorithm, see [13], [16].

In order to train a NN with PSO, each particle of the swarm represents the weight vector of the NN, including the biases. The dimension of the search space is therefore determined by the number of weights and biases. The fitness of a particle is determined by constructing the NN from the weight vector and calculating the mean squared error over the training set.

C. Concept Drift

Dynamic environments can often be observed in real life, for example, the stock exchange, or traffic conditions on roads. Given a problem in such an environment, it is clear that a solution once found may become suboptimal, because the characteristics of the environment will change over time. In case of classification problems, the changes in the hidden context may cause class boundaries to shift over time. This type of dynamic changes is known as concept drift. Concept drift can either be gradual, when the decision boundaries shift over time, or abrupt, when the old boundaries are altogether replaced by new boundaries [1].

The goal of optimisation algorithms in dynamic environments is therefore not only to find the optimal solution to the problem, but also to detect the environment changes and to adjust the solution accordingly.

D. Dynamic PSO

A number of PSO algorithms for dynamic environments have been developed. These include split adaptive PSO [9], fine-grained PSO [8], charged PSO [5], and quantum PSO [7].

All dynamic PSO variants have to face the following two major problems:

1) Outdated memory: Once the environment changes, previous values stored in PSO memory (personal best and global best positions) are no longer representative of the new environment [16].
2) Loss of diversity: It was formally proved [14], [15] that all particles will converge on a point, which is a weighted average of personal best and global best positions. Once converged, PSO will not explore any longer, even if the environment changes [16].

Simple solutions to the above problems are:

- Reevaluate particle positions when a change occurs. Reevaluation is effective only when particles have not yet converged.
- Reinitialise a percentage of particles when a change occurs. This will increase diversity, but at the same time valuable information about the search space will be lost.

Two of the most successful PSO algorithms for dynamic environments are the charged PSO and the quantum PSO. The main objective of this paper is to apply these algorithms to NN training in dynamic environments, and to compare their performances to that of standard gradient descent training.

1) Charged PSO: The charged PSO [5] is based on electrostatic principles. All particles in a charged PSO store a charge, represented by a positive scalar value. A charge magnitude equal to zero means that a particle is neutral (i.e. does not bear a charge), and a value greater than zero indicates a charged particle. Charge magnitude can not be negative, and does not change during algorithm execution. Charged particles repel from one another if the distance between them is small. This prevents charged particles from converging to a single point, thus facilitating exploration. Refer to [5] for a detailed description of the algorithm.

As empirical results have shown [5], the most efficient charged PSO architecture has 50% of the swarm charged, and the rest of the particles are neutral. Neutral particles are normal PSO particles that obey standard PSO position and velocity update rules. Thus, one half of the population acts as a standard PSO swarm and refines found solutions, in this way facilitating exploitation. Hence, the charged PSO achieves a balance between exploration and exploitation.

2) Quantum PSO: The quantum PSO [7] is vaguely based on the model of an atom. The orbiting electrons of an atom are replaced by a quantum cloud, where the position of each electron is determined not by its previous position and certain trajectory dynamics, but by a probability distribution instead. Similarly, a percentage of particles in the quantum swarm are treated as the "quantum cloud", and at each iteration the cloud is randomised in the spherical area with radius r_{cloud} centred around the global best particle of the swarm. The particles that do not constitute the cloud behave according to the standard PSO velocity and position update equations. Since the quantum cloud is randomised at each iteration, the swarm is not allowed to completely converge on a small area, hence swarm diversity is preserved. The non-quantum particles refine the current solution while the quantum cloud searches for new and better solutions. In this manner, a good balance between exploration and exploitation is achieved.

3) Cooperative Charged PSO: Cooperation was suggested as an improvement to the standard charged PSO [10]. The cooperative charged PSO (CCPSO) is a hybrid between the divide-and-conquer mechanism of the cooperative split PSO [17] and the diversity preserving charged PSO. The search space is divided into smaller subspaces, with each subspace being optimised by a separate charged PSO. The best solutions from all sub-swarms are combined to form the overall solution. For an elaborate discussion of the algorithm, refer to [10].

III. Experimental Results

The different algorithm and problem parameters used in the experiments that were carried out are outlined in this section. All algorithms were implemented using the CIlib (Computational Intelligence library) framework. CIlib is an open-source project written in the Java programming language. The latest version that includes implementations of all the algorithms analysed in this paper, together with problem simulations, can be downloaded from `http://cilib.sourceforge.net`.

A. Experimental Setup

The aim of the experiments was to investigate the performance of various NN training algorithms on a representative selection of dynamic classification problems. The description of the problems used in the experiments and the corresponding algorithm and problem parameters are outlined below.

1) Back propagation: Standard back propagation with sigmoid activation functions in the hidden and output layers was used for the experiments. The mean squared error (MSE) was used as the error function. Learning rate and momentum were

TABLE I

NUMBER OF HIDDEN UNITS

Problem	# Hidden Units
Moving Hyperplane	6
Dynamic Sphere	4
Sliding Thresholds	3

TABLE II

DYNAMIC SCENARIOS

Parameter	Scenario I	Scenario II	Scenario III
Window size	1000	1000	1000
Step size	100	500	1000
# iterations on a window	100	500	1000

both set to 0.5, since it proved to give optimal performance under the scenarios considered. All NNs used had one hidden layer, and the number of hidden units used for each problem is shown in Table I.

2) PSO: In all the experiments, the inertia weight was set to 0.729844, while the values of the acceleration coefficients were set to 1.496180. This choice is based on [18], where it was shown that such parameter settings give convergent behaviour. Although preserving diversity is vital in the context of dynamic environments, convergent behaviour is still desirable, since it is important to find a solution quickly. The main challenge of dynamic optimisation is to achieve a good balance between exploration and exploitation.

The value of $vMax$ was set to 2, since it improved the swarm's ability to exploit good solutions in the changing environments. The standard fully informed Global Best swarm topology was used. The swarm size was set to 30 particles.

3) Quantum PSO: The radius of the quantum cloud was set to 2 for all the experiments, and 50% of the swarm constituted the cloud. These values were based on the empirical studies presented in [7].

4) Charged PSO: Parameters for the charged PSO were based on empirical and theoretical studies published in [7]. Radii of repulsion R_c and R_p were set to 1 and 1000, respectively. Such a wide range was chosen in order to constrain acceleration the least. All charged particles bore a charge magnitude of $Q = 0.3$. 50% of each swarm was charged with another 50% kept neutral.

5) Cooperative Charged PSO: In order to be comparable with the other PSO variants, the cooperative charged PSO used the same total number of particles. Since the swarm size was fixed to be 30, two versions of the CCPSO were used: CCPSO with 2 sub-swarms of 15 particles each, and CCPSO with 3 sub-swarms of 10 particles each.

B. Benchmark Problems

The benchmark problems used to compare algorithm performance are described in this section.

1) Moving Hyperplane: The first dynamic classification problem used a 10-dimensional hyperplane to separate two classes. The hyperplane is given by

$$\sum_{i=1}^{N}(a_i x_i) + c = a_0,$$

where N is the number of dimensions over which the hyperplane is defined, $a_i, i \in \{1, 2, \ldots, N\}$ are linear coefficients, and c is a constant. All points satisfying $\sum_{i=1}^{N}(a_i x_i) + c > a_0$ are labelled as class A, and all points satisfying $\sum_{i=1}^{N}(a_i x_i) + c \leq a_0$ as class B. For the purpose of this study, N was set to 10, yielding a 10-dimensional problem. The linear coefficients and c were real numbers chosen from the interval $[0, 1]$. A set of 1000 10-dimensional points, $\{\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_{1000}\}, \vec{x}_i \in [0, 1]^{10}, i \in \{1, \ldots, 1000\}$, was randomly generated, and these were classified into two classes A and B according to the randomly generated hyperplane. The hyperplane was generated by randomising its coefficients, $\{a_1, a_2, \ldots, a_N\} \in [0, 1]$, the constant $c \in [0, 1]$ and the threshold value, $a_0 \in [0, 1]$. The hyperplane was regenerated 20 times, and each time the points $\{\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_{1000}\}$ were recorded with updated classification. This resulted in a data set of 20,000 patterns, where each pattern mapped 10 input variables to one of the two classes. Dynamic environments were simulated by sliding a window over this data set. Since the aim was to simulate decision boundaries that change over time, the data set was not shuffled to preserve the original pattern order. The patterns were only shuffled inside the window, this was done to prevent NNs from learning the pattern order instead of the classification boundaries.

Shifting the window by N patterns implies discarding the first N patterns from the window, and appending the next N patterns from the data set to the window. The larger is the value of N, the more drastic are the changes introduced, since a lot of new information is added while a lot of previously valid information is discarded. If the decision boundary changes every M patterns in the data set, any shift by $N < M$ patterns will introduce a *new decision boundary* into the window while still keeping the data patterns classified according to the previous decision boundary inside the window. As the window slides along the data set, more patterns classified according to the previous decision boundary will be replaced by the patterns classified according to the current boundary, until the previous boundary is completely discarded. This means that the training algorithm will often have to deal with more than one decision boundary, with a chance of having boundaries that contradict each other, i. e. classify the same pattern into opposite classes. It makes adaption more challenging for the training algorithms.

Three different dynamic scenarios were considered. For each scenario, two parameters had to be set: the number of patterns by which the window should shift in order to simulate change, or, in other words, the step size; and the number of iterations that the current training algorithm should be allowed to run before the window shifts. The size of the sliding window was fixed to 1000 patterns. Different parameter settings used are shown in Table II.

In Scenario I, a window of 1000 patterns was shifted by 100 patterns every 100 iterations of the training algorithm. In Scenario II, more drastic changes were simulated by increasing the step size to 500 patterns. Since the changes were more drastic, the training algorithms were given more time to adapt to a change - the environment changed every 500 iterations instead of 100. In Scenario III, the step size was equal to the window size, which means that all patterns in the window were replaced when a shift occurred. The algorithms were trained for 1000 iterations on each window between environment changes.

2) Dynamic Sphere: The second dynamic classification problem was obtained by generating a hypersphere in $\mathbb{R}^3$ and using it to divide the space into two mutually exclusive classes. The hypersphere is defined as

$$\sum_{i=1}^{N}(x_i + c_i) = R^2, \tag{1}$$

where R is the radius of the sphere, and $\vec{c}$ is the centre of the sphere. For the purpose of this study, N was set to 3. A set of 1000 3-dimensional points, $\{\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_{1000}\}$, $\vec{x}_i \in [0,1]^3$, $i \in \{1, \ldots, 1000\}$, was randomly generated, and each point x_i was classified either as class A or class B, depending on whether it was inside or outside of the sphere, respectively. The sphere was generated by randomising its centre point, $\vec{c} \in [0,1]^3$, and radius, $R \in [0,1]$. The sphere was randomised 20 times, and each time the points $\{\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_{1000}\}$ were recorded with updated classification according to Equation (1). A data set of 20,000 patterns representing a dynamic environment was thus obtained. Three dynamic scenarios as described earlier and outlined in Table II were applied to the dynamic sphere problem.

3) Sliding Thresholds: For the last problem, the Cartesian space was subdivided into three classes by means of two parallel linear thresholds, $f_1(\vec{x})$ and $f_2(\vec{x})$, defined as

$$\begin{aligned} f_1(\vec{x}) &= t_1 \\ f_2(\vec{x}) &= t_2 \\ t_1 &< t_2, \end{aligned}$$

where t_1 and t_2 are constant values, therefore $f_1(\vec{x})$ and $f_2(\vec{x})$ are parallel vertical lines. Thus, a 2D point can be classified based on its x component only. Classification was done as follows:

$$Classification(x) = \begin{cases} Class\ A & \text{if } x \leq t_1 \\ Class\ B & \text{if } x \geq t_2 \\ Class\ C & \text{otherwise} \end{cases} \tag{2}$$

A set of 1000 2-dimensional points, $\{\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_{1000}\}$, $\vec{x}_i \in [0,1]^2$, $i \in \{1, \ldots, 1000\}$, was randomly generated, and each point x_i was classified according to Equation (2). Thresholds were generated by setting t_1 and t_2 to random numbers from the interval $[0,1]$ such that $t_1 < t_2$. Thresholds were randomised 20 times, and each time the points $\{\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_{1000}\}$ were recorded with updated classification according to Equation (2). As with the previous two problems, a data set consisting of 20,000 patterns was obtained. This problem was considered under the three different scenarios given in Table II.

TABLE III
MOVING HYPERPLANE

Algorithm	Scenario I	Scenario II	Scenario III
Charged PSO	0.23121	0.19286	0.23287
Quantum PSO	0.23018	0.19176	0.23103
CCPSO, 2 subswarms	0.23460	0.19452	0.24902
CCPSO, 3 subswarms	0.24561	0.20636	0.25044
Gradient Descent	0.23016	0.18871	0.22719

All reported results are averages over 30 simulations. Each algorithm ran for 9000 iterations.

C. Performance Measurement

The aim of this paper was to investigate the performance of NN training algorithms in the presence of shifting decision boundaries, thus only classification problems were considered. Classification error expressed as the percentage of patterns incorrectly classified by the NN was used to measure algorithm performance.

Morrison [19] showed that a representative performance measurement in a dynamic environment should reflect algorithm performance "across the entire range of landscape dynamics". Standard performance measurements reflect the current algorithm state only, and therefore can not be used. Morrison [19] suggested using mean fitness instead. Mean fitness is the average over all previous fitness values:

$$F_{mean}(T) = \frac{\sum_{t=1}^{T} F_t}{T}$$

where T is the total number of iterations, and F_t is algorithm fitness after iteration t. The mean fitness represents the entire algorithm performance history, hence it is valid in dynamic environments. This performance measurement was used to compare the experimental results described below.

D. Analysis of Empirical Data

The mean fitness results obtained after 9000 iterations on moving hyperplane, dynamic sphere, and sliding thresholds are reported in Tables III, IV, and V, respectively. In addition to this, four graphs are drawn to visualise the results obtained on the dynamic sphere under scenario I and sliding thresholds under scenarios I, II, and III (see Figures 1, 2, 3, and 4). The following naming conventions are used:

- In all tables, CCPSO stands for cooperative charged PSO.
- In all tables and figures, scenarios I, II and III refer to dynamic scenarios as described in previous sections and summarised in Table II.
- The standard gradient descent based back propagation NN training algorithm is referred to as gradient descent.

Algorithm performance on the three problems considered is discussed below.

TABLE IV
DYNAMIC SPHERE

Algorithm	Scenario I	Scenario II	Scenario III
Charged PSO	0.16629	0.21324	0.24240
Quantum PSO	0.16431	0.21346	0.24325
CCPSO, 2 subswarms	0.16625	0.21481	0.24208
CCPSO, 3 subswarms	0.17388	0.22586	0.25488
Gradient Descent	0.23016	0.21292	0.24202

TABLE V
SLIDING THRESHOLDS

Algorithm	Scenario I	Scenario II	Scenario III
Charged PSO	0.17695	0.23117	0.29454
Quantum PSO	0.18686	0.24907	0.31053
CCPSO, 2 subswarms	0.19297	0.26857	0.30366
CCPSO, 3 subswarms	0.19314	0.25229	0.30899
Gradient Descent	0.66071	0.59966	0.59167

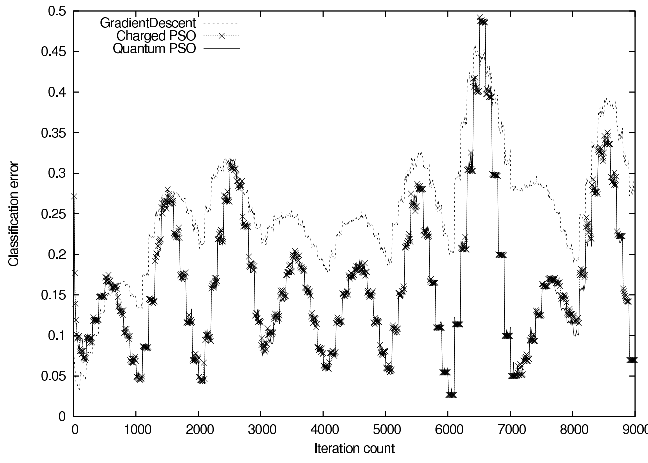

Fig. 1. Dynamic Sphere, Scenario I

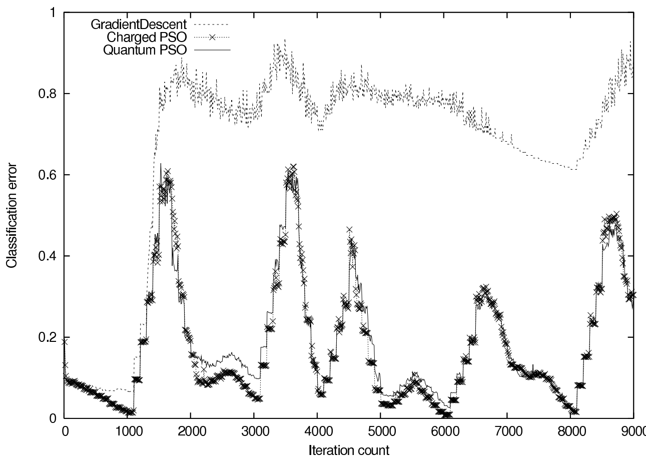

Fig. 2. Sliding Thresholds, Scenario I

1) Moving Hyperplane: The moving hyperplane involved a single decision boundary which was linear. A 10-dimensional version of this problem was used in the experiments. As discussed in the previous sections, a dynamic environment was simulated by gradually (scenarios I and II) or abruptly (scenario III) introducing a new decision boundary, and discarding the previous decision boundary correspondingly.

The mean fitness results given in Table III show that all the training algorithms tested worked best under scenario II. No algorithm managed to significantly outperform the other, with gradient descent being slightly better than the rest, and quantum PSO taking the second place. It can be concluded from these results that various dynamic PSO versions are certainly comparable to the classic gradient descent, and can be considered as an alternative.

Cooperative charged PSO showed the worst performance out of the five training algorithms tested. This can be due to the fact that neural networks are highly non-separable problems. Furthermore, the CCPSO showed to give significant improvements in high-dimensional spaces [10], and the dimensionality of the moving hyperplane problem may not be high enough to take advantage of the dimension-wise optimisation.

2) Dynamic Sphere: The dynamic sphere is a 3D problem that involves a rather complex spherically shaped decision boundary. Table IV displays the results obtained for this problem under the three different dynamic scenarios.

Under scenario I, which simulated gradual environment change, all PSO algorithms performed roughly the same, while gradient descent struggled to compete. As further visualised in Figure 1, gradient descent started better than the PSO, but from the first change to the end of the run struggled to recover. It took about 100 algorithm iterations for the PSO to find a minimum, but once it did, the swarm steadily tracked the minimum and successfully recovered from changes throughout the experiment. Only non-cooperative charged and quantum PSO are shown on the figure for the sake of clarity, since the cooperative version produced roughly the same results. As can be seen from Table IV, cooperation did not introduce any improvement once again.

Under scenarios II and III, all algorithms produced very similar results, again proving that dynamic PSO is a viable alternative to gradient descent.

3) Sliding Thresholds: Sliding thresholds is a 2D problem that has only linear decision boundaries. Although linear decision boundaries are trivial to learn, this problem is rather difficult to optimise due to the sliding window approach used to simulate the dynamic environment. As previously discussed in this paper, a dynamic environment is simulated by sliding a window over a large data set. If the data set is traversed in order, the classification of the data patterns will change every 1000 patterns. The previous two problems discussed deal with one decision boundary for every 1000 patterns, and the sliding window of N patterns, $N < 1000$, contains at least one decision boundary and at most two. In case of the sliding thresholds, there are two decision bound-

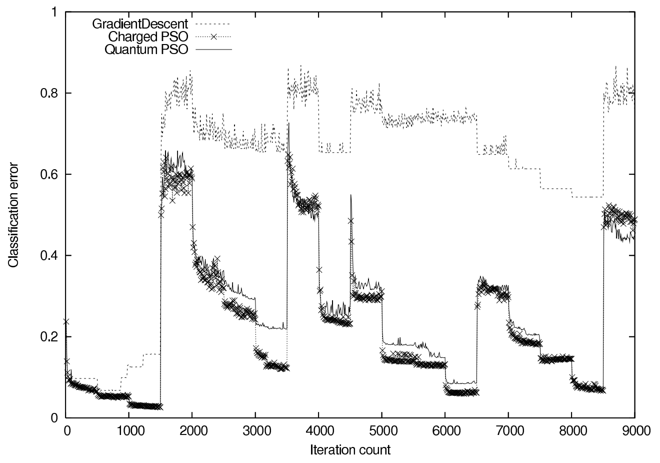

Fig. 3. Sliding Thresholds, Scenario II

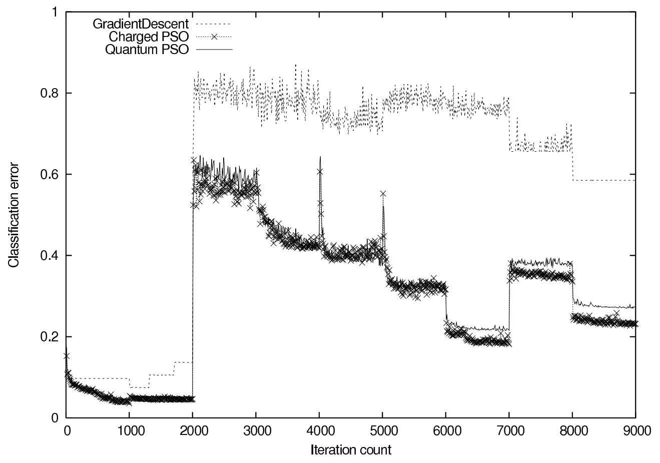

Fig. 4. Sliding Thresholds, Scenario III

aries for every 1000 data patterns, and, since every 1000 patterns of the data set two new boundaries are introduced, the sliding window contains at least two decision boundaries and at most four. Although linear boundaries are trivial, it can be problematic for the training algorithm to simultaneously detect two new boundaries. Furthermore, old boundaries and new boundaries may happen to be mutually exclusive, i. e. classify the same pattern into different classes, since new boundaries are generated randomly.

The mean fitness results obtained are shown in Table V, and Figures 2, 3, and 4 were drawn to better illustrate the performance of the training algorithms considered. Only the charged and quantum PSO are shown in the figures for the sake of clarity, since cooperative PSO produced almost identical results.

It can immediately be concluded from the table that the charged PSO performed better than the other algorithms under all dynamic scenarios. Quantum PSO was the second-best performer, and both cooperative versions of the charged PSO, third-best. Gradient descent produced an average classification error of more than 50% under all dynamic scenarios.

As can be seen in Figure 2, PSO handled gradual changes very well, consistently recovering throughout the algorithm run. Scenarios II and III were harder to adapt to. However, PSO searched the area effectively, and consistently found better solutions than that offered by classic gradient descent.

IV. Conclusion

The aim of this paper was to examine the applicability of dynamic PSO as a NN training algorithm in dynamic environments, as well as to compare the PSO performance to that of the standard gradient descent NN training algorithm in the dynamic context.

Experiments with different training algorithms were run on three different dynamic classification problems under three different dynamic scenarios. Analysis of the experimental results showed that dynamic PSO is indeed applicable to NN training in dynamic environments. PSO performed comparably well, and in some cases outperformed the classic NN training algorithm by a significant factor.

It was also observed that cooperation, although effective in dynamic environments, did not improve the accuracy of NN training for the problems considered. This may be due to the fact that only high-dimensional problems can take advantage of the dimension-wise optimisation that cooperation offers, and all the problems considered were relatively low-dimensional.

Further research will include a more elaborate study of various dynamic PSO algorithms applied to NN training. Various training algorithm properties not investigated in this paper, such as the speed of recovery after a change, will be taken into consideration. The applicability of PSO training to a wider range of dynamic problems will also be investigated.

References

[1] A. Tsymbal, “The problem of concept drift: denitions and related work,” Department of Computer Science, Trinity College Dublin, Ireland, Tech. Rep. TCD-CS-2004-15, April 2004.

[2] A. P. Engelbrecht and A. Ismail, “Training product unit neural networks,” *Stability and Control: Theory and Applications*, 1999.

[3] F. Van den Bergh and A. P. Engelbrecht, “Cooperative learning in neural networks using particle swarm optimizers,” *South African Computer Journal*, 2000.

[4] V. G. Gudise and G. K. Venayagamoorthy, “Comparison of particle swarm optimization and backpropagation as training algorithms for neural networks,” in *Proceedings of the 2003 IEEE Swarm Intelligence Symposium (SIS ’03)*, April 2003, pp. 110–117.

[5] T. M. Blackwell and P. J. Bentley, “Dynamic search with charged swarms,” in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2002, pp. 19–26.

[6] A. J. Carlisle, “Applying the particle swarm optimizer to non-stationary environments,” Ph.D. dissertation, Auburn University, Auburn, Alabama, USA, 2002.

[7] T. Blackwell and J. Branke, “Multiswarms, exclusion, and anti-convergence in dynamic environments,” *IEEE Transactions on Evolutionary Computation*, vol. 10, no. 4, pp. 459–472, 2006.

[8] X. Li and K. H. Dam, “Comparing particle swarms for tracking extrema in dynamic environments,” in *Proceedings of the Congress on Evolutionary Computation*, 2003, pp. 1772–1779.

[9] J. P. Coelho, P. B. De Moura Oliviera, and J. Boa Ventura Cunha, “Non-linear concentration control system design using a new adaptive pso,” in *Proc. of the 5th Portuguese Conference on Automatic Control*, 2002.

[10] A. Rakitianskaia and A. P. Engelbrecht, “Cooperative charged particle swarm optimiser,” in *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2008)*, 2008, pp. 933–939.

[11] S. Lawrence, A. C. Tsoi, and A. D. Back, “Function approximation with neural networks and local methods: bias, variance and smoothness,” in *Proceedings of the Australian Conference on Neural Networks (ACNN 96)*, 1996, pp. 16–21.

[12] M. Jinli and S. Zhiyi, “Application of combined neural networks in nonlinear function approximation,” in *Proceedings of the 3rd World Congress on Intelligent Control and Automation*, no. 2, 2000, pp. 839–841.

[13] J. Kennedy and R. Eberhart, “Particle swarm optimization,” in *Proc. of the IEEE International Joint Conference on Neural Networks*. IEEE Press, 1995, pp. 1942–1948.

[14] F. Van den Bergh, “An analysis of particle swarm optimizers,” Ph.D. dissertation, University of Pretoria, 2001.

[15] M. Clerc and J. Kennedy, “The particle swarm-explosion, stability, and convergence in a multidimensional complex space,” *IEEE Transactions on Evolutionary Computation*, 2002.

[16] A. P. Engelbrecht, *Fundamentals of Computational Swarm Intelligence*. John Wiley & Sons Ltd., 2005.

[17] F. Van den Bergh and A. Engelbrecht, “A cooperative approach to particle swarm optimization,” *IEEE Transactions on Evolutionary Computation*, 2004.

[18] R. C. Eberhart and Y. Shi, “Comparing inertia weights and constriction factors in particle swarm optimization,” in *Proc. of the Congress on Evolutionary Computation*, 2000.

[19] R. W. Morrison, “Performance measurement in dynamic environments,” in *GECCO Workshop on Evolutionary Algorithms for Dynamic Optimization Problems*, J. Branke, Ed., no. 5-8, 2003.