Swarm Intell (2012) 6:233–270
DOI 10.1007/s11721-012-0071-6

Training feedforward neural networks with dynamic particle swarm optimisation

A.S. Rakitianskaia · A.P. Engelbrecht

Received: 7 February 2012 / Accepted: 21 August 2012 / Published online: 8 September 2012

Abstract Particle swarm optimisation has been successfully applied to train feedforward neural networks in static environments. Many real-world problems to which neural networks are applied are dynamic in the sense that the underlying data distribution changes over time. In the context of classification problems, this leads to concept drift where decision boundaries may change over time. This article investigates the applicability of dynamic particle swarm optimisation algorithms as neural network training algorithms under the presence of concept drift.

Keywords Swarm intelligence · Particle swarm optimisation · Neural networks · Dynamic environments · Classification · Concept drift

1 Introduction

Efficient mining of continuous data streams in real time is an important real-life problem. Example applications of real-time data mining are spam filtering (Fdez-Riverola et al. 2007) and security applications such as anomaly and fraud detection (Liao et al. 2007; Thuraisingham 2008). An important difficulty with these real-world problems is that the underlying data distribution may change over time, causing the current empirical model to lose precision (Chu et al. 2004; Liao et al. 2007; Thuraisingham 2008). Temporal changes in the underlying data distribution are often referred to as *concept drift* in the literature (Schlimmer and Granger 1986; Tsymbal 2004).

Most existing approaches to handle concept drift do not deal with the process of concept learning, but rather with the way in which data is presented to the learner (Klinkenberg 2004;

A.S. Rakitianskaia
JINR LIT, Joliot-Curie 6, 141980 Dubna, Moscow region, Russia
e-mail: annar@jinr.ru

A.P. Engelbrecht (✉)
Department of Computer Science, University of Pretoria, 0002 Pretoria, South Africa
e-mail: engel@cs.up.ac.za

Last 2002; Widmer and Kubat 1996). This study offers a different perspective on addressing concept drift by focusing on the learning process of a single classifier, namely a feedforward neural network.

Neural networks (NNs) (Bishop 1995; Dreyfus 2005; Zurada 1992) are powerful mathematical models inspired by the human brain. A trained NN is capable of representing any nonlinear relationship between input and output data. Thus, an NN can carry out tasks such as pattern recognition, classification, and function approximation (Dreyfus 2005; Hassoun 1995; Patterson 1998; Rojas 1996). NNs have remained conservative towards the emerging field of optimisation in the presence of concept drift. It has been assumed that the standard NN training algorithms that employ gradient descent are implicitly dynamic (Ismail 2001; Kuncheva 2004; Rokach 2010), and if the NN fails to adapt to the drifting concepts, then restarting the training process would be the most efficient solution. In order to avoid retraining, redundancy in the form of ensemble classifiers has also been proposed (Street and Kim 2001; Tsymbal 2004; Tsymbal et al. 2008). The chances of obtaining at least one acceptable solution using ensemble classifiers are increased by training a number of separate NNs on the same problem over different time periods. However, the ensemble approach does not offer any training algorithm improvements to make each classifier aware of concept drift.

This study deals with classification problems only. In case of classification, concept drift yields changes in the decision boundaries that separate classes. The boundaries between classes may shift, new boundaries may appear, and old boundaries may become obsolete. The objective of classification algorithms applied to such dynamic environments is not only to approximate the decision boundaries, but also to detect changes in the decision boundaries and promptly adapt to them, which might entail dismissing the old boundaries entirely. To be applicable in dynamic environments, algorithms developed for static environments have to be changed in order to include these additional objectives.

An example of successfully adapting an algorithm originally developed for static environments to dynamic environments is particle swarm optimisation (PSO), a population-based optimisation technique that models social behaviour of a bird flock (Kennedy and Eberhart 1995; Poli et al. 2007). Due to the success of the PSO in static environments, numerous variants of PSO that cater for environment changes have been developed to date, including the simple restarting PSO (Eberhart and Shi 2001), the charged PSO inspired by electrostatic principles (Blackwell and Bentley 2002), and the quantum PSO inspired by the model of an atom (Blackwell and Branke 2006), amongst others.

NNs are widely used in real life (Cournane and Hunt 2004; Watanasusin and Sanguansintukul 2011; Zhao et al. 2011), and it is necessary to ensure that NNs can be effectively trained in dynamic environments or, in other words, under the presence of concept drift. PSO has been successfully applied to NN training before (Kennedy et al. 2001; Engelbrecht and Ismail 1999; Gudise and Venayagamoorthy 2003; van den Bergh and Engelbrecht 2000), and in this work the applicability of dynamic PSOs to NN training in the presence of concept drift is studied. The main focus of this work is on classification problems with dynamic decision boundaries, referred to as dynamic classification problems in the remainder of this text. The goal of this work is to provide a proof of concept that dynamic PSO algorithms can efficiently cope with concept drift. This is achieved by analysing the presented algorithms on artificial problems where the characteristics of the concept drift are well understood. In addition to the artificial problems, one real-world problem is included to show that the proposed dynamic PSO algorithms perform better than gradient-based training algorithms on a real-world, temporal problem.

The rest of the paper is organised as follows. Section 2 outlines the static PSO algorithm. Section 3 briefly discusses NNs and back propagation. Section 4 discusses concept drift and the dynamic PSOs used in this study. Section 5 discusses the existing approaches to NN training in the presence of concept drift. Section 6 presents the empirical study conducted. Section 7 provides a summary of the conclusions arrived at and lists possible future research directions.

2 Particle swarm optimisation

Particle swarm optimisation is a nature-inspired population-based optimisation technique. PSO operates on a set (referred to as a swarm) of particles, where every particle represents a candidate solution to the optimisation problem. For an n-dimensional optimisation problem, a particle is represented by an n-dimensional vector, $\mathbf{x}$, also referred to as the particle position. Every particle represents a candidate solution to the optimisation problem. The n-dimensional search space of the problem is the environment in which the swarm operates. In addition to a position within the search space, each particle possesses a velocity vector $\mathbf{v}$, which determines the step size and direction of the particle's movement. Social interaction is imitated by forming neighbourhoods within a swarm. Each particle stores its own best position found so far and can also query neighbouring particles for the best position as discovered by the neighbourhood thus far. PSO searches for an optimum by moving the particles through the search space. At each time step t, the position $\mathbf{x}_y(t)$ of particle y is modified by adding the particle velocity $\mathbf{v}_y(t)$ to the previous position vector:

$$\mathbf{x}_y(t) = \mathbf{x}_y(t-1) + \mathbf{v}_y(t) \tag{1}$$

The velocity vector determines the step size and direction in which the particle moves. The velocity update equation is given by

$$\begin{aligned} \mathbf{v}_y(t) = \omega \mathbf{v}_y(t-1) &+ c_1 \mathbf{r}_1 \otimes \big(\mathbf{x}_{\text{pbest},y}(t-1) - \mathbf{x}_y(t-1)\big) \\ &+ c_2 \mathbf{r}_2 \otimes \big(\mathbf{x}_{\text{nbest},y}(t-1) - \mathbf{x}_y(t-1)\big) \end{aligned} \tag{2}$$

where $\otimes$ refers to component-wise multiplication of vectors; ω is the inertia weight (Shi and Eberhart 1999), controlling the influence of previous velocity values on the new velocity; c_1 and c_2 are acceleration coefficients used to scale the influence of the *cognitive* (second term of Eq. (2)) and *social* (third term of Eq. (2)) components; $\mathbf{r}_1$ and $\mathbf{r}_2$ are vectors with each component sampled from a uniform distribution $U(0,1)$; $\mathbf{x}_{\text{pbest},y}(t)$ is the personal best of particle y or, in other words, the best position encountered by this particle so far; similarly, $\mathbf{x}_{\text{nbest},y}(t)$ is the neighbourhood best of particle y, or the best position found by any of the particles in the neighbourhood of particle y. Thus, each particle is attracted to both the best position encountered by itself so far, as well as the overall best position found by the neighbourhood so far. A maximum velocity $\mathbf{V}_{\max}$ (Poli et al. 2007; Shi and Eberhart 1998) is sometimes used to limit (or clamp) particle velocity in every dimension. Velocity clamping is done to prevent particles from traversing the search space too fast, since unreasonably large steps prevent particles from exploiting good regions. $\mathbf{V}_{\max}$ is enforced by restricting $\mathbf{v}_y(t)$ per dimension:

$$v_{yl}(t) = \begin{cases} V_{\max,l} & \text{if } v_{yl}(t) > V_{\max,l} \\ -V_{\max,l} & \text{if } v_{yl}(t) < -V_{\max,l} \\ v_{yl}(t) & \text{otherwise} \end{cases} \tag{3}$$

Various PSO neighbourhood topologies have been proposed in the literature and applied in practice. A particle's neighbourhood is determined topologically rather than spatially, meaning that the distance between particles is determined by particle indices and not by the actual position in the search space (Kennedy 1999). The structure and size of the neighbourhood determines the way in which information is shared between the particles. Thus, choosing an appropriate neighbourhood topology is crucial for the overall efficiency of the optimisation process.

The Von Neumann topology was first introduced by Kennedy and Mendes (2002). This neighbourhood topology connects the particles in a grid-like structure such that every particle connects to its four immediate neighbours. The Von Neumann topology can be visualised as a square lattice, the extremities of which are connected. Peer et al. (2003) showed that the Von Neumann neighbourhood topology maintains swarm diversity due to the fact that the influence of a single particle propagates through the structure slowly, thus making it harder for a single particle to dominate the swarm. It was empirically shown by Li and Khanh (2003) that PSO utilising the Von Neumann topology (sometimes also referred to as the *fine-grained PSO*) performs well in dynamic environments. The fine-grained PSO managed to outperform PSO with other information sharing strategies on a selection of high-dimensional dynamic problems (Li and Khanh 2003). The Von Neumann neighbourhood topology was therefore used in all experiments conducted for this study.

3 Neural networks

A neural network is a simple mathematical model inspired by the structure of the brain. NNs are able to carry out tasks such as pattern recognition, classification, and function approximation (Dreyfus 2005; Hassoun 1995; Patterson 1998; Rojas 1996). An NN is essentially a collection of interconnected neurons aligned in layers. It was theoretically proved in Lawrence et al. (1996); Jinli and Zhiyi (2000) that an NN can represent any nonlinear mapping between the input space and the target space, provided that the hidden layer has enough neurons. The NN itself is just a structure capable of representing a function, requiring to be trained on a problem in order to learn the mapping between input space and output space. Learning 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 targets. Given a set of data patterns with known targets, a NN is trained to learn the mapping between the inputs and the targets. A trained NN is then capable of accurately approximating outputs for data patterns it has never seen before. Such ability is known as the ability to generalise. The generalisation ability of an NN can deteriorate if the data set is not representative of the mapping to be learned, contains noise if the NN is trained for too long and if there are too many weights in the NN. The deterioration of the generalisation ability of the NN due to learning unnecessary information is known as overfitting. An NN that cannot generalise has no practical use, since such NN will only be able to predict the outputs for the previously seen patterns. For a more extensive discussion of overfitting, refer to (Bishop 1995; Blackwell and Chen 2009).

The rest of this section is dedicated to NN training algorithms applied in this study. Section 3.1 discusses back propagation. Section 3.2 explains how PSO can be applied to NN training and outlines the advantages of PSO compared to back propagation.

3.1 Back propagation

The most commonly used and popular NN training algorithm is back propagation (BP) (Werbos 1974), which uses gradient descent to iteratively adjust weight values in order to decrease the resulting NN output error. The stopping criteria of BP is usually defined by a maximum number of algorithm iterations, or epochs, by setting a threshold on the classification error, or by setting a threshold on the mean squared error (MSE) produced by the generalisation set.

BP is a gradient descent algorithm, and its major disadvantage is susceptibility to convergence on local minima. Another disadvantage of gradient-based approaches is the dependence on the starting point of the search, which would be the initial weights in case of NNs.

3.2 Particle swarm optimiser training

In order to train an NN using PSO, an appropriate objective function has to be defined, which is usually simply the MSE. An appropriate representation of candidate solutions also has to be determined. Each particle is used to represent a candidate solution, which is a vector of all of the weights and biases of an NN. Every element of a particle represents a single weight or bias, using floating-point numbers. Therefore, each particle has a dimension equal to the total number of weights and biases in the NN. The PSO is then used, as discussed in Sect. 2, to adjust the weight and bias values (using the particle velocity and position updates) such that the given objective function is minimised.

Recent research has shown PSO to be an effective NN training algorithm (Engelbrecht and Ismail 1999; Gudise and Venayagamoorthy 2003; Kennedy et al. 2001; Mendes et al. 2002; van den Bergh and Engelbrecht 2000). PSO outperformed BP on a selection of classification, function approximation, and prediction problems. The advantages that PSO offers in comparison with BP are:

- weaker dependence on the initial weight values, since multiple starting points (i.e., particles) are used in the search process,
- derivative information of the activation functions and the error function is not used, thus the activation functions and the error function do not have to be differentiable, and
- more robust on rugged surfaces, since population-based search is less prone to premature convergence on local minima than back propagation (Mendes et al. 2002).

The major disadvantages of PSO, as compared to BP, are slower speed of convergence and more algorithm parameters to tune before optimal performance can be expected.

4 Dynamic environments

This section focuses on dynamic classification problems, also referred to as classification problems with concept drift, and the relevant dynamic optimisation algorithms. Section 4.1 discusses the phenomena of concept drift and its properties. Section 4.2 lists existing approaches to concept drift. Section 4.3 discusses the dynamic PSO algorithms applied to train NNs for problems with concept drift in this study.

4.1 Concept drift

Mining continuous data sets is an important problem that cannot be solved without the aid of computers. For example, terabytes of data have to be processed rapidly for the purpose of security (Liao et al. 2007; Thuraisingham 2008), and thousands of e-mails have to be classified every minute as either spam or legitimate correspondence (Fdez-Riverola et al. 2007). Such

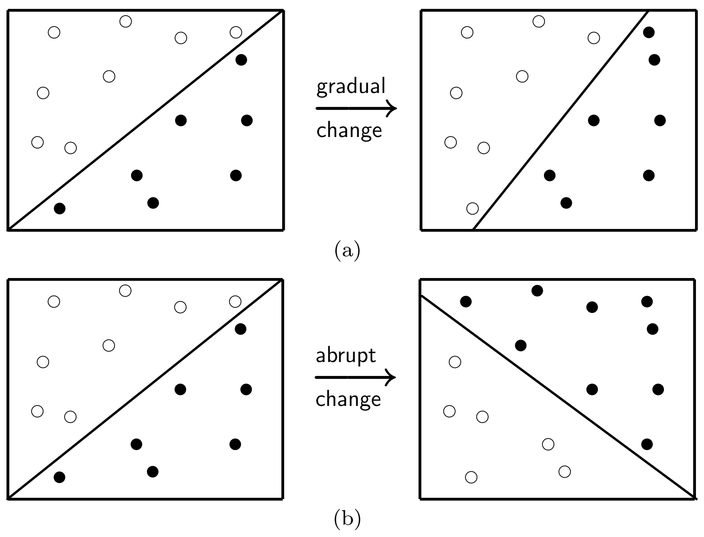


Fig. 1 Spatial severity applied to concept drift. (**a**) Gradual decision boundary change. (**b**) Abrupt decision boundary change

applications require real-time solutions, and real-life environments are not static: spam patterns change, and bots modify their strategies (Fdez-Riverola et al. 2007; Liao et al. 2007; Thuraisingham 2008). Temporal properties introduce extra complexity into any problem, since a solution once found will have to be adapted every time a temporal property changes (Nickabadi et al. 2012). In case of classification, the underlying data distribution may change over time, causing changes in the decision boundaries. Changes in the decision boundaries will yield changes in the target concepts, that is, the mapping between inputs and targets will change. This phenomena is referred to as concept drift (Schlimmer and Granger 1986).

The term "concept drift" belongs to the field of data mining and refers to drifting concepts as observed in large data sets and continuous data sets over time. In case of classification, drifting concepts imply changes in the decision boundaries that separate classes. The boundaries between classes may shift, new boundaries may appear, and old boundaries may become obsolete. Concept drift was shown to be present in almost every large data set (Tsymbal 2004). The severity of concept drift may vary from gradual changes, where the decision boundaries change by small offsets, to abrupt changes, where the old boundaries are abruptly replaced by new boundaries (Tsymbal 2004). This property of concept drift is referred to as spatial severity. Figure 1 illustrates spatial severity of changes as applied to concept drift. Decision boundaries may change continuously, at regular time intervals, or unpredictably. This property of concept drift is referred to as temporal severity, or frequency of change.

Concept drift complicates the process of concept learning by the NN, because the learned concepts become obsolete as the actual concepts drift, requiring the learned model to be revised. If decision boundaries change over time, the NN will have to detect and track such changes in order to update the learnt model accordingly. Different combinations of temporal severity and spatial severity result in different types of dynamic environments, ranging from dynamic environments exhibiting infrequent gradual changes, to dynamic environments exhibiting frequent abrupt changes. Frequent abrupt changes are the hardest to track and adapt to, since optimisation algorithms are required to make significant adjustments in a short period of time.

4.2 Optimisation in the presence of concept drift

Most existing approaches to handle concept drift do not deal with the process of concept learning, but rather with the way in which data is presented to the learner (Klinkenberg 2004; Last 2002; Widmer and Kubat 1996). Three major categories of concept drift handling techniques can be distinguished, namely instance selection, instance weighting, and ensemble learning (Tsymbal et al. 2008).

In a dynamic environment, the learner can never assume the current predictive model to be final. This implies that the learner must always try to keep the model up to date with the changing environment. In case of data mining in the presence of concept drift, up to date concepts can only be learnt from up to date data, obtained by instance selection (Last 2002; Widmer and Kubat 1996). The goal of the learner is to discover concepts, and if data accurately represents the existing concepts, the learner's success will depend entirely on the learner's ability to train and generalise. The simplest form of instance selection is windowing, implemented by fixing the number of instances in the training set and dismissing the oldest instances as new instances arrive (Widmer and Kubat 1996). More complex forms of instance selection that delete noisy, irrelevant and redundant instances have also been developed (Delany et al. 2005; Last 2002). The windowing approach to instance selection applied in this study is described in more detail in Sect. 6.

Instance weighting is sometimes used instead of instance selection with learning algorithms that have the ability to process weighted instances (Klinkenberg 2004). An example of such algorithms are support vector machines (SVMs) (Cortes and Vapnik 1995). Instances are weighted according to their age and relevance, and most recent and most relevant instances contribute the most to the learning process. This study, however, deals with NN training, to which instance weighting is not applicable.

A more advanced approach that deals with the classifiers rather than the training data is ensemble learning (Rokach 2010; Tsymbal 2004). With ensemble learning, a selection of classifiers is combined. Each classifier maintains a separate concept description. The quality of concept descriptions provided by each classifier is measured, and the best-performing classifier has the most influence on the final classification (Tsymbal 2004). Worst performing classifiers can be dynamically replaced by new classifiers, which start learning the concept from scratch. The chances of obtaining at least one acceptable concept description are increased by training a number of separate classifiers on the same problem over different time periods. Much research on handling concept drift using classifier ensembles was done (Chu et al. 2004; Rokach 2010; Street and Kim 2001; Tsymbal 2004; Tsymbal et al. 2008). However, the ensemble approach treats individual classifiers as black boxes and does not look into the learning process of a classifier. Ensemble classifiers offer no training algorithm improvements to make each classifier more adaptive and flexible in dynamic environments, assuming redundancy to be the only effective solution. Redundancy can indeed be effective. However, the performance of ensemble classifiers can be further improved by improving the performance of each single classifier (an NN in this study) by adapting the learning process, or, in other words, the training algorithm, to the drifting concepts.

In contrast to the aforementioned concept drift handling techniques, this study deals with the learning process of a single classifier, and the chosen classifier is a feedforward NN. Dynamic NN training algorithms are suggested, and the dynamic properties of back propagation are studied.

4.3 Dynamic particle swarm optimisation

In general, dynamic optimisation algorithms require mechanisms to detect changes and to respond to these changes. The next subsections discuss how PSO algorithms implement such mechanisms.

4.3.1 Change detection

In order to respond to a change in the environment, the change has to be detected by the PSO. Change detection is usually accomplished by making use of a *sentry*, which is either a dedicated particle or a fixed point in the search space (Carlisle and Dozier 2000, 2002). The only difference between normal particles and sentry particles is that the sentry particles keep a record of the quality of their previous positions. In the beginning of each iteration, sentry particles are re-evaluated, and if the difference between the previous and the new objective function value exceeds a certain threshold, it can be assumed that a change has occurred. The number of sentry particles to use in order to efficiently detect changes is problem dependent.

4.3.2 Response to the change

Standard PSO faces the following problems when optimisation in dynamic environments is required:

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 (Engelbrecht 2005) and thus provide the swarm with misleading information instead of leading the search towards an optimum.
2. *Loss of swarm diversity*: It was formally proved (Clerc and Kennedy 2002; van den Bergh 2002) that with a standard PSO, the swarm will gradually lose diversity from iteration to iteration, until all particles converge on a weighted average of the personal best and global best positions. Once converged, PSO will not explore any longer, because particle velocities, according to Eq. (2), will tend to zero as the distance between the current and the global best position, as well as the distance between the current and the personal best position decrease. A converged PSO has no exploration capabilities and will not be able to adapt to an environment change (Engelbrecht 2005).

A number of PSO variations have been developed, differing in the way the above issues are addressed. A review of dynamic PSO algorithms used in this study is given below.

Reinitialising PSO The reinitialising PSO approaches the aforementioned problems in a simple, naive manner. The outdated memory issue is addressed by re-evaluating particle positions, as well as the stored global and personal best positions. Diversity of the swarm is boosted by means of reinitialising the positions, velocities and personal best positions of a percentage of particles. The particles to be reinitialised are randomly selected. The disadvantage of this approach is partial loss of knowledge about the search space due to particle reinitialisation (Hu and Eberhart 2002). The reinitialisation ratio is problem dependent and should be chosen empirically. For the purpose of this study, two versions of the reinitialising PSO were used: normal reinitialising PSO (RPSO), where the reinitialising ratio was empirically chosen, and reinitialise-all PSO (RAPSO), where the entire swarm was reinitialised after every environment change. The latter was used to determine whether a complete restart of the PSO is an efficient approach to classification problems with concept drift.

Charged PSO The charged PSO (Blackwell and Bentley 2002) is inspired by 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 cannot be negative and does not change during algorithm execution. Charged particles repel from one another if the distance between them is small enough. This prevents charged particles from converging to a single point, thus facilitating exploration and addressing the diversity loss problem. Repelling forces are introduced by adding an acceleration term to the standard velocity equation (refer to Blackwell and Bentley (2002)). Acceleration is inversely proportional to the distance between the charged particles, and the farther two charged particles are from each other, the weaker they will repel. Thus, repelling forces maintain swarm diversity without yielding divergent behaviour.

The problem of outdated swarm memory is addressed by re-evaluating the quality of each particle in the swarm, the personal best of each particle, and the global best of each neighbourhood, whenever a change occurs.

Quantum PSO The quantum PSO (Blackwell and Branke 2006) is inspired by 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 trajectory dynamics, but by a probability distribution instead. A percentage of particles in the 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 completely randomised at each iteration, the swarm does not 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.

The problem of outdated memory is again addressed by complete re-evaluation of the swarm memory.

5 Neural networks in dynamic environments

Back propagation uses gradient descent to adjust NN weight values (Werbos 1974). The algorithm minimises the objective function by following its steepest slope. In order to visualise the behaviour of back propagation in a dynamic environment, it is important to remember that the objective function being optimised is the error function of the NN, and not the mapping between inputs and targets as represented by the data set. This mapping is thus the context in which the error function exists, and changes in the mapping, or, in other words, in the environment, yield changes in the error function. A change in the environment may yield an increase in NN error, since the current weight vector would no longer accurately represent the environment. An increase in NN error implies that the current position is not necessarily an optimum anymore, thus gradient descent may resume moving in the negative gradient of the objective function. This automatic response to environment changes makes back propagation an implicitly dynamic training algorithm.

Implicit dynamism, however, does not eliminate the premature convergence problem. The error function hypersurface may have flat regions where gradient descent is inefficient, and local minima where the algorithm may become stuck (Fogel et al. 1990; Gallagher 2000;

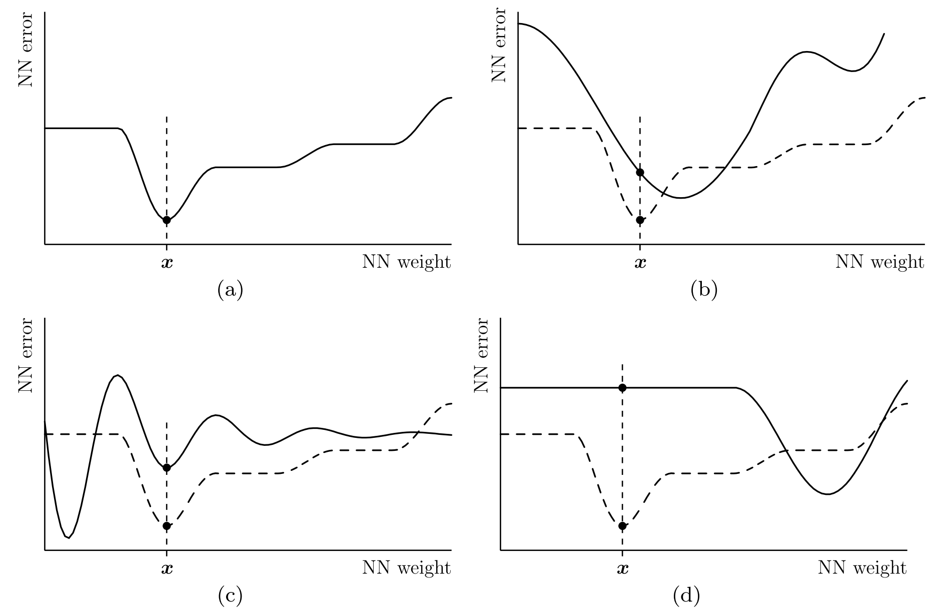


Fig. 2 Back propagation in dynamic environments. (**a**) Error function before environment change, x is the current position. (**b**) Current position becomes a position on the slope after environment change. (**c**) Current position becomes a local minimum after environment change. (**d**) Current position becomes a position on a plateau after environment change

Gudise and Venayagamoorthy 2003; Hush et al. 1992). When the error surface changes due to a change in the environment, the current position of the weight vector may happen to map to a region of the changed error surface that is hard to optimise, yielding poor adaptation to the change. A few examples of possible scenarios are given below.

Figure 2(a) illustrates the error function before a change in the environment. For the sake of clarity, a one-dimensional space is used, where x refers to the current position of the NN in the weight space. Figure 2(b) illustrates a scenario where a change in the environment causes the current position to be on the slope of the error function. Under such scenario, back propagation exhibits implicit dynamism and will find the new optimum by following the error function slope in the downhill direction. Figure 2(c) illustrates a scenario where the changed error function causes the current position to become a local minimum. Back propagation will not be able to escape the local minimum and will therefore fail to locate the global minimum. Figure 2(d) illustrates a scenario where the current position becomes a position on a flat region after the environment change, once again preventing gradient descent from discovering the new global optimum.

Thus, even though implicit dynamism of back propagation cannot be altogether denied, back propagation should not be relied upon as an ultimate dynamic training algorithm, since a number of dynamic scenarios exist under which back propagation will fail to either detect or track the changes.

The success of back propagation is also highly dependent on the initial set of weights (Fogel et al. 1990; Gudise and Venayagamoorthy 2003; Porto and Fogel 2002): choosing a good starting point is crucial for algorithm convergence. In the context of dynamic environments, the surface of the error function changes, and the success of adaptation after each change also depends on the current weight vector. Thus, the algorithm's success becomes dependent not only on the initial weights, but also on the current weights.

This study evaluates the implicit dynamism of back propagation. Four variants of back propagation were applied on a selection of dynamic classification problems: standard back propagation (BP), standard back propagation with stagnation detection (SBP), reinitialising back propagation (RBP), and reinitialising back propagation with stagnation detection (SRBP). RBP applied the standard BP algorithm between environment changes and completely reinitialised NN weights and biases whenever an environment change occurred. For the purposes of this article, environments changed at predetermined intervals to eliminate the need for change detection strategies. RBP was used to test the hypothesis, made in Chu et al. (2004), Rokach (2010), Street and Kim (2001), and Tsymbal (2004), that restarting back propagation after environment changes is an efficient approach to NN training in the presence of concept drift. SBP and SRBP operated exactly like BP and RBP, respectively, but with the added functionality of stagnation detection: NN weights and biases were completely reinitialised whenever the MSE failed to decrease for five consecutive algorithm iterations. Stagnation detection was done at the end of each epoch. Stagnation detection was applied to BP algorithms in order to improve their competitiveness by alleviating their natural tendency to converge prematurely. Stagnation detection also provides the BP algorithms a chance to restart the search for optimal weights if stagnation is detected before the next environment change. The best weight set at the point of stagnation is stored, and BP restarts to try and find a better solution before the next environment change. When the environment changes, the best solution between the stored solution and the current solution is used.

The pitfalls described above, which are mainly due to the use of gradients to direct search trajectories, indicate that back propagation may provide suboptimal solutions under certain dynamic scenarios. Since PSO does not make use of gradients and has been shown to be less susceptible to the disadvantages caused by gradients when used to train NNs in static environments, this study proposes that dynamic PSOs be applied to train NNs in the presence of concept drift.

6 Empirical analysis

This section provides a description of the experimental procedure followed and experimental results obtained for this study. This study hypothesises that dynamic PSO algorithms can be applied to efficiently train NNs in the presence of concept drift. To test this hypothesis, a number of dynamic PSO algorithms were applied to train NNs on a selection of dynamic classification problems, namely the reinitialising PSO (RPSO), the reinitialise-all PSO (RAPSO), the charged PSO (CPSO), and the quantum PSO (QPSO). The research field of dynamic PSO is still relatively young, and no standard or optimal approach to optimisation in dynamic environments with PSO has been identified yet (Duhain 2011). The algorithms listed above were chosen based on their relative popularity. The RPSO was chosen as the most natural, naive way of adapting the standard PSO to dynamic environments. The RAPSO was chosen to test whether a complete restart of the search is an efficient approach to concept drift. The CPSO and the QPSO were chosen due to the relatively solid theoretical and empirical research behind them that showed these algorithms to be effective on a selection of dynamic optimisation problems (refer to Blackwell and Branke (2006), Blackwell and Bentley (2002), Blackwell (2005), Li et al. (2008), Rakitianskaia and Engelbrecht (2008), and Rakitianskaia and Engelbrecht (2009)). The performance of the dynamic

PSO algorithms is compared to the performance of BP, SBP, RBP, and SRBP, and the behaviour of these algorithms in the presence of concept drift of varying spatial and temporal severity is investigated on five different dynamic classification problems.

This experimental study has the following three goals:

- To provide a proof of concept that dynamic PSO algorithms can efficiently cope with concept drift in training NNs as classifiers.
- To evaluate the performance of the dynamic PSO algorithms in comparison with that of BP algorithms.
- To determine whether restarting an algorithm is more efficient than algorithms that continuously track changing boundaries.

The rest of the section is structured as follows: Section 6.1 describes the experimental procedure followed. Performance measures are described in Sect. 6.2. Section 6.3 discusses the data sets used. Experimental results are presented in Sect. 6.4.

6.1 Experimental procedure

This section describes the experimental procedure followed. Section 6.1.1 describes how problems with concept drift were simulated. Section 6.1.2 provides a description of the parameter optimisation process.

6.1.1 Simulating concept drift

This study used one real-life data set and four data sets that were artificially generated to simulate dynamic classification problems of varying dimensionality and decision boundary shape. Although concept drift is present in most large enough real-life data sets (Tsymbal 2004), measuring the extent and nature of concept drift in such data sets is problematic. Artificial data sets, on the other hand, provide knowledge of the specific concept drift properties, which allows one to derive more precise conclusions from the empirical results. The process of generating a data set with concept drift applied in this study is described below.

It is assumed for all problems that a process of temporal labelling of training patterns exists, that is, training patterns are labelled as these patterns become available over time. Note that old patterns are not relabelled, as these patterns simply constitute stale data.

M points were randomly chosen from the specified domain. M data patterns were then obtained by assigning a target classification to every input vector according to current problem-specific decision boundaries. The boundaries were updated N times, thus simulating N environment changes. After every such change, the target classification of every pattern, $p = 1, \dots, p_M$, was updated accordingly, and the updated M patterns were appended to the previous M patterns. Thus, the total number of data patterns in a complete data set is

$$P = M + M \cdot N$$

Classification problems with concept drift were simulated by sliding a window over such a data set. In this study, the size of the window was set to M. Thus, at every iteration the NN was presented with data patterns which represented a complete set of M points, without repetitions (i.e., only a single instance of every point was present at every iteration). The data patterns inside the window were split into two subsets for training and generalisation purposes: 80 % of the patterns were used as a training set, D_T, and the other 20 % were used as a generalisation set, D_G. Since the aim was to simulate decision boundaries that change

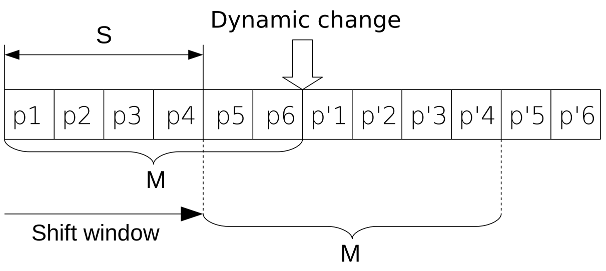


Fig. 3 Introducing new decision boundaries by window shifts

over time, the original data set was not shuffled to preserve the pattern order. The patterns were only shuffled inside the window before being split into D_T and D_G to prevent NNs from learning the pattern order instead of the classification boundaries.

Shifting the window by S patterns implies discarding the first S patterns from the window and appending the next S patterns from the data set to the window. The window step size, S, controls the spatial severity of changes: changes become more drastic for larger values of S, since a lot of new information is added while a lot of previously valid information is discarded. If the decision boundaries change every M patterns in the data set and the window size is equal to M, a shift by $S < M$ patterns may introduce *new decision boundaries* into the window while still keeping the data patterns classified according to the previous decision boundaries inside the window. An example to illustrate this process is shown in Fig. 3. Here, $M = 6$ and $S = 4$. When the window shifts, four patterns, $\{p_1, p_2, p_3, p_4\}$, are discarded, and new patterns $\{p'_1, p'_2, p'_3, p'_4\}$ are added, while p_5 and p_6 remain in the window. Thus the shifted window contains patterns representing the environment both before and after the dynamic change, or, in other words, the window contains patterns representing both the old decision boundaries and the updated decision boundaries. As the window slides along the data set, more patterns classified according to the previous decision boundaries are replaced by the patterns classified according to the current decision boundaries, until the previous boundaries are completely discarded. This implies that the training algorithm will often have to deal with more than one decision boundary within the data set, which makes dynamic adaptation more challenging for the training algorithms. However, it is not always possible to distinguish between stale and up-to-date data in a real-life context, and thus the ability of a training algorithm to handle the presence of stale data is a desirable characteristic.

As described in Sect. 4, two major characteristics of a dynamic problem are spatial severity of changes and temporal severity of changes. In order to provide a representative coverage of the existing types of concept drift, different combinations of spatial severity and temporal severity were simulated, resulting in a number of different dynamic scenarios. Every dynamic scenario is characterised by two variables:

- The step size, S, which refers to the number of patterns by which the window shifts in order to simulate an environment change. This attribute determines the abruptness of the environment change or, in other words, the level of spatial severity.
- The number of algorithm iterations, F, that the current training algorithm is allowed to run before the window shifts. This attribute controls the frequency of changes or, in other words, the level of temporal severity.

Since an exhaustive evaluation of the considered training algorithms under all possible combinations of values of the above two variables is practically infeasible, discrete ranges of values were considered for both variables on every problem. All possible combinations of values in these discrete ranges were considered. The discrete ranges were problem-specific and are reported later in this section.

For every scenario considered, every training algorithm had to traverse the entire data set. Since both the step size and the frequency of changes varied from scenario to scenario, every scenario required a different total number of iterations to traverse the entire data set. The number of iterations is given by

$$T = F * \frac{P - P_w}{S} + F \tag{4}$$

where F is the number of iterations on a window between the window shifts (i.e., change frequency), P is the total number of patterns in the data set, P_w is the window size, and S is the step size. The collective mean error results and the corresponding standard deviation values reported in this work were obtained after the number of iterations as given by Eq. (4). The window size was problem-dependent, and the values of both F and S were determined by the scenario in use.

6.1.2 Parameter tuning

When comparing a set of algorithms on a specific problem, good values for the control parameter values of the algorithms are usually found in order to ensure that no algorithm is at an advantage. Since good values for control parameters are problem dependent, good values were found for each algorithm, for each problem. Note that, for each problem, control parameters were tuned over all the environment changes, and not for each individual environment change. Found values were therefore those that resulted in average good performance over all environment changes. A single dynamic scenario was used in the process of parameter tuning for each problem. The choice of scenarios used for parameter tuning was arbitrary, and the found parameter values cannot be considered good for all dynamic scenarios. However, since the same control parameter tuning process was applied to all the algorithms, no algorithm received any preferential treatment. The efficiency of parameter tuning in dynamic environments remains open for discussion, since any once-found parameters may be expected to become suboptimal over time. Research into control parameter tuning in dynamic environments is currently being conducted by Leonard and Engelbrecht (2012).

An iterative approach to algorithm parameter tuning was used. Algorithm parameters were optimised one at a time. For each parameter, the algorithm was tested under a selected range of possible values for this parameter, while the other parameters remained fixed. In order to keep the optimisation process statistically sound, 30 independent runs were conducted for every value in the chosen discrete range. For every simulation, the data inside the sliding window was randomly divided into training and generalisation sets (the ratio of data to be used for generalisation was set to 20 %). After every environment change and the resulting change of sliding window content, the data was again randomly sub-sampled. Repeating random sub-sampling for 30 simulations is in effect an implementation of repeated random sub-sampling—a cross-validation technique, sometimes also referred to as Monte Carlo cross-validation (Picard and Cook 1984; Shao 1993; Xu and Liang 2001). In contrast to a full k-fold or leave-one-out cross-validation procedure, repeated random sub-sampling has been shown to be asymptotically consistent (Shao 1993), resulting in more pessimistic predictions of the test data compared to k-fold and leave-one-out cross-validation. The predictions of the test data gave a realistic estimation of the predictions of external validation data (Xu and Liang 2001).

Table 1 Parameter ranges for BP algorithms

Parameter	Range
NN weight initialisation range w_{int}	$\{[-1, 1], [-2, 2], [-3, 3], [-4, 4], [-5, 5]\}$
Learning Rate η	$\{0.1, 0.3, 0.5, 0.7, 0.9\}$
Momentum α	$\{0.1, 0.3, 0.5, 0.7, 0.9\}$

Table 2 Parameter ranges for the dynamic PSOs

Parameter	Range
NN weight initialisation range w_{int}	$\{[-1, 1], [-2, 2], [-3, 3], [-4, 4], [-5, 5]\}$
Swarm Size S_P	$\{15, 20, 30, 50\}$
$V_{\max}$	$\{0.1, 0.5, 1, 2, 5, 10, 20, +\infty\}$
RPSO: Reinitialisation Ratio ϱ	$\{0.25, 0.5, 0.75, 1.0\}$
CPSO: Charge Magnitude Q	$\{0.1, 0.3, 1, 5, 10, 20\}$
QPSO: Radius R	$\{1, 1.5, 2, 3, 5, 10\}$

The parameter value yielding the lowest average generalisation errors for the current parameter being tuned was subsequently chosen as the value to be used, and tuning proceeded to the next parameter. For tuning the remaining parameters, all the parameters for which good values were already found were fixed to their good values. The discrete parameter value ranges used for the BP algorithms are listed in Table 1, and the discrete parameter value ranges used for the dynamic PSO algorithms are listed in Table 2. Good parameter values as obtained for each problem are listed in Table 3. Parameters were tuned in the order as presented in Table 3. All NNs used a single hidden layer. The number of hidden units used for each problem is given in the corresponding problem descriptions in Sect. 6.3. For all dynamic PSO experiments, the inertia weight was set to 0.729844, while the values of the acceleration coefficients were set to 1.49618. This choice is based on Eberhart and Shi (2000), where it was shown that such parameter settings facilitate convergent behaviour. Although preservation of diversity is vital in the context of dynamic environments, convergent behaviour is still necessary, since a solution must be found in between environment changes. Initial particle velocities were set to 0. For the CPSO and the QPSO, 50 % of the particles were labelled charged and quantum, respectively. All PSO algorithms used the Von Neumann neighbourhood topology, where the topology dimensions were derived from the size of the swarm: the sides of the lattice were chosen such that the difference between the sizes was minimal.

For the sake of avoiding bias, two datasets were generated for every artificial problem as described in the next section, such that both datasets represented the same decision boundaries and the same environment changes. One data set was then used for parameter tuning using Monte Carlo cross-validation, and the other data set was used for NN training and testing. For the real-life data set problem, 20 % of the data was randomly sampled from the data and used for parameter tuning with Monte Carlo cross-validation. The remaining 80 % of the data was subsequently used for training and validation.

Table 3 Good parameter values (SEA refers to the streaming ensemble algorithm concept problem)

		SEA concepts	Moving hyperplane	Dynamic sphere	Sliding thresholds	Electricity pricing
BP	w_{int}	[−5, 5]	[−2, 2]	[−3, 3]	[−1, 1]	[−1, 1]
	η	0.1	0.1	0.1	0.1	0.1
	α	0.7	0.7	0.3	0.1	0.7
SBP	w_{int}	[−2, 2]	[−1, 1]	[−2, 2]	[−3, 3]	[−3, 3]
	η	0.1	0.3	0.3	0.1	0.1
	α	0.5	0.9	0.3	0.1	0.9
RBP	w_{int}	[−5, 5]	[−2, 2]	[−3, 3]	[−1, 1]	[−1, 1]
	η	0.1	0.1	0.1	0.1	0.1
	α	0.7	0.7	0.3	0.2	0.7
SRBP	w_{int}	[−1, 1]	[−1, 1]	[−3, 3]	[−3, 3]	[−3, 3]
	η	0.5	0.3	0.7	0.3	0.3
	α	0.7	0.9	0.1	0.5	0.9
RPSO	w_{int}	[−3, 3]	[−1, 1]	[−1, 1]	[−1, 1]	[−1, 1]
	s_P	50	30	50	50	30
	$V_{\max}$	0.5	0.5	1	1	2
	ϱ	0.25	0.5	0.5	0.75	0.25
RAPSO	w_{int}	[−3, 3]	[−1, 1]	[−1, 1]	[−1, 1]	[−1, 1]
	s_P	50	30	50	50	30
	$V_{\max}$	0.5	0.5	1	1	2
CPSO	w_{int}	[−3, 3]	[−2, 2]	[−5, 5]	[−1, 1]	[−1, 1]
	s_P	30	50	50	50	30
	$V_{\max}$	0.5	0.5	0.1	2	5
	Q	0.3	20	5	20	0.1
QPSO	w_{int}	[−1, 1]	[−1, 1]	[−1, 1]	[−1, 1]	[−3, 3]
	s_P	50	30	20	50	50
	$V_{\max}$	0.5	0.1	0.1	2	5
	R	1.5	1	1.5	2	1.5

6.2 Measuring neural network performance in dynamic environments

In dynamic environments, the algorithms must not only find an optimum, but also detect changes in the optimum, track the optimum, and locate new better optima as they appear. Clearly, standard performance measures that reflect the current algorithm state only do not provide any information regarding the change detection and response to the change exhibited by the algorithm, and thus cannot be used in dynamic environments. This is why Morrison (2003) suggested that a representative performance measure in a dynamic environment should reflect algorithm performance "across the entire range of landscape dynamics". Morrison (2003) proposed that the collective mean objective function value, or the average over all previous objective function values, be used as given by

$$f_{\text{mean}}(T) = \frac{\sum_{t=1}^{T} f(t)}{T} \tag{5}$$

where T is the total number of iterations, and $f(t)$ is the best objective function value obtained after iteration t. The collective mean objective function value represents the entire algorithm performance history and hence gives an indication of the adaptive properties of the algorithm. This measure allows for convenient statistical comparison between algorithms and does not depend on any additional knowledge about the search space such as the location of the global optimum.

When referring to solution quality in the context of NNs, it should be clarified what exactly is meant by this term. In the current work, the MSE calculated over the data set during each epoch was used to measure the quality of each candidate solution, that is, NN. This measure reflects the NN's ability to recognise the training patterns for the training MSE (E_T) and the NN's ability to generalise for the generalisation MSE (E_G). It was theoretically shown by (Wan 1990) that minimisation of the MSE consequently minimises the probability of misclassification. Thus, MSE does not lose its meaning when classification problems are considered. The collective mean MSE was used as a performance measure in all the experiments conducted in this study. All reported E_T and E_G values were calculated according to Eq. (5), where $f(t) = E_T(t)$ for the training error, and $f(t) = E_G(t)$ for the generalisation error.

A study of overfitting is outside the scope of this work, thus no measures were taken to prevent overfitting of training data. Counter-overfitting techniques developed to date were designed for static problems (Lau 1994; Williams 1995; Zhang 2005), and, to the authors' knowledge, no studies of overfitting in the context of dynamic environments were published to date. Existing techniques may require modifications in order to become applicable to dynamic problems. The generalisation error was nonetheless recorded throughout the experiments and reported in the analysis that follows.

All reported results are averages over 30 independent simulations. The next section describes the data sets used in the experiments.

6.3 Dynamic classification problems

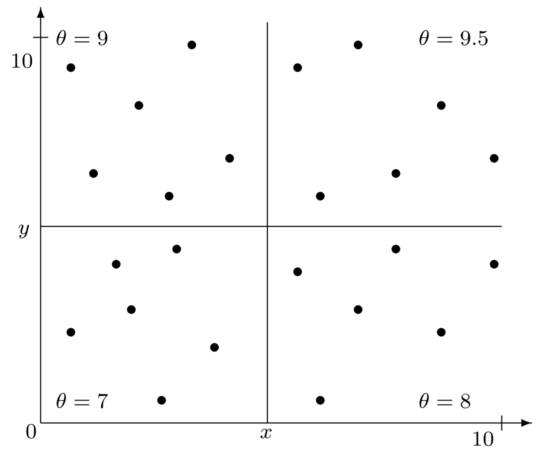
Four synthetically generated classification problems and one real-life classification problem were used in the experiments. Section 6.3.1 describes the streaming ensemble algorithm (SEA) concepts problem, where the data was divided into four mutually exclusive concept blocks. Section 6.3.2 describes a problem where a 10-dimensional hyperplane was used to divide the space into two classes. Section 6.3.3 describes a problem where a three-dimensional sphere was used as a decision boundary between two classes. Section 6.3.4 describes a problem where a three-dimensional space was divided into three classes by two parallel thresholds. Section 6.3.5 describes a problem based on electricity market prices, where the task of the NN was to predict whether the electricity price will increase or decrease given the environment parameters.

6.3.1 SEA concepts

The SEA concepts problem was adopted from Street and Kim (2001) and Tsymbal et al. (2008). The data set consisted of 10 000 patterns, obtained by randomly generating 10 000 three-dimensional points, $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_{10000}\},\ \mathbf{x}_i \in [0; 10)^3,\ i = 1, \dots, 10000$. The generated points were divided into four equal concept blocks, 2500 points each, as illustrated in Fig. 4. In each block, the class label of each data point $\mathbf{x}$ was determined as follows:

$$\text{Classification}(\mathbf{x}) = \begin{cases} \text{Class } A & \text{if } x_1 + x_2 \leq \theta \\ \text{Class } B & \text{otherwise,} \end{cases} \tag{6}$$

Fig. 4 SEA concepts



where x_1 and x_2 are the values of the first two dimensions, and θ is the threshold value. Thus, only the first two dimensions determined the class label of a point. Threshold values of 8, 9, 7, and 9.5 were used in the four blocks, and 10 % class noise was inserted into each block by changing the class label of randomly chosen data patterns in that block. The patterns were then recorded into a single data set in sequential order, block by block, resulting in a data set of 10 000 patterns. A window was slided over the data set to simulate a dynamic environment. The window size was fixed to 2500 patterns, equal to the size of a concept block.

It should be noted at this point that, in case of NN training, the dimensionality of the optimisation problem is determined by the total number of weights and biases, and not by the dimensionality of the input patterns. An NN comprises of three layers: an input layer, a hidden layer, and an output layer. For all the experiments conducted in this study, fully connected NNs were used. Thus, the total number n_w of NN weights for each problem, taking bias units into account, is

$$n_w = (I+1)J + (J+1)K \tag{7}$$

where I is the number of inputs, J is the number of hidden units, and K is the number of outputs. An NN with three input units, four hidden units and one output unit was trained on the SEA concepts problem. According to Eq. (7), the total number of weights and biases, corresponding to the dimensionality of the problem, was equal to 21.

Twenty different dynamic scenarios as outlined in Table 4 were applied to the SEA concepts problem. As shown in Table 4, the values for both the frequency of change and the step size increase nonlinearly, because the influence of parameter values was expected to be stronger for small values. Change frequencies of 10, 50, 100, and 250 iterations were considered, labelled by letters from A to D, respectively. Different levels of spatial severity were simulated by shifting the sliding window by 50, 100, 500, 1000, and 2500 patterns per step, labelled with numbers from 1 to 5, respectively.

6.3.2 Moving hyperplane

The hyperplane is given by

$$\sum_{i=1}^{n} a_i x_i + c = a_0$$

Table 4 Dynamic scenarios for the SEA concepts problem

Frequency (F)	Step size (S)				
	50	100	500	1000	2500
10	A1	A2	A3	A4	A5
50	B1	B2	B3	B4	B5
100	C1	C2	C3	C4	C5
250	D1	D2	D3	D4	D5

Table 5 Dynamic scenarios for the moving hyperplane, dynamic sphere, sliding thresholds, and electricity pricing problems

Frequency (F)	Step size (S)			
	50	100	500	1000
10	A1	A2	A3	A4
50	B1	B2	B3	B4
100	C1	C2	C3	C4
250	D1	D2	D3	D4

where n is the number of dimensions over which the hyperplane is defined, a_i for $i = 1, 2, \dots, n$ 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 hyperplane. The linear coefficients a_i and the constant c are real numbers chosen from the interval $[0, 1]$. A data set was generated according to the procedure described in Sect. 6.1.1, where the number of data points, M, was set to 1000, and the number of environment changes, N, was set to 10. A set of M 10-dimensional points, $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_{1000}\}$, was randomly generated such that $\mathbf{x}_j \in [0, 1]^{10}$, $j = 1, \dots, 1000$. The hyperplane was generated N times by uniformly randomising its coefficients, $\{a_1, a_2, \dots, a_N\} \in [0, 1]$, the constant $c \in [0, 1]$ and the threshold value, $a_0 \in [0, 1]$. Target classification of M patterns was updated N times, and every time the updated patterns were appended to the data set, yielding a data set of 11 000 patterns. The size, M, of the sliding window was set to 1000.

An NN with 10 input units, six hidden units, and a single output unit was used for the moving hyperplane problem. According to Eq. (7), the total number of weights for this problem was equal to 73. Thus, the dimensionality of the moving hyperplane problem was 73.

The moving hyperplane problem was considered under sixteen different dynamic scenarios. Parameter settings corresponding to each dynamic scenario are listed in Table 5. Numbers from 1 to 4 were used to label different levels of spatial severity, and letters from A to D were used to label different levels of temporal severity, as indicated in Table 5.

6.3.3 Dynamic sphere

This dynamic classification problem was obtained by generating a three-dimensional hypersphere and using it to divide the space into two mutually exclusive classes. The hypersphere is given by

$$\sum_{i=1}^{n} (x_i - b_i)^2 = R^2 \tag{8}$$

where R is the radius of the sphere, and $\mathbf{b}$ is the centre of the sphere. For the purpose of this study, n was set to 3. A three-dimensional point outside the hypersphere is labelled as class A, and a three-dimensional point inside the hypersphere is labelled as class B. A data set was generated according to the procedure described in Sect. 6.1.1, where the number of data points, M, was set to 1000, and the number of environment changes, N, was set to 10. A set of M three-dimensional points, $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_{1000}\}$ was randomly generated such that $\mathbf{x}_j \in [0, 1]^3$, $j = 1, \dots, 1000$. The sphere was generated N times by randomising its centre point, $\mathbf{b} \in [0, 1]^3$, and radius, $R \in [0, 1]$. Target classification of M patterns was updated N times, and every time the updated patterns were appended to the data set, yielding a data set of 11 000 patterns in total. The size of the sliding window was set to 1000.

An NN with three input units, four hidden units and one output unit was trained on the Dynamic Sphere problem. The total number of weights and biases, corresponding to the dimensionality of the problem, was equal to 21.

Sixteen different dynamic scenarios as described in Sect. 6.3.2 and outlined in Table 5 were applied to the dynamic sphere problem. Parameter settings corresponding to each dynamic scenario are listed in Table 5.

6.3.4 *Sliding thresholds*

For this problem, the Cartesian space was subdivided into three classes by means of two parallel linear thresholds, $f_1(\mathbf{x})$ and $f_2(\mathbf{x})$, given by

$$f_1(\mathbf{x}) = t_1$$
$$f_2(\mathbf{x}) = t_2$$
$$t_1 < t_2$$

where t_1 and t_2 are constant values, and therefore $f_1(\mathbf{x})$ and $f_2(\mathbf{x})$ are parallel vertical lines. Thus, a 2D point can be classified based on its x-axis component only (x_1). Classification was done as follows:

$$\text{Classification}(\mathbf{x}) = \begin{cases} \text{Class } A & \text{if } x_1 \le t_1 \\ \text{Class } B & \text{if } x_1 \ge t_2 \\ \text{Class } C & \text{otherwise} \end{cases} \tag{9}$$

A data set was generated according to the procedure described in Sect. 6.1.1, where the number of points, M, was set to 1000, and the total number of environment changes, N, was set to 10. A set of M two-dimensional points $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_{1000}\}$ was randomly generated such that $\mathbf{x}_j \in [0, 1]^2$, $j = 1, \dots, 1000$, and each point $\mathbf{x}_j$ was assigned a class value according to Eq. (9). Thresholds were generated N times by setting t_1 and t_2 to random numbers from the interval $[0, 1]$ such that $t_1 < t_2$. Target classification of M patterns was updated N times, and every time the updated patterns were appended to the data set, yielding a data set of 11 000 patterns in total. The size of the sliding window was set to 1000.

An NN with two input units, three hidden units, and three output units was trained on the Sliding Thresholds problem. According to Eq. (7), the total number of weights and biases, corresponding to the dimensionality of the problem, was equal to 21.

Sixteen different dynamic scenarios as described in Sect. 6.3.2 and outlined in Table 5 were considered for the sliding thresholds problem. Sliding thresholds is a 2D problem that has only linear decision boundaries. Although linear decision boundaries are trivial to learn, this problem was rather difficult to optimise due to the sliding window approach used to simulate dynamic environments. As discussed in Sect. 6.1.1, 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 $M = 1000$ patterns. The previous problems discussed dealt with one decision boundary per 1000 patterns. Thus, for the previous problems, a sliding window of N patterns, $N \leq M$, contained at least one decision boundary and at most two. For the sliding thresholds problem, there were two decision boundaries for every 1000 data patterns. After every 1000 patterns traversed, two new boundaries were introduced. Therefore, for the sliding thresholds problem, the sliding window contained 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.

6.3.5 Electricity pricing

This problem used a real-life data set based on the electricity market in the Australian state of New South Wales. The data set was adopted from Harries (1999).

Prices in the electricity market are determined by matching the present demand for electricity with the least expensive combination of electricity from all available power stations. Both market prices and electricity price schedules published by each power station are frequently recalculated and updated. Market prices depend on both demand and supply of electrical power. Significant factors affecting the demand are season, weather, time of day and central business district population density. The supply is affected mainly by the number of active electricity generators. Thus, this environment is subject to both regular long-term changes, such as seasonal changes, and irregular short-term changes, such as weather fluctuations (Harries 1999).

A dynamic classification problem was constructed based on the changing electricity market price. Six parameters on which the price is dependent were identified. These include day of week, time of day, and electricity demand estimates. Parameter values were recorded every half an hour, from 7 May 1996 to 5 December 1998. In this manner, a data set of 27 552 samples was recorded. Each pattern was labelled as either class A or class B. The class label identified whether the current price is higher (class A) or lower (class B) than a moving average price over the last 24 hours. The task of the NN was thus to predict whether the price will go up or down based on the given input values.

Dynamic environments were simulated by sliding a window over the data set. The original temporal order of the data was preserved, and the patterns were shuffled only inside the window. The size of the window was set to 1000 patterns. An NN with six input units, six hidden units and one output unit was trained on the electricity pricing problem. According to Eq. (7), the total number of weights and biases, corresponding to the dimensionality of the problem, was equal to 49.

Sixteen different dynamic scenarios as described in Sect. 6.3.2 and outlined in Table 5 were applied to the electricity pricing problem.

6.4 Analysis of empirical data

Table 6 lists the average ranks obtained by the considered algorithms under the five considered problems, where a lower rank value indicates a better performing algorithm. Algorithms were ranked based on their collective mean E_T and E_G values reported in Appendix 7. Algorithm ranking involved the two-tailed non-parametric Mann–Whitney U test (Mann and Whitney 1947), used to determine whether the difference in performance between any two algorithms was statistically significant. The choice of the significance test is based on Demšar (2006), where the authors showed that the Mann–Whitney U test is safer than parametric tests such as the t-test, since the Mann–Whitney U test assumes neither normal distributions of data nor homogeneity of variance. The null hypothesis $H_0 : \mu_1 = \mu_2$, where μ_1 and μ_2 are the means of the two samples being compared, was evaluated at a significance level of 95 %. The alternative hypothesis was defined as $H_1 : \mu_1 \neq \mu_2$. Thus, any p-value less than 0.05 corresponded to rejection of the null hypothesis that there is no statistically significant difference between the sample means. Algorithms that produced significantly lower E_G were assigned a lower rank value than algorithms that produced significantly higher E_G. When no statistically significant difference between produced E_G values was observed, the algorithms were assigned equal rank. Thus, algorithm ranks reflect statistical significance.

It follows from the overall average ranks in Table 6 that, on average, all three dynamic PSO algorithms except RAPSO obtained a better rank than the four BP algorithms. Thus, it can immediately be concluded that the dynamic PSOs proved to be a viable alternative to BP algorithms. RBP obtained the worst average rank, thus showing that restarting the training algorithm after every environment change was the least successful approach to NN training under concept drift. SRBP performed consistently better than RBP, however, still inferior to BP and SBP for most problems. Out of the four dynamic PSOs, RAPSO obtained the worst rank, thus confirming the inefficiency of the restarting approach to dynamic classification problems, irrespective of the temporal severity.

Figure 5(a) illustrates the E_G profiles obtained by the algorithms on the electricity pricing problem under infrequent gradual changes. Figure 5(a) illustrates that RBP's, SRBP's, and RAPSO's errors not only fluctuated severely, but also failed to reach a minimum E_G as obtained by BP and the dynamic PSOs (only QPSO is illustrated for the sake of clarity). Mann–Whitney U test produced p-values less than 0.0001 when BP, RBP, SRBP, RAPSO, and QPSO were compared to one another. Thus, the difference between algorithm performance was significant. RBP, SRBP, and RAPSO made no use of previously learned information when a change occurred, and started the search for decision boundaries anew every time that the environment changed. However, previously learned information remained useful when changes were spatially gradual (i.e., only a few new patterns were added to the sliding window) or when the new patterns were derived from the previous patterns (e.g., electricity pricing problem). Thus, the algorithms which made use of previously learned information had an advantage over the algorithms which did not.

The ranks in Table 6 also show that RBP's E_T rank was often better than the corresponding E_G rank, indicating that RBP did not generalise well. Figure 5(b) illustrates E_G profiles obtained by the algorithms on the moving hyperplane problem, under a semi-abrupt scenario, where exactly one half of the sliding window was replaced by new patterns whenever a change occurred. Figure 5(b) illustrates that RBP performed comparably well to other algorithms when the sliding window contained a single concept (complete replacement of patterns inside the window) but failed to generalise when two bordering and possibly conflicting concepts were present in the sliding window. Thus, RBP exhibited a strong sensitivity to the presence of stale data inside the sliding window. Figure 5(b) illustrates that restarting RBP at the sign of stagnation (SRBP) significantly improved performance (p-value < 0.0001). However, SRBP was still significantly outperformed by algorithms which did not employ complete reinitialisation of weights (e.g., BP and CPSO, p-values < 0.0001). The fact that poor generalisation of RBP was effectively remedied by SRBP indicates that RBP's susceptibility to poor generalisation was due to the sensitivity of the gradient descent methods to the starting point of the search.

Figure 5(c) illustrates a scenario under which all variants of BP (namely, BP, SBP, RBP, and SRBP) failed to optimise decision boundaries between environment changes. As discussed in Sect. 5, BP will fail to adapt to an environment change if the change of the error function landscape causes the current position of the NN weight vector to be in an unfruitful region such as a plateau or a local minimum. The failure of BP to adapt to changes illustrated in Fig. 5(c) indicates that these changes trapped BP in a region from which BP could

Table 6 Average algorithm ranks

Algorithm		SEA concepts	Moving hyperplane	Dynamic sphere	Sliding thresholds	Electricity pricing	Average rank
BP	R(E_T)	3.825	5.21875	5.25	7.46875	2.75	4.9025
	R(E_G)	**3.625**	3.8125	3.9375	7.15625	2.90625	4.2875
SBP	R(E_T)	3.925	5.125	4.96875	6.09375	3.53125	4.72875
	R(E_G)	3.725	4.5625	**3.34375**	5.25	3.40625	4.0575
RBP	R(E_T)	7.05	**2.59375**	**1.8125**	5.4375	6.65625	4.71
	R(E_G)	7.225	7.46875	5.71875	7.25	7.90625	7.11375
SRBP	R(E_T)	5.45	6.84375	5.6875	6.4375	6.90625	6.265
	R(E_G)	3.975	5.0625	4.1875	5.5	6.125	4.97
RPSO	R(E_T)	2.8	3.65625	2.75	**1.75**	4.0625	3.00375
	R(E_G)	3.85	3.5	3.9375	**1.8125**	3.78125	3.37625
RAPSO	R(E_T)	6.55	5.40625	4.09375	3.40625	7	5.29125
	R(E_G)	5.9	5.1875	4.90625	3.4375	6.59375	5.205
CPSO	R(E_T)	3.65	3.1875	5.46875	2.75	2.875	3.58625
	R(E_G)	3.85	**2.96875**	4.71875	2.84375	2.6875	3.41375
QPSO	R(E_T)	**2.75**	3.96875	5.96875	2.65625	**2.21875**	3.5125
	R(E_G)	3.85	3.4375	5.25	2.75	**2.59375**	3.57625

not escape. RBP was reinitialised after every change, and SBP and SRBP were reinitialised whenever stagnation was observed. Thus, RBP, SBP, and SRBP did not become trapped in such unfruitful regions as easily as BP (produced p-values were <0.0001, indicating statistical significance). However, Fig. 5(c) illustrates that RBP, SBP, and SRBP were also susceptible to stagnation between changes and performed significantly worse than the dynamic PSOs (only RPSO is illustrated in Fig. 5(c) for the sake of clarity, p-value <0.0001 obtained when comparing RPSO to RBP, SBP, and SRBP). Stagnation of the BP algorithms is attributed to the strong dependency of the gradient-based algorithms on the starting point of the search: RBP, SBP, and SRBP failed to find an optimum when the weight vector was located in an unfruitful region of the search space after reinitialisation.

Table 6 indicates that RPSO obtained the best rank on the sliding thresholds problem, where, as explained in Sect. 6.3.4, multiple decision boundaries were introduced, making the problem of stale data more severe. Figure 6(a) illustrates algorithm E_G profiles obtained for the sliding thresholds problem under scenario C1 and indicates that the RPSO's E_G profile peaked lower than that of the other algorithms (p-values <0.0001 obtained when comparing RPSO to other algorithms). The RPSO is the least memory-dependent of the three dynamic PSOs considered, because RPSO reinitialises a percentage of randomly selected particles, not sparing the neighbourhood best particles if these particles happen to be chosen for reinitialisation. Weaker dependence on memory helps RPSO to promptly "unlearn" obsolete information after every environment change, which proved useful on a problem where multiple decision boundaries had to be updated.

Table 6 shows that CPSO and QPSO performed better than RPSO and RAPSO on problems where the ability to preserve previously learned information was more important than

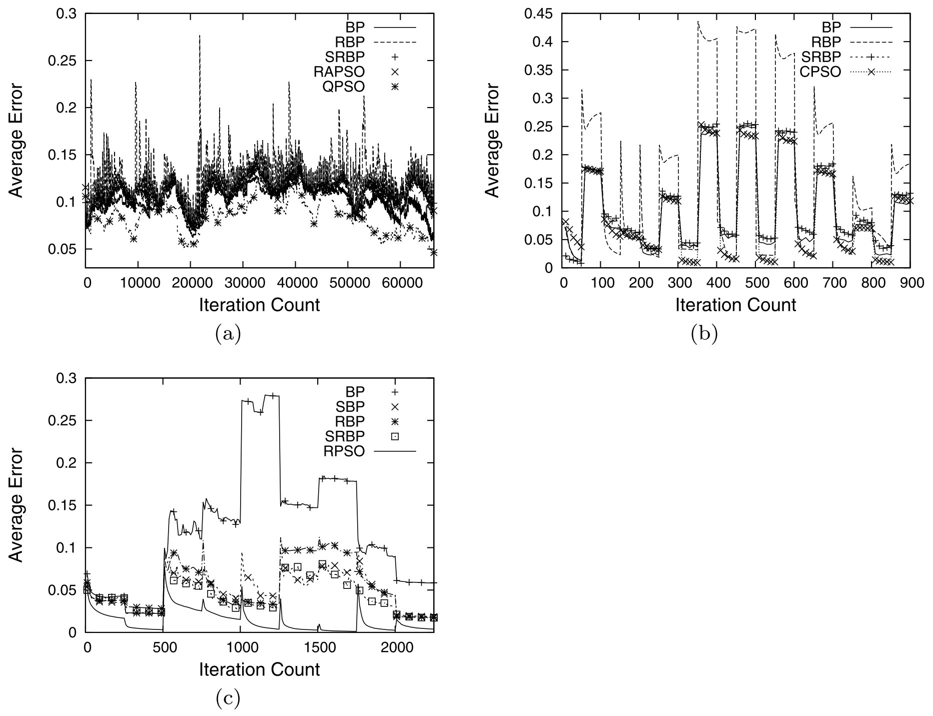


Fig. 5 Generalisation error profiles over time. (**a**) Electricity pricing, E_G for D2. (**b**) Moving hyperplane, E_G for B3. (**c**) Sliding thresholds, E_G for D4

the ability to promptly "unlearn" the stale data. Examples of such problems are the electricity pricing problem, where new electricity prices were derived from old electricity prices, and the moving hyperplane problem, where the reinitialising approach did not prove efficient due to the dimensionality of the problem. Figure 6(b) illustrates algorithm E_G profiles obtained for the electricity pricing under a gradual temporally severe scenario. Here, CPSO and QPSO were visibly more apt at tracking the moving optima than RPSO (worst performing RAPSO not shown for the sake of clarity). Superiority of CPSO and QPSO over RPSO was confirmed to be statistically significant (p-values <0.0001). Under frequent changes, RPSO did not have enough time to properly exploit the found optima, whereas CPSO and QPSO used their memory of previous solutions to promptly optimise the solution at hand.

Table 7 lists the best performing algorithms under various sliding window step sizes (i.e., spatial severity) considered for the five dynamic classification problems. Table 7 shows that the dynamic PSOs were more successful under gradual to semi-gradual scenarios, whereas the BP algorithms were more successful under abrupt scenarios. An important advantage of the gradient-based approach to NN training is the fast speed of convergence, and it is precisely this property which allowed BP, SBP, and SRBP to outperform the dynamic PSOs under abrupt changes. Figure 7(a) illustrates that under an abrupt scenario, the dynamic PSOs took longer to converge than both BP, SBP, and RBP (only SBP and RBP are shown for the sake of clarity): the three dynamic PSOs took 25 iterations to reach the minimum attained by SBP and RBP in 10 iterations. Figure 7(b) illustrates another abrupt scenario where the dynamic PSOs exhibited no signs of stagnation but were unable to match the

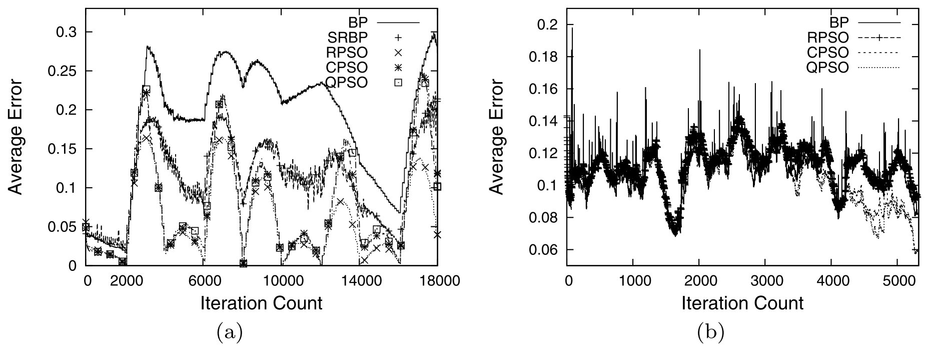


Fig. 6 Generalisation error profiles over time. (**a**) Sliding thresholds, E_G for C1. (**b**) Electricity pricing, E_G for A1

Table 7 Best algorithms under varying spatial severity

Step size	50	100	500	1000	2500
SEA concepts	RPSO, QPSO, CPSO	RPSO, QPSO	RPSO, QPSO	SBP, SRBP	SRBP
Moving hyperplane	RPSO	RPSO	CPSO	BP	–
Dynamic sphere	RPSO	RPSO	SBP	SRBP	–
Sliding thresholds	RPSO	RPSO	RPSO	QPSO	–
Electricity pricing	CPSO, QPSO	CPSO, QPSO	BP	BP, SBP	–

convergence speed of the gradient-based algorithms. Thus, slow convergence speed of the dynamic PSOs often made these algorithms less efficient than BP under abrupt changes.

However, Figs. 5(a) and 7(c) illustrate that, once the dynamic PSOs reached a minimum, the dynamic PSOs maintained the obtained minimum better than the BP algorithms: under gradual scenarios, the dynamic PSOs were able to reach a lower minimum error than that obtained by the BP algorithms. The difference between BP and the dynamic PSOs was statistically significant (p-values <0.05). The dynamic PSOs also tracked gradual changes more closely than the gradient-based algorithms. Gradual changes imply that little new data is added to the sliding window after every change, and thus the new optimum is expected to be in the near proximity of the old optimum. The population-based principle allowed the dynamic PSOs to promptly evaluate the area covered by the swarm, and immediately find a more up-to-date solution amongst the particles, while BP algorithms had to move in the direction of the negative gradient towards the changed optimum. The ability to explore a wider area around the current solution made the dynamic PSOs more apt than the gradient-based algorithms at tracking gradual changes.

Table 8 lists the best performing algorithms under various change frequencies (i.e., temporal severity). Table 8 shows that BP and SBP were more successful under frequent changes, which corresponds to the already made observation that BP algorithms converged faster than the dynamic PSOs. However, when given enough time to train, the dynamic PSOs outperformed the BP algorithms on most problems considered. The only exception to this rule was the dynamic sphere problem: Table 8 shows that, although RPSO was the best performer under the most temporally severe scenario on the dynamic sphere problem, RPSO was significantly outperformed by SBP under all other change frequencies considered for the same problem. This indicates that, although the dynamic PSOs typically took

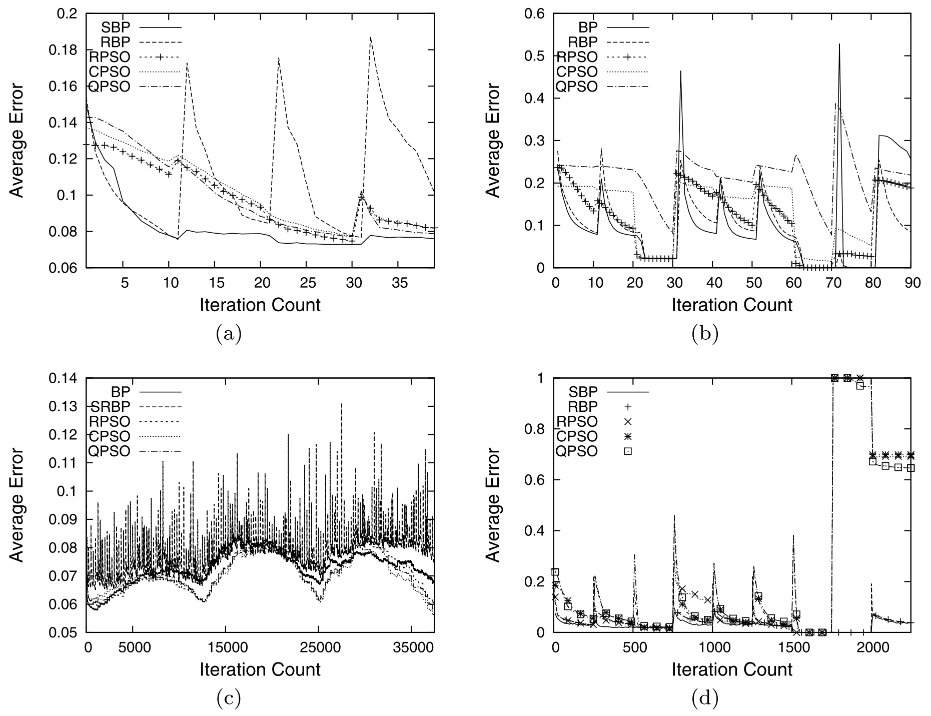


Fig. 7 Training and generalisation error profiles over time. (**a**) SEA concepts, E_G for A5. (**b**) Dynamic sphere, E_G for A4. (**c**) SEA concepts, E_G for D1. (**d**) Dynamic sphere, E_G for D4

Table 8 Best algorithms under varying temporal severity

Change frequency	A (10)	B (50)	C (100)	D (250)
SEA concepts	SBP	RPSO, CPSO, QPSO	RPSO, QPSO	RPSO, QPSO
Moving hyperplane	SBP	CPSO	CPSO	RPSO
Dynamic sphere	RPSO	SBP	SBP	SBP
Sliding thresholds	CPSO, QPSO	RPSO	RPSO	RPSO
Electricity pricing	BP	BP	CPSO, QPSO	CPSO, QPSO

longer than the BP algorithms to exploit a fruitful area, exploiting a fruitful area for too long can indeed result in poor performance due to over-exploitation. Figure 7(d) illustrates the algorithms' E_G profiles obtained on the dynamic sphere problem under abrupt infrequent changes. Figure 7(d) illustrates that, after iteration 1500, all dynamic PSOs obtained $E_G \approx 0$ and produced a very high error after the next environment change. Low temporal severity allowed the dynamic PSOs not only to find a good solution, but also to over-exploit it, leading the swarms to an unfruitful search space region (such as a plateau or a local minimum), from which the swarms struggled to escape after a change in the error landscape. The fact that neither SBP nor RBP exhibited over-exploitation under the same scenario indicates that the dynamic PSOs are more prone to over-exploitation than BP algorithms when trained for too long.

In general, it can be observed that the dynamic PSOs exhibited a stronger sensitivity to such characteristics of dynamic environments as spatial severity and temporal severity. Figures 8(a) and 8(b) illustrate the collective mean E_G values obtained by the algorithms under various dynamic scenarios for the electricity pricing problem and the moving hyperplane problem, respectively. In Figs. 8(a) and 8(b), the performance of the dynamic PSOs deteriorated as both the temporal severity and the spatial severity increased. In Fig. 8(b), the performance of dynamic PSOs also deteriorated under gradual scenarios of low temporal severity, because the dynamic PSOs were allowed to train for too long and thus over-exploited. In both Figs. 8(a) and 8(b), BP produced more robust results than the dynamic PSOs: the BP error was less affected by the extent of spatial and temporal severity. The dynamic PSO algorithms involve more parameter tuning than BP, since, in addition to the standard PSO parameters, parameters specific to each dynamic PSO also require tuning. Hence, the dynamic PSOs require more scenario-specific parameter tuning than the BP algorithms but have a high potential to outperform the gradient-based algorithms when properly tuned.

Figure 8(c) illustrates the collective mean E_G values obtained by the algorithms under various dynamic scenarios for the sliding thresholds problem. Only the best performing dynamic PSO and BP algorithms are depicted. Figure 8(c) illustrates that the dynamic PSOs outperformed the BP algorithms under most scenarios considered. Superior performance of the dynamic PSOs on a problem where multiple decision boundaries were present indicates that the dynamic PSOs were less sensitive to the total number of decision boundaries, as well as the presence of stale data inside the sliding window. Thus, the dynamic PSOs may be expected to outperform BP on more rugged NN error surfaces.

7 Conclusions and future work

The main objective of this paper was to show that dynamic PSO algorithms have potential to be efficiently applied to NN training in the presence of concept drift. An experimental procedure was designed to compare the training algorithms on a representative selection of dynamic classification problems under a representative selection of dynamic environments. A sliding window was used for pattern selection, and dynamic environments of varying spatial and temporal severity were simulated by adjusting the step size of the sliding window and the number of algorithm iterations between the sliding window shifts, respectively. Four artificially generated dynamic classification problems and one real-life data set with concept drift were used in the experiments. The problems differed in terms of dimensionality, decision boundary shape, and the total number of decision boundaries. Backpropagation (BP), BP with stagnation detection (SBP), reinitialising BP (RBP), and RBP with stagnation detection (SRBP) were compared with four dynamic PSO algorithms, namely reinitialising PSO (RPSO), reinitialising-all PSO (RAPSO), charged PSO (CPSO), and quantum PSO (QPSO).

The obtained empirical results showed the dynamic PSOs to be viable NN training algorithms in the context of dynamic classification problems. The dynamic PSOs exploited and tracked decision boundaries better than the BP algorithms, allowing the dynamic PSOs to outperform the BP algorithms under scenarios exhibiting infrequent to moderately infrequent gradual changes. The BP algorithms converged faster than the dynamic PSOs, which enabled BP algorithms to outperform the dynamic PSOs under frequent abrupt changes. RBP, SRBP, and RAPSO exhibited the worst performance under most scenarios for most problems, thus showing that a complete reinitialisation of NN weights is an inefficient approach to concept drift problems. The dynamic PSOs exhibited stronger sensitivity to the

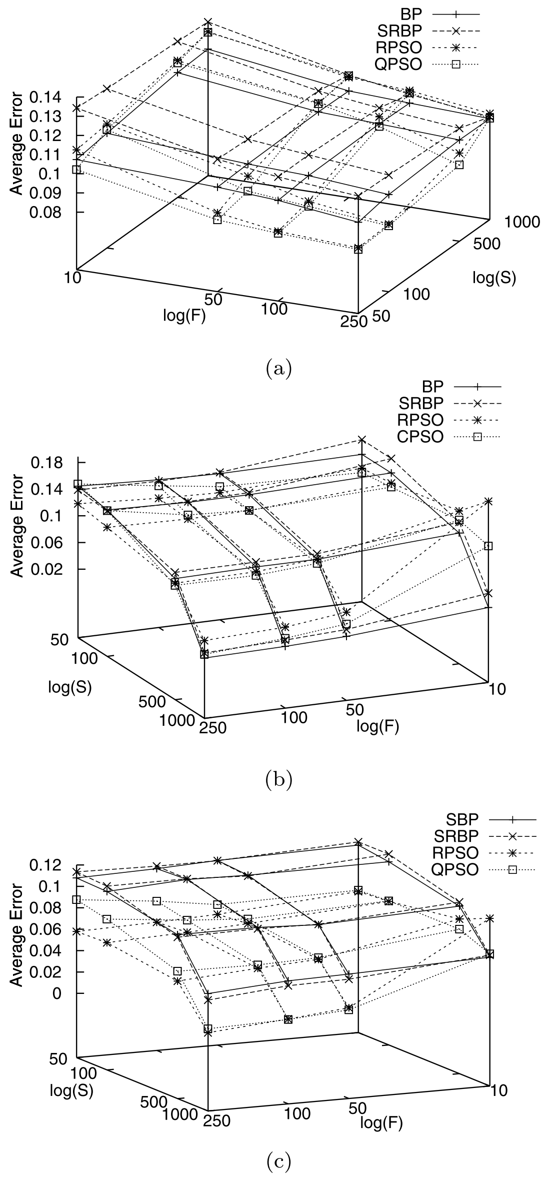


Fig. 8 Average generalisation error results across all dynamic scenarios considered (*F* refers to temporal severity, *S* refers to spatial severity). (**a**) Average generalisation error results for electricity pricing. (**b**) Average generalisation error results for moving hyperplane. (**c**) Average generalisation error results for sliding thresholds

extent of temporal and spatial severity than the BP algorithms. However, the dynamic PSOs were less sensitive to the total number of decision boundaries or the presence of stale data in the sliding window than the BP algorithms. Thus, the dynamic PSOs are expected to perform better than gradient-based algorithms on rugged NN error surfaces. The RPSO performed better than the CPSO and the QPSO on problems which required prompt "unlearning" after a change (e.g., stale data present in the sliding window, numerous decision boundaries). The CPSO and the QPSO performed better than the RPSO and RAPSO on problems which required previously learned information to be preserved (e.g., no or little stale data present, new decision boundaries derived from old decision boundaries).

Further research will include a study of overfitting behaviour exhibited by the dynamic PSOs applied to NN training under concept drift. The applicability of dynamic PSO NN training to dynamic problems other than classification problems, for example, dynamic function approximation, also needs to be investigated. The authors plan to apply the proposed approach to large-scale real-life scenarios. The dependence of the dynamic PSO performance on choosing optimal control parameter values can be lessened by developing self-adapting parameter strategies for dynamic environments. It would also be interesting to apply other dynamic population-based techniques, such as dynamic genetic algorithms, to NN training in dynamic environments. The most popular approach to concept drift handling, namely ensemble learning, can also be improved by employing dynamic PSOs as training algorithms for the single classifiers involved.

Appendix: Collective mean error values

This appendix reports the collective mean E_T and E_G values obtained by the algorithms for the five dynamic classification problems considered. Tables 9 and 10 list E_T and E_G obtained for the SEA concepts problem. Table 11 lists E_T and E_G obtained for the moving hyperplane problem. Table 12 lists E_T and E_G obtained for the dynamic sphere problem. Table 13 lists E_T and E_G obtained for the sliding thresholds problem. Table 14 lists E_T and E_G obtained for the electricity pricing problem.

Table 9 SEA concepts results for scenarios A1 to B5

Scenario			BP	SBP	RBP	SRBP	RPSO	RAPSO	CPSO	QPSO
A1	E_T	Mean	0.075058	0.075448	0.11765	0.084086	0.073435	0.083159	0.074025	0.073285
		StdDev	0.00167	0.001603	0.004987	0.001245	0.000793	0.001223	0.000728	0.000881
	E_G	Mean	0.076264	0.073648	0.117498	0.082611	0.074669	0.082612	0.074008	0.074031
		StdDev	0.005138	0.004647	0.007621	0.005009	0.004909	0.004059	0.004829	0.004996
A2	E_T	Mean	0.075058	0.075448	0.11765	0.084086	0.073435	0.083159	0.074025	0.073285
		StdDev	0.00167	0.001603	0.004987	0.001245	0.000793	0.001223	0.000728	0.000881
	E_G	Mean	0.076264	0.073648	0.117498	0.082611	0.074669	0.082612	0.074008	0.074031
		StdDev	0.005138	0.004647	0.007621	0.005009	0.004909	0.004059	0.004829	0.004996
A3	E_T	Mean	0.075058	0.075448	0.11765	0.084086	0.073435	0.083159	0.074025	0.073285
		StdDev	0.00167	0.001603	0.004987	0.001245	0.000793	0.001223	0.000728	0.000881
	E_G	Mean	0.076264	0.073648	0.117498	0.082611	0.074669	0.082612	0.074008	0.074031
		StdDev	0.005138	0.004647	0.007621	0.005009	0.004909	0.004059	0.004829	0.004996
A4	E_T	Mean	0.075058	0.075448	0.11765	0.084086	0.073435	0.083159	0.074025	0.073285
		StdDev	0.00167	0.001603	0.004987	0.001245	0.000793	0.001223	0.000728	0.000881
	E_G	Mean	0.076264	0.073648	0.117498	0.082611	0.074669	0.082612	0.074008	0.074031
		StdDev	0.005138	0.004647	0.007621	0.005009	0.004909	0.004059	0.004829	0.004996
A5	E_T	Mean	0.075058	0.075448	0.11765	0.084086	0.073435	0.083159	0.074025	0.073285
		StdDev	0.00167	0.001603	0.004987	0.001245	0.000793	0.001223	0.000728	0.000881
	E_G	Mean	0.076264	0.073648	0.117498	0.082611	0.074669	0.082612	0.074008	0.074031
		StdDev	0.005138	0.004647	0.007621	0.005009	0.004909	0.004059	0.004829	0.004996
B1	E_T	Mean	0.075058	0.075448	0.11765	0.084086	0.073435	0.083159	0.074025	0.073285
		StdDev	0.00167	0.001603	0.004987	0.001245	0.000793	0.001223	0.000728	0.000881
	E_G	Mean	0.076264	0.073648	0.117498	0.082611	0.074669	0.082612	0.074008	0.074031
		StdDev	0.005138	0.004647	0.007621	0.005009	0.004909	0.004059	0.004829	0.004996
B2	E_T	Mean	0.075058	0.075448	0.11765	0.084086	0.073435	0.083159	0.074025	0.073285
		StdDev	0.00167	0.001603	0.004987	0.001245	0.000793	0.001223	0.000728	0.000881
	E_G	Mean	0.076264	0.073648	0.117498	0.082611	0.074669	0.082612	0.074008	0.074031
		StdDev	0.005138	0.004647	0.007621	0.005009	0.004909	0.004059	0.004829	0.004996
B3	E_T	Mean	0.075058	0.075448	0.11765	0.084086	0.073435	0.083159	0.074025	0.073285
		StdDev	0.00167	0.001603	0.004987	0.001245	0.000793	0.001223	0.000728	0.000881
	E_G	Mean	0.076264	0.073648	0.117498	0.082611	0.074669	0.082612	0.074008	0.074031
		StdDev	0.005138	0.004647	0.007621	0.005009	0.004909	0.004059	0.004829	0.004996
B4	E_T	Mean	0.075058	0.075448	0.11765	0.084086	0.073435	0.083159	0.074025	0.073285
		StdDev	0.00167	0.001603	0.004987	0.001245	0.000793	0.001223	0.000728	0.000881
	E_G	Mean	0.076264	0.073648	0.117498	0.082611	0.074669	0.082612	0.074008	0.074031
		StdDev	0.005138	0.004647	0.007621	0.005009	0.004909	0.004059	0.004829	0.004996
B5	E_T	Mean	0.075058	0.075448	0.11765	0.084086	0.073435	0.083159	0.074025	0.073285
		StdDev	0.00167	0.001603	0.004987	0.001245	0.000793	0.001223	0.000728	0.000881
	E_G	Mean	0.076264	0.073648	0.117498	0.082611	0.074669	0.082612	0.074008	0.074031
		StdDev	0.005138	0.004647	0.007621	0.005009	0.004909	0.004059	0.004829	0.004996

Table 10 SEA concepts results for scenarios C1 to D5

Scenario			BP	SBP	RBP	SRBP	RPSO	RAPSO	CPSO	QPSO
C1	E_T	Mean	0.073506	0.073984	0.077631	0.076269	0.070671	0.073329	0.07088	0.070579
		StdDev	0.001122	0.001314	0.002366	0.001408	0.000646	0.000733	0.000762	0.00085
	E_G	Mean	0.072902	0.073635	0.086025	0.07449	0.071146	0.073593	0.071306	0.072789
		StdDev	0.003983	0.00514	0.005988	0.003283	0.004587	0.004568	0.004753	0.005814
C2	E_T	Mean	0.073506	0.073984	0.077631	0.076269	0.070671	0.073329	0.07088	0.070579
		StdDev	0.001122	0.001314	0.002366	0.001408	0.000646	0.000733	0.000762	0.00085
	E_G	Mean	0.072902	0.073635	0.086025	0.07449	0.071146	0.073593	0.071306	0.072789
		StdDev	0.003983	0.00514	0.005988	0.003283	0.004587	0.004568	0.004753	0.005814
C3	E_T	Mean	0.073506	0.073984	0.077631	0.076269	0.070671	0.073329	0.07088	0.070579
		StdDev	0.001122	0.001314	0.002366	0.001408	0.000646	0.000733	0.000762	0.00085
	E_G	Mean	0.072902	0.073635	0.086025	0.07449	0.071146	0.073593	0.071306	0.072789
		StdDev	0.003983	0.00514	0.005988	0.003283	0.004587	0.004568	0.004753	0.005814
C4	E_T	Mean	0.073506	0.073984	0.077631	0.076269	0.070671	0.073329	0.07088	0.070579
		StdDev	0.001122	0.001314	0.002366	0.001408	0.000646	0.000733	0.000762	0.00085
	E_G	Mean	0.072902	0.073635	0.086025	0.07449	0.071146	0.073593	0.071306	0.072789
		StdDev	0.003983	0.00514	0.005988	0.003283	0.004587	0.004568	0.004753	0.005814
C5	E_T	Mean	0.073506	0.073984	0.077631	0.076269	0.070671	0.073329	0.07088	0.070579
		StdDev	0.001122	0.001314	0.002366	0.001408	0.000646	0.000733	0.000762	0.00085
	E_G	Mean	0.072902	0.073635	0.086025	0.07449	0.071146	0.073593	0.071306	0.072789
		StdDev	0.003983	0.00514	0.005988	0.003283	0.004587	0.004568	0.004753	0.005814
D1	E_T	Mean	0.073506	0.073984	0.077631	0.076269	0.070671	0.073329	0.07088	0.070579
		StdDev	0.001122	0.001314	0.002366	0.001408	0.000646	0.000733	0.000762	0.00085
	E_G	Mean	0.072902	0.073635	0.086025	0.07449	0.071146	0.073593	0.071306	0.072789
		StdDev	0.003983	0.00514	0.005988	0.003283	0.004587	0.004568	0.004753	0.005814
D2	E_T	Mean	0.073506	0.073984	0.077631	0.076269	0.070671	0.073329	0.07088	0.070579
		StdDev	0.001122	0.001314	0.002366	0.001408	0.000646	0.000733	0.000762	0.00085
	E_G	Mean	0.072902	0.073635	0.086025	0.07449	0.071146	0.073593	0.071306	0.072789
		StdDev	0.003983	0.00514	0.005988	0.003283	0.004587	0.004568	0.004753	0.005814
D3	E_T	Mean	0.073506	0.073984	0.077631	0.076269	0.070671	0.073329	0.07088	0.070579
		StdDev	0.001122	0.001314	0.002366	0.001408	0.000646	0.000733	0.000762	0.00085
	E_G	Mean	0.072902	0.073635	0.086025	0.07449	0.071146	0.073593	0.071306	0.072789
		StdDev	0.003983	0.00514	0.005988	0.003283	0.004587	0.004568	0.004753	0.005814
D4	E_T	Mean	0.073506	0.073984	0.077631	0.076269	0.070671	0.073329	0.07088	0.070579
		StdDev	0.001122	0.001314	0.002366	0.001408	0.000646	0.000733	0.000762	0.00085
	E_G	Mean	0.072902	0.073635	0.086025	0.07449	0.071146	0.073593	0.071306	0.072789
		StdDev	0.003983	0.00514	0.005988	0.003283	0.004587	0.004568	0.004753	0.005814
D5	E_T	Mean	0.073506	0.073984	0.077631	0.076269	0.070671	0.073329	0.07088	0.070579
		StdDev	0.001122	0.001314	0.002366	0.001408	0.000646	0.000733	0.000762	0.00085
	E_G	Mean	0.072902	0.073635	0.086025	0.07449	0.071146	0.073593	0.071306	0.072789
		StdDev	0.003983	0.00514	0.005988	0.003283	0.004587	0.004568	0.004753	0.005814

Table 11 Moving hyperplane results for scenarios A1 to D4

Scenario			BP	SBP	RBP	SRBP	RPSO	RAPSO	CPSO	QPSO
A1	E_T	Mean	0.155954	0.155336	0.114547	0.181174	0.112531	0.140746	0.107024	0.108173
		StdDev	0.001654	0.001883	0.001191	0.001702	0.000872	0.001634	0.001584	0.00201
	E_G	Mean	0.13821	0.140406	0.237748	0.160352	0.117451	0.146158	0.110943	0.112485
		StdDev	0.005143	0.005242	0.004665	0.003842	0.003418	0.003258	0.004261	0.003668
A2	E_T	Mean	0.155954	0.155336	0.114547	0.181174	0.112531	0.140746	0.107024	0.108173
		StdDev	0.001654	0.001883	0.001191	0.001702	0.000872	0.001634	0.001584	0.00201
	E_G	Mean	0.13821	0.140406	0.237748	0.160352	0.117451	0.146158	0.110943	0.112485
		StdDev	0.005143	0.005242	0.004665	0.003842	0.003418	0.003258	0.004261	0.003668
A3	E_T	Mean	0.155954	0.155336	0.114547	0.181174	0.112531	0.140746	0.107024	0.108173
		StdDev	0.001654	0.001883	0.001191	0.001702	0.000872	0.001634	0.001584	0.00201
	E_G	Mean	0.13821	0.140406	0.237748	0.160352	0.117451	0.146158	0.110943	0.112485
		StdDev	0.005143	0.005242	0.004665	0.003842	0.003418	0.003258	0.004261	0.003668
A4	E_T	Mean	0.155954	0.155336	0.114547	0.181174	0.112531	0.140746	0.107024	0.108173
		StdDev	0.001654	0.001883	0.001191	0.001702	0.000872	0.001634	0.001584	0.00201
	E_G	Mean	0.13821	0.140406	0.237748	0.160352	0.117451	0.146158	0.110943	0.112485
		StdDev	0.005143	0.005242	0.004665	0.003842	0.003418	0.003258	0.004261	0.003668
B1	E_T	Mean	0.155954	0.155336	0.114547	0.181174	0.112531	0.140746	0.107024	0.108173
		StdDev	0.001654	0.001883	0.001191	0.001702	0.000872	0.001634	0.001584	0.00201
	E_G	Mean	0.13821	0.140406	0.237748	0.160352	0.117451	0.146158	0.110943	0.112485
		StdDev	0.005143	0.005242	0.004665	0.003842	0.003418	0.003258	0.004261	0.003668
B2	E_T	Mean	0.155954	0.155336	0.114547	0.181174	0.112531	0.140746	0.107024	0.108173
		StdDev	0.001654	0.001883	0.001191	0.001702	0.000872	0.001634	0.001584	0.00201
	E_G	Mean	0.13821	0.140406	0.237748	0.160352	0.117451	0.146158	0.110943	0.112485
		StdDev	0.005143	0.005242	0.004665	0.003842	0.003418	0.003258	0.004261	0.003668
B3	E_T	Mean	0.155954	0.155336	0.114547	0.181174	0.112531	0.140746	0.107024	0.108173
		StdDev	0.001654	0.001883	0.001191	0.001702	0.000872	0.001634	0.001584	0.00201
	E_G	Mean	0.13821	0.140406	0.237748	0.160352	0.117451	0.146158	0.110943	0.112485
		StdDev	0.005143	0.005242	0.004665	0.003842	0.003418	0.003258	0.004261	0.003668
B4	E_T	Mean	0.155954	0.155336	0.114547	0.181174	0.112531	0.140746	0.107024	0.108173
		StdDev	0.001654	0.001883	0.001191	0.001702	0.000872	0.001634	0.001584	0.00201
	E_G	Mean	0.13821	0.140406	0.237748	0.160352	0.117451	0.146158	0.110943	0.112485
		StdDev	0.005143	0.005242	0.004665	0.003842	0.003418	0.003258	0.004261	0.003668
C1	E_T	Mean	0.155954	0.155336	0.114547	0.181174	0.112531	0.140746	0.107024	0.108173
		StdDev	0.001654	0.001883	0.001191	0.001702	0.000872	0.001634	0.001584	0.00201
	E_G	Mean	0.13821	0.140406	0.237748	0.160352	0.117451	0.146158	0.110943	0.112485
		StdDev	0.005143	0.005242	0.004665	0.003842	0.003418	0.003258	0.004261	0.003668
C2	E_T	Mean	0.155954	0.155336	0.114547	0.181174	0.112531	0.140746	0.107024	0.108173
		StdDev	0.001654	0.001883	0.001191	0.001702	0.000872	0.001634	0.001584	0.00201
	E_G	Mean	0.13821	0.140406	0.237748	0.160352	0.117451	0.146158	0.110943	0.112485
		StdDev	0.005143	0.005242	0.004665	0.003842	0.003418	0.003258	0.004261	0.003668
C3	E_T	Mean	0.155954	0.155336	0.114547	0.181174	0.112531	0.140746	0.107024	0.108173
		StdDev	0.001654	0.001883	0.001191	0.001702	0.000872	0.001634	0.001584	0.00201
	E_G	Mean	0.13821	0.140406	0.237748	0.160352	0.117451	0.146158	0.110943	0.112485
		StdDev	0.005143	0.005242	0.004665	0.003842	0.003418	0.003258	0.004261	0.003668
C4	E_T	Mean	0.155954	0.155336	0.114547	0.181174	0.112531	0.140746	0.107024	0.108173
		StdDev	0.001654	0.001883	0.001191	0.001702	0.000872	0.001634	0.001584	0.00201
	E_G	Mean	0.13821	0.140406	0.237748	0.160352	0.117451	0.146158	0.110943	0.112485
		StdDev	0.005143	0.005242	0.004665	0.003842	0.003418	0.003258	0.004261	0.003668
D1	E_T	Mean	0.155954	0.155336	0.114547	0.181174	0.112531	0.140746	0.107024	0.108173
		StdDev	0.001654	0.001883	0.001191	0.001702	0.000872	0.001634	0.001584	0.00201
	E_G	Mean	0.13821	0.140406	0.237748	0.160352	0.117451	0.146158	0.110943	0.112485
		StdDev	0.005143	0.005242	0.004665	0.003842	0.003418	0.003258	0.004261	0.003668
D2	E_T	Mean	0.155954	0.155336	0.114547	0.181174	0.112531	0.140746	0.107024	0.108173
		StdDev	0.001654	0.001883	0.001191	0.001702	0.000872	0.001634	0.001584	0.00201
	E_G	Mean	0.13821	0.140406	0.237748	0.160352	0.117451	0.146158	0.110943	0.112485
		StdDev	0.005143	0.005242	0.004665	0.003842	0.003418	0.003258	0.004261	0.003668
D3	E_T	Mean	0.155954	0.155336	0.114547	0.181174	0.112531	0.140746	0.107024	0.108173
		StdDev	0.001654	0.001883	0.001191	0.001702	0.000872	0.001634	0.001584	0.00201
	E_G	Mean	0.13821	0.140406	0.237748	0.160352	0.117451	0.146158	0.110943	0.112485
		StdDev	0.005143	0.005242	0.004665	0.003842	0.003418	0.003258	0.004261	0.003668
D4	E_T	Mean	0.155954	0.155336	0.114547	0.181174	0.112531	0.140746	0.107024	0.108173
		StdDev	0.001654	0.001883	0.001191	0.001702	0.000872	0.001634	0.001584	0.00201
	E_G	Mean	0.13821	0.140406	0.237748	0.160352	0.117451	0.146158	0.110943	0.112485
		StdDev	0.005143	0.005242	0.004665	0.003842	0.003418	0.003258	0.004261	0.003668

Table 12 Dynamic sphere results for scenarios A1 to D4

Scenario			BP	SBP	RBP	SRBP	RPSO	RAPSO	CPSO	QPSO
A1	E_T	Mean	0.171742	0.171996	0.131876	0.193156	0.113942	0.130711	0.119896	0.120057
		StdDev	0.002489	0.001788	0.002916	0.001462	0.001718	0.000857	0.002418	0.00155
	E_G	Mean	0.137699	0.140721	0.199343	0.157206	0.115899	0.132334	0.121289	0.120964
		StdDev	0.006209	0.00394	0.005105	0.004174	0.003382	0.00401	0.00364	0.003453
A2	E_T	Mean	0.171742	0.171996	0.131876	0.193156	0.113942	0.130711	0.119896	0.120057
		StdDev	0.002489	0.001788	0.002916	0.001462	0.001718	0.000857	0.002418	0.00155
	E_G	Mean	0.137699	0.140721	0.199343	0.157206	0.115899	0.132334	0.121289	0.120964
		StdDev	0.006209	0.00394	0.005105	0.004174	0.003382	0.00401	0.00364	0.003453
A3	E_T	Mean	0.171742	0.171996	0.131876	0.193156	0.113942	0.130711	0.119896	0.120057
		StdDev	0.002489	0.001788	0.002916	0.001462	0.001718	0.000857	0.002418	0.00155
	E_G	Mean	0.137699	0.140721	0.199343	0.157206	0.115899	0.132334	0.121289	0.120964
		StdDev	0.006209	0.00394	0.005105	0.004174	0.003382	0.00401	0.00364	0.003453
A4	E_T	Mean	0.171742	0.171996	0.131876	0.193156	0.113942	0.130711	0.119896	0.120057
		StdDev	0.002489	0.001788	0.002916	0.001462	0.001718	0.000857	0.002418	0.00155
	E_G	Mean	0.137699	0.140721	0.199343	0.157206	0.115899	0.132334	0.121289	0.120964
		StdDev	0.006209	0.00394	0.005105	0.004174	0.003382	0.00401	0.00364	0.003453
B1	E_T	Mean	0.171742	0.171996	0.131876	0.193156	0.113942	0.130711	0.119896	0.120057
		StdDev	0.002489	0.001788	0.002916	0.001462	0.001718	0.000857	0.002418	0.00155
	E_G	Mean	0.137699	0.140721	0.199343	0.157206	0.115899	0.132334	0.121289	0.120964
		StdDev	0.006209	0.00394	0.005105	0.004174	0.003382	0.00401	0.00364	0.003453
B2	E_T	Mean	0.171742	0.171996	0.131876	0.193156	0.113942	0.130711	0.119896	0.120057
		StdDev	0.002489	0.001788	0.002916	0.001462	0.001718	0.000857	0.002418	0.00155
	E_G	Mean	0.137699	0.140721	0.199343	0.157206	0.115899	0.132334	0.121289	0.120964
		StdDev	0.006209	0.00394	0.005105	0.004174	0.003382	0.00401	0.00364	0.003453
B3	E_T	Mean	0.171742	0.171996	0.131876	0.193156	0.113942	0.130711	0.119896	0.120057
		StdDev	0.002489	0.001788	0.002916	0.001462	0.001718	0.000857	0.002418	0.00155
	E_G	Mean	0.137699	0.140721	0.199343	0.157206	0.115899	0.132334	0.121289	0.120964
		StdDev	0.006209	0.00394	0.005105	0.004174	0.003382	0.00401	0.00364	0.003453
B4	E_T	Mean	0.171742	0.171996	0.131876	0.193156	0.113942	0.130711	0.119896	0.120057
		StdDev	0.002489	0.001788	0.002916	0.001462	0.001718	0.000857	0.002418	0.00155
	E_G	Mean	0.137699	0.140721	0.199343	0.157206	0.115899	0.132334	0.121289	0.120964
		StdDev	0.006209	0.00394	0.005105	0.004174	0.003382	0.00401	0.00364	0.003453
C1	E_T	Mean	0.171742	0.171996	0.131876	0.193156	0.113942	0.130711	0.119896	0.120057
		StdDev	0.002489	0.001788	0.002916	0.001462	0.001718	0.000857	0.002418	0.00155
	E_G	Mean	0.137699	0.140721	0.199343	0.157206	0.115899	0.132334	0.121289	0.120964
		StdDev	0.006209	0.00394	0.005105	0.004174	0.003382	0.00401	0.00364	0.003453
C2	E_T	Mean	0.171742	0.171996	0.131876	0.193156	0.113942	0.130711	0.119896	0.120057
		StdDev	0.002489	0.001788	0.002916	0.001462	0.001718	0.000857	0.002418	0.00155
	E_G	Mean	0.137699	0.140721	0.199343	0.157206	0.115899	0.132334	0.121289	0.120964
		StdDev	0.006209	0.00394	0.005105	0.004174	0.003382	0.00401	0.00364	0.003453
C3	E_T	Mean	0.171742	0.171996	0.131876	0.193156	0.113942	0.130711	0.119896	0.120057
		StdDev	0.002489	0.001788	0.002916	0.001462	0.001718	0.000857	0.002418	0.00155
	E_G	Mean	0.137699	0.140721	0.199343	0.157206	0.115899	0.132334	0.121289	0.120964
		StdDev	0.006209	0.00394	0.005105	0.004174	0.003382	0.00401	0.00364	0.003453
C4	E_T	Mean	0.171742	0.171996	0.131876	0.193156	0.113942	0.130711	0.119896	0.120057
		StdDev	0.002489	0.001788	0.002916	0.001462	0.001718	0.000857	0.002418	0.00155
	E_G	Mean	0.137699	0.140721	0.199343	0.157206	0.115899	0.132334	0.121289	0.120964
		StdDev	0.006209	0.00394	0.005105	0.004174	0.003382	0.00401	0.00364	0.003453
D1	E_T	Mean	0.171742	0.171996	0.131876	0.193156	0.113942	0.130711	0.119896	0.120057
		StdDev	0.002489	0.001788	0.002916	0.001462	0.001718	0.000857	0.002418	0.00155
	E_G	Mean	0.137699	0.140721	0.199343	0.157206	0.115899	0.132334	0.121289	0.120964
		StdDev	0.006209	0.00394	0.005105	0.004174	0.003382	0.00401	0.00364	0.003453
D2	E_T	Mean	0.171742	0.171996	0.131876	0.193156	0.113942	0.130711	0.119896	0.120057
		StdDev	0.002489	0.001788	0.002916	0.001462	0.001718	0.000857	0.002418	0.00155
	E_G	Mean	0.137699	0.140721	0.199343	0.157206	0.115899	0.132334	0.121289	0.120964
		StdDev	0.006209	0.00394	0.005105	0.004174	0.003382	0.00401	0.00364	0.003453
D3	E_T	Mean	0.171742	0.171996	0.131876	0.193156	0.113942	0.130711	0.119896	0.120057
		StdDev	0.002489	0.001788	0.002916	0.001462	0.001718	0.000857	0.002418	0.00155
	E_G	Mean	0.137699	0.140721	0.199343	0.157206	0.115899	0.132334	0.121289	0.120964
		StdDev	0.006209	0.00394	0.005105	0.004174	0.003382	0.00401	0.00364	0.003453
D4	E_T	Mean	0.171742	0.171996	0.131876	0.193156	0.113942	0.130711	0.119896	0.120057
		StdDev	0.002489	0.001788	0.002916	0.001462	0.001718	0.000857	0.002418	0.00155
	E_G	Mean	0.137699	0.140721	0.199343	0.157206	0.115899	0.132334	0.121289	0.120964
		StdDev	0.006209	0.00394	0.005105	0.004174	0.003382	0.00401	0.00364	0.003453

Table 13 Sliding thresholds results for scenarios A1 to D4

Scenario			BP	SBP	RBP	SRBP	RPSO	RAPSO	CPSO	QPSO
A1	E_T	Mean	0.153493	0.134399	0.103516	0.137472	0.070907	0.09859	0.072955	0.071743
		StdDev	0.015032	0.009187	0.000502	0.001443	0.000673	0.00159	0.004674	0.003221
	E_G	Mean	0.136737	0.115268	0.147665	0.118282	0.071907	0.098238	0.073412	0.073101
		StdDev	0.014298	0.01055	0.003263	0.002846	0.002433	0.002966	0.005112	0.00487
A2	E_T	Mean	0.153493	0.134399	0.103516	0.137472	0.070907	0.09859	0.072955	0.071743
		StdDev	0.015032	0.009187	0.000502	0.001443	0.000673	0.00159	0.004674	0.003221
	E_G	Mean	0.136737	0.115268	0.147665	0.118282	0.071907	0.098238	0.073412	0.073101
		StdDev	0.014298	0.01055	0.003263	0.002846	0.002433	0.002966	0.005112	0.00487
A3	E_T	Mean	0.153493	0.134399	0.103516	0.137472	0.070907	0.09859	0.072955	0.071743
		StdDev	0.015032	0.009187	0.000502	0.001443	0.000673	0.00159	0.004674	0.003221
	E_G	Mean	0.136737	0.115268	0.147665	0.118282	0.071907	0.098238	0.073412	0.073101
		StdDev	0.014298	0.01055	0.003263	0.002846	0.002433	0.002966	0.005112	0.00487
A4	E_T	Mean	0.153493	0.134399	0.103516	0.137472	0.070907	0.09859	0.072955	0.071743
		StdDev	0.015032	0.009187	0.000502	0.001443	0.000673	0.00159	0.004674	0.003221
	E_G	Mean	0.136737	0.115268	0.147665	0.118282	0.071907	0.098238	0.073412	0.073101
		StdDev	0.014298	0.01055	0.003263	0.002846	0.002433	0.002966	0.005112	0.00487
B1	E_T	Mean	0.153493	0.134399	0.103516	0.137472	0.070907	0.09859	0.072955	0.071743
		StdDev	0.015032	0.009187	0.000502	0.001443	0.000673	0.00159	0.004674	0.003221
	E_G	Mean	0.136737	0.115268	0.147665	0.118282	0.071907	0.098238	0.073412	0.073101
		StdDev	0.014298	0.01055	0.003263	0.002846	0.002433	0.002966	0.005112	0.00487
B2	E_T	Mean	0.153493	0.134399	0.103516	0.137472	0.070907	0.09859	0.072955	0.071743
		StdDev	0.015032	0.009187	0.000502	0.001443	0.000673	0.00159	0.004674	0.003221
	E_G	Mean	0.136737	0.115268	0.147665	0.118282	0.071907	0.098238	0.073412	0.073101
		StdDev	0.014298	0.01055	0.003263	0.002846	0.002433	0.002966	0.005112	0.00487
B3	E_T	Mean	0.153493	0.134399	0.103516	0.137472	0.070907	0.09859	0.072955	0.071743
		StdDev	0.015032	0.009187	0.000502	0.001443	0.000673	0.00159	0.004674	0.003221
	E_G	Mean	0.136737	0.115268	0.147665	0.118282	0.071907	0.098238	0.073412	0.073101
		StdDev	0.014298	0.01055	0.003263	0.002846	0.002433	0.002966	0.005112	0.00487
B4	E_T	Mean	0.153493	0.134399	0.103516	0.137472	0.070907	0.09859	0.072955	0.071743
		StdDev	0.015032	0.009187	0.000502	0.001443	0.000673	0.00159	0.004674	0.003221
	E_G	Mean	0.136737	0.115268	0.147665	0.118282	0.071907	0.098238	0.073412	0.073101
		StdDev	0.014298	0.01055	0.003263	0.002846	0.002433	0.002966	0.005112	0.00487
C1	E_T	Mean	0.153493	0.134399	0.103516	0.137472	0.070907	0.09859	0.072955	0.071743
		StdDev	0.015032	0.009187	0.000502	0.001443	0.000673	0.00159	0.004674	0.003221
	E_G	Mean	0.136737	0.115268	0.147665	0.118282	0.071907	0.098238	0.073412	0.073101
		StdDev	0.014298	0.01055	0.003263	0.002846	0.002433	0.002966	0.005112	0.00487
C2	E_T	Mean	0.153493	0.134399	0.103516	0.137472	0.070907	0.09859	0.072955	0.071743
		StdDev	0.015032	0.009187	0.000502	0.001443	0.000673	0.00159	0.004674	0.003221
	E_G	Mean	0.136737	0.115268	0.147665	0.118282	0.071907	0.098238	0.073412	0.073101
		StdDev	0.014298	0.01055	0.003263	0.002846	0.002433	0.002966	0.005112	0.00487
C3	E_T	Mean	0.153493	0.134399	0.103516	0.137472	0.070907	0.09859	0.072955	0.071743
		StdDev	0.015032	0.009187	0.000502	0.001443	0.000673	0.00159	0.004674	0.003221
	E_G	Mean	0.136737	0.115268	0.147665	0.118282	0.071907	0.098238	0.073412	0.073101
		StdDev	0.014298	0.01055	0.003263	0.002846	0.002433	0.002966	0.005112	0.00487
C4	E_T	Mean	0.153493	0.134399	0.103516	0.137472	0.070907	0.09859	0.072955	0.071743
		StdDev	0.015032	0.009187	0.000502	0.001443	0.000673	0.00159	0.004674	0.003221
	E_G	Mean	0.136737	0.115268	0.147665	0.118282	0.071907	0.098238	0.073412	0.073101
		StdDev	0.014298	0.01055	0.003263	0.002846	0.002433	0.002966	0.005112	0.00487
D1	E_T	Mean	0.153493	0.134399	0.103516	0.137472	0.070907	0.09859	0.072955	0.071743
		StdDev	0.015032	0.009187	0.000502	0.001443	0.000673	0.00159	0.004674	0.003221
	E_G	Mean	0.136737	0.115268	0.147665	0.118282	0.071907	0.098238	0.073412	0.073101
		StdDev	0.014298	0.01055	0.003263	0.002846	0.002433	0.002966	0.005112	0.00487
D2	E_T	Mean	0.153493	0.134399	0.103516	0.137472	0.070907	0.09859	0.072955	0.071743
		StdDev	0.015032	0.009187	0.000502	0.001443	0.000673	0.00159	0.004674	0.003221
	E_G	Mean	0.136737	0.115268	0.147665	0.118282	0.071907	0.098238	0.073412	0.073101
		StdDev	0.014298	0.01055	0.003263	0.002846	0.002433	0.002966	0.005112	0.00487
D3	E_T	Mean	0.153493	0.134399	0.103516	0.137472	0.070907	0.09859	0.072955	0.071743
		StdDev	0.015032	0.009187	0.000502	0.001443	0.000673	0.00159	0.004674	0.003221
	E_G	Mean	0.136737	0.115268	0.147665	0.118282	0.071907	0.098238	0.073412	0.073101
		StdDev	0.014298	0.01055	0.003263	0.002846	0.002433	0.002966	0.005112	0.00487
D4	E_T	Mean	0.153493	0.134399	0.103516	0.137472	0.070907	0.09859	0.072955	0.071743
		StdDev	0.015032	0.009187	0.000502	0.001443	0.000673	0.00159	0.004674	0.003221
	E_G	Mean	0.136737	0.115268	0.147665	0.118282	0.071907	0.098238	0.073412	0.073101
		StdDev	0.014298	0.01055	0.003263	0.002846	0.002433	0.002966	0.005112	0.00487

Table 14 Electricity pricing results for scenarios A1 to D4

Scenario			BP	SBP	RBP	SRBP	RPSO	RAPSO	CPSO	QPSO
A1	E_T	Mean	0.107272	0.109893	0.130724	0.136868	0.1123	0.130703	0.103669	0.101964
		StdDev	0.001239	0.001609	0.000274	0.001065	0.000892	0.000786	0.005592	0.00475
	E_G	Mean	0.107447	0.110417	0.148172	0.134327	0.112489	0.13076	0.103975	0.102248
		StdDev	0.001796	0.002576	0.001327	0.00193	0.002648	0.001805	0.005727	0.005023
A2	E_T	Mean	0.107272	0.109893	0.130724	0.136868	0.1123	0.130703	0.103669	0.101964
		StdDev	0.001239	0.001609	0.000274	0.001065	0.000892	0.000786	0.005592	0.00475
	E_G	Mean	0.107447	0.110417	0.148172	0.134327	0.112489	0.13076	0.103975	0.102248
		StdDev	0.001796	0.002576	0.001327	0.00193	0.002648	0.001805	0.005727	0.005023
A3	E_T	Mean	0.107272	0.109893	0.130724	0.136868	0.1123	0.130703	0.103669	0.101964
		StdDev	0.001239	0.001609	0.000274	0.001065	0.000892	0.000786	0.005592	0.00475
	E_G	Mean	0.107447	0.110417	0.148172	0.134327	0.112489	0.13076	0.103975	0.102248
		StdDev	0.001796	0.002576	0.001327	0.00193	0.002648	0.001805	0.005727	0.005023
A4	E_T	Mean	0.107272	0.109893	0.130724	0.136868	0.1123	0.130703	0.103669	0.101964
		StdDev	0.001239	0.001609	0.000274	0.001065	0.000892	0.000786	0.005592	0.00475
	E_G	Mean	0.107447	0.110417	0.148172	0.134327	0.112489	0.13076	0.103975	0.102248
		StdDev	0.001796	0.002576	0.001327	0.00193	0.002648	0.001805	0.005727	0.005023
B1	E_T	Mean	0.107272	0.109893	0.130724	0.136868	0.1123	0.130703	0.103669	0.101964
		StdDev	0.001239	0.001609	0.000274	0.001065	0.000892	0.000786	0.005592	0.00475
	E_G	Mean	0.107447	0.110417	0.148172	0.134327	0.112489	0.13076	0.103975	0.102248
		StdDev	0.001796	0.002576	0.001327	0.00193	0.002648	0.001805	0.005727	0.005023
B2	E_T	Mean	0.107272	0.109893	0.130724	0.136868	0.1123	0.130703	0.103669	0.101964
		StdDev	0.001239	0.001609	0.000274	0.001065	0.000892	0.000786	0.005592	0.00475
	E_G	Mean	0.107447	0.110417	0.148172	0.134327	0.112489	0.13076	0.103975	0.102248
		StdDev	0.001796	0.002576	0.001327	0.00193	0.002648	0.001805	0.005727	0.005023
B3	E_T	Mean	0.107272	0.109893	0.130724	0.136868	0.1123	0.130703	0.103669	0.101964
		StdDev	0.001239	0.001609	0.000274	0.001065	0.000892	0.000786	0.005592	0.00475
	E_G	Mean	0.107447	0.110417	0.148172	0.134327	0.112489	0.13076	0.103975	0.102248
		StdDev	0.001796	0.002576	0.001327	0.00193	0.002648	0.001805	0.005727	0.005023
B4	E_T	Mean	0.107272	0.109893	0.130724	0.136868	0.1123	0.130703	0.103669	0.101964
		StdDev	0.001239	0.001609	0.000274	0.001065	0.000892	0.000786	0.005592	0.00475
	E_G	Mean	0.107447	0.110417	0.148172	0.134327	0.112489	0.13076	0.103975	0.102248
		StdDev	0.001796	0.002576	0.001327	0.00193	0.002648	0.001805	0.005727	0.005023
C1	E_T	Mean	0.107272	0.109893	0.130724	0.136868	0.1123	0.130703	0.103669	0.101964
		StdDev	0.001239	0.001609	0.000274	0.001065	0.000892	0.000786	0.005592	0.00475
	E_G	Mean	0.107447	0.110417	0.148172	0.134327	0.112489	0.13076	0.103975	0.102248
		StdDev	0.001796	0.002576	0.001327	0.00193	0.002648	0.001805	0.005727	0.005023
C2	E_T	Mean	0.107272	0.109893	0.130724	0.136868	0.1123	0.130703	0.103669	0.101964
		StdDev	0.001239	0.001609	0.000274	0.001065	0.000892	0.000786	0.005592	0.00475
	E_G	Mean	0.107447	0.110417	0.148172	0.134327	0.112489	0.13076	0.103975	0.102248
		StdDev	0.001796	0.002576	0.001327	0.00193	0.002648	0.001805	0.005727	0.005023
C3	E_T	Mean	0.107272	0.109893	0.130724	0.136868	0.1123	0.130703	0.103669	0.101964
		StdDev	0.001239	0.001609	0.000274	0.001065	0.000892	0.000786	0.005592	0.00475
	E_G	Mean	0.107447	0.110417	0.148172	0.134327	0.112489	0.13076	0.103975	0.102248
		StdDev	0.001796	0.002576	0.001327	0.00193	0.002648	0.001805	0.005727	0.005023
C4	E_T	Mean	0.107272	0.109893	0.130724	0.136868	0.1123	0.130703	0.103669	0.101964
		StdDev	0.001239	0.001609	0.000274	0.001065	0.000892	0.000786	0.005592	0.00475
	E_G	Mean	0.107447	0.110417	0.148172	0.134327	0.112489	0.13076	0.103975	0.102248
		StdDev	0.001796	0.002576	0.001327	0.00193	0.002648	0.001805	0.005727	0.005023
D1	E_T	Mean	0.107272	0.109893	0.130724	0.136868	0.1123	0.130703	0.103669	0.101964
		StdDev	0.001239	0.001609	0.000274	0.001065	0.000892	0.000786	0.005592	0.00475
	E_G	Mean	0.107447	0.110417	0.148172	0.134327	0.112489	0.13076	0.103975	0.102248
		StdDev	0.001796	0.002576	0.001327	0.00193	0.002648	0.001805	0.005727	0.005023
D2	E_T	Mean	0.107272	0.109893	0.130724	0.136868	0.1123	0.130703	0.103669	0.101964
		StdDev	0.001239	0.001609	0.000274	0.001065	0.000892	0.000786	0.005592	0.00475
	E_G	Mean	0.107447	0.110417	0.148172	0.134327	0.112489	0.13076	0.103975	0.102248
		StdDev	0.001796	0.002576	0.001327	0.00193	0.002648	0.001805	0.005727	0.005023
D3	E_T	Mean	0.107272	0.109893	0.130724	0.136868	0.1123	0.130703	0.103669	0.101964
		StdDev	0.001239	0.001609	0.000274	0.001065	0.000892	0.000786	0.005592	0.00475
	E_G	Mean	0.107447	0.110417	0.148172	0.134327	0.112489	0.13076	0.103975	0.102248
		StdDev	0.001796	0.002576	0.001327	0.00193	0.002648	0.001805	0.005727	0.005023
D4	E_T	Mean	0.107272	0.109893	0.130724	0.136868	0.1123	0.130703	0.103669	0.101964
		StdDev	0.001239	0.001609	0.000274	0.001065	0.000892	0.000786	0.005592	0.00475
	E_G	Mean	0.107447	0.110417	0.148172	0.134327	0.112489	0.13076	0.103975	0.102248
		StdDev	0.001796	0.002576	0.001327	0.00193	0.002648	0.001805	0.005727	0.005023

References

Bishop, C. M. (1995). *Neural networks for pattern recognition*. Oxford: Oxford University Press.

Blackwell, T. M. (2005). Particle swarms and population diversity. *Soft Computing—A Fusion of Foundations, Methodologies and Applications*, *9*(11), 793–802.

Blackwell, T. M., & Bentley, P. J. (2002). Dynamic search with charged swarms. In *Proceedings of the genetic and evolutionary computation conference* (pp. 19–26). New York: Morgan Kaufmann.

Blackwell, T. M., & Branke, J. (2006). Multiswarms, exclusion, and anti-convergence in dynamic environments. *IEEE Transactions on Evolutionary Computation*, *10*(4), 459–472.

Blackwell, W. J., & Chen, F. W. (2009). *Neural networks in atmospheric remote sensing*. London: Artech House.

Carlisle, A., & Dozier, G. (2000). Adapting particle swarm optimization to dynamic environments. In *Proceedings of the international conference on artificial intelligence* (Vol. 1, pp. 429–434). Las Vegas: CSREA Press.

Carlisle, A., & Dozier, G. (2002). Tracking changing extrema with adaptive particle swarm optimizer. In *Proceedings of the 5th biannual world automation congress* (Vol. 13, pp. 265–270). Piscataway: IEEE.

Chu, F., Wang, Y., & Zaniolo, C. (2004). An adaptive learning approach for noisy data streams. In *Proceedings of the IEEE international conference on data mining* (pp. 351–354). Piscataway: IEEE.

Clerc, M., & Kennedy, J. (2002). The particle swarm—explosion, stability and convergence in a multidimensional complex space. *IEEE Transactions on Evolutionary Computation*, *6*(1), 58–73.

Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine Learning*, *20*(3), 273–297.

Cournane, A., & Hunt, R. (2004). An analysis of the tools used for the generation and prevention of spam. *Computers & Security*, *23*(2), 154–166.

Delany, S. J., Cunningham, P., Tsymbal, A., & Coyle, L. (2005). A case-based technique for tracking concept drift in spam filtering. *Knowledge-Based Systems*, *18*(4–5), 187–195.

Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, *7*, 1–30.

Dreyfus, G. (2005). *Neural networks: methodology and applications*. Berlin: Springer.

Duhain, J. G. O. L. (2011). *Particle swarm optimisation in dynamically changing environments—an empirical study*. Master's thesis, University of Pretoria, Pretoria, South Africa.

Eberhart, R. C., & Shi, Y. (2000). Comparing inertia weights and constriction factors in particle swarm optimization. In *Proceedings of the IEEE congress on evolutionary computation* (Vol. 1, pp. 84–88). Piscataway: IEEE.

Eberhart, R. C., & Shi, Y. (2001). Tracking and optimizing dynamic systems with particle swarms. In *Proceedings of the IEEE congress on evolutionary computation* (Vol. 1, pp. 94–100). Piscataway: IEEE.

Engelbrecht, A. P. (2005). *Fundamentals of computational swarm intelligence*. New York: Wiley.

Engelbrecht, A. P., & Ismail, A. (1999). Training product unit neural networks. *Stability and Control: Theory and Applications*, *2*(1–2), 59–74.

Fdez-Riverola, F., Iglesias, E. L., Daz, F., Mndez, J. R., & Corchado, J. M. (2007). Applying lazy learning algorithms to tackle concept drift in spam filtering. *Expert Systems with Applications*, *33*(1), 36–48.

Fogel, D. B., Fogel, L. J., & Porto, V. W. (1990). Evolving neural networks. *Biological Cybernetics*, *63*(6), 487–493.

Gallagher, M. R. (2000). *Multi-layer perceptron error surfaces: visualization, structure and modelling*. PhD thesis, University of Queensland, St Lucia 4072, Australia.

Gudise, V. G., & Venayagamoorthy, G. K. (2003). Comparison of particle swarm optimization and backpropagation as training algorithms for neural networks. In *Proceedings of the IEEE swarm intelligence symposium* (pp. 110–117). Piscataway: IEEE.

Harries, M. (1999). *Splice-2 comparative evaluation: electricity pricing* (Technical Report UNSW-CSE-TR-9905). Artificial Intelligence Group, School of Computer Science and Engineering, The University of New South Wales, Sydney 2052, Australia.

Hassoun, M. H. (1995). *Fundamentals of artificial neural networks*. *Bradford books*. Cambridge: MIT Press.

Hu, X., & Eberhart, R. C. (2002). Adaptive particle swarm optimization: detection and response to dynamic systems. In *Proceedings of the IEEE congress on evolutionary computation* (Vol. 2, pp. 1666–1670). Piscataway: IEEE.

Hush, D. R., Horne, B., & Salas, J. M. (1992). Error surfaces for multilayer perceptrons. *IEEE Transactions on Systems, Man and Cybernetics*, *22*(5), 1152–1161.

Ismail, A. (2001). *Training and optimization of product unit neural networks*. Master's thesis, University of Pretoria, Pretoria, South Africa.

Jinli, M., & Zhiyi, S. (2000). Application of combined neural networks in nonlinear function approximation. In *Proceedings of the 3rd world congress on intelligent control and automation* (Vol. 2, pp. 839–841). Piscataway: IEEE.

Kennedy, J. (1999). Small worlds and mega-minds: Effects of neighborhood topology on particle swarm performance. In *Proceedings of the IEEE congress on evolutionary computation* (Vol. 3, pp. 1931–1938). Piscataway: IEEE.

Kennedy, J., & Eberhart, R. C. (1995). Particle swarm optimization. In *Proceedings of the IEEE international conference on neural networks* (Vol. IV, pp. 1942–1948). Piscataway: IEEE.

Kennedy, J., & Mendes, R. (2002). Population structure and particle swarm performance. In *Proceedings of the congress on evolutionary computation* (pp. 407–412). Piscataway: IEEE.

Kennedy, J., Eberhart, R. C., & Shi, Y. (2001). *Swarm intelligence*. San Francisco: Morgan Kaufmann.

Klinkenberg, R. (2004). Learning drifting concepts: example selection vs. example weighting. *Intelligent Data Analysis*, *8*(3), 281–300.

Kuncheva, L. (2004). Classifier ensembles for changing environments. In F. Roli, J. Kittler, & T. Windeatt (Eds.), *Lecture notes in computer science*: *Vol. 3077*. *Multiple classifier systems* (pp. 1–15). Berlin: Springer.

Last, M. (2002). Online classification of nonstationary data streams. *Intelligent Data Analysis*, *6*(2), 129–147.

Lau, R. (1994). *Adaptive statistical language modeling*. Master's thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Massachusetts, USA.

Lawrence, S., Tsoi, A. C., & Back, A. D. (1996). Function approximation with neural networks and local methods: bias, variance and smoothness. In *Proceedings of the Australian conference on neural networks* (pp. 16–21). Canberra: Australian National University.

Leonard, B., & Engelbrecht, A. P. (2012). *Control parameter optimisation for dynamic optimisation problems* (Technical report). Department of Computer Science, University of Pretoria, South Africa.

Li, F., Meng, Q. H., Bai, S., Li, J. G., & Popescu, D. (2008). Probability-PSO algorithm for multi-robot based odor source localization in ventilated indoor environments. In C. Xiong, H. Liu, Y. Huang, & Y. Xiong (Eds.), *Lecture notes in computer science* (Vol. 5314, pp. 1206–1215). Berlin: Springer.

Li, X., & Khanh, H. D. (2003). Comparing particle swarms for tracking extrema in dynamic environments. In *Proceedings of the congress on evolutionary computation* (Vol. 3, pp. 1772–1779). Piscataway: IEEE.

Liao, Y., Vemuri, V. R., & Pasos, A. (2007). Adaptive anomaly detection with evolving connectionist systems. *Journal of Network and Computer Applications*, *30*(1), 60–80.

Mann, H. B., & Whitney, D. R. (1947). On a test of whether one of two random variables is stochastically larger than the other. *Annals of Mathematical Statistics*, *18*(1), 50–60.

Mendes, R., Cortez, P., Rocha, M., & Neves, J. (2002). Particle swarms for feedforward neural network training. In *Proceedings of the international joint conference on neural networks* (Vol. 2, pp. 1895–1899). Piscataway: IEEE.

Morrison, R. W. (2003). Performance measurement in dynamic environments. In J. Branke (Ed.), *Proceedings of the genetic and evolutionary computation conference workshop on evolutionary algorithms for dynamic optimization problems* (pp. 5–8). New York: ACM Press.

Nickabadi, A., Ebadzadeh, M., & Safabakhsh, R. (2012). A competitive clustering particle swarm optimizer for dynamic optimization problems. *Swarm Intelligence*, *6*(3), 1–30.

Patterson, D. W. (1998). *Artificial neural networks: theory and applications* (1st ed.). Upper Saddle River: Prentice Hall.

Peer, E. S., van den Bergh, F., & Engelbrecht, A. P. (2003). Using neighborhoods with guaranteed convergence PSO. In *Proceedings of the IEEE swarm intelligence symposium* (pp. 235–242). Piscataway: IEEE.

Picard, R. R., & Cook, R. D. (1984). Cross-validation of regression models. *Journal of the American Statistical Association*, *79*(387), 575–583.

Poli, R., Kennedy, J., & Blackwell, T. (2007). Particle swarm optimization. *Swarm Intelligence*, *1*(1), 33–57.

Porto, V. W., & Fogel, D. B. (2002). Alternative neural network training methods (active sonar processing). *IEEE Expert*, *10*(3), 16–22.

Rakitianskaia, A., & Engelbrecht, A. P. (2008). Cooperative charged particle swarm optimiser. In *Proceedings of the IEEE congress on evolutionary computation* (pp. 933–939). Piscataway: IEEE.

Rakitianskaia, A., & Engelbrecht, A. P. (2009). Training neural networks with PSO in dynamic environments. In *Proceedings of the IEEE congress on evolutionary computation* (pp. 667–673). Piscataway: IEEE.

Rojas, R. (1996). *Neural networks: a systematic introduction*. Berlin: Springer.

Rokach, L. (2010). *Pattern classification using ensemble methods*. *Series in machine perception and artificial intelligence*. Singapore: World Scientific, Toh Tuck.

Schlimmer, J. C., & Granger, R. H. (1986). Incremental learning from noisy data. *Machine Learning*, *1*(3), 317–354.

Shao, J. (1993). Linear model selection by cross-validation. *Journal of the American Statistical Association*, *88*(422), 486–494.

Shi, Y., & Eberhart, R. C. (1998). Parameter selection in particle swarm optimization. In V. Porto, N. Saravanan, D. Waagen, & A. Eiben (Eds.), *Lecture notes in computer science*: *Vol. 1447*. *Evolutionary programming VII* (pp. 591–600). Berlin: Springer.

Shi, Y., & Eberhart, R. C. (1999). Empirical study of particle swarm optimisation. In *Proceedings of the IEEE congress on evolutionary computation* (Vol. 3, pp. 1945–1950). Piscataway: IEEE.

Street, W. N., & Kim, Y. S. (2001). A streaming ensemble algorithm (SEA) for large-scale classification. In *Proceedings of the international conference on knowledge discovery and data mining* (pp. 377–382). New York: ACM Press.

Thuraisingham, B. (2008). Data mining for security applications: mining concept-drifting data streams to detect peer to peer botnet traffic. In *IEEE international conference on intelligence and security informatics* (pp. xxix–xxx). Piscataway: IEEE.

Tsymbal, A. (2004). *The problem of concept drift: definitions and related work* (Technical Report TCD-CS-2004-15). Trinity College, Dublin, Ireland.

Tsymbal, A., Pechenizkiy, M., Cunningham, P., & Puuronen, S. (2008). Dynamic integration of classifiers for handling concept drift. *Information Fusion*, *9*(1), 56–68.

van den Bergh, F., & Engelbrecht, A. P. (2000). Cooperative learning in neural networks using particle swarm optimizers. *South African Computer Journal*, *26*, 84–90.

van den Bergh, F. (2002). *An analysis of particle swarm optimizers*. PhD thesis, Department of Computer Science, University of Pretoria, Pretoria, South Africa.

Wan, E. A. (1990). Neural network classification: a Bayesian interpretation. *IEEE Transactions on Neural Networks*, *1*(4), 303–305.

Watanasusin, N., & Sanguansintukul, S. (2011). Classifying chief complaint in ear diseases using data mining techniques. In *International conference on digital content, multimedia technology and its applications* (pp. 149–153). Piscataway: IEEE.

Werbos, P. J. (1974). *Beyond regression: new tools for prediction and analysis in the behavioural sciences*. PhD thesis, Harvard University, Boston, USA.

Widmer, G., & Kubat, M. (1996). Learning in the presence of concept drift and hidden contexts. *Machine Learning*, *23*(1), 69–101.

Williams, P. M. (1995). Bayesian regularization and pruning using a Laplace prior. *Neural Computation*, *7*(1), 117–143.

Xu, Q. S., & Liang, Y. Z. (2001). Monte Carlo cross validation. *Chemometrics and Intelligent Laboratory Systems*, *56*(1), 1–11.

Zhang, T. (2005). Class-size independent generalization analysis of some discriminative multi-category classification. *Advances in Neural Information Processing Systems*, *17*, 1625–1632.

Zhao, J., Chen, M., & Luo, Q. (2011). Research of intrusion detection systems based on neural networks. In *International conference on communication software and networks* (pp. 174–178). Piscataway: IEEE.

Zurada, J. M. (1992). *Introduction to artificial neural systems*. St. Paul: West Publishing Company.